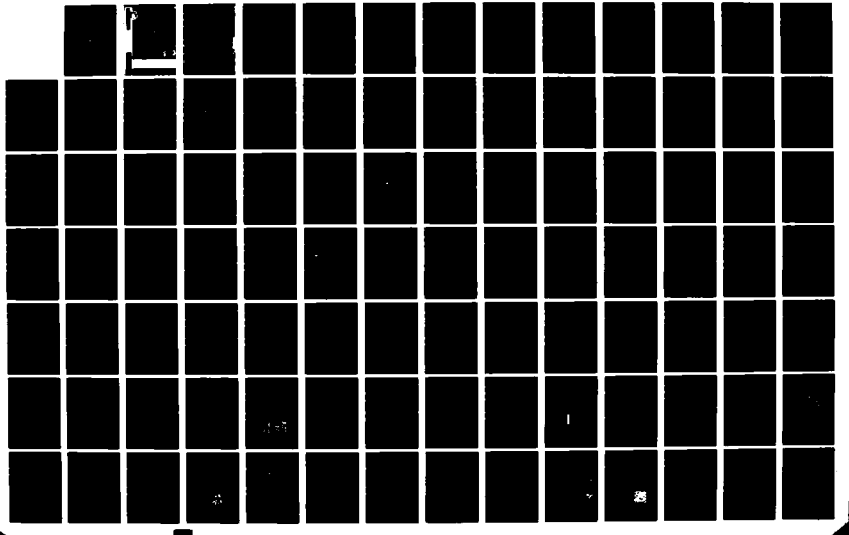
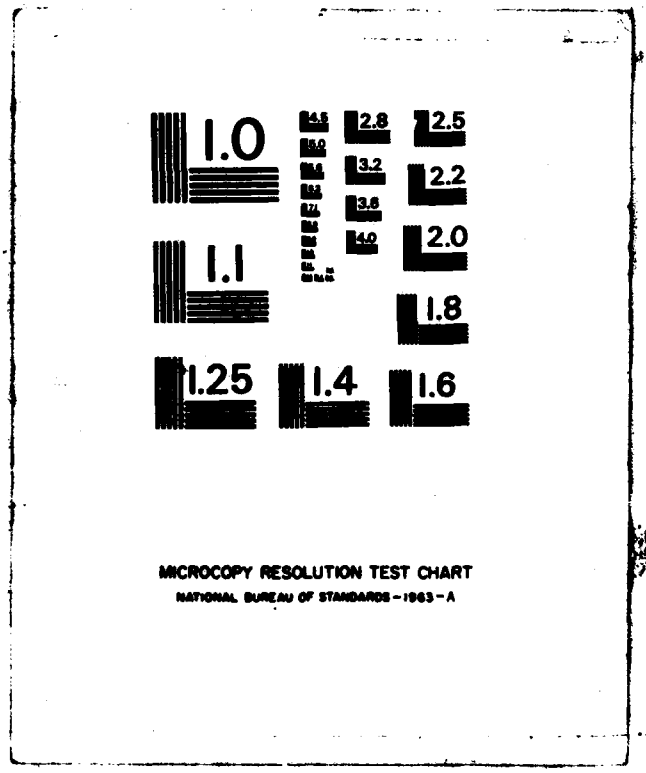
AD-A122 000 REAL SCAN EVOLUTION(U) UNIVERSITY OF CENTRAL FLORIDA 1/6
ORLANDO DEPT OF ELECTRICAL ENGINEERING B W PATZ ET AL.
FEB 82 NAVTRAEQUIPC-80-D-D014-2 N61339-80-D-0014

UNCLASSIFIED F/G 9/2 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A122 000

Technical Report: NAVTRAEQUIPCEN 80-D-0014-2

REAL SCAN EVOLUTION

By

Benjamin W. Patz
Phil Gatt
Gerald L. Becker
Sam M. Richie
Richard LeBlanc
Linda Coulter
Department of Electrical Engineering
University of Central Florida
P.O. Box 25000
Orlando, Florida 32816

15 February 1982

For period 1 January 1981 through 31 December 1981

DoD DISTRIBUTION STATEMENT
Approved for public release;
distribution unlimited.

DTIC
ELECTE
NOV 30 1982
S D
E

Prepared for

NAVAL TRAINING EQUIPMENT CENTER
Orlando, Florida 32813

COPY

GOVERNMENT RIGHTS IN DATA STATEMENT

Reproduction of this publication in whole or in part is permitted for any purpose of the United States Government

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NAVTRAEQUIPCEN 80-D-0014-2	2. GOVT ACCESSION NO. AD-A122005	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Real Scan Evolution		5. TYPE OF REPORT & PERIOD COVERED Final Report for January 81 - December 81 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Benjamin W. Patz, Sam M. Richie Phil Gatt, Richard LeBlanc Gerald L. Becker, Linda Coulter		8. CONTRACT OR GRANT NUMBER(s) N61339-80-D-0014
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Electrical Engineering University of Central Florida POB 25000, Orlando, FL 32816		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS PE62757N WF57526490 Task 8734
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Simulation Concepts Lab, Code N-731 Naval Training Equipment Center Orlando, FL 32813		12. REPORT DATE Feb 1982 13. NUMBER OF PAGES 630
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Image Generation
Visual Simulation
Computer Graphics Algorithm
Geometric Modeling

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the development and evaluation of a computer graphics algorithm capable of rendering high detail imagery of real world visual environments modeled as a single valued elevation function of horizontal location. The objective of the development was to analyze the feasibility of a real time implementation. The results indicate that, although technically feasible, the real time implementation is too computationally expensive to consider. Recommendations for future work to reduce expenses are made.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

PREFACE

The Advanced Simulation Concepts Laboratory at the Naval Training Equipment Center has recognized the need for high detail density visual simulation in training systems. In pursuing a solution to the complex visual environment problem, at least two factors have significant importance: Does the solution provide an image generator capable of creating complex imagery in real time? Is the solution amenable to efficient off-line modeling of complex environments? The Real Scan concept of utilizing a radially scanned hierarchy of fixed grid environment models was originated by Dr. A. M. Spooner of this laboratory and the process of providing answers to the two key questions was developed by the undersigned. The University of Central Florida, as personified by the authors of this report, assisted in this task by utilizing the Real Scan concept to develop algorithms, code them in software, and evaluate imagery in non-real time.

DENIS R. BREGLIA
Project Engineer

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	

NAVSEA COPY FURNISHED 2

TABLE OF CONTENTS

Section		Page
I.	INTRODUCTION	17
	General	17
	The Problem	18
	Automatic Data Base Generation Problems	19
	Development of New System Concept	19
	General REAL SCAN System Description	20
	The REAL SCAN Data Base	25
	Example of a Data Base Construction	25
II.	ALGORITHMS AND SOFTWARE	29
	Coordinate Systems	32
	World	32
	Eye	32
	Screen	33
	Nadir Centric World	34
	Eye Centric World	34
	Fortran Routine Documentation	35
	PGCOMDAT.FOR	35
	PGBUF.FOR	35
	PGMAIN.FOR	35
	PGSCENE.FOR	40
	PGSL.FOR	44
	PGVIS.FOR	46
	PGSCAN.FOR	47
	PGPROJ.FOR	51
	PGLOGLOAD.FOR	57
	PGFILTER.FOR	57
	PGDATA.FOR	58
	Second Generation Software	58
	Color Capability	59
	Program Organization	60
	Scaled Data Points	60
	Eye Centric World Coordinates	61
	"No Overflow" Multiply	61
	Scan Line Start	62
III.	THE DATA BASE PROBLEM	65
	REAL SCAN Data Base Models	68
	Data Base Modeling Estimates	69
	Terrain Modeling with Random Numbers (Noise)	72
	Uniform Distribution	77
	Weighting Arrays	79

Section		Page
	Reflection From Diffuse Surfaces	86
	Interpolation Formula	96
	The Incremental Distances	96
IV.	ANTI-ALIASING FILTER METHODS	101
	Final Filter	101
	Display Defects	102
	Final Filter Tests	103
	Filtering	106
V.	REAL SCAN ARCHITECTURE	113
	System Parameters	115
	Frame Buffer and Bus Control	117
	Architecture	119
	Alternate Concentrator Architecture	125
	Pong Frame Buffer Architecture	129
VI.	PICTURE GENERATION SOFTWARE	133
	Stand Alone Routines	134
	DICBWSA.FOR	134
	DICCOLSA.FOR	136
	Subroutines	138
	SDICBW.FOR	138
	SDICCOL.FOR	139
	Image Parameter Text Generation Routines	140
	Operational Overview	140
	DATACHAR.FOR	142
	DICCHAR.FOR	145
	CHCONV.FOR	151
	Operation Example	154
	Recommendations	158
VII.	CONCLUSIONS AND RECOMMENDATIONS	159
	Accomplishments	159
	Advantages/Disadvantages of REAL SCAN	161
	Recommendations	163
	System Hardware Recommendations	172
	Recommendation Summary	174
	REFERENCES	177

Section	Page
BIBLIOGRAPHY	179
Algorithms	179
Architecture	181
Data Base Methods	181
General	184
Vision or Perception	186
APPENDICES	
Appendix A - Analysis of the Memory Required for the Cell Data Base	189
The Field of View	189
Altitude-Velocity Relation	189
Calculation of the Working Cell Data Base Size	191
Appendix B - Memory Required for a Large Gaming Area Having Display Limited Detail	197
Appendix C - Intermediate Projection Planes	201
Posing the Locus Problem	201
Display Limited Parallax Motion	205
Projection Plane Parallax	205
Parallax Onset Time as a Function of Eye Motion Direction	209
Appendix D - Polynomial Data Bases	217
Polynomial Family Development	218
Evaluation of the Polynomial Coefficients	220
Appendix E - Arbitrary Eye Orientation	223
NPIX, IAXIS Sequence	230
Appendix F - Improved Scan Line Initialization Procedure	233
Appendix G - Display Projection to Fractional Pixel Accuracy	241
Appendix H - Improved World to Display Projection	245
Appendix I - Terrain Modeling Distributions	251
Triangular Distribution	251
Parabolic Distribution	253
Cusp Distribution	255

Section | Page

Appendix J - Lower Bound 257

Appendix K - Projection Processor 263

Appendix L - CPU Time Comparison 271

Appendix M - Effect of ANGFACT on CPU Time and Aliasing . . 273

Appendix N - Overhauser-Coons Interpolation 277

Introduction 277
The Overhauser-Coons Function 277
The Overhauser-Coons Bicubic Patch Routine 280
The Test of the Overhauser-Coons Bicubic Patch Routine . . 293

Appendix O - Development of Variable Increments Along Scan Lines 307

Appendix AA - DICOMED Start-Up Procedure 313

Appendix AB - DICOMED Driver Routines 315

Appendix AC - Subroutine AST = AST.FOR 321

Appendix AD - Subroutine BIT10 = BIT10.FOR 323

Appendix AE - Common Block = COMMON.FOR 325

Appendix AF - Subroutine D47CMD = D47CMD.FOR 327

Appendix AG - Subroutine D47OUT = D47OUT.FOR 329

Appendix AH - Subroutine D47POS = D47POS.FOR 331

Appendix AI - Subroutine D47STAT = D47STAT.FOR 333

Appendix AJ - Data Block = DEF.FOR 335

Appendix AK - Subroutine DELAY = DELAY.FOR 337

Appendix AL - Subroutine ERR = ERR.FOR 339

Appendix AM - Subroutine FILADV = FILADV.FOR 341

Appendix AN - Subroutine FILSEL = FILSEL.FOR 343

Appendix AO - Subroutine FILTER = FILTER.FOR 345

Section Page

Appendix AP - Common Block 2 = IO.FOR 347

Appendix AQ - Subroutine OPCHK = OPCHK.FOR 349

Appendix AR - Subroutine READY = READY.FOR 351

Appendix AS - Subroutine VDINIT = VDINIT.FOR 353

Appendix AT - Subroutine VIDCLS = VIDCLS.FOR 355

Appendix AU - Subroutine VIDOUT = VIDOUT.FOR 357

Appendix AV - Subroutie VIDSET = VIDSET.FOR 359

Appendix BA - Appendix B* Description 361

Appendix BB - Random Positioning and Arbitrary Positioning Tests Using Ranim.For 363

Appendix BC - Color Tests Using SRCAMSA.FOR 369

Appendix CA - Appendix C* Description 381

Appendix CB - DICBWSA.FOR 385

Appendix CC - DICCOLSA.FOR 389

Appendix CD - SDICBW.FOR 395

Appendix CE - SDICCOL.FOR 399

Appendix DA - Routines for Text Generation 403

Appendix DB - DATACHAR.FOR 405

Appendix DC - DICCHAR.FOR 411

Appendix DD - CHCONV.FOR 419

Appendix DE - DICCAMMAN.FOR 421

Appendix DF - EGLPICMAN.FOR 425

Appendix DG - EGLDATA.FOR 431

Appendix DH - Text Character Set 445

Appendix DI - Alternate Character Constructions 451

Section	Page
Appendix EA - First Generation REAL SCAN Software	453
Appendix EB - PGBUF.FOR	455
Appendix EC - PGCOMDAT.FOR	457
Appendix ED - PGDATA.FOR	459
Appendix EE - PGFILTER.FOR	475
Appendix EF - PGLOGLOAD.FOR	477
Appendix EG - PGMAIN.FOR	479
Appendix EH - PGPROJ.FOR	487
Appendix EI - SCAN.FOR	493
Appendix EJ - PGSCENE.FOR	497
Appendix EK - PGSL.FOR	501
Appendix EL - PGVIS.FOR	505
Appendix EM - Typical Use of REAL SCAN Programs	507
Appendix FA - Second Generation REAL SCAN Routines	511
Appendix FB - PZBUF.FOR	513
Appendix FC - PZCOMDAT.FOR	515
Appendix FD - PZDATA3.FOR	517
Appendix FE - PZFILTER.FOR	525
Appendix FF - PZLOGLOAD.FOR	527
Appendix FG - PZMAIN.FOR	529
Appendix FH - PZMULT.FOR	533
Appendix FI - PZNOISE.FOR	535
Appendix FJ - PZPROJ.FOR	537
Appendix FK - PZSCAN.FOR	541

Section	Page
Appendix FL - PZSCENE.FOR	545
Appendix FM - PZSLINIT.FOR	555
Appendix GA - Overhauser-Coons Routines	559
Appendix GB - LCOVRCNS.FOR	561
Appendix GC - LCSTRAIT.FOR	573
Appendix HA - Noise and Texture	583
Appendix HB - RLPCAL.FOR	585
Appendix HC - TRYTEX.FOR	587
Appendix IA - TSTBW.FOR	597
Appendix IB - SRCAMERA.FOR	599
Appendix JA - Analysis of Appendix J Solutions	603
Appendix JB - LOWER.FOR	613
Appendix JC - LOWDIST.FOR	619
Appendix JD - LOWANG.FOR	621
Appendix JE - LOWDDIST.FOR	623
Appendix JF - LOWDANG.FOR	625
Appendix JG - LOWSCALE.FOR	627
Appendix JH - LOWBITS.FOR	629

LIST OF FIGURES

Figure		Page
1	Display Window Projection	21
2	Functional Block Diagram of REAL SCAN System	22
3	Scan Line Geometry	31
4	Eye and World Coordinate Systems	33
5	ITEMBUF and Projection Screen	34
6	Screen and Eye Coordinate System	34
7-A	Start of the Flowchart for Subroutine Run	38
7-B	Completion of the Flowchart for Subroutine Run	39
8	Eye Coordinate System and Screen	40
9	Dropped Screen Corners	41
10	Scan Lines	44
11	Wedge Constants	46
12	Bresenham Type Scan Line	49
13	Visibility	49
14	Projection Triangles	52
15	Three Pixel Boundaries Along the IXS Axis	54
16	Scan Line Start Improvement	63
17	Noise Files	72
18	Filtering Concept	74
19	Breakpoint Calculation	75
20	Uniform Distribution Plot	77
21	Sun Angle Geometry	80
22	Altitude Map, Filter Size: 25, Distribution: uniform, Filter Type: simple average	81
23	Sunlight Picture, Filter Size: 25, Distribution: uniform, Filter Type: simple average	81
24	Sine Weighting Array for Filter Size of 7	82
25	Altitude Map, Filter Size: 25, Distribution: uniform, Filter Type: sin(x)*sin(y)	83
26	Sunlight Picture, Filter Size: 25, Distribution: uniform, Filter Type: sin(x)*sin(y)	83
27	Polar Weighting Array Filter Size of 7	84
28	Altitude Map, Filter Size: 25, Distribution: uniform, Filter Type: polar filter	85
29	Sunlight Picture, Filter Size: 25, Distribution: uniform, Filter Type: polar filter	86
30	Idealized Filtered Noise Contour	89
31	Sunlight Pictures as a Function of Filter Size	91
32	Sunlight Pictures as a Function of Polar Angle	92
33	Sunlight Pictures as a Function of Average Texture Slope	94
34	Sunlight Pictures as a Function of Cylindrical Angle	95
35	Square Interpolation Patch Geometry	97
36	rms Elevation Error versus Normalized Patch Size	99

Figure		Page
37	Radial Line Aliasing versus Contrast Ratio	105
38	Flowchart of Filter 1	107
39	REAL SCAN Block Diagram	114
40	Data Rates from the Processors to the Display	118
41	Block Diagram of Concentrator, Routing and Frame Buffer	121
42	Block Diagram of the Memory Controllers	122
43	A Mapping of Memory Pages to Display Space	123
44	Block Diagram of Concentrator Control of Figure 41	124
45	Concentrator Accumulation Architecture	126
46	Memory Map of Screen Coordinates to Accumulation Addresses	127
47	Block Diagram of Routing and Frame Buffer Architecture with Concentrator Accumulation	128
48	Block Diagram of Display Side of the Frame Buffer	129
49	Output from DICBWSA	136
50	Output from DICCOLSA	137
51	Flowchart of Image Generation and Text Routine Integration	142
52	ASCII Codes for Text Initialized in ASCH(I=1-17,J=1-39)	144
53	Array ASCH(I=1-17,J=1-39) with Character Strings in Relative Positions	147
54	Function Flowchart for Subroutine DICCHAR	148
55	Function Flowchart for Subroutine CHCONV	152
56	Image Generated by Sample Run	158
57	One Frame of City Block Motion Picture	159
58	Simulated Terrain from Noise Files	160
A-1	Illustration of the Field of View Pyramid	189
A-2	Fixed Eye's Field of View Projected to a Flat Earth Where τ_c is the Worst Case Field of View, R_{max} is the Horizon Limit	191
A-3	Additional Data Base Required for Motion Along the Line of Sight	195
A-4	Additional Data Base Required for Motion Perpendicular to the Line of Sight	196
B-1	Levels Required to Represent Gaming Area	198
C-1	Parallax Illustration	202
C-2	Set of Planes Which Match a Three Dimensional Scene for an Allowed Parallax Error	203
C-3	Coordinate System for the Locus Problem	204
C-4	Motion from E to P Achieving Display Limited Parallax	207
C-5	Projection Planes	208
C-6	Optimum Parallax Surfaces	211
C-7	Surfaces of Constant Parallax for Motion of E Along Y Axis	213
D-1	Definition of Square Grid	217
E-1	Display Plane Dropped to XY Plane	223

Figure		Page
E-2	Illustration of Dropped Display, Dropped Display Center, C, and a Nadir, n	224
E-3	Nine Possible Nadir Locations	225
E-4	Nadir Outside Screen Footprint, One Lower Bound Line	226
E-5	Nadir Outside Screen Footprint, Two Lower Bound Lines	226
E-6	Nadir Inside Screen Footprint	226
E-7	Nadir On Screen Boundary Line, One Lower Bound Line	226
E-8	Nadir at Corner	226
E-9	Eye Above the Screen Conditions	228
E-10	Eye Below the Screen Conditions	228
F-1	Side View of Scan Line	233
F-2	Top View of Scan Line	234
F-3	Illustration of the Scan Line Initiation Sequence	235
F-4	Another Illustration of Scan Line Initiation Sequence	236
F-5	Illustration of h/2 Starting Point	238
F-6	The Optimum Scan Line Initiation Procedure	240
I-1	Triangle Distribution Plot	251
I-2	Parabolic Distribution Plot	253
I-3	Cusp Distribution Plot	255
J-1	Lower Bound	257
K-1	Initial World, Eye, and Screen System at Time T = 0	264
K-2	World, Eye, and Screen System at Time> 0	265
K-3	Projection	267
L-1	Scene Using PG*.FOR Routines	271
L-2	Scene Using PZ*.FOR Routines	272
M-1	ANGFACT = 5 CPU TIME = 14 MIN 241 SCAN LINES	273
M-2	ANGFACT = 4 CPU TIME = 18 MIN 301 SCAN LINES	274
M-3	ANGFACT = 3 CPU TIME = 22 MIN 401 SCAN LINES	274
M-4	ANGFACT = 2 CPU TIME = 34 MIN 601 SCAN LINES	275
M-5	ANGFACT = 1 CPU TIME = 68 MIN 1201 SCAN LINES	275
M-6	ANGFACT = .5 CPU TIME = 135 MIN 2401 SCAN LINES	276
N-1	Sample Data Array (a) for Overhauser-Coons Patch Region, (b) for Area Interpolated by 4 Patches with Sample Data Array in the Form of a Grid	278
N-2	Variables in Overhauser Functions	279
N-3	Illustration of Variables (PICWIT, ISAMPSP, DPTDEN, DELTA), Sample Data Array, and Interpolated Points	281
N-4	Indices (a) for sample data array, (b) for adjacent points of patch region	283
N-5	Limits IPP and IUP	284
N-6	Example with DPTDEN=1.5	285
N-7	Initialization of Increments with XX	286
N-8	Change of Boundary Lines and Boundary Points after a Boundary Crossing (a) after an Increment along the Y Axis (b) after an Increment along the X Axis	288

Figure		Page
N-9	Plot of RMS Divided by Maximum Height of Test Function 127 vs. the Sample Spacing Divided by One-Half the Period, 512, for 0.25 Million Interpolated Points by Overhauser-Coons and Straight Line Approximation	303
N-10	Plot of ISAMPSP/512 vs. RMS/127 for PICWIT=2048.00 and Points 0.25 Million for the Overhauser-Coon's and the Straight Line Approximation	305
O-1	Scan Line Increment Δr_1 in the Hierarchy Level Bounded Between R and 2R	307
AB-1a	General Flowchart of DICOMED Picture Generation Routines	317
AB-1b	DICOMED Picture Generation Routines	318
AB-1c	DICOMED Picture Generation Routines	319
AB-1d	DICOMED Picture Generation Routines	320
BB-1	Terminal Session Photograph per Table BB-1	366
BB-2	Map of the DICOMED Image Field	367
BC-1	Output of the Color Bars Section of SRCAMSA	369
BC-2	Output of the Color Triangle Section of SRCAMSA	369
BC-3	Output from the Color Gradient Section of SRCAMSA	370
CA-1a	Flowchart for DICOMED Appendix	382
CA-1b	Flowchart for DICOMED Appendix	383
EM-1	Sample Picture	509

LIST OF TABLES

Table		Page
1	REAL SCAN SOFTWARE ROUTINES	29
2	INPUT DATA	36
3	OUTPUT DATA	37
4	REAL SCAN VARIABLES INITIALIZED IN PGSCENE	43
5	PGVIS.FOR INITIALIZATION VARIABLES	46
6	SECOND GENERATION REAL SCAN SOFTWARE ROUTINES	59
7	10cm GRID DATA POINTS	71
8	CONCENTRATOR TO FRAME BUFFER BUS PARAMETERS	119
9	PAGES OF MEMORY VERSUS FRAME BUFFER CYCLE TIME	120
10	SDICBW ARGUMENTS	138
11	SDICCOL ARGUMENTS	140
12	TEXT VARIABLES	145
13	CHARACTER CONVERSION VARIABLES	151
14	DATA BASE TESTING PARAMETERS	155
15	RECOMMENDED TASKS	175
A-1	CELL HIERARCHIAL DATA	192
B-1	MEMORY REQUIREMENT VERSUS GAMING PARAMETERS	198
B-2	MEMORY REQUIREMENT VERSUS GAMING AREA	199
C-1	DATA FOR FIGURE B-7	212
C-2	SCALED DISTANCE TO A PROJECTION PLANE PARALLEL TO EYE MOTION, FOR CONSTANT PARALLAX UPDATE RATE	215
E-1	CROSS PRODUCT SIGNS WITH THE EYE ABOVE THE SCREEN	227
E-2	NPIX AND IAXIS VERSUS KSC	231
H-1	VARIABLE RESOLUTION	249
N-1	O-C INTERPOLATION WITH PICWIT=7200	298
N-2	O-C INTERPOLATION WITH PICWIT=2048	299
N-3	O-C INTERPOLATION WITH PICWIT=2048	300
N-4	O-C and S-L INTERPOLATION COMPARISON I	302
N-5	O-C and S-L INTERPOLATION COMPARISON II	304
O-1	HIERARCHY LEVEL PARAMETERS	309
O-2	IMPROVED HIERARCHY LEVEL PARAMETERS	309
O-3	SCAN LINE PARAMETERS ACHIEVING A SPECIFIED RESOLUTION WITH A GUARANTEED UPPER BOUND ON THE NUMBER OF TEST POINTS ALONG THE SCAN LINE	311
O-4	SAMPLE 3 BIT, 4 BIT, AND 6 BIT SCAN LINE PSEUDO FRACTION	312
AA-1	EXPOSURE CALIBRATION	314
BB-1	DICOMED POSITIONING IN FIGURE BB-1	366
EM-1	INPUT INFORMATION FOR THE SAMPLE SCENE	508
JA-1	MATH OPERATIONS FOR EACH SOLUTION	605
JA-2	RMS ERROR FOR SOLUTIONS A, B, AND C VERSUS DIST	605
JA-3	RMS ERROR FOR SOLUTIONS A, B, AND C VERSUS ANG	606
JA-4	PERPENDICULAR ERROR FOR SOLUTIONS A, B, AND C VERSUS DIST	607
JA-5	PERPENDICULAR ERROR FOR SOLUTIONS A, B, AND C VERSUS ANG	607

Table		Page
JA-6	OPTIMUM VALUE OF SCALE VERSUS DIST	608
JA-7	RMS ERROR FOR SOLUTION D VERSUS SCALE APPROXIMATIONS . .	609
JA-8	RMS ERROR FOR SOLUTIONS D, C, AND D VERSUS DIST	609
JA-9	PERPENDICULAR ERROR FOR SOLUTIONS D, C, AND D VERSUS DIST	610
JA-10	PERPENDICULAR ERROR FOR SOLUTIONS D, C, AND D VERSUS ANG	611

SECTION I

INTRODUCTION

General

REAL SCAN - Real Environment Algorithm for Line SCANning - is a research program under Naval Training Equipment Center Contract N61339-80-D-0014. The objective of this program is to investigate techniques for providing computer synthesized imagery to a pilot trainee in a flight simulator. Existing computer image generators (CIG) have two basic deficiencies: the creation of polygon models of real world environments is an off-line process which is extremely manpower intensive and expensive, and the limitation on scene complexity (i.e., the number of polygons in the scene) caused by the limited number of polygons which can be processed in real time. REAL SCAN research addresses both of these deficiencies. The research investigates methods for automatic environment modeling by conversion from existing geodetic data combined with cultural planimetry of real world geographic areas. The REAL SCAN research has studied computer architectures for processing the environment model in real time with algorithms which are limited by the number of details which are capable of being displayed.

The goals of the REAL SCAN research are:

1. Produce highly complex terrain scenes with detail approaching the limit of a unique scene element for each image display picture element.

2. Allow direct and automatic conversion of real world height, color and reflectivity information into the CIG data base to model the real world.

3. Provide for large and variable gaming areas from which true perspective scenes can be computed and displayed in real time.

4. Provide compatibility of the REAL SCAN system with polygon model type CIG for optimum flexibility and economy in defining a total image generation and display system.

These general goals have been pursued via the following studies:

1. Investigate various means to compute display limited scenes with the goal of minimizing total scene computation time and minimizing computation complexity.

2. Investigate various regular methods for creating a data compression algorithm for a real world data base.

3. Develop computer simulation procedures for evaluating both the scene computation algorithms and the data base compression algorithms.

4. Develop software to make pictures on the DICOMED equipment by means of REAL SCAN simulation procedures.

This report documents the accomplishments made to date, develops and describes all algorithm and data base efforts, discusses the picture making software, presents potential architectures for implementing REAL SCAN, develops recommendations for improvements to the REAL SCAN algorithms and suggests further research efforts.

The Problem

One of the outstanding problems of visual simulation is the generation of a complex scene. Although satisfactory training can be achieved for many simulation tasks using current computer image generation (CIG) systems at their present stage of development, close approach to terrain in daylight reveals the severe limitations for tasks such as air/ground weapon delivery, confined area maneuvering, low level flight and harbor/channel navigation with both surface vessels and submarines.

With conventional CIG, (1,2) the environment is modeled by defining the coordinates of points in space, grouping pairs of points to define "edges", connecting edges to form polygons, assigning color and reflectivity to the polygons and grouping them to form polyhedra which approximate the environment to a level limited by the processing power of the real time CIG hardware. The number of edges processed per displayed scene is the usual metric for such hardware with the state-of-the-art currently around 8,000.

For modeling regular objects such as a runway, polygon modeling is economical in terms of the size of data base and the amount of processing hardware. However, a complex environment such as an area of countryside consisting of contoured terrain having detailed surface texture and solid objects such as trees, building, etc., on it, is not realistically reproduced by present CIG systems for two reasons: modeling cost and processing hardware limitations of the polygon modeling technique. Further development in polygon model type CIG processing hardware is in progress by several manufactures (the "edge

---

[1]Roberts, L. G., "Machine Preception of Three Dimensional Solids", TR315, MIT Lincoln Laboratory, May, 1963.

[2]Sutherland, I.E.; Sproull, R. F.; and Schumacher, R. A. "A Characterization of Ten Hidden-Surface Algorithms", Computing Surveys, Vol. 6, No. 1, March, 1974, pp. 1-55.

war") but the limiting factor is not so much the real time processing hardware as the cost of generating the increasingly complex environment models or data bases required.

Automatic Data Base Generation Problems

Studies (3) at the Naval Training Equipment Center (NAVTRAEQUIPCEN) on the automatic generation of polygon type data bases from stereo photographs showed promise for individual objects but underlined the great difficulty of the problem for terrain. Automating the production of polygon models of real world environments is a task being addressed by the operational equipment community specifically with regard to Cruise Missile guidance systems, and success is not expected for many years.

The reason for the difficulty can be appreciated if one looks at almost any aerial photograph and mentally tries to break it down into separate objects that can be approximated by polygons. Much of what is visible can only be recognized by a skilled photo-reconnaisance technician. The difficulty, therefore, is fundamental: the process, to be automated, requires a degree of artificial intelligence greater (for this task) than that of the average human being. This basic difficulty emphasizes the need for considering the entire image generation system from data base modeling to image display when developing an advanced CIG concept, and for evaluating non-polygon type data bases as a way out of the automatic data base generation problem.

Development of New System Concept

If one attacks the goals individually, the following system characteristics emerge. First of all, high scene detail requires a substantial data base; however, the storage requirements are less if two of the ground coordinates are addresses rather than data. Large areas of the world are mapped as uniform grid models, such as the Terrain File of the Digital Landmass System (DLMS) developed by The Defense Mapping Agency and the ortho color photos produced by the U.S. Geological Survey (4). Models of this type are relatively easy to generate by low-skill processes (as compared with polygon models) and offer the greatest possibilities for use in a CIG system of the type being considered.

---

[3]Breglia, D. R., "Automating CIG Data Base Development", Technical Report NAVTRAEQUIPCEN IH-318, Naval Training Equipment Center, Orlando, Florida.

[4]Defense Mapping Agency "Product Specifications for Digital Landmass System (DLMS) Data Base", July, 1977.

To obtain constant angular resolution of visible details over the display, the corresponding ground resolution required decreases with range from the viewpoint. Since a change of viewpoint means all parts of the data base will at some time be viewed at close range, the data representing the gaming area must all be available at high resolution. However, for any given viewpoint, to avoid having to process high resolution data representing distant terrain in real-time, a lower-detail version of the data may be used for distant parts of the scene. This leads to the concept of a hierarchy of resolution levels in the data base, analagous to the levels of detail used in polygon model CIG.

However, whereas the levels of detail in a polygon model data base must all be created by the modeler, with a regular grid data base each level can be derived automatically off-line.

In current real-time CIG systems, sorting algorithms are used to produce the display image following transfer of the polygons forming the scene into the display plane. The processing power required for sorting grows proportionally to the square of the detail processed. For systems dealing with a great amount of detail in real time, an architecture is needed in which processing power grows only linearly with detail, and this should be a feature of the new system. To process and display a great amount of displayed detail in real time a modular structure is indicated making use of basically independent parallel processors, each a pipeline.

The use of a ground coordinate grid allows incremental processing in a fixed data base. This feature eliminates the need for floating point operations and repeated need for divide operations. Hence, computationally efficient integer algorithms can be developed which incorporate a pseudo exponentiation upon crossing from one level of the data base to the next (using a step of 2:1 in linear detail between levels).

Finally, the generated image should be in one of the standard television formats, e.g., 1023 lines/frame, 30 frames/sec. This allows existing displays to be used and combined image generation systems to be developed using both polygon and grid data forms of CIG.

General REAL SCAN System Description

REAL SCAN utilizes a uniform square grid data base to produce highly detailed terrain displays. A hierarchy of resolution levels models the environment and many parallel-pipelined processors perform the required visibility determination followed by perspective transformation from world coordinates to display coordinates. The processing is based upon computationally efficient integer algorithms.

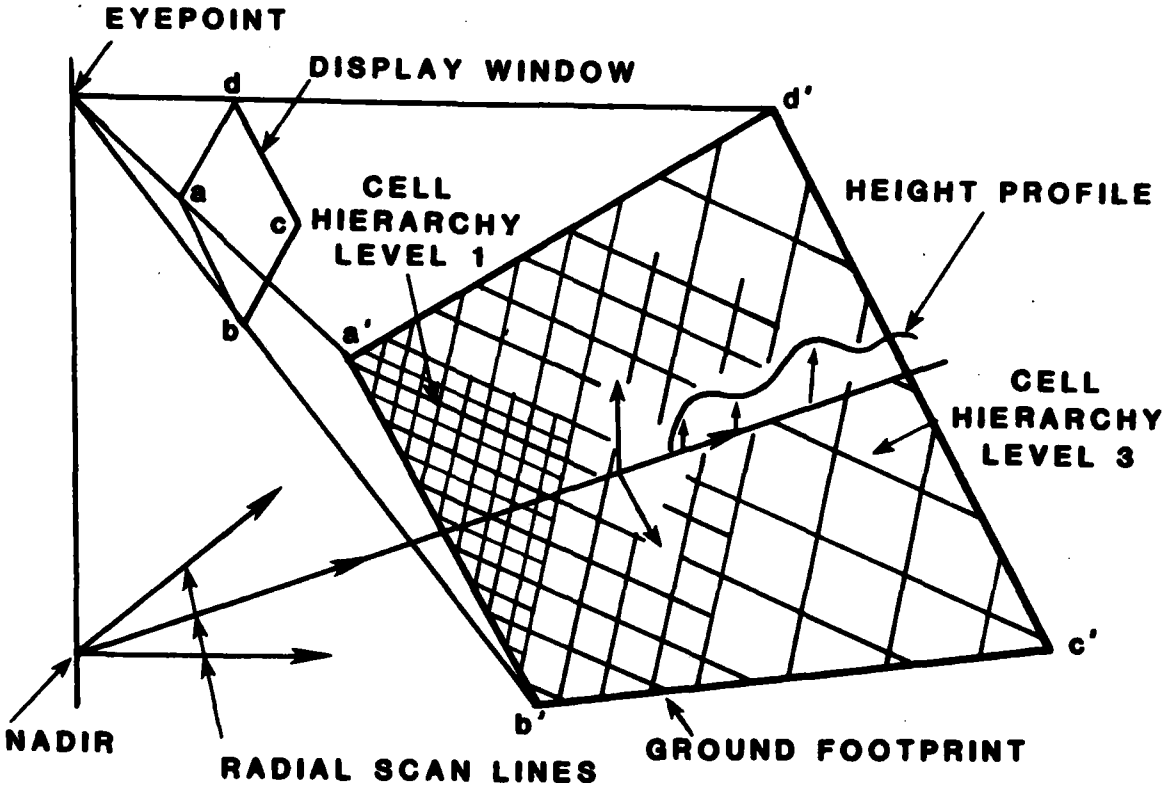


Figure 1. Display Window Projection.

Figure 1 shows the projection of the display window on to the ground plane to give the "ground footprint" from which data must be transformed into the display window coordinates. Each level of the data base consists of a square array of "cells", each cell containing digital color and height data for that elementary area of the terrain modeled. Although all levels of the hierarchy are available over the whole gaming area, level 1 is used for parts of the terrain nearest to the eyepoint and levels 2, 3, etc., are used as the range increases. The determination of which points are visible is made by radially scanning the ground footprint in a manner similar to that used in radar landmass simulation. Each radial line drawn from the nadir point (the point directly below the eyepoint) through the ground footprint maps into a line across the display window. As explained later, a frame buffer is used to store the information to be displayed such that the information can be read out in accordance with any standard television scanning pattern for display (e.g., with horizontal scan lines).

New scan lines are initiated through the data base as the scanned distance from the nadir grows, such that the visible part of the data

base is correctly sampled. It is necessary, for acceptable image quality, to avoid aliasing (i.e., interaction between the regular structure of the digital data base and the regular structure of the scanning pattern which generates spurious detail in the displayed image). The information in each displayed picture element (pixel) must be derived from the data stored in several adjacent cells, in accordance with a weighting algorithm that varies with the position of the pixel in the window. This requires averaging, in real time, of the processed data.

Real time averaging over many cells is impracticable, but the hierarchical data base concept, by providing pre-averaged information, reduces this problem to manageable proportions. Filtering is used to eliminate any remaining unwanted image components.

In transforming data from the ground plane into the viewing window, a perspective transformation has to take place. Each scan line through the data base is dealt with separately, the spacing between image details along the line being changed to allow for the oblique view of the terrain. Details which would be hidden by high ground, are of course omitted from the display.

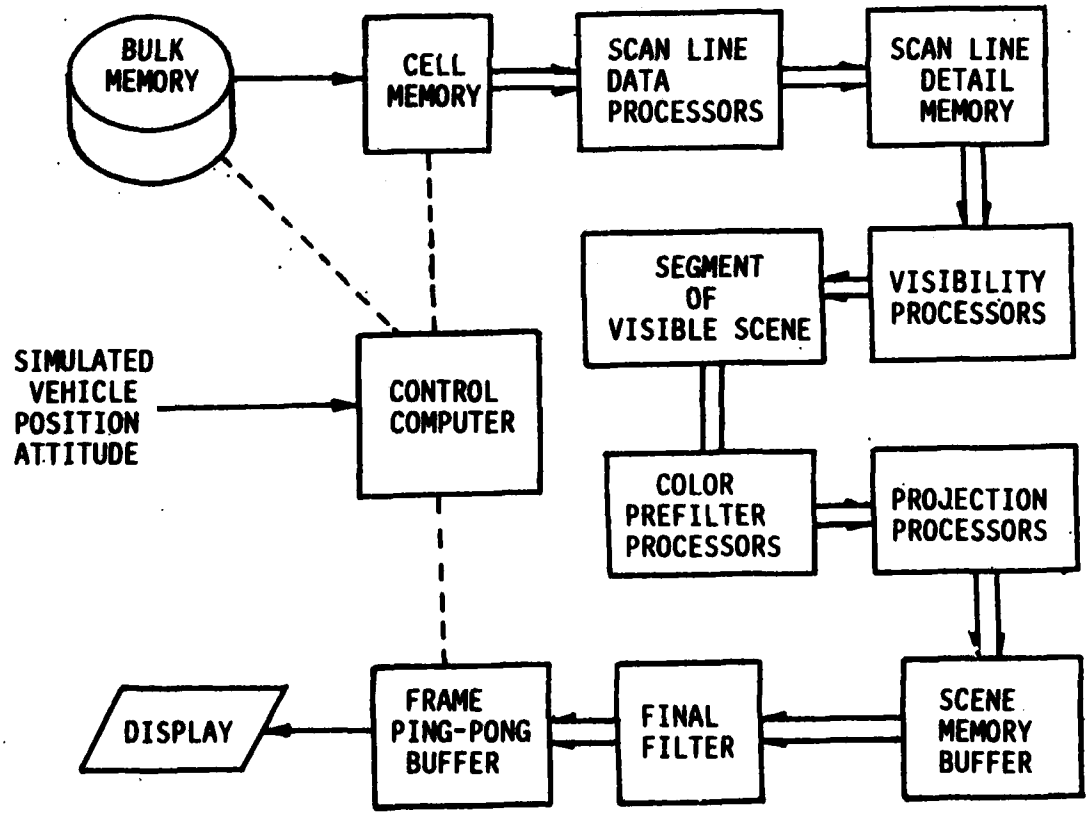


Figure 2. Functional Block Diagram of REAL SCAN System

Figure 2 is a functional block diagram of the proposed system. The color and height data for the complete gaming area are stored in bulk memory; a videodisc is preferred in view of the large number of bits that can be stored in a compact space with reasonably rapid access (this will be considered in more detail later in relation to the generation of the data base.) The cost of videodisc bulk memory storage is currently estimated to be $0.10/megabyte as compared to magnetic disc storage costs of $3/megabyte (5).

The control computer accepts simulated vehicle position and attitude data from the host computer and is used to control the flow of data to the REAL SCAN system. For each television frame of picture information, the following processing steps need to be carried out:

1. Obtain observer view point location and viewing direction (simulated vehicle location and attitude) from the host computer.

2. Compute, in the control computer, the viewing pyramid intercepts with the ground reference plane.

3. Use linear and angular rates of the simulated vehicle to predict the potential field of view for future television frames.

4. Determine, using the control computer, which blocks of data representing hierarchical areas of terrain in the bulk memory are to be accessed for generating the required scene, given the nadir point location and field of view.

5. Using the control computer, transfer these blocks of data containing all the information necessary to compute a display of the current field of view plus additional information for future frames, to the "cell memory" (a highspeed, virtually addressed random access memory). After the cell memory has been initially loaded, there is always information available to generate the next frames. The cell memory is then the first stage of a pipeline computational process leading to a frame generation. After the current frame's information is read from the cell memory, those blocks which are no longer in the potential field of view are overwritten by new information.

6. Scan the cell memory using a group of identical scan line data processors. Each processor calculates the elevation and reflectivity information along its scan line using the

---

[5]"Videodisc Based Storage Technology", Computer, Vol. 13, No. 6, pp. 87, June, 1980.

regular grid data in the cell memory. The radial lines are scanned in parallel groups to minimize the time taken for this process. The data is assembled in the scan line detail memory, data for each scan line being separately stored.

7. Transfer the data, in parallel as needed, from the scan line detail memory to the visibility processor and carry out visibility processing on each line of data. This process determines which parts of the data from the cell memory are to contribute to the displayed scene and which are hidden from view. The algorithm used to determine visibility is similar to that used to determine radar shadows in digital radar landmass simulation systems (6,7), suitably modified for the hierarchical data structure.

   At this stage only the elevations of cell data enter the computation; cell reflectivity/color information is carried with no modification.

8. Accumulate, in the segment of visible scene buffer, reflectivity/ color data which is computed only for visible point, in first-in first-out format, to allow for timing variability in the pipeline up to this point.

9. Pass the data from the segment of visible scene buffer to the color pre-filter processor and perform initial filtering along the lines to avoid aliasing while correctly passing sudden luminance changes corresponding to non-aliased edges in the scene.

10. Pass the data from the color prefilter processor to the projection processor where the reflectivity/color data is perspectively transformed from ground coordinates to display coordinates.

11. Pass the output from the projection processor to the scene memory buffer.

12. Perform final filtering using a weighted average processing filter to generate display pixels.

---

[6]Greenly, R. and Marchegiani, D. "Digital Radar Landmass Simulation", Proceedings of the Fourth Annual Naval Training Device Center/Industry Conference, pp. 131-141, November, 1969.

[7]Hoog, T.; Dahlberg, R.; and Robinson, R. "Project 1183 - An Evaluation of Digital Radar Landmass Simulation", Proceedings of the Seventh NAVTRAEQUIPCEN/Industry Conference, pp. 55-79, November, 1974.

13. Pass the data into a random write/serial read ping-pong frame buffer.

14. Read the frame buffer continuously into the display.

The REAL SCAN Data Base

The desired data base form is a hierarchy of two-dimensional arrays of elevation and reflectivity/color. Each array corresponds to the entire gaming area at a specific ground resolution. The size of the data base is a function of the desired ground detail size, the size of the gaming areas, and the amount of information stored at each array location (see Appendices A and B).

For example, consider a gaming area of 100 km^2, a desired ground detail size of 0.1m, and 48 bits of information stored at each array location. The largest array, corresponding to the highest resolution, would contain 10^{10} grid positions or cells and $5x10^{11}$ bits of information. If the cell size in each array is doubled or the resolution is half that of the adjacent array in the hierarchy then the total data base required for the entire hierarchy is (1 + 1/4 + 1/16 + 1/64...) times the information stored in the highest resolution array, or approximately $7x10^{11}$ bits for the total data base. Although this is a large number, the optical disc digital storage technology has progressed to the point where such large data bases are feasible. In fact, optical disc storage configurations have been proposed for systems having a capacity of 10^{14} bits (8). The next question is: What is the value of a regular grid data base with 0.1m ground resolution? For close approach to the ground, as with a helicopter maneuvering in a confined area landing site, a minimum range from the pilot's eye to ground could be taken as 5m.

Considering a television display with 1000 horizontal scanning lines and 50° side field of view (a typical CIG "viewing window"), the ground resolution (detail size) should be of the order of 5mm. This is smaller than the figure of 0.1m chosen above by a factor of 20. The desired line detail is provided in the display by interpolating special functions.

Example of Data Base Construction

A logical set of steps to create the hierarchial data base can be illustrated as follows:

1. DLMS terrain file (Level I) consists of an elevation array with a grid spacing of approximately 100m. This array would

---

[8]Ammon, G "Wide band Optical Disc Data Recorder Systems", SPIE Vol. 200 Laser Recording and Information Handling, pp. 64-72, 1979.

be interpolated using bicubic functions to form the initial hierarchy arrays. In the case of a desired 0.1m ground detail the number of arrays generated would be 10 corresponding to cell sizes of 0.1, 0.2, 0.4 ..... 25.6, 51.2m.

2. The enhancement of the elevation models can be carried out in a variety of ways. By making use of the DLMS cultural file, generic cultural features which have been stored in a feature library in the gridded elevation format at the various resolution levels can be added to (or subtracted from) the initial elevation data. The generic models can be obtained by photogrammetric techniques using stereo photos of representative real world cultural features.

3. Reflectivity/color information can also be generic or specific. This information can be derived from digitized/quantized color ortho photos. An ortho photo is a product of automatic analytic stereo photogrammetric equipment in which the image displacement due to relief is corrected. If the reflectivity/color model for a specific $100km^2$ area is desired, approximately fifty color aerial photographs would be required to obtain the raw reflectivity photographic data to 0.1m cell size. If the generic route is chosen, appropriate color ortho photos of representative cultural features are utilized to form a library which corresponds to the generic elevation library.

4. The high resolution reflectivity information is interpolated to form the reflectivity data for succeeding lower levels in the hierarchy. In this way scene compatibility across hierarchical levels is ensured.

5. In addition, elevation and reflectivity/color function codes can be assigned to each cell. These codes describe the fine detail referred to in 2 above and provide for blending of reflectivity/color and terrain height between cells so as to avoid discontinuities. This scheme enables the aim of making the system generate as much detail as can be displayed to be achieved (display limited detail) without excessive memory, although individual objects cannot be dealt with in this way. Further elaboration to the system shown in Figure 2 is, of course, required.

The net result is a highly automated data base modeling system which is not dependent on subjective decisions by the modeler and can be implemented using available data and technologies in the photogrammetric community.

The concept of essentially overlaying a digital elevation model with photographically derived reflectivity data to form an environment

model is not new (7,8,9), but the use of such a model in a real time CIG system has not been attempted to our knowledge.

The reasons for pursuing this approach to environment modeling are summarized as:

1. The need for efficiently produced low cost, highly detailed environment models.

2. The availability of equipments and technologies which can produce highly detailed environment models in a uniform grid format.

3. The availability of technology to store and access large amounts of data.

4. The feasibility of an automatically generated level of detail hierarchy which greatly reduces computational processing load, since simulated long ranges utilize coarser levels of the hierarchy.

The environment model described has been initially restricted to terrain surfaces which are single valued in elevation, and the acceptability for training using such a model has not been evaluated.

---

[9]Unruh, J.; Alspaugh, D.; and Mikhail, E. "Image Simulation from Digital Data", Proceedings of American Congress on Surveying and Mapping, 1977.

SECTION II

ALGORITHMS AND SOFTWARE

The purpose of this section of the report is to document the REAL SCAN algorithms and software. First, the software used to create the May 1981 black and white movie will be described, then software improvements will be presented. The movie software creates a visual image of what an observer would see if looking through a window of a cockpit. The REAL SCAN "Eye" is the observer's eye. It is assumed that the observer is looking directly into the center of his window, the REAL SCAN "Screen". The environment that the observer sees is the REAL SCAN "World". This environment is modeled by REAL SCAN, with reference to a horizontal ground plane with integer coordinates. Each integer coordinate contains a single height value and a reflectance value. The REAL SCAN system uses the location and the orientation of the observer relative to the world origin to create a perspective view of the environment. This view or scene is displayed on a 512 by 512 picture element display screen. The REAL SCAN software files are listed in Table 1.

TABLE 1. REAL SCAN SOFTWARE ROUTINES.

1.	PGCOMDAT.FOR	
2.	PGBUF.FOR	
3.	PGMAIN.FOR	integer
4.	PGSCENE.FOR	floating point
5.	PGSL.FOR	floating point
6.	PGVIS.FOR	integer
7.	PGSCAN.FOR	integer
8.	PGPROJ.FOR	integer
9.	PLOGLOAD.FOR	integer
10.	PGFILTER.FOR	integer
11.	PGDATA.FOR	integer

These FORTRAN files are listed in Appendices EB through EL. Appendix EM describes how these files are used. This includes: (1) how to compile the routines, (2) how to link the routines, (3) a sample picture and the input information used to create this sample picture. Appendix J develops the mathematics for the lower bound computation and Appendix K develops the mathematics for the projection processor.

This documentation will explain the function, derive the mathematics, and discuss any special techniques used in each routine. A brief overview of the REAL SCAN software technique for creating the scene is required to familiarize the reader with the technique for producing a REAL SCAN visual image.

The function of the routines in Table 1 is to produce a visual image whose resolution is display limited, (i.e., The resolution of the scene is dependent on the resolution of the display device.) This image is created by scanning a data base, which is a mathematical model of a terrain surface defined as having a single elevation for any location on the surface, with many radial scan lines. Each scan line extends from the nadir, the point on the ground plane (Z = 0) directly below the eye to a point on the ground plane which corresponds to an outer screen boundary. If the database is flat then this point corresponds to a point on the outer edge of the footprint, the poloygon defined by projecting the four corners of the screen onto the ground plane. Figure 3 depicts this idea.

The first scan line extends from the nadir to corner 4. Each successive scan line is found by rotating the previous scan line through an angle, ANG, such that at least two scan lines cut through each picture element, called a pixel. A pixel is one of the (512,512) squares of the screen of Figure 3. Hence, the display limit has been arbitrarily chosen to approximate TV, and match the DICOMED picture making facilities available.

Once a scan line has been identified, a Bresenham (12) type line drawing algorithm is used to increment through a square grid database in a regular fashion. This integer increment is of proper world dimension to allow at least two world data points to map to each pixel. As the range from the viewpoint increases the required ground resolution decreases. Therefore, a database with many levels of resolution has been developed as a part of the REAL SCAN concept. Each level of resolution, called a hierarchy level, will have half the resolution of the previous hierarchy level.

Each hierarchy level is identified by a pseudo exponent called ICL. The relative integer increment, NCR, between data points in a particular hierarchy level is identified by

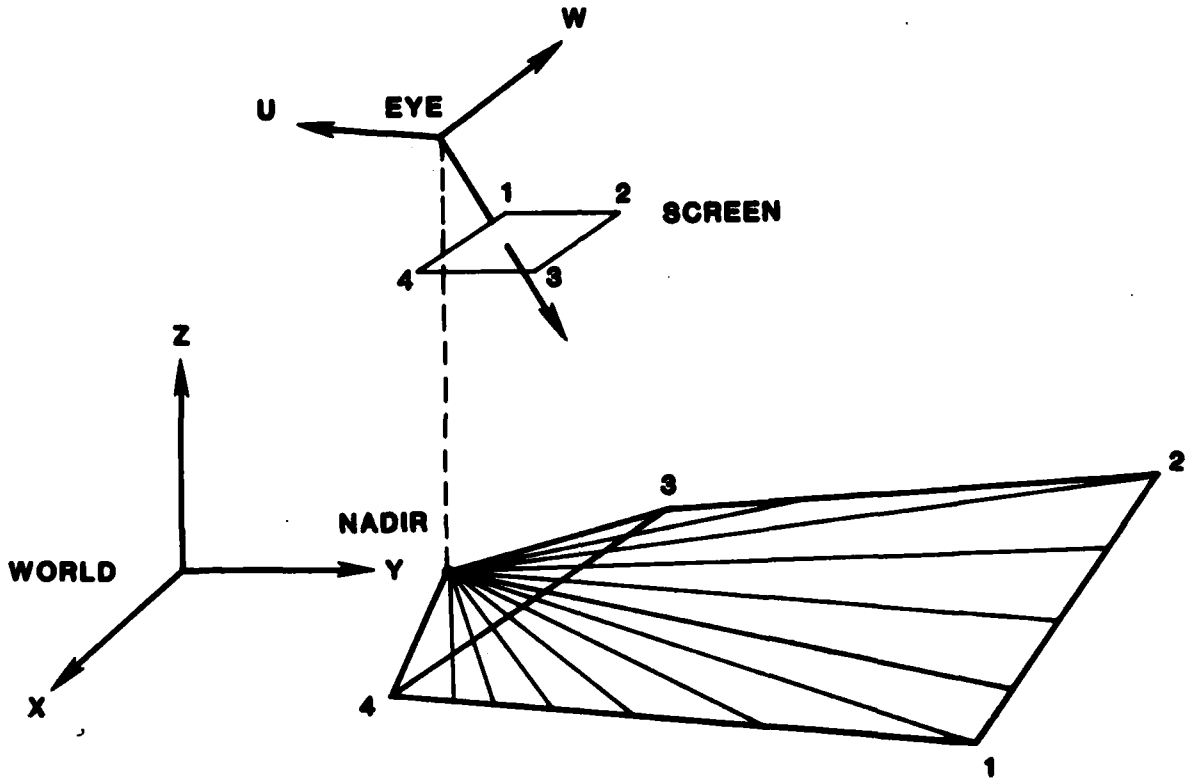


Figure 3. Scan Line Geometry.

$$NCR = 2^{(ICL-1)} \qquad (1)$$

The database is required to provide height and reflectance information for any valid database coordinate (IPP(1), IPP(2)). The database consists of mathematical functions to model the environment. Therefore, a large variety of databases can be developed in a short period of time.

The height data is used to compute visibility. Visibility is determined by the method of similar triangles. These triangles are in a common plane defined by the eye point and the scan line. With this technique, the test point is compared against the last visible point.

Once a visible point is found, the intensity of the point is calculated and an integer projection routine computes the screen location for that point. These screen coordinates are used as an address to sum the reflectance value in a screen buffer called ITEMBUF. This buffer is then filtered and the result is ready for display.

The development of a fast and efficient CIG (Computer Image Generation) system requires as few divides and multiplies as possible, except in powers of two since this implies a shift of the bit positions. Also integer arithmetic is preferred to floating point unless the routine has a negligible impact on the computation time. Several techniques have been developed to minimize divides. These techniques are employed in the integer **FORTRAN** routines of **REAL SCAN** listed in Table 1.

Coordinate Systems

The following right handed orthogonal coordinate systems are used in the **REAL SCAN** routines:

1. World
2. Eye
3. Screen
4. Nadir Centric World
5. Eye Centric World

The coordinate systems are defined below.

WORLD

The World coordinate system uses the symbols X,Y,Z to represent the axes. The X and Y axes map directly to the square grid data base. The Z axis represents altitude above a flat world.

EYE

The Eye coordinate system uses U,V,W for axes symbols. The U and W axes map directly to the screen axes, while the V axis represents the view vector. The Eye coordinate system is related to the World coordinate system by the eye location, (IXE,IYE,IZE), and the rotation matrix, ROT. ROT defines the present eye orientation relative to the reference eye orientation in which, U=X, V=Y, W=Z. These systems are depicted in Figure 4.

A point in the World coordinate system (X,Y,Z) is transformed into a point in the Eye coordinate system (U,V,W) by the matrix multiplication

$$\begin{bmatrix} U \\ V \\ W \end{bmatrix} = [ROT] \begin{bmatrix} X-IXE \\ Y-IYE \\ Z-IZE \end{bmatrix} \qquad (2)$$

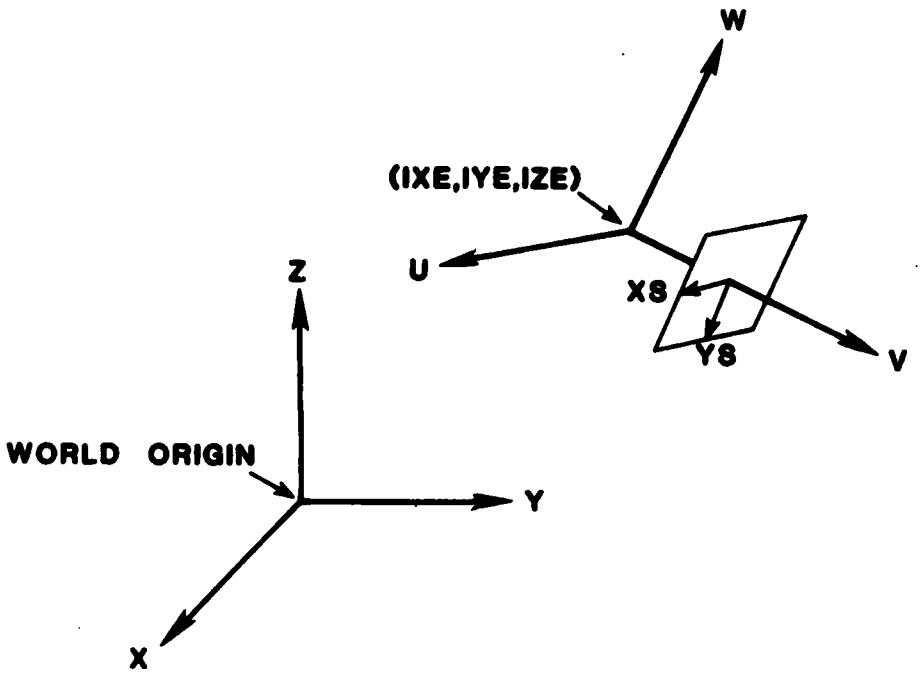


Figure 4. Eye and World Coordinate Systems.

SCREEN

The display screen, called ITEMBUF, will consist of an array of (NX,NY) pixels. This screen buffer can be accessed in the same manner as any FORTRAN array. Therefore, the top lefthand pixel will have an address of (1,1) and the bottom righthand corner will have an address of (NX,NY). The simplest screen structure, from a mathematical standpoint, for the projection algorithm is a center oriented screen. If the screen has an odd number of pixels, then the origin is in the center, but if the screen has an even number of pixels, then the origin is arbitrarily chosen to be in the pixel to the lower right of the center. Figure 5 depicts both screen formats.

A point on the screen (IXS,IYS) is mapped into a point on the screen buffer (IXBS,IYBS) by the following equations.

$$IXBS = IXS + NX/2 + 1 \quad (3)$$

$$IYBS = IYS + NY/2 + 1 \quad (4)$$

The rectangular screen with square pixels and physical dimensions LX and LY will be oriented perpendicular to the view vector. The center of the screen will be at any arbitrary location (IUS,IVS,IWS), as long as IVS is greater than zero. The eye U and W axes correspond to the XS and -YS axes of the screen. The Screen and Eye coordinate systems are depicted in Figure 6.

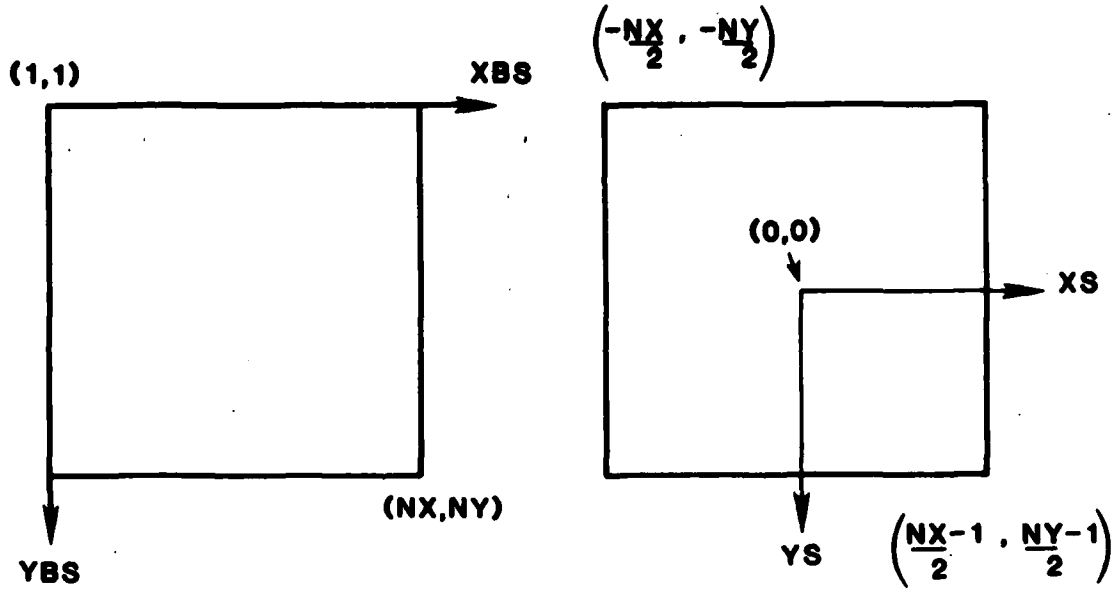


Figure 5. ITEMBUF and Projection Screen.

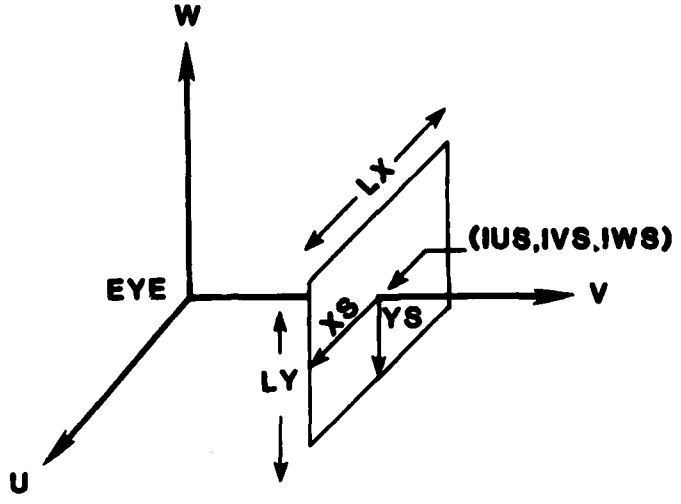


Figure 6. Screen and Eye Coordinate System

NADIR CENTRIC WORLD

The Nadir Centric World coordinate system has the same orientation as the World coordinate system except the origin is at the nadir.

EYE CENTRIC WORLD

The Eye Centric World coordinate system has the same orientation as the World coordinate system except the origin is at the eye.

Fortran Routine Documentation

The FORTRAN files are separated into two types, FORTRAN routines and FORTRAN common modules. The common modules (PGCOMDAT.FOR and PGBUF.FOR) aid in program development.

PGCOMDAT.FOR

PGCOMDAT.FOR is a FORTRAN file which identifies a global common area of memory. Variable types and dimensions are specified in this routine. PGCOMDAT is incorporated into other routines by the non-standard FORTRAN statement INCLUDE PGCOMDAT.FOR.

PGBUF.FOR

PGBUF.FOR identifies the Screen Buffer, ITEMBUF, as an array with dimension (512,512) of two byte signed integer. This common area of memory is only allocated, when needed, to a routine through the INCLUDE statement.

PGMAIN.FOR

PGMAIN.FOR is the main FORTRAN file which controls the program flow. When this program is executed, the following menu is given to the user:

1. RUN SCENE
2. WRITE ITEMBUF
3. READ ITEMBUF
4. CREATE A PICTURE ON THE DICOMED
5. STOP

The FORTRAN file, PGMAIN.FOR, consists of the menu and the following subroutines: READ, INPUT, OUTPUT, TESTPR, RUN. These subroutines are described below:

1. Subroutine Read

READ allows the user to read a previously created scene into ITEMBUF with the intention of creating a picture on the DICOMED, a machine which creates a Polaroid picture of ITEMBUF, by converting the reflectance value into light intensity. The most efficient method, which we have found, to store large amounts of data on the VAX-11/780 is by the

"FORMAT (66A2)" statement. This format converts INTEGER*2 data into ASCII format. The read has the following format:

```
      READ (IFILE,1000) ((ITEMBUF(I,J),I=1,IFILDIM),J=1,IFILDIM)
1000  FORMAT(66A2)
```

IFILE specifies which file is to be read and IFILDIM is the dimension of the screen in pixels.

2. Subroutine INPUT

INPUT allows the user to input all the required information to compute a scene. Table 2 identifies this input information.

The screen center and the screen length define the field of view, while screen size and screen length define the number of square pixels in the screen. IFIRST, IEND, and ANGLEFACTOR are used to define the scan lines. Test pictures are created in a short period of time by using the following two techniques: (1) Identify a large scale factor for the angle between scan lines (ANGFACTOR). This technique samples the scene. (2) Identify the starting scan line (ISTART) and the last scan line (IEND) numbers, such that only a wedge of the scene is computed.

TABLE 2. INPUT DATA

FORTRAN VARIABLES	DESCRIPTION	VALUES
(IXE,IYE,IZE)	EYE LOCATION	----------
PITCH BANK HEADING	EYE ORIENTATION	----------
(IUS,IVS,IWS)	SCREEN CENTER	0,1,0
LX,LY	SCREEN LENGTH	1,1
IFILDIM	SCREEN SIZE	512
IDATBAS	DATABASE CHOICE	(1, ... 5)
ISTART,IEND	FIRST AND LAST SCAN LINES	1,2000
ISCALE	THE SCALE FACTOR FOR THE ROTATION MATRIX	2500
IDIV	A DIVISION FACTOR TO PREVENT OVERFLOW	3
ANGFACTOR	A SCALE FACTOR FOR THE ANGLE BETWEEN SCAN LINES	.86
ICASE	A VARIABLE TO SELECT ONE OF THREE ROTATION MATRIX APPROXIMATIONS	3

3. Subroutine Output

OUTPUT prints the following pertinent information shown in Table 3. This data is used to identify the scene, and to aid in program development.

TABLE 3. OUTPUT DATA

NAME	DESCRIPTION
ROT(3,3)	The rotation matrix which defines the eye's orientation
PROJ(4)	An array which identifies if a corner projects to the ground (1) or the horizon limit (0)
NADIR	A flag which identifies if the nadir is outside (0) or inside (1) the footprint
COR(4,3)	An array which contains the location of the 4 corners in EYE CENTRIC WORLD coordinates
CW(4,2)	An array which contains the 4 corners of the footprint
ANG	The angle between scan lines in radians
CANG(4)	The angle through the nadir to each corner relative to corner 4, thus CANG(4) = 0

4. Subroutine Testpr

TESTPR, which is for test purposes only, will find the screen location of any world point. The standard floating point projection technique for finding the screen coordinates is given by

$$XS = ICONST*UP/VP \quad (5)$$

$$YS = -CONST*WP/VP \quad (6)$$

(UP,VP,WP) is the point in the EYE coordinate system. ICONST is a factor which is the product of the number of

pixels per unit screen length and IVS. XS and YS are the screen coordinates. Subroutine TESTPR aided in program development.

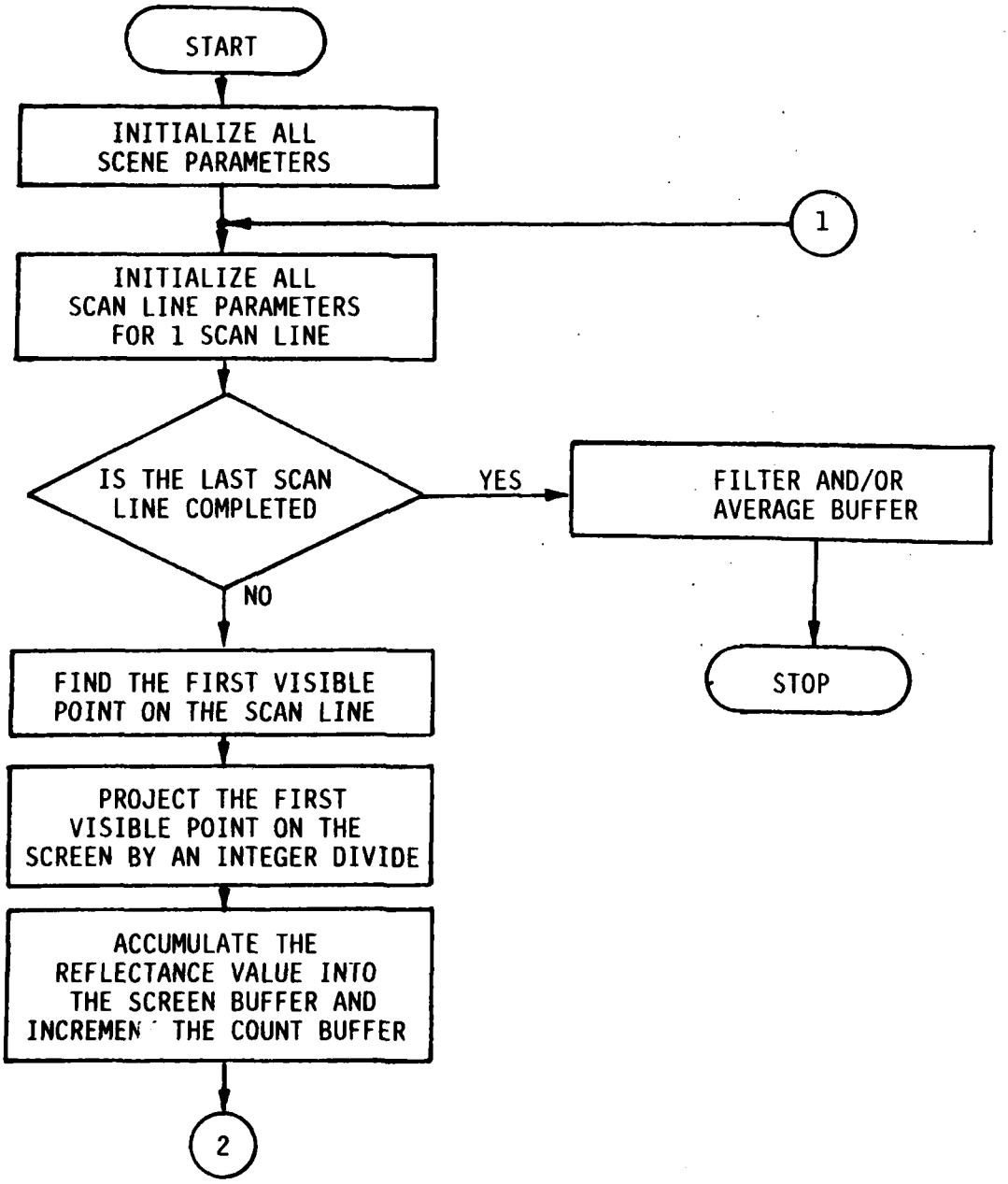


Figure 7-A. Start of the Flowchart for Subroutine Run.

5. Subroutine Run

RUN is used to control the program flow for computing the scene. RUN calls all the FORTRAN subroutines in the proper order to create the scene. This subroutine is depicted in the flowchart of Figures 7-A and 7-B. Each rectangular block of the flowchart identifies a function which is performed by one of the subroutines listed in Table 1.

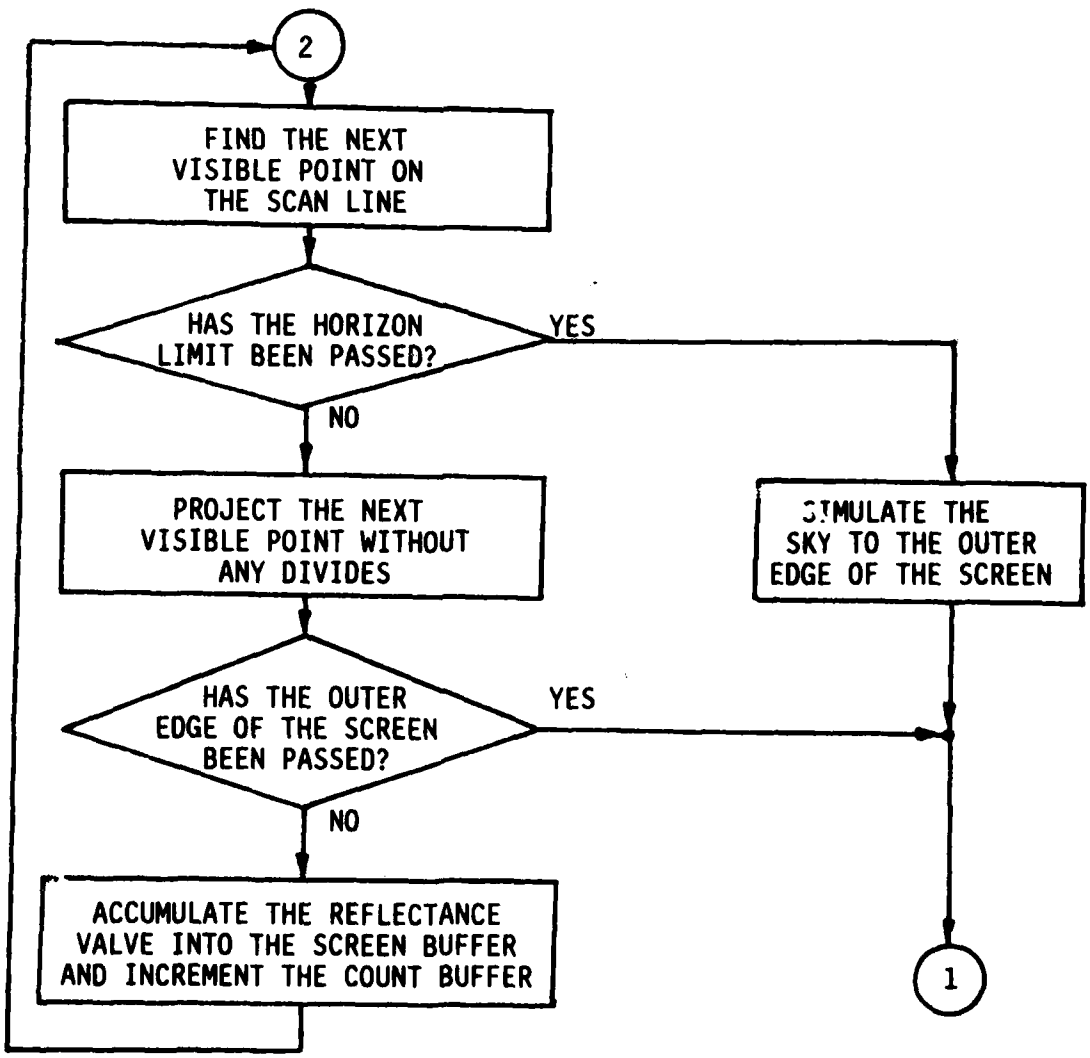


Figure 7-B. Completion of the Flowchart for Subroutine Run.

PGSCENE.FOR

The purpose of this subroutine is to initialize or compute all the scene constants. Figure 8 depicts the EYE coordinate system and the screen. The eye orientation is defined by the pitch, bank, and heading angles. The eye location in WORLD coordinates is (IXE,IYE,IZE). The screen corners are numbered from one to four as shown in Figure 8. These numbers define the indices of the variables in Table 3. The scene constants are described below.

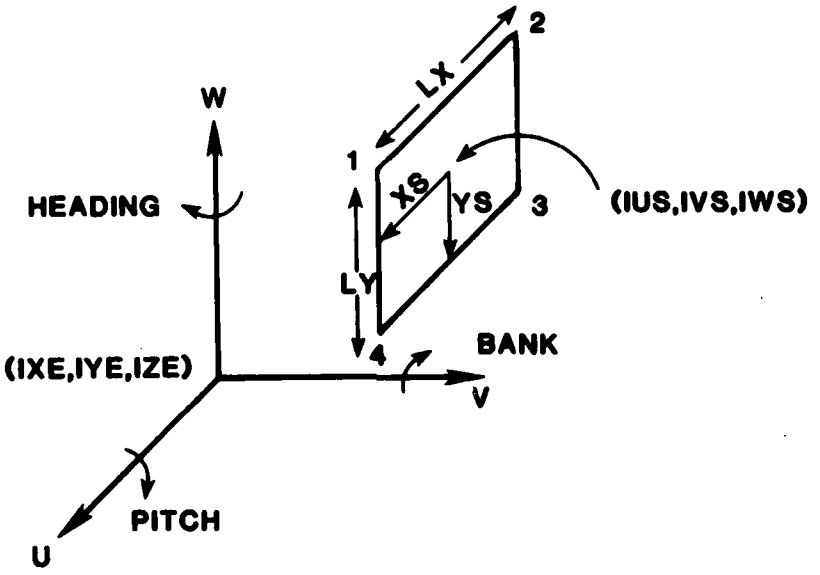


Figure 8. Eye Coordinate System and Screen.

1. ROT(3,3)

   ROT is the rotation matrix which defines the eye orientation. ROT is computed from the matrix multiplication of the matricies for heading, pitch, and bank in that order.

2. IROT(3,3)

   IROT is the scaled integer representation of ROT. (i.e., IROT = ISCALE*ROT)

3. COR(4,3)

   COR is the location of the four Screen corners in the EYE CENTRIC WORLD coordinate system. COR is computed by the matrix multiplication of the constant location of the corners in the EYE coordinate system by the inverse or tranpose of ROT.

4. CMAG(4)

CMAG is the distance from the nadir to each corner along the flat earth, (i.e., Z =0). Figure 9 shows the location of the dropped corners. The asterisk (*) used in Figure 9 denotes all possible values rather than a single value, (i.e., COR(1,*) means the point COR(1,1), COR(1,2), COR(1,3)). This notation is used throughout this paper.

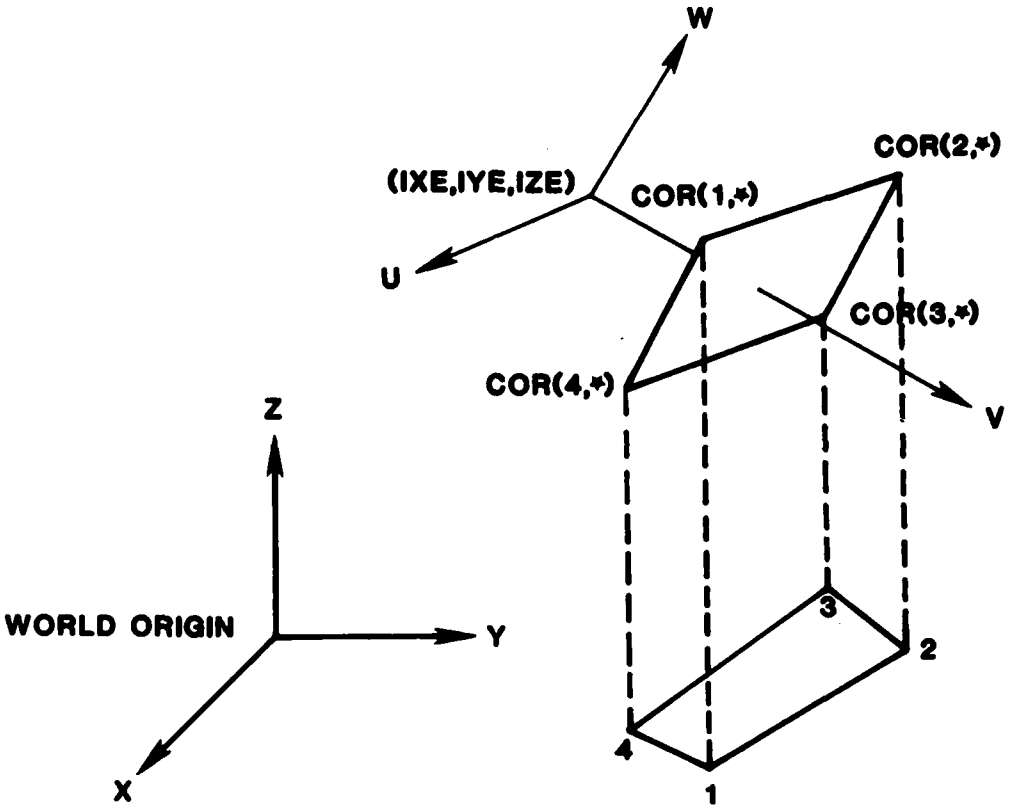


Figure 9. Dropped Screen Corners

5. CROSS(4)

CROSS is the cross product of adjacent dropped screen corners in the order 4,1,2,3,4. The corss products are in the Z direction in Figure 9.

Thus, CROSS(1) corresponds to COR(4) X COR(1), while CROSS (4) corresponds to COR(3) X COR(4). Sign changes in CROSS determine where the nadir is in relation to the footprint on the ground.

6. CANG(4)

CANG is the positive angle around the nadir from COR(4) to each of the other corners. The arctangent function is used to compute these angles.

The current version of PGSL.FOR, the routine which defines each scan line, requires that CANG be such that:

$$CANG(3) \geqq CANG(2) \geqq CANG(1) \geqq 0$$

This requirement simplifies scan line generation.

7. NADIR

NADIR is a flag which indicates whether the nadir is inside (NADIR = 1) or outside (NADIR = 0) the footprint. If all the cross products have the same sign then the nadir is inside the footprint.

8. ANG

ANG is the angle in radians between each scan line. The goal in designing these routines is to calculate ANG such that at least two scan lines cut through each pixel. The current method which calculates ANG is given by

```
ANG = ANGFACTOR*(LX*LX + LY*LY)/(3*CMAG(1)*CMAG(1)*NX)    (7)
```

Perhaps the best method to compute ANG would be one which delivers a constant number of scan lines, at least 1024 and perhaps limited to around 1500. This is accomplished by dividing the total scene angle by the desired number of scan lines. This technique will also yield a signed angle which is required.

9. PROJ(4)

PROJ is a flag which identifies if a corner projects to the ground or the horizon. PROJ is determined by the sign of COR(*,3). If COR(*,3) is greater than zero, then the corresponding corner is above the eye, and PROJ(*) equals zero. If the corner does project to the ground then PROJ(*) equals one.

10. CW(4,2)

CW is the NADIR CENTRIC location of the intercepts of the line from the eye through the corner onto the ground plane (i.e., Z = 0). If the corner does not project to the

ground, (i.e., PROJ(*) = 0), then CW(*,*) has a length equal to the horizon limit, HORIXONLIM. If the corner does project to the ground, (i.e., PROJ(*) =1), then CW(*,*) is computed from the method of similar triangles).

11. AA(4,2)

AA identifies the bounding lines of the footprint of Figures 1 or 3 and is computed by subtracting X and Y components of adjacent corners, CW. AA is used to compute the initial lower bound point, for visibility purposes, for each scan line.

12. ADDITIONAL VARIABLES

The remaining variables or constants initialized in PGSCENE are described in Table 4.

TABLE 4. REAL SCAN VARIABLES INITIALIZED IN PGSCENE

NAME	DESCRIPTION	EQUATION
ICRASH	The scaled distance from the Eye to the Screen	IVS*ISCALE/IDIV
NPL	The number of pixels per unit length	IFILDIM - 1
NX	Pixels along the X axis	NPL*LX
NY	Pixels along the Y axis	NPL*LY
ICONSTX	a constant used in PGPROJ	NPL*VS
JCONSTX	A constant used in PGPROJ	NPL*US
JCONSTY	A constant used in PGPROF	NPS*WS
NPIX	The value of the end pixel test in PGPROJ	NX + 1
IAXIS	The axis index for the end pixel test in PGPROJ	1
HORIZONLIM	The distance from the Eye to the Horizon	45000
IFIRST	A flag which identifies the first scan line	1
IFINISHED	A flag which identifies that the last scan line is completed	0
SUMANG	The angle accumulator for the scan lines	0
KSL	A corner counter	1
IXSO	The last X Screen Coordinate	0
IYSO	The last Y Screen Coordinate	0
KCOUNT	The counter for the number of times a pixel is addressed on one scan line	1
IPOWER	The value of the power of two that the pixel count is compared to	1

PGSL.FOR

PGSL.FOR is a subroutine, called once per scan line, whose purpose is to calculate all scan line parameters. Each scan line passes through the nadir, the lower bound (LOW), and the horizon (IXH,IYH). However, the horizon, in NADIR CENTRIC coordinates, completely defines the scan line, (i.e., as a vector from the nadir to the horizon). Figure 10 depicts the scan lines.

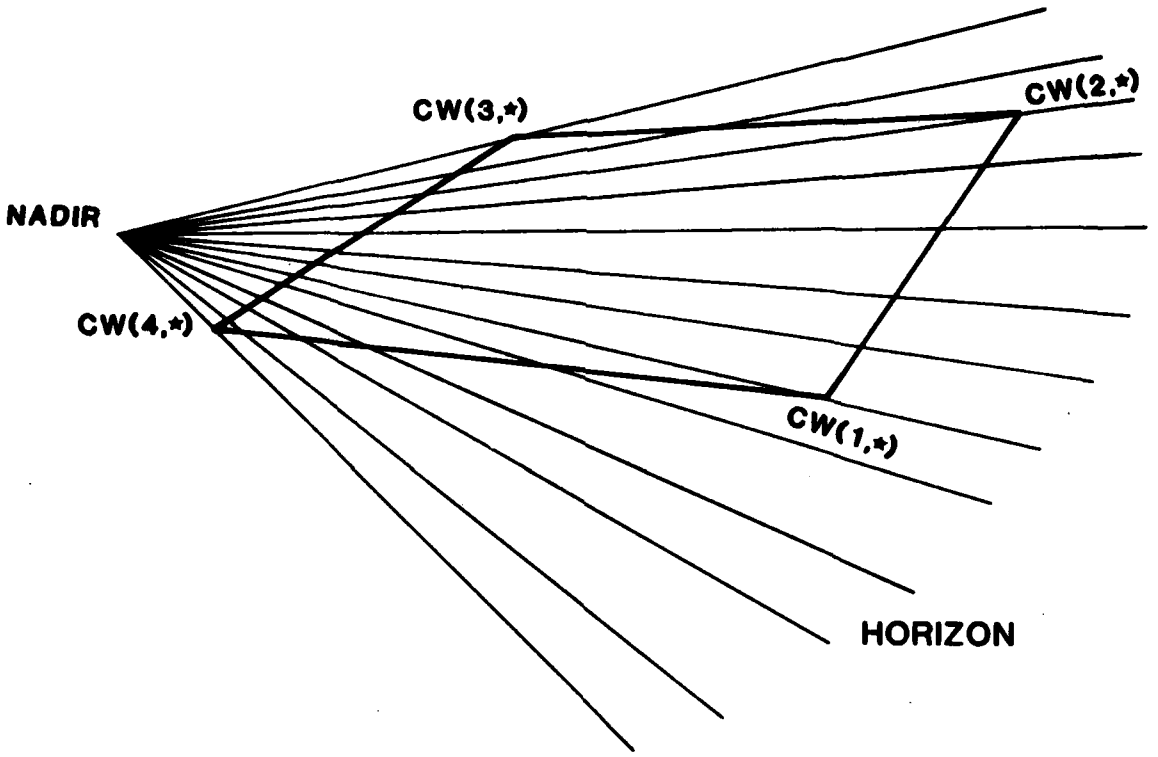


Figure 10. Scan Lines.

The present version of PGSL.FOR requires all scan lines to cut through the line between corners 3 and 4, (i.e., CW(3,*), CW(4,*) of Figure 10). However, Appendix E describes an improved version not subject to this restriction. The output of this routine is the horizon and the lower bound for each scan line, which are defined as follows:

1. IXH,IYH

   The first scan line extends from the nadir to corner 4, or CW(4,*). Thus, the horizon for the first scan line (IXH, IYH) is computed by the method of similar triangles

$$SCALE = HORIZONLIM/CMAG(4) \quad (8)$$

$$IXH = SCALE*COR(4,1) \quad (9)$$

$$IYH = SCALE*COR(4,2) \quad (10)$$

In general, we can compute the horizon for a new scan line by rotating the previous scan line thorugh the scan line angle, ANG,. Equation (11) illustrates this rotation

$$\begin{bmatrix} IXH \\ IYH \end{bmatrix} = \begin{bmatrix} COS(ANG) & SIN(ANG) \\ -SIN(ANG) & COS(ANG) \end{bmatrix} \begin{bmatrix} IXH \\ IYH \end{bmatrix} \quad (11)$$

Since ANG is small, the rotation matrix is approximated by equations (12) and (13). This eliminates the need to compute the SIN and COS functions

$$COS(ANG) \sim 1 - ANG*ANG/2 \quad (12)$$

$$SIN(ANG) \sim ANG \quad (13)$$

LOW(2)

The lower bound, LOW, is the point on the scan line which defines the angle for the first visible point. If the nadir is inside the footprint then LOW is the nadir, otherwise LOW will be the first intercept of the scan line with the footprint. This first intercept has been limited to the line between corners 3 and 4 in Figure 10, defined by AA(4,*). The equation for computing the lower bound is developed in Appendix J, and is

$$LOW(1)' = LOW(1) + A*T \quad (14)$$

$$LOW(2)' = LOW(2) + B*T \quad (15)$$

LOW(1) and LOW(2) are the last computed lower bound and LOW(1)' and LOW(2)' are the new lower bound. A and B is the signed distance along the X and Y axes between corners 3 and 4. T determines the change in lower bound points based upon the footprint and the scan line. T is dimensionless.

The subroutine PGSL determines which wedge of the scene is being computed by means of SUMANG. The angle accumulator, SUMANG, is tested against the angle to the next corner, CANG(*), to determine when a wedge has been completed. If the wedge has been completed, several wedge constants must be set to new values. These constants are:

IAXIS The screen axis to test against

NPIX The pixel along the IASIX, which identifies the end of the scan line

KSL The corner counter

Figure 11 identifies three wedges and the values of the corresponding wedge constants.

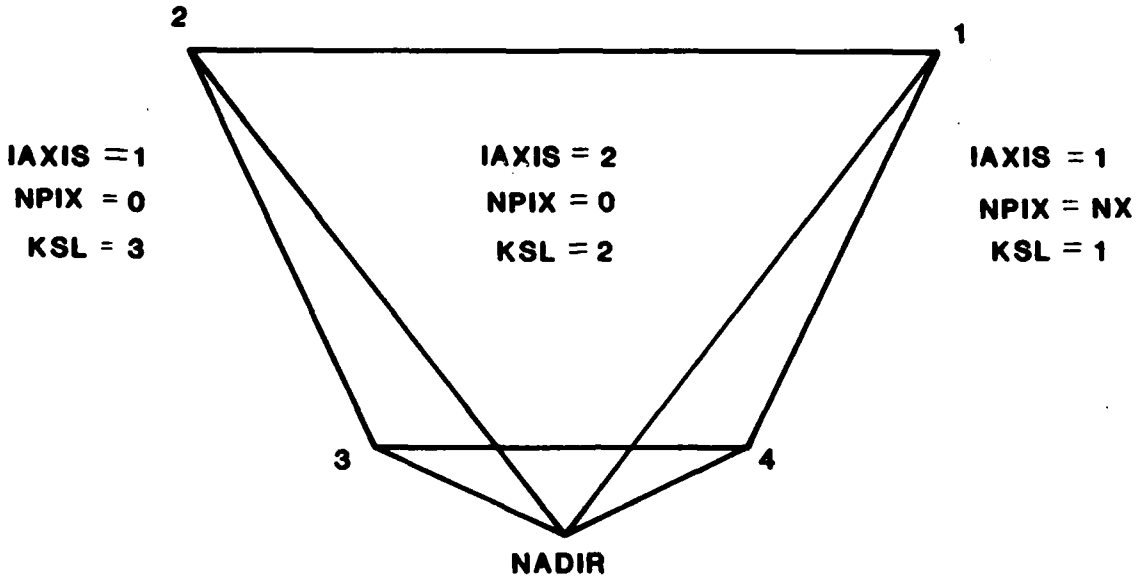


Figure 11. Wedge Constants.

PGVIS.FOR

The purpose of this subroutine is to initialize the Bresenham line drawing algorithm in PGSCAN.FOR. The Bresenham algorithm is described in the PGSCAN.FOR subsection of this section. The variables of Table 5 are initialized in PGVIS.FOR.

TABLE 5. PGVIS.FOR INITIALIZATION VARIABLES

NAME	DESCRIPTION
ICASE	The value of the fastest changing index
IOTHR	The value of the slowest changing index
IXH	The NADIR CENTRIC WORLD X coordinate for the horizon
IYH	The NADIR CENTRIC WORLD Y coordinate for the horizon
IF(2)	Equivalenced to IXH,IYH
NCR	The relative distance between data points on the ICASE axis
ICL	The database class which defines the resolution
JXY2	The distance to the horizon along the IOTHR axis
IXY2	The distance to the horizon along the ICASE axis
IDXY	One half of IXY2

INC	The direction of the ICASE axis, (i.e., ±1)
INCIO	The direction of the IOTHR axis, (i.e., ±1)
IP(3)	The world point in NADIR CENTRIC WORLD coordinates IP is equivalenced to IXP,IYP,IZP
IPP(2)	The point in WORLD coordinates (i.e., IPP = IP + IEYE)
IEEE	The Bresenham error accumulation term
NPN(2)	An index which is used to determine database hierarchies
IQ(2)	IPP divided by 1024*NCR (for hierarchy crossing logic)

PGVIS.FOR computes the following scan line initializations:

1. ICASE, and IOTHR

The ICASE axis is the fast moving axis index while IOTHR is the slow moving axis index. Figure 12 depicts the ICASW and IOTHR axes. ICASW and IOTHR are determined as shown by the following FORTRAN code.

```
IF (ABS(IXH) .GT. ABS(IYH)) THEN
          ICASE = 1
          IOTHR = 2
ELSE
          ICASE = 2
          IOTHR = 1
ENDIF
```

2.
```
ICL = 1
NCR = 1
```

3.
```
INC = ISIGN(1,IF(ICASE)     !(.ie ±1)
INCIO = ISIGN(1,IF(IOTHR)   !(.ie ±1)
```

4.
```
IXY2 = ABS(IF(ICASE))
JXY2 = ABS(IF(IOTHR))
IDXY = IXY2/2
IEEE = JXY2 - IDXY
```

5.
```
IP(*) = 0
IPP(*) = IEYE(*)
NPN(*) = IPP(*)/1024
IQ(*) = IPP(*)/1024
```

PGSCAN.FOR

This routine performs five major functions for each new point along a scan line:

1. Compute the point along the scan line by the Bresenham line drawing algorithm.

2. Perform a call to the database subroutine to obtain the point's elevation.

3. Determine if the point is visible.

4. Determine if the point's range exceeds a hierarchy level boundary.

5. Determine if the point's range exceeds the horizon limit.

The algorithm used for line drawing is very similar to the algorithm developed by Bresenham (12). However, it is designed so that each iteration changes the ICASE axis by INC*NCR, the proper integer increment in each hierarchy. The other coordinate may or may not change depending on the error term, IEEE, maintained by the algorithm. This error term contains a scaled integer representation of the distance between the exact path of the line and the closest integer coordinate generated.

The error term is initialized to represent -1/2. Each iteration adds to the error term until it is greater than zero, which identifies a need to increment the IOTHR coordinate by INCIO*NCR, and decrement the error term. The following FORTRAN code implements this version of Bresenham's algorithm.

```
     IEEE = JXY2 - IDXY
10   IF (IEEE .GT. 0) THEN
          IP(IOTHR) = IP(IOTHR) + INCIO*NCR
          IEEE      = IEEE - IXY2
     ENDIF
     IP(ICASE) = IP(ICASE) + INC*NCR
     IEEE = IEEE + JXY2
     perform visibility test
     GO TO 10
```

Thus, the error of the points created by the Bresenham line is at most NCR/2 units. As the points on the ICASE axis are incremented, the error term (IEEE) becomes more positive. When IEEE is greater than zero the IOTHR axis is incremented. Figure 12 depicts a typical Bresenham line. This algorithm only involves adding and testing to create the line. No multiplication is needed.

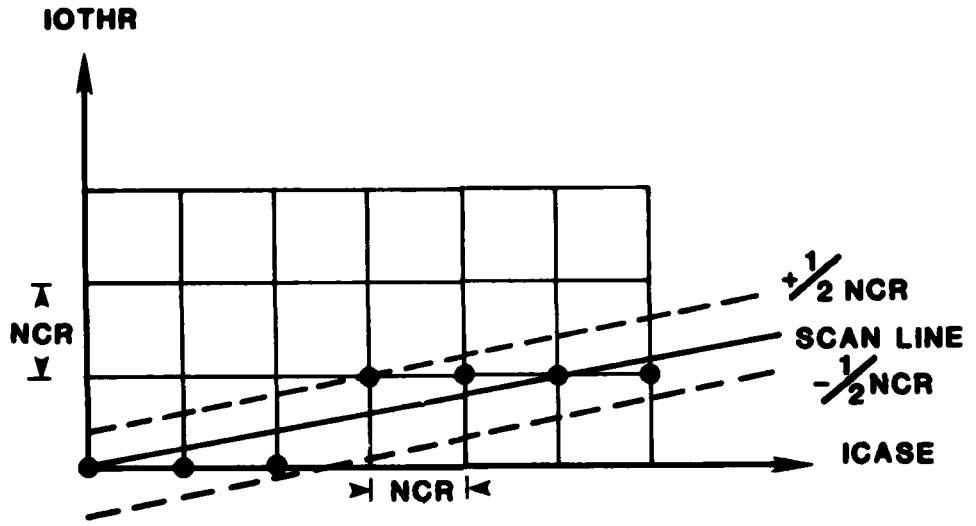


Figure 12. Bresenham Type Scan Line.

Visibility is a test which determines if the point is visible or not. Visibility is performed using the method of similar triangles. Figure 13 illustrates the procedure for the visibility test.

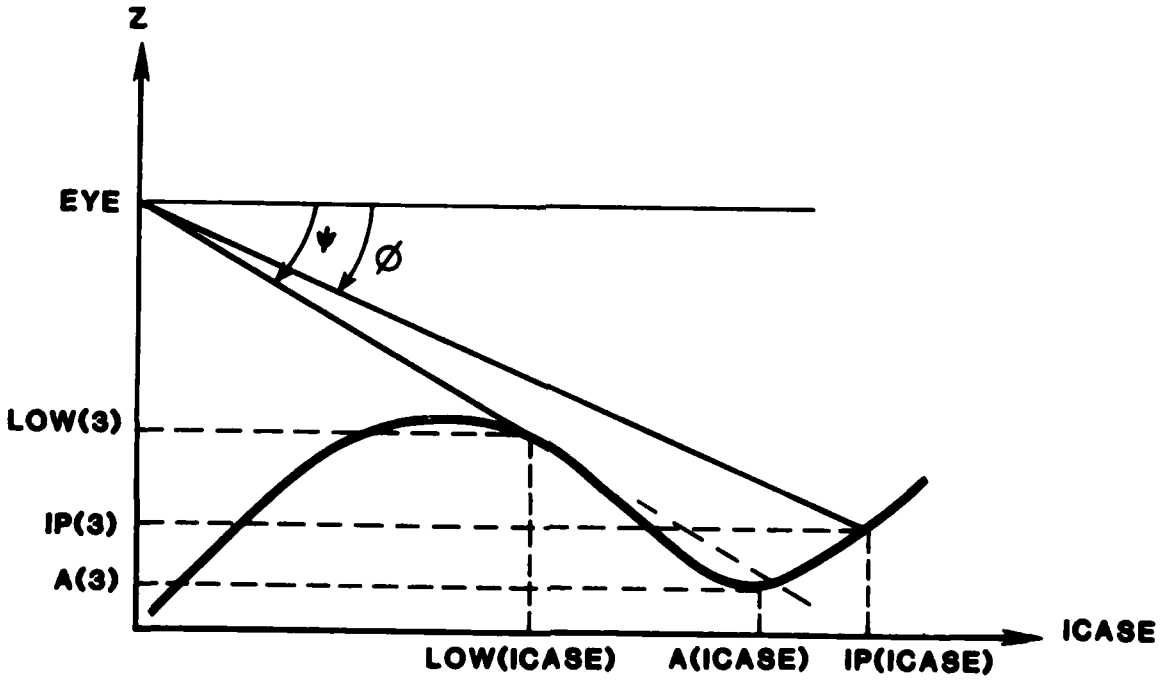


Figure 13. Visibility.

LOW is the initial lower bound computed in PGSL.FOR. The test point is IP. IP is visible if the angle form the horizontal to the line

from the eye to IP,ϕ, is less than the angle for the last visible point,ψ. Thus, the point A is not visible. The method of similar triangles implies IP is visible if

$$[IE(3) - LOW(3)]/LOW(ICASE) < [IE(3) - IP(3)]/IP(ICASE) \quad (16)$$

or if

$$[IE(3) - LOW(3)]*IP(ICASE) < [IE(3) - IP(3)]*LOW(ICASE) \quad (17)$$

If this occurs, then LOW is set to IP and IP is visible. Therefore, the visibility test is performed without any divides.

The data base hierarchy boundaries occur at fixed World Coordinates. Each hierarchy is identified by the class number, ICL. The distance between valid points in any class is NCR, where NCR = 2**(ICL-1). The width of each class has been arbitrarily set equal to 1024*NCR, which approximates the desired ground accuracy of 1/2 pixel resolution for a display screen having 512 pixels on each edge. (Appendices F and O discuss scan line improvements.)

The class assignments are made relative to the nadir. In other words the nadir is always in class 1. Two variables NPN and IQ are maintained to identify a class change. NPN and IQ are initialized in PGVIS.FOR to IEYE/1024. After the Bresenham type algorithm produces a point IPP, then IQ is set to IQ = IPP/(1024*NCR). The test for the hierarchy boundary crossing is given by the following **FORTRAN** code:

```
IF((IQ - NPN) .GT. 1) THEN
        "NEW CLASS"
        ICL = ICL + 1
        NCR = NCR*2
        NPN = NPN/2
ENDIF
```

When a class boundary has been crossed, the point may or may not be a valid point in the new class. Valid points are determined by

$$VALID\ POINT = NCR*1024 + NCR/2 \quad (18)$$

Thus, all class two points, in Eye coordinates, are odd. After the closest valid point to the scan line has been determined, the Bresenham error term must be modified to account for this new point (IP1,IP2). The Bresenham error term is reinitialized as depicted by the following **FORTRAN** code:

```
IEEE = -IDXY + (IP1*JXY2*INC - IP2*IXY2*INCIO)/NCR
IF(IEEE .GT. 0) THEN
        IP2 = IP2 = INCIO*NCR
        IEEE = IEEE - IXY2
ENDIF
```

The point IP is then set equal to (IP1,IP2).

If the horizon limit, HORIZONLIM, has been reached by the scan line before the end pixel on the screen has been reached, then sky is simulated. The sky corresponds to a wall with infinite height, located at the horizon, whose reflectance is an exponential function from light to dark.

PGPROJ.FOR

The main function of this routine is to compute the screen coordinates of a visible world point (IXP,IYP,IZY). This routine also determines when the scan line is finished. The mathematics for this routine are developed in Appendix K. Appendix H describes an improved projection algorithm. Generally, a projection of every point in the scene requires two floating point divides, one for each screen axis. But since data flows in a regular fashion, along a scan line, it has been possible to develop a projection algorithm which only requires two integer divides per scan line. Thus, eliminating approximately one thousand divides for the visible points that are processed on a typical scan line, at the expense of additional multiplies, adds, and compares for each visible point.

These two divides are required to initialize the projection algorithm, by computing the closest integer location for the first visible point and setting up the projection error terms. These error terms represent the fractional portion of the exact screen location for the visible point. Each successive point is represented by its relative distance from the previous visible point. This relative distance is then used to update the error terms, which are then tested for a pixel boundary crossing. In order for this routine to perform properly, adjacent visible points must map to the same or an adjacent pixel. Hence, the length NCR must map to less than one pixel side's length. There are two parts to the projection algorithm, initialization and the iterative incremental part.

The initialization section is called once per scan line, for the first visible point. The function of the initialization portion is to:

1. Compute the screen location of the first visible point.

2. Initialize the error terms JERRU and JERRW which represent the fractional portion of the screen coordinates.

3. Compute the scan line direction on both screen axes, (INCXS, INCYS).

The general procedure to compute the screen coordinates of a visible point is to find the location of that point in the EYE coordinate system by multiplying the EYE CENTRIC WORLD point by the tran-

pose of ROT. Then to project that point onto the screen by the method of similar triangles.

Since the visible point, (IXP,IYP,IZP), is given in NADIR CENTRIC WORLD coordinates, one must subtract the eye's height from the height of the visible point and matrix multiply the result by IROT to find the point in EYE coordinates, (IUP,IVP,IWP). The equations used to transform a point in WORLD coordinates to a point in EYE coordinates are

$$IZPT = IZP - IEYE(3) \tag{19}$$

$$\begin{bmatrix} IUP \\ IVP \\ IWP \end{bmatrix} = [IROT] \begin{bmatrix} IXP \\ IYP \\ IZPT \end{bmatrix} \tag{20}$$

Figure 14 depicts how one projects the visible point, (IUP,IVP, IWP), onto the screen when the screen center is at (0,IVS,0).

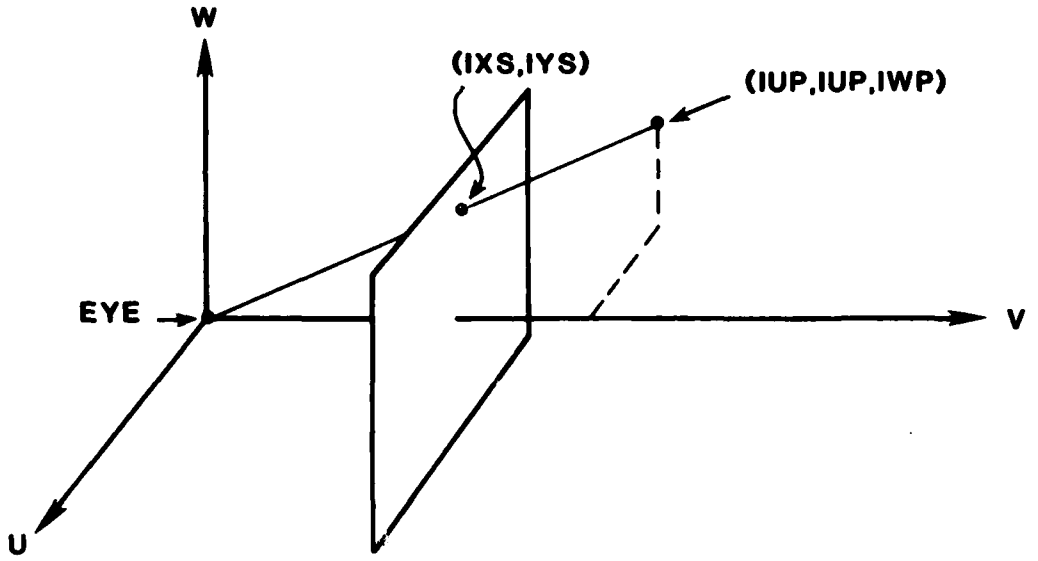


Figure 14. Projection Triangles.

From similar triangles, the screen coordinates (IXS,IYS) are given by

$$IXS = (IVS*IUP/IVP)*(NX/LX) \tag{21}$$

$$IYS = -(IVS*IWP/IVP)*(NY/LY). \tag{22}$$

The term (NX/LX) is a scale factor corresponding to pixels per unit screen length. If the screen center is at an arbitrary location (IUS,IVS,IWS) then the screen coordinates are

$$IXS = (IVS*IUP/IVP)*(NX/LX) - IUS*(NX/LX) \quad (23)$$

$$IYS = -(IVS*IWP/IVP)*(NY/LY) - IWS*(NY/LY). \quad (24)$$

Since the pixels are square, one can define the number of pixels per unit length, NPS, as

$$NPS = NX/LX = NY/LY. \quad (25)$$

Then the screen coordinates are

$$IXS = [IVS*IUP/IVP - IUS]*NPL \quad (26)$$

$$IYS = [-IVS*IWP/IVP - IWS]*NPL \quad (27)$$

If we define

$$ICONST = IVS*NPL \quad (28)$$

$$JCONSTX = IUS*NPL \quad (29)$$

$$JCONSTY = IWS*NPL \quad (30)$$

then

$$IXS = ICONST*IUP/IVP - JCONSTX \quad (31)$$

$$IYS = -ICONST*IWP/IVP - JCONSTY. \quad (32)$$

Since the screen coordinates (IXS,IYS) are integers, two error terms are created to manage the fractional portion of the screen coordinates. Each pixel is defined at the center by its screen coordinate (IXS,IYS). Therefore, the pixel boundary occurs at (IXS ± 1/2, IYS ± 1/2). Figure 15 depicts three pixel boundaries.

Therefore, based on Figure 15 the pixel boundary crossing occurs at 1/2 or -1/2 of a pixel width. Since the exact screen coordinate XS is

$$XS = IXS + FRACX = ICONST*IUP/IVP - JCONSTX \quad (33)$$

then

$$FRACX*IVP = ICONST*IUP - (JCONSTX + IXS)*IVP. \quad (34)$$

Therefore, the error term is

$$JERRU = FRACX*IVP \quad (35)$$

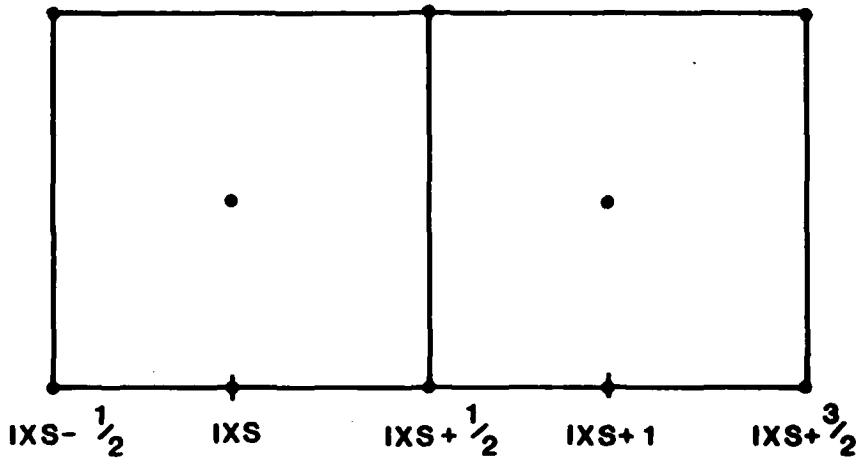


Figure 15. Three Pixel Boundaries Along the IXS Axis.

A comparison of JERRU to IVP/2 or -IVP/2 is equivalent to comparing FRACX to 1/2 or -1/2, which defines a pixel boundary. An efficient method to compute JERRU is

$$JERRU = ICONST*IUP - JIX*IVP \tag{36}$$

JIX is defined by

$$JIX = JCONSTX + IXS \tag{37}$$

Once the error terms have been initialized, they are compared to IVP/2 or -IVP/2. This comparison will determine if the first point actually belongs in an adjacent pixel, due to truncation. The following FORTRAN code depicts how JERRU and IXS are updated:

```
IF ((2*JERRU.GE.IVP).OR.(2*JERRU.LT.-IVP))THEN
        INC = ISIGN(1,JERRU)
        IXBS  = IXBS  + INC
        JIX   = JIX   + INC
        JERRU = JERRU - IVP*INC
ENDIF
```

A similar error term JERRW for IYS is

$$JERRW = -ICONST*IWP - JIY*IVP \tag{38}$$

JIY is defined by

$$JIY = IYS - JCONSTY \tag{39}$$

These two error terms have now been initialized for the next visible point.

One can identify the direction in which the scan line moves across the screen (INCXS,INCYS) as [-1, 0, +1]. The value of (INCXS, INCYS) will determine the pixel boundary crossing test which is to be applied to the error terms. Appendix K develops the mathematics for determining (INCXS,INCYS). Appendix G develops pixel projection to fractional pixel accuracy.

After the projection initialization is complete, the projection algorithm is ready to receive successive visible points along the scan line. Each new point is represented by its incremental distance from the last visible point (IXPINC,IYPINC,IZPINC). To compute the screen coordinates for the new point one must update the two error terms (JERRU,JERRW) to compensate for this incremental change, and test the error terms for a pixel boundary crossing. Thus, the projection of a new point does not require any divides, but does require 2 multiplies, and 2 adds to update each error term, and 2 compares to determine if a pixel boundary is crossed. If the pixel boundary is crossed then 1 add and 2 increments are additionally required.

If the new point (IXP′,IYP′,IZP′) is represented by its relative distance from the last visible point (IXP,IYP,IZP), then the relative distance is

$$\begin{bmatrix} IXPINC \\ IYPINC \\ IZPINC \end{bmatrix} = \begin{bmatrix} IXP' - IXP \\ IYP' - IYP \\ IZP' - IZP \end{bmatrix} \tag{40}$$

These incremental distances can then be rotated into the EYE coordinate system to yield (IUPINC,IVPINC,IWPINC), by the matrix multiplication of IROT

$$\begin{bmatrix} IUPINC \\ IVPINC \\ IWPINC \end{bmatrix} = [IROT] \begin{bmatrix} IXPINC \\ IYPINC \\ IZPINC \end{bmatrix} \tag{41}$$

Once these incremental distances are transformed into the EYE coordinate system, they are accumulated into the error terms. These error terms are then tested for pixel boundary crossing. The accumulation is

$$\begin{bmatrix} IUP' \\ IVP' \\ IWP' \end{bmatrix} = \begin{bmatrix} IUP + IUPINC \\ IVP + IVPINC \\ IWP + IWPINC \end{bmatrix} \tag{42}$$

where (IUP′,IVP′,IWP′) is the new point in the EYE coordinate system. Substituting Equation (42) into the old error term, Equation (36), yields the new error term

$$JERRU' = ICONST*(IUP + IUPINC) - JIX*(IVP + IVPINC) \quad (43)$$

or

$$JERRU' = [ICONST*IUP - JIX*IVP] + [ICONST*IUPINC - JIX*IVPINC] \quad (44)$$

or

$$JERRU' = JERRU + ICONST*IUPINC - JIX*IVPINC \quad (45)$$

A similar equation for JERRW is

$$JERRW' = JERRW - ICONST*IWPINC - JIY*IVPINC \quad (46)$$

Once these error terms have been updated for the new point, they are tested for pixel boundary crossing. The pixel boundary crossing test for the new point is similar to that of the first point except that INCXS and INCYS define the test to be applied. The following FORTRAN code is the pixel boundary crossing test for the XS axis:

```
IVP = IVP + IVPINC

IF (INCXS .EQ. 1) THEN
        ITEST = IVP -JERRU -JERRU
        IF (ITEST .LE. 0) THEN
                JERRU = JERRU -IVP
                JIX = JIX + 1
                IXBS = IXBS + 1
        ENDIF

ELSE IF (INCXS .EQ. -1) THEN
        ITEST = -IVP - JERRU -JERRU
        IF (ITEST .GT. 0) THEN
                JERRU = JERRU + IVP
                JIX = JIX -1
                IXBS = IXBS -1
        ENDIF
ENDIF
```

The projection algorithm also tests each screen coordinate produced to determine if the end of a scan line has been reached. This test simply compares the screen coordinate ISBS of IYBS, depending on the IAXIS value, against the end pixel value, NPIX. NPIX is set to 0, NX, or NY as defined in Figure 11. The following FORTRAN code is the end pixel test:

```
EQUIVALENCE (IXBS,ISCRN(1)),(IYBS,ISCRN(2))
IF (NPIX .EQ. 0) THEN
        IF (ISCRN(IAXIS) .LE. NPIX) RETURN1
ELSE
```

```
            IF (ISCRN(IAXIS) .GE. NPIX) RETURN1
    ENDIF
```

RETURN 1 is a special return to signify that the scan line is finished.

When the scan line crosses into a new database hierarchy, the ground resolution is reduced by a factor of 2 raised to the power of the class change. The class change is not always one since the next visible point may be far from the last visible point. The following FORTRAN code identifies the database hierarchy crossing procedure for the projection algorithm:

```
IF (IRESOL .NE. IOLDRESOL) THEN
        NEWRF = 2**(IRESOL - IOLDRESOL)
        JERRU = JERRU/NEWRF
        JERRW = JERRW/NEWRF
        IV = (IV + NEWRF/2)NEWRF
        IOLDRESOL = IRESOL
ENDIF
```

PGLOGLOAD.FOR

This routine stores the scene data in ITEMBUF. Each time a visible point is projected into the screen, the reflectance data is accumulated into ITEMBUF and a count buffer, ICNT, is incremented. Thus, the scene data is averaged by dividing the accumulated pixel data by its corresponding count.

To prevent overflow of the byte buffer, ICNT this routine only accumulates pixel data if the count is a power of two. Only data points whose count is 1,2,4,8, ... will be accumulated into the screen buffer. This function is called "log filtering."

If the count for the projected point is of a power of two, then this routine calls the database to determine the reflectance value for the visible point. Thus, saving the reflectance computation for points that are discarded.

PGFILTER.FOR

Averaging of the display's data, ITEMBUF, is accomplished by the subroutine PGFILTER. PGFILTER is called once at the end of picture creation. The present version of PGFILTER divides the "log filtered" accumulated pixel reflectance by the number of accumulated data points projected into that pixel. This yields an average reflectance value. If a pixel has no data then it is termed a missed pixel, and no average is performed. The average is computed as shown in the following FORTRAN code.

```
      DO 100 I = 1,IFILDIM
      DO 100 J = 1,IFILDIM
      KDIV = ICNT (I,J)
      IF (KDIV.EQ.0) KDIV = 1
      ITEMBUF(I,J) = ITEMBUF(I,J)/KDIV
100   CONTINUE
```

There is a special entry point called FILT2, within the subroutine PGFILTER, whose purpose is to fill any missed pixels (i.e., pixels for which ICNT = 0). The fill is computed as the average of the nonzero adjacent pixels. This is accomplished by summing the reflectance from the adjacent nonzero pixels and dividing the sum by the number of nonzero adjacent pixels.

This filter is useful to fill the inherent missed pixels that are a result of creating test pictures, which use a large angle factor. The use of the large angle factor is discussed in the subroutine INPUT section of this paper.

Another PGFILTER special entry point is CLEAR that clears ITEMBUF to zero and is called once at the start of the scene computation.

PGDATA.FOR

PGDATA.FOR is the data base used to create the first movie. This data base consists of block type buildings, holes and roads. The building tops are painted with a psuedo noise to demonstrate **REAL SCAN**'s capability for displaying high detail. The data base is a mathematical model which returns a single height and reflectance value for any given set of ground coordinates.

The data base coordinates are computed by taking the modulo (1024) of the ground coordinates. Modulo arithmetic is used so that only a portion of the environment is defined by the data base. Thus the data base repeats its pattern every 1024 units.

Second Generation Software

This section of the report will document the major differences between the second generation **REAL SCAN** software routines (PZ*.FOR) and the first generation **REAL SCAN** software routines (PG*.FOR). These routines are listed in Table 6. A complete **FORTRAN** listing of these routines is given in Appendices FB through FM. Appendix L compares the two sets of routines for picture creation capability and the CPU time required to compute an identical scene with both sets of routines.

TABLE 6. SECOND GENERATION REAL SCAN SOFTWARE ROUTINES

1.	PZCOMDAT.FOR	
2.	PZBUF.FOR	
3.	PZNOISE.FOR	
4.	PZMAIN.FOR	INTEGER
5.	PZSCENE.FOR	INTEGER
6.	PZSLINIT.FOR	INTEGER AND FLOATING POINT
7.	PZSCAN.FOR	INTEGER
8.	PZPROJ.FOR	INTEGER
9.	PZMULT.FOR	INTEGER "NO OVERFLOW"
10.	PZLOGLOAD.FOR	INTEGER
11.	PZFILTER.FOR	INTEGER
12.	PZDATA.FOR	INTEGER

The routines of Table 6 differ from the first generation REAL SCAN software routines (PG*.FOR) in the following major areas:

1. They have color capability.
2. The programs are organized in a more functional manner.
3. The variables IP(*) and LOWN(*) which represent the data points along the scan line are scaled to a specific number of bits.
4. Variables are in EYE CENTRIC WORLD coordinates rather than NADIR CENTRIC WORLD coordinates.
5. The projection algorithm utilizes a "NO OVERFLOW" multiply routine to compute the projection error terms.
6. The scan lines don't always start at the Nadir.

This section of the report will discuss the differences listed above.

COLOR CAPABILITY

Color capability is achieved by creating data bases which return a vector reflectance. The reflectance has three components: blue, red, and green. The dimension of the screen buffer is changed from the black and white size of (512,512) or 0.5 megabytes to (3,512,512) or 1.5 megabytes. This larger size allows all three reflectance values to be accumulated into each pixel of the screen buffer.

PROGRAM ORGANIZATION

The black and white routines of Table 1 (PG*.FOR) were reorganized in the following files:

PZSCENE.FOR consists of all one-time scene operations or initializations. The file PZSCENE.FOR is a combination of the subroutines PGSCENE.FOR, INPUT, and OUTPUT.

The two scan line initialization routines PGSL.FOR and PGVIS.FOR in the first generation software were combined into one routine PZSLINIT.FOR. Thus, PZSLINIT.FOR performs all one time scan line initializations.

SCALED DATA POINTS

The purpose of the scaled data points is to limit or bound the number of bits required to maintain the data point's resolution along the scan line. This will also bound the number of bits required to manipulate these data points. The data points IP(*) and LOWN(*) are scaled by the hierarchy levels.

The data points (i.e., world coordinate) along the scan line grow in magnitude as the scan line extends from the nadir to the horizon. But since the data base consists of hierarchy levels, the data points along the scan line are scaled by the hirarchy level. Thus, the absolute location of the data point in NADIR CENTRIC coordinates is a function of the data point's coordinate value and the hierarchy level in which the data point resides. For example, if the data point is in a hierarchy level where the relative increment between data points, NCR, is 2 and one coordinate of the data point is 1000, then that absolute data point coordinate in NADIR CENTRIC WORLD coordinates is 2 times 1000, or 2000.

The idea of scaled coordinates is achieved by right shifting the coordinate each time a hierarchy level is crossed. Since each hierarchy level has twice the relative increment between coordinates as the previous hierarcial level. The hierarchy levels are separated by NCRLIM*NCR units. NCRLIM is a constant corresponding to twice the display resolutions, usually 1024. NCR is the relative increment between data points in a particular hierarchy level. Thus, the largest the scaled data point coordinate can be is NCRLIM units. Therefore, the number of bits ,NBITS, required to maintain the data point's accuracy is the constant

$$NBITS = LOG_2 (NCRLIM) + 2 \qquad (47)$$

The extra bits are accommodate sign and round off.

Scaled data points are of benefit to the projection algorithm by bounding the range of the projection error terms, JERRU and JERRW. The projection error terms are bounded since the scaled relative increment between data points is a constant (usually -1, 0, or +1) for any hierarchy level. Thus, each time a new hierarchy level is crossed the projection error terms are right shifted to half their prevous significance. The set of FORTRAN equations which define the hierarchy boundary crossing are listed below:

```
      IF(NCRLIM .LE. ABS(IP(ICASE)) THEN !NEW HIERARCHY
          DO 10 I = 1,3
          IP(I) = IP(I)/2
10        LOWN(I) = LOWN(I)/2
          JERRU = JERRU/2
          JERRW = JERRW/2
          IVP = IVP/2
      ENDIF
```

EYE CENTRIC WORLD COORDINATES

All variables which represent altitude are in the EYE CENTRIC WORLD coordinate system, (i.e., the altitude of the point is relative to the eye's altitude). This representation of altitude is more efficient than using the ground plane for a reference, since the visibility and projection algorithms both use the altitude of the point relative to the eye. Therefore, when the data base returns an altitude the eye's altitude is subtracted from the points altitude. Then the EYE CENTRIC altitude is scaled by the relative increment in the hierarchy level by dividing by NCR (i.e., scaling to the hierarchy level without actual division).

"NO OVERFLOW" MULTIPLY

One problem with the projection algorithm of PGPROJ.FOR was that the computation of the projection error terms (JERRU and JERRW) caused integer overflows on the VAX-11/780 at NTEC. The new projection routine, PZPROJ.FOR, utilizes a "NO OVERFLOW" multiply routine to compute the projection error terms, and yields accurate results. Thus, the software has been brought a step closer to the anticipated hardware solution.

From equations (37) and (38) PGPROJ.FOR computes the projection error terms as:

```
JERRU = ICONSTI*IUP - JIX*IVP                    (48)

JERRW = ICONST*IWP - JIY*IVP                     (49)
```

The error terms are computed as the sum of two products. If either of these two products overflows then the computation process for the error term overflows and causes a halt in the execution of the scene.

The VAX-11/780 system allocates 4 bytes to integers. A close analysis of equation (48) shows that JERRU can overflow the 4 bytes allocated. This analysis is as follows:

1. ICONST, corresponding to the number of display pixels, is typically 512 units or 9 bits.

2. IUP is the product of IROT and the data point's range, IP.

3. IROT is the rotation matrix scaled by 2 raised to the N power, where N is typically 12, corresponding to quarter pixel resolution.

4. Thus using standard FORTRAN execution processes, JERRU will overflow the 32 bit range if the point's range is greater than 11 bits (32 - 9 - 12).

5. Thus if a scaled coordinate exceeds 1024 or -1024 JERRU can overflow.

This problem is eliminated in the PZPROJ.FOR routine by scaling coordintes and by using a "NO OVERFLOW" multiply routine. A "NO OVERFLOW" routine is a routine which is compiled with the NON STANDARD FORTRAN QUALIFIER NO OVERFLOW. This qualifier suppresses the overflow detection process on the VAX-11/780 system. Thus the PZPROJ.FOR routine allows the products to overflow but accurate results are guaranteed. The result is accurate since, the value of JERRU is always smaller than IVP. This is guaranteed because successive visible points on the scan line are required to map to the same or adjacent pixels on the screen as the last visible point.

Thus, since IVP does not exceed the 4 byte range, then JERRU will not exceed the 4 byte range. This implies that the high order bits of the two products used to compute JERRU cancel in the summation. Therefore, a "NO OVERFLOW" multiply routine which sums two large products to yield a small result is guaranteed to yield accurate results for the projection routine.

Another significant item is that since the data point's coordinates along the scan line are scaled by the hierarchy levels, then IVP is bounded by a predetermined number of bits. The number of bits required to maintain IVP is the sum of the number of bits required for IP and N or 24 bits. Thus the number of bits required to maintain the projection error terms (JERRU and JERRW) is only 24 bits, rather than 32 bits.

SCAN LINE START

Another significant difference in the second generation REAL SCAN software routines is that the scan lines don't have to start at the

nadir. If the maximum height of the data base is known then the lower bound angle for a particular scan line is used to determine a safe starting point for the scan line. Figure 16 illustrates this feature.

The value of IP(ICASE) is determined from the method of similar triangles as depicted by the following FORTRAN equation:

```
IP(ICASE) = LOWN(ICASE)*(EYEHEIGHT-MAXHEIGHT)/EYEHEIGHT
```

The value of the other coordinate IP(IOTHR) is found by multiplying IP(ICASE) by the slope of the scan line.

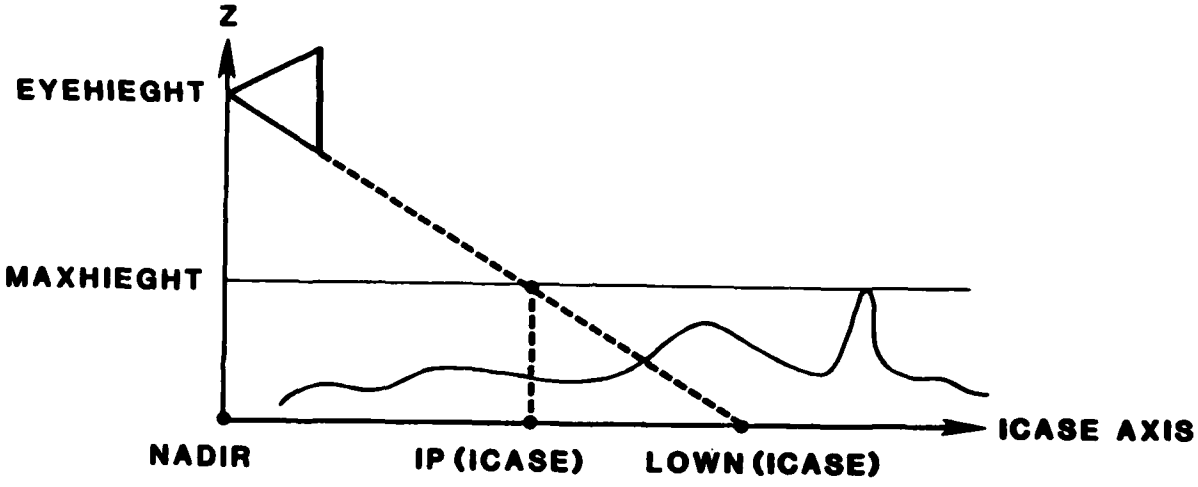


Figure 16. Scan Line Start Improvement.

SECTION III

THE DATA BASE PROBLEM

What does it mean to develop a simple method for modeling the world to display limited resolution? This question must be understood and answered before one can be guaranteed a solution to simply modeling the world. Some of the aspects of a simple real world model are:

1. Capability of modeling highly complex scenes in a simple way (i.e., simple computation and small amount of data).

2. Need to limit the model's detail to the display's resolution to eliminate aliasing.

3. Need for anti-aliasing procedures dealing with visible edge boundaries (i.e., looking over a sequence of distinctly colored hedge rows running from left to right across the viewing window, hence partially occulting more distant hedge rows).

4. Capability to easily match the Kell factor resolution (i.e., approximately 3 display pixels per scene line pair) (13).

5. Capability to easily incorporate the fact that resolvable detail along any line of sight subtends a constant spherical angle. Hence, the world model's detail should decrease with range from the eye.

The form of real world data and the form of known modeling methods impacts the data base problem. A model must be able to represent the following data types:

1. Man made ground cover such as

    a. buildings
    b. roads
    c. structures
    d. farm fields
    e. orchards
    f. landscaping
    g. rubble

---

[13]Kell, R. O.; Bedford, A. B.; and Trainer, M. A. "An Experimental Television System", Proceedings of IRE, Vol. 22, No. 11, p. 1247, November, 1934.

2. Natural ground cover such as

   a. desert
   b. forest
   c. swamp
   d. lake or sea
   e. stream
   f. mountain
   g. beach
   h. field

It seems most reasonable to expect different modeling techniques would be utilized to efficiently model these data types. For example, some form of polynomial seems appropriate to the undulation of terrain while painted planar surfaces seems appropriate to many buildings and structures.

The modeling methods impact two aspects of the data base problem: the form of the source data and the form of the modeled data used to calculate the scene. The forms may be obviously compatible (i.e., a terrain elevation map) or clearly require a mapping or translation (i.e., culture or texture descriptors describing various forms of ground cover, but without exact detail).

The **REAL SCAN** method of producing a realistic scene for training has as one of its goals, the capability of automatically transforming real world data into the computer math model used to calculate the display. For example, one might imagine photo-planimetry could be used to accurately describe a high detail scene, such that one would obtain detailed elevation and color data. However, one is also aware that few, if any, parts of the world are constant. Vegetation grows and changes color. The sun produces various shading and shadowing effects. Hence, the translation of real world data to any computer math model will probably make use of culture classifiers and intermediate modeling to fill in detail suggested by photographs.

It is important to consider the forms of the original data and the forms of the math models so that the **REAL SCAN** efforts can be placed in perspective. Real world source data may be in the form of:

1. Photographs, suggesting automatic planimetric conversion via intermediate models

2. Elevation maps on a regular grid with culture files describing the ground cover, suggesting conversion via intermediate models.

3. Some combination of 1 and 2 above. The clear need for modeling suggests representative terrain could be directly

generated from the intermediate models, and placed in the form of the real time computer generated math model. For example, trees could be modeled via Csuri procedures (14,15) giving full volumetric modeling. The appearance of tree leaves could also be "grown" on the surface of an ellipsoid which allowed both color and transparency so that full 3-D effects are present (16). Without the complexity of volumetric modeling, trees could also be grown on a plane surface which allowed color and transparency but the full 3-D effect would not be present.

These examples suggest the need to classify various modeling methods generally and correlate the modeling method with characteristics of the real world data. Current CIG methods make use of plane surfaces to model the training scene. Large planar surfaces achieve a substantial degree of data compression. However, complex scenes suggest that planar surfaces, with complex texture painted on the surfaces, would have their dimension reduced to approximately pixel size. If a surface dimension approaches four pixels, then it is clear that both less data and fewer computations would be required if the data were stored in ground coordinates. First of all, two variables need not be stored for a fixed world point since these correspond to the address of the data. Secondly, no sorting is required to access data. Finally, the same model could be used to interpolate over the surface, but the color data in real world coordinates need not be calculated except if the data is visible. These arguments against planar surfaces are not valid when the scene detail is sparse, but the breakpoint where minimal **REAL SCAN** detail requirements matches planar surface modeling has not been determined.

Some of the ways that one can model highly complex scenes where two coordinates correspond to data base address are:

1. Polynomial coefficients describing elevation or color surfaces.

---

[14]Csuri, C.; Hackathorn, R.; Parent, R.; Carlson, W. and Howard, M. "Towards an Interactive High Visual Complexity Animation System", SIGGRAPH '79 Proceedings, p. 289-299, August, 1979.

[15]Marshall, R.; Wilson, R.; and Carlson, W. "Procedure Models for Generating Three-Dimensional Terrain", SIGGRAPH '80 Conference Proceedings, pp. 254-259, July, 1980.

[16]Gardner, G. "Computer Generated Texturing to Model Real World Features", Proceedings of the 1st Interservice/Industry Training Equipment Conference, pp. 239-245, November, 1979.

2. Interpolation formula using the data base grid (i.e., Overhauser-Coons patch, etc.).

3. Function coefficients such as Fourier coefficients with a trigometric look up table.

4. Culture map dependent upon world coordinate and data type parameter (i.e., creation of look up tables for various data types having proper statistical properties such that a modulo index of the world address also corresponds to the look up table address).

5. Use of signed numbers in the color generation models to generate nearly arbitrary curves between regular grid points (i.e., a stream's meandering boundaries can be generated by forcing a parabolic or higher order function to interpolate a regular grid such that the function changes sign to indicate blue).

6. Feature maps such as the surfaces of buildings or structures, or feature patterns (i.e., color and transparency on an ellipse to represent a tree or bush).

These are modeling schemes which have been considered for **REAL SCAN**. However, only some of them have been investigated to any depth.

REAL SCAN Data Base Models

The modeling approaches which have been investigated to date are all based upon a regular grid of addresses to access the model. Further, only single value elevation models have been considered to date. The models are:

1. Functions to model terrain elevation, such as Z(X,Y)=A(sin W X)(sin WY). Reflectance information is obtained via an arbitrary sun vector and assuming the surface is a diffuse reflector.

2. Planar surfaces to model buildngs. Reflectance is assigned to the surface as a function of position, creating regular or arbitrary patterns.

3. Culture maps to simulate a range of terrain including forest, field, lake, and beach. The maps are interpolated for both elevation and color. Reflectance information is obtained via an arbitrary sun vector and assuming the terrain is a diffuse reflector.

The recognition of decreased resolution with range has been implemented by making 11 data base levels in our hierarchy. Hence, if

one is in hierarchy 1, nearest the eye, then ground coordinates change in units (i.e., 1,2,3,4,5, ... an increment of one). If the scene's data next comes from hierarchy 2, then ground coordinates would increment by two's. Hierarchy boundaries depend upon range from the eye to the ground coordinates. The change between hierarcy 1 and hierarchy 2 occurs approximately where the spherical angle subtended in hierarchy 2 matches the desired real world resolution (i.e., we are investigating the impact of about two ground points per linear pixel dimension or about four ground points per pixel.)

Details of the **REAL SCAN** data base models will be presented later in this section. At this point, a development based upon the sampling theorem and eye point motion is presented to generate estimates of potential data compression and memory size required to accommodate both the currently generated scene and a gaming area. Appendix B further develops memory requirements versus gaming area.

Data Base Modeling Estimates

Let us consider a frequency limited function such that the shortest period (corresponding to the highest frequency) is defined. The sampling theorem requires at least two points per shortest period to reconstruct (i.e., closely approximate) the original function. This may be interpreted to limit the extremums between sample points to at most, one (i.e., no more than one maximum or minimum between sample points, the "or" is significant). This suggests at least quadratic equations are needed to accurately model a function sampled on a regular grid having maximum grid spacing. But the theorem also suggests the converse, no more than one extremum can be interpolated for a sampled function.

Let us digress for a moment, to consider the number of Fourier coefficients required to model a space when the sampling corresponds to the grid spacing. We see the following terms sinx, sin2x, siny, sin2y, (sinx)(siny), (sinx)(sin2y), (sin2x)(siny), and (sin2x)(sin2y). This yields 8 trigometric terms compared to the 4 terms defining the original grid. Hence, courser spacing than specified by the sampling theorem can only be considered if the use of multiple coefficients substantially reduces the computation requirements, since a doubling of data base size is required for comparable accuracy or resolution.

Let us now return to considering an interpolated function. One can set a lower bound to the number of interpolated data points for a scene producing display limited resolution. Let n points be averaged per pixel linear dimension (i.e., **REAL SCAN** considers $n = 2$). The Kell factor suggest 3 pixels are required to generate a line pair. Hence, 3n data points would map to the highest resolvable frequency in the display. This yields 3n/2 interpolation points per smallest ground detail on a fixed grid data base. Hence, if the dimension of the smallest ground detail is A and the gaming dimension is R, then

the highest resolution hierarchy need have no more than $(2R/(3nA))^2$ data points. If n = 2, A = 1 cm and R = 1 Km then $(2R/(3nA))^2 = 1.1x10^9$ data points.

However, if one assumes that detail can be taken from culture memory maps (i.e., representing waves or trees or fields, etc.) and that an inexpensive memory map could describe about 10,000 points, then the amount of interpolation increases substantially. If we assume the culture memory map detail has been prefiltered to provide 3n/2 points per length for the highest frequency line pair, and if we assume a mapping of the 10,000 points onto an M*M size interpolation grid will not create a noticeable repeating pattern, unless the repeating pattern is desired, then the number of data points needed to model the highest resolution hierarchy is $(2MR/(300nA))^2$. If n = 2, A = 1 cm, R = 1 Km and M = 10 then $(2MR/(300nA))^2 = 1.1*10^7$ data points. The use of cultural mapping functions (i.e., memories) suggests the ability of achieving 0.3 cm resolution over high resolution gaming areas of 10 KM * 10 KM for about $1.2*10^{10}$ data points. The corresponding grid space would be 9 cm for M = 10, n = 2, A = 0.3 cm. While the grid spacing is large, and suggets a moderate amount of data compression, the 9 cm grid spacing requires some method, such as signed texture parameters, to realistically map arbitrary road, stream, or building boundaries.

This analysis suggests reasonable estimates of data compression will likely range between 3 and 30 interpolations per data grid dimension. The lower bound of 3 interpolations seems to be well founded via the Kell factor. The upper bound is clearly an estimate depending upon both the cultural mapping method and subjective appraisal of the data bases realism.

Let us assume a data grid spacing of 10 cm corresponding to high detail resolution of 0.33 cm with a culture mapped detail interpolation. Then the size of the data base can be estimated for various sized gaming areas. Let us further allow 16 bits of elevation, 24 bits of color and 8 bits of culture code. Table 7 illustrates the number of data points required to model each hierarchy level as a function of the area covered by the hierarchy level. All levels beyond the first, whether 10 or 20 or more, contribute only 1/3 of the first level's data points.

If we imagine a gaming region extending to the horizon (i.e., say about 80 KM) such that a 10KM x 10KM high resolution region forms the center, then the total data point requirement can be estimated as $1.34*10^{10}$ data points in 15 levels.

TABLE 7. 10cm GRID DATA POINTS

Hierarchy Level	10KM*10KM	20KM*20KM	40KM*40KM
1	10^{10}	$4 * 10^{10}$	1.6×10^{11}
2	2.5×10^{9}	10^{10}	$4 * 10^{10}$
3	$6.25 * 10^{8}$	2.5×10^{9}	10^{10}
4	$1.56 * 10^{8}$	$6.25 * 10^{8}$	2.5×10^{9}
5	3.9×10^{7}	$1.56 * 10^{8}$	$6.25 * 10^{8}$
6	$0.10 * 10^{8}$	3.9×10^{7}	$1.56 * 10^{8}$
7	$2.5 * 10^{6}$	10^{7}	3.9×10^{7}
8	6.24×10^{5}	$2.5 * 10^{6}$	10^{7}
9	1.56×10^{5}	6.24×10^{5}	$2.5 * 10^{6}$
10	$3.9 * 10^{4}$	1.56×10^{5}	6.24×10^{5}
SUM	$1.33 * 10^{10}$	$5.33 * 10^{10}$	2.13×10^{11}

Using 6 bytes per data points yields a data base of 8.4×10^{10} bytes to describe the gaming region. This data base is within the capability of video disk technology (5).

Appendix A derives the cell memory requirements. The formula for the minimum cell memory is

$$Mmin = (4+3n)\, r_1^2\, \tau_c \quad (8a^2) + 4\, V r_1 \sin(\tau_c/2)/a^2 \qquad (50)$$

where

r_1 = distance over which the highest cell detail exists (meter).
a = distance between highest cell detail (meter).
τ_c = worst case field of view (radian).
V = velocity of the eye (meter/sec).

---

[5]"Videodisc Based Storage Technology", Computer, Vol. 13, No. 6, pp. 87, June, 1980.

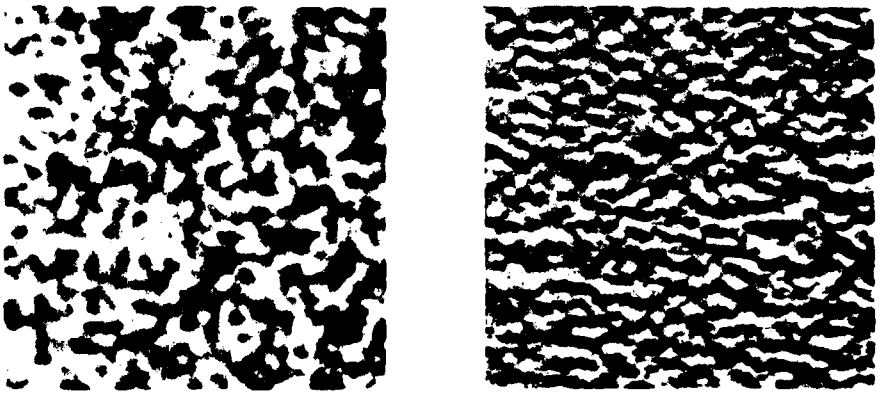
T = the time to transfer data from disk covering the predicted field of view (sec).

$2^n r_1$ = the range limit, Rmax (meter).

For example, if r_1 = 10 meter, a = 0.1 meter, τ_c = 60 ~ 1 radian, V = 50 meter/sec, T = 0.3 sec, Rmax = $2^n r_1$ ~ 20 kilometer, and n = 11; then Mmin = $80*10^3$ data points or 480 kilobytes of memory if 6 bytes are used to describe each data point. Further 30,000 data points would need to be transferred in the 0.3 second interval for an average disk transfer rate of 750 kilobytes/sec. Since disk transfer rates exceeding 1 megabyte/sec are common, one can conclude that a minimum cell memory of between 400 kilobytes to 1 megabyte will be required. The cell memory can be larger if large disk blocks are needed to block out the virtual address space. The smaller the virtual address block the more nearly one can size cell memory to the minimum required.

Terrain Modeling with Random Numbers (Noise)

Terrain modeling is done by means of random number files in REAL SCAN. This provides the ability to create patterns which are described by only a few parameters: amplitude distribution and filter frequency. *Figure 17a illustrates an elevation map of a noise file* having amplitudes 0 through 255. Black corresponds to 0 and white corresponds to 255. Figure 17b illustrates a picture of a diffusely reflective surface having the elevation map of Figure 17a. The ways in which noise files are created and an introduction to the method used to create these pictures are presented in this section.



(a) (b)

Figure 17. Noise Files.

Single valued arrays of random numbers have been created to simulate terrain and texture. These arrays provide a data base to test the concepts of hierarchy and the mapping of real world features by patterns.

The program TRYTEX (Appendix HC) was created to produce this random number array. Two parameters control the creation of the array from a standard random number generator. The first is the frequency of peaks and valleys in the noise. For example, terrain would be a slowly varying pattern while detail, such as trees and leaves, would have higher frequencies. The filtered array is an array that has this desired spatial frequency distribution. Another controlled parameter is the elevation distribution. The elevation distribution of the filtered array is nearly Gaussian. A redistribution technique was developed to change the filtered array to one of four possible distributions: uniform, triangular, parabolic, or cusp.

To generate the filtered redistributed array, the following information is required:

1. A seed for the random number generator
2. The size of the filter
3. The size of the final array
4. Type of elevation distribution
5. The number of passes through the filter

The following technique is used to generate the filtered array. First, an array of random numbers, called the random array is created using a standard random number generator. Next, an array called the weighting array is generated. The dimensions of the weighting array are fs x fs where fs is the filter size. To compute the value of the element (i,j) in the filtered array, the weighting array is placed over the random array at (i,j). The products of the random array values and the weighting array values are summed for all overlapping points. The sum is the value of the filtered array element. Figure 18 shows the point (i,j) of the filtered array being computed. The corresponding point (i,j) in the initial random array is averaged with all the points within the square surrounding it. There are many weighting arrays that could be used. The three that were investigated and pictures of the data bases they generated will be discussed in later sections.

Any number of passes can be made through the filter. These can occur before or after the filtered array has been redistributed.

Two things become apparent. First, all array indices are integer so the filter size must be odd if a center point is used to characterize the filter. Secondly, each time the array is filtered, a number of points are "lost" since there aren't enough neighbors to compute a weighted average. Because of this, the beginning random array must be

larger than the final array by a number of points. This difference is the product of the filter size and the number of passes through the filter. The array size before filtering is called the working size.

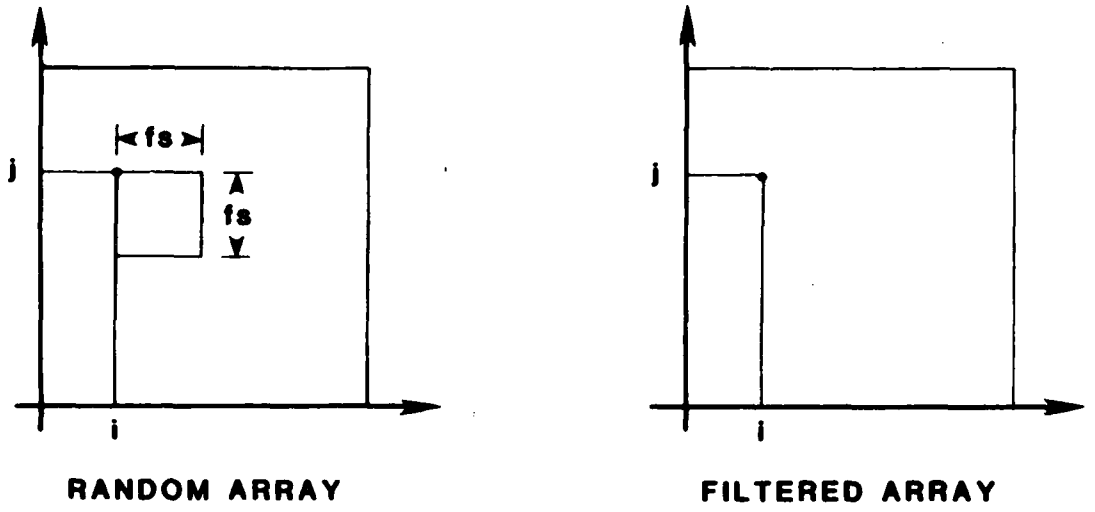


Figure 18. Filtering Concept.

The method of generating the filtered array in the program TRYTEX is more efficient than outlined above, but logically the same. An array of dimensions (filter size * working size) is filled with random numbers. One row of the filtered array can then be computed. Next, the top line of this random array is destroyed, the remaining lines are renumbered as if to shift them up one line, a new line is created in the deleted lines location, but numbered as if it were inserted at the bottom. The next line of the filtered array is then computed.

Once the random array has a desired frequency distribution, it can be redistributed to one of the four amplitude distributions.

The following is an outline of the exact solution for an amplitude redistribution:

1. The array elements are sorted, by magnitude, into a vector of length N*N, the dimensions of the filtered array.

2. The number of values, or buckets, the final distribution will have and how many points should be in each bucket are determined.

3. Starting with the first, each bucket is filled with points from the N*N vector. When a bucket is full, the vector value that marks the end of that bucket and the beginning of

the next is tabulated. These points are called breakpoints and mark the upper and lower bounds of vector values that fall into each bucket. Figure 19 shows breakpoints being calculated for a uniform distribution. Each bucket will hold four points. The first three breakpoints are a1, a5, and a9.

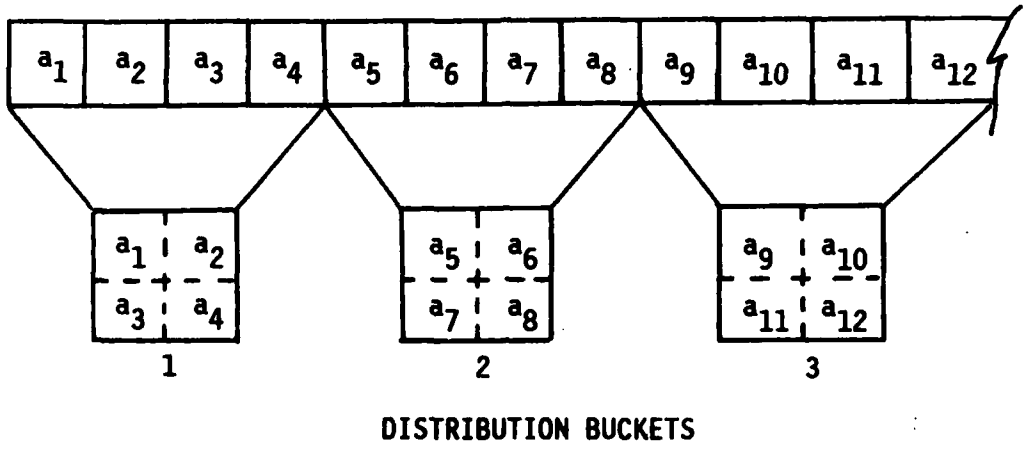


Figure 19. Breakpoint Calculation.

Each array element is assigned the value of the bucket into which it fell by comparing the elements value to the list of breakpoints. Referring again to Figure 19; a1, a2, a3, and a4 will be assigned the value 1.

Such a procedure requires on the order of $N^2(N^2 + 1)/2$ operations for the sort and N^2 operations for the assignment. An approximation method for determining breakpoints which requires on the order of N^2 operations is outlined below:

1. The number of buckets in the final distribution (distribution buckets) is determined.

2. The filtered array is divided into ten times as many equally spaced intervals as there are distribution buckets. Each interval in the filtered array is called an array bucket. The following equation is used to compute the points that mark the boundaries of the intervals:

$$bnd(i) = i[(max - min)/(10ndb)]$$

where

bnd(i) is the point that marks the upper bound of interval i and the lower bound of interval i + 1.
max is the maximum value in the filtered array
min is the minimum value in the filtered array.
ndb is the number of distribution buckets

3. The number of elements in each of the array buckets is determined. This is accomplished in TRYTEX by scaling the array to the number of array buckets and using the scaled values as an index for a vector. This vector counts the number of values in each array bucket. The distribution of filtered array amplitudes in any array bucket is assumed to be uniform between the breakpoints associated with that array bucket.

The **FORTRAN** code that accomplishes steps (2) and (3) is shown below:

```
C
C
C   G(K) IS THE Kth ARRAY BUCKET
C   BUF IS THE FILTERED ARRAY
C   DEP = 1/(FILTMAX - FILTMIN)
C       FILTMAX IS THE MAXIMUM VALUE IN BUF
C       FILTMIN IS THE MINIMUM VALUE IN BUF
C   LIMT IS THE NUMBER OF ARRAY BUCKETS
C
C
C
C          CREATE THE DISTRIBUTION ARRAY
C
          DO 15 K = 1,LIMT + 1
             G(K)=0
 15       CONTINUE
          DO 21 I = 1,N
             DO 21 J = 1,N
                K=(BUF(J,I) - FILTMIN)*DEP + 1
                G(K) = G(K) + 1
 21       CONTINUE
```

4. The number of points in each of the distribution buckets is determined.

5. Each of the distribution buckets is filled by taking full array buckets or portions of array buckets, and again determining where the breakpoints should occur for the final distribution.

6. Each element of the filtered array is assigned to the value of the distribution bucket between whose breakpoints it falls.

Step one indicates that the number of array elements contained in each distribution bucket must be determined for any of the possible distribution.

The following variables are used in the discussion of distribution bucket size determination:

R is the range of the final array (or the number of distribution buckets)
A is the number of occurences of the most frequent value
N is the dimension of the filtered array
f is the location of the distribution peak as a fraction of R.

UNIFORM DISTRIBUTION

A uniform distribution has the frequency diagram shown in Figure 20.

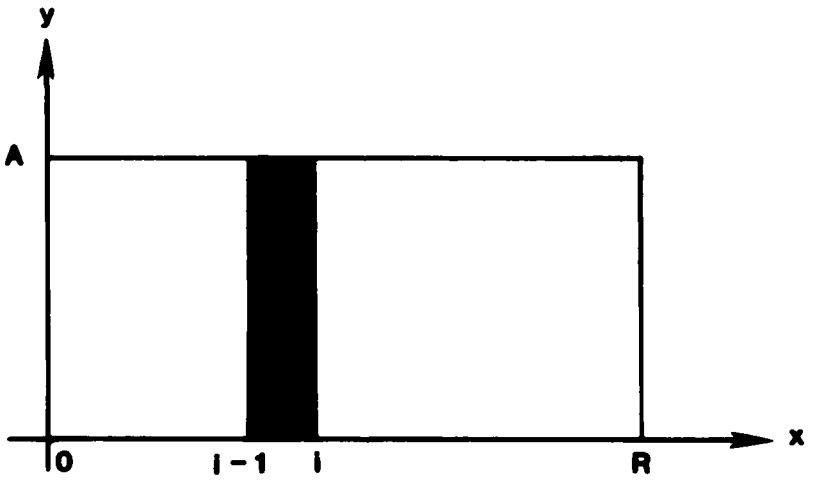


Figure 20. Uniform Distribution Plot.

The equation describing this curve is:

$$y(x) = A; \text{ for } 0 < x < R \tag{51}$$

Since the area under the curve is A*R and the total number of points under the curve is N^2 and

$$A = N^2/R. \tag{52}$$

NU(i), the number of points in the ith distribution bucket (i.e., shaded area in Figure 20), will be N^2/R. The FORTRAN code that computes this is shown below:

```
C
C
C  BUCK(I) = NU(i)
C
C  UNIFORM
C
 2100   TEMP1 = (1.*N*N)/R
        DO 2010 I = 1,R
            BUCK(I) = TEMP1
 2010   CONTINUE
        GOTO 2114
```

The mathematical development of the triangular, parabolic, and cusp distributions is developed in Appendix I.

The number of points in each distribution bucket for a particular distribution has been determined. The following procedure is used to redistribute the filtered array to this new distribution:

1. Start with the first distribution bucket and the first array bucket.

2. The size of the array bucket is compared to the number of the points that will fit into the distribution bucket:

   a. If all the points from the array bucket will fit into the distribution bucket, the number of points needed to fill the distribution bucket is decreased by the number of points in the array bucket. The above comparison is made using the same distribution bucket and the next array bucket.

   b. If all points from the array bucket will not fit into the distribution bucket, enough points are removed from the array bucket to fill the distribution bucket, and a breakpoint is determined for the distribution bucket. This breakpoint is a value between the upper and lower bounds of the array bucket. The actual value depends on the fraction of the array bucket points that have been removed and is determined as follows:

$$\text{breakpoint} = \text{min} + \text{fraction}(\text{max} - \text{min})$$

$$\text{min} = \text{array bucket lower bound}$$

max = array bucket upper bound

fraction = fraction of array bucket used

The following FORTRAN code computes the distribution bucket breakpoints:

```
C   FILTMIN = THE MINIMUM VALUE OF THE FILTERED ARRAY
C   LIMT = THE NUMBER OF ARRAY BUCKETS
C   EPSILON = THE DISTRANCE BETWEEN EACH ARRAY BUCKET
C
C
C      COMPUTE THE BREAKPOINTS
C
          K = 0
          A = 0
          DO 2800 I=1,QSIZE
 2750        IF(A.LT.BUCK(I))THEN
                K=K+1
                IF(K.GT.LIMT)GOTO 2800
                A=A+G(K)
                GOTO 2750
             ENDIF
             A=A-BUCK(I)
             B(I)=(K-A/G(K))*EPSILON+FILTMIN
 2800     CONTINUE
          B(QSIZE)=FILTMAX
```

3. When all the distribution buckets are full, final values are assigned to the array elements. By comparing the present value of the array element to the breakpoints for the distribution buckets, the bucket into which the point should go can be found. The value of that distribution bucket is then assigned to the array element.

WEIGHTING ARRAYS

The following subsection is a discussion of three weighting arrays used to filter the random array. The three types of filters covered are the simple average filter, the sin(x)*sin(y) filter and the polar filter. Pictures of data bases created using each filter are included with the discussion of that filter. Therefore, a brief introduction of the types of pictures generated and the methods used to create them is necessary. Two types of pictures will be used to illustrate the results, the altitude map and the sun reflection picture. Each of these offers different information about the data base.

The easiest picture to create is the altitude map. The highest elevations are white, the lowest elevations are black, and other

points are various shades of gray, depending on their elevation. Pictures of this type offer little information about relative slopes or how the pattern will appear in three dimensions, but show the exact elevation of a point.

The other type of picture, the sun reflection picture, places the observer directly above the data base. The light source can be placed anywhere above the data base. Pictures generated using this routine give light intensities that depend on the angle between the reflecting surface and the incoming illumination. Figure 21 illustrates the vector from an arbitrary point on a surface, Z(x,y), pointed toward the source of illumination. This gives results similar to a diffuse object lit by a point source. This technique is discussed in detail under Reflection From Diffuse Surfaces.

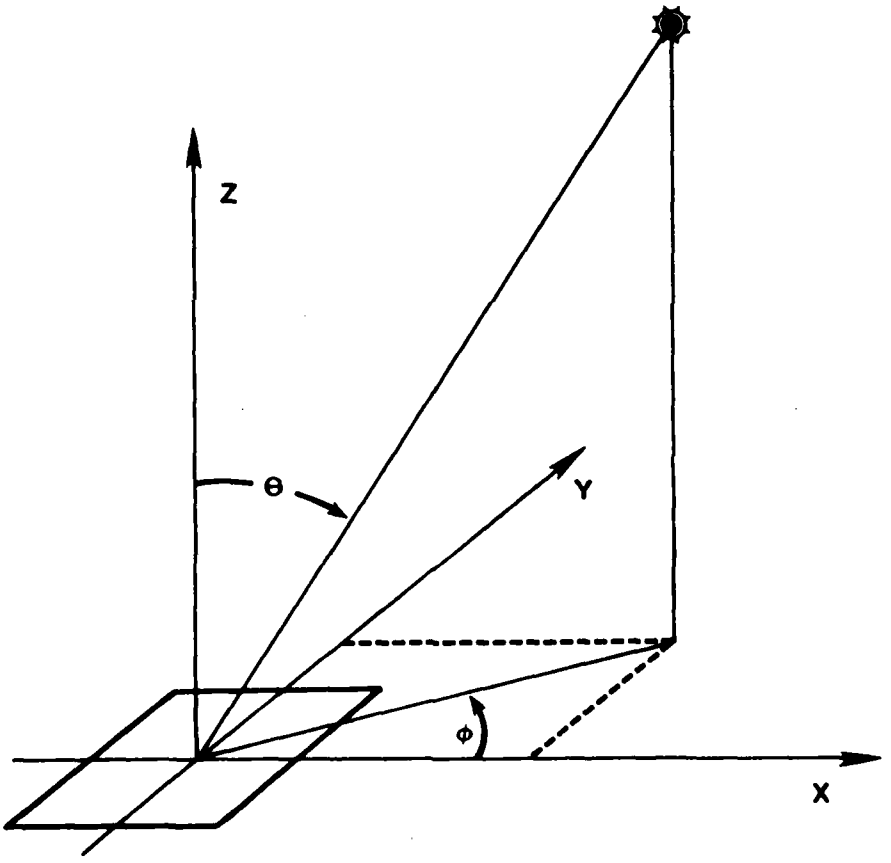


Figure 21. Sun Angle Geometry.

All the pictures that follow were made using the **DICOMED**. Since the **DICOMED** is capable of producing 256 different light intensities, the number of distribution buckets is set at 256. The array size for all the pictures is 512 x 512.

The first and simplest filter tried was the straight average filter. All the weighting array elements are given the same value. This is similar to a moving average in two dimensions. The shortcomings of this particular filter can easily be seen by looking at Figures 22 and 23. They are strongly correlated in the x and y directions. Figure 22 is the altitude map of a 512 x 512 array made with this simple filter. Figure 23 is a sun reflection picture of the same data base, with the sun vector pointed from the top to the bottom of Figure 22 (i.e., Figure 21 parameters $\theta = 45°$, $\Phi = 90°$).

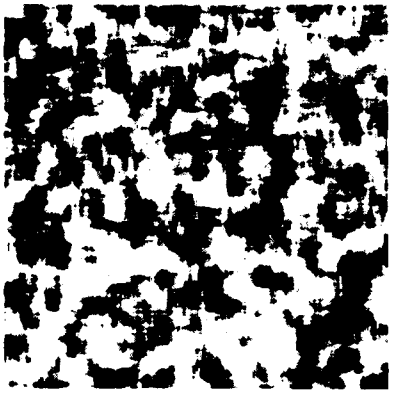


Figure 22. Altitude Map.
Filter Size: 25
Distribution: uniform
Filter Type: simple average

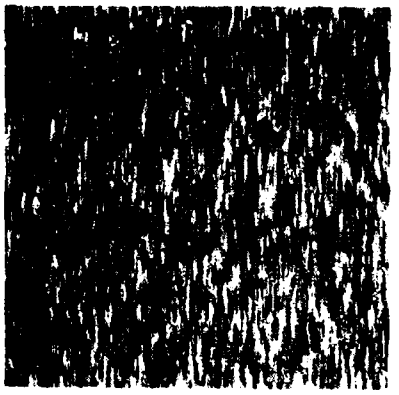


Figure 23. Sunlight Picture.
Filter Size: 25
Distribution: uniform
Filter Type: simple average

The first filter that offered an acceptable picture was the single pulse sin(x)*sin(y) filter. The weighting array for this filter is computed using

$$w(i,j) = \sin[\pi i/(fs+1)] \sin[\pi j/(fs+1)]$$

where fs = filter size

The half period of the sine function is one greater than the filter size so the weighting factor for all points inside the array is non-zero. The sine weighting array for a filter size of seven is shown in Figure 24.

0.146	0.271	0.358	0.383	0.358	0.271	0.145
0.271	0.500	0.604	0.707	0.604	0.500	0.271
0.358	0.604	0.854	0.924	0.854	0.604	0.358
0.387	0.707	0.924	1.000	0.924	0.707	0.387
0.358	0.604	0.854	0.924	0.854	0.604	0.358
0.271	0.500	0.604	0.707	0.604	0.500	0.271
0.146	0.271	0.358	0.383	0.358	0.271	0.146

Figure 24. Sine Weighting Array for Filter Size of 7.

The **FORTRAN** code for computing the weighting factors of the single pulse sin(x)*sin(y) filter follows:

```
C       SETUP THE WEIGHT FOR FILTER ARRAY
 5            WI = 3.14159/(IFLTSIZE + 1)
              DO 200 I = 1,IFLTSIZE
                 FST = SIN(WI*I)
                 DO 200 J = 1,IFLTSIZE
                    W(J,I) = FST*SIN(WI*J)
 200          CONTINUE
```

It can be shown that the sin(x)*sin(y) filter does not possess full polar symmetry. Looking again at Figure 24, the weighting factor of the point (4,8) (outside the square) is, of course, 0. It's distance from the center is 4 units. The point (7,7) has a weighting factor of .146, but its distance from the center is 4.24. This is further than the point (4,8), yet its weighting factor is greater. This appearance of "extra" values at the corners of the sin(x)*sin(y) filter leads to a correlation in the x and y directions and causes visible lines in the pictures in these directions. Figures 25 and 26 were generated using the single pulse sin(x)*sin(y) filter.

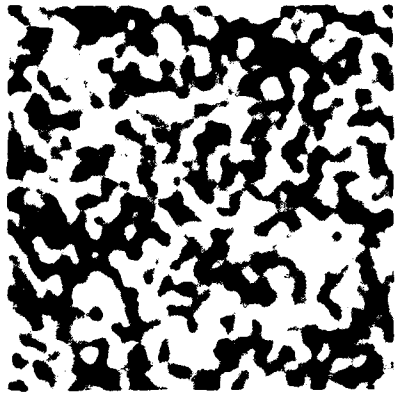


Figure 25. Altitude Map.
Filter Size: 25
Distribution: uniform
Filter Type: sin(x)*sin(y)



Figure 26. Sunlight Picture.
Filter Size: 25
Distribution: uniform
Filter Type: sin(x)*sin(y)

A third filter, with full polar symmetry, was tried, and succeeded in eliminating the horizontal and vertical line structures. The weighting factors for the polar filter are computed as follows: first, the distance from the array element to the center is calculated. If this distance is greater than (fs + 1)/2, one-fourth of the polar filters period, the point is given a weighting factor of zero. Otherwise, the array element is given a value from the following equation:

$$w(i,j) = \cos\sqrt{[(\pi/fs + 1))((i - c)^2 + (j - c)^2))]} \qquad (52)$$

where (c,c) is the center, and fs is the filter size.

An example of the polar weighting array, using a filter size of seven, is shown in Figure 27.

0.000	0.154	0.323	0.383	0.323	0.154	0.000
0.154	0.444	0.639	0.707	0.639	0.444	0.154
0.323	0.639	0.850	0.924	0.850	0.639	0.323
0.383	0.707	0.924	1.000	0.924	0.707	0.383
0.323	0.639	0.850	0.924	0.850	0.639	0.323
0.154	0.444	0.639	0.707	0.639	0.444	0.154
0.000	0.154	0.323	0.383	0.323	0.154	0.000

Figure 27. Polar Weighting Array Filter Size of 7.

The **FORTRAN** code to compute the weighting factors (which takes advantage of the symmetry) is shown below:

```
C       SETUP THE WEIGHT FOR FILTER ARRAY
C
        FAC = 3.14159265/(IFLTSIZE + 1)
        ICENT = IFLTSIZE/2 + 1
        IDIAM = IFLTSIZE + 1
        W(ICENT,ICENT) = 1.
        DO 200 J = ICENT + 1,IFLTSIZE
           K = J - ICENT
           SQR = K*K
           KY = IDIAM - J
           DO 200 I = ICENT,J
```

```
                K = I - ICENT
                XK = SQRT(K*K + SQR)
                KX = IDIAM - I
                IF(XK.GE.ICENT)THEN
                   VAL = 0.0
                ELSE
                   VAL = COS(XK*FAC)
                ENDIF
                W(I,J) = VAL
                W(J,I) = VAL
                W(I,KY) = VAL
                W(KY,I) = VAL
                W(J,KX) = VAL
                W(KX,J) = VAL
                W(KX,KY) = VAL
                W(KY,KX) = VAL
200     CONTINUE
```

Figures 28 and 29 were created using the polar filter. All other parameters remained the same as in the pictures produced for the sin(x)*sin(y) filter of Figures 25 and 26.

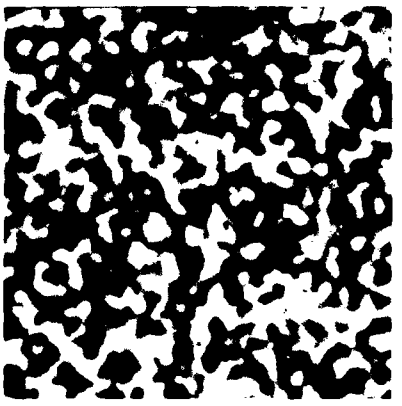


Figure 28. Altitude Map.
Filter Size: 25
Distribution: uniform
Filter Type: polar filter



Figure 29. Sunlight Picture
Filter Size: 25
Distribution: uniform
Filter Type: polar filter

REFLECTION FROM DIFFUSE SURFACES

The following is a detailed discussion of the algorithm used to create the sun pictures. The intent is to create an image of the data base that appears to be illuminated by a point source of light. The light source is an infinite distance away, so the angle of incidence of the light is the same at all points in the data base. The observer is placed directly above the data base. The amount of light reflected to the observer (i.e., the apparent intensity) depends on the difference in the angle of the surface normal and the incident angle of the light. This approximation is exact if the light is reflected from a perfect diffuse reflector.

Using vector notation, the intensity of a point can be calculated as follows:

$$I = \overline{h} \cdot \overline{ds}; \text{ if } \overline{h} \cdot \overline{ds} > 0 \text{ (i.e., sunny side)} \quad (53)$$

$$I = 0 \quad ; \text{ if } \overline{h} \cdot \overline{ds} < 0 \text{ (i.e., dark side)} \tag{54}$$

where $\overline{h}$ is the vector direction to the sun

and $\overline{ds}$ is the outward vector normal from the surface.

Letting $\hat{ds} = \overline{ds}/\ \overline{ds}$, $\overline{ds}$ can be computed by taking the cross-product of any two vectors tangent to the surface element, $\overline{ds}$. The two that are most convenient to compute are the two parallel to the coordinate axes.

Given the surface $z = f(x,y)$, two tangent vectors at the point $(Xo,Yo,f(Xo,Yo))$ can be computed by considering: $z_1 = f(x,Yo)$ and $z_2 = f(Xo,y)$. This produces two lines on perpendicular planes, both of which pass through the point $(Xo,Yo,f(Xo,Yo))$. By computing the tangents to these lines, the normal to the surface at that point can be computed.

$$\hat{a} = (1,0,dz_1/dx) \quad ; \text{ for } (x,y) = (Xo,Yo) \tag{55}$$

$$\hat{b} = (0,1,dz_2/dy) \quad ; \text{ for } (x,y) = (Xo,Yo) \tag{56}$$

$$\overline{ds} = \hat{a} \times \hat{b} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ 1 & 0 & dz_1/dx \\ 0 & 1 & dz_2/dy \end{vmatrix} \tag{57}$$

$$\hat{ds} = \frac{-dz_1/dx\ \hat{i} - dz_2/dy\ \hat{j} + \hat{k}}{\sqrt{1 + (dz_1/dy)^2 + (dz_2/dx)^2}} \tag{58}$$

For a discrete value data base, the partial derivatives at the point (Xo,Yo) are:

$$\partial z/\partial x = dz_1/dx = f(Xo+1,Yo) - f(Xo,Yo) \tag{59}$$

$$\partial z/\partial y = dz_2/dy = f(Xo,Yo+1) - f(Xo,Yo) \quad (60)$$

The sun angles are entered into the program in polar coordinates. Only two angles must be entered. The angle of the sun down from the z axis is the polar angle, θ. The cylindrical angle, ϕ, is the angle from the x axis towards the y axis to the projection of the sun on the X-Y plane. Figure 21 shows these angles graphically.

$\bar{h}$ can be converted to its rectangular coordinates as follows:

$$h_i = |h| \sin\theta \cos\phi \quad (61)$$

$$h_j = |h| \sin\theta \sin\phi \quad (62)$$

$$h_k = |h| \cos\theta \quad (63)$$

Converting Equation (53) to its rectangular coordinate equivalent, yields

$$I = (h_i \cdot ds_i) + (h_j \cdot ds_j) + (h_k \cdot ds_k) \quad (64)$$

$$I = \frac{|h|}{\sqrt{1 + (\partial z/\partial x)^2 + (\partial z/\partial y)^2}} [-(\partial z/\partial x)\sin\theta \cos\phi - (\partial z/\partial y)\sin\theta \sin\phi + \cos\theta] \quad (65)$$

h can be used as a scaling factor to restrict the magnitude of the intensities to any range.

The average maximum slope in the scene can be controlled. This allows noise files amplitude range to be normalized to a predetermined value. For example, all test noise files are stored as byte. This allows for 256 different elevations. If the filter size for a particular noise file is small (e.g., 3) the resulting picture will contain close tall thin projections. The amplitudes of the noise files must be scaled if the apparent slope is to be properly controlled to simulate real world terrain or features. The method chosen to accomplish this is by slope limiting.

The contour of the data base is estimated to be sinusoidal, as shown in Figure 30.

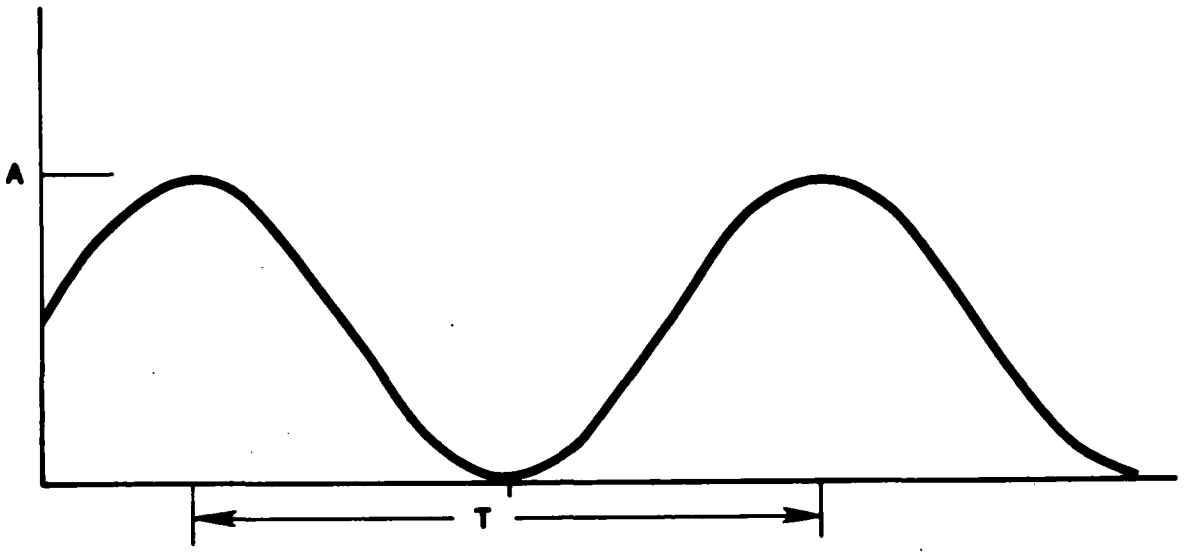


Figure 30. Idealized Filtered Noise Contour.

The equation that describes Figure 30 is

$$y = A \sin (2\pi/T)t \tag{66}$$

where T depends on the filter size. To determine the maximum slope, differentiate Equation (66)

$$dy/dt = (2\pi A/T)\cos(2\pi t/T) = 0 \tag{67}$$

The maximum slope occurs at T/2, so the maximum slope is

$$dy/dt = 2\pi A/T \tag{68}$$

T can be estimated by counting the light to dark transitions along a straight horizontal or vertical line in an altitude map of the data base. The empiracally determined equation that relates the period, T, to the filter size, fs, is:

$$T = 1.6fs = \text{period (in pixels)} \tag{69}$$

The maximum slope, ms, can be computed from Equations (68) and (69)

$$ms = 2\pi A/(1.6fs) = 3.93\ A/fs \tag{70}$$

In order to properly investigate the texture created by filtered and redistributed noise, one must simulate the slope of the texture pattern, as that pattern will be scaled in a real world scene. Hence, an estimate of the real world slope for Figure 30 is made. This is called the desired slope, md. Then one can calculate a slope limiting factor, SLF

$$SLF = md/ms = 0.2546\ md\ fs/A \tag{71}$$

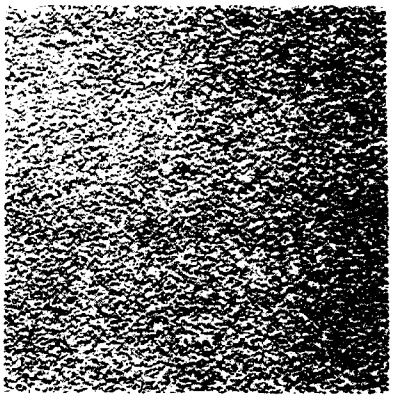
For example if the texture is to represent terrain then $md \sim 0.1$ whereas if the texture is to represent trees then $1 < md < 10$. Since the texture resolution has been set to 8 bits, the slope limiting factor allows one to compare the filtered noise patterns with photographs of established appropriate scaling parameters.

Three parameters are required to make a sunlight picture (i.e., Figures 26 or 29) for evaluating filtered noise textures as real world types. The parameters are the polar angle of the sun from the z axis, Θ; the cylindrical angle, Φ; and the slope limiting factor, SLF.

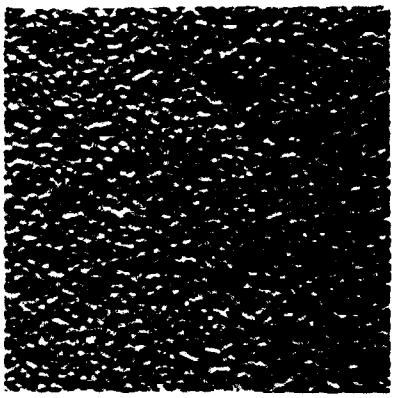
The following sets of pictures are results of the program RLPCAL (i.e., RLPCAL is listed in Appendix HB).

Figure 31 shows the effect of changing the filter size, fs, and the elevation distribution. Figure 31a has a filter size of 3 on a uniform elevation distribution. FIgure 31b has a filter size of 11 on a cusp distribution having few low elevations and many high elevations. Figure 31c illustrates a filter size of 25 on a parabolic elevation distribution peaked at the elevation range midpoint. The cylindrical angle, Φ, is 90° for all the photographs of Figure 31. The polar angle, Θ, is 60° for Figures 31a and 31b and 45° for Figure 31c. The slope factor used to scale Figure 31a to simulate a texture somewhere between grass and leaves is 0.125 yielding a desired slope of about 20. Figure 31b has a slope factor of 0.022 yielding a desired slope of about 1.0 for simulating grass or tree texture. Figure 31c has a slope factor of 0.0075 yielding a desired slope of about 0.15 for simulating terrain.

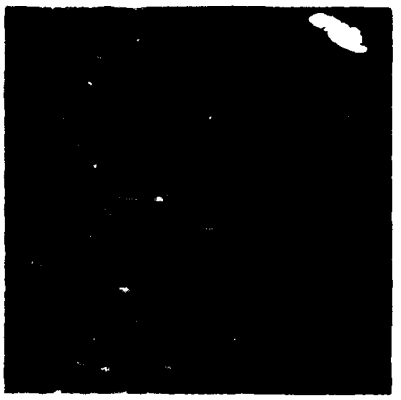
Figure 32 illustrates the effect of varying the polar angle. All pictures are for the same noise file and all other parameters constant (i.e., fs is 25, parabolic distribution, Φ is 90°, SLF is 0.125). There is no effect of sun shadowing in Figure 32, only the effect of changing the light source direction from 45° from vertical for Figure 32a to 80° from vertical for Figure 32c.



31a
Filter Size, fs = 3
Uniform Distribution
Polar Angle, $\theta = 60°$
Cylindrical Angle, $\phi = 90°$
Slope Factor, SLF = 0.125.



31b
Filter Size, fs = 11
Cusp Distribution
Distribution Peak, f = 1.0
Polar Angle, $\theta = 60°$
Cylindrical Angle, $\phi = 90°$
Slope Factor, SLF = 0.022

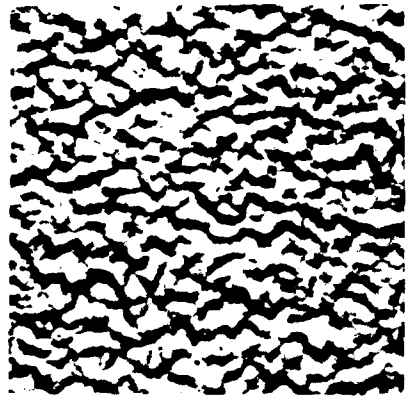


31c
Filter Size, fs = 25
Parabolic Distribution
Distribution Peak, f = 0.5
Polar Angle, $\theta = 60°$
Cylindrical Angle, $\phi = 90°$
Slope Factor, SLF = 0.125

Figure 31. Sunlight Pictures as a Function of Filter Size.

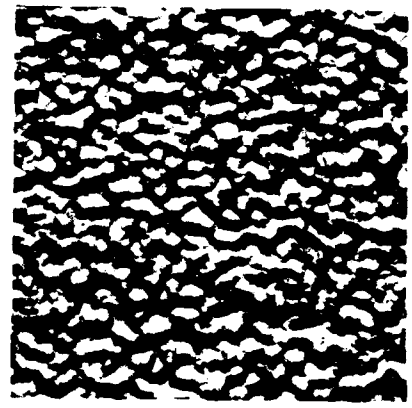
32a

Polar Angle, θ = 45°



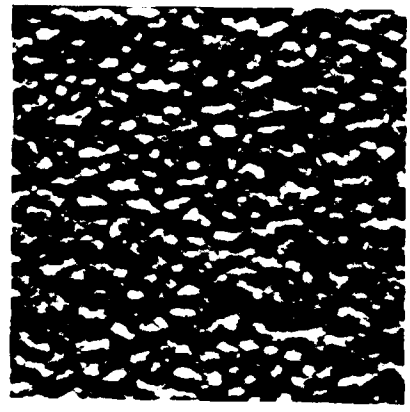
32b

Polar Angle, θ = 60°



32c

Polar Angle, θ = 80°



Filter Size, fs = 25; Cylindrical Angle, Φ = 90°; Slope Factor, SLF = 0.125; Parabolic Elevation Distribution; Distribution Peak, f = 0.50.

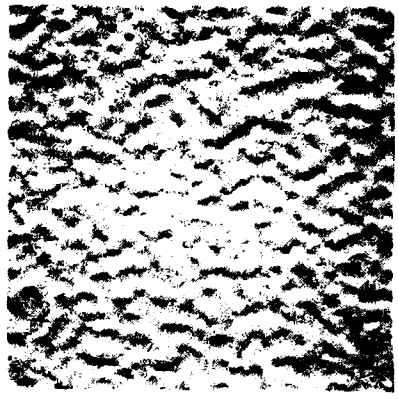
Figure 32. Sunlight Pictures as a Function of Polar Angle.

Figure 33 illustrates the effect of varying the slope factor, SLF. All pictures are of the same file (i.e., filter size of 25, polar angle of 60°, cylindrical angle of 90°, parabolic distribution with the distribution peak centered about the elevation range). The slope factor for Figures 33a through 33c are respectively 0.0075, 0.050 and 0.125.

Figure 34 illustrates the effect of varying the cylindrical angle. All pictures are for the same parabolic distribution file (i.e., filter size of 25, polar angle of 60°, and slope factor of 0.050). The cylindrical angle varies from 90° to 180°, showing apparent ridge structure dependent upon the direction to the light source.

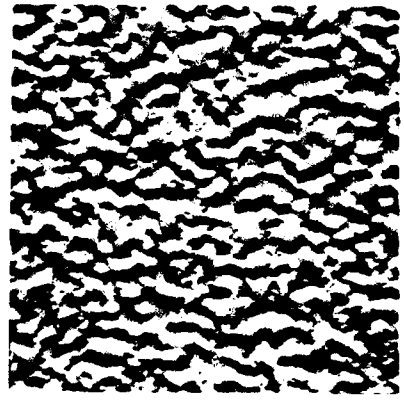
33a

Slope Factor, SLF = 0.0075



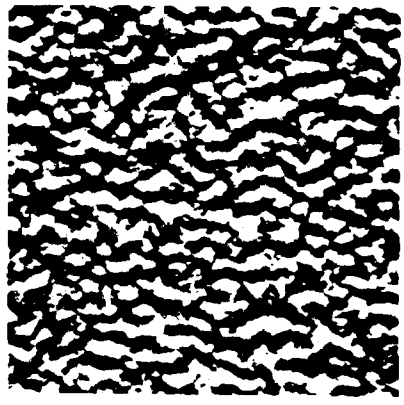
33b

Slope Factor, SLF = 0.050



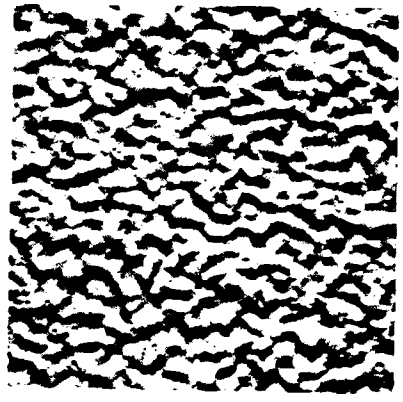
33c

Slope Factor, SLF = 0.125



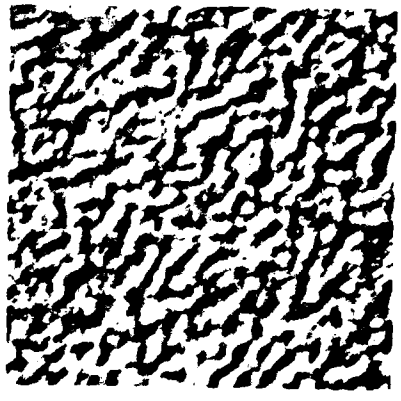
Filter Size, fs = 25; Cylindrical Angle, $\Phi = 90°$; Polar Angle, $\Theta = 60°$; Parabolic Elevation Distribution; Distribution Peak, f = 0.50.

Figure 33. Sunlight Pictures as a Function of Average Texture Slope.



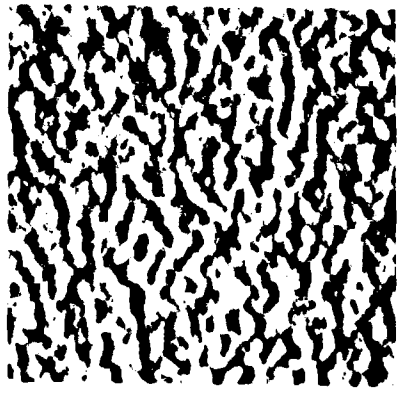
34a

Cylindrical Angle $\Phi = 90°$



34b

Cylindrical Angle $\Phi = 135°$



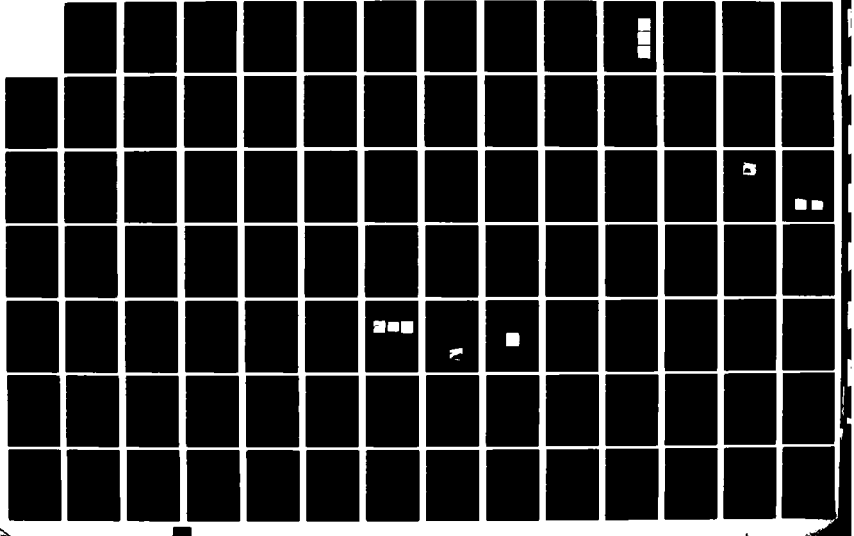
34c

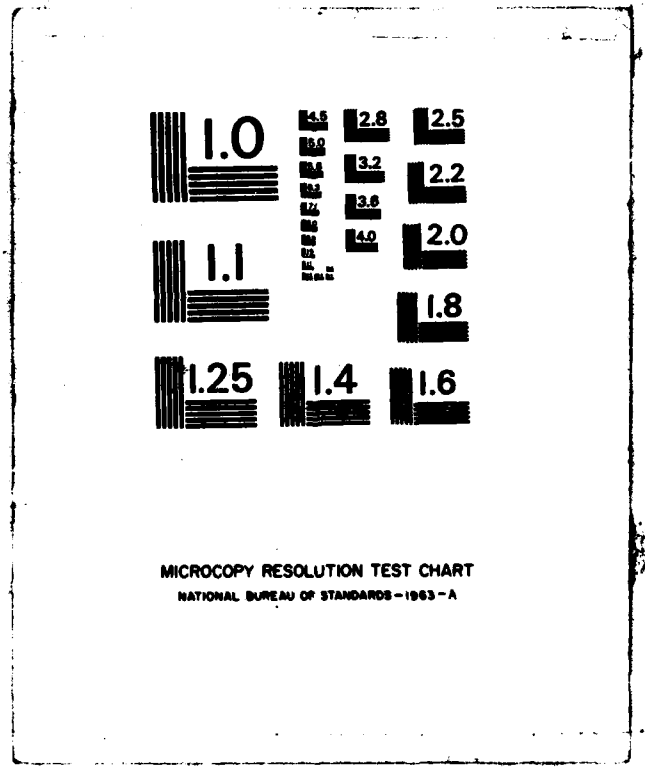
Cylindrical Angle $\Phi = 180°$

Filter Size, fs = 25; Polar Angle, $\theta = 60°$; Slope Factor, [illegible]
Parabolic Elevation Distribution; Distribution Peak [illegible]

Figure 34. Sunlight Pictures as a [illegible]

AD-A122 000 REAL SCAN EVOLUTION(U) UNIVERSITY OF CENTRAL FLORIDA 2/6
ORLANDO DEPT OF ELECTRICAL ENGINEERING B W PATZ ET AL.
FEB 82 NAVTRAEQUIPC-80-D-0014-2 N61339-80-D-0014
UNCLASSIFIED F/G 9/2 NL




1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

INTERPOLATION FORMULA

Two methods have been investigated for interpolating data on a fixed grid, Overhauser-Coons (17) and the straight line patch. Only the straight line patch method has been used to create simulated terrain, since it requires substantially less computation than Overhauser-Coons formula. Appendix N develops the mathematics for testing the elevation error associated with the Overhauser-Coon approximation to a given six(x)*sin(y) function. Appendices GB and GC list the programs used in the elevation error test. Appendix D develops additional interpolation formula for further evaluation.

An interpolation function is used for reducing the size of the data base required to describe a single value surface. The function interpolates the height of points on a surface given the height values of points adjacent to the region. A surface is interpolated by sectioning the surface into many square patches. The data base required is a sample data array of the surface containing the adjacent points to each patch. One way of testing the interpolation accuracy of a routine is by evaluating the difference of the interpolated value and the exact value of a test function, FT(X,Y), at a point. Another way is by evaluating the angular difference between the surface normals for the exact and the test function. The maximum error is found to be a function of the distance between the sample points in the data array. More specifically, the smaller the distance between the sample data points, the smaller the error. Therefore, selection of the distance between the sample points is a compromise between the need for data base reduction and the tolerance for maximum error.

THE INCREMENTAL DISTANCES

The incremental distances within the patch, t(X) and t(Y), are determined for each point interpolated within the patch. Given a point (X,Y) to be interpolated within the patch, the incremental distance t(X), is the distance from the point to the boundary line C2(Y), Figure 35. The incremental distance, t(Y), is the distance from the point to the boundary to the line C1(X). The incremental distance is then normalized by dividing by the width of the patch illustrated in Figure 35.

Figure 35 illustrates a regular patch geometry. The points P_4, P_5, P_8 and P_9 bound the patch. Only the four bounding patch points are needed for the straight line interpolation formula

---

[17] Brewer, J. and Anderson, D. "Visual Interaction with Overhauser-Coons Curves and Surfaces", Computer Graphics, Vol. 11, No. 2, pp. 133-137, Summer 1977.

$$Z(x,y) = Z_1 + (Z_2 - Z_1)(x - P_8(x))/d + (Z_3 - Z_1)(y - P_8(y))/d + (Z_4 - Z_3 - Z_2 + Z_1)(x - P_8(x))(y - P_8(y))/d^2 \quad (72)$$

where

d is the patch width

$P_8(x)$ is the x coordinate of point P_8

$P_8(y)$ is the y coordinate of point P_8.

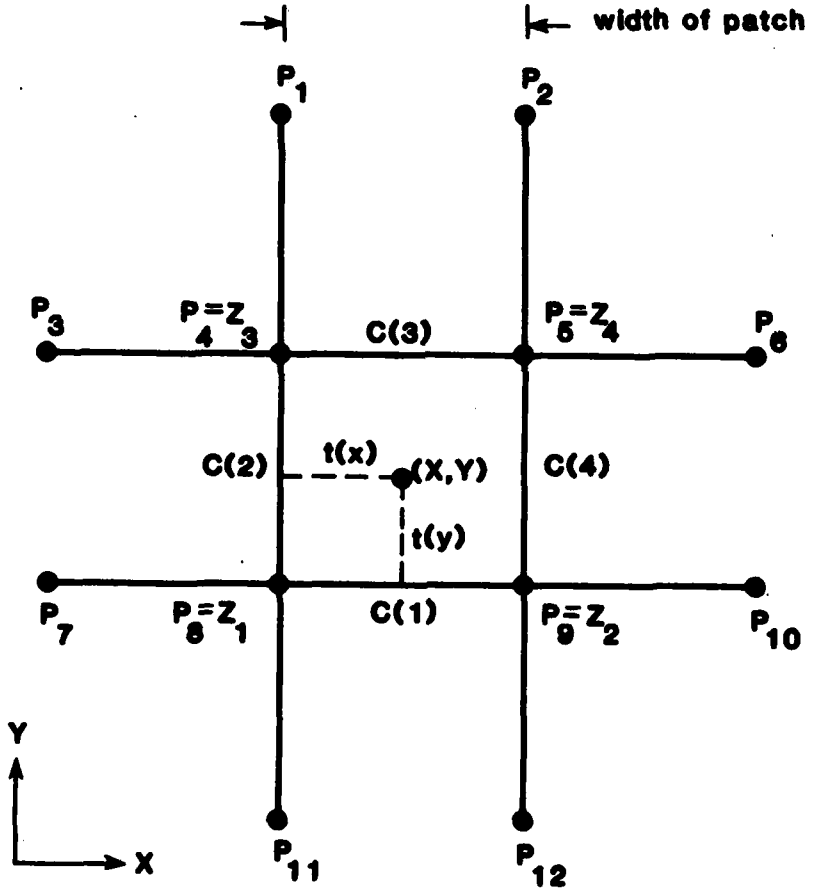


Figure 35. Square Interpolation Patch Geometry.

The Overhauser-Coons (O-C) interpolation formula requires all twelve points P_1 through P_{12}. The beauty of the O-C formula is that it generates a bi-cubic function possessing slope and elevation continuity everywhere while the straight line approximation exhibits

slope discontinuity at the patch boundaries, C(1) through C(4) of Figure 35. However, the O-C interpolation formula is more complex

$$Z(x,y) = C1(x)B0(y) - Z_1B0(x)B0(y) + C2(y)B0(x) - Z_2B1(x)B0(y) + C3(x)B1(y) - Z_3B0(x)B1(y) + C4(y)B1(x) - Z_4B1(x)B1(y) \quad (73)$$

where

$$Cn(x) = \sum_{i=1}^{4} V(n,i)\, t^{4-i}(x); \text{ Overhauser function}$$

$$Cn(y) = \sum_{i=1}^{4} V(n,i)\, t^{4-i}(y); \text{ Overhauser function}$$

$B0(p) = 1 - 3t^2(p) + 2t^3(p)$; Coon's blending function

$B1(p) = 3t^2(p) - 2t^3(p)$; Coon's blending function

$t(x) = (x - P_8(x))/d$; normalized x interpolation distance

$t(y) = (y - P_8(y))/d$; normalized y interpolation distance

Simulated pictures will generate a subjective measure of the quality of an interpolation formula. However, effective evaluation must be based upon appropriate numerical measures of the formula (i.e., elevation error, surface normal error, texture statistics, etc.). If appropriate numerical measures can be discovered then only a few subjective evaluations need be made for a few types of scenes. The numerical measures can be used to perform a large volume of analysis via computer. The computer analysis will then separate and classify the quality of the interpolation function for mapping various real world features. Then subjective evaluations can be performed to verify that the numerical measures actually provide valid evaluations. The computer analysis allows analysis to be performed on unique features and allows the objective evaluation of a large amount of data and ranking by the means of the numerical measure.

Only one numerical measure has been developed to date, the elevation error measure. Three elevation error measures are calculated: maximum error, average absolute error, and rms error. Analysis to date does not indicate any of the three superior to the others, since they are ordered from maximum, average, to rms and typically have ratios about 12:5:4.

Figure 36 is a plot of the normalized rms elevation error for both the straight line interpolation formula and the Overhauser-Coons interpolation formula as a function of normalized patch size. The function interpolated is Asin(x)·sin(y). The normalized error is rms error divided by A. The patch size is normalized to the Nyquist rate (i.e., patch size, d, divided by half the period of the trigometric term).

Figure 36 indicates the sample space must be about 9 times the Nyquist rate to achieve a 1% error for the straight line interpolation, or about four times the Nyquist rate for O-C. If a 10% elevation error is tolerable the sampling interval is about three times the Nyquist rate for the straight line interpolation and about 1.8 times for the O-C interpolation formula.

Further, work needs to be done to create subjective picture evaluations correlated with elevation error measures. The surface normal error measure also needs to be created and evaluated against interpolation formula such as those developed in Appendix D.

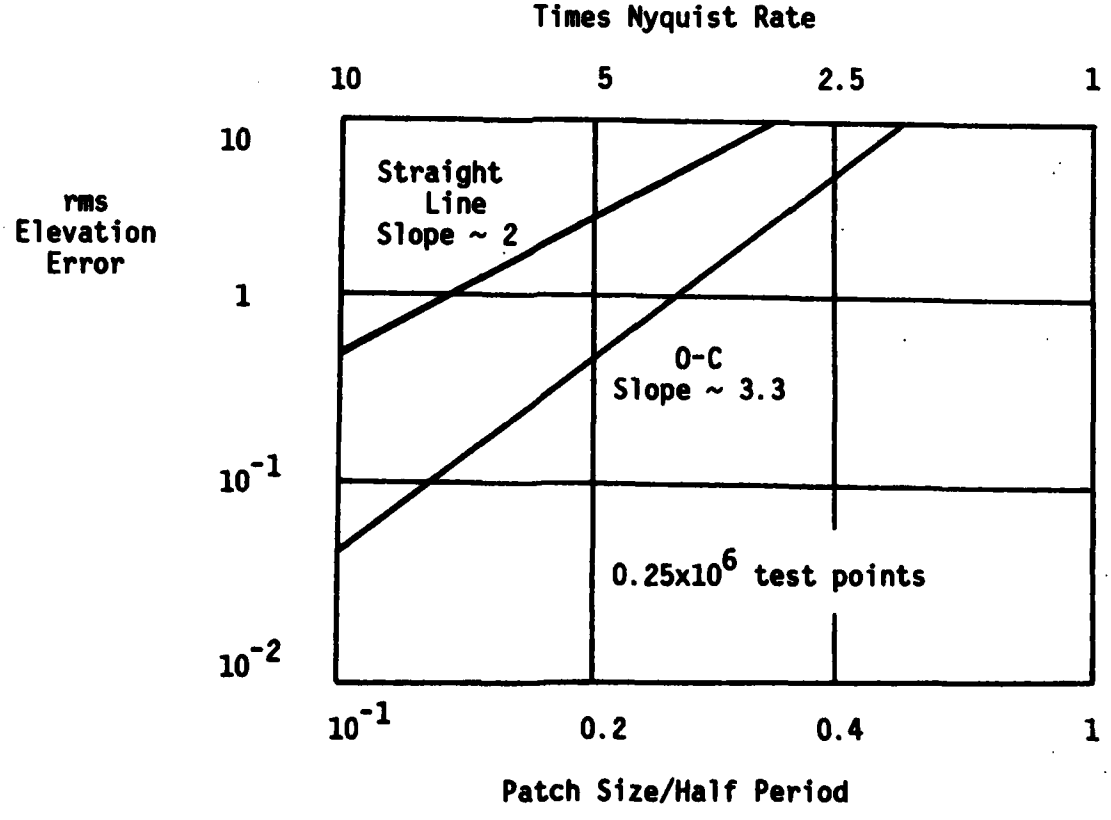


Figure 36. rms Elevation Error versus Normalized Patch Size.

SECTION IV

ANTI-ALIASING FILTER METHODS

Realistic scene creation requires the elimination of CIG display defects induced by the quantization effects of the display and defects due to the sampling of the data base. If the data base used is of greater resolution than that of the display, then the elimination of image defects by the processing of high density, high resolution data may be defined as filtering. REAL SCAN attempts to match the computed scene detail to the display limit in two ways; first, by providing prefiltered hierarchical data base and, second, by slightly over-sampling the detail limited data base. Scene detail is projected back to the display at sub-pixel accuracy but is limited to display no higher resolution than can properly be resolved on the display. REAL SCAN offers an approach to the high pass-low pass filter decision because data sorts are not required as in planar CIG systems. High pass edge accentuation is used when scene detail does not exceed the Kell factor limit of 3 pixels per line pair, while low pass edge laboring is used when scene detail does exceed the Kell factor limit. Contiguous data being processed to a display pixel sequentially allows one to make the high pass-low pass decision simply and with negligible ambiguity.

Final Filter

In the initial REAL SCAN algorithm simulation, the subsystem which receives scene information from other system components and processes this information for screen display is named the Final Filter. The task of the Final Filter is to compensate for the unwanted display artifacts produced by the sampled data base points, which are projected to display screen pixel locations. The Final Filter will process the data base information such that the reflectance/color of the projected points is distributed on the display screen in a manner that results in an image which reproduces the necessary visual cues, thus appearing real to the eye.

In general, the Final Filter must provide data compression that is not only desirable from the economical and real time computation view points, but is dictated by the mathematics of the physical effects of human sight. Further, the proper filter must be able to distinguish between detail beyond the capability of the display and, hence, act as a low pass filter; yet, where visibility and perspective provide significant detail, the filter must accentuate the detail. The goal is to accomplish these ends using simple algorithms approximating the ideal case. Simplicity, based on knowledge and simulation results, allows speed and economy.

The Final Filter requires, as input from the REAL SCAN system, the screen coordinates and the reflectance/color information for each data base point projected to the screen. A reflectance/color buffer is used for the filter manipulations and eventual output of the filtered image to the display device. The filter development presented is for black and white image generation but each filter may be readily extended to color image generation. Therefore, the data base information required by the final filter will be the screen coordinates and the grey scale reflectance value of the sampled data base points. The output device for the filter analysis is a DICOMED D-47 Image Recorder which was used to produce all of the images on Polaroid film. Picture generation with the DICOMED is discussed in Section VI.

Display Defects

When two areas of varying color or intensity in a scene are adjacent, the resulting contrast between the areas produces a visual boundary which will appear as a line or a curve. The extent to which this boundary is evident to the eye is a function of the rate of change of the contrast with respect to the spacial dimensions (18). These boundaries are a fundamental component of image definition. To gain realism in a CIG display, these boundaries must be reproduced without inducing defects. Also, in order to display images that appear authentic, sufficiently detailed data base must exist. However, unwanted visual effects appear when displaying samples of such a data base on a finite element display device (19, 20, 21). The quantization effects of the display device along with the sampling error produced from accessing the data base are the cause of these effects, known as aliasing. Only high detail of spacial frequency less than one-half the sampling frequency can be reproduced without aliasing (22). The quantization of the display device limits the detail that can be reproduced since each pixel in the display can have only one color value. The effects of this quantization and sampling

---

[18]Crow, F. "The Use of Gray Scale for Improved Raster Display of Vectors and Characters", SIGGRAPH '80 Proceedings, pp. 286-293, July, 1980.

[19]Crow, F. "The Aliasing Problem in Computer Generated Shaded Images", Communications of the ACM, November, 1977.

[20]Catmull, E. "A Hidden-Surface Algorithm with Anti-Aliasing", Computer Graphics, pp. 6-10, 1979.

[21]Weiman, C. "Continuous Anti-Aliased Rotation and Zoom of Raster Images", SIGGRAPH '80 Proceedings, pp. 286-293, July, 1980.

[22]Oppenheim, A. V. and Schafer, R. Digital Signal Processing. Prentice Hall, Englewood Cliffs, New Jersey, 1975.

error are seen as jagged or "staircased" edges when trying to display a high spacial frequency contrast boundary. Also, low frequency patterns which are "aliases" of the high frequency data base will be present in the display (i.e., Moire fringes).

Aliasing effects can be reduced by two means, either increase the resolution of the display device along with the sampling frequency beyond the limits of the data base's resolution, or limit the spacial frequency of the data base information to be displayed below aliasing bounds. The first suggestion is limited by technology, cost and computational efficiency of the sampling algorithm. The alternate technique is to filter high resolution information to display resolution. The filtering is not easily defined, but two classes stand out:

1. Alias elimination requires that the spacial spectral content of the scene cannot exceed the display's capability. This requirement imposes range versus spacial frequency constraints.

2. Pixels with data from nonspacially related data base points (i.e. nonocculting, nonneighbors) must possess the "average" as seen by the eye. More accurately, the data must be prefiltered to compensate for pixel granularity and its effects on perception.

Final Filter Tests

In order to test the Final Filter algorithms and evaluate their effects on scene image display defects, a test data base is needed which will induce display defects. To induce aliasing in a display, a data base must contain information of spacial frequency on the order of and exceeding half the sampling spacial frequency. To create staircasing, straight-high-contrast boundaries at oblique angles to the display's pixel geometry must be present. A test data base was created and will be discussed in detail. This data base will be used to evaluate the effects of the Final Filter algorithms on these display defects, by generating a high resolution, 1024x1024 pixel, images of the test data base and comparing it to lower resolution images (i.e., 512x512 or fewer pixels). The comparison procedure will consist of subjective viewing of both the high resolution reference image and the low resolution filtered images.

The test data base consists of a set of radial lines converging to a point. The lines consist of alternating areas with differing grey contrasts that have constant angular widths. The point of line convergence is fixed at a display corner. This limits the display to ninety degrees of alternating lines. The contrast of the alternating areas is varied from a maximum (i.e., black and white adjacent areas) to zero (i.e., grey to grey adjacent areas) as a linear function over the ninety degrees of display angle. This contrast test has a further advantage of keeping the average intensity of any gross region constant. The straight contrast boundaries, which vary in angular

orientation to the display pixel orientation, induce the desired staircasing. By letting these lines converge to a point, the lines become smaller in width until they are beyond the resolution of the display. Therefore, attempting to display this high frequency information will induce alias patterns.

The radial line data base is generated in FORTRAN code by first dividing the ninety degrees of display into 128 areas of equal angular width. This specific angular width, given the FORTRAN variable name TEST1, was choosen since it produces clearly evident aliasing defects. The data base is sampled once for each display pixel. Each sample point has associated with it the angle between the line formed from the sample point to the corner of line convergence and the line formed by the display edge containing the corner of line convergence. This is the angle called TEST2. The number of the angular line is computed by an integer divide of TEST2 by TEST1. If this number is even, the reflectance value of the sample point will be between black and grey as determined by TEST2, where a TEST2 value of zero produces black and a TEST2 value of ninety degrees produces grey. If TEST2 divided by TEST1 is odd, the reflectance of the sample point will be between white and grey as determined by TEST2, where a TEST2 value of zero produces white and a TEST2 value of ninety degrees produces grey. The DEC FORTRAN IV PLUS code implementation of the radial line data base is given below. The variable IR represents the reflectance value of the sample point IX,IY. The display corner associated with IX=1 and IY=1 is the corner of radial line convergence.

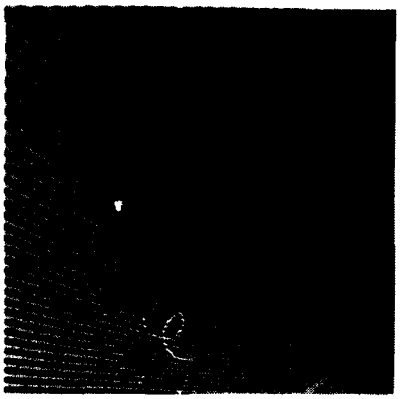
```
C
C
C-----------------------------------------------------------------
C
C          RADIAL LINE DATA BASE
C
         PI=3.14159
         TEST1=PI/256
         TEST2=ATAN2(IX-.99999,IY-.99999)
         K=TEST2/TEST1
         IF((K/2)*2.NE.K) GOTO 100
         IR=(-254./PI)*TEST2+255
         GOTO 200
100      IR=(254./PI)*TEST2
200      CONTINUE
C
C-----------------------------------------------------------------
C
C
```

Figure 37 shows the radial line data base computed at three resolutions using the routines of TSTBW.FOR, found in Appendix IA, and SRCAMERA.FOR, found in Appendix IB, as linked using

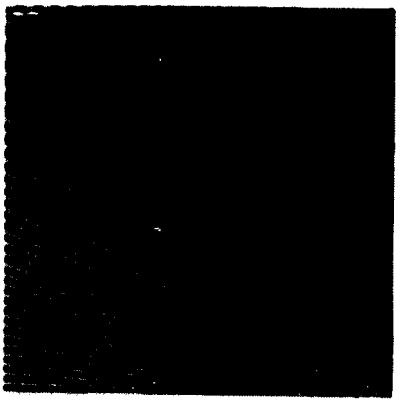
```
$LINK/EXE=TSTBW-
DRA0:[UTILITY.DICOMED]BLKDAT,DBA1:[P8734.SAM]SRCAMERA, -
TSTBW,DRA0:[UTILITY.DICOMED]DICOMED/LIB
```

Each pixel element contains one sample point from the data base.

37a
1024x1024 resolution



37b
512x512 resolution



37c
256x256 resolution

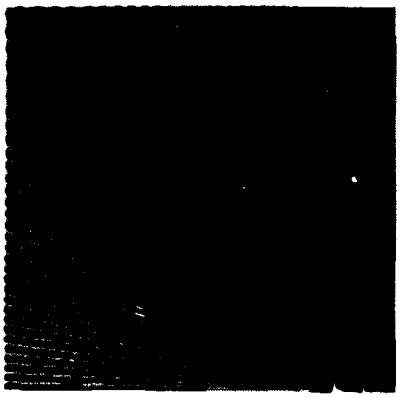


Figure 37. Radial Line Aliasing versus Contrast Ratio.

Filtering

For the initial image generation tests, it was desired to display the data projected from the sampled data base with minimal computation and manipulation of the raw data such that an idea of the REAL SCAN system's characteristics may be seen. Since the REAL SCAN simulation algorithms project varying numbers of data base points to each particular screen pixel, a reflectance value is computed as a function of all the projected data base points in each pixel in order to see graphically the system's characteristics.

The initial function used was to assign a pixel reflectance value equal to the sum of the reflectance values for all points that are projected to the same pixel divided by the number of points which contributed to the reflectance sum for that pixel. Therefore, the distribution of reflectance information from each projected data base is confined to only the particular pixel into which the point was projected.

Mathematically we can describe this as follows, if we define the screen pixel i,j, then

$$\text{REFLECTANCE FOR PIXEL } i,j = \frac{\sum_{n=1}^{N} \text{DATA BASE POINT } i,j,n}{N} \qquad (74)$$

where the system will project all the data base points from i,j,1 to i,j,N to pixel i,j. This algorithm is referred to as Filter 1.

The Filter 1 as described was developed into FORTRAN code which is compatible with the REAL SCAN algorithms previously developed. The Final Filter system requirements are:

INPUT:
- -Pixel Address in Screen Coordinates
- -Reflectance Value of Projected Data Base Point
- -Flag to determine when Scene Generation is Complete

OUTPUT:
- -Screen Buffer containing a Single Reflectance Value for each Pixel Element

INTERNAL:
- -Count Buffer to count the number of Data Dase Points projected to each Pixel

A flowchart of the algorithm is given in Figure 38.

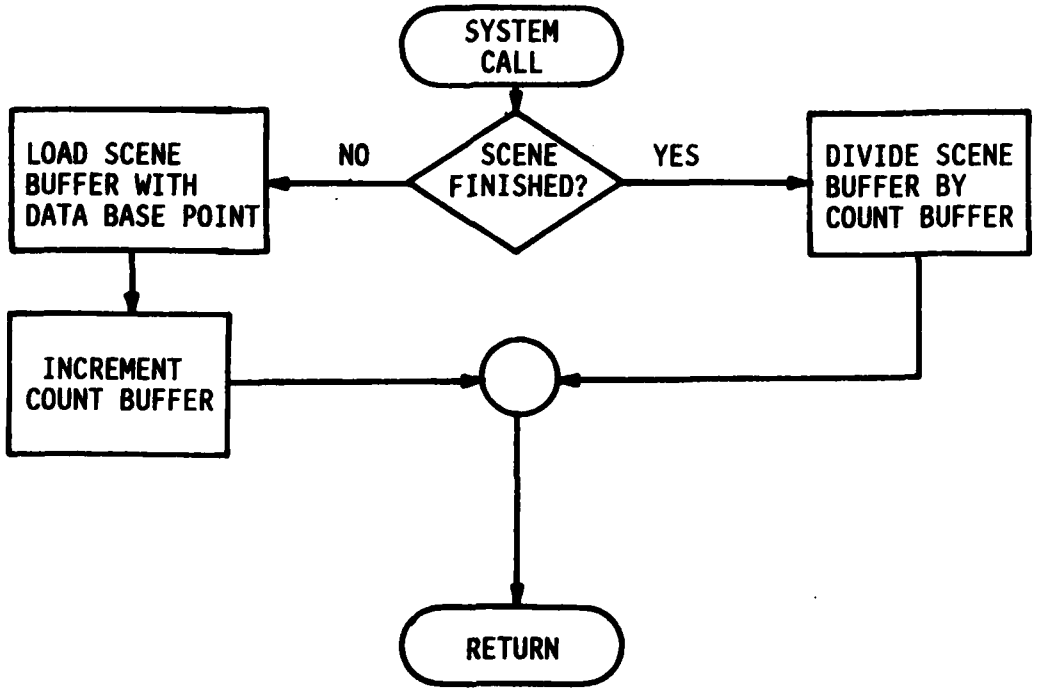


Figure 38. Flowchart of Filter 1.

The DEC FORTRAN IV PLUS code implementation of the flowchart of Figure 38 is given as follows for use with a 512x512 pixel display device.

```
C
C     INPUT VARIABLES/ARRAYS -
C         IXBS - screen horizontal coordinate
C         IYBS - screen vertical coordinate
C         IRBW - reflectance value from 0 to 255 for
C                pixel at IXBS,IYBS
C         IFINISHED - flag to designate when scene
C                     generation is complete
C
C     OUTPUT VARIABLES/ARRAYS -
C         ITEMBUF(512,512) - buffer containing the reflectance
C                            values for the screen scene image
C
```

```
C     INTERNAL VARIABLES/ARRAYS -
C         ICNT(512,512) - byte buffer containing the count of
C                         points projected to each screen pixel
C
C
C-----------------------------------------------------------------------
C
               IF(IFINISHED.EQ.1) GOTO 100
               ITEMBUF(IYBS,IXBS)=ITEMBUF(IYBS,IXBS) + IRBW
               ICNT(IYBS,IXBS)=ICNT(IYBS,IXBS) + 1
               RETURN
100            DO 200 J=1,512
               DO 200 I=1,512
                      KDIV=ICNT(I,J)
                      IF(ICNT(I,J).EQ.0) KDIV=1
                      ITEMBUF(I,J)=(ITEMBUF(I,J)+KDIV/2)/KDIV
200            CONTINUE
               RETURN
C
C-----------------------------------------------------------------------
C
C
```

The images shown in this report use the FORTRAN code from label "100" to label "200" listed for Filter 1. This code is contained in PGFINFIL.FOR listed in Appendix EE. The code listed above label "100" has been modified as described in the discussion of the Log Load Filter to follow.

Scenes were produced with fewer than one projected data point per typical pixel to subjectively evaluate the trade-off between display quality and scene calculation time. Those pixels with no data are called "missed" pixels. In order to assign reflectance values to the "missed" pixels, a complementary filter was developed which is referred to as Fill Filter.

Fill Filter uses the reflectances assigned to each pixel by Log Load Filter and computes a reflectance value for a "missed" pixel as a function of the "missed" pixels eight nearest neighbors. The assigned reflectance value for a "missed" pixel is determined by summing the reflectance values of the missed pixel's eight nearest neighbors who, themselves, are not "missed" and dividing by the number of pixels which contributed to the reflectance sum.

Mathematically this operation can be described as follows, for screen pixel i,j, then

$$\text{REFLECTANCE FOR PIXEL } i,j = \frac{\sum_{k=i-1}^{k=i+1} \sum_{m=j-1}^{m=j+1} \text{PIXEL } k,m}{N} \qquad (75)$$

where N is the number of pixels contributing non-zero values to the reflectance sum. Also, if pixel i,j is on a display boundary or corner, then the reflectance sum is computed from the pixel's five or three nearest neighbors respectively.

Fill Filter has the following requirements:

INPUT: -Dimensions of Display Device in Pixels
-Screen Reflectance Buffer Containing a Single Reflectance Value per Pixel
-Count Buffer of Number of Projected Data Base Point for each Pixel

OUTPUT: -Screen Reflectance Buffer containing a Single Reflectance Value per Pixel

The DEC FORTRAN IV PLUS code implementation of Fill Filter is given below for use with 512x512 pixel device.

```
C
C
C        INPUT VARIABLES/ARRAYS -
C          NLIN - vertical screen dimension in pixels
C          NELE - horizontal screen dimension in pixels
C          ITEMBUF(512,512) - buffer containing the
C                 reflectance values for each pixel
C          ICNT(512,512) - buffer containing the count
C                 of points projected to each pixel
C
C        OUTPUT VARIABLES/ARRAYS -
C          ITEMBUF(512,512) - buffer containing the
C                 reflectance values for each pixel
C
C
C
C
C-------------------------------------------------------------------
C
C        SCENE IS FILTERED TO FILL ANY MISSED PIXELS
C
         DO 111 IYBS=NLIN,1,-1
         DO 222 IXBS=NELE,1,-1
         IF(ICNT(IXBS,IYBS).NE.0) GOTO 220
```

```
          JCOUNT=0
          JSUM=0
          DO 300 K=IYBS-1,IYBS+1
                 IF((K.LT.1).OR.(K.GT.NLIN)) GOTO 300
          DO 440 L=IXBS-1,IXBS+1
                 IF((L.LT.1).OR.(L.GT.NELE)) GOTO 440
                 IF(ICNT(L,K).EQ.0) GO TO 440
                        JCOUNT=JCOUNT+1
                        JSUM=JSUM+ITMEBUF(L,K)
   440           CONTINUE
   300           CONTINUE
          ITEMBUF(IXBS,IYBS)=JSUM/JCOUNT
          ICNT(IXBS,IYBS)=-1
   220           CONTINUE
   222           CONTINUE
   111           CONTINUE
                ENDIF
C
C-------------------------------------------------------------------
C
C
C
```

The code listed above is contained in PGFINFIL.FOR found in Appendix EE.

The results of Fill Filter on images with varying densities of missed pixels can be seen in the data base images of Appendix M where Fill Filter is also referred to as Filter 2.

During initial tests, statistical analysis of the density of projected data base points showed that a wide variation occurred. This variation was a function of the viewing vector and data base characteristics such that, whenever the normal vector to the data base surface and the viewing vector approached ninety degrees, a peak in data density projected to a pixel would occur (i.e., looking at the edge of a sheet of paper). As a means of suppressing these peaks in density a filter algorithm compatible with Filter 1 and Fill Filter was developed, referred to as Log Load Filter, which permits only a logarithmically derived sampling of those projected points to a pixel to be allowed to actually contribute to scene generation. A count is kept on the number of projected points to each pixel from a single scan line. The reflectance and count buffers are only loaded with points that correspond to the Nth power of two projected points where N is an integer starting with zero. This lets only the first, second, fourth, eighth, etc., projected point contibute to the data base image. The Log Load Filter also limits the number of data base points projected to any one pixel to 126 total from all scan lines by testing the current pixel count before accumulating new points.

The Log Load Filter reduced the peak number of points projected to a pixel from above 100 to approximately 16 depending on the scene parameters. The following is the DEC FORTRAN IV PLUS code for the implementation of the Log Load Filter. The variables IXSO and IYSO must be initialized to zero as shown by the DATA statement below.

```
C
C
C-----------------------------------------------------------------------
C
      DATA IXSO,IYSO /0,0/
C
C-----------------------------------------------------------------------
C
C
C       INPUT VARIABLES/ARRAYS -
C               IXBS - horizontal screen coordinate
C               IYBS - vertical screen coordinate
C               IR - data base reflectance value
C
C       OUTPUT VARIABLES/ARRAYS - NONE
C
C
C-----------------------------------------------------------------------
C
        KCOUNT=KCOUNT+1
        IF((IXBS.NE.IXSO).OR.(IYBS.NE.IYSO)) THEN
                IPOWER=2
                KCOUNT=1
                IXSO=IXBS
                IYSO=IYBS
                GOTO 100
        ELSE IF(IPOWER.EQ.KCOUNT) THEN
                IPOWER=2*IPOWER
  100           CALL DATABASE
        IF(ICNT(IYBS,IXBS) .GT.126 ) RETURN
        IF((IXBS.GE.(IFILDIM + 1)).OR.(IXBS.LE.0)) THEN
                RETURN
        ELSE IF((IYBS.GE.(IFILDIM+1)).OR.(IYBS.LE.0))THEN
                RETURN
        ENDIF
        ITEMBUS(IYBS,IXBS)=ITEMBUS(IYBS,IXBS)+IR
        ICNT(IYBS,IXBS)=ICNT(IYBS,IXBS)+1
        ENDIF
C
C-----------------------------------------------------------------------
C
C
```

The code listed above is contained in PGLOGLOAD.FOR found in Appendix EF. The three dimensional images using the REAL SCAN algorithms shown in this report use the Log Load Filter.

SECTION V

REAL SCAN ARCHITECTURE

This section develops estimates of the processing requirements for a high detail, real time, computer image generation system based upon **REAL SCAN** algorithms. Further those processing requirements, which are sufficiently well defined at the present stage of algorithm development, are carried to a feasible architecture. Figure 39 illustrates the computation sequence and the buffer memories needed to ensure smooth processing. We assume a large disk store containing the data base which describes the gaming area. The analysis of a hierarchial data base carried out in Appendix B indicates that a gaming area of 50 km x 50 km will require a disk memory on the order of $5\ 10^{11}$ bytes of disk. Video disk technology suggests this requirement can be met.

REAL SCAN computes the current eye location and orientation which defines the viewing window. Further, **REAL SCAN** makes a prediction based upon the present motion vector parameters (velocity, acceleration, control surfaces, etc.) of the eye locaton and orientation during the next 0.1 sec to 1.0 sec. This prediction allows one to down load the potential data base parameters into a high speed cell memory. The cell memory is a high speed virtual memory which accepts actual ground coordinates in the gaming area at the various hierarchial levels, to acquire the appropriate data base parameters from physical locations in the cell memory. The data in the cell memory consists of altitude or elevation information/parameters, color/reflectance and texture code parameters.

Figure 39 indicates the cell memory is first accessed to determine the elevation of a point along a scan line. This elevation/height at a world coordinate is used to determine if the point is visible or occulted by some nearer point. If the point is visible, then Figure 39 indicates that the pixel coordinates to which this visible point would project are calculated, if this is a power of two point (i.e., first, second, fourth, eighth, sixteenth, ... 2^n th), then the second set of cell parameters are used to calculate the color and intensity of the visible point.

The visible point's color is then conceptually projected to the display screen. The physical operation is shown in Figure 39 as an accumulation of color and intensity data into a ping-pong frame buffer addressed according to the display screen's coordinates. In the current system, the final filter is the averaged accumulation. However, **REAL SCAN** anticipates the use of a point spread filter function to allow for weighted averages of the visible points projected to the screen and a high pass filter edge accentuator - low pass filter edge

blurrer. The high pass - low pass test is based upon the rate of color change along a scan line. If data variation occurs at a rate exceeding the kell factor display limit (i.e., 3 pixels to display a line pair) then the data is low pass filtered. However, if data variations along a scan line occur at less than the kell factor display limit, then edges are enhanced.

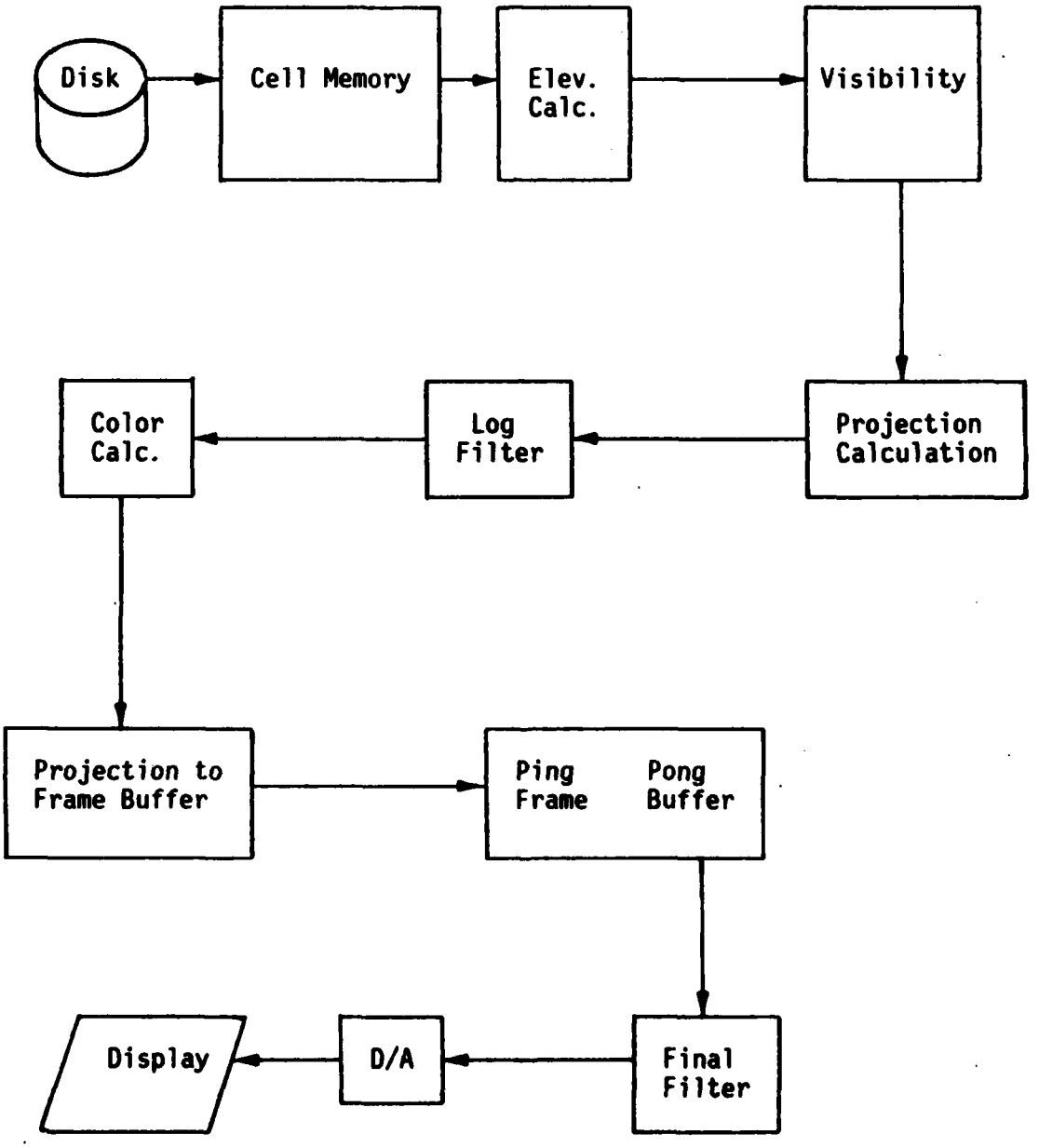


Figure 39. REAL SCAN Block Diagram.

Processing/hardware estimates for REAL SCAN evolve into two areas:

1. The data rates between the blocks of Figure 39.

2. The numbers of instructions required to perform a processing function (i.e., calculate elevation, determine visibility, project, calculate color, etc.)

System Parameters

REAL SCAN assumes that multiple world points will be calculated for each display pixel. The current algorithms are set for about four world points projected to one pixel. Further testing is needed to establish over sampling and filtering requirements.

A 512 x 512 display has been used to typify a viewing window/ screen. Clearly, the higher the detail required, the more complex the hardware. REAL SCAN assumes display limited resolution will be calculated from the real world data base. Hence, if two systems were to be compared, one having twice the number of pixels as another, with both operating at the same computed display rate; then the system displaying twice the data would requre about twice the hardware.

Let us consider a CIG system having the following system specifications:

1. Display of 512 x 512 pixels per screen.

2. Display rate of 60 full scenes of 512 x 512 pixels per second.

3. Computation rate of about 4 world points per display pixel and an upper bound of 8 world points per pixel along a scan line.

These specifications further imply a REAL SCAN system having about 2 scan lines passing through each pixel. Hence, an upper bound between 16 and 32 exists for the number of world points projected to a pixel, dependent upon screen orientation.

One can estimate the following system characteristics based upon the system specifications:

1. The average data rate of points passing to the screen/ display frame buffer is 512 x 512 [pixels per display] *4 [points per pixel] *60 [displays per sec] or 60 mega points per second. Each point consists of 3 bytes of color.

2. The worst case computation rate at either the elevation or visibility processor is 1,024 [full scan lines per display]* 10 [hierarchies] *1,024 [world points per hierarchy] * 60 [displays per sec] or 600 mega points per sec.

3. The worst case computation rate at the color/intensity processor is 1,024 [full scan lines per display] * [512 x 512 pixels per scan line] * 8 [world points per pixel per scan line] * 60 [displays per sec] or 240 mega points per sec.

4. The average computation rate at the color/intensity processor is about 1/4 the worst case rate or 60 mega points per sec since 2 world points per pixel per scan line is the system goal.

5. The worst case projection processor computation would correspond to the 600 mega points per sec of the visibility processor if all points are visible. However, the need for REAL SCAN assumes a high resolution-high detail scene. Hence, one anticipates substantial occulting with the use of local slope over a calculation grid for allowing larger intervals between "safe" computations along a scan line as one samples the space far from the eye.

The system characteristics have been presented in terms of REAL SCAN however, they define bounds for any CIG system attempting to provide high resolution imagery according to the specification.

The REAL SCAN system of Figure 39 does not partition along scan lines. Hence, the largest degree of parallelism which one can achieve is about 1,024 parallel pipelined processors (i.e., one pipe line per scan line). Appendix C discusses the use of intermediate projection planes, which allow parallel processing for one scan line on the order of the number of intermediate planes. The hardware estimates in this section are based on Figure 39, where a scan line defines the longest train of sequential steps.

The shortest time interval for calculating one point along a worst case scan line is 600 * 10^6 [points per sec per display] ÷ 1,024 [scan lines per display] or 1.6 microsec per point. This corresponds to 1/10 the interval needed to create the display specification of 60 mega points per sec. Since microcomputer instruction times are currently longer than 800 nanosec, the processing pipe line cannot be made, in 1982, from micro computers such as 8080, 6800, 68000 or 8086. Microprocessors (i.e., bit slice arrays) currently operate at micro programmed instruction times of 100 nanosec (i.e., AMD 29xx family). Hence, it is clear that the pipe lined processing segments which make up one scan line will be made from bit slice or faster technology. Efficient use of bit slice technology suggests that each processor in

the pipe line will execute a short program. A time interval of between 1.6 microsec to 16 microsec guarantees short programs.

A worst case estimate of the number of small and dedicated 100 nanosec processors can be made as follows:

1. The worst case data rate is 600 mega points per sec.

2. An estimate of the number of optimized bit slice instructions needed to perform REAL SCAN is 500 instructions per point (i.e., between 250 and 1000 instructions per point).

3. The number of processors is therefore 600 * 10^6 [points per sec] * 10^{-7} [sec per instruction] * 500 [instruction per point] or 3 x 10^4 processors. The processors could fit in 20 cabinets each holding about 70 boards if about 7 processors can be placed on one 12 inch x 12 inch board. Clearly more work remains to both reduce the number of points which need to be processed and the number of instructions per point. The biggest computational burden here resides in determining elevation, color and intensity for a complex display limited scene.

The detail of the pipelined processors will not be considered further since this detail requires a relatively fixed but necessarily optimized REAL SCAN algorithm. REAL SCAN is evolving in a number of ways, all aimed at achieving the specified ultimate computation rate of 60 mega points per sec. However, the detail of the Frame Buffer, or of the general bus controller structure directing data from the processing string to the Frame Buffer can be more nearly specified for an architecture. Hence, a preliminary architecture for the Frame Buffer and the processor to Frame Buffer control is developed next.

The cell memory is not developed further, since it is a virtual memory system and virtual memory systems are available. Further, since an intermediate plane memory is recommended as an improvement over the REAL SCAN system of Figure 39, it is clear that the cell memory requirements and data transfer speeds are likely to change substantially. This also obviates the need for carrying the cell memory architecture further than estimating its size as (i.e., Appendix A) and defining its addressing structure as virtual.

Frame Buffer and Bus Control

Figure 40 illustrates the interface between the Frame Buffer and the processors. A ping-pong Frame Buffer consists of two buffers. Each buffer contains a full display frame (i.e., one scene). While the ping side is accumulating one picture via scan line coordinates,

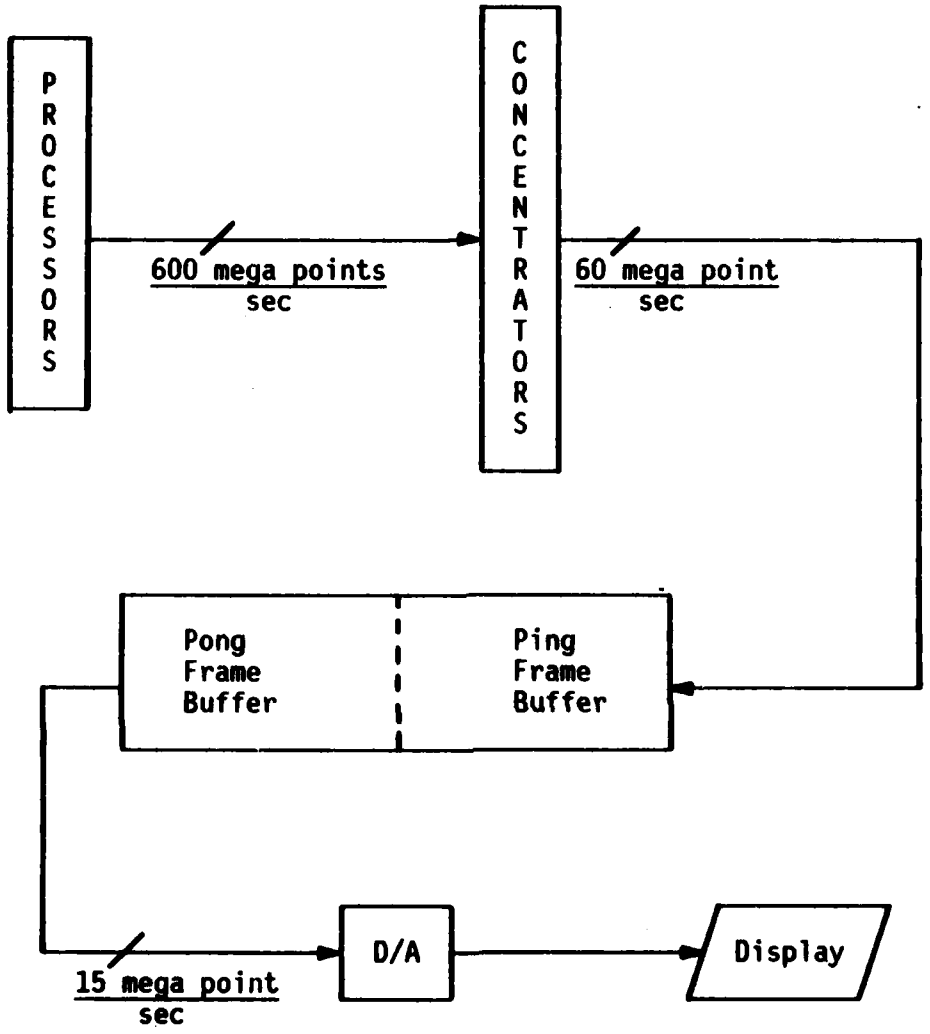


Figure 40. Data Rates from the Processors to the Display.

the pong side is displaying a picture via screen master scan coordinates. Hence, the function of the ping and pong memories switch once per frame cycle. Clearly this operation is most easily accomplished if the memory I C's have separate data input and data output terminals. The data rates between these elements is also shown on Figure 40.

The architecture developed below assumes some form of concentration, such that the data rate from the concentrator does not exceed about 2 times the specification of 60 mega points per sec. Then an architecture for accomplishing the routing of pixel data from the concentrator to the addressed cell of the Frame Buffer is developed.

A modification to this Frame Buffer architecture is presented at the end of this section. This modification allows for partial data accumulation to pixel locations to be performed at the Concentrators. Concentrators would be housed in cabinets with the Processors. Hence, the inter-cabinet Concentrator to Frame Buffer bus speed requirements are reduced by the modification.

Once the number of pages of Frame Buffer memory has been determined, an architecture is developed for performing the final filter function on the output of the Frame Buffer.

ARCHITECTURE

If one allows a budget of 4 microsec. to calculate one point for one Concentrator pipe line feeding data to the Frame Buffer; then, using a data rate of 128 mega points per sec., as the architecture design goal from the Concentrator to the Frame Buffer, yields 512 Concentrator pipe lines. Since the Frame Buffer consists of two full displays, each 512 x 512 pixels; each pixel having no more than 14 bits per color and no more than 6 bits to hold the accumulated count of points per pixel, then the Frame Buffer memory size is less than 1.6 mega byte.

Let us assume that one cabinet will accommodate the memory routing, Frame Buffer, and D/A converters. Since multiple cabinets will be required to house the processors, it is clear that a signal bus between cabinets will be required to pass data from the Concentrators to the Frame Buffers. Table 8 indicates the maximum number of wires needed to communicate 60 mega points of data to the Frame Buffer Router at a rate of 128 mega points per sec.

TABLE 8. CONCENTRATOR TO FRAME BUFFER BUS PARAMETERS

(Data Only)	Bus Dimension (Data and Address)	Data Interval on Bus nanosec
310	540	100
460	810	150
620	1080	200
770	1350	250
930	1620	300

Table 8 assumes 24 bits of data and a negligible number of control lines (i.e., less than 3 control lines) are needed on a data only bus. If the bus must pass the address each time rather than more simply encode the address increments on the control lines, then 18 bits of address is also required per point. Table 8 indicates that one can budget about 800 lines to the Concentrator - Frame Buffer bus, or eight 100-pair connections.

The number of pages of memory can be estimated for various speed memories based upon 512 x 512 pixels accumulating at a rate of 128 mega points per sec. Table 9 illustrates various memory partitions as a function of the sum of memory read, memory write, and accumulate time, which is termed the Frame Buffer cycle time.

TABLE 9. PAGES OF MEMORY VERSUS FRAME BUFFER CYCLE TIME

Memory Pages	Words per Page *1024	Frame Buffer Cycle Time nanosec	Year when Inexpensive
26	10	200*	1984
38	7	300	
51	5	400	1982
64	4	500	
76	3.3	600	
90	2.8	700	
102	2.5	800	1980
128	2	1000	

*The recently announced (23) 2^k*4 static RAM TMS2149 operates at a maximum read time of 35 nanosec. The TMS2149 costs $8.45 in lots of 100.

If T is the Frame Buffer cycle time and n is the number of pages, then 128 mega points/sec = n/T. Memory cycle times appear to be decreasing by a factor of 2 about every two years. In the 1979-1980 time frame inexpensive memory had a cycle time of 300 to 800 nanosec. One anticipates inexpensive memory in the 1982-1983 time frame will have cycle times of 100 to 400 nanosec. The routing control and Frame Buffer architecture will be developed for 64 memory pages having 4^k pixels per page. Each page of memory will be able to accumulate a data point back into a pixel location in less than 500 nanosec.

Figure 41 illustrates the Concentrator to routing and accumulating side of the Frame Buffer. To most easily develop and explain the architecture, it will be assumed that both address and data are passed per point to the router.

---

[23]"Electronic Products" Vol. 24, No. 10, Hearst Business Communications, Inc., Garden City, New York, p. 80, January, 1982.

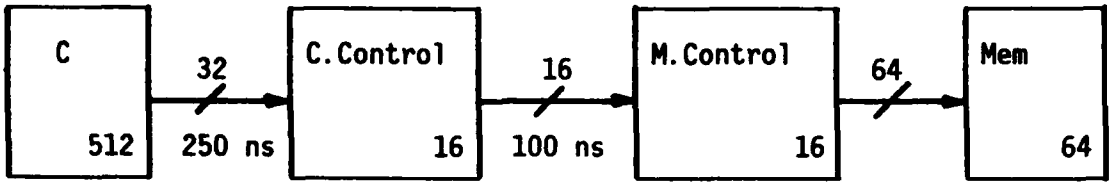


Figure 41. Block Diagram of Concentrator, Routing and Frame Buffer.

The following symbology is used in Figure 41:

1. C is a Concentrator. There are 512 Concentrators indicated in Figure 41 operating at 4 microsec per point. The bus structure indicates 16 Concentrators feed one C.Control bus.

2. C.Control (Concentrator Controller) is a router taking 32 Concentrator's data (i.e., potentially 64 scan lines or a strip of about 32 pixels wide) and decoding the 4 most significant address bits to identify one of 16 page groups of memory. There are 16 C.Controllers indicated in Figure 41.

3. M.Control is a memory controller which accesses each C.Control for data on a single bus shared among the 16 C.Controlers. There are 16 M.Controllers indicated in Figure 41.

4. Mem is the actual Frame Buffer. There are 4^k pixels per page at 14 bits per each of the three colors and one 6 bit counter to record the number of points accumulated into a pixel. Each memory controller services 4 pages of memory. Figure 41 indicates a total of 64 pages of memory.

Figure 42 illustrates the architecture for a Memory Controller with its four pages of memory. There are 16 buffers shown, one between each Concentrator Controller.

Each Concentrator Controller has one output bus which enters every Memory Controller's buffer 1. It is assumed that the Concentrator Controller is in the Memory Buffer cabinet. Hence, the 32 busses shown in Figure 41 connecting the Concentrators, go between cabinets and are budgeted at 250 nanosec per word, whereas the 16 busses between the Memory Controllers are budgeted at 100 nanosec per word. The Memory Controller is budgeted to accumulate data into memory at 350/4 nanosec or 87.5 nanosec Since all pipe lines are budgeted to operate on data at a rate slightly faster than the pipe line accepts the data, the elimination of bottlenecks is guaranteed for Figure 41. Figure 42 indicates that the color adder is budgeted with 350/4

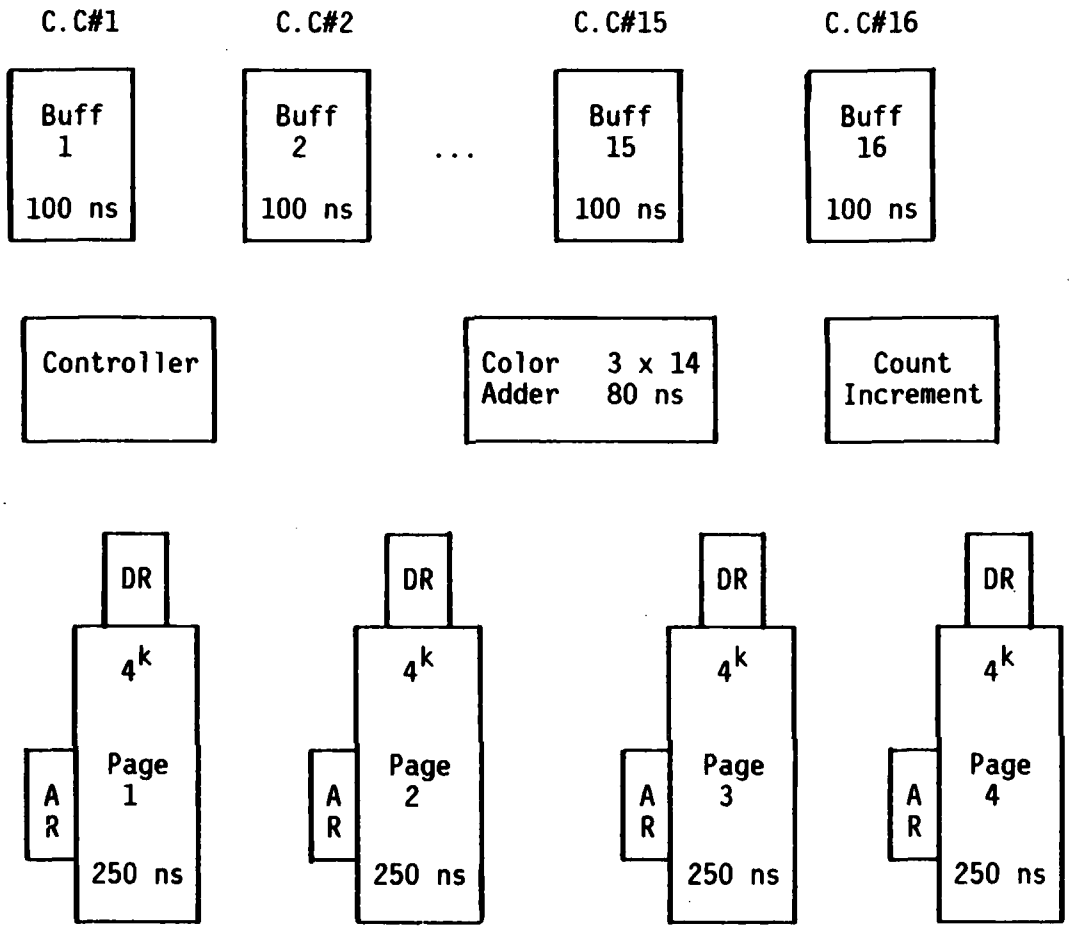


Figure 42. Block Diagram of the Memory Controllers

nanosec of accumulation delay. An adder producing its 14 bit sum in 87 nanosec or less can be shared without contention between about 4 pages of memory.

The controller of Figure 42 performs the following functions:

1. Check 16 buffers at one time using a priority decode such that the oldest buffer has the highest priority.

2. Accumulate the contents from highest priority buffer into its pixel on the free page in memory. There must be at least one free page of the four available. However, it is possible that all buffers currently have their ready word targeted to an active page in memory. Even if all buffers initially point to the same page in memory, after 4 words have been accumulated from 4 buffers, all the memories

should be cycling among the buffers and accumulating pixels. An average data rate is guaranteed by addressing contigous screen pixels to one of the 64 pages in memory. Figure 43 illustrates a page numbering scheme for placing geometrically near pixels in unique memory pages. All pixels labeled 1 are on page 1. All same page pixels have at least an 8 pixel space separation from a geometric neighbor also targeted for page 1. Pages 1, 2, 3, and 4 are controlled by the same Memory Controller.

				63				64	
14	1	5	9	13	2	6	10	14	1
	17	21	25		18	22		30	
	33				34	38		48	
62	49	53	57	61	50	54	58	62	
16	3	7	11	15	4	8	12	16	3
	19			31	20	24	28	32	
	35	39		47	36	40		48	
64	51	55	59	63	52	56	60	64	
14	1			13	2				

Figure 43. A Mapping of Memory Pages to Display Space.

A two-step controller is best hardwired. It is clear that the function of accumulation will not change (i.e., Mem(Addrs)←Mem (Addrs) + Buffer Data; CNT(Addrs)←CNT (Addrs) + 1).

One final comment about Figure 42, the 16 Buffs are assumed to be FIFO's (i.e., First-In-First-Out devices). Hence, they automatically load data from a C.Control bus when the data available flag says "load".

The buffers, Buff 1 and Buff 2, need be no more than dual registers which ping-pong between loading data from a Concentrator group and passing the data to the FIFO buffer, OUT. The Buff's serve one of the functions performed by FIFO's, to allow asynchronous operation between stages.

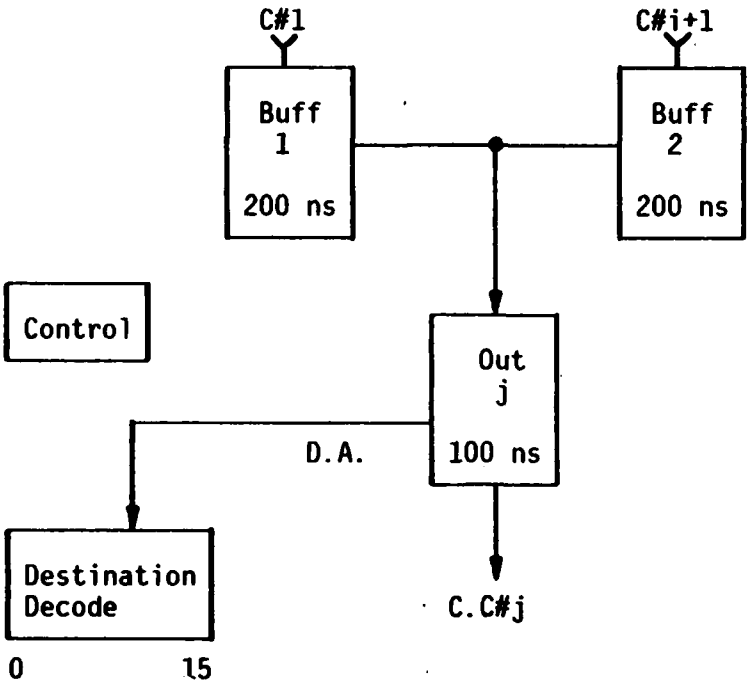


Figure 44. Block Diagram of Concentrator Control of Figure 41.

Figure 44 illustrates one of the C.Control elements of Figure 41.

The control in Figure 44 alternates access to the OUT buffer between Buff 1 and Buff 2, and commands OUT to load if the accessed Buff has data available. This architecture assumes each Buff has tri-state outputs. Hence, the control function can be placed in the following algorithm:

1. Deactivate Buff 2 tri-state drivers; activate Buff 1 tri-state drivers; determine if Buff 1 has data.

2. Load data to OUT if Buff 1 has data.

3. Deactivate Buff 1 tri-state drivers; activate Buff 2 tri-state drivers; determine if Buff 2 has data.

4. Load data to OUT if Buff 2 has data; go to step 1.

The OUT buffer output is connected to one port of each M.Control, as shown in Figure 42. Figure 44 indicates that the address bits (e.g. each of the 2 least significant row and column pixel locators) are decoded to identify which M.Control will receive the data. This decode is activated if the data available (D.A.) line of OUT is activated.

The architecture of Figure 44 allows for asynchronous operation between Concentrator groups and Memory Controllers. The architecture is simple and most reasonably would be hardwired due to the 100 nanosec. transfer requirement.

At this point, it is appropriate to review the architecture of Figure 41 and investigate the reordering of at least part of the accumulation function. If all accumulation could be accomplished at the Concentrators of Figure 40, then the data rate into the Frame Buffer would be less than 16 mega points per sec, a savings, by a factor of 8, over the architecture just presented since a 128 mega point per sec rate was utilized to account for bursts of data. Let us investigate an architecture for accumulating data at a Concentrator, buffering these accumulations and then sequentially passing them to the Frame Buffer.

ALTERNATE CONCENTRATOR ARCHITECTURE

In order to accumulate scan line data at a Concentrator in a reasonably small buffer memory, one must accomplish the following:

1. The scan lines must be grouped to a Concentrator according to the geometric region being calculated, to facilitate *accumulation.*

2. A simple address decoding scheme must exist to transform a strip swept across the screen into buffer addresses, to facilitate accumulation with a buffer memory which is but a fraction of the display size.

If a Concentrator accumulation stage is introduced, then the rate of data concentration at a concentrator of Figure 41 will not change, however the rate at which accumulated pixel information will be passed to the C.Control will decrease by about a factor of 8 (i.e., note however, that the width of the bus from C to C.Control will increase). Hence, by decreasing the number of busses from 32 to 16, the data rate shown in Figure 41 will decrease from a word every 250 nanosec to a word every 1 microsec on the average. Further, since picture data would be accumulated at a concentrator, the order which picture data will be placed on the inter-cabinet bus can be controlled, such that all memory planes are active (i.e., the average data rate is maintained without substantial contention of data destined for any one page of memory and hence the FIFO's of C.Control and M.Control can be eliminated). All accumulation will not be possible at the concentrators, unless they share data. Since a frame buffer is required to hold a full scene, interconnection between the concentrators is not warranted.

Figure 45 illustrates an architecture which satisfies all the requirements. Eight scan line concentrators servicing a connected

region of the world are accumulated to a simply transformed memory. As the scan lines pass over a region of the scene and start projecting information to other screen pixels, the data accumulated into "old" pixels can be routed toward the Frame Buffer. Figure 45 allows one accumulated data for one pixel to be routed forward every 3000 nanosec. The last stage is a double buffered accumulator allowing pixel data from any of the four accumulation memories to be routed to a C.Control, if from the interior of its wedge; or to be accumulated with its neighbor, if from a wedge boundary.

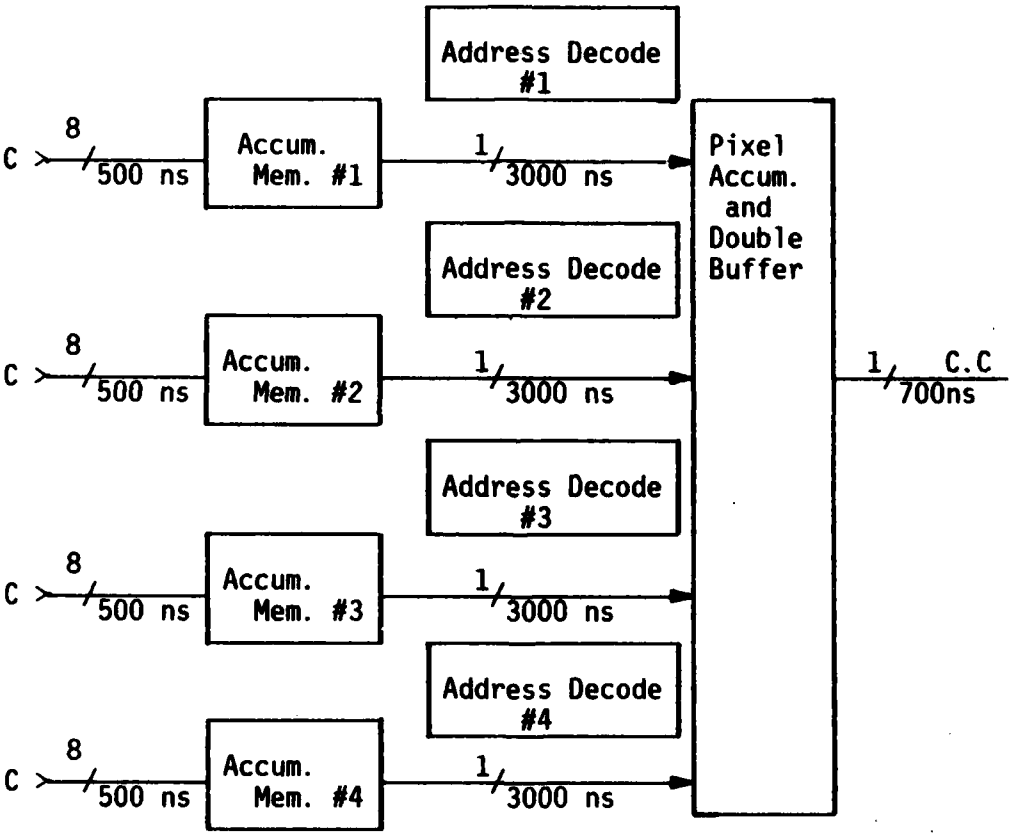


Figure 45. Concentrator Accumulation Architecture

The address decode portion of Figure 45 can be simply accomplished defining two axes:

1. Screen axes Xs, Ys

2. Accumulation memory axes X_A, Y_A.

Pixel data is computed in terms of Xs,Ys but stored in the accumulation memory of Figure 45 in terms of X_A, Y_A. One of the screen axes

will vary more rapidly, call this the ICASE axis. If Xs is the ICASE axis let X_A = MOD(Mx,Xs), where Mx is the address length along the X axis of the memory. Modular arithmetic is suggested for the ICASE axis since accumulated data is routed forward as the scan line moves to succeeding screen pixels. Let the axis associated with the more slowly varying coordinate change be called IOTHR. Then if Ys is the IOTHR axis Y_A = MOD(My,Ys - αXs), where α corresponds to the slope, ΔYs/ΔXs, of a wedge.

Figure 46 illustrates an address decode from physical space to Concentrator accumulation memory space. The ICASE axis corresponds to Xs. The slope is nearly 45°.

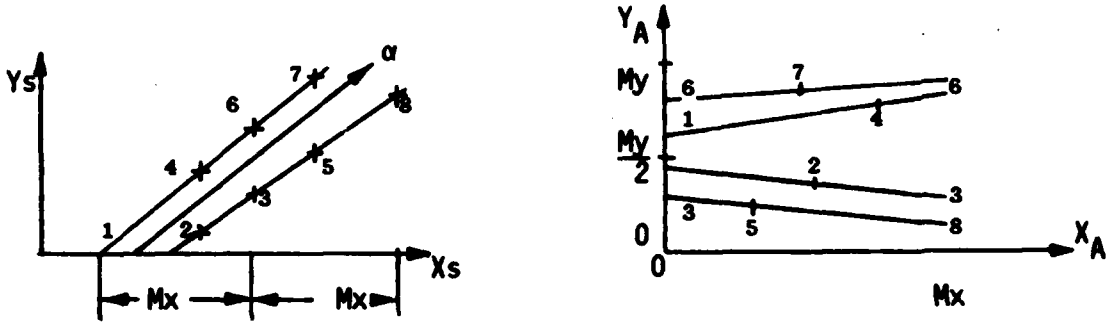


Figure 46. Memory Map of Screen Coordinates to Accumulation Addresses.

The address decode for translating the Concentrator accumulation memory back into screen coordinates can be accomplished by maintaining two base registers. These base registers can be either at the Double Buffer of Figure 45 or at C.Control of Figure 41. The conversion is

$$Xs = X_A + Nx \tag{76}$$

$$Ys = Y_A + \alpha Xs + Ny \tag{77}$$

where Nx, Ny corresponds to an initialization offset plus an integer number of memory dimensions Mx and My respectively. Figure 46 illustrates that line α in screen coordinates maps to the line My/2 in accumulator space. My need be no larger than Mx according to Figure 46. My must be at least as large as the number of pixels swept by a group plus about 4 to allow for address decode round off. However, Mx = My = M allows for shifting between ICASE and IOTHR axes with minimum effort. For example, a 1^K memory allows M = 32 which provides at least 4 group widths for a screen wedge.

The addressing scheme for unloading the Double Buffer of Figure 45 is simplest if the ICASE axis coordinate is held constant with the IOTHR axis varying. For example, if Xs is the fast axis, then unloading would be accomplished by $X_S = X_A + N_A$ and holding X_A constant until a full segment of $Ys = Y_A + \alpha Xs + Ny$ pixels had been unloaded to the next stage. Unloading of data would be associated with clearing the count for data points for that pixel to zero, as well as clearing the three intensities associated with the pixel.

The Accumulation Memory of Figure 45 needs to have a speed of about 500·300/3500 = 428 nanosec. The maximum word length is 14 bits per color plus 6 bits for count or 6 bytes. A memory speed for read, accumulate, write of better than 400 ns would allow the use of one high speed accumulation buffer at 1^k words of 6 bytes per word. This high speed memory would be arranged with 32 address per axis. However, less than 24 address would be needed for the IOTHR axis, if one uses 2 scan lines per pixel.

The general structure of Figure 41 is not changed by allowing concentration accumulation, however the routing complexity is reduced. Figure 47 illustrates the impact of concentrator accumulation on the routing of data to the Frame Buffer.

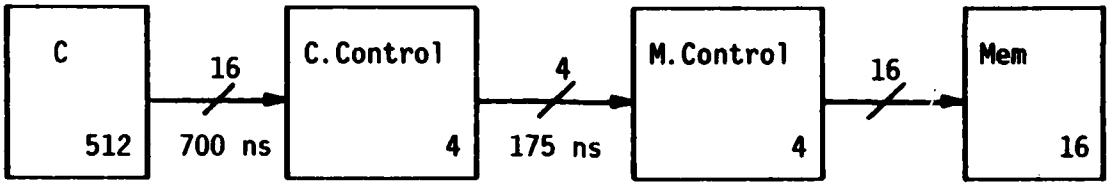


Figure 47. Block Diagram of Routing and Frame Buffer Architecture with Concentrator Accumulation.

The reduced data rates desired have been achieved in Figure 47. Further, the routers (i.e., M.Control and C.Control) have the number of interconnecting busses reduced, but the bus width has been increased. The data bus for one point in Figure 41 was 24 bits. The data bus in Figure 47 is 3*14 + 6 = 48 bits. Hence, the apparent factor of 8 reduction in data rate (i.e., words per sec) is actually only a factor of 4 (i.e., bits per sec). Some slight improvement is possible if the count (i.e., number of data points contributing to a pixel) is limited to 4 bits of representation. That is all colors are divided by count and multiplied by the largest allowed count value (i.e., one may encode the transmission of count such that 0 means one data point and 15 means 16 data points). The rescaled color and the maximum count would be routed to the frame buffer. The width of the data bus in the case count is bounded by 16, is 3*12 + 4 = 40 bits.

PONG FRAME BUFFER ARCHITECTURE

The last element to be described for the REAL SCAN architecture is the Display side of the Frame Buffer. The Display side of the Frame Buffer is the "Pong" memory of Figure 39 plus the Final Filter. The Final Filter takes the accumulated color per pixel and divides this accumulation by the count (i.e., the number of data points which make up the accumulated color). In this way, edge blurring and staircasing is at least partially eliminated from the displayed scene. Figure 48 illustrates the Display side of the Frame Buffer.

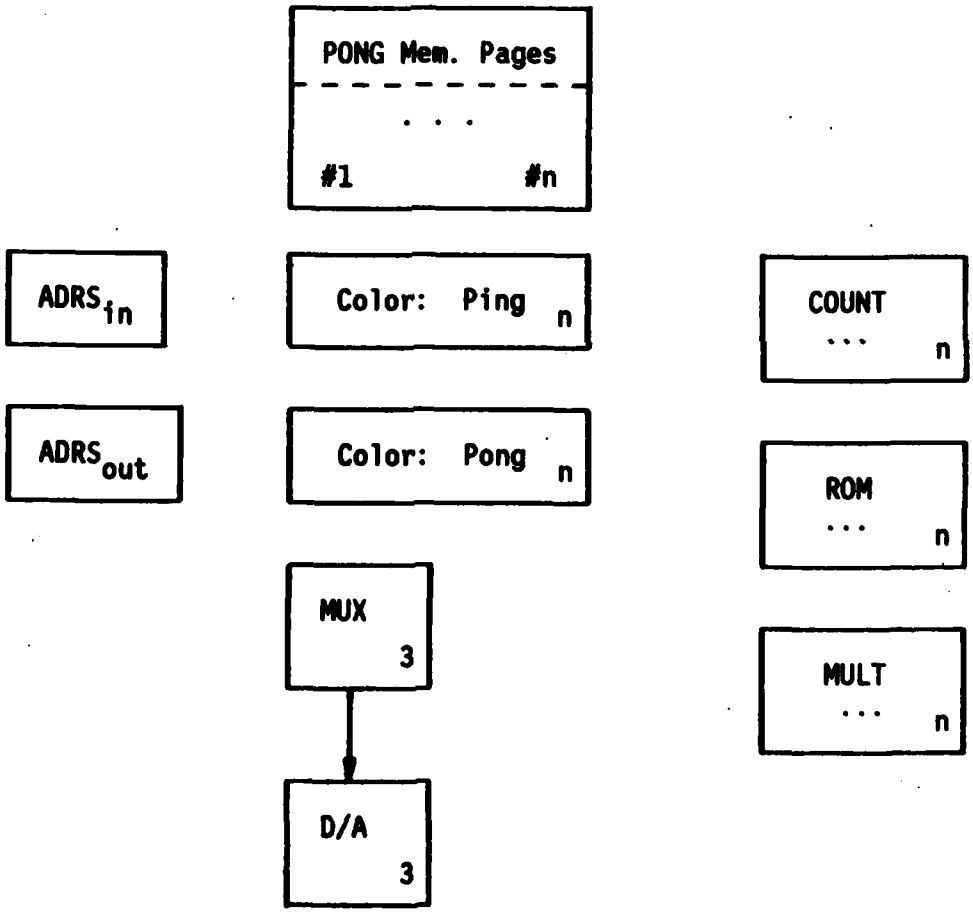


Figure 48. Block Diagram of Display Side of the Frame Buffer.

The Frame Buffer memory is arranged in pages. Figure 48 indicates that "n" pages of the "pong" side of the Frame Buffer are read in parallel into the ping color register and corresponding count register. There is one color location and one pixel count for each of the

pages of memory written to the color/count registers. Each 6 bit count value is inverted by a high speed ROM and then the product is formed of each of the 3 colors (red, green, blue) by using three of the Mult's per pixel. This product (i.e., one word per color) corresponds to the average color to be diplayed at a pixel. The product is loaded back into the "ping" color register.

During the time the ROM is inverting count and the average color is being formed, the "pong" Frame Buffer locations just read are being cleared to zero. Clearing the count and color per pixel location at this time allows proper accumulation during the ping cycle, and can be accomplished without further burden on Frame Buffer access time.

Current thinking utilizes 8 bits per color since color tests, which we have performed, indicate that discernable Mach banding occurs at somewhere between 6 and 7 bit resolution for display intensities available on the DICOMED. If one were to use REAL SCAN with a device capable of four times the DICOMED's apparent display intensity, then one would need about a 10 bit D/A result.

The averaged, and hence correct, pixel color is multiplexed via the MUX to the 3 high speed D/A converters for driving the RGB display. The availability of tristate registers eliminates the need for a physical mutiplexer, since the "pong" register output can be selectively enabled.

The number of pages of "pong" memory which need to be averaged in parallel can be calculated as follows:

1. Determine the pixel interval

2. Determine the averaging calculation time (i.e., sum of "pong" memory fetch, ROM read, multiply, and "ping" register write)

3. The minimum number of memory pages is the integer greater than the ratio of average calculation time and pixel interval.

For example, if the pixel interval is 100 nanosec and the pong memory fetch is 200 nanosec, the ROM read is 100 nanosec, the multiply is 150 nanosec, and the "ping" register write is 10 nanosec; then the number of Frame Buffer pages is 5. Figure 47 indicates that loading the Frame Buffer places a requirement of at least 16 pages. Hence, 16 pages of memory, operating with the parallel function speeds above, could present a new pixel every 28 nanosec.

The architecture presented in this section demonstrates the feasibility of the Frame Buffer concept to create high detail displays. However, work still remains to define an optimum REAL SCAN

algorithm for calculating the data base in real time. When an optimum data base calculation algorithm is determined, then the detailed processor and concentrator structure should be determined. The Frame Buffer places a burden of only one display time delay on the pipelined process. Hence, if the pipelined processors have negligible pipeline delay and they can load the "ping" of memory in one display time, then REAL SCAN potentially has only two display time delays between identifying the eye location and creating the last display pixel (i.e., one to load the "ping" memory and one to display the "pong" memory).

SECTION VI

PICTURE GENERATION SOFTWARE

The purpose of this section is to document four routines that are used in conjunction with the DICOMED D47 Image Recorder (DICOMED) and our Image Parameter Text Generation routines. The four DICOMED routines are entitled:

1. DICBWSA.FOR (DICOMED Black and White Stand Alone routine)
2. DICCOLSA.FOR (DICOMED Color Stand Alone routine)
3. SDICBW.FOR (Subroutine DICOMED Black and White)
4. SDICCOL.FOR (Subroutine DICOMED Color)

DICBWSA.FOR and DICCOLSA.FOR are to be used with previously generated data that is stored on disk or magnetic tape (mag tape). SDICBW.FOR and SDICCOL.FOR are subroutines that may be incorporated into data generation routines for immediate output to the DICOMED. These routines are in the P8734 directory of NTEC's VAX-11/780. Related topics also included in this section deal with the DICOMED itself and the DICOMED driver routines written by Jeff Rubin prior to 1979. Also included are techniques for using options available on the DICOMED.

This section assumes some knowledge of FORTRAN code and VAX-11/780 command language. Equations are given in FORTRAN code and FORTRAN program listings are included. Several VAX-11/780 command language procedures are included. This section updates a former P8734 DICOMED report dated 14 December 1979 which describes limited black and white use of the DICOMED. It is recommended that the reader read:

IMAGE RECORDERS

Models D46 and D47

Operation and Programming Manual

This document describes the general use of the DICOMED. A copy is located in the NTEC Computer Lab (N-74).

In the following paragraphs, FORTRAN variables are identified by all capital letters, e.g., COLOR, ITEMBUF(512/512). VAX-11/780 files names are identified by their name and file specification, e.g., DICBWSA.FOR, RANIM.EXE, SRCAMSA.COM. Words that reference FORTRAN statements are in single quotes (´), e.g., ´called´ for subroutine calls, ´included´ for INCLUDE statements. INCLUDE statements allow a file to be ´included´ in the FORTRAN compilation. INCLUDE is usually used for data declarations, variables, and arrays that are common to different routines.

Appendix DA through DI is the documentation of a text generator for the **DICOMED**. The text generator allows information about the picture to be printed as part of the photograph.

The **DICOMED** D47 Image Recorder is used to translate binary data into Polaroid or 35mm photographs. One binary data word is assigned to each picture element (pixel) in the photograph. The set of binary data words for a photograph is referred to as picture data. Each data word may have either 6 or 8 bits of accuracy. Six bits implies 64 intensities, 8 bits implies 256 intensities. The **DICOMED** driver routines, written by Jeff Rubin prior to 1979, only allow for the use of 8 bit data words. Eight bit data is referred to as a **FORTRAN** byte data word and has a range from -128 to 127. Zero (0) corresponds to black, 127 to grey, -128 to grey +1, and -1 to white in the normal output mode. Black becomes white and white black in the complement mode. Rubin's routines only allow normal mode output. If the generated picture data ranges from 0 to 255, a shift should be performed as follows to avoid a **FORTRAN** overflow error.

```
IR = < data >      !0 < data < 255
ISHIFT = 256 * (IR/128)
BUF1(IXBS) = IR - ISHIFT
```

where BUF1 is a byte array and IXBS is the data's location in that array.

Stand Alone Routines

Stand alone routines are used when the data has been previously generated by another program and stored in a disk or mag tape file. Stand alone routines require that the picture data be read into the program in order to be output as a picture. Two stand alone routines exist: DICBWSA.FOR for black and white, DICCOLSA.FOR for color.

DICBWSA.FOR

DICBWSA.FOR is the **DICOMED** Black and White Stand Along routine which allows the output of previously generated black and white picture data. The following is a step-by-step procedure for using DICBWSA.FOR.:

1. Store the picture data in a VAX **FORTRAN** data file. These files are disk files entitled FOROnn.DAT. where nn is an integer between 10 and 99. A number between 1 and 10 may be used, but there is a risk of interfering with system assigned devices.

2. Set up the **DICOMED** using the **DICOMED** start up procedure described in Appendix AA.

3. Compile DICBWSA.FOR.

4. Link DICBWSA.FOR with the DICOMED driver routines: @ DICBWSA. The @ executes a VAX command procedure entitled DICBWSA.COM. DICBWSA.COM is listed here.
```
$LINK/EXE=[P8734]DICBWSA -
DIC:BLKDATA,[P8734]DICBWSA, -
DIC:DICOMED/LIB
```

5. DICBWSA is now ready to run:
```
RUN DICBWSA
```

6. A terminal session showing the compile, link, run, and response to requests from DICBWSA is shown here for reference. Normal type represents computer response, boldface represents user response.

```
$ FOR DICBWSA
$ @DICBWSA
$ RUN DICBWSA

GENERAL STAND ALONE BLACK AND WHITE DICOMED ROUTINE

THE PICTURE DIMENSIONS ARE SET AT 512,512
ENTER TYPE OF FILM: POLAROID=0. 35mm=1.
0
DO YOU WISH TO FILTER THE DATA? (YES=1)
1
THE RESOLUTION IS SET AT 3.
IS A READ REQUIRED? (YES=1)
1
ENTER THE FILE YOU WANT TO READ ITEMBUF
25
    READING ITEMBUF FROM FILE       25
    READING INT2 FROM FILE          25
FILLING ANY MISSED PIXELS
MISSED PIXEL AT IXBS,IYBS=   512   512
  SENDING DATA TO THE DICOMED
THE END
DO YOU WANT TO DO ANOTHER PICTURE? (YES=1)
0
FORTRAN STOP
$ PURGE DICBWSA.*
```

The output is on Polaroid black and white Type 52 film. An example is shown in Figure 49. The program listing of DICBWSA.FOR is given in Appendix CB.

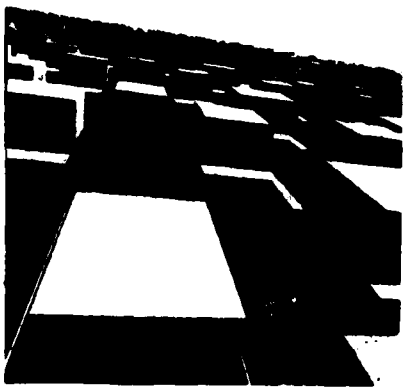


Figure 49. Output from DICBWSA.

DICCOLSA.FOR

DICCOLSA.FOR is the **DICOMED** Color Stand Alone routine which allows for the output, of previously generated color picture data, to the **DICOMED**. Step-by-step use of DICCOLSA.FOR is given here:

1. Store the picture data in a VAX **FORTRAN** data file. These files are disk files entitled FOROnn.DAT, where nn is an integer between 10 and 99. A number less between 1 and 10 may be used, but there is a risk of interfering with system assigned devices.

2. Complete the **DICOMED** start up procedure described in Appendix AA.

3. Compile DICCOLSA.FOR:
   $ FOR DICCOLSA

4. Link DICCOLSA with the **DICOMED** driver routines:
   $ @DICCOLSA

The @ executed the VAX command procedure DICCOLSA.COM is listed here:

```
$LINK/EXE=[P8734]DICCOLSA -
DIC:BLKDAT,[P8734]DICCOLSA, -
DIC:DICOMED/LIB
```

5. DICCOLSA is now ready for execution:
   A typical terminal session is illustrated here (regular type is computer, bold face is user):
   $ FOR DICCOLSA
   $ @DICCOLSA
   $ RUN DICCOLSA

```
GENERAL STAND ALONE COLOR DICOMED ROUTINE
THE PICTURE DIMENSIONS ARE SET AT 512,512
ENTER TYPE OF FILM: POLAROID=0. 35mm=1.
0
DO YOU WISH TO FILTER THE DATA? (YES=1)
1
THE RESOLUTION IS SET AT 3.
ENTER THE NUMBER OF DICOMED PASSES
2
IS A READ REQUIRED? (YES=1)
ENTER THE FILE YOU WANT READ ITEMBUF
30
     READING ITEMBUF FROM FILE       30
     READING INT2 FROM FILE       30
FILLING ANY MISSED PIXELS
SENDING NEUTRAL DATA TO THE DICOMED
SENDING GREEN(1),RED(2),BLUE(3)=      1
SENDING GREEN(1),RED(2),BLUE(3)=      2
SENDING GREEN(1),RED(2),BLUE(3)=      3
PASS    1 COMPLETED
SENDING NEUTRAL DATA TO THE DICOMED
SENDING GREEN(1),RED(2),BLUE(3)=      1
SENDING GREEN(1),RED(2),BLUE(3)=      2
SENDING GREEN(1),RED(2),BLUE(3)=      3
PASS    2 COMPLETED
THE END
DO YOU WANT TO DO ANOTHER PICTURE? (YES=1)
0
FORTRAN STOP
```

The output from the terminal session is given in Figure 50.

The FORTRAN listing of DICCOLSA.FOR is given in Appendix CC.



Figure 50. Output from DICCOLSA.

SUBROUTINES

DICOMED output routines are used when the picture data is to be sent directly to film and not necessarily written to disk or mag tape. SDICBW.FOR is used when only black and white picture data is being sent to the **DICOMED**. SDICCOL.FOR is used for sending color picture data to the **DICOMED**.

SDICBW.FOR

SDICBW.FOR (Subroutine DIComed Black and White) can be used where the picture data is generated and a photograph is desired immediately. The main program which reference the subroutine SDICBW must have the following declarations inserted at the beginning of its **FORTRAN** code:

```
INTEGER*2 ITEMBUR(512,512), INT2(512,526)

BYTE ICNT(512,512)

EQUIVALENCE (ICNT,INT2)

COMMON ITEMBUF,ICNT
```

Three parameters must be sent to SDICBW.FOR via a **FORTRAN** subroutine 'call' statement as follows:

```
CALL SDICBW(IMOV,IFLT,IAN)
```

where Table 10 defines the agruments.

TABLE 10. SDICBW ARGUMENTS

VARIABLE	VALUE	DEFINITION
IMOV	0	The type of film is Polaroid
	1	The type of film is 35mm
IFLT	0	Do **NOT** filter missed pixels
	1	**DO** filter missed pixels
IAN	0	Do **NOT** make a duplicate photograph
	1	**DO** make a duplicate photograph

A short test routine has been written to demonstrate the use of SDICBW.FOR.

```
C       TEST ROUTINE FOR SDICBW.FOR
C       HOWEVER, THE DATA IS READ INSTEAD OF GENERATED.
C
        INTERGER*2 ITEMBUF(512,512), INT2(512,256)
        BYTE ICNT (512,512)
        EQUIVALENCE (ICNT,INT2)
        COMMON ITEMBUF,ICNT
        READ (20,1000) ((ITEMBUF(I,J), I=1,512),J=1,512)
        READ (20,1000) ((INT2(I,J),I=1,512),J=1,256)
  1000 FORMAT (66A2)
        IMOV=0
        IFLT=1
        IAN=0
        CALL SDICBW(IMOV,IFLT,IAN)
        STOP
        END
```

SDICCOL.FOR

SDICCOL.FOR (Subroutine DIComed COLor) can be used when immediate color photographs are desired from the calling routine. The following FORTRAN 'include' statement must be inserted at the beginning of the main routine's FORTRAN code:

```
INCLUDE '[P8734]DICBUF.FOR'
```

A listing of DICBUF.FOR is shown here:

```
INTEGER*2 ITEMBUF(512,512,3),INT2(512,256)
BYTE NEUTRAL(512,512),ICNT(512,512)
EQUIVALENCE (ICNT,INT2)
COMMON ITEMBUF,NEUTRAL,ICNT
```

SDICCOL.FOR requires four parameters from the calling routine. The FORTRAN 'call' statement used is as follows:

```
CALL SDICCOL(IMOV,IFLT,IPASS,IAN)
```

where Table 11 defines the arguments.

TABLE 11. SDICCOL ARGUMENTS

VARIABLE	VALUE	DEFINITION
IMOV	0	The type of film is Polaroid
	1	The type of film is 35mm
IFLT	0	Do NOT filter missed pixels
	1	DO filter missed pixels
IPASS	x	x is the number of times data is duplicated on the film to give good color pictures (normal is 2)
IAN	0	Do NOT make a duplicate photograph
	1	DO make a duplicate photograph

Image Parameter Text Generation Routines

During testing of the **REAL SCAN** algorithms, numerous scene images are created on both Polaroid and 35mm film. The scene parameters which are used by the routines along with the date and time of scene generation must be correctly identified with each picture.

The following is a description of the operation of the **DEC FORTRAN IV PLUS** routines which create a field of alphanumeric text containing the scene and data base parameters. The text field is output by a separate **DICOMED** control routine adjacent to the scene image. The operation of each of the text generation routines will be described and the input/output variables and arrays will be identified. The routines documented in this subsection are:

1. DATACHAR.FOR
2. DICCHAR.FOR
3. CHCONV.FOR

Included will be inputs and outputs from an example execution of the data base test routine in EGLPICMAN.FOR. The routine used for the **DICOMED** control in the example is from DICCAMMAN.FOR and the data base used is from EGLDATA.FOR. Program listings for DATACHAR.FOR, DICCHAR.FOR, CHCONV.FOR, DICCAMMAN.FOR, EGLPICMAN.FOR, AND EGLDATA.FOR are included in Appendices DB, DC, DD, DE, DF, DG respectively.

OPERATIONAL OVERVIEW

The image parameter text generation routines are subroutines which must be called by a separate **DICOMED** control routine. Either

the scene generation control routine or the DICOMED control routine must transfer the data base parameters to the text generation routines. The text generation routines then output the text adjacent to the scene image.

The text routines generate the output text for the DICOMED control routine through the manipulation of three arrays, ICH, ASCH, and ITEXT. ICH, dimensioned as BYTE ICH(7,9,51), contains the character set consisting of 51 characters. Each character is 7 elements wide by 9 elements in length. ASCH, dimensioned as BYTE ASCH(17,39), defines the entire text field as 17 character spaces wide by 39 character spaces in length. For ASCH(I,J), an index value of I equal to 1 defines the leftmost and an I equal to 17 defines the rightmost column of the text field. An index value of J equal to 1 defines the top row of the text field. ASCH is coded with the decimal ASCII code for each character in the text field. ITEXT, dimensioned as BYTE ITEXT(170, 512), is the output array from the text generation routines. For ITEXT(I,J), index values of I equal to 1 and I equal to 170 defines the leftmost and rightmost elements of the first and last character spaces in row J, respectively, and index values of J equal to 1 and J equal to 512 defines the top and bottom elements of the first and last character spaces in column I, respectively. ITEXT defines each element of the 7 by 9 characters identified by ASCH as an array element of ITEXT over the entire text field.

The character ASCII codes of ASCH, for the text which is fixed, and the character set of ICH are initialized via DATA statements. Array ASCH is loaded with all of the input text variables by either the image generation control routine or the DICOMED control routine. The text generation routines input array ASCH via a common block of memory. ITEXT is then created from ASCH by accessing the character ASCII code in each element of ASCH and filling the corresponding elements of ITEXT with the character of ICH pointed to by ASCH. As an example, let ASCH have as element ASCH(3,20)=81. This defines, in the text field, that the character space 3rd from left to right and the 20th from top to bottom has the decimal ASCII code 81, which is the letter "Q". The 63 elements corresponding to that character space in ITEXT are filled with the 7 by 9 element character construction for "Q" as defined by ICH. Once ITEXT is filled from ASCH, control is returned to the DICOMED control routine which uses ITEXT to print the text adjacent to the scene image. Figure 51 gives a functional block diagram of the text generation routines operation in conjunction with the image generation routines.

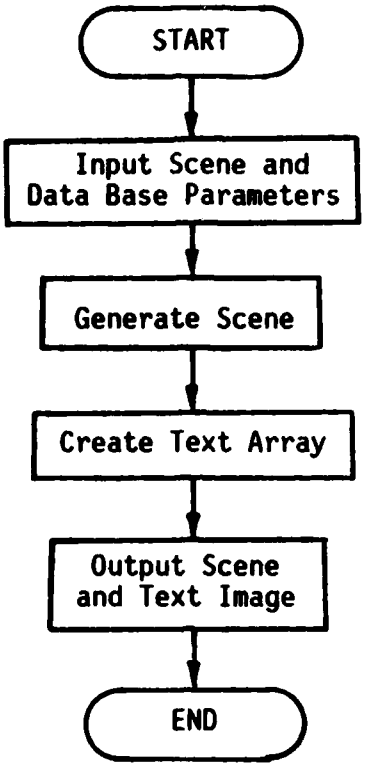


Figure 51. Flowchart of Image Generation and Text Routine Integration.

DATACHAR.FOR

The FORTRAN file DATACHAR.FOR contains no executable statements. DATACHAR.FOR has only three DATA statements, which are used to initialize the character set array ICH and the character text array ASCH. The statements are executed in Subroutine DICCHAR via the FORTRAN statement INCLUDE DATACHAR.FOR. Due to the size of ICH and the constraint placed upon the FORTRAN code by the DEC FORTRAN IV PLUS compiler limit of a maximum of 99 lines of continuation, ICH must be initialized in two parts. This is performed by dimensioning ICH1 (7,9,22) and ICH2(7,9,29) and equivalencing them to ICH via the FORTRAN statement below:

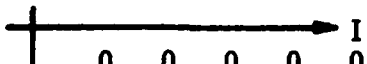
```
EQUIVALENCE (ICH,ICH1),(ICH(1,1,23),ICH2)
```

This allows the use of a DATA statement for ICH1 and a DATA statement for ICH2 to accomplish the initialization of ICH.

The DATA statements for ICH1 and ICH2 in DATACHAR.FOR initialize each of the 51 characters of ICH(7,9,51) by defining each element in each character as either -1 or 0. The constants -1 and 0, expressed

in byte form, represent the maximum and minimum intensities, respectively, that can be output by the DICOMED. A -1 designates white and 0 designates black as seen on photographs produced on the DICOMED. The character set defined by ICH is shown in Appendix DH with each character having an asterisk for a displayed element and a space for elements not being displayed. The character constructions shown in Appendix DH are used since their low degree of similarity among characters with similar symbols leads to a high degree of recognition. Also, shown in Appendix DI are some of the possible variations in letter constructions.

The third DATA statement in DATACHAR.FOR initialized ASCH with the text information which is fixed. ASCH is initialized with the decimal ASCII code for each character in the fixed text and with the decimal number 0 for character spaces which are to be left blank. Figure 52 shows the array ASCH after initialization.



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	88	61	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	89	61	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	65	61	0	0	46	0	0	0	0	0	0	0	0	0	0
0	0	66	61	0	0	46	0	0	0	0	0	0	0	0	0	0
0	0	67	61	0	0	46	0	0	0	0	0	0	0	0	0	0
0	0	78	67	82	61	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	84	69	82	82	65	73	78	0	0	0	0	0
0	0	83	73	90	69	61	0	0	0	0	0	0	0	0	0	0
0	0	84	89	80	69	61	0	0	0	0	0	0	0	0	0	0
0	0	72	79	82	90	61	0	0	0	0	0	0	0	0	0	0
0	0	86	69	82	84	61	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	84	82	69	69	83	47	82	79	67	75	83	0	0	0
0	0	83	73	90	69	61	0	0	0	0	0	0	0	0	0	0
0	0	84	89	80	69	61	0	0	0	0	0	0	0	0	0	0
0	0	72	79	82	90	61	0	0	0	0	0	0	0	0	0	0
0	0	86	69	82	84	61	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	71	82	65	83	83	47	76	69	65	86	69	83	0	0	0
0	0	83	73	90	69	61	0	0	0	0	0	0	0	0	0	0
0	0	84	89	80	69	61	0	0	0	0	0	0	0	0	0	0
0	0	72	79	82	90	61	0	0	0	0	0	0	0	0	0	0
0	0	86	69	82	84	61	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	76	65	75	69	0	0	0	0	0	0	0
0	0	83	67	65	76	69	61	0	0	0	0	0	0	0	0	0
0	0	65	88	73	83	0	83	84	82	61	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	70	73	69	76	68	83	0	0	0	0	0	0
0	0	83	76	79	80	69	61	0	0	0	0	0	0	0	0	0
0	0	83	73	90	69	61	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	83	85	78	0	83	72	65	68	79	87	58	0	0	0	0	0
0	0	70	73	82	83	84	58	0	0	0	0	0	0	0	0	0
0	0	83	69	67	79	78	68	58	0	0	0	0	0	0	0	0

Figure 52. ASCII Codes for Text Initialized in ASCH(I=1-17,J=1-39)

DICCHAR.FOR

The FORTRAN file DICCHAR.FOR contains subroutine DICCHAR which performs the character text generation. Subroutine DICCHAR requires as input, from an external source, all of the text variables and arrays listed in Table 12 for the generation of the text field.

TABLE 12. TEXT VARIABLES

INPUT VARIABLES:

NAME	TYPE	COMMENT
A	REAL	Sun Vector X Component
B	REAL	Sun Vector Y Component
C	REAL	Sun Vector Z Component
IX	I*4	X Position of Scene Center
JY	I*4	Y Position of Scene Center
NCR	I*2	Database Coordinate Scale Factor
LSCAL	I*4	Lake Height Scale Factor
IWAVSCAL	I*4	Lake Coordinate Scale Factor

INPUT ARRAYS:

NAME	TYPE	COMMENT
FLDSL(8)	CHAR*1	Field Scale Factor
FLDSZ(9)	CHAR*1	Field Size Factor
GRAHOR(9)	CHAR*1	Grass Ground Scale Factor
GRASZ(9)	CHAR*1	Grass Noise Filter Size
GRATP(9)	CHAR*1	Grass Noise Type
GRAVER(9)	CHAR*1	Grass Height Factor
ROCHOR(9)	CHAR*1	Rock Ground Scale Factor
ROCSZ(9)	CHAR*1	Rock Noise Filter Size
ROCTP(9)	CHAR*1	Rock Noise Type
ROCVER(9)	CHAR*1	Rock Height Factor
SUNFT(8)	CHAR*1	Sun Shadow First Factor
SUNSD(7)	CHAR*1	Sun Shadow Second Factor
SUNSH(3)	CHAR*1	Sun Shadow? yes or no
TERHOR(9)	CHAR*1	Terrain Ground Scale Factor
TERSZ(9)	CHAR*1	Terrain Noise Filter Size
TERTP(9)	CHAR*1	Terrain Noise Type
TERVER(9)	CHAR*1	Terrain Height Factor
TITLE(15)	CHAR*1	Title Character String

All of the variables and arrays input by subroutine DICCHAR are either input from the user, computed by either the DICOMED control routine or the image generation control routine, or defined by the

data base. The variables and arrays are of two types, character string and numeric. The numeric variables must be converted to character string arrays via 'calls' to subroutine CHCONV. The text input variables which are character strings are loaded into their positions in ASCH by the routine that input the text variables via the following equivalence statement found in the input routine.

```
      EQUIVALENCE (ASCH(7,4),XPOS(1)),
     * (ASCH(7,5),YPOS(1)),(ASCH(8,6),SUNA(1)),
     * (ASCH(8,7),SUNB(1)),(ASCH(8,8),SUNC(1)),
     * (ASCH(8,9),NCRCH(1)),(ASCH(9,12),TERSZ(1)),
     * (ASCH(9,13),TERTP(1)),(ASCH(9,14),TERHOR(1)),
     * (ASCH(9,15),TERVER(1)),(ASCH(9,18),ROCSZ(1)),
     * (ASCH(9,19),ROCTP(1)),(ASCH(9,20),ROCHOR(1)),
     * (ASCH(9,21),ROCVER(1)),(ASCH(9,24),GRASZ(1)),
     * (ASCH(9,25),GRATP(1)),(ASCH(9,26),GRAHOR(1)),
     * (ASCH(9,27),GRAVER(1)),(ASCH(10,30),LAKSC(1)),
     * (ASCH(13,31),LAKAS(1)),(ASCH(10,34),FLDSL(1)),
     * (ASCH(9,35),FLDSZ(1)),(ASCH(14,37),SUNSH(1)),
     * (ASCH(10,38),SUNFT(1)),(ASCH(11,39),SUNSD(1)),
     * (ASCH(2,1),TITLE(1)),(ASCH(2,2),CHDATE),
     * (ASCH(12,2),CHTIM)
```

The converted character strings are loaded in their positions in ASCH by the use of an identical FORTRAN equivalence statement by subroutine DICCHAR to the equivalence statement described above.

Figure 53 shows array ASCH with the ASCII codes for each fixed initialized character translated into its alphanumeric symbol and the input character strings in their corresponding positions.

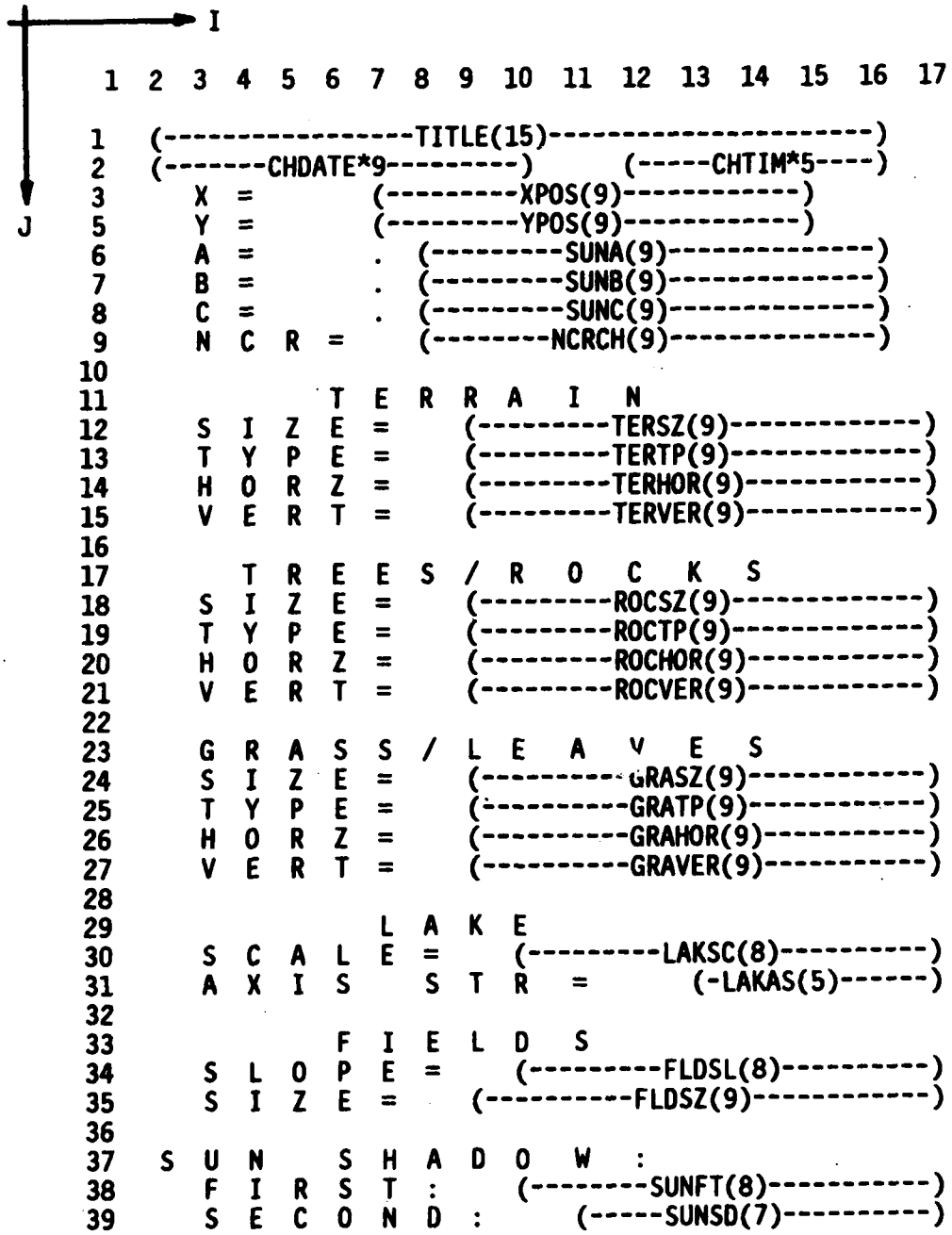


Figure 53. Array ASCH(I=1-17,J=1-39) with Character Strings in Relative Positions.

Subroutine DICCHAR executes the generation of the image parameter text by performing the function illustrated in Figure 54.

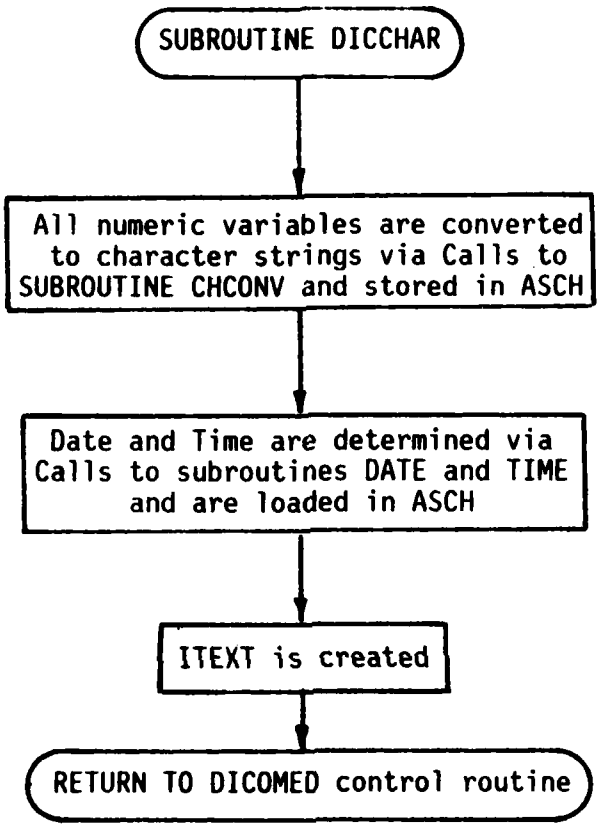


Figure 54. Function Flowchart for Subroutine DICCHAR.

A detailed description of each function follows:

1. All integer and real variables are converted to character string arrays and stored in ASCH. The conversions are performed by first assigning the conversion type flag, IFGT, to 0 for converting integer variables to character strings or assigning IFGT to 1 for converting real variables of magnitude less than 1 to character strings. The variable to be converted is then assigned to ITST if it is integer, or RTST if it is real. Subroutine CHCONV is called and returns with the byte array BCHTST(9), which contains the ASCII code with up to 9 digits of converted variable. A sign flag, ISNT, is returned from Subroutine CHCONV and has the value of 1 if the converted variable is negative and 0 if the converted variable is positive. ASCH is then loaded with the sign of the variable, if negative. Subroutine DICCHAR suppresses leading zeros when loading integer variable conversion character strings or suppresses trailing zeros when loading real variable conversion character strings.

The FORTRAN code is shown below for an example of an integer variable conversion 'call' and for a real variable conversion call. The integer variable being converted is IX and the real variable being converted is A.

```
C
C-------------------------------------------------------------
C
C       INTEGER CONVERSION EXAMPLE FROM SUB DICCHAR
C
        IFGT=0          !  SET CONVERSION TYPE FLAG FOR INT.
        ITST=IX         !  IX IS VARIABLE FOR CONVERSION
        CALL CHCONV     !  CALL CONVERSION SUBROUTINE
        IF(ISNT.EQ.1) ASCH(6,4)=45   !  NEG.SIGN
        IF(ISNT.EQ.0) ASCH(6,4)=0    !  NO SIGN FOR POS.
        DO 10 K=1,9     !  LEADING ZERO SUPPRESS IDENT
                IND=K
                IF(BCHTST(IND).NE.48)  GOTO  11
10      CONTINUE
11      KK=0
        DO 12 K=IND,9   !  FILL ASCH VIA EQUIV.
                KK=KK+1
12      XPOS(KK)=CHTST(K)
        DO  13  K=KK+1,9
13      XPOS(K)=SUNA(10)  !  SUNA(10) IS  0  CONSTANT
C
C-------------------------------------------------------------
C
C       REAL CONVERSION EXAMPLE FROM SUB. DICCHAR
C
        IFGT=1          !  SET CONVERSION TYPE FLAG
        RTST=A          !  A IS VARIABLE FOR CONVERSION
        CALL CHCONV     !  CALL CONVERSION SUBROUTINE
        IF(ISNT.EQ.1)   ASCH(6,6)=45  !  NEG.SIGN
        IF(ISNT.EQ.0)   ASCH(6,6)=0   !  NO SIGN FOR POS.
        DO 40 K=7,1,-1  !  TRAILING ZERO SUPPRESS IDENT
                IND=K
                IF(BCHTST(K).NE.48)  GOTO  41
40      CONTINUE
41      KK=0
        DO  42 K=1,IND  !  FILL ASCH VIA EQUIV.
                 KK=KK+1
42      SUNA(K)=CHTST(K)
        DO  43 K=KK+1,9
43      SUNA(K)=SUNA(10)   !  SUNA(10) IS 0 CONSTANT
C
C-------------------------------------------------------------
C
```

2. The VAX/VMS system subroutines DATE and TIME are 'called' to generate the character strings for the date and time in the text field.

3. ITEXT is filled with the character constructions of ICH for each character is ASCH. The characters of ICH are arranged in order of ascending ASCII codes such that only a shift of decimal 42 is needed to determine the character index of ICH, which is the third index "z" in ICH(x,y,z), from the character's decimal ASCII code in ASCH. Once the third index of ICH is determined, the character's 7 by 9 construction is filled in the appropriate elements of ITEXT. Each character space in ASCH has a corresponding 10 element wide by 13 element long space in ITEXT. This leaves 3 elements of ITEXT for spacing between the characters and 4 elements of ITEXT for spacing between text field lines. If the input character string contains spaces, which have decimal ASCII code 32, as ITEXT is being created, the characters in ASCH which are spaces are changed to decimal 0, since no construction for a space occurs in ICH. When an element of ASCH has the value 0, all 7 by 9 corresponding elements of ITEXT are assigned the value 0. In the case where the text generation routines are linked with an image generation routine which allows for the generation of more than one scene during each execution, the array ITEXT is generated for the first image and then for subsequent scenes only those parameters which can change are updated. Array ITEXT is structured such that the scene parameters, which may change for each scene, are filled in ITEXT prior to the data base parameters, which will not change for each execution of the scene generation routines. ITEXT is created via a nested FORTRAN DO loop which fills each text line in ITEXT from the top to the bottom of the text field. Therefore ITEXT can be filled and updated using a variable to designate the last line of ITEXT to be filled. This ending line number variable, called IEND3, is initialized to 39 in the DATA statement, DATA IEND3/39/, for the first complete generation of ITEXT.

After the DO loop is completed, IEND3 is assigned the value 9 which allows for the updating of the scene parameters in lines 1 through 9 of the text. The FORTRAN code which creates ITEXT is listed below.

```
C
C-------------------------------------------------------------
C
                DO  100 K1=1,IEND3
                DO  200 K2=1,17
                     IF(ASCH(K2,K1).EQ.32)  ASCH(K2,K1)=0
                     K3DEX=(K1-1)*13 + 2
                     K4DEX=(K2-1)*10
```

```
                    ICHDEX=ASCH(K2,K1)-42
                    DO  300 K3=1,9
                    DO  300 K4=1,7
                         IF(ASCH(K2,K1).EQ.0) THEN
                         ITEXT(K4+K4DEX,K3+K3DEX)=0
                         ELSE
                         ITEXT(K4+K4DEX,K3+K3DEX)=
          *                  ICH(K4,K3,ICHDEX)
                         ENDIF
300                 CONTINUE
200               CONTINUE
100               CONTINUE
                  IEND3=9
C
C-------------------------------------------------------------
C
```

4. Return to **DICOMED** control routine and pass output array ITEXT via the common block of memory COMMON/CHR3/ ITEXT.

CHCONV.FOR

The **FORTRAN** file CHCONV.FOR contains subroutine CHCONV which performs the conversion of integer variables to character strings and the conversion of real variables of magnitude less than one to character strings. Subroutine CHCONV is only called by subroutine DICCHAR and returns to only subroutine DICCHAR.

All of the input and output variables used by Subroutine CHCONV are listed in Table 13.

TABLE 13. CHARACTER CONVERSION VARIABLES

INPUT VARIABLES:

NAME	TYPE	COMMENT
IFGT	I*4	Flag for Real or Integer Convert
ITST	I*4	Integer Variable for Conversion
RTST	I*4	Real Variable for Conversion

OUTPUT ARRAYS AND VARIABLES:

NAME	TYPE	COMMENT
BCHTST	BYTE	ASCII String from Conversion
ISNT	I*4	Sign Flag for Conversion

Subroutine CHCONV performs the conversion from numeric variables to character strings by performing the functions illustrated in Figure 55.

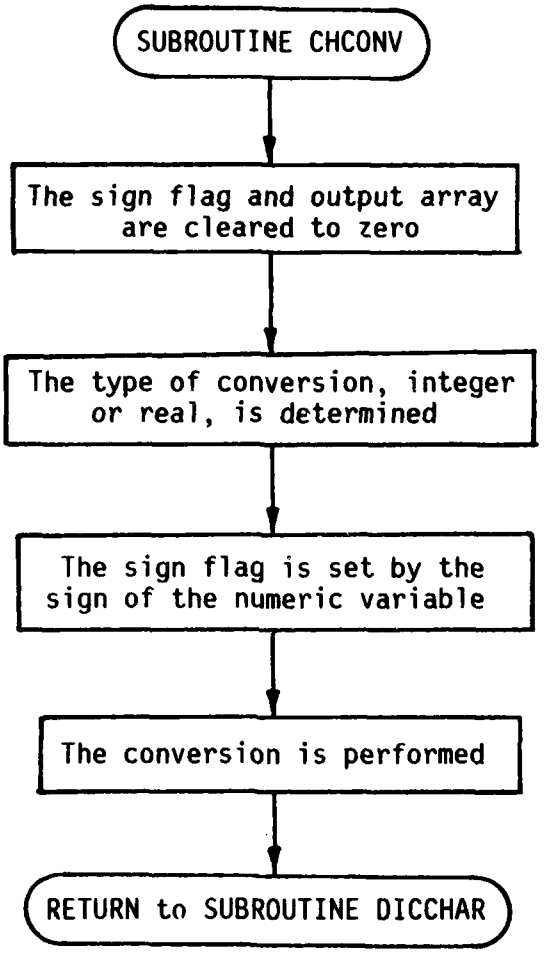


Figure 55. Function Flowchart for Subroutine CHCONV.

Each function of Figure 55 is detailed as follows:

A. The sign flag, ISNT, and the output array, BCHTST, are cleared to zero.

B. Conversion type flag, IFGT, is tested to transfer control to the integer conversion section of subroutine CHCONV if IFGT equals 0 or to the real conversion section of subroutine CHCONV if IFGT equals 1.

C. The conversion variable is tested for sign and the sign flag ISNT is assigned the value of 1 if the conversion variable is negative or 0 if the conversion variable is positive.

D. The conversion is then performed. For converting integer values, the digit corresponding to the 10 raised to the "i" power, called D; is given by

$$D_i = \text{INT}\,[V/10^i] - \text{INT}\,[V/10^{i+1}] * 10 \qquad (78)$$

where "V" is the integer value being converted and INT[X] defines the operation of taking the integer value of argument "X". The FORTRAN code which executes the conversion for integer variables is listed below.

```
C
C-------------------------------------------------------------
C
      ITST=IABS(ITST)
      IR1=ITST
      IR2=ITST
      DO  10 I=1,9
            IR1=IR1/10
            NUM=IR2-IR1*10
            IR2=IR2/10
            BCHTST(10-I)=NUM+48   !  48  ADDED TO GET ASCII
10    CONTINUE
C
C-------------------------------------------------------------
C
```

The conversion is carried out by first taking the magnitude of the variable to be converted, called ITST, and assigning it to two local variables, called IR1 and IR2. A loop, with loop counter I, is entered which first takes the value of IR1 and assigns it the value of IR1/10, where an integer divide is used. The digit which occupies the (I-1) power of 10, called NUM, is determined by subtracting IR1*10 from IR2. IR2 is integer divided by 10 to prepare for the next pass through the loop. The ASCII code for the digit in NUM is calculated by adding decimal 48 to NUM. This value is stored in BCHTST(10-I). Nine passes through the loop determines the string of 9 characters which represents the value of ITST. If the value of ITST is less than 9 digits leading spaces are created in BCHTST.

For real variables of magnitude less than one, the value of the "-i" digit, called D_i is

$$D_i = \text{INT}[V*10^i] - \text{INT}[V*10^{i-1}] * 10 \qquad (79)$$

where "V" is the real variable being converted and INT[X] is as defined above. The FORTRAN code which executes the conversion is listed below.

```
C
C-------------------------------------------------------------
C
        RTST=ABS(RTST)
        IR1=0
        DO  20 I=1,7
        RTST=RTST*10
        IR1=IR1*10
        NUM=RTST-IR1
        BCHTST(I)=NUM+48   !  48 ADDED TO GET ASCII
        IRI=RTST
 20     CONTINUE
C
C-------------------------------------------------------------
C
```

The conversion is carried out by first taking the magnitude of the fraction to be converted, called RTST. The local variable IR1 is initialized to 0. This initialization sets the value of IR1 equal to the integer portion of RTST. A loop, with loop counter I, is entered which shifts RTST and IR1 one digit to the left by multiplying both RTST and IR1 by 10. This sets IR1 equal to the integer portion of RTST with the exception that the units digit of IR1 is guaranteed to be zero. Then the 10 raised to the (-I) power digit of RTST, called NUM, can be determined by subtracting IR1 from RTST. Decimal 48 is added to NUM to get the ASCII code for the digit in NUM and this value is assigned to BCHTST(I). IR1 is then assigned the value of RTST in order to set up for the next pass through the loop. The conversion loop is executed 7 times to obtain 7 digits of the real variable in RTST.

E. Subroutine CHCONV returns to subroutine DICCHAR and passes the character string output array BCHST(9) and sign flag ISNT to subroutine DICCHAR via the common block of memory defined by the following FORTRAN statement: COMMON/CONV/ IFGT,ITST, RTST,BCHTST,ISNT.

OPERATION EXAMPLE

The text generation routines with an example of the DICOMED outputs are illustrated by using the following routines, EGLPICMAN. FOR, EGLDATA.FOR, DICCAMMAN.FOR, in conjunction with the text generation routines of DATACHAR.FOR, DICCHAR.FOR, and CHCONV.FOR.

The FORTRAN file EGLPICMAN.FOR contains program EGLPICMAN which is a data base test program. Program EGLPICMAN generates a 512 by 512 resolution scene by accessing the data base once for each of the (512, 512) display locations. This results in an image which has the effect of viewing the data base from infinity. The relative viewing altitude

is determined by the data base sample point spacing, called NCR. Program EGLPICMAN takes as input the parameters defined in Table 14.

TABLE 14. DATA BASE TESTING PARAMETERS

NAME	TYPE	COMMENT
IX	I*4	X OFF-SET TO SCENE CENTER
JY	I*4	Y off-set to scene center
NCR	I*2	Sample Point Spacing
TZDEG	REAL	Polar Sun Angle From Z Axis
TXDEG	REAL	Cylindrical Sun Angle From X Axis

Program EGLPICMAN generates a 512 by 512 by 3 array called IBUFX which contains the three color reflectance values for the scene that is used by subroutine BPDICCAM to create the DICOMED scene image.

The FORTRAN file DICCAMMAN.FOR contains subroutine BPDICCAM which is a DICOMED output routine for the scene generated in program EGLPICMAN. Subroutine BPDICCAM takes the text array ITEXT(170,512) and fills the neutral output array NEUTRAL (682,512) as shown below.

```
C
C-------------------------------------------------------------
C
C
          DO  114 KSR=1,512
          DO  114 KSM=1,170
          NEUTRAL(KSM+512,513-KSR)=ITEXT(KSM,KSR)
   114    CONTINUE
C
C
C-------------------------------------------------------------
C
```

The array NEUTRAL is then output to the DICOMED image recorder along with the color scene array IBUFX.

In the following example the commands used are valid for execution of a DEC VAX11/780 using VAX/VMS operating system. The compiler being called is the DEC FORTRAN IV PLUS compiler. Listed below are the necessary compile and link commands:

```
$ FORTRAN EGLPICMAN
$ FORTRAN EGLDATA
$ FORTRAN DICCAMMAN
$ FORTRAN/CONT=99 DICCHAR
$ FORTRAN CHCONV
$ LINK/EXE=EGLPICMAN EGLPICMAN,EGLDATA,DICCHAR, -
  CHCONV,DICCAMMAN,DRAO:[UTILITY.DICOMED]BLKDAT, -
  DRAO:[UTILITY.DICOMED]DICOMED/LIB
```

As an example of the user conversation with the text routine, a sample execution of EGLPICMAN.EXE as generated from the routines discussed is shown. The user responses are shown enclosed in boldface, and are followed by hitting "RETURN".

```
$ RUN EGLPICMAN
READING NOISE
DONE READING NOISE
WOULD YOU LIKE THE IMAGE PARAMETER TEXT GENERATES?
  YES=1, NO=0
1
ANSWER THE FOLLOWING QUESTIONS WITH A CHARACTER
  STRING OF LENGTH SPECIFIED BY (LENGTH)

ENTER TITLE (15)
TEST RUN
ENTER TERRAIN NOISE PARAMETERS
   SIZES (9)
512,512
   TYPES (9)
PARB F=.5
   HORIZONTAL SCALES (9)
733,2483
   VERTICAL SCALES (9)
8 , 31
ENTER TREES/ROCKS NOISE PARAMETERS
   SIZES (9)
512,512
   TYPES (9)
CP91,CRF1
   HORIZONTAL SCALES (9)
7 , 31
   VERTICAL SCALES (9)
0.5 , 2.5
ENTER GRASS/LEAVES NOISE PARAMETERS
   SIZES (9)
512,512
   TYPES (9)
UNIFORM
```

```
   HORIZONTAL SCALES (9)
1 , 31
   VERTICAL SCALES (9)
0.03,0.12
ENTER FIELD SLOPE (8)
31,IJ=77
ENTER FIELDS SIZE (9)
25 , 31
IS SUN SHADOW PRESENT IN THE DATA BASE?  YES OR NO
NO
TYPE XOFF-SET TO SCENE CENTER
25575
TYPE YOFF-SET TO SCENE CENTER
298750
TYPE SAMPLE POINT SPACING,AN INTEGER
128
Type the polar sun angle from the Z axis (DEGREES)
30
Type the cylindrical sun angle from the
 X axis toward the Y axis (DEGREES)
120
X,Y=    25575  298750     SCALE=  128  NOISE?=  0
STOP=1,CONTINUE=0
0
SHALL WE MAKE A PICTURE? (YES=1)
1

DATA WILL BE OUTPUT TO THE DICOMED
ENTER TYPE OF FILM:  POLAROID=0; 35mm=1
0
ENTER THE NUMBER OF DICOMED PASSES
2
CREATING TEXT ARRAY
  SENDING NEUTRAL DATA TO THE DICOMED
SENDING GREEN(1),RED(2),BLUE(3)=   1
SENDING GREEN(1),RED(2),BLUE(3)=   2
SENDING GREEN(1),RED(2),BLUE(3)=   3
PASS      1 COMPLETED
  SENDING NEUTRAL DATA TO THE DICOMED
SENDING GREEN(1),RED(2),BLUE(3)=   1
SENDING GREEN(1),RED(2),BLUE(3)=   2
SENDING GREEN(1),RED(2),BLUE(3)=   3
PASS      2 COMPLETED
THE END
DO YOU WANT TO DO ANOTHER PICTURE?(YES=1)
0
FORTRAN STOP
$
```

Before answering the "STOP=1,CONTINUE=0" with a "0" the DICOMED Image Recorder must be loaded with the appropriate film, and adjusted. The image shown in Figure 56 was created by the above terminal session example.

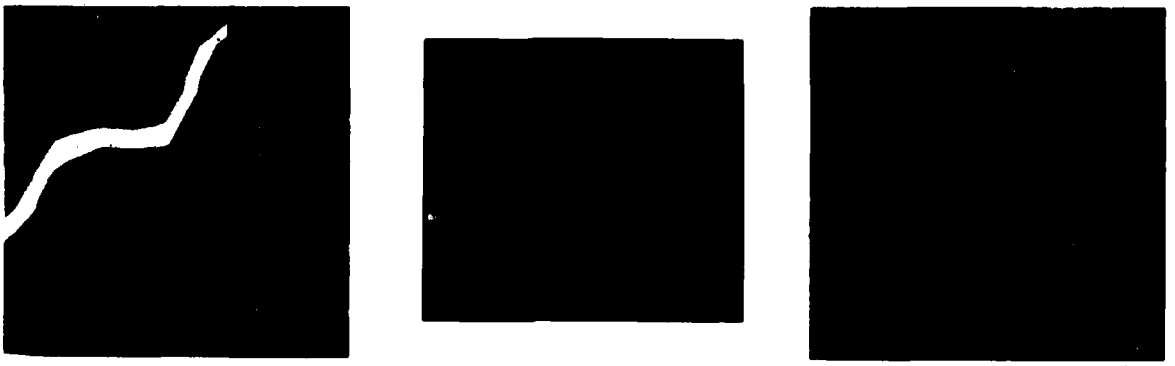


Figure 56. Image Generated by Sample Run.

RECOMMENDATIONS

The image parameter text generation routines require the user to input the text information concerning the data base scale factors. Since these are available to the text routines, the necessary modifications to the existing text generation routines should be made such that program data can be obtained from the data base routine and not input by the user. This would eliminate the need for the user to recall the data base scale factors at execution time. The sun vector components should be eliminated from the display and the sun angles from which the sun vector was computed should be displayed, since the sun angles are more easily interpreted than the sun vector components.

SECTION VII

CONCLUSIONS AND RECOMMENDATIONS

The purpose of this section is to summarize the accomplishments of the **REAL SCAN** research effort, to define tasks which can impact the development of real-time computer image generation of display limited resolution, and to recommend a course of further research.

Accomplishments

The **REAL SCAN** research effort has accomplished the following:

1. Developed integer algorithms operating on an integer data base, where the data base address corresponds to the data's location. These integer algorithms produce display limited, high detail scenes.

2. Demonstrated that a hierarchial ground referenced data base is possible. Further, the averaging of detail from high detail levels of the hierarchy to low detail levels yields CIG pictures and movies where aliasing or image defects are difficult to ascertain. Figure 57 illustrates a typical photo of the block city data base.

3. Demonstrated negligible aliasing occurs for a ground referenced data base when calculated via integer algorithms at the rate of about four ground points per display pixel. Appendix M and Figure 57 illustrate **REAL SCAN** aliasing.



Figure 57. One Frame of City Block Motion Picture.

4. Demonstrated the size of the ground referenced data base for a hierarchial system only adds about 33.4% additional memory requirement over and above the high detail memory. Yet the computational load is limited to the display resolution. The total memory required for an 80km by 80km gaming area is between 10^{11} to 10^{12} bytes. Appendices A and B derive memory requirements. Section II develops an estimate for a high detail central gaming region bounded by a region of decreasing resolution extending to the horizon.

5. Developed DICOMED routines to generate pictures. Section VI and Appendices AA through CE document the use of the DICOMED.

6. Made color pictures which indicate that a course elevation map with a superimposed hierarchy of culture file detail can produce desired high detail realistic scenes. Figure 58 illustrates a 3-D image of terrain produced with filtered noise files. A course elevation map with superimposed detail suggests a total memory requirement of 10 to 1000 mega bytes for an 80Km by 80Km gaming area.

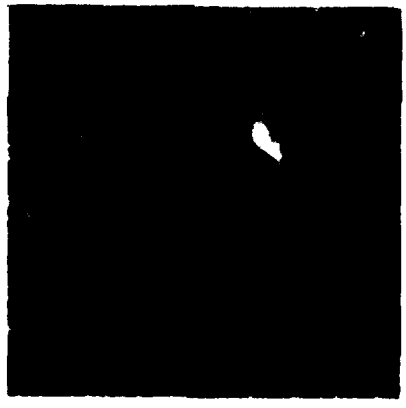


Figure 58. Simulated Terrain from Noise Files.

7. Developed methods to simulate terrain texture and produce culture files. Further, methods have been developed to test the simulated culture files. Section III describes the methods of filtering and redistributing noise. Appendix BC lists the color test programs. Appendices HA through HC list the texture production software.

8. Developed processing requirements and architectural constraints for a CIG system producing display limited resolution. Section V develops the architectural constraints and

provides a **REAL SCAN** structure based on parametric constraints which provide a modular system. Any display limited CIG system producing 512 x 512 anti-aliased pixels 60 times per second will require an integer computation rate of about $5x10^9$ operations per sec. It is estimated that current bit-slice technology requires about 20 well designed cabinets to achieve $5x10^9$ operations per sec. Since this world based scan line geometry does not match the display geometry, a simple and small buffer scheme was invented to accommodate the geometry transformation.

9. Developed algorithm improvements for speeding up the scene calculation process. Appendices E, F, G, H, J and O document six algorithm improvements.

10. Developed the intermediate plane concept which impacts all CIG procedures. This concept, derived in Appendix C, shows that a set of planes "painted" with the scene volume between two adjacent planes and projected to the near plane, can substantially reduce the real-time scene computational load. The intermediate planes are established at distances such that resolvable parallax error does not occur. If the resolution of a pixel is 1/500, and if resolvable parallax is 1/1000, then the number of intermediate planes required is no less than the scaled eye displacement, in resolution units, between intermediate plane updates. Hence, if the scaled range to the nearest point is 1000 and if 1000 units of motion (i.e., 45° change with respect to near field to far field parallax) is allowed then 1000 intermediate planes would be required to hold approximately 10^6 data points (i.e., about 4 intermediate plane data points per pixel). Hence, if the relative near field to far field angular displacement per 1/60 sec is 4.5°, then the intermediate planes need be calculated at only 0.1 the rate of the full scene. Hence, a display 10 times as complex as current technology can be obtained by means of a modest buffer memory.

11. Two black and white movies were produced by flying through the block data base. The first movie produced 128 frames with about four ground points per pixel. The second movie computed the intermediate 128 frames for a total of 256 frames. Since the block data base repeats over a 4 x 4 grid, the movie frames were repeated for about a 2-minute movie. These movies did not show noticeable aliasing problems.

Advantages/Disadvantages of REAL SCAN

The **REAL SCAN** algorithms have been derived for producing scenes to display limited resolution in as simple a way as possible. In this regard **REAL SCAN** computation grows linearly with display capability. A conventional CIG, without Z depth buffer, has a computational load

basically separable into two parts: (1) detail which grows linearly with detail and (2) sorting which grows as the square of detail using current hardware (i.e., n log n for optimum sorting hardware). Moreover, **REAL SCAN** uses integer arithmetic whereas conventional CIG uses floating point arithmetic. Therefore, **REAL SCAN** can perform most operations with scaled 12 bit integers or 24 bit intermediate results. Whereas, floating point must be performed with 48 bit or more precision, even though a 512*512 display requires geometric accuracy of only 9 bits. Further, floating point operations are clearly more complex than integer operations, requiring number alignment before and after operations.

However, while **REAL SCAN** appears to offer an advantage for computing complex scenes, conventional CIG appears to offer an advantage for computing low detail scenes. This advantage occurs because **REAL SCAN** over samples all visible ground points to eliminate aliasing across straight line boundaries, while conventional CIG just determines the line in screen coordinates and then the anti-aliasing gray scale can be set up with an integer multiply and add. Hence, if one visible floating point projection and line determination corresponds to about 30 pixels (i.e., give or take a factor of 4) then conventional CIG will require less hardware for real time simulated scenes.

If conventional real time CIG were to use intermediate projection planes in place of a Z depth buffer, then a factor of approximately 10 more scene complexity should be available for negligible extra hardware.

The current **REAL SCAN** algorithms do not allow for multiple elevation scenes, such as a bridge, or an Eiffel Tower. This limitation must be removed if **REAL SCAN** is to compete. However, it is not known whether the number of elevations per ground point should be limited, or if some method similar to conventional planar data bases should be incorporated to compute multi-elevation data.

REAL SCAN seems to offer an advantage in terms of the number of interpolation algorithms which can be used. Conventional CIG is limited to surfaces of quadratic order or less if contours are to be determined from a fixed number of points independent of the view vector. However, this advantage is partially offset by the fact that a real time system cannot afford a complex interpolation algorithm. The data base complexity problem is still open. However, all conventional CIG suggests that a data base which is limited to planes yields cartoon type imagery without sufficient detail for nap of the earth flight training.

Any CIG system which models the world via available DMA data is going to have an advantage for data base creation. **REAL SCAN** forsees two methods of ground based models. Both models or interpolation means are directly derivable from DMA via computer. It appears that

no one has demonstrated an inexpensive and automated means for producing a high detail ground model made up of planar surfaces when the CIG's computational limits are imposed. **REAL SCAN** does not, and cannot have this problem, since the interpolation algorithms are tied to fixed grid world coordinates (i.e., interpolation is clearly bounded in terms of display resolution). Hence, automated data base generation appears to be the most sizeable advantage that **REAL SCAN** has over conventional CIG especially if one desires display limited resolution.

The use of intermediate projection planes may eliminate this advantage for **REAL SCAN**. Suppose one utilizes complex ground based interpolation formula automatically derived from DMA data. Then this data can be regularly projected through a "view point" (i.e., where the view point is not the eye, but is in the vicinity of the eye) to an intermediate projection plane. At this point the distinction between conventional CIG and **REAL SCAN** seems to vanish, since the scene is now no more than an orderly projection (i.e., memory mapping) from the intermediate planes to the screen coordinates.

All CIG forces a delay time between the time when a view point is defined and when the scene is displayed. The shortest delay, would be approximately one display time, if one assumes the view point is defined at the beginning of the display cycle and the last displayed pixel is used to measure the delay. Conventional CIG utilizes delays of three or more field times. **REAL SCAN** needs no more than two field times. Further, the use of intermediate projection planes suggests that any CIG system, conventional or **REAL SCAN**, can be produced with the optimal single field display time. This occurs because projection (i.e., memory mapped routing) from the intermediate planes can be initiated directly to the display upon calculation of the current or predicted eye point for the display.

Recommendations

The recommendations for further CIG research are separated into three categories, data base, algorithms and architecture/system hardware. It appears that algorithm development, including intermediate projection planes will account for the least amount of effort but yield the most immediate results. Data base research runs the range from interpolation formula to developing culture files. The completion of the data base efforts will require a substantial picture-making effort based on real world data which has been tested on a suitable set of "typical subjects". System hardware recommendations are developed in the context of a preconceived scenario. Immediate hardware development is not recommended at this time. It seems most appropriate to develop a clear data base conversion means and multi elevation algorithms first. Moreover, since VHSIC is developing toward custom procedures for circuit development, it appears most appropriate to develop **REAL SCAN** using these efficient methods.

All recommendations are tasked and estimated for time and manpower.

TASK 1: INTERMEDIATE PROJECTION PLANES

The intermediate projection planes should be incorporated into **REAL SCAN**. This involves projecting a data base to fixed X-Z and Y-Z planes and then projecting these planes to an arbitrarily oriented screen. The location of the planes and the time to perform the plane's update need to be developed along the lines of Appendix C. However, initial simulation results can be obtained by updating the planes once the parallax error (i.e., 1/2 pixel) has been achieved. A sequence of pictures, using the slower update requirement on the intermediate planes, should be made to size the realistic computational savings, estimated as about a factor of 10.

A significant perception related unknown needs to be resolved. Consider a high speed vehicle moving through a data base such that the near field is completely changed every frame rate. That is the near field moves at an angular rate equal or greater than 3600° per sec (i.e., 60° in 1/60 sec). This high a near field rate implies that intermediate projection planes would not provide any saving (i.e., data base computation averaged over 10 or more frames) over calculating the scene anew for each display. However, intermediate projection planes may not provide any computational burden to **REAL SCAN**, since it appears to be only a reordering of the projection computation process. A near field rate of 1800° per sec or less yields an improvement for intermediate projection planes.

However, one can argue that producing a new scene each display time would not allow a trainee to focus or resolve any information in the near field. This suggests that a rapidly changing near field could be simulated with some "appropriate" pattern possessing simply computed gross characteristics and not the near scene detail. Only that part of the scene possessing image coherence from frame to frame would need to be calculated accurately.

A counter argument for high scene detail can also be made. Since people can apparently register subliminal scenes (i.e., single detailed frames inserted into a movie such that an individual does not consciously recognize the single frame in the movie, but yet the individual clearly acts on the information contained in the single frame), then one can argue that the near field, non-coherent, information may be unconsciously processed to yield important nap of the earth flight cues.

If significant image coherence can only occur for near field rates of less than or equal to 600° per sec (i.e., 10° in 1/60 sec), then intermediate projection planes will clearly afford substantial

hardware reductions for CIG systems capable of producing real time display limited imagery.

It is estimated that the algorithm development (i.e., incorporating intermediate planes into REAL SCAN software) will require between 0.25 to 0.5 man-year effort. The creation of appropriate movie frames to measure the efficiency of intermediate projection planes is estimated to require an additional 0.25 to 0.5 man-year.

The creation of a human factors study to help resolve the detail versus subliminal question might consist of the following steps:

1. Create a movie of about two-minute duration (i.e., 24*120 = 2880 frames) having a near field rate of change of about 1800° per sec. It is estimated that each frame will require about 0.5 hr of computer time and if both NTEC and Herndon VAX computer facilities are used for a total of 168 hrs per week, then 2880 frames require about seventeen weeks of effort.

2. Create a movie of the same flight path but whereby the near field, having an apparent motion from 1800° per sec to about 60° per sec (i.e., matching the tracking eye movement rate) is a blur, and intermediate planes are used for the far field (i.e., scene rate of change equal or less than the tracking eye movement rate). It is estimated that such a movie will require about two weeks of effort.

3. Perform a human factors study to determine if a subject can distinguish between the two movies. If some subjects can distinguish between the movies, then use the half interval search approach by picking a suitable blur to near field separator rate, like 320° per sec for the next test. If no subjects can distinguish between the movies, then reduce the near field separator rate by some appropriate factor till the effect is noticeable by about 50% of the subjects. If the tracking eye movement rate corresponds to about 50% of the subjects distinguishing between the movies, then make two new movies one with a separation rate of 120° per sec and another with a separation rate of 30° per sec. Measure the ability of an observer to distinguish movie difference even if the viewer does not know what is different and correlate this with the individual's tracking eye movement rate.

It is estimated that these three efforts will require about two individuals each working half time for about one and a half years if the eye tracking equipment is already available.

TASK 2: ALGORITHM IMPROVEMENTS

The algorithm improvements defined in Appendices E,F,G,H, J and O should be implemented. Then the changes and their improvement on scene computation time should be documented. The effort will involve modifying the current **REAL SCAN** algorithms to satisfy the changes as per the Appendices. Then making sample pictures using the new algorithms for comparison to the old algorithms. Details such as minimum, average, and maximum ground points per pixel should be used to compare the algorithms and to illustrate the capability of the new algorithms.

It is estimated that this software and documentation effort will require about two half-time people for one year to one and a half years.

TASK 3: OPERATION COUNT

The current software should be sized to illustrate how many and of what type instructions are needed to make **REAL SCAN** pictures. This effort will guide further algorithm improvement and serve to measure and compare any algorithm improvement. The instruction count should be directly related to the hardware model of **REAL SCAN** (i.e., the **REAL SCAN** architecture of Section V). Hence, instruction type should be listed by *fundamental type and by data dimension*, since we propose an integer machine.

The instruction count for the PG* software defined in Section II is largely complete. The block data base yields about $1.8*10^9$ integer instructions per picture.

It is estimated that about 3 months for one half-time individual would be required to code the current date base algorithms. About another half man-month would be required to document results. About one man-month will be required to code and document any algorithm improvements, such that the new instruction count can be compared to the original algorithm.

TASK 4: MULTIPLE ELEVATION ALGORITHMS

The ability to handle Eiffel tower data is not part of **REAL SCAN**. Methods for creating and handling bridges, trees, and other multivlued elevation models must be developed. These methods must be tested for simplicity. The use of intermediate projection planes suggests that current CIG methods may be incorporated to define an intermediate projection plane having "holes" or transparent regions.

The first page of this effort consists of a listing of methods available to handle multivalued elevations. One needs to investigate ways to count data gaps other than coding a point as possessing color, partial transparency or total transparency, since these gaps could

place a substantial memory requirement on the intermediate projection planes. Finally, the least complex methods and the methods producing the most real world derived/simulated detail need to be coded and evaluated.

This effort is estimated as about one individual half-time for about two years to test and document an efficient integer multi-elevation method.

TASK 5: MODEL REAL WORLD FEATURES

The entire data base for REAL SCAN has been invented. A data base referenced to a real world typical gaming area is needed. The data base should possess typical features for ocean, river, lake, bay, field, forest, mountain, valley, and desert; as well as man-made features. There appear to be two methods for initially transforming real world elevation maps, U.S. survey maps, and photo-reconnaissance plates into a mathematical data base. Each method requires a substantial software effort.

One method uses photo-reconnaissance plates to obtain detail information. The high resolution detail can be read automatically on a SPEC-SCAN and stored on magnetic tape. This information can be filtered for derived sun angle. If stereo equipment is available, then the cultural feature's local elevation can be measured, rather than estimated. However, it is clear that any method of real world data derivation for such features as trees, fields or deserts which change daily as they flower or the weather effects them. Hence, these features need not be exact. Therefore, typical feature characteristics are desirable, both to conserve memory and computation.

Part of the real world high resolution data gathering task should be as follows:

1. Mathematically define at least three kinds of trees. That is their color versus season, wind, and other pertinent parameters. The tree model should be parametric such that the detail of each tree need not be stored. Color and elevation are required. Various mathematical models need to be sudied so a choice of the simplest can be made. Such modeling means as texturing clear ellipses and summing elevation functions need to be evaluated for model quality and simplicity.

2. Mathematically define, perhaps by memory maps, at least three man-made structures. These models should also be parametric so that cities of houses or office building could be constructed.

3. Mathematically define at least three real world field textures such as wheat, corn, and grass. These textures have

height and color information. Various models need to be evaluated for power and simplicity. Texture is imagined different from tree feature because of the range of elevation information and the rate of daily change possible in the perceived scene.

4. Mathematically define features of large extent such as rivers, roads, and airport runways. These must allow small detail such as waves or skid marks yet be simple and parametric.

The mathematical modeling efforts outlined above require the discovery of simple characteristics wherever possible. These models require simple methods for representing such complex high detail features as trees and corn.

Completion of such an effort suggests that a course elevation map could be "painted" (i.e., both elevation and color) with derived detail. However, one could still use the origianl **REAL SCAN** concept of growing the detail offline into a 10^{12} byte gaming area data base. Each of these modeling efforts is estimated as a minimal two-man-year effort with no guarantee that simple, realistic methods suitable for real time CIG will be discovered.

For example, the following simpler efforts have been initiated in the body of this report and its Appendices. These efforts are judged simpler because some part of the modeling effort has been defined and in many cases partially tested. The efforts are:

1. Following Appendix D, determine the most efficient means of using a regular grid data base to represent terrain elevation. Assume terrain features can be created by summing a small number of coded features. However, if this method is used, a realistic method for treating any boundary discontinuity must be solved. Hence, this effort resolves the choice of polynomial for single value elevation mapping and determines when and if feature summing is better than a more complex interpolation function.

   It is estimated that one person half time for one-half year will be required to code each interpolation formula chosen for evaluation. Currently the straight line interpolation function and the Overhauser-Coons interpolation function have been coded. It is estimated that about one-man-year of effort would be required to create sample photos, to create movies to compare up to three new interpolation formula, and to write up each result (i.e., two years for one person half time).

2. Solve the boundary problem when summing features to create terrain or other culture. The boundary problem occurs at region boundaries classified with different features. Different regions require the summing of different feature functions, but their elevations are not guaranteed to be equal at the boundary. Such methods as boundary gradient functions, clipping function, etc. need to be investigated. The noise derived data base of Section III and Appendix I would be appropriate to this investigation since one is guaranteed variability yet statistical control of both elevation and slope. A simple solution would aid the development of a culture feature data base allowing superposition of files and interpolation formula or memory map to create detail.

   It is estimated that about six months of a half time individual will be required to code and test each boundary solution. Another six months of half time effort is then estimated as required to document the result.

3. Evaluate terrain and feature modeling distributions following the methods of Appendix I, the color tests of Appendix BC, and the software in Appendices DA through DH. These tests would be subjective concerning the quality of the modeling, since systematic trial and error procedures are required to test distribution modeling of significant real world features.

   Machine measures of feature color, color and intensity distribution, and elevation distribution of real world features to be modeled is desired. However, the effects of sun shadowing need to be removed from the measure. Hence, it appears that a combination of subjective comparisons of distribution parameters to real world features, and detailed measurement of real world features should be combined.

   The result of such a study would be parametric elevation and color mathematical methods for representing real world features, to high detail, and appropriate to a real time CIG system.

   It is estimated that the cateloguing and documenting of distributions which can be generated via Appendix I will take about two years of one half-time-person. Special equipment, like a SPEC SCAN, is required to measure real world photographs for their color and distribution information. Assuming a color reading SPEC SCAN were available and operational, then it is estimated that between six months to one year of a half-time-person would be required to create the software to measure color and distributions. It is estimated that about

another year of half-time-person effort would be required to measure and fully document the measurements of about ten features. However, the effect of sun shadowing would not yet be clearly resolved from the measurements.

It is estimated that about one-man-year of effort, involving iterative simulation and evaluation, would be required to resolve sun shadowing effects. Real world distributions of features need be available, multi-elevation modeling software needs to be operating, and sun shadowing algorithms need be available to operate on the models to duplicate the measured characteristics.

One can see that a few judicious choices (lucky guesses) of methods can make a substantial impact in the time spent and ultimate mathematical simplicity to model features.

4. Evaluate the use of signed parameters to create region boundary designations. Current boundary designations are based either on elevation and slope, and/or a number patch which covers a large square of world (i.e., large compared to the feature's detail). These methods do not allow modeling of rivers or streams. Further, all field to forest boundaries occur along orthogonal straight lines due to the region designator number patches. This work should impact both high detailed hierarchial data bases or feature modeled data bases, since the hierarchial data bases are constructed on a regular grid. Hence, irregular features passing through the regular grid currently pose a modeling problem.

   It is estimated that about one year of a half-time-person will be required to test and evaluate signed parameters for modeling irregular region boundaries.

5. Evaluate the methods of conventional CIG modeling. Conventional CIG stores data using an object designator, breaking the objects into simple parts, and then defining the object parts by their set of connected world coordinates. **REAL SCAN** allows the world coordinate to be the address to the elevation and color data. A simple means of incorporating conventional CIG data bases into **REAL SCAN** is desired. Some simple multi-elevation models are the desired result. Since the data base is evaluated in fixed ground coordinates, **REAL SCAN** offers a larger choice of function or interpolation methods to represent multi-elevation models, whether trees, tanks, buildings or airplanes. However, since conventional CIG has been able to model many man-made objects very well, these methods should be evaluated for their ease of incorporation into **REAL SCAN**.

It is estimated that about two man-years of effort are required to incorporate at least one tank or airplane and to determine how to effectively employ conventional CIG models in REAL SCAN.

However, if intermediate projection planes prove to be as effective and easy to implement as suspected, then this task only needs to resolve the differences between REAL SCAN and conventional CIG addressing methods.

TASK 6: FILTER EVALUATIONS

The current REAL SCAN effort has shown negligible aliasing problems associated with about four ground points per display pixel. Since conventional CIG suggests that effective anti-aliasing requires real world subdivision to about 16 ground points to the pixel, aliasing needs to be further evaluated. It is possible that the REAL SCAN method (i.e., data base in ground coordinates) need be filtered no better than is currently done. Such features as hill edges are naturally oversampled and hence "properly" anti-aliased. Further, far field edges are also sampled and contribute to anti-aliasing. However, highway lines are only slightly over sampled and a careful scrutiny of Figure 57 demonstrates some aliasing.

The picture quality needs to be carefully documented as a function of the point spread filter, the number of points accumulated on average to a pixel; high pass-low pass filtering over edges and evaluated as a function of the total display complexity (i.e., terrain clutter, sharp lines and edges, narrow lines like wings, and other significant scene features).

It is estimated that the software initiated will require about six months of a half-time-person and the evaluation and documentation will require about one to one and a half man-year of effort.

If it is discovered that effective anti-aliasing requires more than the currently used four ground points per pixel, then the memory required for intermediate projection planes would be accordingly increased. For instance, if three points are required per pixel edge, then nine times the display would be the minimum memory required. If four points are required per pixel length, then 16 times the display's detail would be the minimum memory required.

The sampling theorem suggests that the intermediate plane need store no more than twice the display density per length. This is the proposed intermediate plane structure. It requires an acceptable requirement of four times the memory of the display's detail.

TASK 7: CREATE COLOR MOVIES OF NAP OF THE EARTH FLIGHT

A sequence of motion pictures should be created using the available equipment (i.e., currently **DICOMED** but upgrading to IKONAS TV equiment). These pictures should evaluate filter algorithms, data base methods, processing algorithm improvements, and generally serve to show the capability and progress of **REAL SCAN**.

Using **DICOMED** equipment, each movie requires about three months of a half-time-person. This time is spent acquiring a few test frames to judge the flight trajectory, then storing the frames on magnetic tape, and finally writing the frames to 35mm film. The film is then processed and reduced to 16mm for viewing.

The IKONAS equipment will allow direct transfer to 16mm film and also TV evaluation of short motion picture segments because of the disk storage incorporated into the system.

SYSTEM HARDWARE RECOMMENDATIONS

Two data base concepts have been demonstrated as workable, but substantial work remains to be done to define efficient data base compression algorithms. Hence, one must admit that the detail of the **REAL SCAN** data base has not been thoroughly defined. Therefore, it does not appear appropriate to attempt a complete **REAL SCAN** hardware development at this time. However, detail designs for some functions can be initiated now, recognizing that even well defined integer operations, such as projection will still evolve.

The complexity of **REAL SCAN** has been estimated as about twenty cabinets of digital electronics. This estimate assumes that one carefully sizes and scales the algorithms to operate on integer arithmetic of the resolution required for the display. If one assumes that 20 cabinets of properly sized hardware is required to do a job at 12 bit resolution, then if the job were performed at 16 bit resolution, one could anticipate an additional 33% more hardware or a total of about 27 caibnets. If one were to use "standard" 32 bit parts which had been optimized because of million piece production quantities, then one might anticipate 30 to 40 cabinets of hardware. Further, the power of a 32 bit microcomputer's instruction set would be largely unused, since short runs of code appear to offer an optimal pipeline structure.

Current VHSIC efforts appear to be directed at functional deign and automated test procedures based on standard building block circuits whenever possible. Hence, many parallel efforts, but in different technologies, appear to be working toward the same goals of feasible production runs in the thousands and turnaround time of six months or less from design to a tested functioning custom IC.

Such a scenario suggests that efficient CIG hardware should be designed for a parallel pipelined structure, that the hardware be based upon the desired display resolution, and that a replicated modular structure be the form of the hardware. A standard building block approach implies that complex custom designs will be feasible in the near future (1985 to 1987). Further, since the hardware is dependent upon complexity, algorithms which have been sized and made simple can be efficiently constructed with fewer integrated circuits than if one attempts to use general purpose microcomputer parts.

A general purpose microcomputer design appears to yield the simplest and most economical digital designs for pipelines that can run sequences of 1000 or more instructions where the instructions average about 1 microsec for execution. However, if the above scenario is correct, then custom VHSIC will be available just about when a microcomputer system could be developed for CIG. One should not overlook the possibility of transferring a microcomputer design to VHSIC. However, if the algorithms have not been carefully analyzed to determine the simplest and fastest solutions; and sized to define the resolution specified word size, then one would not expect a substantial reduction in the amount of hardware required to implement such a CIG system. If the above scenario is correct, then by 1985 to 1987 many facilities will be available for custom design, fabrication and test. Further, the two to three year delay between initial design and final working IC will be trimmed to about six months. One should note, that all the algorithm development and system architecture completed to date has been directed at simplicity. One still requires processing parts, but the range of instruction required has been reduced. One still requires memories, but the addition of simple processing functions to the memories yields a modular pipelined design. This work is in keeping with the above scenario.

There does not appear to be any CIG system producing display limited images in real time. There are a few real time systems producing scenes from objects defined by planar surfaces. If one seeks to be able to simulate real world scenes having a resolution limited by the display means then all the data base work recommended must be successfully completed. The hardware for implementing real time algorithms will be simplest if the hardware is properly sized. One should note, that the addition of one bit to represent a number increases resolution by a factor of two. Hence, accurately sizing any real time CIG appears to be a worthwhile first step before hardware implementation.

More algorithmic work remains to be done on **REAL SCAN** (i.e., multi-valued elevation, clearly defined optimal data base mathematics, and evaluation of intermediate projection planes). In keeping with the VHSIC scenario offered, of custom design with six-month working turnaround in the 1985 to 1987 time frame, the following hardware tasks are outlined.

TASK 8: TRACK VHSIC DEVELOPMENTS

This effort involves visiting VHSIC fabrication facilities, reading technical and sales literature about the design facilities, and eventual sample designs of REAL SCAN modules. At least two people should be assigned to this task. They should become thoroughly familiar with design facilities (i.e., design software packages allowing automated testing). Then the design of a comparable module should be completed using two competing design packages. This will allow accurate comparison between the design packages, such that we can decide when the VHSIC design of REAL SCAN should begin in earnest and which particular facilities, or technology, will most nearly satisfy our needs. Many sophisticated VHSIC facilities offering building block designs are anticipated to be nearly simultaneously available.

It is estimated that about six months of half-time effort will be required to familiarize two people with VHSIC and current technologies, such as gate arrays. These people would then maintain currency but contribute to the REAL SCAN design effort.

TASK 9: DETAILED REAL SCAN DESIGN

Since it is anticipated that rapid development VHSIC design tools will not exist for about two years, one needs to refine hardware designs of the REAL SCAN system and compare various modular structures for implementing the parallel pipeline structure needed to implement a REAL SCAN solution. Further, one needs to keep abreast of VHSIC developments and the evolution of current building block design tools, just in case these facilities become available sooner.

The hardware designs should be based on a well documented digital design language such as DDL, CDL, or AHPL. Further, the hardware designs should be modular, in terms of both display resolution and display type (i.e., spherical display surface, planar surface display of given dimension, etc.). The designs should be highly efficient for such operations as projection and visibility; and the designs should be flexible (programmable) yet efficient for such operations as data base interpolation and function determination.

Including the two half-time people of Task 8, it is estimated that about three half-time people would be required for about three years to complete the detail design of REAL SCAN.

Recommendation Summary

The tasks identified for further research and development have been summarized in Table 15. All efforts have been given in terms of half-time-persons, since graduate students cannot devote full-time to research efforts.

TABLE 15. RECOMMENDED TASKS

	TASK NAME	HALF-TIME PERSONS	TASK DURATION (YR.)
1.	Intermediate Projection Planes	2	1
	Projection Planes Human Factors	2	1.5
2.	Algorithm Improvements	2	1 to 1.5
3.	Operation Count	1	1
4.	Multi-Elevation Algorithm	1	2
5.	Model World Features	4*	2*
	Interpolation	2	1
	Boundary Problem	2	1
	Feature Catalog	2	1
	Sun Shadowing	1	1
	Rivers/Streams/Road	1	1
	Conventional CIG Evaluation	2	2
6.	Point Spread Filter	2	1
7.	Movies	-	2
8.	VHSIC Review	2	-
9.	REAL SCAN Hardware	3	3

*Potential long-range projects

REFERENCES

1. Roberts, L. G., "Machine Preception of Three Dimensional Solids", TR315, MIT Lincoln Laboratory, May, 1963.

2. Sutherland, I.E.; Sproull, R. F.; and Schumacher, R. A. "A Characterization of Ten Hidden-Surface Algorithms", Computing Surveys, Vol. 6, No. 1, March, 1974, pp. 1-55.

3. Breglia, D. R., "Automating CIG Data Base Development", Technical Report NAVTRAEQUIPCEN IH-318, Naval Training Equipment Center, Orlando, Florida.

4. Defense Mapping Agency "Product Specifications for Digital Landmass System (DLMS) Data Base", July, 1977.

5. "Videodisc Based Storage Technology", Computer, Vol. 13, No. 6, pp. 87, June, 1980.

6. Greenly, R. and Marchegiani, D. "Digital Radar Landmass Simulation", Proceeding of the Fourth Annual Naval Training Device Center/Industry Conference, pp. 131-141, November, 1969.

7. Hoog, T.; Dahlberg, R.; and Robinson, R. "Project 1183 - An Evaluation of Digital Radar Landmass Simulation", Proceedins of the Seventh NAVTRAEQUIPCEN/Industry Conference, pp. 55-79, November, 1974.

8. Ammon, G. "Wide band Optical Disc Data Recorder Systems", SPIE Vol. 200 Laser Recording and Information Handling, pp. 64-72, 1979.

9. Unruh, J.; Alspaugh, D.; and Mikhail, E. "Image Simulation from Digital Data", Proceedings of American Congress on Surveying and Mapping, 1977.

10. Schater, B. "Computer Generation of Full Colored Textured Terrain Images", Proceedings of 1st Tri-Service/Industry Training Equipment Conference, pp. 367-374, November, 1979.

11. Stenger, T.; Dungan, W.; and Reynolds, R. "Computer Image Generation Texture Study", AFHRL-TR-79-2, ADA 074019, August, 1979.

12. Bresenham, J. "Algorithm for Computer Control of a Digital Plotter", IBM Systems Journal, Vol. 4, No. 1, pp. 25-30, 1965.

13. Kell, R. O.; Bedford, A.V.; and Trainer, M. A. "An Experimental Television System", Proceedings of IRE, Vol. 22, No. 11, p. 1247, November, 1934.

14. Csuri, C.; Hackathorn, R.; Parent, R.; Carlson, W.; and Howard, M. "Towards an Interactive High Visual Complexity Animation System", SIGGRAPH '79 Proceedings, p. 289-299, August, 1979.

15. Marshall, R.; Wilson, R.; and Carlson, W. "Procedure Models for Generating Three-Dimensional Terrain", SIGGRAPH '80 Proceedings, pp. 254-259, July, 1980.

16. Gardner, G. "Computer Generated Texturing to Model Real World Features", Proceedings of the 1st Interservice/Industry Training Equipment Conference, pp. 239-245, November, 1979.

17. Brewer, J. and Anderson, D. "Visual Interaction with Overhauser-Coons Curves and Surfaces", Computer Graphics, Vol. 11, No. 2, pp. 133-137, Summer 1977.

18. Crow, F. "The Use of Gray Scale for Improved Raster Display of Vectors and Characters", SIGGRAPH '80 Proceedings, pp. 286-293, July, 1980.

19. Crow, F. "The Aliasing Problem in Computer Generated Shaded Images", Communications of the ACM, November, 1977.

20. Catmull, E. "A Hidden-Surface Algorithm with Anti-Aliasing", Computer Graphics, pp. 6-10, 1979.

21. Weiman, C. "Continuous Anti-Aliased Rotation and Zoom of Raster Images", SIGGRAPH '80 Proceedings, pp. 286-293, July, 1980.

22. Oppenheim, A. and Schafer, R. Digital Signal Processing. Prentice Hall, Englewood Cliffs, New Jersey, 1975.

23. "Electronic Products" Vol. 24, No. 10, Hearst Business Communications, Inc., Garden City, New York, p. 80, January, 1982.

24. Chestnut, H. and Mayer, R. "Servomechanisms and Regulating System Design Vol. 1", John Wiley & Son, New York, New York, pp. 226-228, 1951.

25. D'Azzo, J. and Houpis, C. "Feedback Control System Analysis and Synthesis", McGraw Hill, New York, New York, pp. 366-369, 1966.

BIBLIOGRAPHY

Algorithms

Atherton, P.; Weiler, K.; and Greenberg, D. "Polygon Shadow Generation", SIGGRAPH '78 Proceedings, Vol. 12, No. 3, pp. 275-281, August, 1978.

Badler, N. and Bajcsy, R. "Three-Dimensional Representations for Computer Graphics and Computer Vision", SIGGRAPH '78 Proceedings, Vol. 12, No. 3, pp. 153-160, August, 1978.

Blinn, J. and Newell, M. "Clipping Using Homogeneous Coordinates", SIGGRAPH '78 Proceedings, Vol. 12, No. 3, pp. 245-251, August, 1978.

Bresenham, J. "A Linear Algorithm for Incremental Digital Display of Circular Arcs", Communications of the ACM, Vol. 20, No. 2, pp. 100-106, February, 1977.

Bunker, W. and Heartz, R. "Perspective Display Simulation of Terrain", Air Force Human Resources Laboratory, AFHRL-TR-76-39, June, 1976.

Catmull, E. "A Hidden-Surface Algorithm with Anti-Aliasing", SIGGRAPH '78 Proceedings, Vol. 12, No. 3, pp. 6-10, August, 1978.

Catmull, E. and Smith, A. R. "3-D Transformations of Images in Scanline Order", SIGGRAPH '80 Proceedings, Vol. 14, No. 3, pp. 279-285, July, 1980.

Clark, J. H. "Hierarchical Geometric Models for Visible Surface Algorithms", Communications of the ACM, Vol. 19, No. 10, pp. 547-554, October, 1976.

Coons, S.A. "Transformations and Matrices", Course Notes No. 6, University of Michigan, November, 1969.

Crow, F. C. "Shadow Algorithms for Computer Graphics", Computer Graphics, Vol. 11, No. 2, pp. 242-248, Summer 1977.

Eastman, C. and Yessios, C. "An Efficient Algorithm for Finding the Union, Intersection and Differences of Spatial Domains", Research Report No. 31, Institute of Physical Planning, Carnegie-Mellon University, 1973.

Franklin, W. "Evaluation of Algorithms to Display Vector Plots on Raster Devices", Computer Graphics and Image Processing, Vol. 11, No. 4, pp. 377-397, December, 1979.

Franklin, W. R. "A Linear Time Exact Hidden Surface Algorithm", SIGGRAPH '80 Proceedings, Vol. 14, No. 3, pp. 117-123, July, 1980.

Hamlin, Jr., G. and Gear, C. "Raster-Scan Hidden Surface Algorithm Techniques", Computer Graphics, Vol. 11, No. 2, pp. 206-213, Summer, 1977.

Knowlton, K. and Cherry, L. "ATOMS-A Three-D Opaque Molecule System", Computers and Chemistry, Vol. 1, No. 3, pp. 161-166, 1977.

Mahl, R. "Visible Surface Algorithms for Quadric Patches", IEEE Trans. Computers, Vol. C-21, No. 1, pp. 1-4, January, 1972.

Pitteway, M. and Watkinson, D. "Bresenham's Algorithm with Grey Scale", Communications of the ACM, Vol. 23, No. 11, pp. 625-626, November, 1980.

Sechrest, S. and Greenberg, D. "A Visible Polygon Reconstruction Algorithm", SIGGRAPH '81 Proceedings, Vol. 15, No. 3, pp. 17-27, August, 1981.

Sproull, R. "Using Program Transformations to Derive Line-Drawing Algorithms", Technical Report, Carnegie-Mellon University, Computer Science Department, 1981.

Sproull, R. and Sutherland, I. "A Clipping Divider", AFIPS Conf. Proc., Vol. 33, 1968 FJCC, pp. 765-775.

Sutherland, I.; Sproull, R.; and Schumacher, R. "A Characterization of Ten Hidden-Surface Algorithms", Computing Surveys, Vol. 6, No. 1, pp. 1-55, March, 1974.

Weiler, K. and Atherton, P. "Hidden Surface Removal Using Polygon Area Sorting", Computer Graphics, Vol. 11, No. 2, pp. 214-222, Summer 1977.

Weiman, C. F. R. "Continuous Anti-Aliased Rotation and Zoom of Raster Images", SIGGRAPH '80 Proceedings, Vol. 14, No. 3, pp. 286-293, July, 1980.

Whitted, T. "An Improved Illumination Model for Shaded Display", Communications of the ACM, Vol. 23, No. 6, pp. 343-349, June, 1980.

Williams, L. "Casting Curved Shadows on Curved Surfaces", SIGGRAPH '78 Proceedings, Vol. 12, No. 3, pp. 270-274, August, 1978.

Architecture

Booth, J. M. and Schroeder, J. B. "Design Considerations for Digital Image Processing Systems", Computer, p. 15, August, 1977.

Fuchs, H. and Johnson, B. "An Expandible Architecture for Video Graphics", Proceedings of the Sixth Symposium on Computer Architecture, April, 1979.

Gardner, G. "A Computer Image Generation System with Efficient Image Storage", SPIE Technical Symposium East '79, April, 1979.

Kaplan, M. and Greenberg, D. "Parallel Processing Techniques for Hidden Surface Removal", SIGGRAPH '79 Proceedings, Vol. 13, No. 2, pp. 300-307, August, 1979.

Parke, F. I. "Simulation and Expected Performance Analysis of Multiple Processor Z-Buffer Systems", SIGGRAPH '80 Proceedings, Vol. 14, No. 3, pp. 48-56, July, 1980.

Data Base Methods

Artzy, E.; Frieder, G.; and Herman, G. "The Theory, Design, Implementation and Evaluation of a Three-Demensional Surface Detection Algorithm", SIGGRAPH '80 Proceedings, Vol. 14, No. 3, pp. 2-9, July, 1980.

Blinn, J. F. "Computer Display of Curved Surfaces", Dissertation, University of Utah, 1978.

Blinn, J. F. "Models of Light Reflection for Computer Synthesized Pictures", SIGGRAPH '77 Proceedings, Vol. 11, No. 2, pp. 192-198, Fall 1977.

Blinn, J. F. "A Scan Line Algorithm for Displaying Parametrically Defined Surfaces", SIGGRAPH '78 Proceedings, Vol. 12, No. 3, August, 1978.

Blinn, J. F.; Lane, J. M.; Carpenter, L. C.; and Whitted, T. "Scan Line Methods for Displaying Parametrically Defined Surfaces", Communication of the ACM, Vol. 23, No. 1, pp. 23-34, January, 1980.

Blinn, J. F. and Newell, M. E. "Texture and Reflection in Computer Generated Images", Communication of the ACM, Vol. 19, No. 10, pp. 542-547, October, 1976.

Bunker, M. "CIG Translucent Face Simulation Provides Multiple Benefits", Proceedings of Interservice/Industry Training Equipment Conference, pp. 229-238, November, 1979.

Bunker, W. and Ferris, N. "Computer Image Generation Imagery Improvement: Circles, Contours, and Texture", AFHRL-TR-77-66, September, 1977.

Carpenter, L. "Computer Rendering of Fractal Curves and Surfaces", SIGGRAPH '80 Proceedings, Vol. 14, No. 3, p. 109, July 1980.

Catmull, E. "Computer Display of Curved Surfaces", Proceedings of the Conference on Computer Graphics, Pattern Recognition, and Data Structures, IEEE Catalog No. 75 CH0981-1C, pp. 11-17, May, 1975.

Catmull, E. "A Subdivision Algorithm for Computer Display of Curved Surfaces", UTEC-CSC-74-133, PhD Thesis, Computer Science Department, University of Utah, December, 1974.

Cohen, D. "On Linear Difference Curves", Proceedings International Symposium CG-70, Vol. 1, Brunel University, Uxbridge, England, April, 1970.

Coons, S. "Surfaces for Computer-Aided Design of Space Figures", Report MAC-M-255, Project MAC, MIT, Cambridge, Massachusetts, January, 1964.

Coons, S. "Surfaces for Computer-Aided Design of Space Forms", Report MAC-TR-41, Project MAC, MIT, Cambridge, Massachusetts, June, 1967.

Csuri, C.; et al. "Towards an Interactive High Visual Complexity Animation System", SIGGRAPH '79 Proceedings, Vol. 13, No. 2, pp. 289-299, August, 1979.

Defense Mapping Agency, Product specifications for digital landmass system (DLMS) data base. PS/ICD (EFG)/100, July, 1977.

Dugan, Jr., W. "A Terrain and Cloud Computer Image Generation Model", SIGGRAPH '79 Proceddings, Vol. 13, No. 2, pp. 143-147, August, 1979.

Earnshaw, R. "Line Generation for Incremental and Raster Devices", SIGGRAPH '77 Proceedings, Vol. 11, No. 2, pp. 199-205, July, 1977.

Feibush, E. A.; Levoy, M.; and Cook, R. L. "Synthetic Texturing Using Digital Filters", SIGGRAPH '80 Proceedings, Vol. 14, No. 3, pp. 294-301, July, 1980.

Forrest, A. "On Coons and Other Methods for Representation of Curved Surfaces", Computer Graphics and Image Processing, Vol. 1, No. 4, pp. 341-359, 1972.

Forrest, A. "Recent Trends in Computer-Aided Geometric Design", Proceedings of the 1978 Interactive Techniques in Computer-Aided Design, IEEE Catalog No. 78 CH1289-8C, pp. 141-146, 1978.

Forrest, A. "On the Rendering of Surfaces", SIGGRAPH '79 Proceedings, Vol. 13, No. 2, pp. 253-257, August, 1979.

Fournier, A. and Fussell, D. "Stochastic Modeling in Computer Graphics", SIGGRAPH '80 Proceedings, Vol. 14, No. 3, p. 108, July, 1980.

Fowler, R. and Little, J. "Automatic Extraction of Irregular Network Digital Terrain Models", SIGGRAPH '79 Proceedings, Vol. 13, No. 2, pp. 199-207, August, 1979.

Freeman, H. "On the Encoding of Arbitrary Geometric Configurations", IRE Trans. Electron Computers, EC-10, pp. 260-268, 1961.

Gardner, G. "Computer-Generated Texturing to Model Real-World Features", Proceedings of the Interservice/Industry Training Equipment Conference, pp. 239-246, November, 1979.

Gardner, G. "Computer Image Generation of Curved Objects for Simulator Displays", Eleventh NTEC-Industry Conference Proceedings, November, 1978.

Horn, B. "Hill-Shading and the Reflectance Map", Proceedings: Image Understanding Workshop, pp. 79-120, April, 1979.

Jancaitis, J. "Modeling and Contouring Irregular Surfaces Subject to Constraints", Army Engineer Topographic Laboratories, ETL-CR-74-19, January, 1975.

Kay, D. S. and Greenberg, D. "Transparency for Computer Synthesized Images", SIGGRAPH '79 Proceedings, Vol. 13, No. 2, pp. 158-165, August, 1979.

Lang, C. "A Three-Dimensional Model Making Machine", Cambridge University Computer-Aided Design Group, Doc. 74, September, 1972.

Marshall, R.; Wilson, R.; and Carlson, W. "Procedure Models for Generating Three-Dimensional Terrain", SIGGRAPH '80 Proceedings, Vol. 14, No. 3, pp. 154-162, July, 1980.

Newell, M. E. "The Utilization of Procedure Models in Digital Image Synthesis", UTEC-CSC-76-218, Computer Science Department, University of Utah, 1975.

Pavlidis, T. "Contour Filling in Raster Graphics", SIGGRAPH '81 Proceedings, Vol. 15, No. 3, pp. 29-36, August, 1981.

Pavlidis, T. "Filling Algorithms for Raster Graphics", Computer Graphics Image Proceedings 10, pp. 126-141, 1979.

Reddy, D. R. and Rubin, S. "Representation of Three-Dimensional Objects", CMU-CS-78-113, Department of Computer Science, Carnegie-Mellon University, April, 1978.

Roese, J. and McCleary, L. "Stereoscopic Computer Graphics for Simulation and Modeling", SIGGRAPH '79 Proceedings, Vol. 13, No. 2, pp. 41-47, August, 1979.

Schafer, R. and Rabiner, L. "A Digital Signal Processing Approach to Interpolation", Proceedings of the IEEE, Vol. 61, pp. 692-702, 1973.

Sproull, R. "Review", Computing Reviews, p. 500, November, 1979.

Yan, J. "Real-Time Generation and Smooth Shading of Quadric Surfaces", Proceedings of the Interservice/Industry Training Equipment Conference, pp. 247-260, November, 1979.

Yessios, C. "Computer Draftng of Stones, Wood, Plant and Ground Materials", SIGGRAPH '79 Proceedings, Vol. 13, No. 2, pp. 190-198, August, 1979.

General

Andrews, H. F. "Digital Image Processing", IEEE Spectrum, pp. 38-49, April, 1979.

Castleman, K. R. Digital Image Processing. Englewood Cliffs, New Jersey: Prentice-Hall, 1979.

Crow, F. C. "The Aliasing Problem in Computer-Generated Shaded Images", Communications of the ACM, Vol. 20, No. 11, pp. 799-805, November, 1977.

Crow, F. C. "The Use of Gray Scale for Improved Raster Display of Vectors and Characters", SIGGRAPH '78 Proceedings, Vol. 12, No. 3, SIGGRAPH '78 Proceedings, Vol. 12, No. 3, pp. 1-5, August, 1978.

Faintich, M. "Digital Scene and Image Generation", Naval Weapons Laboratory Technical Report, TR-3147, 1974.

Forrest, A. R. "A Computer Peripheral for Making Three-Dimensional Models", Automatisme 19, June-July, 1976.

Forrest, A. R. "Coordinates, Transformations, and Visualization Techniques", Computer Laboratory CAD Document 45, University of Cambridge, June, 1969.

Freeman, H., "Computer Processing of Line-Drawing Images", Computing Surveys G-1, pp. 57-97, March, 1974.

Gupta, S. and Sproull, R. "Filtering Edges for Gray-Scale Displays", SIGGRAPH '81 Proceedings, Vol. 15, No. 3, pp. 1-5, August, 1981.

Haas, M. and Guldenpfennig, P., "The Influence of Full-Mission Simulation on Visual System Capability", Proceedings of the Interservice/Industry Training Equipment Conference, pp. 53-56, November, 1979.

Hatfield, L. and Herzog, B. "Graphics Software - from Techniques to Principles", IEEE Computer Graphics and Applications, Vol. 2, No. 1, pp. 59-80, January, 1982.

Kajiya, J. and Ullner, M. "Filtering High Quality Text for Display on Raster Scan Devices", SIGGRAPH '81 Proceedings, Vol. 15, No. 3, pp. 7-16, August, 1981.

Lippman, A. "Movie-Maps: An Application of the Optical Videodisc to Computer Graphics", SIGGRAPH '80 Proceedings, Vol. 14, No. 3, pp. 32-42, July, 1980.

McKinnon, M. and Raptis, D. "Aircraft Simulators: Recent Improvements and Areas of Research", Proceedings of the Interservice/Industry Training Equipment Conference, pp. 91-98, November, 1979.

Metzger, R. "Computer Generated Graphic Segments in a Raster Display", Proceedings AFIPS 1969 SJCC, AFIPS Press, Montvale, New Jersey, pp. 161-172.

Newell, M.; Newell, R.; and Sancha, T. " A New Approach to the Shaded Picture Problem", ACM Proceedings, 1973.

Newman, W. and Sproull, R. Principles of Interactive Graphics. Second edition, New York: McGraw-Hill, 1979.

Parent, R. and Chandrasekaran, B. "Moulding Computer Clay", Pattern Recognition and Artificial Intelligence. edited by C. H. Chen, New York: Academic Press, pp. 86-107, 1977.

Pavlidis, T. Structural Pattern Recognition. Berlin-Heidelberg, New York: Springes Verlag, 1977.

Roberts, L. "Homogeneous Matrix Representation and Manipulation of N-Dimensional Constructs", Report MS1405, MIT Lincoln Laboratory, May, 1965.

Rogers, D. F. and Adams, J. A. Mathematical Elements For Computer Graphics. New York: McGraw-Hill, 1976.

Rosenfeld, A. and Kak, A. Digital Picture Processing. New York: Academic Press, 1976.

Rubin, S. M. and Whitted, T. "A 3-Dimensional Representation for Fast Rendering of Complex Scenes", SIGGRAPH '80 Proceedings, Vol. 14, No. 3, pp. 110-116, July, 1980.

Schachter, B. "Computer Generation of Full Colored Textured Terrain Images", Proceedings of the Interservice/Industry Training Equipment Conference, pp. 367-374, November, 1979.

Shapiro, L. "Data Structures for Picture Processing", SIGGRAPH '78 Proceedings, Vol. 12, No. 3, pp. 140-146, 1978.

Sutherland, I. E. and Hodgeman, G. W. "Reentrant Polygon Clipping", Communications of the ACM, June, 1974.

Thompson, J. "Straight Lines and Graph Plotters", Computer Journal, Vol. 4, No. 3, p. 227, 1964.

Warnock, J. E. "The Display of Characters Using Gray Level Sample Arrays", SIGGRAPH '80 Proceedings, Vol. 14, No. 3, pp. 302-307, July, 1980.

Vision or Perception

Bajcsy, R. "A Computational Structure for Color Perception", Moore School of EE Technical Report, University of Pennsylvania, Philadelphia, 1975.

Biberman, L. Perception of Displayed Information. New York: Plenum Press, pp. 3-4, 1973.

Blackwell, H. R. "Contrast Thresholds of the Human Eye", Journal of the Optical Society of America, Vol. 36, No. 11, pp. 624-643, November, 1946.

Campbell, F. "The Human Eye as an Optical Filter", Proceedings of the IEEE SG-6, pp. 1009-1014, June, 1968.

Campbell, F. and Green, D. "Optical and Retinal Factors Affecting Visual Resolution", Journal of Physiology 181, pp. 576-593, 1965.

Cornsweet, T. "Visual Perception. New York: Academic Press, 1970.

Doucette, A. R. "Color Discrimination in Digital Displays", Society for Information Display (SID) Digest of Technical Papers, p. 48, 1977.

Erickson, R. and Hemingway, J. "Visibility of Raster Lines in a Television Display", Journal of the Optical Society of America, Vol. 60, No. 5, pp. 700-701, May, 1970.

Hamerley, J. and Springer, R. "Perception of Raggedness of Edge Images", Optical Society of America Meeting, San Francisco, California, November, 1978.

Hunt, R. W. G. The Reproduction of Color. 3rd Edition, New York: John Wiley and Sons, 1975.

Joblove, G. H. and Greenberg, D. "Color Spaces for Computer Graphics", SIGGRAPH '78 Proceedings, Vol. 12, No. 3, pp. 20-25, August, 1978.

Julesz, B. "Experiments in the Visual Perception of Texture", Scientific American 232, pp. 34-43, 1975.

Julesz, B. Foundations of Cyclopean Vision. Chicago: University of Chicago Press, 1971.

Kajiya, J. "Toward a Mathematical Theory of Perception", PhD Thesis, University of Utah, 1979.

Leler, W. J. "Human Vision, Anti-Aliasing, and the Cheap 4000 Line Display", SIGGRAPH '80 Proceedings, Vol. 14, No. 3, pp. 308-313, July, 1980.

Limb, J. O.; Rubinstein, C. B.; and Thompson, J. E. "Digital Coding of Color Video Signals" IEEE Transactions on Communications, Vol. 25, p. 1349, November, 1977.

MacAdam, D. L. "Colorimetric Data for Samples of OSA Color Scales", Journal of the Optical Society of America, Vol. 68, p. 121, January, 1978.

MacAdam, D. L. "Uniform Color Scales", Journal of the Optical Society of America, Vol. 64, p. 1691, December, 1974.

McCollough, C. "Color Adaptation of Edge-Detectors in the Human Visual System", Science 149, pp. 1115-1116, 1965.

Meyer, G. W. and Greenberg, D. P. "Perceptual Color Spaces for Computer Graphics", SIGGRAPH '80 Proceedings, Vol. 14, No. 3, pp. 254-261, July, 1980.

Neal, C. B. "Television Colorimetry for Receiver Engineers", IEEE Transactions on Broadcast Television and Receiver, August, 1973.

Ratliff, F. Mach Bands, San Francisco: Holden-Day, 1965.

Riggs, L. "Visual Acuity", C. H. Graham (Ed.), Vision and Visual Perception, New York: Wiley and Sons, pp. 321-349, 1965.

Sachs, M.; Nachmias, J." and Robson, J. "Spatial Frequency Channels in Human Vision", Journal of the Optical Society of America, Vol. 61., pp. 1176-1186, 1971.

Szabo, N. "Digital Image Anomalies: Static and Dynamic", SPIE Vol. 162, Visual Simulation and Image Realism, pp. 11-15, 1978.

Thompson, F. "Television Line Structure Suppression", Journal of the Society of Motion Picture and Television Engineers, Vol. 66, No. 10, pp. 602-606, October, 1957.

Weisstein, N. "Metacontrast", Visual Psychophysics. edited by L. M. Hurvich and D. Jameson, Heidelberg: Springer-Verlag, pp. 233-272, 1972.

Wyszecki, G. and Stiles, W. S. Color Science. New York: John Wiley and Sons, Inc., 1967.

APPENDIX A

ANALYSIS OF THE MEMORY REQUIRED FOR THE CELL DATA BASE

This appendix develops an estimate for the cell memory requirements for REAL SCAN. The cell memory is the virtual addressed buffer between the video disc, holding the full gaming area in its various hierarchies, and the processors calculating the visible scene. The cell memory is repetitively accessed to calculate each scene via REAL SCAN algorithms.

The memory estimate developed depends upon the following parameters:

1. The display as measured by pixels and mapped to the world by the field of view pyramid.

2. The rate at which the scene changes as measured by the eye's linear and angular velocities.

3. The data compression inherent in the interpolation formula, as measured by the number of scene pixels calculated per interpolation patch.

The Field of View

Let us assume that the two orthogonal angles defining the field of view are less than 90° (i.e., currently we are considering $\alpha \cong 40°$ and $\beta \cong 50°$ giving a diagonal, Υ, of $\cong 60°$). Figure A-1 illustrates the field of view pyramid.

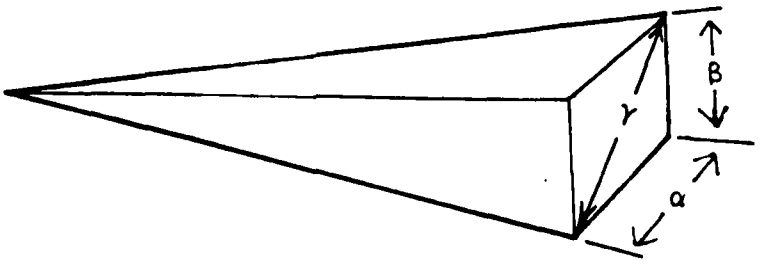


Figure A-1. Illustration of the Field of View Pyramid.

Altitude-Velocity Relation

Let us assume that the maximum eye velocity is proportional to the eye altitude

$$V_{max} = KZ$$

where $K \cong 5 \text{ sec}^{-1}$

(i.e., $V_{max} = 50$ m/s if $Z = 10$ meter)

Let us assume that the maximum eye acceleration is fixed

$$a_{max} = G$$

where $G \cong 2 \text{ m/s}^2$

We have separated eye motion, and therefore scene change as due to a known velocity and an unknown but bounded acceleration. Their relative effect is given by E, the ratio of velocity induced motion to acceleration induced motion.

$$E = \frac{KZT}{\frac{1}{2}GT^2} = \frac{2KZ}{GT} \qquad \text{(A-1)}$$

where T is a computational/memory fetch interval.

The period T corresponds to two complimentary time intervals:

1. The interval over which the unknown acceleration acts to change the eye's predicted position between each new scene determination.

2. The time necessary to set up and complete a disk transfer (i.e., identify the disk blocks, initiate the block transfer, track seek time, and finally actual transfer).

The second interval will define T. T will probably be in the range 0.1 sec to 1.2 sec.

A typical range of "E" based on the altitudes causing the maximum rate at which one would fly by the highest density data yields:

$$Z = 10 \text{ meter}$$

$$T = 1 \text{ sec}$$

$$K = 5 \text{ sec}^{-1} \qquad \text{(A-2)}$$

$$G = 2 \text{ m/sec}^2$$

$$E = 50$$

We can combine the results of Equation A-2 and Figure A-1 to define a worst case computed field of view based upon motion, Υ_c

$$\Upsilon_c \cong \Upsilon + 2/E; \text{ if } E > 10$$

where Υ and Υ_c are expressed in radians and Υ represents the predicted field of view. If $\Upsilon \cong 60^\circ \cong 1$ radian, then 2/E has almost negligible effect.

Calculation of the Working Cell Data Base Size

One can now compute the size of the working cell data base memory. The calculation will take place in three steps:

1. Data base memory required for a fixed eye.
2. Additional data base memory required for an eye moving basically in the same direction as the field of view vector.
3. Additional data base memory required for an eye moving basically perpendicular to the field of view vector.

The geometry bounding a fixed eye's field of view is illustrated in Figure A-2.

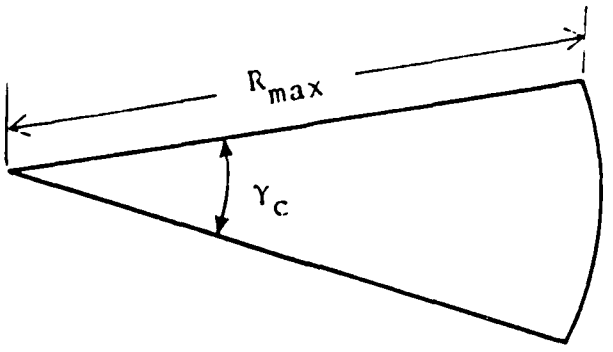


Figure A-2. Fixed Eye's Field of View Projected to a Flat Earth Where Υ_c is the Worst Case Field of View. R_{max} is the Horizon Limit.

Since we recognize that a data base describing the field of view, need not contain more information than resolvable in the display, we choose to use a hierarchy of levels of detail to describe the scene. Let the highest precision level of detail be associated with the minimum altitude of the eye above the scene. Further, let the highest

resolution data be used for a distance, r_1, from the nadir; let the next highest precision level of detail be used from r_1 to $2r_1$, and so forth. This assumes that each higher level of detail maps twice the linear dimension of the preceeding level of detail.

Table A-1 illustrates the effect of hierarchical levels on the memory required. Table A-1 relates the radius over which a given level of detail will exist. For instance, the distance from $2r_1$ to $4r_1$ from the nadir is covered by cells having a linear dimension "4a" covering an area of $16a^2$. Hence, the number of data points required to cover level 2 at the computed field of view, Υ_c is $3\Upsilon_c r_1^2/8a^2$.

TABLE A-1. CELL HIERARCHIAL DATA

Level	Radius Meter	Area Meter^2	Precision Meter^2	Number of Data Points
0	r_1	$\Upsilon_c r_1^2/2$	a^2	$\Upsilon_c r_1^2/(2a^2)$
1	$2r_1$	$3\Upsilon_c r_1^2/2$	$4a^2$	$3\Upsilon_c r_1^2/(8a^2)$
2	$4r_1$	$6\Upsilon_c r_1^2$	$16a^2$	$3\Upsilon_c r_1^2/(8a^2)$
3	$8r_1$	$24\Upsilon_c r_1^2$	$64a^2$	$3\Upsilon_c r_1^2/(8a^2)$
n	$2^n r_1$	$3*2^{2n}\Upsilon_c r_1^2/8$	$2^{2n}a^2$	$3\Upsilon_c r_1^2/(8a^2)$

Hence, we can compute the cell memory needed for one field of view, illustrated in Figure A-3:

$$M = (4 + 3n)\ \Upsilon_c r_1^2/(8a^2) \text{ data points} \qquad (3)$$

For example, if $r_1 = 10$ meter, $R_{max} = 2^n r_1 \cong 20$ kilometer, $a = 0.1$ meter, and $\Upsilon_c = \cong 1$ radian; then $n = 11$ and $M = \frac{370 \cdot 10^3}{8}$ data points

If we use 6 bytes per data point, then about 300 kilo bytes are needed to define one stationary field of view.

Let us now consider the effects of eye motion, on the amount of cell memory required. Figure A-3 illustrates the increased data base required for motion along the line of sight. Figure A-4 illustrates the increased data base required for motion perpendicular to the line of sight. One may easily approximate the additional area of each level of detail required to create the scene based upon eye motion for the period of time, Γ, to locate and the transfer cell data from disk to the active cell memory.

The dimensions of any shaded area in Figure A-3 are:

1. KZT is the length along the line of sight.
2. Approximately $2^{n+1}r_1 \sin(\tau_c/2)$ for the length perpendicular to the line of sight, and n is the level as per Table A-1.

Hence, the increased memory required, M_{new}, is calculated by summing the number of data points in each area (i.e., data points = area/precision)

$$M_{new} \leq \frac{2KZTr_1}{a^2} \sin(\tau_c/2) \sum_{i=0}^{n} \frac{2^i}{2^{2i}} \tag{A-4}$$

$$M_{new} \leq \frac{4KZTr_1}{a^2} \sin(\tau_c/2) \tag{A-5}$$

Figure A-4 illustrates that the new area covered is more complicated to describe when the motion is perpendicular to the line of sight. However, the area is approximately the distance moved, KZT, times the line of sight dimension. The line of sight dimension is approximately r_1 for level 0. For higher levels, such as 3 and above, we may compute the area as the sum of a trapazoid and a cycle. The cycle can be approximated as a triangle of base, KZT and slant height $2^{n-1}r_1\tau_c \sin(\tau_c/2)$ by unwrapping the cycle. The trapazoid has base KZT and height of less than $2^n r_1$. The new memory is made up of two parts

$$M_{trap} = (KZTr_1/a^2) \sum_{i=0}^{n-1} 2^{-2i} \tag{A-6}$$

$$M_{trap} \cong 4KZTr_1/(3a^2) \tag{A-7}$$

and

$$M_{cyc} = (KZTr_1\tau_c \sin(\tau_c/2)/a^2) \sum_{i=1}^{n} (2^{i-1}/2^{-2i}) \tag{A-8}$$

$$M_{cyc} \cong KZTr_1\tau_c \sin(\tau_c/2)/(2a^2) \tag{A-9}$$

The sum of M_{trap} and M_{cyc} yields the new memory requirement for motion perpendicular to the line of sight as shown in Figure A-4.

$$M_{new} = \frac{KZTr_1}{a^2} \quad (4/3 + (\Upsilon_c/2) \sin (\Upsilon_c/2) \qquad \text{(A-10)}$$

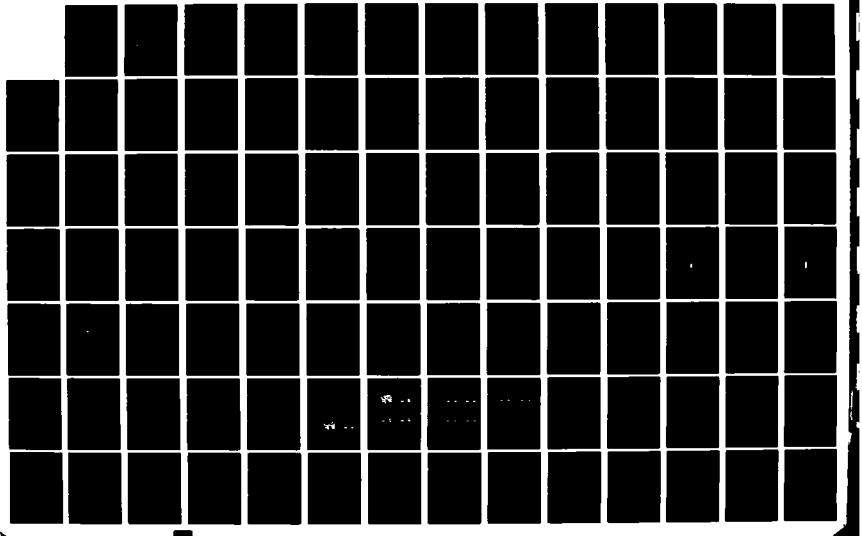
The ratio of new memory required for motion parallel to the line of sight compared to new memory needed for motion perpendicular to the line of sight is the ratio of Equation A-5 to A-10.

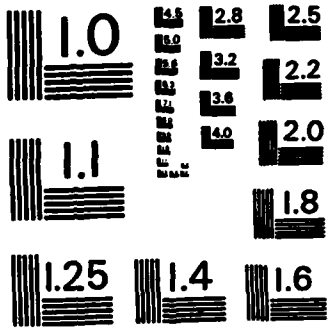
$$R = \frac{12 \sin (\Upsilon_c/2)}{4+3\Upsilon_c \sin (\Upsilon_c/2)}; \text{ where } \Upsilon_c \leq 180^\circ$$

Further, if $60^\circ \leq \Upsilon_c \leq 150^\circ$, then $\frac{12}{11} < R < 1$. Hence, one should use the new memory estimates based on motion parallel to the line of sight to calculate worst case memory requirements. For example if $\Upsilon_c \cong 60^\circ$, $r_1 = 10$ meter, $a = 0.1$ meter, $KZ = 50$ meter/sec, and $T = 0.3$ sec, then $M_{new} = 30{,}000$ data points or about 0.65 of those required for a stationary scene.

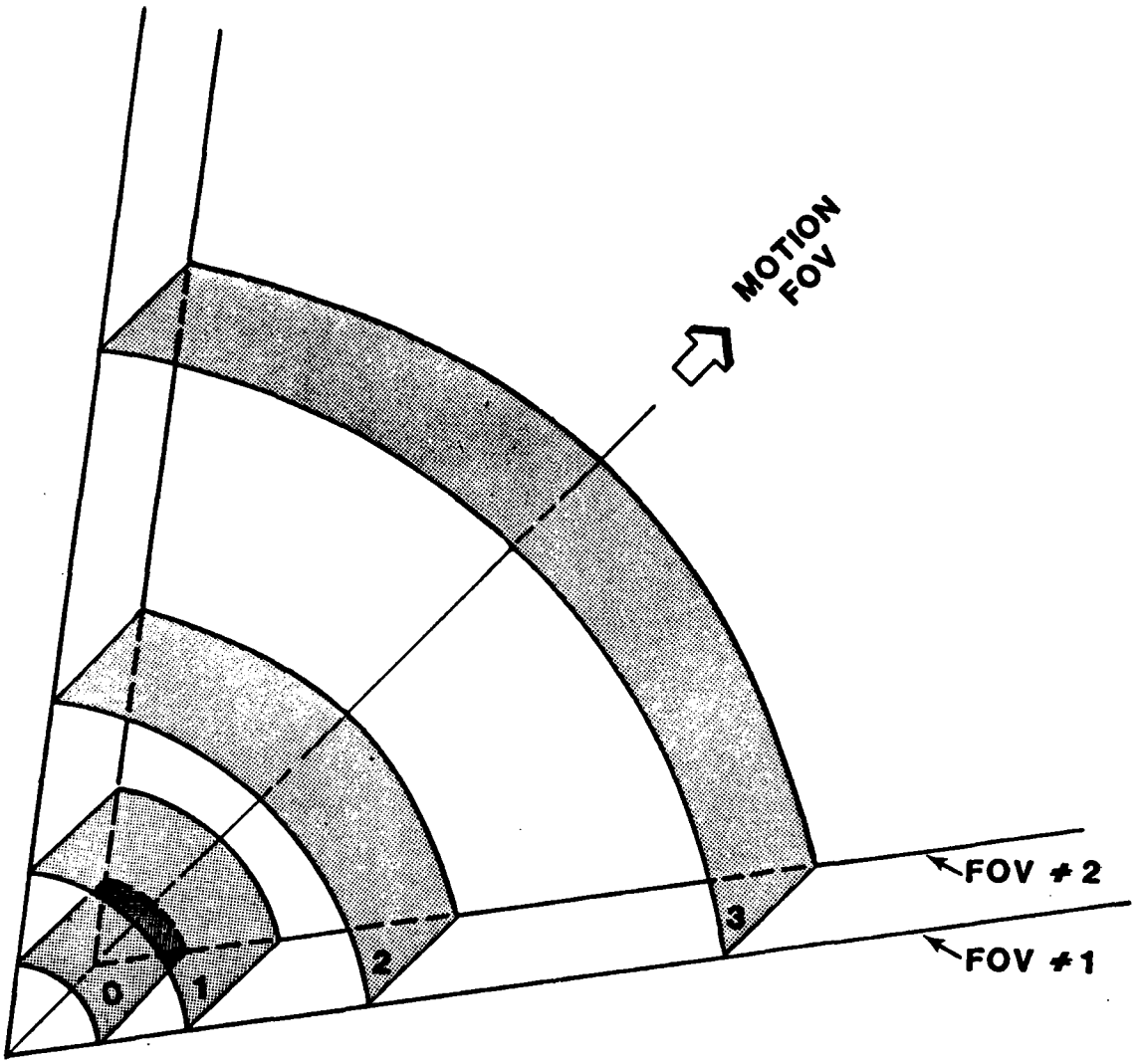
AD-A122 000 REAL SCAN EVOLUTION(U) UNIVERSITY OF CENTRAL FLORIDA 3/6
ORLANDO DEPT OF ELECTRICAL ENGINEERING B W PATZ ET AL.
FEB 82 NAVTRAEQUIPC-80-D-D014-2 N61339-80-D-0014

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A



NOTE: THE SHADED AREAS REPRESENT THE INCREASED MEMORY REQUIREMENT

Figure A-3.
Additional Data Base Required for Motion Along the Line of Sight.

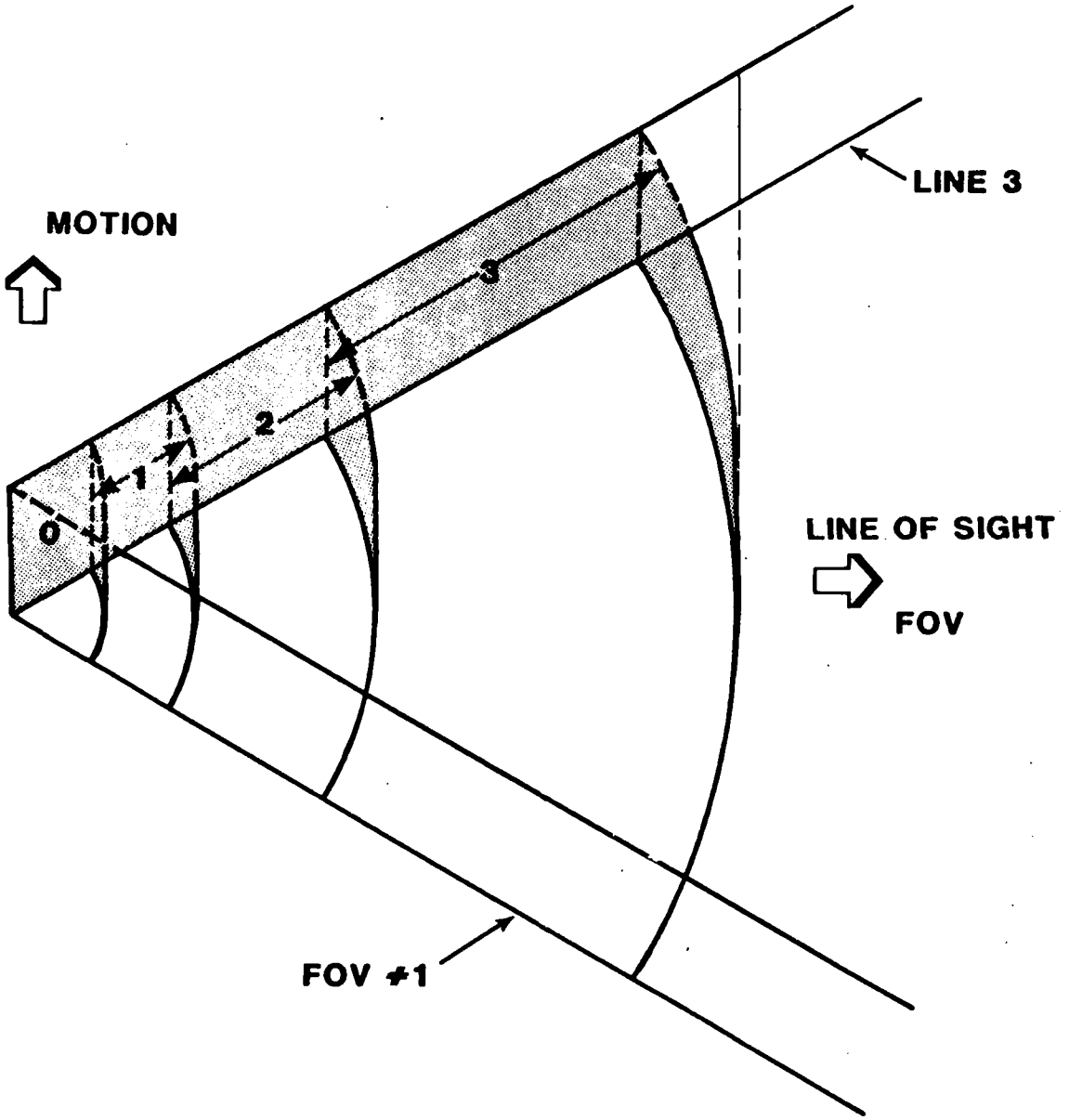


Figure A-4.
Additional Data Base Required for Motion Perpendicular to the Line of Sight.

APPENDIX B

MEMORY REQUIRED FOR A LARGE GAMING AREA HAVING DISPLAY LIMITED DETAIL

This appendix develops estimates of the video disk memory requirement for a REAL SCAN system. The parameters used to develop the estimate are as follows:

1. Gaming area.

2. Number of hierarchies required to create a scene.

3. Data compression available to encode the gaming area's detail.

Let the high resolution requirement be defined as 10^{-3} radian. Then, if a nearest approach of 5 meter is specified, one determines a need for 0.5 cm detail within the scene. The far viewing distance may be estimated as between 5 km to 50 km. If the range of viewing distance is 5 m to 5 km, then 11 levels are required for a hierarchy based on doubling the linear detail between succeeding levels. If the range of viewing distance is 5 m to 50 km but a 10:1 interpolation is used over the data base, the 11 levels of hierarchy are again required. If one adds three additional levels of the hierarchy then a nearest approach to the simulated world of 5/8 meters is possible.

This appendix will assume the need of 12 hierarchies. The data base will be interpolated at about 16:1 with high density detail. However, the total memory estimates developed are negligibly increased even if the number of levels is doubled.

Let the gaming area be a square. Let the center of the square define a region of highest detail (i.e. allowing nearest approach) and let the boundary of the square require less detail (i.e. keeping the nearest approach within the central square).

Figure B-1 illustrates the hierarchy levels required in the total gaming area. Figure B-1 shows all 12 levels are required within the central patch and fewer levels are required to cover larger and larger boarder regions.

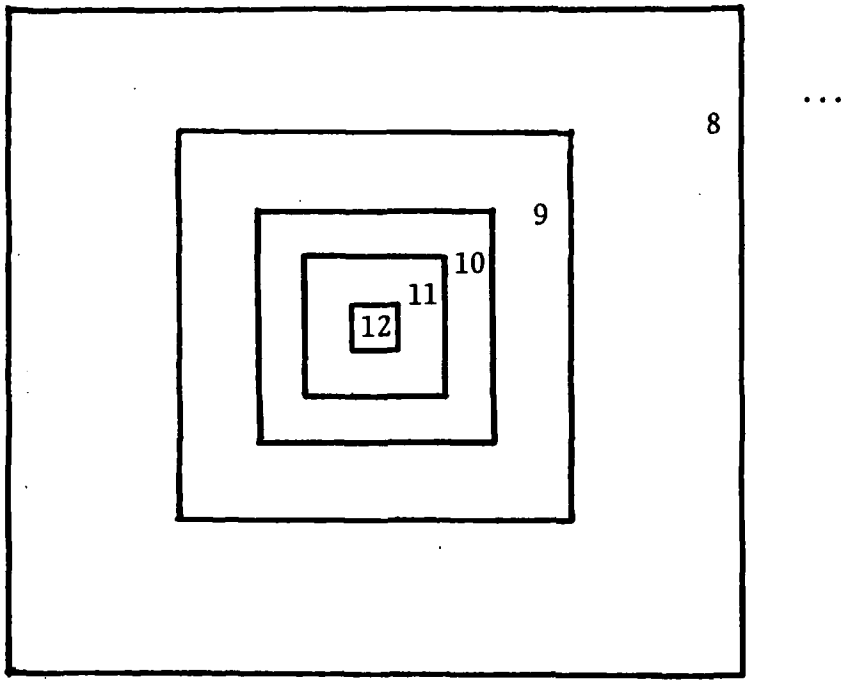


Figure B-1. Levels Required To Represent Gaming Area.

Table B-1 indicates the number of points in a patch, assuming a 16:1 interpolation, and the patch area as a function of resolution. Since linear resolution is proportional to patch dimension, the number of data points representing a patch remains constant at 4^K.

Table B-1. Memory Requirement Versus Gaming Parameters.

Resolution Meter	Data Spacing Meter	Patch Length Meter	Data Points	Memory Bytes
0.004	0.064	4	4096	24,576
0.008	0.128	8	4096	24,576
.016	0.256	16	4096	24,576
.032	0.512	32	4096	24,576
.064	1.024	64	4096	24,576
.128	2.048	128	4096	24,576
.256	4.096	256	4096	24,576
.512	8.192	512	4096	24,576
1.024	16.384	1024	4096	24,576
2.048	32.768	2048	4096	24,576
4.096	65.536	4096	4096	24,576
8.192	131.072	8192	4096	24,576

The memory requirement in Table B-1 is estimated by allowing two bytes per elevation, three bytes per full color, and one byte for other parameters such as texture.

An estimate of the total memory required to represent a given gaming area can be made by counting the number of each patch (i.e. level) type needed. In the case of the central square, the number of patches is approximately $4(S/4)^2/3$ where S is the length of the side of the square in meters. The number of patches in a boarder of dimension d beyond the square is $((S+2d)^2 - S^2)/d^2$ or $4(S+d)/d$. Table B-2 illustrates the number of patches needed to represent regions having interior gaming squares of 8 km, 16 km, 24 km, and 32 km on a side. Table B-2 indicates a video disk requirement of 10^{11} to 10^{12} bytes.

Table B-2. Memory Requirement Versus Gaming Area.

Central Patch Length Meter	Total Gaming Area km^2	Central Patches $(4(S/4)^2/3$ $*10^6$	Boarder Patches $8(S/4)$ $*10^3$	Memory 10^{10} bytes
8192	580	5.33	16.3	12.8
16384	2320	21.33	32.8	25.6
24576	2900	26.67	49.1	38.4
32768	9280	85.33	65.6	51.2

APPENDIX C

INTERMEDIATE PROJECTION PLANES

Cartoon animation has clearly demonstrated the vitality of using artist 2-D renderings for simulating scenes. However, training requires correct depth cues. Parallax, the perceived relative motion between distant points on the same line of sight, must be faithfully reproduced. Figure C-1 illustrates parallax. Points A and B are in the scene along the same line of sight from point E, the eye. Figure C-1 illustrates the perceived displacement between A and B due to motion of E to the locus of constant perceived angular separation, θ.

Figure C-2 suggests that display limited image generation of complex scenes might benefit by determining a set of intermediate projection planes located in the scene and valid for a space, such as the scene between planes A and B. The intermediate projection planes calculated for the scene would correspond to the artist renderings used for cartoons, but have display limited fidelity. Figure C-2 illustrates the concept. Only a few of the planes required to faithfully reproduce the scene are shown. Figure C-2 illustrates that plane B maps the scene projected from the space between planes A and B, and projected through E to plane B. Likewise, plane C corresponds to the space between planes B and C and projected through E to plane C. Further, Figure C-2 illustrates that, for the case of rapid eye motion, the display may be more efficiently calculated from the scene's data base for that part of the scene between the eye and plane C rather than resorting to intermediate planes.

This appendix solves the problem of determining the locus illustrated in Figure C-1 and evaluating when intermediate planes of the type shown in Figure C-1 reduce the computation required to generate the display from a given database.

Posing the Locus Problem

Given: Cartesian coordinate system (x,y,z) and two points, A and B.

Problem: Determine the locus, P, on any plane containing A and B such that the angle $\angle APB$ is constant.

Solution: Figure C-3 shows points A and B separated a distance d where:

(1) the X axis is directed from A to B with an origin midway between A and B.

(2) the Y axis is perpendicular to the line A B and directed away from the line.

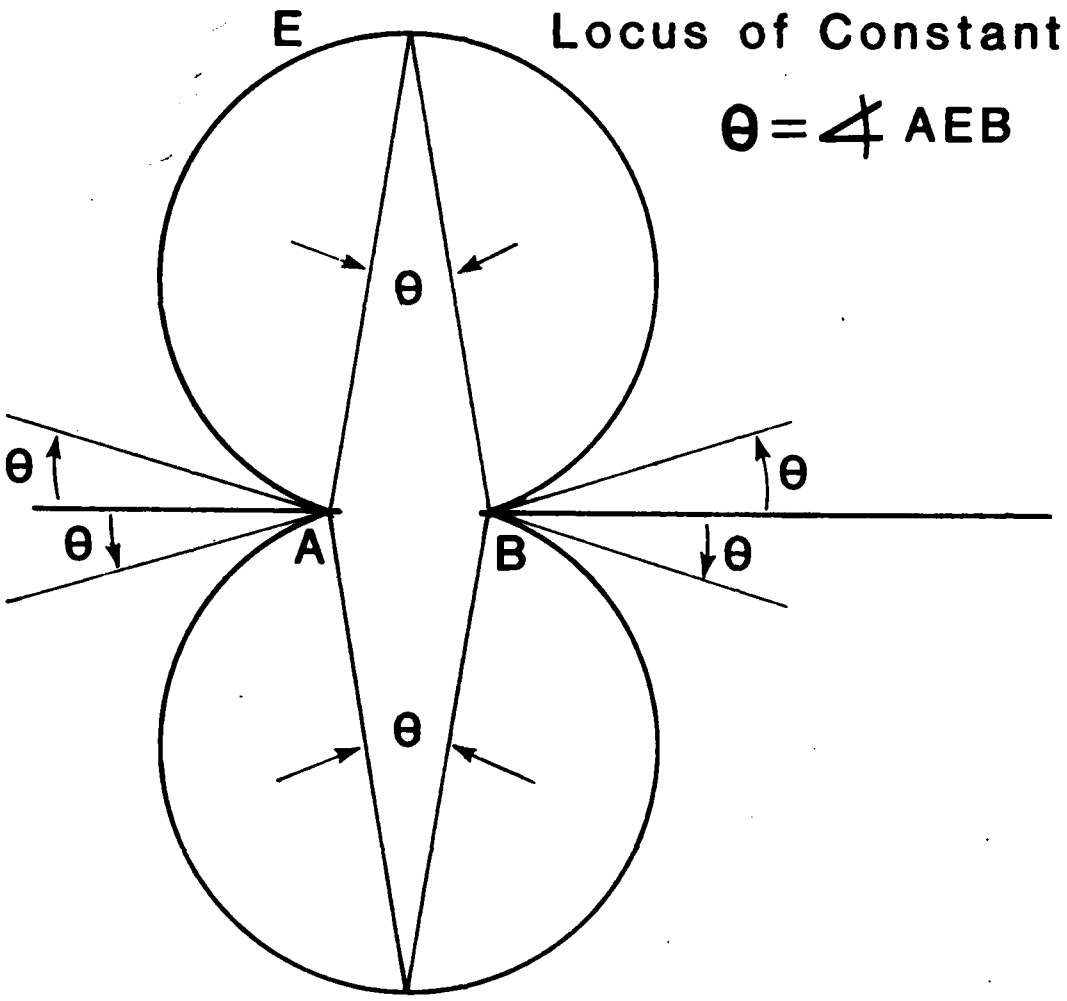


Figure C-1. Parallax Illustration.

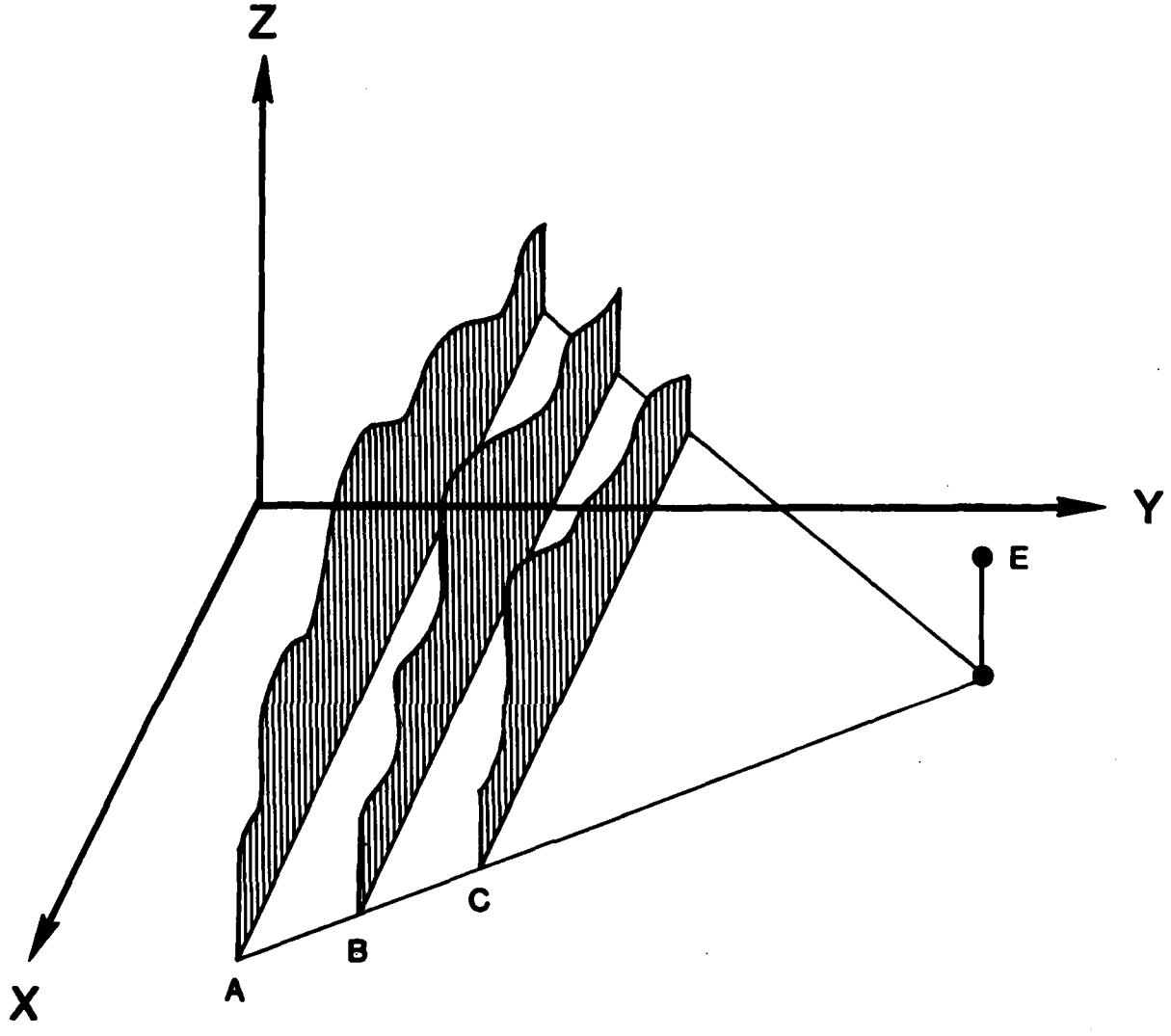


Figure C-2. Set of Planes Which Match a Three Dimensional Scene for an Allowed Parallax Error.

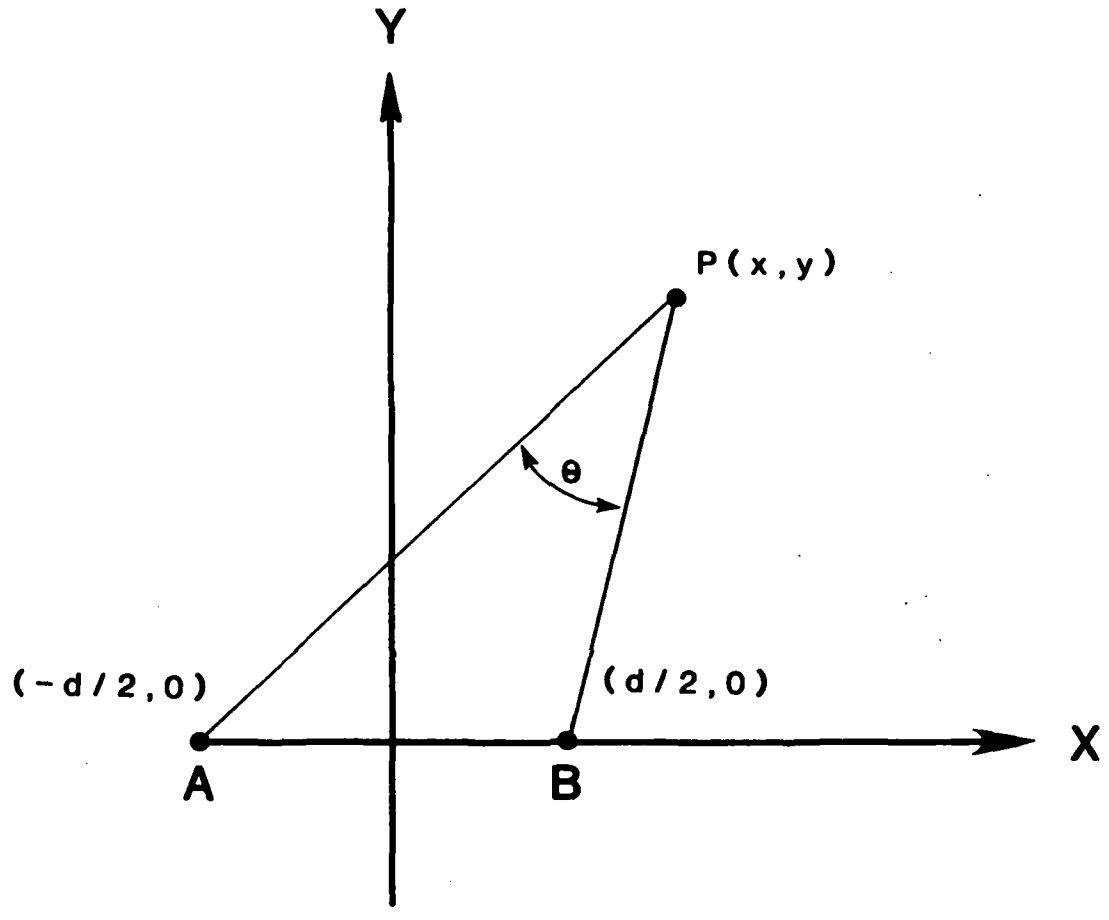


Figure C-3. Coordinate System for the Locus Problem.

The locus, P(X,Y) can be solved by using the cross product of AP and BP, and the dot product of AP and BP.

$$|AP \times BP| = |AP| * |BP| * |\sin\theta| \qquad (C\text{-}1)$$

$$|AP \cdot BP| = |AP| * |BP| * |\cos\theta| \qquad (C\text{-}2)$$

so

$$|AP \times BP| = |AB| \cdot |BP| * |\tan\theta| \qquad (C\text{-}3)$$

where

$$AP = (-d/2-X,-Y)$$

$$BP = (d/2-X,-Y)$$

$$|AP \cdot BP| = X^2 - d^2/4 + Y^2$$

$$|AP \times BP| = (0,0, Yd)$$

Equation C-3 becomes

$$\pm Yd \cot\theta = X^2 + Y^2 - d^2/4 \qquad (C\text{-}4)$$

which is the equation for a circle, as illustrated in Figure C-1. (24, 25)

$$X^2 + (Y \pm d \cot\theta/2)^2 = (d/(2 \sin\theta))^2 \qquad (C\text{-}5)$$

One cannot use the approximation $\tan\theta \sim \sin\theta \sim \theta$ for small θ (even though $\theta < 1/100$), since this yields a locus tangent to the origin in Figure C-1, which is incorrect for analysis in the neighborhood of either point A or B. The locus in the neighborhood of A or B (i.e., within 10 d of A or B) is the interesting region for parallax analysis since it will determine at which point one should begin to use intermediate planes rather than calculate directly from the data base.

Let $\tan\theta = a$, then $\sin^2\theta = a^2/(1+a^2)$ and Equation C-5 becomes

$$X^2 + (Y \pm d/(2a))^2 = d^2 (1 + a^2)/(2a)^2 \qquad (C\text{-}6)$$

Display Limited Parallax Motion

Given: The locus of constant parallax (Figure C-1 and Equation C-6) in terms of the parallax angle parameter, a, and point separation, d.

Problem: Determine the shortest distance from E, shown in Figure C-1, to the locus. Parameterize this distance in terms of a, d, the maximum velocity, V, of E, the display update time, T, and the distance from B to E, D. (The goal of this problem's solution is to determine the conditions under which intermediate plane projections, as shown in Figure C-2, could reduce the computational load of computer generated imagery.)

---

[24]Chestnut, H. and Mayer, R. "Servomechanisms and Regulating System Design Vol. 1", John Wiley & Son, New York, New York, pp. 226-228, 1951.

[25]D'Azzo, J. and Houpis, C. "Feedback Control System Analysis and Synthesis", McGraw Hill, New York, New York, pp. 366-369, 1966.

Solution: Figure C-4 illustrates this problem's parameters. E is shown to move from its initial position, (D + d/2, 0), to P the nearest point where display limited parallax occurs. P is on the line from E to (0, ± d/(2a)). The time, t, to travel from E to P, at speed, V, is

$$t = \left(\sqrt{(d/2a)^2 + (D + d/2)^2} - \sqrt{(d/2a)^2 (1 + a^2)}\right)/V \qquad (C\text{-}7)$$

$$t = \frac{d}{2aV}\left(\sqrt{1 + \left(\frac{2a}{d}\right)^2 (D + d/2)^2} - \sqrt{1 + a^2}\right) \qquad (C\text{-}8)$$

Since $a << 1$ and $(2a/d)(D + d/2) << 1$, Equation C-7 can be approximated as

$$t = (aD/V)\,(1 + D/d - a^2D^3/d^3 - 2a^2D^2/d^2) \qquad (C\text{-}9)$$

Equation C-9 illustrates that the trigometric approximation suggested by Figure C-4 is useful if $D < d$. However, if $D > d$ then $t = aD/V$ is much too conservative, and one does not recognize the true value of intermediate plane projections. Equations C-8 and C-9 suggest the normalized parallax equations

$$Vt/aD = 1 + D/d; \text{ for } d >> 2aD \qquad (C\text{-}10)$$

and

$$Vt/D = 1; \text{ for } 2aD >> d \qquad (C\text{-}11)$$

where

a = display limited parallax parameter [radian]
V = eye's speed [meter/second]
d = separation between planes [meter]
D = distance from eye to nearest visible point on the nearer plane [meter]
t = minimum time to move to a position where display limited parallax occurs [second]

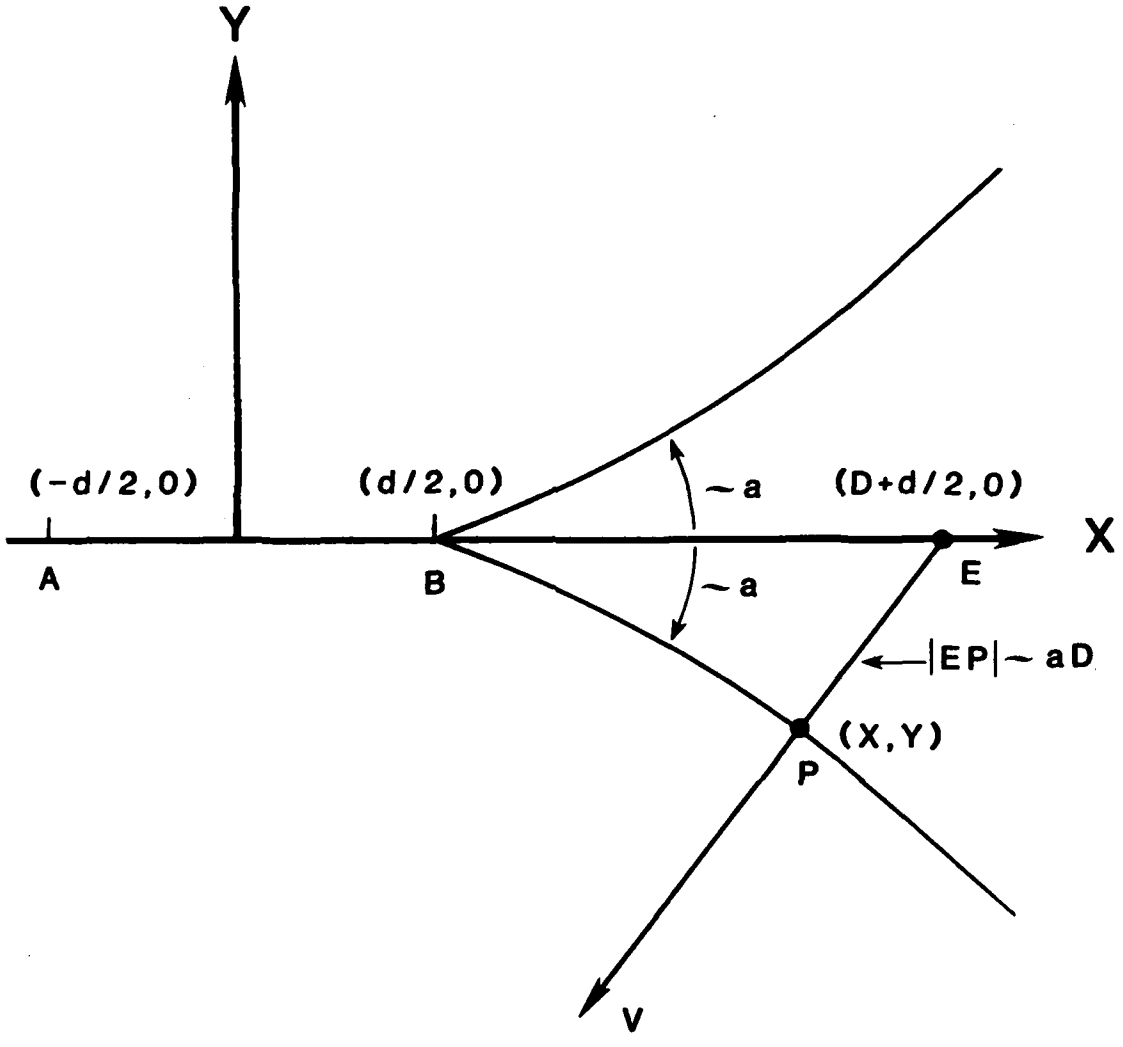


Figure C-4. Motion from E to P Achieving Display Limited Parallax.

Projection Plane Parallax

Given: Equations C-10 and C-11, Figure C-4 and the projection planes illustrated in Figure C-5.

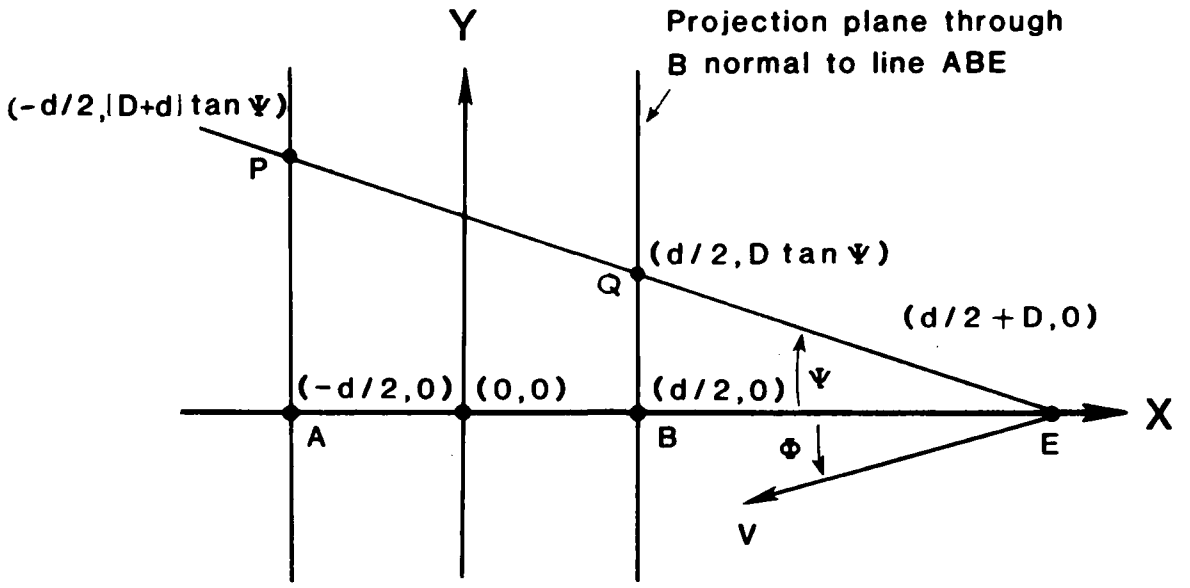


Figure C-5. Projection Planes.

Problem: Determine the minimum time to parallax onset as a function of a,d,D,V and Ψ.

Solution: Equation C-10 indicates the soonest that parallax onset can occur, t_{pq}, for points P and Q of Figure C-5 is approximately

$$t_{pq} = aD(1 + D/d)/(V \cos \Psi) \quad ; \text{ for } d >> 2aD \qquad (C\text{-}12)$$

Equation C-11 indicates the soonest that parallax onset can occur is approximately

$$t_{pq} = D/(V \cos \Psi) \quad ; \text{ for } 2aD >> d \qquad (C\text{-}13)$$

Equations C-12 and C-13 show that an analysis for determining the closest spacing between intermediate projection planes is obviously:

1. Identify the orientation of the projection planes.

2. Determine the distance, D, to the nearest projection plane.

3. Given the display's parallax limit, a, determine the location of the next projection plane such that the time, t, to update the projection planes is the desired number of display updates (i.e., allowing one to save the scene's data on the projection plane without need to recalculate the scene).

However, we also wish to know the shape of an "optimum surface", such that parallax occurs simultaneously over two "optimum surfaces" for given motion vection V. With such a surface one can clearly define the construction of intermediate projection surfaces as a function of eye location, E, and eye motion, V. A simpler problem which clearly has a solution, illustrated by Figure C-5, is the determination of parallax onset for points p, q as a function of the angles: Φ, which defines eye motion direction and Ψ, which defines the p, q orientation to E. This simpler problem will be solved next.

Parallax Onset Time as a Function of Eye Motion Direction

Given: Figure C-5 defining eye motion relative to two arbitrary points p,q on intermediate projection planes a distance d apart and Equation C-6 defining the equal parallax locus.

Problem: Determine the actual time, t, when parallax onset occurs as a function of a,D,d,Ψ, and V.

Solution: The equation for the line through E along V in Figure C-5 is

$$R = Y/\sin\Phi = (X-D-d/2)/\cos\Phi \qquad \text{(C-14)}$$

where R is the range or distance from E to the locus of constant parallax.

The equation for the intercepted locus associated with points A and B is

$$X^2 + (Y + d/(2a))^2 = (d/2a)^2 (1 + a^2) \qquad \text{(C-15)}$$

Solving for the negative value of Y possessing the smallest magnitude, as suggested by Figure C-5, in the neighborhood of point B yields

$$Y^2/\sin^2\Phi + Y((2D + d)/\tan\Phi) + d/a) + D^2 + Dd = 0 \qquad \text{(C-16)}$$

or

$$Y = \sqrt{((d \sin^2\Phi)/(2a)+(D+d/2)\cos\Phi \sin\Phi)^2 - D(D+d)\sin^2\Phi}$$

$$- ((d \sin^2\Phi)/(2a) + (D + d/2)\cos\Phi \sin\Phi) \qquad \text{(C-17)}$$

and the range R, is thus

$$R = - \sqrt{((d \sin\Phi)/(2a) + (D + d/2)\cos\Phi)^2 - D(D+d)}$$

$$+ (d \sin\Phi/(2a) + (D+d/2)\cos\Phi \qquad \text{(C-18)}$$

We are interested in the case

$$(d \sin\Phi/(2a) + (D + d/2)\cos\Phi)^2 >> D(D+d) \qquad \text{(C-19)}$$

which yields an approximation for R

$$R = D(D+d)/((d \sin\Phi)/a + (D+d)\cos\Phi) \qquad \text{(C-20)}$$

since we expect computer generated imagery to yield $a << 1/100$ and $\Phi >> 1/100$. This approximation to R is exact at the limits $\Phi = 0$ and $\Phi = \pi/2$. Equation C-20 may be put in the form of Equation C-9 for determining parallax onset of the points A and B in Figure C-5.

$$t_{AB} = \frac{aD}{V}(1 + D/d) \,/\, (\sin\Phi + a\,(1 + D/d)\cos\Phi) \qquad \text{(C-21)}$$

The general formula for parallax onset of points p and q in Figure C-5 can be obtained from Equations C-12 and C-21.

$$t_{pq} = \frac{aD}{V \cos\Psi}\ (1 + D/d)/(\sin\,(\Phi + \Psi)$$

$$+ a(1 + D/d)\cos\,(\Phi + \Psi);\ \text{for } d > 2aD \qquad \text{(C-22)}$$

Equation C-22 allows one to solve for optimum surfaces through A and B such that parallax is achieved simultaneously at all points on the surface. The optimum surfaces will next be determined. Since $a \sim 0$, let $\Phi = \pi/2$. Define D_1 and d_1 distances along the line from E to the optimum surfaces as shown in Figure C-6.

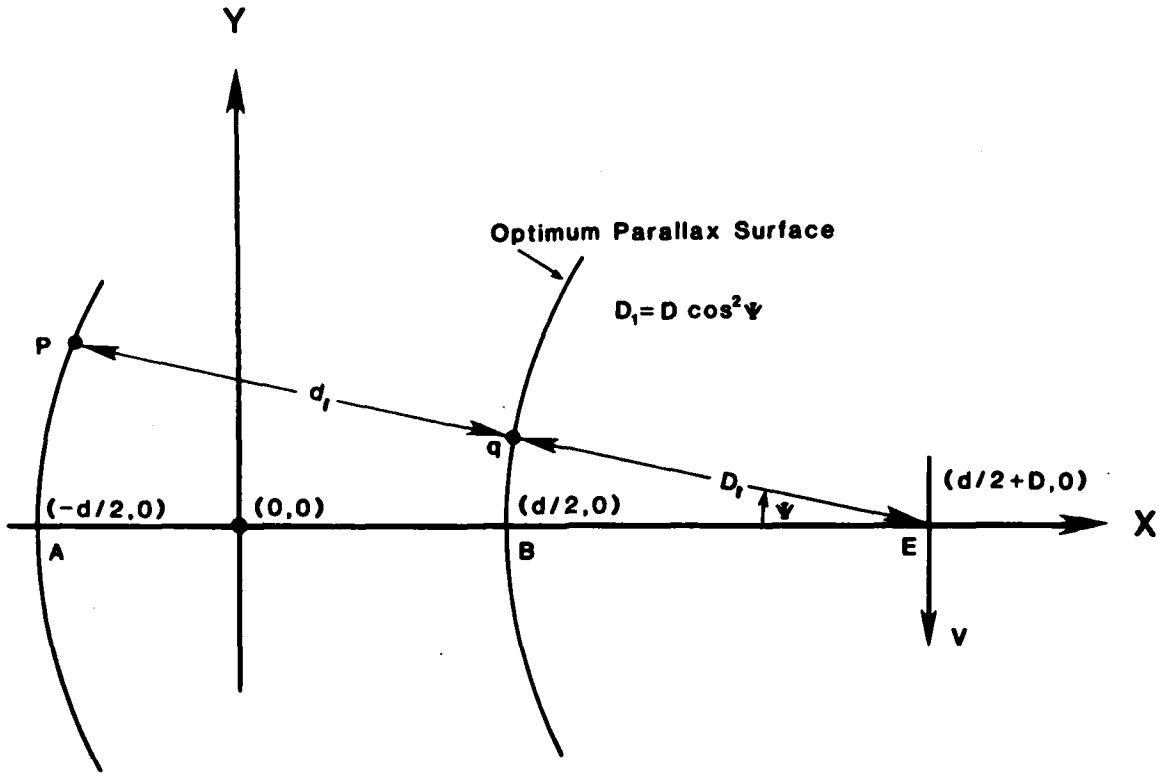


Figure C-6. Optimum Parallax Surfaces.

Equating t_{pq} and t_{AB} to determine the optimum surface yields

$$aD(1 + D/d)/V = aD_1(1 + D_1/d_1)/((V \cos\Psi)(\sin(\pi/2 + \Psi)) + a(1 + D_1/d_1) \cos(\pi/2 + \Psi)); \text{ for } d > 2aD \quad \text{(C-23)}$$

Now D_1 and d_1, can be approximated in terms of D, d, and Ψ as

$$D(1 + D/d) = (D_1/\cos^2\Psi)(1 + D_1/d_1)/(1 - a(1 + D_1/d_1)\tan\Psi); \text{ for } d > 2aD \quad \text{(C-24)}$$

If we further restrict ourselves to the case where

$$1 > a(1 + D_1/d_1) \tan\Psi \quad \text{(C-25)}$$

which covers all regions of interest, then Equation C-24 yields

$$D_1 = D \cos^2\Psi \quad ; \text{ for } d > 2aD \quad \text{(C-26)}$$
$$d_1 = d \cos^2\Psi$$

Figure C-7 is a plot of some optimum parallax surfaces such that all points on the surface would simultaneously need to be reprojected due to motion V, as in Figure C-6. The constant update implies

$$Vt/a = D_o(1 + b_o) = D_1(1 + b_1) = \ldots = D_n(1 + b_n) \qquad \text{(C-27)}$$

where

D_o is the distance to the original surface, normal to the direction of motion.

$D_1 = D_o + d$ is the distance to the next surface, normal to the direction of motion.

$$b_o = D_o/(D_1 - D_o) \qquad \text{(C-28)}$$

$$b_n = D_n/(D_{n+1} - D_n) \qquad \text{(C-29)}$$

Hence, given D_o and b_o one can iteratively determine b_i, D_i. Figure C-7 is plotted for $Do = 5$, $b_o = 5$.

TABLE C-1. DATA FOR FIGURE C-7.

Ψ	$\cos^2\Psi$	D_o	D_1	D_2	D_3	D_4
0	1	5	6	7.5	10	15
10	.969846	4.85	5.82	7.27	9.70	14.55
20	.883022	4.415	5.30	6.62	8.83	13.25
30	.750	3.75	4.50	5.625	7.50	11.25
40	.586824	2.93	3.52	4.40	5.87	8.80
50	.413176	2.066	2.48	3.10	4.13	6.20
60	.250	1.25	1.50	1.875	2.50	3.75

Figure C-7 suggests that two planes will simply bound the optimum surfaces. The optimum surface, initially at distance D_o from E can be approximated by plane p_1 normal to the X axis and plane q_1 normal to the Y axis. The parallax update rate is guaranteed to be less at all points on planes p_1 and q_1 not coincident with the optimum surface but nearer to the eye, than at points coincident with the optimum surface.

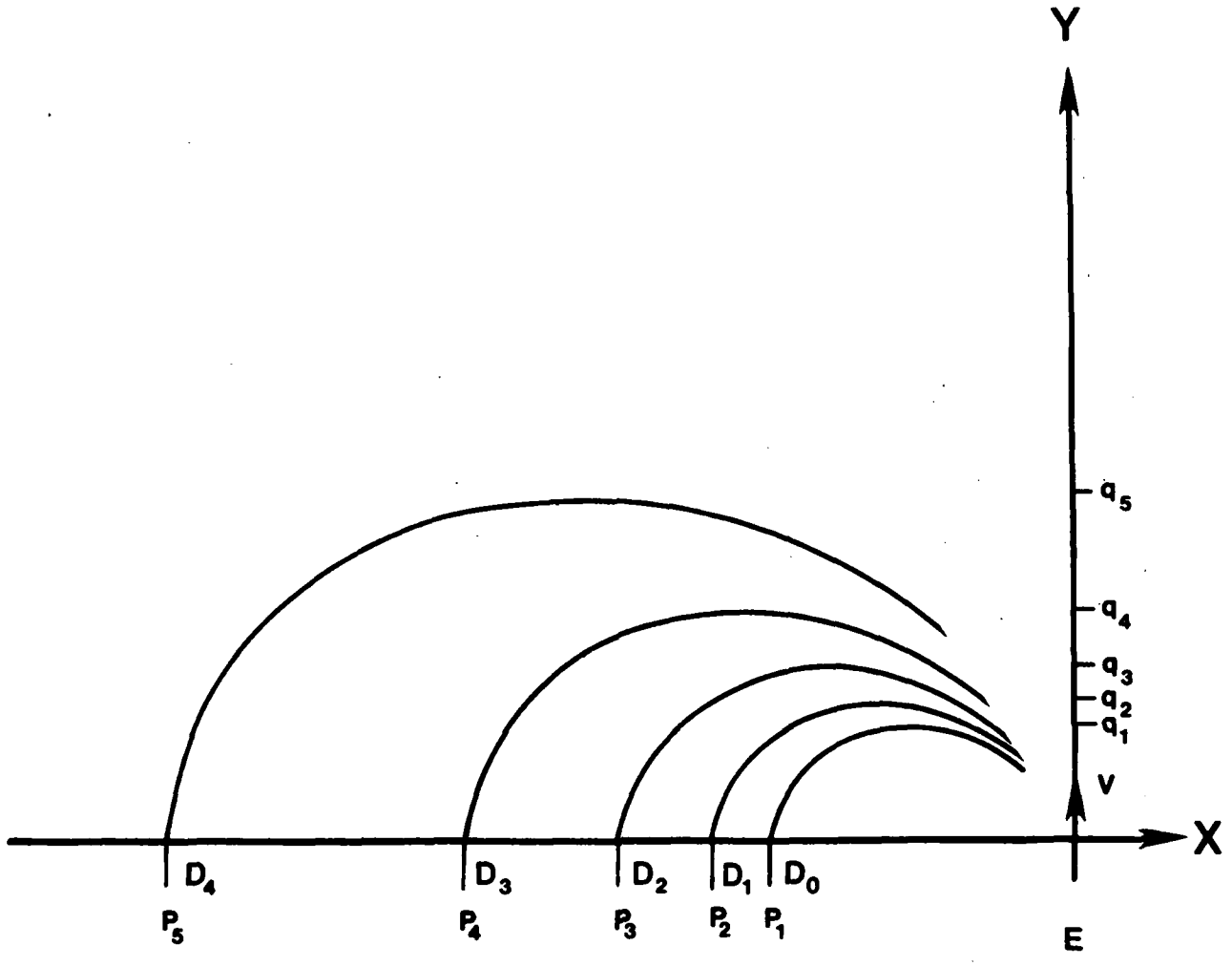


Figure C-7.
Surfaces of Constant Parallax for Motion of E Along the Y Axis.

The distance to plane q_1 from E in terms of D_o can be determined as follows:

1. $Y = D_o \cos^2\Psi \sin\Psi$
2. $dY/d\Psi = 0 = \cos^3\Psi - 2\cos\Psi \sin^2\Psi$
3. $\cos^2\Psi = 2/3$
4. $Y_{max} = 2D_o/(3\sqrt{3}) = 0.385D_o$

One can use Figure C-7 to estimate the location of the first vertical projection plane on a flat earth database when one is moving

parallel to the earth. For instance, if D_o represents the distance of E above earth, such that the desired interval between reprojection of the scene's database onto the flat earth plane is p_1, then the corresponding vertical plane need be no nearer to the nadir than $0.385D_o$ to define plane q_1.

Equations C-27 through C-29 can be used to develop a sequence of distances corresponding to D_5, D_4, D_3 ... D_{-m} in Figure C-7. The distances D_{-1}, D_{-2}, ... D_{-m} correspond to planes nearer the eye, E, but maintaining a constant rate for plane update to eliminate the effect of parallax. Table C-2 contains the sequence of D_5, D_4 ... starting with D_5 scaled to 3000. For example, when $n < -10$ the spacing between adjacent projection planes is less than 13 units and eventually approaches the scene's resolution.

Table C-2 indicates a limit to the use of projection planes. Further, Table C-2 indicates that one should calculate the sequence of projection planes by starting with D_5 or the last plane and determining the intermediate projection planes iteratively, moving from the horizon toward the eye, until the space between the planes nearly matches the hierarchy resolution for the display limited resolution.

TABLE C-2. SCALED DISTANCE TO A PROJECTION PLANE PARALLEL TO EYE MOTION, FOR CONSTANT PARALLAX UPDATE RATE

n	$D_n = 3000/b_{n-1}$	b_n
	length	dimensionless
5	3000.0	0
4	1500.0	1
3	1000.0	2
2	750.0	3
1	600.0	4
0	500.0	5
-1	428.6	6
-2	375.0	7
-3	333.3	8
-4	300.0	9
-5	272.7	10
-6	250.0	11
-7	230.8	12
-8	214.3	13
-9	200.0	14
-10	187.5	15

APPENDIX D

POLYNOMIAL DATABASES

This appendix develops formula for interpolating a single value function f(X,Y) over a regular square grid on (X,Y). Figure D-1 illustrates the regular square grid, identifies the grid points by numbers and the square grid regions by letters. Let grid point 8 have coordinates (X_0,Y_0) and let a grid length be d. Figure D-1 shows Region C with vertices 8,4,5 and 9, where point 5 has coordinates (X_0+d,Y_0+d). The Regions A through E are called patches.

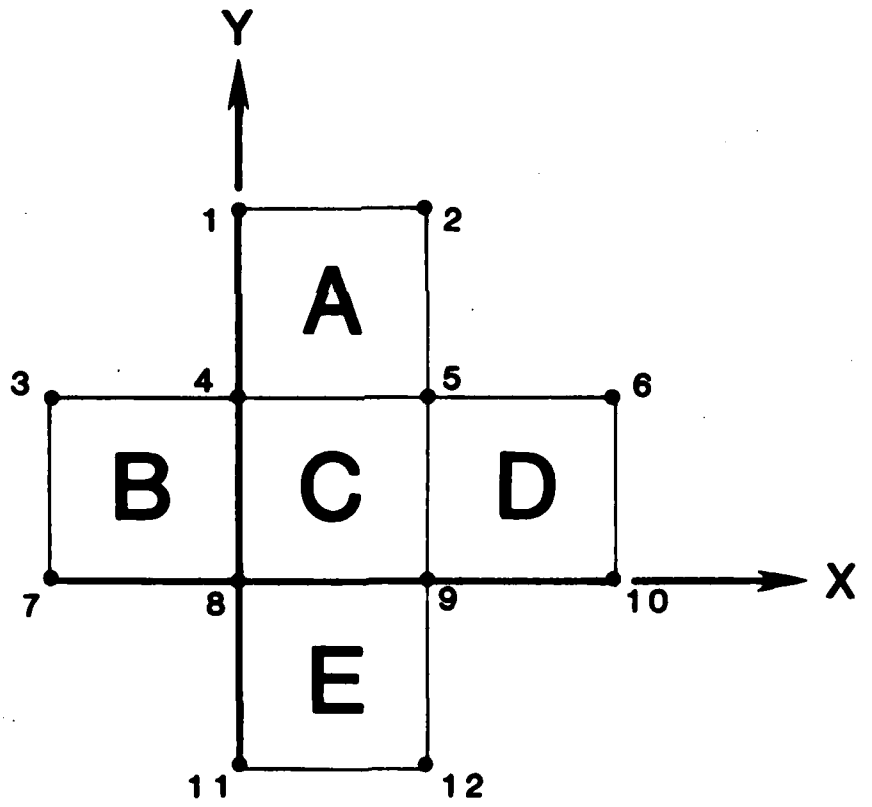


Figure D-1. Definition of Square Grid.

The purpose of this appendix is to develop a regular family of polynomials based upon sampled data at the point 1 through 12, for interpolating f(X,Y) over the Region C. The goals of the interpolation polynomial development are:

1. The set of formula must be simple to evaluate.

2. The coefficients of the polynomial interpolation formula must be simple to evaluate.

3. The family of interpolation formulas should offer a method for regularly increasing the degree (complexity), so that one can evaluate the trade-off between sample point spacing and interpolation formula complexity.

4. Adjacent patch functions must match on common boundaries such as A-C, C-D, and B-C as shown in Figure D-1.

5. As the degree of the polynomial increases, the partial derivatives of the patch formula should approach each other. (It appears that the Overhauser-Coons bi-cubic patch is the lowest order polynomial allowing continuity in function and slope across patch boundaries).

Polynomial Family Development

The simplest interpolation formula which satisfies the goals is:

$$P_o(X,Y) = U_{00} + U_{10}(X-X_o) + U_{01}(Y-Y_o) + U_{11}(X-X_o)(Y-Y_o) \qquad (D\text{-}1)$$

where

X_o is the value of X at points (1,4,8,11)

Y_o is the value of Y at points (7,8,9,10)

$U_{00} = f_8$

$U_{10} = (f_9 - f_8)/d$

$U_{01} = (f_4 - f_8)/d$

$U_{11} = (f_5 - f_4 - f_9 + f_8)/d^2$

f_i is the value of f(X,Y) at point i

d is the length of any side of the grid

Interpolation formula 1 given by Equation D-1, requires at most four multiplications and five additions for evaluation. P_o can be rewritten:

$$P_o(X,Y) = U_{00} + (X-X_o)(U_{10} + U_{11}(Y-Y_o)) + U_{01}(Y - Y_o) \qquad (D\text{-}2)$$

$$P_o(X,Y) = U_{00} + (Y - Y_o)(U_{01} + U_{11}(X-X_0)) + U_{10}(X-X_0) \qquad (D\text{-}3)$$

Then the evaluation of $P_o(X,Y)$ requires five additions and three multiplications.

The generation of the terms producing the next level of complexity appears to rest on some level of insight. If we classify formula 1 as linear or first order, then the next level of complexity would be quadratic or second order. Further, the form of a simple function will be such that terms which add to the formula complexity on boundaries, such as A-C and C-E, produce no contribution to the formula on orthogonal boundaries (i.e., B-C and C-D shown in Figure D-1). One next level term, D_{22} for formula 1 which satisfies these requirements is:

$$D_{22}(X,Y) = (X-X_o)(X-X_1)(a_o + a_1(Y-Y_o)) + (Y-Y_o)(Y-Y_1)$$
$$(b_o + b_1(X-X_o)) + d(X-X_o)(X-X_1)(Y-Y_o)(Y-Y_1) \qquad \text{(D-4)}$$

The form of the polynomial along any four point boundary line in Figure D-1, such as (3,4,5,6) is cubic. Hence, other next level terms are:

$$D_{32} = (X-X_o)(X-X_1)X-X_2)(a_o + a_1\ (Y-Y_o)) + (Y-Y_o)(Y-Y_1)(Y-Y_2)$$
$$(b_o + b_1\ (X-X_o)) + d(X-X_o)(X-X_1)(Y-Y_o)(Y-Y_1) \qquad \text{(D-5)}$$

and

$$D_{33} = (X-X_o)(X-X_1)(X-X_2)(a_o + a_1(Y-Y_o) + a_2(Y-Y_o)(Y-Y_1)) +$$
$$(Y-Y_o)(Y-Y_1)(Y-Y_2)(b_o + b_1(X-X_o) + b_2(X-X_o)(X-X_1))$$
$$+ d(X-X_o)(X-X_1)(Y-Y_o)(Y-Y_1) \qquad \text{(D-6)}$$

where

(X_1,Y_1) is the coordinate of point 5

(X_1,Y_2) is the coordinate of point 2

(X_2,Y_1) is the coordinate of point 6

D_{31} is D_{32} without the d term.

A next level of complexity term, say D_{21}, can also be generated by neglecting the d coefficient in Equation D-4 for D_{22}.

All these terms are symmetric in X and Y, that is equal order of complexity for X and Y polynomials in provided. Further, a progression of more complex polynomials exists as follows:

1. $P_o(X,Y)$
2. $P_1(X,Y) = P_o(X,Y) + D_{21}(X,Y)$
3. $P_2(X,Y) = P(X,Y) + D_{22}(X,Y)$
4. $P_3(X,Y) = P_1(X,Y) + D_{31}(X,Y)$
5. $P_4(X,Y) = P_1(X,Y) = D_{32}(X,Y)$
6. $P_5(X,Y) = P_1(X,Y) = D_{33}(X,Y)$
7. P_6 is the Overhauser-Coons formula

Evaluation of the Polynomial Coefficients

The best method for evaluation of all the coefficients for the P_i formula is not obvious. However, it appears that the terms a_o, a_1, b_o and b_1, can be clearly and simply evaluated. The procedure follows from the need to match boundary conditions.

The evaluation of the term d in Equations D-4 or D-5 is not obvious nor does it follow from matching either first or second derivatives at verticies, since the d term is always zero at the verticies. Therefore, this appendix will describe only the evaluation of coefficients a_o, a_1, b_o and b_1 of Equations D-4 and D-5, allowing one to evaluate and compare only polynomials P_o,P_1, P_3 and P_6 of the set proposed.

EVALUATION OF a_o AND a_1

The a_o and a_1 coefficients in D_{22} or D_{31} must satisfy boundary conditions between region A-C and C-E. Hence, if one writes the formula

$f(X,Y_1)$ for the line 3,4,5,6 as

$$f(X,Y_1) = r_o + r_1(X-X_o) + r_2(X-X_o)(X-X_1) + r_3(X-X_o)(X-X_1)(X-X_2) \quad (D-7)$$

Likewise, for the line 7,8,9,10

$$f(X,Y_o) = t_o + t_1(X-X_o) + t_2(X-X_o)(X-X_1) + t_3(X-X_o)(X-X_1)(X-X_2) \quad (D-8)$$

The terms r_o, r_1, t_o, and t_1 are clearly satisfied by polynomial $P_o(X,Y)$.

Hence, it is obvious that Equations D-4, D-7 and D-8 yield

$$a_o = t_2 \tag{D-9}$$

$$a_1 = (r_2-t_2)/(Y_1-Y_o) = (r_2-t_2)/d \tag{D-10}$$

while Equations D-5, D-7 and D-8 yield

$$a_o = t_3 \tag{D-11}$$

$$a_1 = (r_3-t_3)/d \tag{D-12}$$

By expressing the polynomials in the form of Equations D-7 and D-8, rather than powers of X and Y, we have allowed insight to show obvious conditions. Hopefully, this technique will also yield simple (if not the simplest) coefficient evaluation procedure.

EVALUATION OF b_o AND b_1

If we define

$$f(X_1,Y) = q_o + q_1(Y-Y_o) + q_2(Y-Y_o)(Y-_1) + q_3Y-Y)(Y-Y_1)(Y-Y_2) \tag{D-13}$$

and

$$f(X_o,Y) = s_o + s_1(Y-Y_o) + S_2(Y-Y_o)(Y-Y_1) + s_3(Y-Y_o)(Y-Y_1)(Y-Y_2) \tag{D-14}$$

Then Equations D-4, D-13 and D-14 yield

$$b_o = s_2 \tag{D-15}$$

$$b_1 = (q_2 - s_2)/d \tag{D-16}$$

while Equations D-5, D-13 and D-14 yield

$$b_o = s_3 \tag{D-17}$$

$$b_1 = (q_3 - s_3)/d \tag{D-18}$$

EVALUATION OF q,r,s and t COEFFICIENTS

The coefficients t_o, t_1, t_2 and t_3 relate to the line (7,8,9,10) of Figure D-1. The t coefficients can be evaluated in order, with the result:

$$t_o = f_8 \tag{D-19}$$

$$t_1 = (f_9-f_8)/d \tag{D-20}$$

$$t_2 = (f_{10} - f_8 - 2(f_9 - f_8))/(2d^2) \quad \text{(D-21)}$$

which reduces to

$$t_2 = (f_{10} - 2f_9 + f_8)/(2d^2) \quad \text{(D-22)}$$

$$t_3 = (f_7 - f_8 + (f_9 - f_8) - (f_{10} - 2f_9 + f_8))/(-6d^3) \quad \text{(D-23)}$$

which reduces to

$$t_3 = -(f_7 - 3f_8 + 3f_9 - f_{10})/(6d^3) \quad \text{(D-24)}$$

This procedure can also be used to evaluate the q,r, and s coefficients along their respective line boundaries. The result is:

$$a_o = f_9$$

$$q_1 = (f_5 - f_9)/d$$

$$q_2 = (f_2 - 2f_5 + f_9)/2d_2$$

$$q_3 = (f_{12} - 3f_9 + 3f_5 - f_2)/(6d^2)$$

$$r_o = f_4$$

$$r_1 = (f_5 - f_4)/d$$

$$r_2 = (f_6 - 2f_5 + f_4)/(2d^2)$$

$$r_3 = (f_3 - 3f_4 + 3f_5 - f_6)/(6d^3)$$

$$s_o = f_8$$

$$s_1 = (f_4 - f_8)/d$$

$$s_2 = (f_1 - 2f_4 + f_8)/2d^2)$$

$$s_3 = (f_{11} - 3f_8 + 3f_4 - f_1)/(6d^3)$$

At this point it is clear that formula $P_3(X,Y)$ allows one to match the cubic functions on all boundaries and to guarantee that the partial derivatives of $P_3(X,Y)$ at the four vertices (4,5,8,9) will also match the original cubic functions. Hence, formula $P_3(X,Y)$ appears to be a good candidate for further study.

APPENDIX E

ARBITRARY EYE ORIENTATION

The display consists of a rectangle defined by four corners, C1 through C4 as shown in Figure E-1. The points 1′ through 4′ correspond to C1 through C4 when C1 through C4 are projected to the X-Y plane along the Z axis. Hence for C1 = (X1,Y1,Z1), then 1′ = (X1,Y1,0).

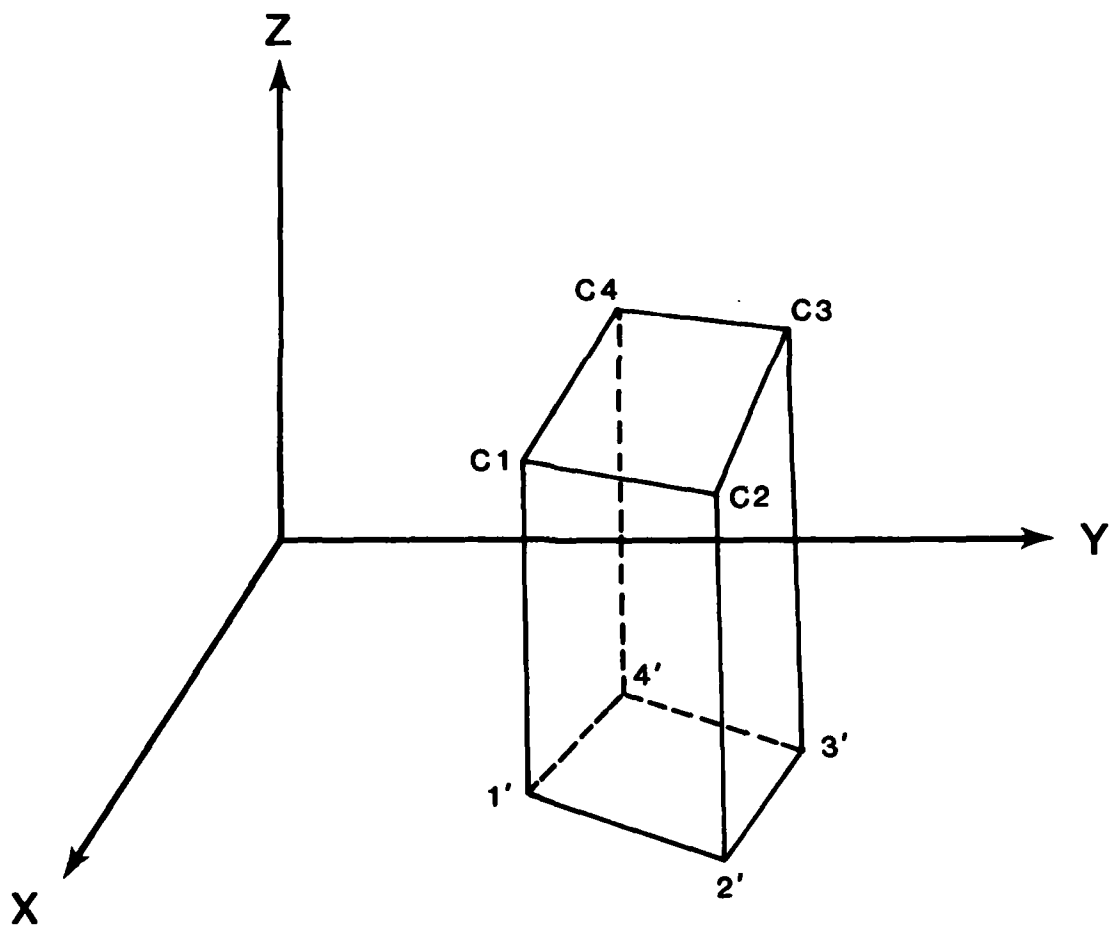


Figure E-1. Display Plane Dropped to XY Plane.

Figure E-2 illustrates the four corners, 1′ through 4′, on the X-Y plane in relation to a nadir n, and the display center, C.

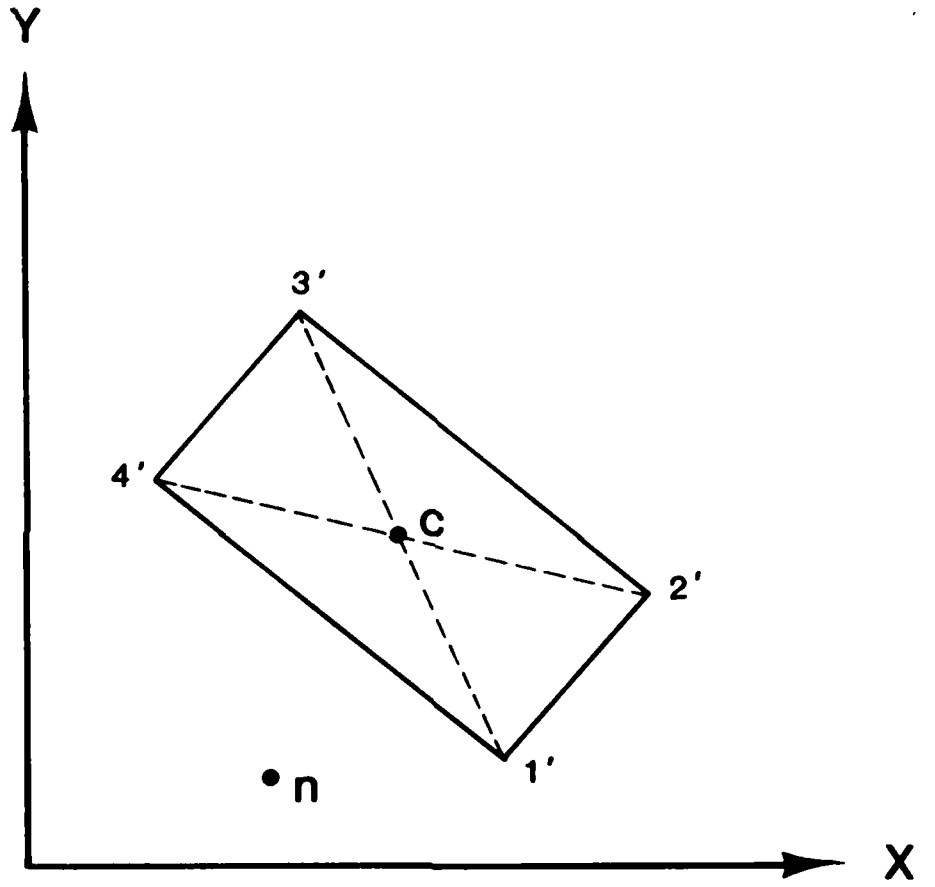


Figure E-2.

Illustration of Dropped Display, Dropped Display Center, C, and a Nadir, n.

This Appendix develops an algorithm for determining the following:

1. Is the nadir, n, inside the dropped display rectangle?

2. If the nadir is outside the dropped display, then what lines defined by what corners, bound the scene in the X-Y plane? (i.e., lines 4′-n and 1′-n bound the scene in Figure E-2).

Problem: The current **REAL SCAN** software only allows eye orientations in which the lower bound line is always between corners 3 and 4 in Figure E-1. It is desired to have the capability of computing the scene for any arbitrary eye orientation.

Solution: Figure E-3 depicts the location of the nadir relative to the four dropped screen corners. There are nine possible locations for the nadir (labeled A-I) in Figure E-3 for the eye above the screen.

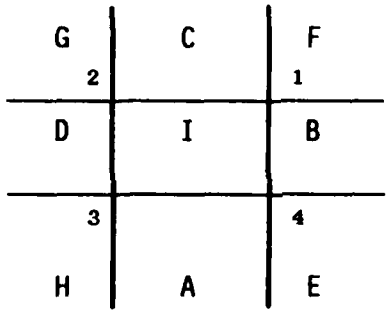


Figure E-3. Nine Possible Nadir Locations.

The four screen corners are labeled from 1 to 4 in a counter clockwise manner. Corner 2 corresponds to the pixel location (1,1) and corner 4 corresponds to the pixel location (NX,NY).

Figure E-3 illustrates the five basic nadir locations which determine the scan line sequence. These five basic situations are shown in Figures E-4 through E-8.

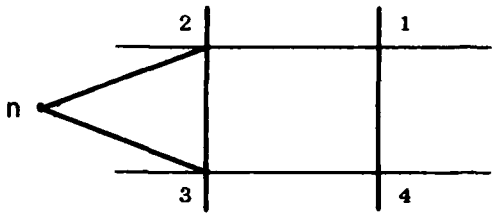


Figure E-4
Nadir Outside Screen Footprint.
(1 lower bound line)

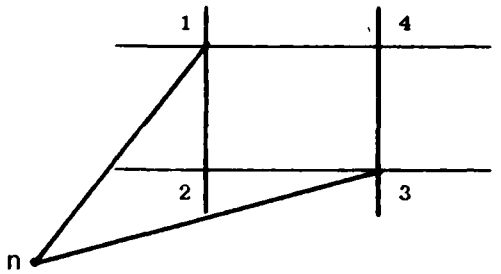


Figure E-5
Nadir Outside Screen Footprint.
(2 lower bound lines)

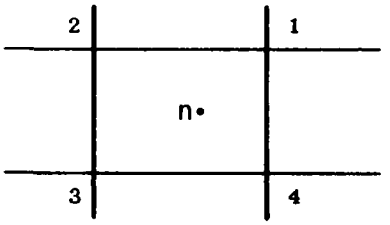


Figure E-6
Nadir Inside Screen Footprint.

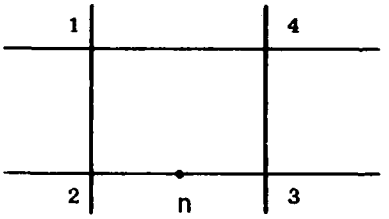


Figure E-7
Nadir On Screen Boundary Line.
(1 lower bound line)

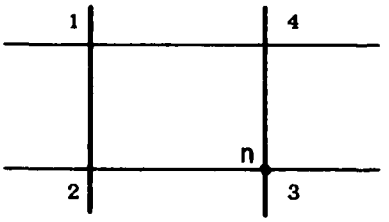


Figure E-8
Nadir = Corner

Figure E-3 suggests a separation into three cases for the nine possible nadir locations. There is an edge case (A,B,C,D), a corner case (E,F,G,H) and inside case (I). Figure E-4 corresponds to the edge case, Figure E-5 corresponds to the corner case, and Figure E-6 corresponds to the inside case. Two other cases exist. These cases

correspond to the nadir falling either on one or two of the lines that bound the nine regions of Figure E-3. Figure E-7 corresponds to the case where the nadir falls on one region separation line, while Figure E-8 corresponds to the case where the nadir falls on one display corner.

The location of the nadir can be found by taking the cross products of adjacent corners. Table E-1 identifies the sign of the cross product for all nine nadir locations, when the eye is above the screen. If the eye is below the screen, then all signs are reversed.

TABLE E-1. CROSS PRODUCT SIGNS WITH THE EYE ABOVE THE SCREEN

NADIR LOCATION	CROSS PRODUCT SIGNS 4x1	1x2	2x3	3x4	BOUNDING CORNERS
A	+	+	+	-	3,4
B	-	+	+	+	4,1
C	+	-	+	+	1,2
D	+	+	-	+	2,3
E	-	+	+	-	3,4,1
F	-	-	+	+	4,1,2
G	+	-	-	+	1,2,3
H	+	+	-	-	2,3,4
I	+	+	+	+	1,2,3,4

Table E-1 suggests that one can determine a logical corner test via a Karnaugh map. Figure E-9 is a Karnaugh map with the four cross products as independent variables and the bounding corners as dependent variables labeled on the map. To further facilitate the translation from Table E-1 to Figure E-9 the nadir location is also labeled on the Karnaugh map. Figure E-9 is the corresponding Karnaugh map for the eye below the screen center.

(4x1)(1x2) \ (2x3)(3x4)	++	+-	--	-+
++	INSIDE $_I$	3,4 $_A$	2,3,4 $_H$	2,3 $_D$
+-	1,2 $_C$			1,2,3 $_G$
--	4,1,2 $_F$			
-+	4,1 $_B$	3,4,1 $_E$		

Figure E-9. Eye Above the Screen Conditions.

(4x1)(1x2) \ (2x3)(3x4)	++	+-	--	-+
++			4,1,2 $_F$	
+-			4,1 $_B$	3,4,1 $_E$
--	2,3,4 $_H$	2,3 $_D$	INSIDE $_I$	3,4 $_A$
-+		1,2,3 $_G$	1,2 $_C$	

Figure E-10. Eye Below the Screen Conditions.

The Karnaugh maps yield the following corner tests to define the field of view lower boundary:

1. Corner = 1 if the EXCLUSIVE OR of the signs of (4x1) and (1x2) is true.

2. Corner = 2 if the EXCLUSIVE OR of the signs of (1x2) and (2x3) is true.

3. Corner = 3 if the EXCLUSIVE OR of the signs of (2x3) and (3X4) is true.

4. Corner = 4 if the EXCLUSIVE OR of the signs of (3x4) and (4x1) is true.

Further, an intermediate corner contributes to defining the lower bondary, as shown in Figure E-5, if the following conditions are satisfied:

1. Center point = 1 if the eye is above the screen and ((4x1) < 0) and ((1x2) < 0) or if the eye is below the screen and ((4x1) > 0) and ((1x2) > 0).

2. Center point = 2 if the eye is above the screen and ((1x2) < 0) and ((2x3) < 0) or if the eye is below the screen and ((1x2) > 0) and ((2x3) > 0).

3. Center point = 3 if the eye is above the screen and ((2x3) < 0) and ((3x4) < 0) or if the eye is below the screen and ((2x3) > 0) and ((3x4) > 0).

4. Center point = 4 if the eye is above the screen center and ((3x4) < 0) and ((4x1) > 0) or if the eye is below the screen center and ((3x4) < 0) and ((4x1) > 0).

One may note that the nine nadir locations in Table E-1 are not expanded to new cases if one or two cross products are zero and if one assigns a positive sign to a zero cross product. However, the two cases illustrated in Figures E-7 and E-8 suggest that the algorithm which determines the nadir location and hence, accommodates arbitrary eye orientation will have some best test order. That order appears to be as follows:

1. Determine the four cross products.

2. Test if two of the cross products are zero. If two cross products are zero one has the case illustrated in Figure E-8 and the boundaries of all the scan lines are well defined.

3. If two of the cross products are not zero, then establish four logical variables having the sign of the cross products (i.e., .TRUE. corresponds to positive).

4. Use the Exclusive OR tests to determine which display corners bound the scene.

5. Establish four logical variables which EQUIVALENCE the sign of each cross product with whether the eye is above or below the screen center (i.e., the new variable is true if the sign is positive and the eye below or if the sign is negative and the eye above).

6. Use the center point AND test to determine which, if any, one display corner becomes a center point.

7. If there is no center point, then test for zero of the cross product of the two corners.

8. If that cross product is zero, then the nadir is inside the footprint as per Figure E-3 case I, but only three triangles of scan lines need be created.

9. If that cross product is not zero, then one of the cases A through D of Table E-1 has been determined.

10. If there is a center point, then initiate a set of tests for cross product zeros. The first test checks for a zero cross product for one corner and the center point. If a zero results then that corner is replaced with the center point, and the center point value is cleared to zero indicating cases A through D. If the first test does not yield a zero cross product then the second test is executed. The second test checks for a zero cross product of the other corner and the center point. If this cross product is zero then the corner is substituted and the center point cleared as in the first test.

11. If neither of these cross product tests with the center point yield a zero, then cases E through H of Table E-1 have been determined.

12. Finally, case I is determined if all the above tests fail. One should note that case I cannot be tested prior to cases A through H, since a zero cross product receives a positive sign, which make steps 2 and 8 equivalent to case I. Case I divides the scene into four triangles, Step 2 divides the scene into two triangles, while Step 8 divides the scene into three triangles.

NPIX, IASIX SEQUENCE

The value of NPIX and IAXIS must be identified. NPIX is the last pixel value for the scan line on the IAXIS coordinate. If we employ a corner counter "KSC" to keep track of the actual outer screen corner to be crossed next, then we can use KSC to identify NPIX and IAXIS.

Table E-2 identifies the NPIX and IAXIS values for each value of KSC when the eye is above the screen. Thus each time an outer corner is crossed, KSC will be updated to MOD(KSC,4)+1 and the corresponding values of NPIX, and IAXIS are found by a table lookup. This assumes that the scene is swept from the corner having the smaller index toward the corner with the larger index.

TABLE E-2. NPIX AND IAXIS VERSUS KSC.

KSC	NPIX	IAXIS
1	NX	1
2	0	2
3	0	1
4	NY	2

APPENDIX F

IMPROVED SCAN LINE INITIALIZATION PROCEDURE

This Appendix develops a scan line initialization procedure which efficiently bypasses the near field scene. Hence, over sampling as in the case of looking directly down at the scene and, over testing as in the case of looking out to the horizon are substantially reduced. The procedure is not scene dependent, hence an arbitrary single valued elevation will still be faithfully reproduced.

Figure F-1 illustrates the side view of a scan line. The scan line is initiated at the nadir directly below the eye. The scene is calculated for each world coordinate along the line. Figure F-2 illustrates the width of a scan line. The scan line has width since we seek to maintain constant display resolution, and the dimension of a world area which projects to a constant display resolution area increases with range. Figures F-1 and F-2 illustrate that the angle subtended by a pixel is $B = \alpha y / \sqrt{h^2 + y^2}$ and $B = \alpha y/h$; if $h >> y$ or $B = \alpha$; if $h << y$.

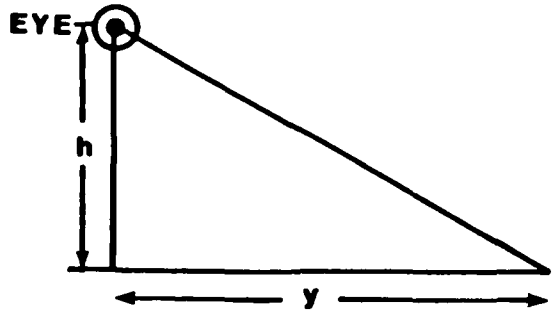


Figure F-1. Side view of Scan Line.

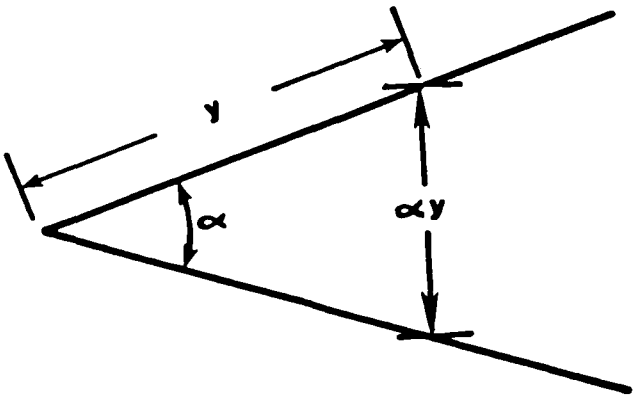


Figure F-2. Top view of Scan Line.

Figures F-3 and F-4 illustrate two ways of representing scan lines. Figure F-3 illustrates a correct geometric layout, where all scan lines converge on the nadir. Figure F-4 spreads the scan lines at the nadir so that one may better represent detail in the vicinity of the nadir. Figure F-4 suggests an improved scan line initiation sequence over starting each scan line at the nadir. Figure F-4 does not indicate the final procedure developed in this Appendix, but it is a good starting point for describing the procedure's development.

Figure F-4 suggests that the first scan line (1) starts at the nadir (i.e., left boundary of Figure F-4) and proceeds to scan the screen moving toward the right. The next scan line is line (2) and starts at a distance of h/8 from the nadir and an angle of 8α from line (1). The next scan line is line (3) which starts no nearer than h/4 to the nadir. If the first visible points having distance equal or greater than h/4 from the nadir along lines (1) and (2) are more distant than h/4, then line (3) starts at the minimum of these line (1) and line (2) start distances. Figure F-4 also illustrates that after line (9) has been swept through the scene, the procedure repeats with line (2) now becoming line (2′). The new line corresponding to (1) is line (1′), separated from (2 - 2′) by angle 8 α. Line (1′) starts testing the scene for visible data at the nadir.

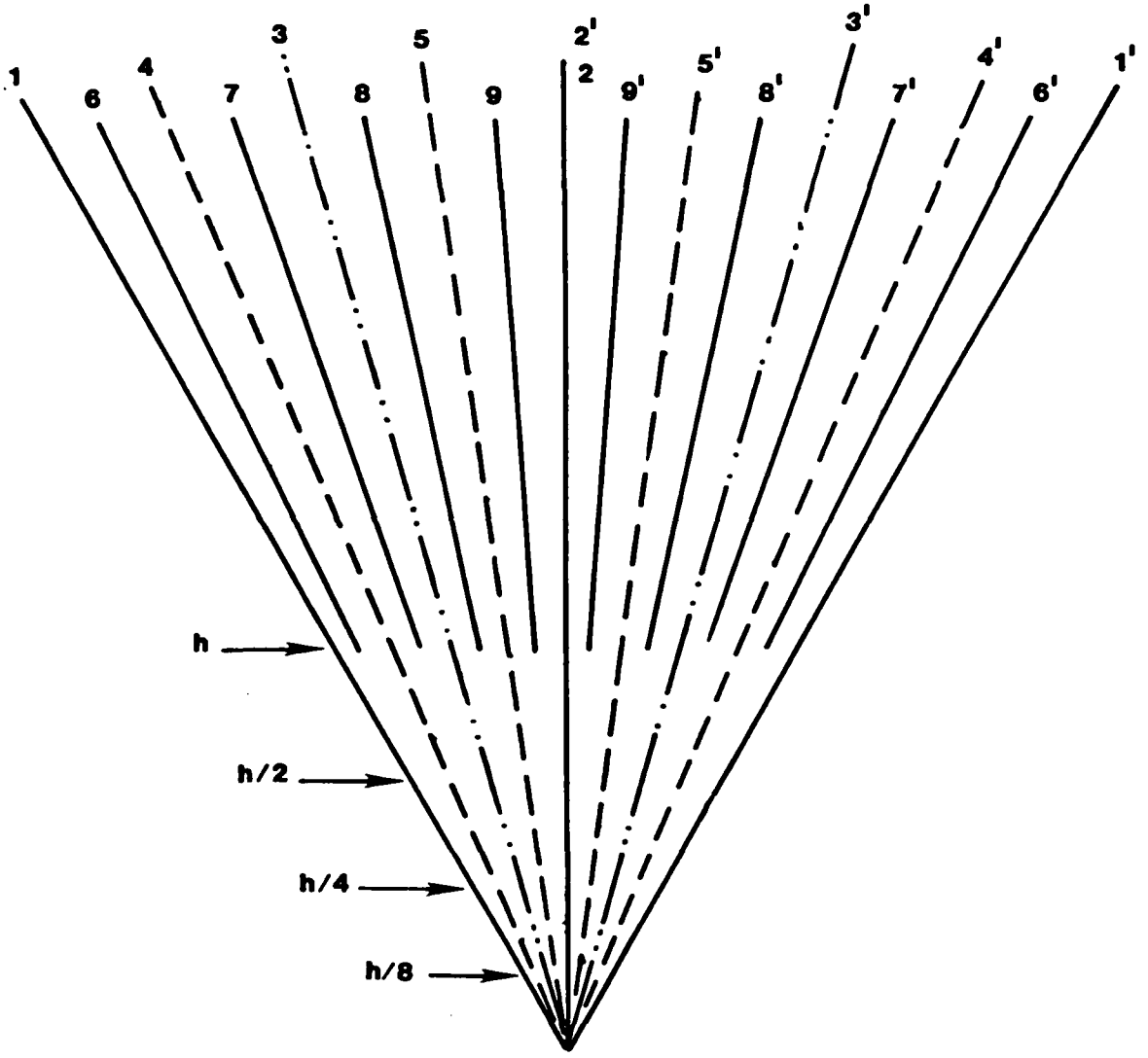


Figure F-3. Illustration of the Scan Line Initiation Sequence (1, 1, 3 ... 8, 9) and (1′, 2′, 3′, ... 8′, 9′).

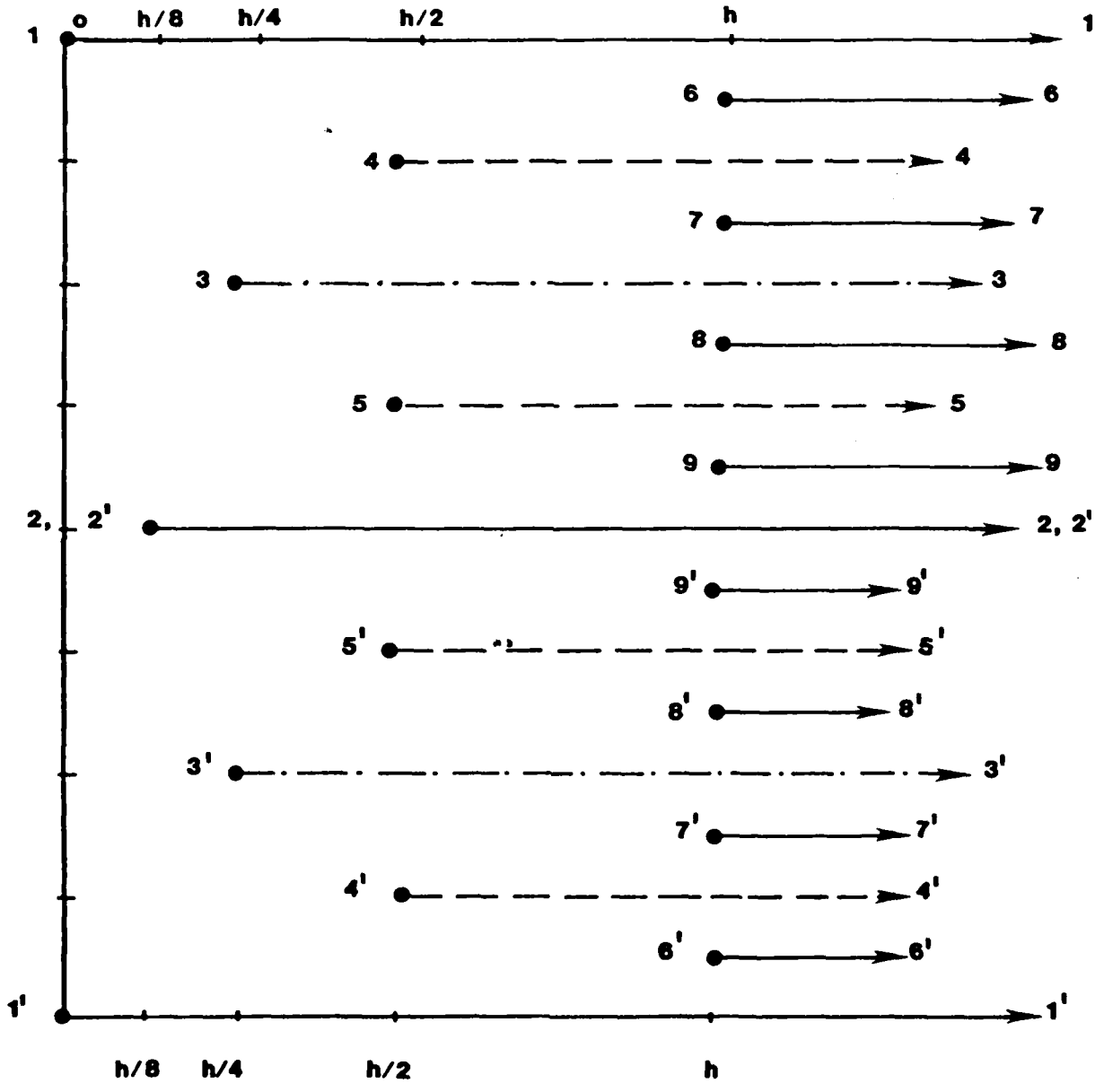


Figure F-4. Another Illustration of Scan Line Initiation Sequence.

Figures F-3 and F-4 clearly demonstrate that an improved scan line initiation procedure is possible. The problem is to develop the most efficient scan line initialization procedure.

Let us consider one more example of an improved scan line initiation procedure before developing what appears to be the most efficient procedure. Figure F-5 illustrates a sequence of scan lines (1-1, 2-1, 1-2, and 2-2) given line (1-0), and the renaming of line (1-2) to line 1-0 so that the procedure repeats. V1 and V1′ represent vector storage of data needed to implement the scan line initialization procedure. Figure F-5 illustrates the following procedure:

1. Given vector V1 for line (1-0). Vector V1 contains the following information:

   a. The location of the first visible point having distance equal or greater than h/2 from the nadir, say V1(h/2).

   b. The location of the first visible point having distance equal or greater than h from the nadir, say V1(h).

2. Initiate line (1-1) at the ICASE axis distance from the nadir given in vector V1(h/2). [Given a cartesian coordinate system on a plane and a line on the plane, as one moves along the line, incremental distance along one of the axes will change more rapidly. This faster changing coordinate axis is called the ICASE axis.]

3. Store the location of the first visible point along line (1-1) having distance equal or greater than h from the nadir in vector V1′, say V1′(h).

4. Calculate the scene along line (1-1) and project visible points to appropriate buffer locations.

5. Initiate line (2-1) at the ICASE axis distance given by the minimum of the V1(h) and V1′(h) distances from the nadir.

6. Calculate the scene along line (2-1).

7. Initiate line (1-2) at an ICASE axis distance of one unit from the nadir.

8. Store the location of the first visible point along line (1-2) having distance equal or greater to h/2 from the nadir in vector V1, say V1(h/2).

9. Store the location of the first visible point along line (1-2) having distance equal or greater to h from the nadir in vector V1, say V1(h).

10. Initiate line (2-2) at the ICASE axis distance given by the minimum of the V1(h) and V1′(h) distances from the nadir (i.e., the ICASE axis is either an X or Y axis. The ICASE axis has the smallest angular separation from the scan line.)

11. Line (1-2) now corresponds to line (1-0) and one repeats steps 1 through 10 until the scene has been calculated.

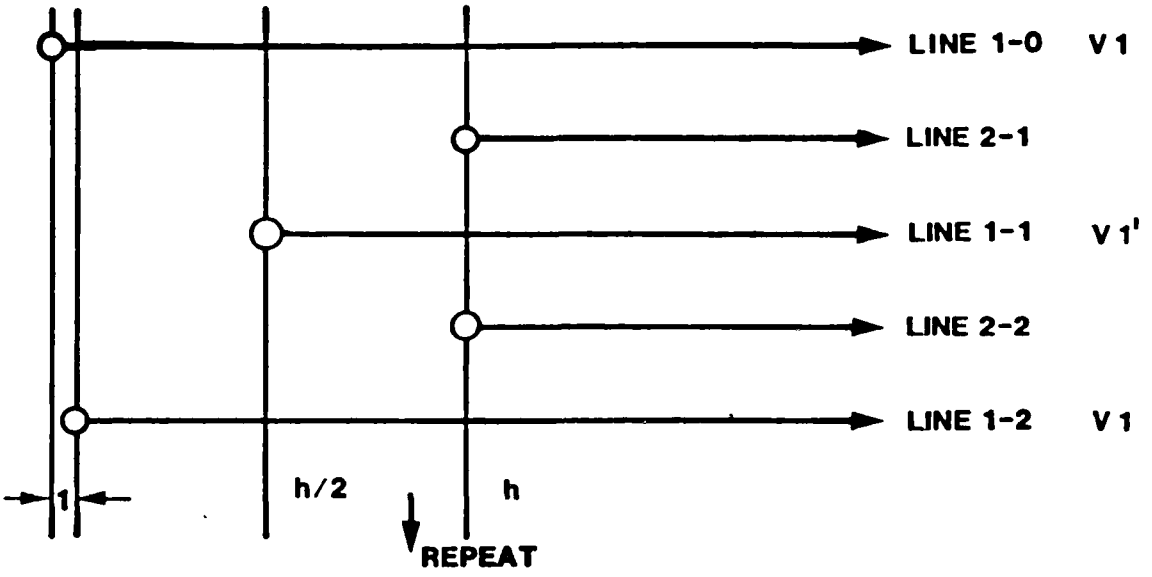


Figure F-5. Illustration of h/2 Starting Point.

Let us compare the improved scan line procedures suggested by Figures F-4 and F-5. Figure F-5 over calculates the scan line length from 1 to approximately h/2 on all lines corresponding to (1-0) or (1-2). The angular separation between all scan lines is, α, the desired resolution for both Figure F-4 and Figure F-5.

Figure F-4 illustrates a relatively complex initiation procedure. If the starting distances correspond to distances h/8, h/4, h/2, and h, then it is clear that over sampling occurs since the first visible point may be further from the nadir. However, if a vector memory, similar to that used for Figure F-5 is considered, then the dimension of the memory appears to be large.

The optimum scan line initiation procedure is illustrated in Figure F-6. Only one vector memory is required. If the display accuracy corresponds to 1/1024 then no more than 33 locations are required in V, the vector memory. (11 point locations, where each point has X, Y, Z axis values or where each point has only the ICASE axis value and Z for 22 memory locations.) The procedure illustrated in Figure F-6 for initiating the scan lines is as follows:

1. Initiate line (1-0)

   a. Store the location of the first visible point whose distance is equal or greater than Rh from the nadir into V, say V(Rh) (R corresponds to the resolution, for example R = 1/1024).

   b. Continue to store the location of the first visible points whose distance is equal or greater than 2^nRh from the nadir into V(2^nRh), until $2^n R = 1$.

2. Initiate line (1-1), separated from line (1-0) by angle 2α, at the ICASE axis distance given by V(h/2). Store the location of the first visible point whose distance is equal or greater than h from the nadir into V(h´).

3. Initiate line (2-1), midway between lines (1-0) and (1-1) at the ICASE axis location given by the minimum of V(h) and V(h´). Then store V(h´) into V(h), updating the vector V.

4. Initiate line (1-2), separated from line (1-1) by angle 2α, at the ICASE axis distance given by V(h/4). Store the location of the first visible point whose distance from the nadir is greater or equal to h/2 into V(h/2). Store the location of the first visible point whose distance from the nadir is greater or equal to h into V(h´).

5. Initiate line (2-2) as per step 3 and store V(h´) into V(h).

6. Succeeding steps are as illustrated in Figure F-6, following the concept laid out in steps 1 through 5.

Figure F-6 illustrates an apparent optimum procedure for a general scene. The minimum memory is used, and no scan line is started nearer the nadir than required to guarantee calculation of telephone pole like spikes in the scene having resolution equal to or better than a pixel, since the resolution, R, is equal to or less than ½ pixel and the angular resolution, α, is also equal to or less than ½ pixel.

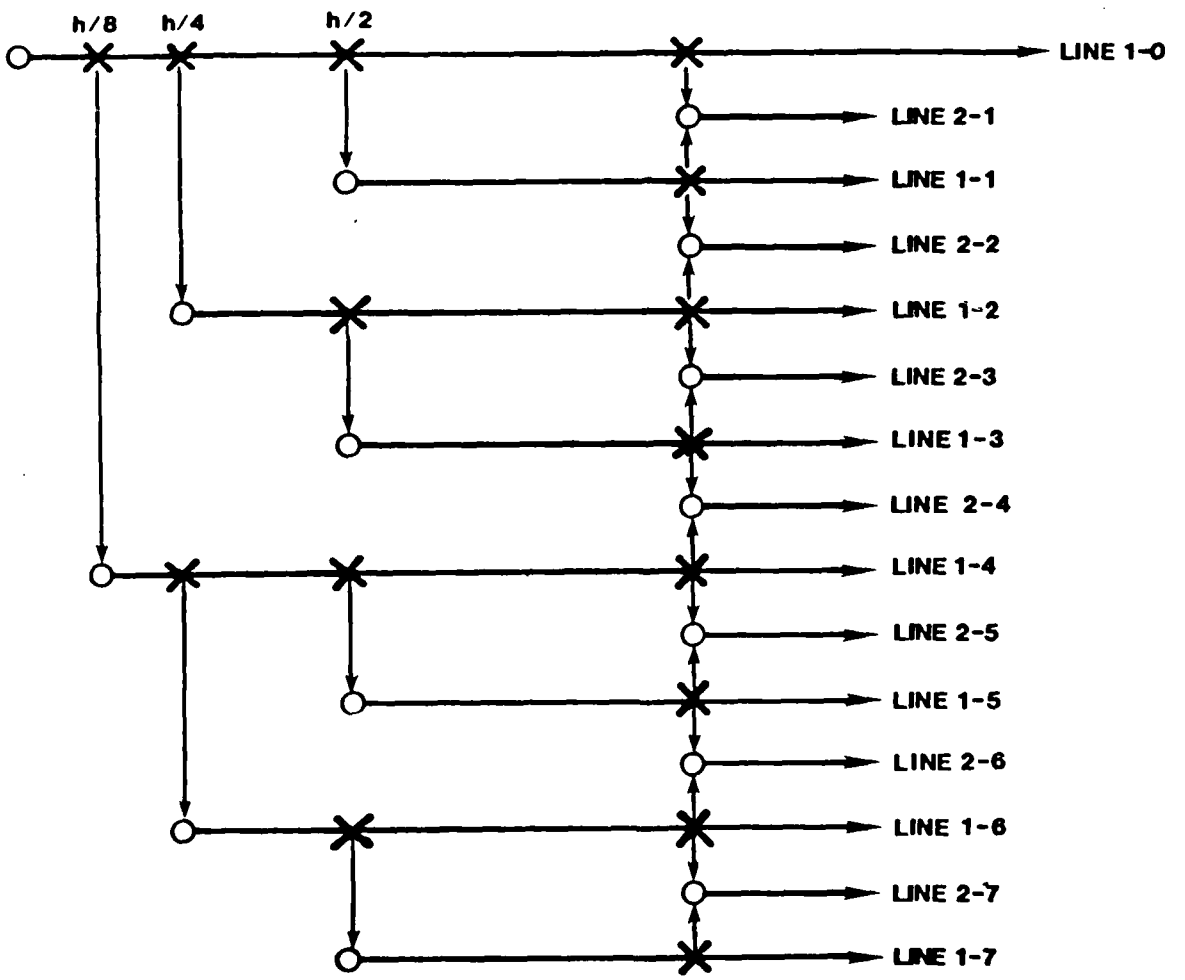


Figure F-6. The Optimum Scan Line Initiation Procedure.

APPENDIX G

DISPLAY PROJECTION TO FRACTIONAL PIXEL ACCURACY

The current **REAL SCAN** algorithms project world data (i.e., the scene) to the 512 x 512 pixel display via a Bresenham type algorithm operating on both the display's X and Y axes. The current **REAL SCAN** algorithm does not resolve to subpixel accuracy, its accuracy is ± 1/2 pixel of the projected pixel location.

There are a number of possible projection methods for achieving subpixel accuracy. All of the projection methods investigated in this appendix assume that one axis is projected from the scene to the display with subpixel resolution. The other axis' coordinate is then derived. Some of the methods for deriving the other axis' coordinate are as follows:

1. Since the equation for a line on the display, Y(X), is

   $$Y - Y_o = (\Delta Y/\Delta X)\ (X - X_o) \qquad (G\text{-}1)$$

   one can continue to perform multiplications $m(X-X_o)$ to determine the other axis, Y. The following symbol notation is used:

   $\Delta Y/\Delta X$ is the slope m, of the line

   ΔX_i, ΔY_i are increments along their respective axis

2. One can set up an error term

   $$\text{Error} = m\ (X_i - X_o) - (Y_i - Y_o) + m\ (X_{i+1} - X_i)$$

   such that the error term becomes $(Y_{i+1}-Y_i)$, the increment along the other axis, since X is assumed to be the primary axis.

3. One can set up a Bresenham type algorithm of the form

   $$\text{Error} = (-(Y-Y_o)\ \Delta X + (X-X_o)\ \Delta Y)\ 2^n + \Delta Y\ 2^{n-1}$$

   where n is the number of bits of accuracy (i.e., n = 1 if ΔX and ΔY are expressed as an integer, while n = 3 if X and Y are expressed on a grid whose dimension is 1/4 pixel).

a. A divide algorithm can be set up making use of the limited accuracy. For instance the pseudo **FORTRAN** code below relates to a grid of 1/4 pixel dimension

```
C   Initialize IEEE = -ΔX*4
C   Display coordinates (X,Y) consist of an integer cat-
C   enated with a pseudo 2 bit fraction.  Hence, for a 512 x
C   512 display the minimum integer dimension for X or Y is
C   1(for sign) + 9 (for display dimention) + 2(for pseudo
C   fraction) = 12 bits.
C   ITERATION
    IEEE = IEEE + ΔX_i*ΔY
C   ΔX_i is the change in the major coordinate axis since the
C   last iteration.
C   ΔX_i may have values of 2, 3 or 4.
    IF(IEEE.GT.0) THEN ! 1
        IEEE = IEEE - ΔX*4
    IF(IEEE.GT.0) THEN ! 2
        IEEE = IEEE -ΔX*2
    IF(IEEE.GT.0) THEN ! 3
        IEEE = IEEE -ΔX*2
        Y = Y + INCY
C  the terms ΔX*4, ΔX*2, INCY, INCY/4, INCY/2, and 3INCY/4
C  are unique constants
    ELSE ! 3
        Y = Y + 3INCY/4
    ENDIF ! 3
    ELSE ! 2
        IEEE = IEEE + ΔX*2
    IF(IEEE.GT.0) THEN ! 4
        IEEE = IEEE - ΔX/4
        Y = Y + INCY/2
    ELSE ! 4
        Y = Y + INCY/4
    ENDIF ! 4
    ENDIF ! 2
    ENDIF ! 1
```

NOTE: No more than 7 steps are required to complete this effective 2-bit division.

b. An actual divide can be set up as illustrated in the following pseudo **FORTRAN** code

```
C   ITERATION
        IEEE = IEEE + ΔX_i*ΔY
    IF(IEEE.GT.0) THEN ! 1
        ΔY_i = (IEEE + ΔX/2)/ΔX
```

$Y = Y + \Delta Y_i$

$IEEE = IEEE - \Delta Y_i * \Delta X$

ENDIF ! 1

NOTE: The true divide algorithm, item 3.b, requires 5 steps to determine the new value of Y whereas the algorithmic 2-bit resolution divide, item 3.a, requires 7 steps without recourse to a divide operation.

Hence, it appears that the algorithm for calculating the display line is best given by 3.a for limited resolution. Actual screen pixel location would be obtained by dividing (X,Y) by 4 (i.e., a 2 bit shift). However, the subpixel resolution of the display allows one to perform final display filtering based upon subpixel point spread filter functions, one of the investigation goals of this research.

Integer display screen resolution can be achieved by using a display screen X coordinate with pseudo fractional part, yet dealing with a display screen Y coordinate accurate to an integer pixel ± 1/2. For example, **FORTRAN** code for a two-bit X fraction and an integer Y coordinate is developed as follows:

1. Let X = N.F, a 3 bit integer, then N = 0, F = 01 represents X = 1/4.

2. Initialize the error term, IEEE to $-\Delta X*4$

3. Iterate IEEE according to the **FORTRAN** formula
   $IEEE = IEEE + (2*\Delta Y) * \Delta X_i - (8*\Delta X) * \Delta Y_i$

4. Let the variables $(\Delta X*4)$, $(2*\Delta Y)$, and $(8*\Delta X)$ be constants of the process.

5. Then pseudo **FORTRAN** code becomes

```
    IEEE = IEEE + (2*ΔY) *ΔX_i
IF(IEEE.GT.0) THEN
    IEEE = IEEE - (8*ΔX)
    Y = Y + INCY
ENDIF
```

APPENDIX H

IMPROVED WORLD TO DISPLAY PROJECTION

This Appendix develops an improved projection algorithm based upon given accuracy requirements.

Problem: To project a sequence of points from world coordinates to screen coordinates using the fewest recurring steps in integer arithmetic.

Given: (1) the accuracy in both world and screen coordinates; (2) the sequence of points falling on a line cutting the screen and on a plane through the eye nadir axis.

Solution: Projection to the screen is accomplished by a two-step process. First, the point is rotated from world to screen coordinates, both having their origin at the eye. Second, the point is projected to the screen using similar triangle geometry.

Given: $(Xp,Yp,Zp)_w$, the visible point in world coordinates

$(Xe,Ye,Ze)_w$, the eye point in world coordinates

(Up,Vp,Wp), the visible point in eye centric screen coordinates

(Xs,Ys), the projection of the visible point to screen coordinates

(R), the 3x3 rotation matrix transforming the visible point from eye centric world coordinates to eye centric screen coordinates

The rotation matrix coordinate transformation yields

$$\begin{bmatrix} Up \\ Vp \\ Wp \end{bmatrix} = [R] \begin{bmatrix} Xp-Xe \\ Yp-Ye \\ Zp-Ze \end{bmatrix} \qquad \text{(H-1)}$$

But we also have two lines:

(1) The line on the screen in screen coordinates

$$Ys = Xs * ms + Yso$$

(2) The scan line on the world

$$Yp-Ye = m(Xp-Xe)$$

or in terms of the scan line's horizon

$$Yh-Ye = m(Xh-Xe) \tag{H-4}$$

Equation H-3 suggests Equation H-1 may be written for $R = (r_{ij})$ as

$$\begin{bmatrix} Up \\ Vp \\ Wp \end{bmatrix} = \begin{bmatrix} r_{11} + m^*r_{12} & r_{13} \\ r_{21} + m^*r_{22} & r_{23} \\ r_{31} + m^*r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} Xp-Xe \\ Zp-Ze \end{bmatrix} \tag{H-5}$$

One can rename the rotation matrix for this 2-D to 3-D transformation as $A = (a_{ij})$

$$\begin{bmatrix} Up \\ Vp \\ Wp \end{bmatrix} = [A] \begin{bmatrix} Xp-Xe \\ Zp-Ze \end{bmatrix} \tag{H-6}$$

Any visible point projected to the screen, a distance h from the eye must satisfy

$$\frac{Xs}{Up} = \frac{-Ys}{Wp} = \frac{h}{Vp} \tag{H-7}$$

If $m < 1$, then Xp-Xe is the rapidly changing axis (i.e., termed the ICASE axis) and one can determine incremental changes in Xs and Ys given incremental changes in Xp and Zp as follows:

1. Use Equations (H-7) and (H-2) to solve for either Ys or Xs

$$Ys = Xs*ms + Yso = Xs*Wp/Up \tag{H-8}$$

so

$$Xs(ms*Up + Wp) = Yso*Up \tag{H-9}$$

2. Define the terms

$$Up = Upo + \Delta Up$$
$$Wp = Wpo + \Delta Wp$$
$$Xs = Xsold + \Delta Xs$$

$$SAV1 = ms*Upo + Wpo$$
$$SAV2 = ms*\Delta UP = \Delta Wp,$$

then one can write the pseudo **FORTRAN** code for calculating ΔXs and iterating the terms above

```
SAV2 = ms*ΔUp + ΔWp
SAV1 = SAV1 + SAV2
IEEE = IEEE - ΔUp*Yso - Xsold*SAV2
ΔXs = IEEE/SAV1
IEEE = IEEE -ΔXs*SAV1
Xsold  Xsold + ΔXs
```

3. Pseudo **FORTRAN** code for ΔYs follows from Equation (H-2)

   ΔYs = ΔXs*ms

If one relates the accuracy of (Xs,Ys) as N.F where

$$|N| \leq 2^n$$
$$|F| * 2^f \leq 1,$$

and if we relate X_s with the fast changing axis, Y_s with the slow changing axis; then X_s needs (n+f) bits of significance where Y_s needs (n2f) bits of significance to account for roundoff.

The number of steps for determining (ΔXs, ΔYs) can be reduced by rewriting Equation (H-9) in terms of ground coordinates and allowing a floating point initialization.

This is done as follows:

1. Combine Equations (H-9) and (H-5) to form

   $$Xs((ms\ a_{11} + a_{31})\ (Xp-Xe) + (ms\ a_{12} + a_{32})(Zp-Ze) =$$
   $$-Xso\ (a_{11}\ (Xp-Xe) + A_{12}\ (Zp-Ze))$$

where

$$b_{11} = ms\ a_{11} + a_{31}$$
$$b_{12} = ms\ a_{12} + a_{32}$$

2. The pseudo **FORTRAN** follows the definitions

   Xp-Xe = Xo + ΔXp

   Zp-Ze = Zo + ΔZp

   Xs = Xsold + ΔXs

$$SAV1 = b_{11} * Xo + b_{12} * Zo$$

$$SAV2 = b_{11} * \Delta Xp + b_{12} * \Delta Zp$$

$$SAV3 = a_{11} * Xo + a_{12} * Zo$$

$$SAV4 = a_{11} * \Delta Xp + a_{12} * \Delta Zp$$

which yields the iteration

$$SAV2 = b_{11} * \Delta Xp + b_{12} * \Delta Zp$$

$$SAV1 = SAV1 + SAV2$$

$$SAV4 = a_{11} * \Delta Xp + a_{12} * \Delta Zp$$

$$SAV3 = SAV3 + SAV4$$

$$IEEE = IEEE - Yso*SAV3 - Xsold*SAV2$$

$$\Delta Xs = IEEE/SAV1$$

$$IEEE = IEEE\ \Delta Xs*SAV1$$

$$Xsold = Xsold + \Delta Xs$$

$$\Delta Ys = \Delta Xs*ms$$

$$Ys = Ys + \Delta Ys$$

This shows that projection can be accomplished with eight (8) multiplications, 9 add/subtracts(s) and one f bit division. This is less complexity than required for the 3x3 rotation as per Equation (H-1). The number of bits required for each variable is shown in Table H-1.

TABLE H-1. VARIABLE RESOLUTION

Variable Name	Resolution for N.F Number Bits
ΔXs	f
ΔYs	n + 2f
Ys	n + f
ms	n + f
a_{ij}	n + f
b_{ij}	n + f
ΔXp	f
ΔZp	f
SAV1	2n + 2f
SAV2	n + 2f
SAV3	2n + 2f
SAV4	n + 2f
Xsold	n + f
Yso	n + f
IEEE	2n + 3f

APPENDIX I

TERRAIN MODELING DISTRIBUTIONS

This Appendix develops three redistributions which can be applied to filtered white noise. The distributions developed allow one to control the relative occurrence of terrain elevations. Hence, different type of ground cover can be modeled.

Triangular Distribution

A triangular distribution has the general frequency diagram shown in Figure I-1.

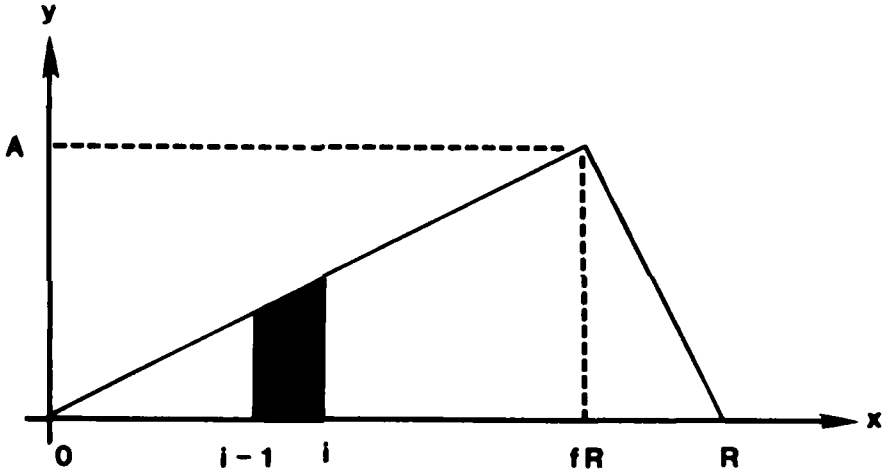


Figure I-1. Triangle Distribution Plot.

The equations describing Figure I-1 are

$$y(x) = ax; \quad \text{for } 0 \leq x \leq fR \tag{I-1}$$

$$y(x) = b(R - x); \quad \text{for } fR \leq x \leq R \tag{I-2}$$

The area under the distribution curve is AR/2, but since this encompasses all points,

$$AR/2 = N^2 \tag{I-3}$$

or

$$A = 2N^2/R \tag{I-4}$$

Note that the location of the peak does not effect the area calculation.

Letting NT(i) be the number of points in the ith bucket covering the interval from x = i - 1 to x = i in range 0 to fR (i.e., shaded area in Figure I-1) then

$$NT(i) = \int_{i-i}^{i} ax\,dx \qquad \text{(I-5)}$$

$$NT(i) = (a/2)(2i - 1) \qquad \text{(I-6)}$$

Integrating Equation (I-1) yields

$$A = afR \text{ or } a = A/fR \qquad \text{(I-7)}$$

Using Equations (I-7) and (I-4)

$$a = 2N^2/(fR^2) \qquad \text{(I-8)}$$

Therefore, from (I-6) and (I-8), the number of points in the ith bucket on the interval 0 to fR is

$$NT(i) = N^2\ (2i - 1)/(fR^2) \qquad \text{(I-9)}$$

Shifting the origin to x = R, the bucket sizes for the interval fR to R can be computed

$$NT(i) = N^2\ (2i' - 1)/((1 - f)R^2) \qquad \text{(I-10)}$$

where $i' = R - i + 1$.

Therefore,

$$NT(i) = N^2\ (2R - 2i + 1)/((1 - f)R^2) \qquad \text{(I-11)}$$

on the interval $fR \leqq i \leqq R$

The **FORTRAN** code that computes these bucket sizes follows:

```
C  IFRAC = f*R
C  FRAC = f
C  BUCK(I) = NT(i)
C
C  TRIANGLE
C
   2101    TEMP1 = (1.*N*N)/R**2
```

```
          TEMP2 = 2*R + 1
          DO  2111 I = 1,R
              IF(I.LE.IFRAC)THEN
                  BUCK(I) = (2*I - 1)*TEMP1/FRAC
              ELSE
                  BUCK(I) = (TEMP2-2*I)*TEMP1/(1 - FRAC)
          ENDIF
      2111 CONTINUE
           GOTO 2114
```

Parabolic Distribution

A split parabolic distribution is shown in Figure I-2.

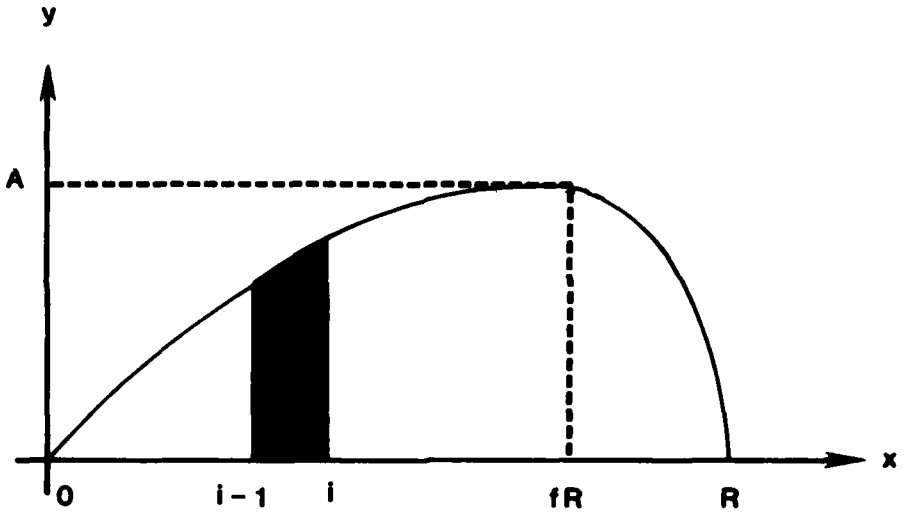


Figure I-2. Parabolic Distribution Plot.

The equations describing the split parabolic distribution are

$$y(x) = ax(1 - x/2fR); \quad \text{for} \quad 0 \leq x \leq fR \qquad \text{(I-12)}$$

$$y(x) = b(R-x)\,[1-(R-x)/(2(1-f)R)]; \quad \text{for } fR \leq x \leq R \qquad \text{(I-13)}$$

The area under the distribution curve is 2AR/3, and also N^2. It follows that

$$A = 3N^2/(2R) \qquad \text{(I-14)}$$

Letting NP(i) be the number of points in the ith bucket, for the shaded area in the Figure I-2, which is in the interval 0 to fR, yields

$$NP(i) = \int_{i-1}^{i} ax(1 - x/2fR)\,dx$$

$$NP(i) = (a/2)(2i - 1) - (a/6fR)(3i^2 - 3i + 1) \qquad \text{(I-15)}$$

Integrating Equation (I-12) yields

$$A = afR/2 \qquad \text{(I-16)}$$

and, from Equations (I-14) and (I-16)

$$a = 3N^2/(fR^2) \qquad \text{(I-17)}$$

Therefore, from Equations (I-15) and (I-17),

$$NP(i) = (3N^2/2fR)[2i-1-(3i^2-3i+1)/(3fR)] \qquad \text{(I-18)}$$

Shifting the origin to i = R, yields

$$NP(i) = [3N^2/2(1 - f)R^2](2i'-1-(3i'^2-3i'+1)/[3(1-f)R]) \qquad \text{(I-19)}$$

where $i' = R-i-1$ in the interval $fR+1 \leqq i \leqq R$

The **FORTRAN** routine for the parabola bucket size calculation follows:

```
C   FRAC = f
C   IFRAC = f*R
C   BUCK(I) = NT(i)
C
C        PARABOLA
C
 2102    TEMP1 = (3.*N*N)/(2*R**2)
         DO 2112 I = 1,R
            IF(I.LE.IFRAC)THEN
               BUCK(I) = (TEMP1/FRAC)*(2*I-1-((1+3.*I*(I-1))/
                         (3.*FRAC*R)))
     *
            ELSE
               TEMP2 = R - I + 1
               BUCK(I) = TEMP1/(1 - FRAC))*(2*TEMP2-1-((1+3
     *                   *TEMP2*(TEMP2 - 1))/
     *                   (3.*(1 - FRAC)*R)))
            ENDIF
 2112    CONTINUE
         GOTO 2114
```

Cusp Distribution

A cusp distribution has the general frequency diagram shown in Figure I-3.

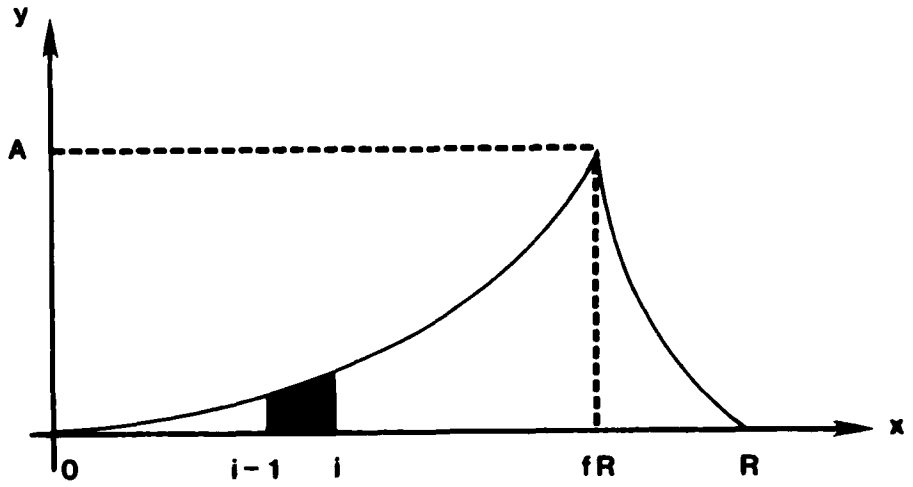


Figure I-3. Cusp Distribution Plot.

The equations describing this curve are

$$y(x) = ax^2; \quad \text{for} \quad 0 \leqq x \leqq fR \tag{I-20}$$

$$y(x) = a(R - x)^2; \quad \text{for} \quad fR \leqq x \leqq R \tag{I-21}$$

The area under the distribution curve again corresponds to the total number of points. Therefore,

$$N^2 = AR/3 \tag{I-22}$$

or

$$A = 3N^2/R \tag{I-23}$$

Letting NC(i) be the number of points in the ith bucket on the interval 0 to fR (i.e., shaded area in Figure I-3), yields

$$NC(i) = \int_{i-1}^{i} ax^2 \, dx \tag{I-24}$$

$$NC(i) = (a/3)[i^3 - (i - 1)^3] \tag{I-25}$$

Integrating Equation (I-20) yields

$$A = a(fR)^2 \qquad \text{(I-26)}$$

and from Equations (I-23) and (I-26)

$$a = 3N^2/(f^2R^3) \qquad \text{(I-27)}$$

Therefore

$$NC(i) = N^2(3i^2 - 3i + 1)/(f^2R^3) \qquad \text{(I-28)}$$

Shifting the origin to x = R,

$$NC(i) = N^2(3i'^2 - 3i' + 1)/[(1 - f)^2R^3] \qquad \text{(I-29)}$$

where $i' = R - i + 1$ in the interval $fR + 1 \leqq i \leqq R$

The **FORTRAN** code that computes the bucket sizes for the cusp distribution follows:

```
C     FRAC = f
C     IFRAC = fR
C     BUCK(I) = NT(i)
C
C        CUSP
C
      2103        TEMP1 = (1,*N*N)/R/R/R
                  DO 2113 I = 1,R
                     IF(I.LE.IFRAC)THEN
                        BUCK(I) = (1 + 3.*I*(I - 1))*TEMP1/(FRAC*FRAC)
                     ELSE
                        TEMP2 = R - I + 1
                        BUCK(I) = (1 + 3*TEMP2*(TEMP2-1))
           *            *TEMP1/((1-FRAC)*(1 - FRAC))
                     ENDIF
      2113        CONTINUE
```

APPENDIX J

LOWER BOUND

PROBLEM: Find the sequence of points on a given line which subtend a constant angle from a point not on the line. The significance of this problem is that it is the REAL SCAN problem for computing the lower bound points. The lower bound points are the intercept of the lower bound line, and the scan lines, which subtend a constant angle from the nadir.

GIVEN:

(1) ANG - The signed angle between the points on Line 1 of Figure J-1.
(2) (XS,YS) - The first point on Line 1 of Figure J-1.
(3) (XE,YE) - he last point on Line 1 of Figure J-1.
(4) (X,Y) - The last computed intercept.

Figure J-1. Depicts The Problem Statement.

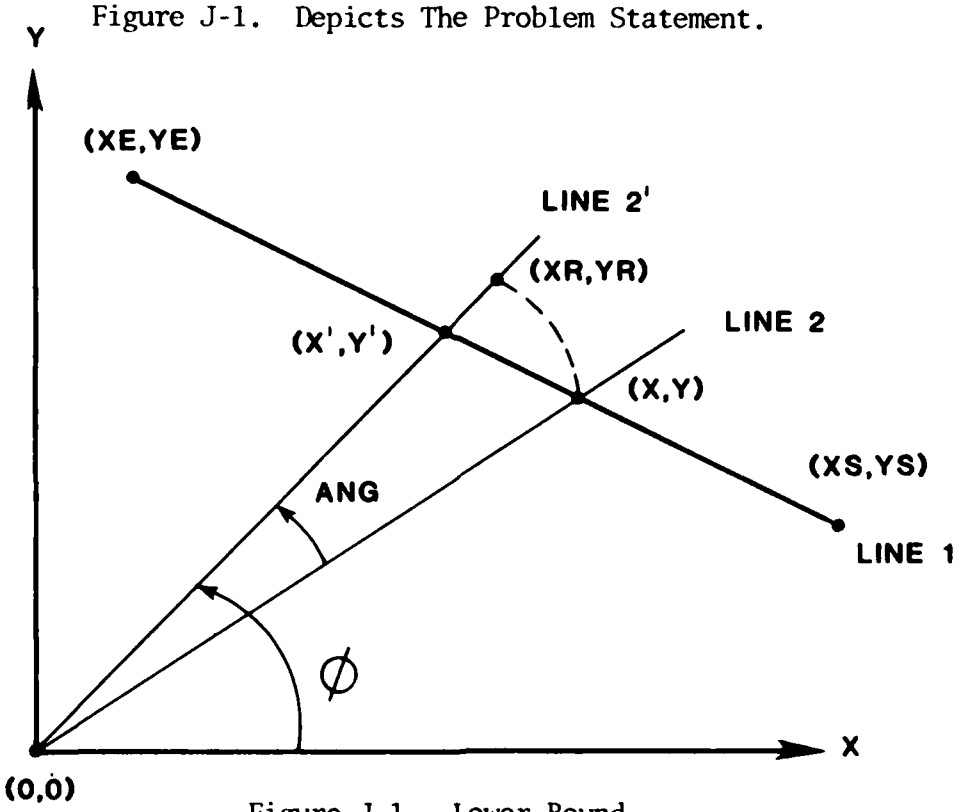


Figure J-1. Lower Bound.

Thus, the problem is to find the next intercept (X',Y').

SOLUTION:

Two solutions have been generated. SOLUTION A has been implemented in REAL SCAN, and SOLUTION B has not been implemented. Both solutions solve for the next point (X',Y'), which is the intercept of Line 2' and Line 1 of Figure J-1. This Appendix compares the two solutions for accuracy and simplicity.

SOLUTION A:

Line 1 of Figure J-1 obeys the following equation

$$(X' - X)/K1 = (Y' - Y)/K2 = T \qquad \text{(J-1)}$$

where

$$K1 = XE - XS$$
$$K2 = YE - YS$$

The equations for the particular point (X',Y') on Line 1 is

$$X' = X + K1*T \qquad \text{(J-2)}$$
$$Y' = Y + K2*T \qquad \text{(J-3)}$$

The derivative of $\tan\Phi$ with respect to t is

$$d/dt(\tan\Phi) = (d\Phi/dt)/(\cos\Phi)^2$$

For small angular changes, since t = 0 for the point (X,Y)

$$d\Phi/dt = ANG/T$$

$$d/dt(\tan\Phi) = (X'^2 + Y'^2)(ANG/T)/X'^2$$
$$= ((X+K1*t)^2 + (Y+K2*t)^2)(ANG/T)/X'^2 \qquad \text{(J-4)}$$

$\tan\Phi$ can also be defined as

$$\tan\Phi = Y'/X'$$

Therefore, another equation for $d/dt(\tan\Phi)$ is

$$d/dt(\tan\Phi) = d/dt(Y'/X')$$
$$= (X'*dY'/dt - Y'*dX'/dt)/(X'*X') \qquad \text{(J-5)}$$

From equations (J-2) and (J-3)

$$dX'/dt = K1$$
$$dY'/dt = K2$$

Therefore, Equation (J-5) becomes

$$d/dt(\tan\Phi) = (K2*X' - K1*Y')/(X'*X'). \quad \text{(J-6)}$$

If we set Equation (J-4) equal to Equation (J-6), and evaluate t at T/2 then ANG((X + K1*T/2)**2 + (Y + K2*T/2)**2) = T*(K2*X - K1*Y). (J-7)

Dropping the T^2 terms and solving for T yields

$$T = ANG*(X*X + Y*Y)/((K2*X - K1*Y) - ANG*(K1*X + K2*Y)) \quad \text{(J-8)}$$

Where (K2*X - K1*Y) is a constant K for any point (X,Y) on Line 1 of Figure (J-1), throughout the iteration. Therefore T is of the form

$$T = M/(K - B) \text{ where } B = ANG(K1*X + K2*Y).$$

If K>>B then $1/(K - B) \simeq (1/K)*(1 + B/K)$

Therefore, T is approximately

$$T \simeq (ANG/K)*(X*X + Y*Y)*(1 + (ANG/K))\ (K1*X1 + K2*Y1) \quad \text{(J-9)}$$

We can now solve for all the points along Line 1 iteratively by solving for T and knowing that

$$X' \simeq X + K1*T \quad \text{(J-10)}$$
$$Y' \simeq Y + K2*T \quad \text{(J-11)}$$

Then we let X = X' and Y = Y' and solve for a new X' and Y'. This solution involves 9 multiplies and 5 adds to compute each new lower bound point (X',Y'). But if T is computed as shown in Equation (J-12), and the point (X',Y') is computed by Equations (J-13) and (J-14) then only 8 multiplies are required.

$$T \simeq KK*(X*X + Y*Y)*\ (1 + KK*(X1 + (K2/K1)*Y1)) \quad \text{(J-12)}$$

Where KK = K1*ANG/K

$$X' \simeq X + T \quad \text{(J-13)}$$

$$Y' \simeq Y + (K2/K1)*T \quad \text{(J-14)}$$

But if K1 is zero then Equation (J-14) is unbounded. Therefore, Equations (J-10) and (J-11) will be used.

SOLUTION B:

This next solution solves for the new lower bound point by finding

the intercept of Line 1 and Line 2' of Figure (J-1), which is

$$(X' - XS) / (XE - XS) = (Y' - YS) / (YE - YS) \quad \text{(J-15)}$$

Let
$$K1 = (XE - XS) \quad \text{(J-16)}$$
$$K2 = (YE - YS) \quad \text{(J-17)}$$
$$K = XS*K2 - YS*K1 = X'*K2 - Y'*K1 \quad \text{(J-18)}$$

Line 2' can be evaluated by rotating the known Line 2 by ANG degrees.

$$\begin{bmatrix} XR \\ YR \end{bmatrix} = \begin{bmatrix} \cos(ANG) & -\sin(ANG) \\ \sin(ANG) & \cos(ANG) \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix}$$

But since ANG is small, the following approximation is valid

$$\cos(ANG) \approx 1$$
$$\sin(ANG) \approx ANG$$

Therefore, the result of the rotation becomes

$$XR = X - ANG*Y \quad \text{(J-19)}$$
$$YR = ANG*X + Y \quad \text{(J-20)}$$

And since Line 2' intercepts the origin

$$X'/XR = Y'/YR$$

or

$$X' = (Y'/YR)*XR \quad \text{(J-21)}$$

Substituting Equation (J-21) into Equation (J-18) yields

$$Y'*(K2*XR/YR - K1) = K$$

or

$$Y' = K/(XR*K2 - YR*K1)*YR \quad \text{(J-22)}$$

Substituting Equation (J-22) into Equation (J-21) yields

$$X' = K/(XR*K2 - YR*K1)*XR \quad \text{(J-23)}$$

The term K/(XR*K2 - YR*K1) can be simplified to yield an approximate solution of (X',Y') from (X,Y) without any divides.

$$TERM = K/(XR*K2 - YR*K1) \quad \text{(J-24)}$$

Substituting Equations (J-19) and (J-20) into (J-24) yields

$$TERM = K/((X*K2 - Y*K1) - ANG*(Y*K2 + X*K1)) \quad \text{(J-25)}$$

But (X*K2 - Y*K1) is the constant K of equation (J-18) therefore, TERM has the form of equation (J-26)

$$TERM = K/(K - B) \qquad \text{(J-26)}$$

where B = ANG*(Y*K2 + X*K1).

If K>> B then K/(K - B) ≈ 1 + B/K.

Thus TERM has been approximated and there are no divides since ANG/K is a multiplication factor. Therefore the final solution for (X',Y') is found by substituting equations (J-19) and (J-20) into equations (J-22) and (J-23) and using the above approximation for TERM.

$$X' = (X - ANG*Y)* 1 + (ANG/K)*(Y*K2 + X*K1) \qquad \text{(J-27)}$$
$$Y' = (ANG*X + Y)* 1 + (ANG/K)*(Y*K2 + X*K1) \qquad \text{(J-28)}$$

This solution requires 7 multiples and 4 adds while, SOLUTION A required 8 multiples and 5 adds. But SOLUTION A is potentially more accurate.

COMPARISON OF SOLUTION A AND SOLUTION B:

A third solution is to simplify SOLUTION A. This solution neglects the ANG/K squared terms of SOLUTION A and is given in equations (J-29) through (J-31) below

$$T = (ANG/K)*(X*X + Y*Y) \qquad \text{(J-29)}$$
$$X' = X + K1*T \qquad \text{(K-30)}$$
$$Y' = Y + K2*T \qquad \text{(J-31)}$$

This simplification reduces SOLUTION A to 5 multiples and 3 adds, and it is still more accurate than SOLUTION B! Therefore equations (J-29) through (J-31) is the best solution.

One way to verify that SOLUTION A and SOLUTION B are approximately the same is to verify that the terms in both solutions which involve ANG/K are equivalent.

The ANG/K term for Y' in SOLUTION A is K2*(X*X + Y*Y), (J-32)

and the ANG/K term for Y' in SOLUTION B is Y*(Y*K2 + X*K1) + X*K. (J-33)

Comparing (J-32) and (J-33) yields

$$K2*X*X + K2*Y*Y :: K2*Y*Y + K1*X*Y + X(X*K2 - Y*K1)$$

or

$$K2*X*X + K2*Y*Y :: K2*X*X + K2*Y*Y$$

Therefore the two solutions are the same in the ANG/K terms. Both SOLUTION A and SOLUTION B use the approximation $K \approx ANG*(K1*X + K2*Y)$.

The magnitude of ANG*(K1*X + K2*Y) is greatest at the end points, (i.e. when (X,Y) is equal to (XE,YE) or (XS,YS)). Therefore, a test of K against ANG*(XE*K1 + YE*K2) and ANG*(XS*K1 + YS*K2) is sufficient to validate the complete iteration for both solutions.

In REAL SCAN, the constant K of Equation (J-11) is zero when the nadir falls on the lower bound line, (i.e. Line 1 if Figure (J-1)). This does not cause a problem since, when the nadir is on the lower bound line, the lower bound is always the nadir and no computation is needed. A worst case is when the nadir is very close to the lower bound line. This situation causes K to become very small. Since all points are integer in REAL SCAN, the closest the nadir can be to the lower bound line is 1. Also, since REAL SCAN uses hierarchical levels, the length of the lower bound line is always less than or equal to 512 units.

Note that the equations derived in this Appendix require that ANG be signed. Positive angles are defined to be counter-clockwise rotations about the Z axis.

APPENDIX K

PROJECTION PROCESSOR

This Appendix describes the mathematics for PGPROJ.FOR, the FORTRAN routine which computes the screen coordinates for a visible point given in WORLD coordinates.

Problem Description

Definition of Symbols:

(X,Y,Z)	WORLD coordinate axes
(X´,Y´,Z´)	EYE CENTRIC WORLD coordinate axes
(U,V,W)	EYE coordinate axes
(XS,YS)	screen coordinate axes
(XP,YP,ZP)	visible point in WORLD coordinates
(XP´,YP´,ZP´)	visible point in EYE CENTRIC WORLD coordinates
(UP,VP,WP)	visible point in EYE coordinates
(XSP,YSP)	visible point in screen coordinates
(XE,YE,ZE)	eye location in WORLD coordinates
(US,VS,WS)	screen center in EYE coordinates
(NX,NY)	the number of pixels along both screen axes
(LX,LY)	the length of the screen along both screen axes
R	The rotation matrix due to the equations of motion, which defines the present eye orientation relative to the initial eye orientation.
RT	The transpose of R scaled to integer.

Figure K-1 depicts the initial system, (i.e., at time T = 0), with the EYE coordinate axes (U,V,W) aligned with the WORLD coordinate axes (X,Y,Z). The eye is at the world origin and the screen is at any arbitrary location (US,VS,WS) as long as VS is greater than zero. The screen coordinates (XS,YS) align with the (U, -W) axes of the EYE coordinate system.

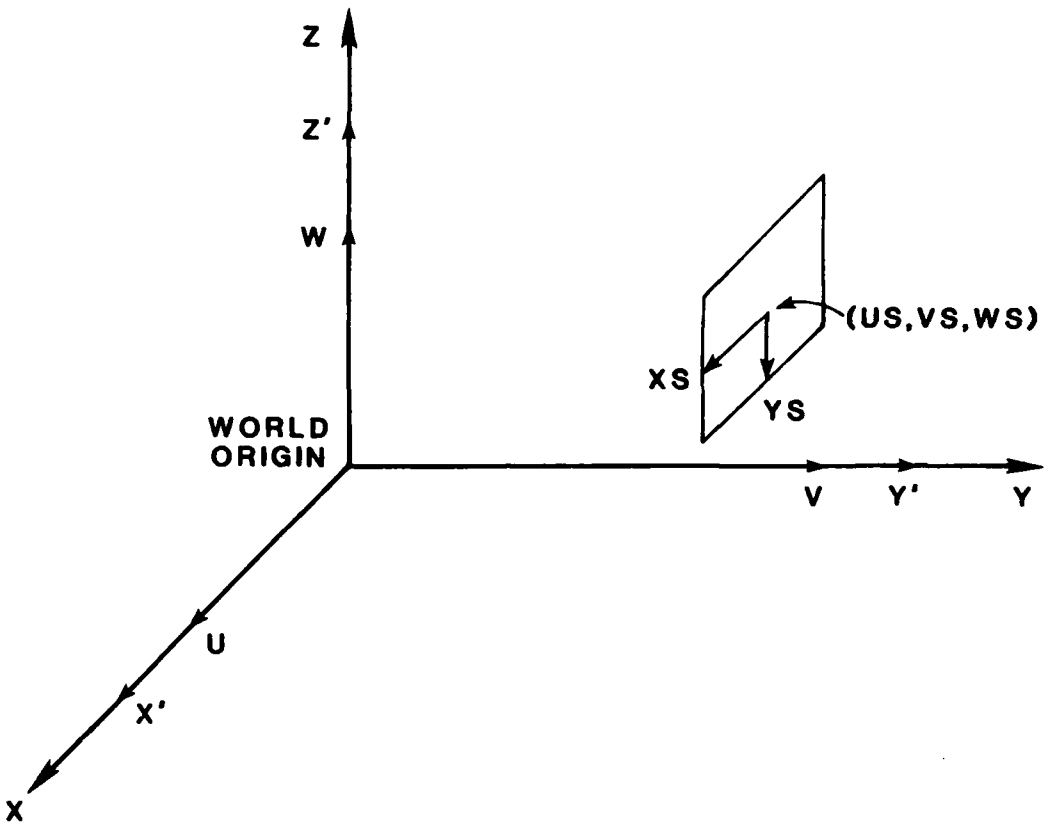


Figure K-1. Initial World, Eye, and Screen System at Time T = 0.

Figure K-2 depicts the system after the eye has traversed through space. The eye coordinate system is related to the EYE CENTRIC WORLD coordinate system by the rotation matrix R. The EYE CENTRIC WORLD coordinate system has the same orientation as the WORLD coordinate system but it is translated to the eye.

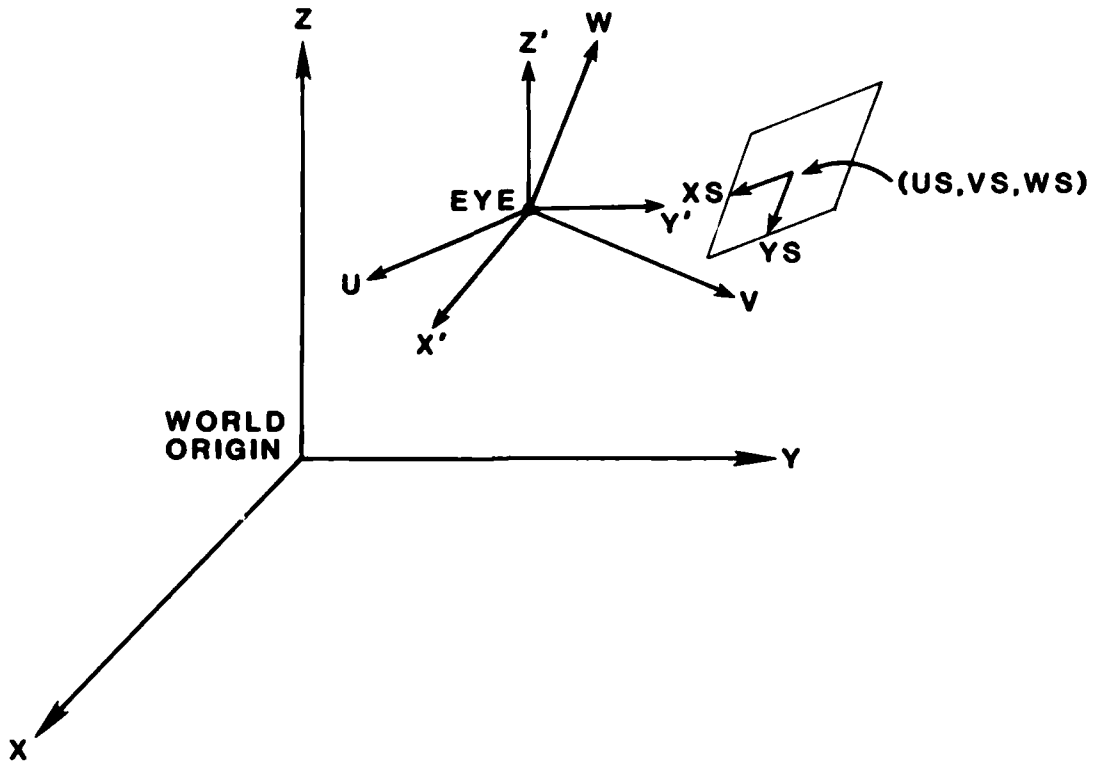


Figure K-2. World, Eye, and Screen System at Time T > 0.

Problem: Find the screen coordinates of all the visible world points on a **REAL SCAN** line, using integer arithmetic.

Solution: The solution can be found by partitioning the problem into several smaller problems as described below:

1. Find the screen coordinates of one visible point, assuming that the screen center is at (0,VS,0).

2. Find the screen coordinates of one visible point with the screen center at any arbitrary location (US,VS,WS).

3. Find the screen coordinates of successive visible points on the scan line.

In order to find the screen coordinates (XSP,YSP) of a visible point (XP,YP,ZP), one must: (1) find the location of the visible point in the EYE coordinate system (UP,VP,SP); (2) project that point onto the screen.

To find the location of the point in the EYE coordinate system, one may consider moving the world point in the opposite direction that the eye has moved. There are two parts to this opposite movement: (1) Translate the point from the WORLD coordinate system to the EYE CENTRIC WORLD coordinate system; (2) Rotate the point in the EYE CENTRIC WORLD coordinate system into the EYE coordinate system. The translation is found by subtracting the eye's location from the point's location as depicted in Equation (K-1).

$$\begin{bmatrix} XP' \\ YP' \\ ZP' \end{bmatrix} = \begin{bmatrix} XP - XE \\ YP - YE \\ ZP - ZE \end{bmatrix} \tag{K-1}$$

The point in the EYE coordinate system can be found by the matrix multiplication of the inverse of R and the point in the EYE CENTRIC WORLD coordinate system. Since the rotation matrix R is a matrix of direction cosines, the inverse of R is equal to the transpose of R or RT. Thus, the point in EYE coordinates is given by Equation (K-2).

$$\begin{bmatrix} UP \\ VP \\ WP \end{bmatrix} = [RT] \begin{bmatrix} XP' \\ YP' \\ ZP' \end{bmatrix} \tag{K-2}$$

Note that the projection algorithm uses integer arithmetic. Therefore, RT must be scaled. If RT is scaled by a power of two, the scaling will correspond to a logical shift of the bit positions. Equation (K-3) depicts the scaling of RT.

$$RT = 2^{n} * R^{-1} \tag{K-3}$$

The value of n is directly related to the accuracy of the projection algorithm.

The method of similar triangles is used to project the point in EYE coordinates onto the screen. This method gives a perspective view of the point. Figure K-3 illustrates this projection problem.

Similar triangles yield

$$XSP = VS*UP/VP \tag{K-4}$$

$$YSP = -VS*WP/VP \tag{K-5}$$

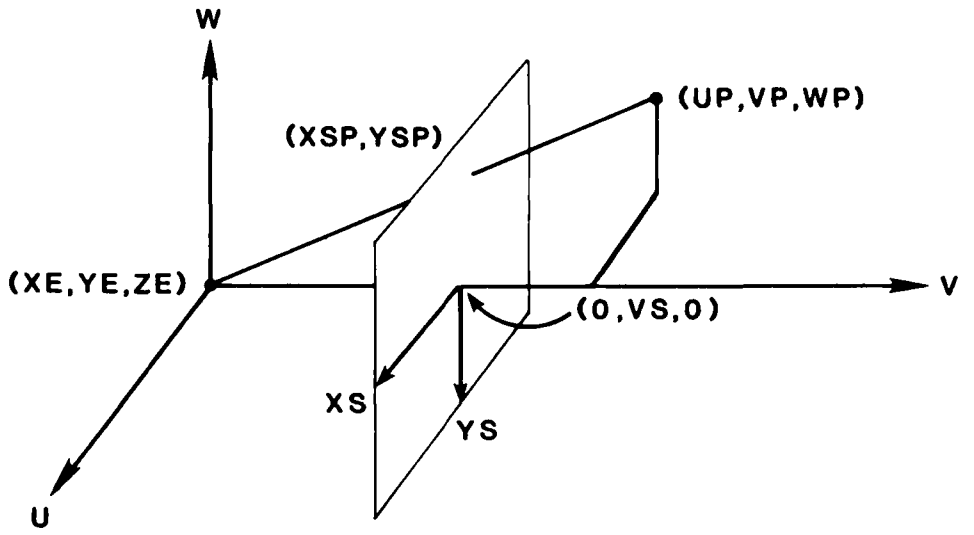


Figure K-3. Projection.

The screen coordinates of Equations (K-4) and (K-5) are in length units. To convert the screen coordinates to pixel units, the length screen coordinates must be multiplied by the number of pixels per unit length. If the screen has square pixels defined by Equation (K-6) then, the screen coordinates are given by Equations (K-7) and (K-8).

$$NPL = NX/LX = NY/LY \quad \text{(K-6)}$$

$$XSP = K*UP/VP \quad \text{(K-7)}$$

$$YSP = -K*WP/VP \quad \text{(K-8)}$$

where

$$K = NPL*VS$$

If the screen center is at an arbitrary location (US,VS,WS), then the screen coordinates (XSP,YSP) must be translated relative to the screen center. Equations (K-9) and (K-10) illustrate this translation

$$XSP = K*UP/VP - US*NPL \quad \text{(K-9)}$$

$$YSP = -K*WP/VP - WS*NPL \quad \text{(K-10)}$$

But since US*NPL and WS*NPL are constants, Equations (K-9) and (K-10) may be written

$$XSP = K*UP/VP - KU \qquad (K\text{-}11)$$

$$YSP = -K*WP/VP - KW \qquad (K\text{-}12)$$

Since all the math is to be integer, the screen coordinates of Equations (K-11) and (K-12) will be truncated. Thus, two integer error terms must be created to manage the fractional portion of the screen coordinates. The real screen coordinates (RXSP,RYSP) can be defined to be the sum of an integer part and a fractional part. Equation (K-13) defines the real screen coordinate along the XS axis, which yields

$$RXSP = XSP + FRACX = K*UP/VP - KU \qquad (K\text{-}13)$$

Then a scaled integer representation of RXSP is given by Equation (K-14)

$$VP*RXSP = XSP*VP + FRACX*VP = K*UP - KU*VP \qquad (K\text{-}14)$$

The scaled integer error term along the XS axis is then defined as FRACX*VP, hence

$$\begin{aligned} ERRORX &= K*UP - (XSP + KU)*VP \\ ERROSX &= K*UP - JIX*VP \end{aligned} \qquad (K\text{-}15)$$

where JIX = XS + KU.

Since XSP is a truncated integer, a test of FRACX against + 1/2 will determine if XSP should actually be XSP + 1. A test of ERRORX against + VP/2 is equivalent to testing FRACX against + 1/2. A similar error term along the US axis is defined by Equation (K-16). Thus, with the use of these two error terms, the actual screen coordinate can be found using only integer arithmetic. The corresponding YS axis terms are

$$ERRORY = -K*WP - JIY*VP \qquad (K\text{-}16)$$

where JIY = YS - KW

$$\begin{bmatrix} XPINC \\ YPINC \\ ZPINC \end{bmatrix} = \begin{bmatrix} XP' - XP \\ YP' - YP \\ ZP' - ZP \end{bmatrix} \qquad (K\text{-}17)$$

The incremental distance between visible points (XPINC,YPINC, ZPINC) is then rotated into the EYE coordinate system by the matrix multiplication of RT, to yield the incremental distance in the EYE coordinate system (UPINC,VPINC,WPINC), as

$$\begin{bmatrix} UPINC \\ VPINC \\ WPINC \end{bmatrix} = [RT] \begin{bmatrix} XPINC \\ YPINC \\ ZPINC \end{bmatrix} \qquad \text{(K-18)}$$

The incremental distance between visible points in the eye coordinate system is then used to update the integer error terms. The new point in the EYE coordinate system (UP´,VP´,WP´) is defined in Equation (K-19)

$$\begin{bmatrix} UP' \\ VP' \\ WP' \end{bmatrix} = \begin{bmatrix} UP + UPINC \\ VP + VPINC \\ WP + WPINC \end{bmatrix} \qquad \text{(K-19)}$$

Substituting Equation (K-19) into Equation (K-15) yields the error term for the new point along the XS axis

$$ERRORX' = K*(UP + UPINC) - JIX*(VP + VPINC)$$

or

$$ERRORX' = (K*UP - JIX*VP) - (K*UPINC - JIX*VPINC)$$

But (K*UP - JIX*VP) is ERRORX. Therefore ERRORX´ is equal to the old error term plus an incremental error term, which accounts for the incremental distance between visible points

$$ERRORX' = ERRORX + K*UPINC - JIX*VPINC \qquad \text{(K-20)}$$

Similarly substituting Equation (K-19) into (K-16) yields ERRORY´, the error term for the new visible point along the YS axis is

$$ERRORY' = ERRORY - K*WPINC - JIY*VPINC \qquad \text{(K-21)}$$

Thus the computation for the new error terms is an iterative operation. The test for pixel boundary crossing is if the absolute value of FRACX or FRACY is greater than 1/2 (See Figure 15). Thus the integer test for the pixel boundary crossing is if the absolute value of ERRORX´ or ERRORY´ is greater than VP/2. If the new point does map to an adjacent pixel, then the corresponding error term must be decremented or incremented by VP, depending on the change in screen coordinate, (i.e., if XSP ← (XSP + 1) then ERRORX ← (ERRORX - VP), or if XSP ← (XSP - 1) then ERRORX ← (ERRORX + VP)).

The direction in which the scan line moves across the screen (INCXS, INCYS) is used to determine which pixel boundary crossing test applies. The value of INCXS or INCYS is (-1, 0, or +1). If INCXS is equal to 1 then the pixel boundary crossing test for the XS axis is if ERRORX is greater than or equal to VP/2. Likewise, if INCXS is equal to minus 1 then the pixel boundary crossing test on the XS axis is if ERRORX is less than minus VP/2.

One can identify the direction of the scan line on the screen (INCXS, INCYS) by projecting the nadir and the horizon on the screen. If the nadir and horizon in EYE coordinates are (UNADIR,VNADIR,WNADIR) and (UHORIZ,VHORIX,WHORIX), then the screen coordinates for the nadir and the horizon are given by Equations (K-22) through (K-25)

$$XSN = K*UNADIR/VNADIR - IU \quad \text{(K-22)}$$

$$YSN = -K*WNADIR/VNADIR - KW \quad \text{(K-23)}$$

$$XSH = K*UHORIZ/VHORIZ \; KU \quad \text{(K-24)}$$

$$YSH = -K*WHORIZ/VHORIZ - KW \quad \text{(K-25)}$$

The slope of the scan line projected on the display screen is

$$SLOPE = (YSH - YSN)/(XSH - XSN) \quad \text{(K-26)}$$

If two terms DELX and DELY are created as

$$DELX = UHORIZ*VNADIR - UNADIR*VHORIZ \quad \text{(K-27)}$$

$$DELY = WHORIZ*VNADIR - WNADIR*VHORIZ \quad \text{(K-28)}$$

then the slope is

$$SLOPE = DELY/DELX \quad \text{(K-29)}$$

If the magnitude of the inverse slope on the slope is less than the slope of the line which defines a half pixel change across the screen, then either INCXS or INCYS should be set to zero respectively. Otherwise INCXS and INCYS should be equal to the sign of DELX and DELY respectively. The **FORTRAN** test for determining if INCXS should be zero is

```
IF(ABS(DELX/DELY) .LE. 1/(NY*2)) INCXS = 0          (K-30)
```

or more efficiently

```
IF(ABS(2*NY*DELX - DELY) .LE. 0) INCXS = 0          (K-31)
```

A similar **FORTRAN** equation for INCYS is

```
IF(ABS(2*NX*DELY - DELX) .LE. 0) INCYS = 0          (K-32)
```

If INCXS is not zero then INCXS = SIGN(1,DELX)*SGN,

IF INCYS is not zero then INCYS = SIGN(1.DELY)*SGN

where SGN = SIGN(1,(VNADIR*VHORIZ)).

APPENDIX L

CPU TIME COMPARISON

This Appendix compares the two sets of software (PG*.FOR) and (PZ*.FOR) for the CPU time required to compute an identical scene. A scene was created with the following input data using both sets of the **REAL SCAN** software routines.

```
EYE LOCATION = 0,0,2000
PITCH        = 5 DEGREES
BANK         = 0 DEGREES
HEADING      = 0 DEGREES
LX           = 1
LY           = 1
NPL          = 512
HORIZON LIM  = 45000
```

The two pictures that were created by the two sets of routines are identical except for the sky shading, and the CPU time required to create the pictures was significantly different. The two pictures are given in Figures L-1 and L-2.

The CPU time for Figure L-1 was 1 hour and 54 minutes while the CPU time for Figure L-2 was 1 hour. The reason for the difference in CPU times is attributed to the fact that in the PZ*.FOR routines the scan lines don't start at the nadir like they do in the PG*.FOR routines.

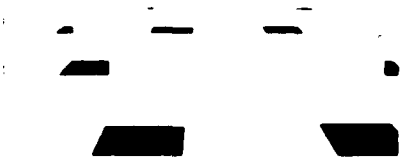


Figure L-1. Scene Using PG*.FOR Routines.



Figure L-2. Scene Using PZ*.FOR For Routines.

APPENDIX M

EFFECT OF ANGFACT ON CPU TIME AND ALIASING

This Appendix compares six pictures for their clarity. All six pictures are of the same scene but each picture has a different angle factor for the angle between scan lines. The sequence of angle factors for the six pictures is 5,4,3,2,1,.5. The input data for these scenes are:

```
EYE LOCATION = 0,0,2000
PITCH        = 15
BANK         = 0
HEADING      = 0
LX           = 1
LY           = 1
NPL          = 512
```

These six pictures are given in Figures M-1 through M-6. Each picture is accompanied by another picture which is the result of filtering any missed pixels. The missed pixels are filled by using the FILT2 routine.

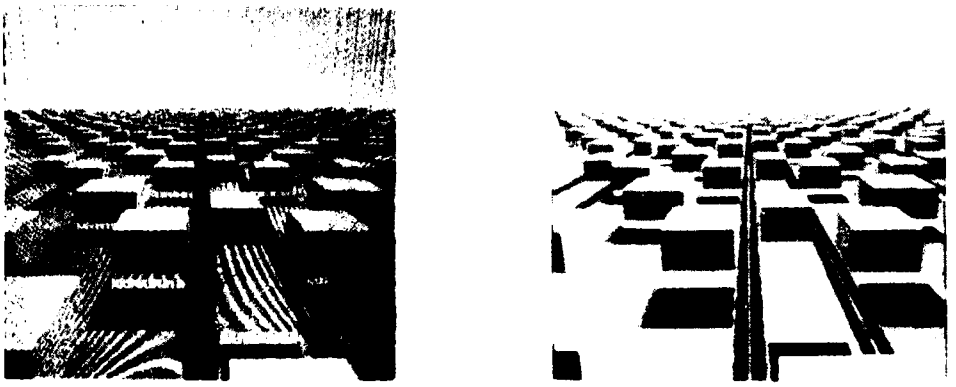


Figure M-1. ANGFACT = 5 CPU TIME = 14 MIN 241 SCAN LINES.

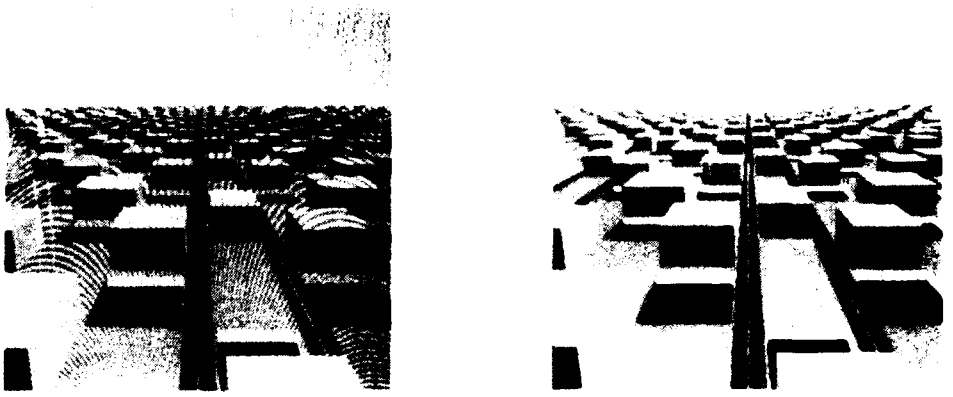


Figure M-2. ANGFACT = 4 CPU TIME = 18 MIN 301 SCAN LINES.

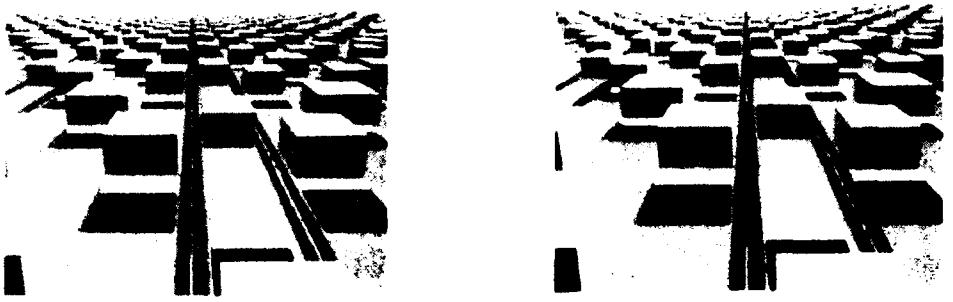


Figure M-3. ANGFACT = 3 CPU TIME = 22 MIN 401 SCAN LINES.

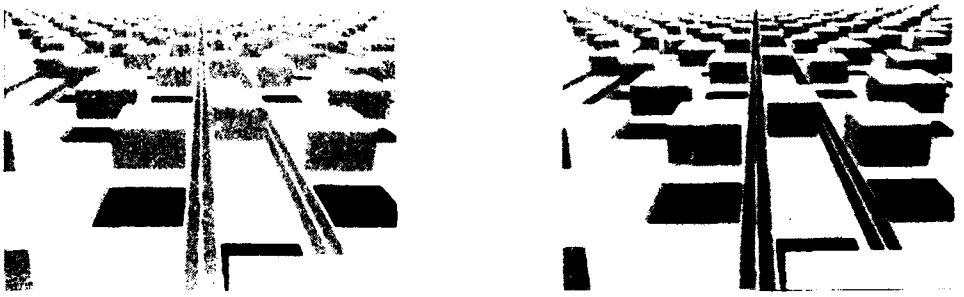


Figure M-4. ANGFACT = 2 CPU TIME = 34 MIN 601 SCAN LINES.

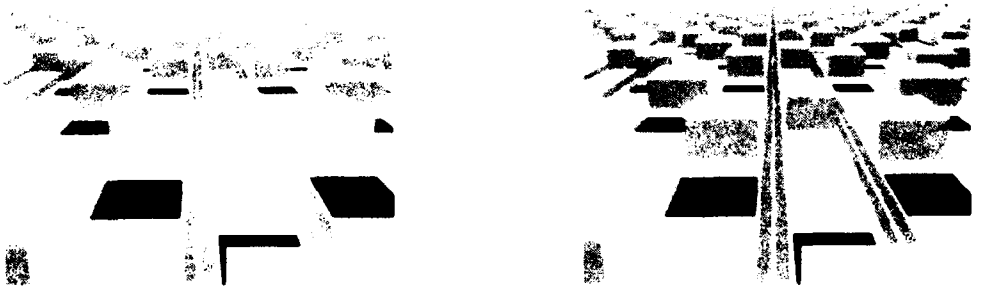


Figure M-5. ANGFACT = 1 CPU TIME = 68 MIN 1201 SCAN LINES.



Figure M-6. ANGFACT = .5 CPU TIME = 135 MIN 2401 SCAN LINES.

The above pictures demonstrate how effective FILT2 is when missed pixels are encountered. As can be seen there is a trade off for CPU time and scene clarity. Figure M-1 was computed in just 14 minutes but the scene has a large amount of aliasing problems, while Figure M-6 was computed in 135 minutes and has almost no aliasing problems.

APPENDIX N

OVERHAUSER-COONS INTERPOLATION

This Appendix develops the Overhauser-Coons interpolation routines and the elevation error evaluation routines.

The elevation error evaluation routine for O-C is listed in Appendix HB and the elevation error evaluation routine for the simpler straight line formula is listed in Appendix HC. Results of both elevation evaluations are presented at the end of this Appendix.

Introduction

It appears that most terrain can be modeled with a data base which describes a single value surface. For rectangular coordinates with axes X and Y, every point on the single value surface has a corresponding height, Z(X,Y). The data base for a complex scene is quite large. One of the problems is to create a means of generating a complex function, Z(X,Y), using a minimal amount of data. The Overhauser-Coons (O-C) Bicubic Patch Function was investigated as a possible solution. The O-C Function interpolates the height of a surface at some point (X,Y) for a given region knowing adjacent height values P1 through P12 as shown in Figure N-1a. The shaded area, in Figure N-1a, is the interpolation region or "patch" where the values of Z(X,Y) are to be computed. A large area is divided into many patches where interpolation of each patch requires adjacent height values. In Figure N-1b, the surface is divided into 16 areas where the four shaded areas are the Overhauser-Coon's Patch regions. The asterisks denote the sample points that are needed to interpolate the four shaded regions. The sample points form a grid which is the sample data array required for the interpolation of the surface. Therefore, the data base describing a surface is reduced since the data base is the sample data array rather than a height corresponding to every point (X,Y) on the surface.

The Overhauser-Coons Function

The Overhauser-Coons (O-C) Bicubic Patch Function, Z(X,Y), is

$$Z(X,Y)=C1(X)B0(Y)-(Z1)B0(X)B0(Y)+C2(Y)B0(X)-(Z2)B1(X)B0(Y)$$
$$C3(X)B1(Y)-(Z3)B0(X)B1(Y)+C4(Y)B1(X)-(Z4)B1(X)B1(Y) \quad (N-1)$$

Shown in Figure N-2 are Z1, Z2, Z3, and Z4 which are the heights at the corners of the patch corresponding to the adjacent points to the patch P8, P9, P48, and P5, respectively. The boundary lines of the patch are C1(X), C2(Y), C3(X), and C4(Y) which are defined by the

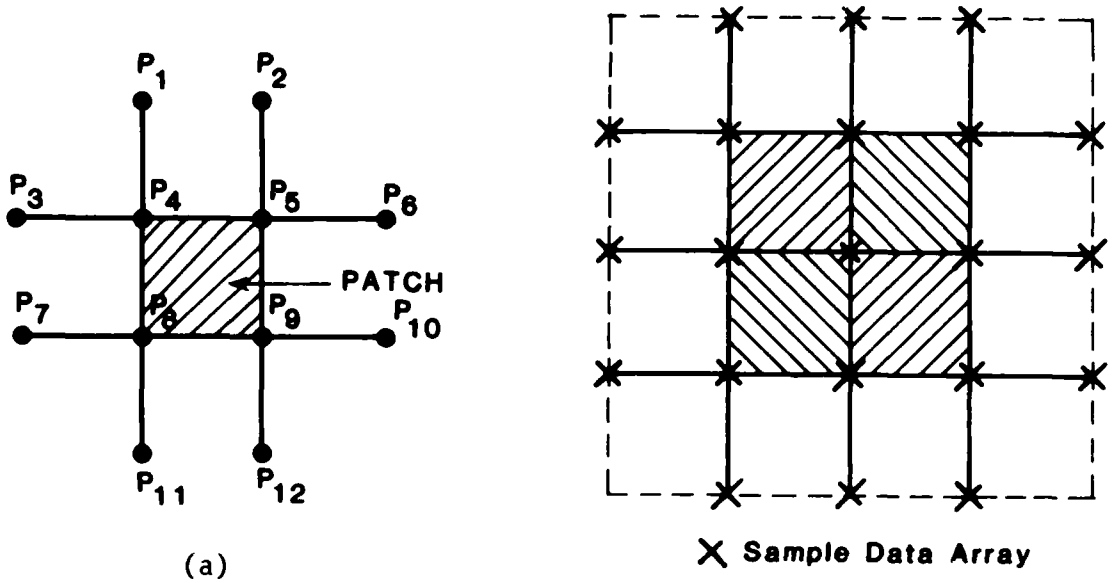


Figure N-1. Sample Data Array (a) for Overhauser-Coons Patch Region, (b) for Area Interpolated by 4 Patches with Sample Data Array in the Form of a Grid.

Overhauser Functions. The Overhauser Functions permit the interpolation of points on a line given four points on that line. Here, the line is one of the boundary lines of the patch. The four points on the line are the points in the sample data array. BO(X), BO(Y), B1(X), and B1(Y) are the Coons Blending Functions which permit interpolation between the boundary lines thereby defining the interior of the patch. The O-C Function is evaluated at any point (X,Y) within the patch as t(X) and in the Y-direction as t(Y). Therefore, "the interior of the patch is expressed in terms of the boundary curves (the Overhauser Functions), the four bounding points of the patch and two Coon's Blending Functions" (17).

The incremental distances within the patch, t(X) and t(Y), are determined for each point interpolated within the patch. Given a point (X,Y) to be interpolated within the patch, the incremental distance t(X), is the distance from the point to the boundary line C2(Y), Figure N-2. The incremental distance, t(Y), is the distance from the point to the boundary to the line C1(X). The incremental distance is then normalized by dividing by the width of the patch illustrated in Figure N-2.

---

[17]Brewer, J. and Anderson, D. "Visual Interaction with Overhauser-Coons Curves and Surfaces", Computer Graphics, Vol. 11, No. 2, pp. 133-137, Summer 1977.

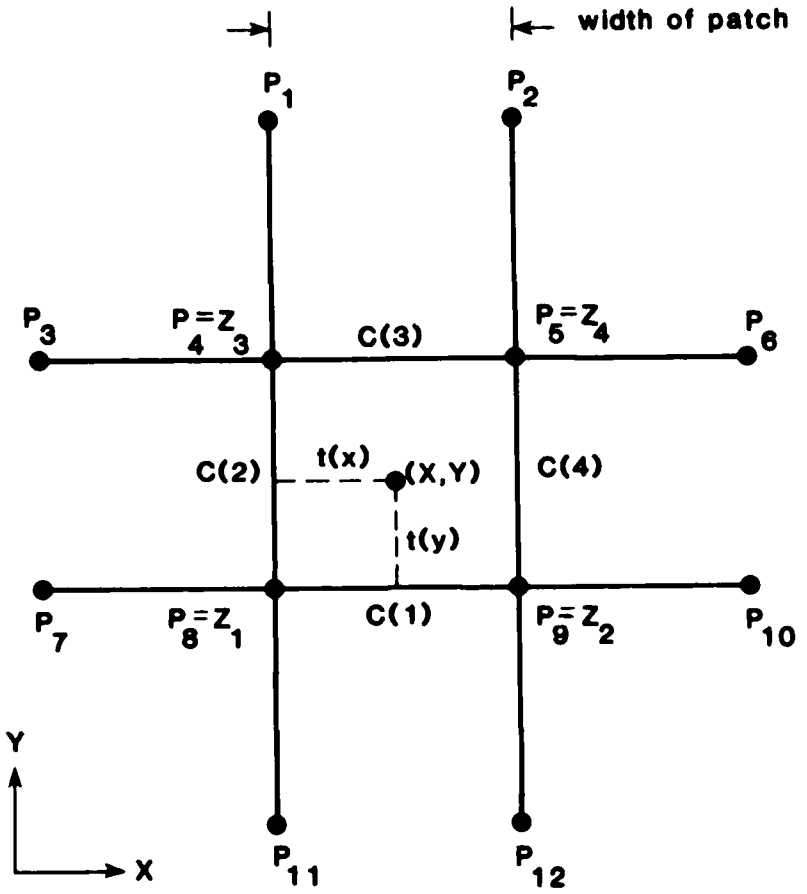


Figure N-2. Variables in Overhauser Functions.

The Overhauser Function of the boundary lines of the patch are of the form

$$Cn(X)=V(n,1)t^3(X)+V(n,2)t^2(X)+V(n,3)t(X)+V(n,4);\ n=1 \text{ or } 3 \quad (N-2)$$

$$Cn(Y)=V(n,1)t^3(Y)+V(n,2)t^2(Y)+V(n,3)t(X)+V(n,4);\ n=2 \text{ or } 4 \quad (N-3)$$

The derivation of the Overhauser Functions is provided in Reference 17. Equations N-2 and N-3 define the X- and Y-dependent boundary lines, respectively. The boundary lines are said to be X-dependent if the sample data points on the line (X,Y), have different values for X but the same value for Y. Referring to Figure N-2, the boundary lines C1(X) and C3(X) are the X-dependent boundary lines. C1(X) is dependent on the sample data points P7, P8, P9, and P10. C3(X) is dependent on the sample data points P3, P4, P5, and P6. The Y-dependent boundary lines are C2(Y) and C4(Y). C2(Y) is dependent on sample data points P1, P4, P8, and P11. C4(Y) is dependent on

points P2, P5, P9, and P12. The value of the incremental distance used in the equations for the X-dependent boundary lines is t(X) and for the Y-dependent boundary lines is t(Y). The coefficients of the Overhauser Functions of the boundary lines are the elements of the array V(4,4). The coefficients for the equations of each boundary lines are obtained by the matrix multiplication of the 4 by 4 data matrix derived in Reference 17 and the 4 by 1 matrix of the four sample data points on the boundary line.

The Coon's Blending Functions, BO(X), B1(X), BO(Y), and B1(Y), are also evaluated for each point interpolated within the patch. The Coon's Blending Functions allow for interpolation of the interior of the patch between the boundary lines. Referring to the O-C Function, Equation N-1, the Overhauser Function of a boundary line in one direction (X or Y) is multiplied by a Coon's Blending Function in the other direction. Also, the height at the corners of the patch, Z1, Z2, Z3, and Z4, are considered in the interpolation by multiplying each by two of the Coon's Functions. The Coon's Blending Functions in the X-direction are defined as

$$BO(X)=1-3t^2(X)+2t^3(X) \quad \text{and} \quad B1(X)=3t^2(X)-2t^2(X)$$

and in the Y-direction, as

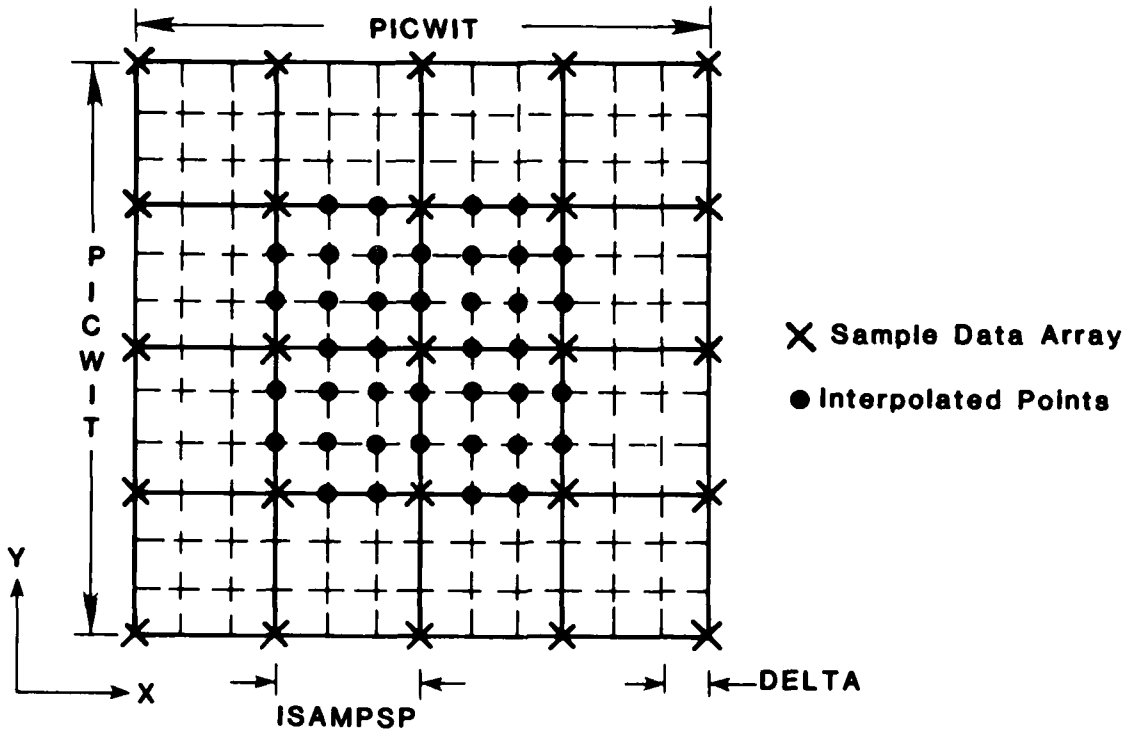
$$BO(Y)=1-3t^2(Y)+2t^3(Y) \quad \text{and} \quad B1(Y)=3t^2(Y)-2t^3(Y).$$

The Overhauser-Coons Bicubic Patch Routine

The Overhauser-Coon's Bicubic Patch Routine is software which controls the computation of the Overhauser-Coon's Function over a single value surface given the sample data array of the surface. The **FORTRAN** file for the O-C Routine, LCOVRCNS.FOR, is provided in Appendix GB.

The Overhauser-Coons Bicubic Patch routine interpolates points (X,Y) in world coordinates in a square surface area. Here, world coordinates are the actual locations (in dimensions such as feet or meters) of points on the surface with reference to some point designated as the origin (0,0). The choice of dimensions is arbitrary so long as the units are consistent for all variables throughout the routine. Figure N-3 illustrates the variables PICWIT, ISAMPSP, and DELTA. The width of the surface is referred to as the picture width (the variable PICWIT). The distance between the sample data points, the width of one patch, is the sample spacing (SAMPSPAC). However, the data base source information (i.e., the sample data array) is created on integer (X,Y) coordinates. Hence, the sample spacing, or patch width, is defined as the integer ISAMPSP where

$$SAMPSPAC-1 < ISAMPSP < SAMPSPAC.$$



SAMSPSPAC-1 < ISAMPSP < SAMPSPAC where SAMPSPAC is a real number and ISAMPSP is an integer.

Figure N-3. Illustration of Variables (PICWIT, ISAMPSP, DPTDEN, DELTA), Sample Data Array, and Interpolated Points.

Any real number can be entered for the value of SAMPSPAC but the integer value, ISAMPSP, will be used in computation of the sample data array coordinates. The number of divisions of this sample spacing by the interpolated points is the data point density (DPTDEN). The distance between the interpolated points is given the **FORTRAN** variable name DELTA which is calculated in the routine as DELTA= ISAMPSP/ DPTDEN. Also, Figure N-3 denotes the location of some sample array points (*) and their corresponding interpolated points(·).

Since adjacent points are required for each patch, an area of length PICWIT and width ISAMPSP along each edge is not interpolated. Therefore, the area of interpolation region can be calculated as $((INT(PICWIT/ISAMPSP) - 2)\ ISAMPSP)^2$, where the function INT( ) yields the greatest interger less than the argument of the function.

In a CIG System, the sample data array for the O-C routine is the actual heights of a known single value surface with coordinates X and Y where the distance between the sample data points in the X and Y directions is ISAMPSP. The sample data array is a grid of the surface, separating the surface into many O-C patches. In developing the routine, a test function for which an exact value at each point (X,Y) could be determined was used to check the accuracy of the O-C procedure. The sample data array for the test was the value of the function for each point (X,Y) at the intersection of the grid.

The sample data points are elements of the **FORTRAN** array SAMPT (IX,IY) where IX and IY are the indices of the array, as shown in Figure N-4a. Since **FORTRAN** requires the indices of an array be non-zero, positive integers, the indices, IX and IY, are counted from 1 to some maximum value. The maximum value is IEND where IEND is a **FORTRAN** variable calculated as

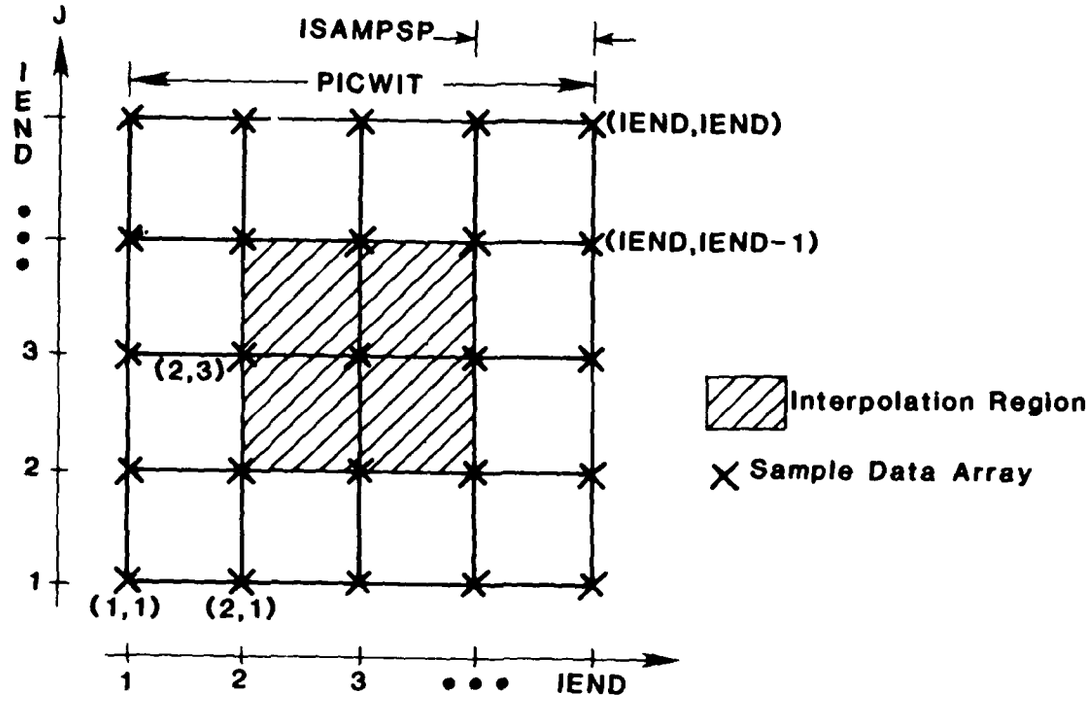
```
DPAC=1./ ISAMPSP                                    (N-4)

IEND=PICWIT*DPAC+1                                  (N-5)
```

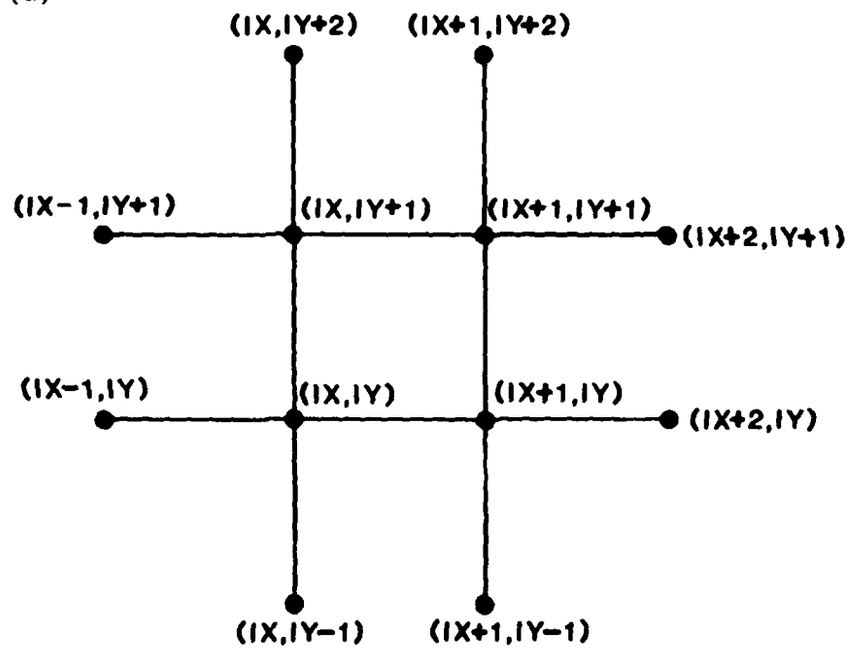
The world coordinates of a sample data point (X,Y) with indices (IX, IY) can be thought of as X=(IX-1)ISAMPSP and Y=(IY-1)ISAMPSP. Therefore, the origin has world coordinates (0,0) and indices (1,1). As a further example, the point with world coordinates of (ISAMPSP, 2(ISAMPSP)) has indices of (2,3). Each patch is referenced by the indices of the sample data point at the lower left corner of the patch, (IX,IY). Figure N-4b shows that all other indices of points adjacent to the patch are determined from the choice of (IX,IY).

Referring to Figure N-3, the order of computation of the interpolation region in the test routine begins at point (1) and increments along Y until reaching point (2). This completes one "segment". The Y increment continues until reaching (3), completing one "line". Then X is incremented to point (4) while Y returns to its initial value. The X and Y increments continue until reaching (5) completing the interpolation region or one "picture" (not the same as PICWIT). Since the Overhauser Functions of the boundary lines are unique for each patch, whenever a boundary line of a patch is crossed it is necessary to compute the Overhauser Functions for the boundary lines of the new patch. The Coon's Blending Functions and the complete Overhauser-Coon's Function, however, are evaluated at each interpolated point within the patch. It should be noted that the particular order of computation used by the test routines is completely arbitrary.

The changes in the X- and Y-coordinates occur in increments of DELTA along each axis. Again, in the O-C routine, DELTA is calculated as DELTA=ISAMPSP/DPTDEN which is the distance between the interpolated points. As shown in Figure N-5, the increments of DELTA are numbered consecutively along the axis starting from 0 at the origin. Let A and



(a)



(b)

Figure N-4. Indices (a) for Sample Data Array, (b) for Adjacent Points of Patch Region.

B, in units of DELTA, be values on the X-axis and the Y-axis which mark the beginning and end of the interpolation region along each axis. In terms of A and B, the first point interpolated, (1), in Figure N-5, has world coordinates (X,Y)=((A)DELTA,(A)DELTA). The last point interpolated, (2), has world coordinates (X,Y)=((B)DELTA,(B) DELTA). The **FORTRAN** DO loops, which perform the interpolation of the region, add DELTA as a first step prior to calculating an interpolated value. Thus, the indices of the DO loops vary from A - 1 to B - 1.

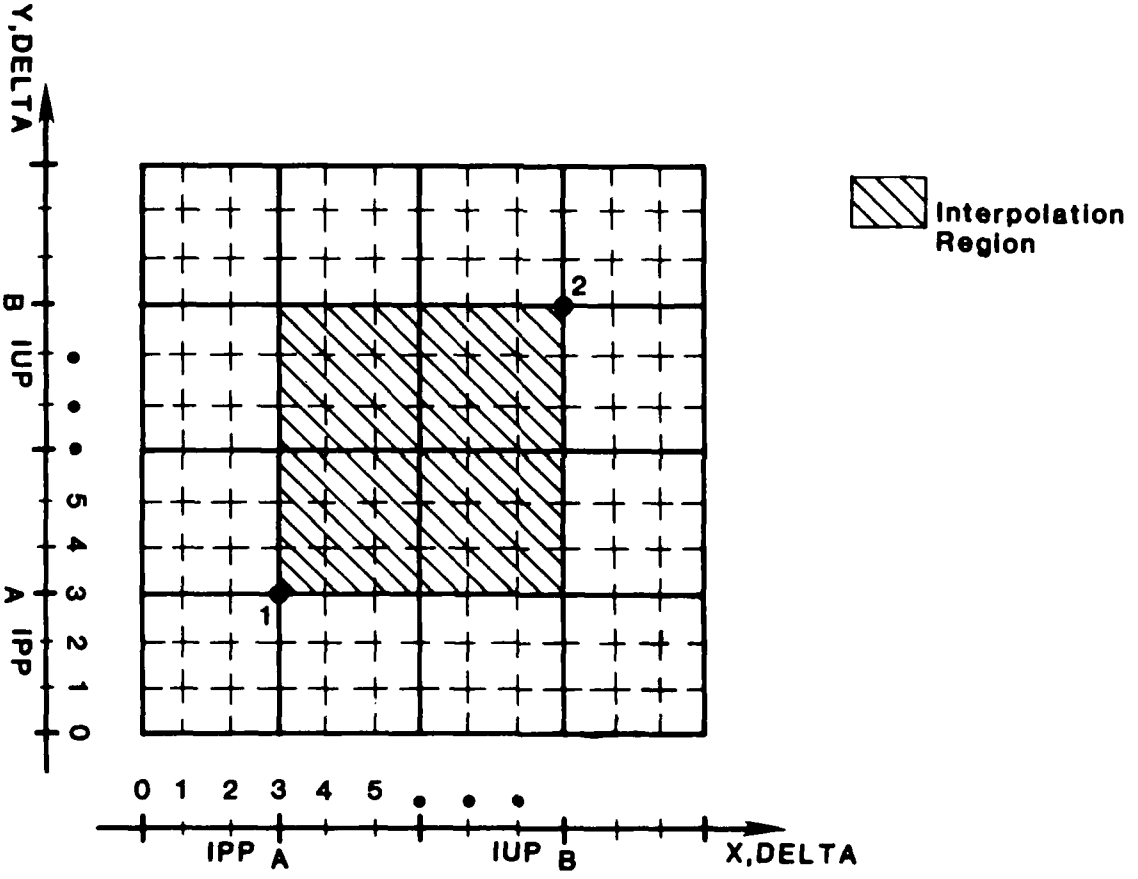


Figure N-5. Limits *IPP* and *IUP*.

The numerical value of A - 1 is given the **FORTRAN** variable name IPP and of B - 1 the name IUP. Figure N-5 illustrates IPP and IUP, which are calculated in the routine as

IPP=DPTDEN-1 (N-6)

IUP=(IEND-2)*DPTDEN-1 (N-7)

Note that in Figure N-5, the solid lines are separated by a distance ISAMPSP. The dashed lines are separated by a distance DELTA. Here,

ISAMPSP is an integer multiple of DELTA, namely 3, which is the data point density, DPTDEN. In this case, the solid and dashed lines coincide every third dashed line. If the DPTDEN is not an integer, then these lines may not necessarily coincide. Figure N-6 depicts the case where DPTDEN=1.5. Using Equations N-6 and N-7, the limits IPP and IUP would be calculated as IPP=0 and IUP=3. These limits are incorrect because the first interpolated point would then have coordinates X=Y=DELTA, which is point (1) in Figure N-6. This point cannot be interpolated by the Overhauser-Coon's Function because the required adjacent sample data points are not available for the patch which includes this point. This error occurs when DPTDEN is not an integer

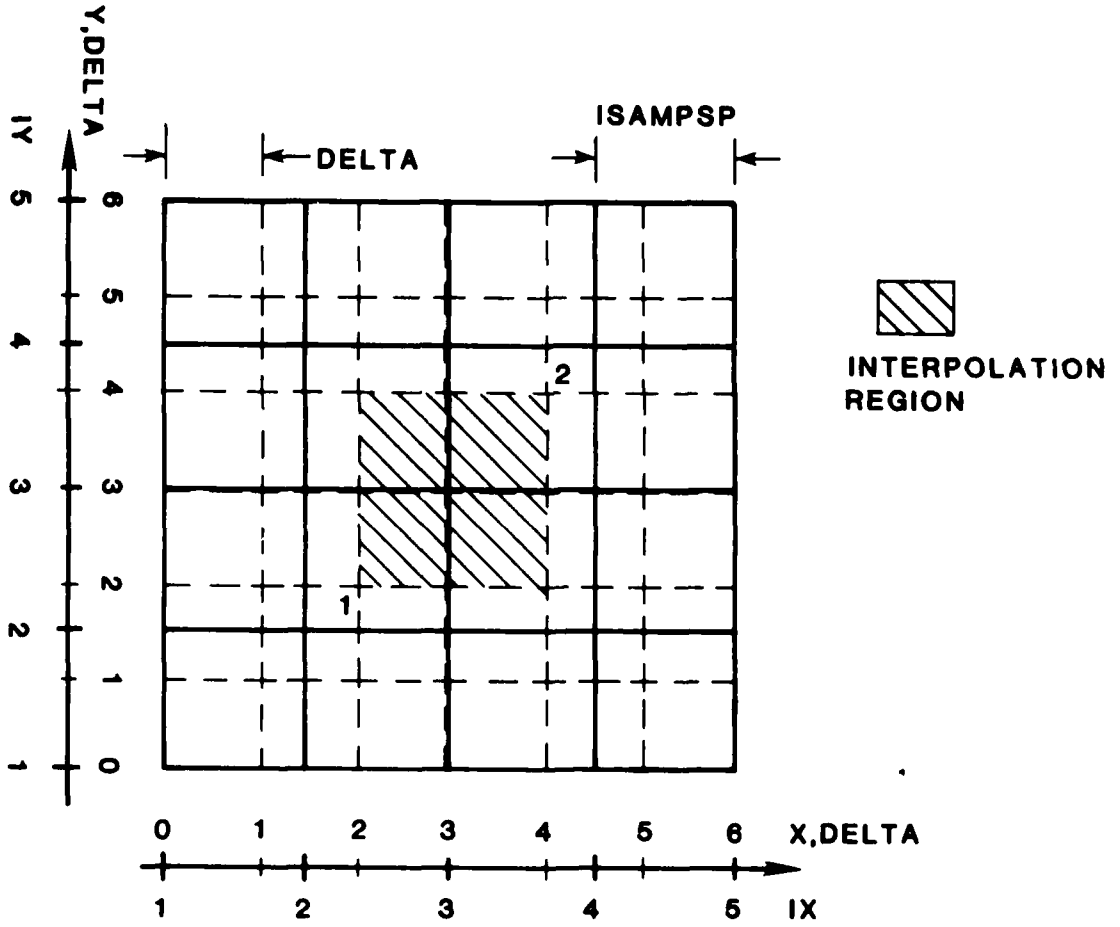


Figure N-6. Example with DPTDEN=1.5.

or when there is a round off error in DELTA. Therefore, the following check is required

```
IF(ISAMPSP.GT.((IPP+1)*DELTA)) IPP=IPP+1

IF(PICWIT-ISAMPSP.LE.(IUP+1)*DELTA) IUP=IUP-1
```

After the check, the example illustrated in Figure N-6 has limits IPP=1 and IUP=4.

For the general case, in terms of IPP and DELTA, the first point, (1), interpolated has world coordinates (X,Y) where X=Y=(IPP+1)DELTA and the last point interpolated, (2), has world coordinates X=Y=(IUP+1)DELTA. For the example in Figure N-6, it follows that the first point interpolated, (2), has world coordinates (X,Y) where X=Y=(IPP+1)DELTA=(2)DELTA. The last point interpolated, (3), has world coordinates (X,Y) where X=Y=(IUP+1)DELTA=(5)DELTA.

The FORTRAN variables "X" and "Y" are used for the values of the X and Y world coordinates, respectively, during the increments along each axis. The variable XX, shown in Figure N-7, is used to initialize both the X and Y increment. XX is calculated in the routine as XX=IPP*DELTA. In terms of XX and DELTA, the first interpolated point, (1), has world coordinates (X,Y)=(XX+DELTA,XX+DELTA).

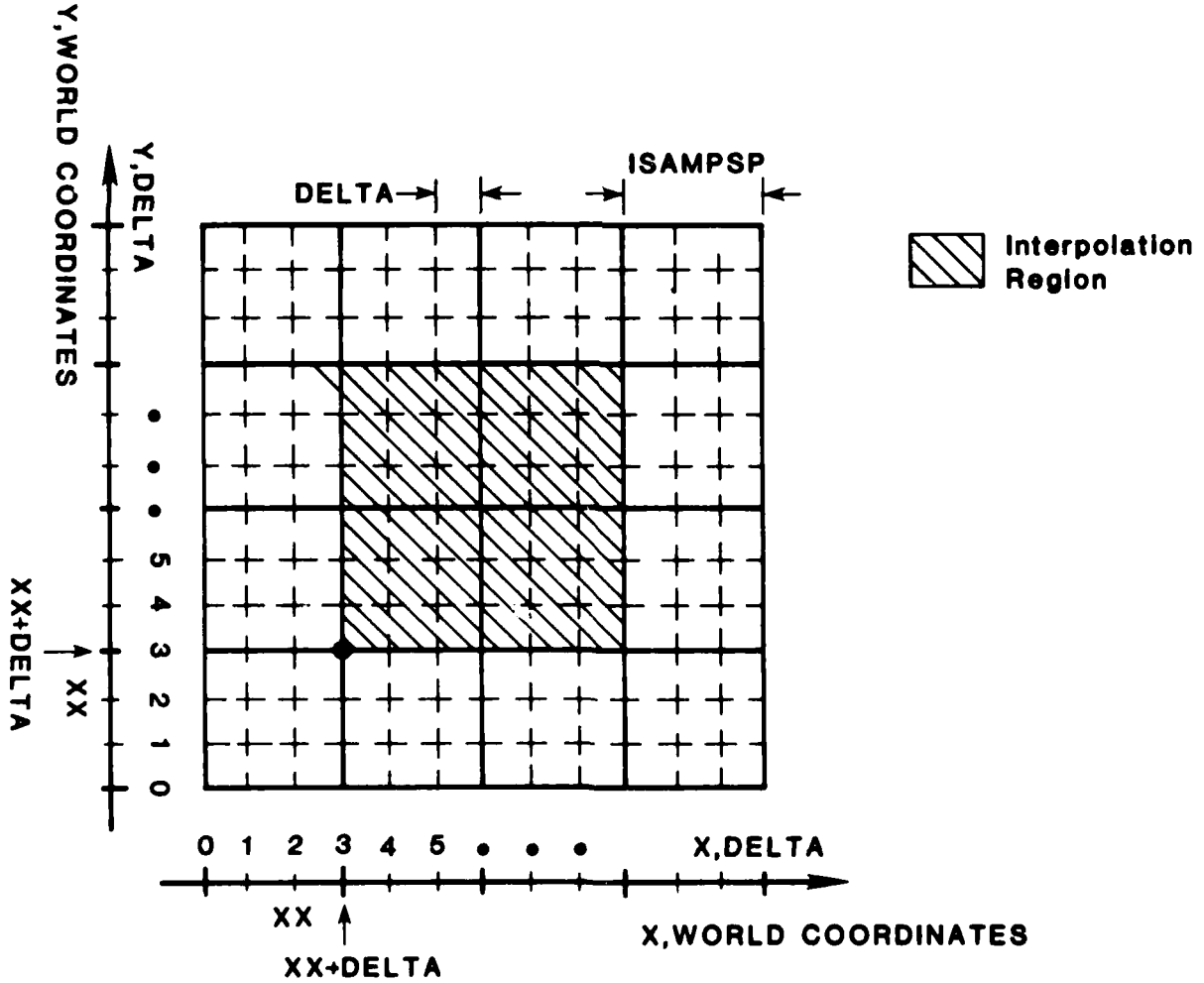


Figure N-8. Initialization of Increments with XX.

To allow for boundary crossing checks after each increment, the variable **FORTRAN** coordinates (IRLX,IRLY) and (IRUX,IRUY) are defined. The point (IRLX,IRLY) is the lower left boundary in world coordinates corresponding to the first potential point of the patch region. In Figure N-4b, this is the point with indices (IX,IY) such that IRLX=(IX)ISAMPSP and IRLY=(IY)ISAMPSP. The point (IRUX,IRUY) is the upper right boundary in world coordinates corresponding to the potential last point of the patch or the point with indices (IX+1,IY+1) such that IRUX=(IX+1)ISAMPSP and IRUY=(IY+1)ISAMPSP. In the **FORTRAN** routine IRLX and IRLY are initialized in the routine as ISAMPSP or IRLX=ISAMPSP and IRLY=ISAMPSP with IRUX and IRUY equal to IRUX=IRLX+ISAMPSP and IRUY=IRLY+ISAMPSP. After an increment, the world coordinate in the direction of the increment, X or Y, is compared to the upper boundary (IRUX if X is incremented or IRUY if Y is incremented). If the world coordinate is greater than the upper boundary then the boundary line has been crossed.

To allow the values of the Overhauser Functions of the boundary lines evaluated at a point (X,Y) to be accessed in the array C(4), the following equivalence is made:

```
C1(X)=C(1)          C3(X)=C(3)

C2(Y)=C(2)          C4(Y)=C(4)
```

From Figure N-8a, the boundary line crossed while incrementing along the Y axis is the upper horizontal line, C(3). With the crossing of the boundary, IRLY and IRUY are incremented such that IRLY = IRUY and IRUY = IRUY+ISAMPSP. The use of the prime notation in Figure N-8 denotes the new value of the variables in the next patch to be interpolated. When C(3) is crossed, the boundaries IRLX and IRUX are unchanged. The Overhauser Functins of the boundary lines are calculated for the next patch. Figure N-8b refers to incrementing along the X-axis so the boundary line crossed is the right vertical line, C(4). Here, the new values of IRLX and IRLY are IRLX = IRUX and IRUX = IRUX+ISAMPSP while IRLY and IRUY remain unchanged. With the crossing of a boundary line, the Overhauser Functions of the boundary lines are then calculated for the next patch. In the O-C routine the code which performs the increment in the X direction, the boundary crossing check, and the subsequent increments of the index and upper and lower boundaries is

```
X=X+DELTA
    IF(X.GT.IRUX)THEN
    IX=IX+1
    IRLX=IRUX
    IRUX=IRUX+ISAMPSP
    ENDIF
```

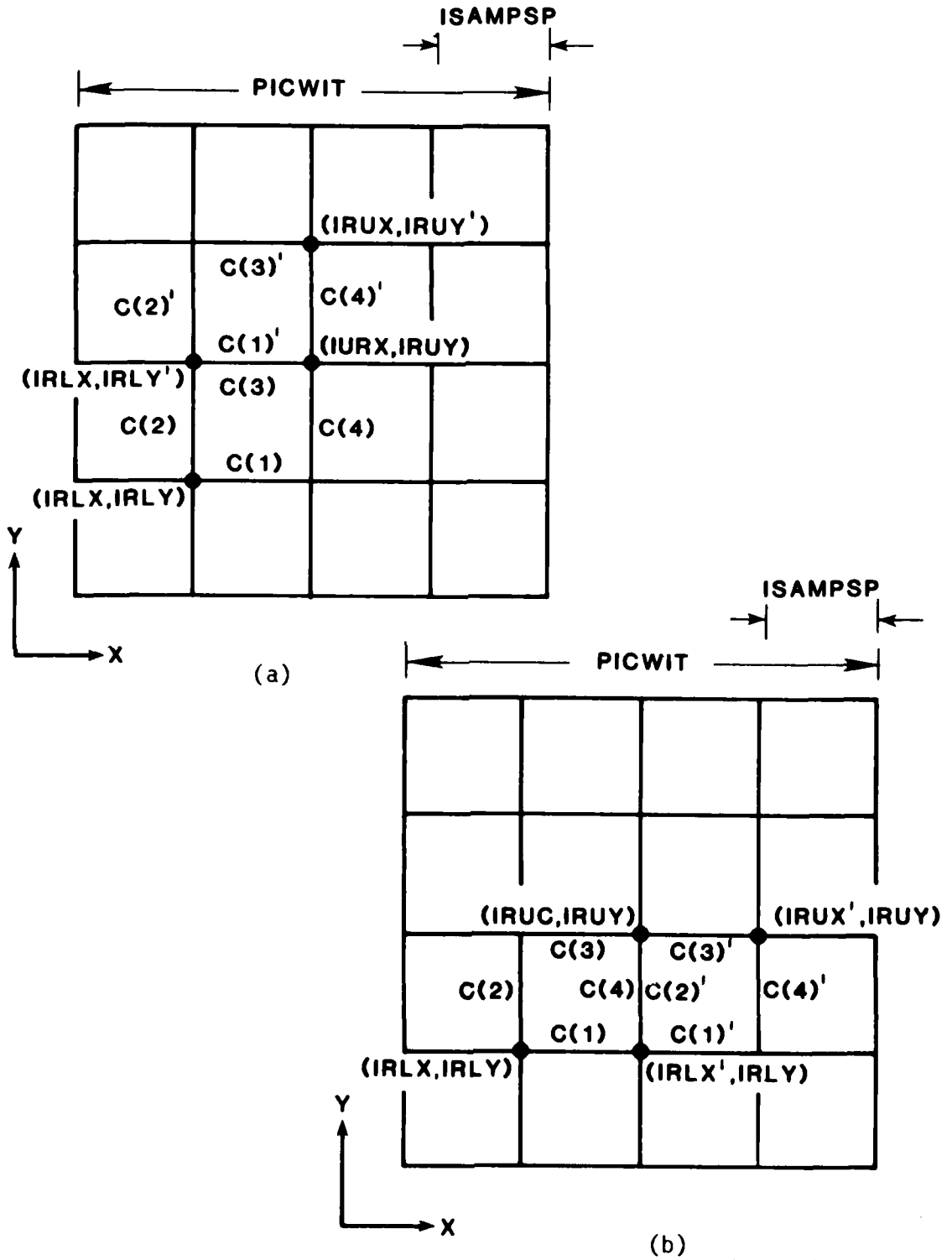


Figure N-8. Change of Boundary Lines and Boundary Points after a Boundary Crossing (a) after an Increment along the Y Axis (b) after an Increment along the X Axis.

In the Y-direction, the code is

```
Y=Y+DELTA
    IF(Y.GT.IRUX)THEN
    IY=IY+1
    IRLY=IRUY
    IRUY=IRUY+ISAMPSP
    ENDIF
```

The Overhauser Functions of the boundary lines and the Coon's Blending Functions are evaluated with the value of the incremental distances, t(X) and t(Y), calculated for each point (X,Y) interpolated within the patch. The incremental distance in either direction is obtained by subtracting the lower boundary from the world coordinate. Then each difference is normalized by dividing by the sample spacing, ISAMPSP. The incremental distance in the X-direction, t(X), is calculated in the O-C routine as

```
DPAC=1./ISAMPSP

TX=(X-IRLX)*DPAC
```

The incremental distance in the Y-direction, t(Y), is calculated as

```
TY=(Y-IRLY)*DPAC
```

When a patch boundary is crossed, the O-C Function is recalculated for the new patch. The Coon's Blending Functions are dependent only on the incremental distances at each point interpolated in the patch. The coefficients of the Overhauser Functions of the boundary lines, the array V(4,4), are generated by the subroutines PTCHNG and C(IC,JC).

In the Subroutine PTCHNG the height at each bounding corner of the patch, from the sample data array, SAMPT, is identified by the indices at the corners of the patch:

```
IX1=IX+1
IY1=IY+1

Z1=SAMPT(IX,IY)          !LOWER LEFT CORNER OF PATCH
Z2=SAMPT(IX1,IY)         !LOWER RIGHT CORNER OF PATCH
Z3=SAMPT(IX,IY1)         !UPPER LEFT CORNER OF PATCH
Z4=SAMPT(IX1,IY1)        !UPPER RIGHT CORNER OF PATCH
```

PTCHNG then calls subroutine C(IC,JC) to generate the coefficients of the boundary lines:

```
      CALL C(2,1)                    !LOWER HORIZONTAL LINE, C(1)
      CALL C(1,1)                    !LEFT VERTICAL LINE, C(2)
      CALL C(2,2)                    !UPPER HORIZONTAL LINE, C(3)
      CALL C(1,2)                    !RIGHT VERTICAL LINE, C(4)
```

The notation C(IC,JC) designates the boundary lines of the patch where IC=1 is a vertical line, IC=2 is a horizontal line, JC=1 is either the left or lower line, and JC=2 is either the right or the upper line.

The four coefficients of the Overhauser Functions are calculated for one line boundary in the subroutine C(IC,JC) and then assigned to the array V(4,4) by

```
      DO -- J2=1,4
              V(n,J2)=ANS(J2)
--    CONTINUE
```

Note that n represents the values 1 through 4 where V(1,J2) assigns coefficients to the equation for the boundary line C(1), and so on. The **FORTRAN** code for the calculation of the coefficients of the Overhauser Functions of the boundary lines and the assignment to the array V(4,4) for all four boundary lines in PTCHNG is accomplished by

```
      CALL C(2,1)                    !LOWER HORIZONTAL LINE, C(1)
      DO 20 J2=1,4
              V(1,2)=ANS(J2)
20            CONTINUE
      CALL C(1,1)                    !LEFT VERTICAL LINE, C(2)
      DO 30 J2=1,4
              V(2,J2)=ANS(J2)
30            CONTINUE
      CALL C(2,2)                    !UPPER HORIZONTAL LINE, C(3)
      DO 40 J2=1,4
               V(3,J2)=ANS(J2)
40             CONTINUE
      CALL C(1,2)                    !RIGHT VERTICAL LINE, C(4)
      DO 50 J2=1,4
              V(4,J2)=ANS(J2)
50            CONTINUE
```

The notation for the boundary lines in PTCHNG allows for a check in subroutine C(IC,JC) such that

```
      IF(IC.EQ.1) THEN !Y-DEPENDENT BOUNDARY LINES, C(2), C(4)
      IP=IX+JC-1
      K=2-IY
      DO 10 I=1,4
      ANS(I)=0
      DO 10 J=IY-1,IY+2
```

```
            ANS (I)=ANS(I)+MAT(I,J+K)*SAMPT(IP,J)
10          CONTINUE
      ELSE          !X-DEPENDENT BOUNDARY LINES, C(1), C(3)
         IP=IY+JC-1
         K=2-IX
         DO 20 I=1,4
              ANS(I)=0
              DO 20 J=IX-1,IX+2
                   ANS(I)=ANS(I)+MAT(I,J+K)*SAMPT(J,IP)
20 CONTINUE
      ENDIF
```

For the condition IC=1, the data matrix, MAT, is multiplied by the four points on one of the vertical (Y-dependent) boundary lines, C(2) or C(4). The four points on the Y-dependent boundary lines are accessed from the sample data array by SAMPT(IP,J). The **FORTRAN** variable IP is equal to the index in the X-direction of the sample data points on the boundary line. For a Y-dependent line, the value of X for the points on the line do not vary. Therefore, IP is a constant and is calculated as IP=IX+JC-1. If JC=1 then IP=IX, designating the left vertical line, C(2) (i.e., see Figure N-4b). If JC=2 then IP= IX+1, designating the right vertical line, C(4). The index J varies in the DO loop as

```
DO 10 J=IY,IY+2.
```

The points on C(2) are accessed from the array SAMPT when JC=1 by

```
SAMPT(IX,IY-1), SAMPT(IX,IY),

SAMPT(IX,IY+1), SAMPT(IX,IY+2).
```

The points on C(4) are accessed from the array when JC=2 by

```
SAMPT(IX+1,IY-1), SAMPT(IX+1,IY)

SAMPT(IX+1,IY+1), SAMPT(IX+1,IY+2).
```

If IC=1, the data matrix is multiplied by the four points on one of the horizontal (X-dependent) boundary lines, C(1) or C(3). The points are accessed by SAMPT(J,IP). The **FORTRAN** variable IP is calculated as IP=IY+JC-1. If JC=1 then IP=IY, designating the lower horizontal boundary line, C(1). If JC=2 the IP=IX1, designating the upper horizontal line, C(3). This time the index J varies in the DO loop as

```
DO 10 J=IX-1,IX+2.
```

The points on C(1) are accessed from the array SAMPT when JC=1 by

SAMPT(IX-1,IY), SAMPT(IX,IY)

SAMPT(IX+1,IY), SAMPT(IX+2,IY).

The points on C(3) are accessed from the array when JC=2 by

SAMPT(IX-1,IY+1), SAMPT(IX,IY+1)

SAMPT(IX+1,IY1), SAMPT(IX+2,IY+1).

The element of the data matrix are referenced in the DO loop by MAT(I,J+K). The index, I, varies from 1 to 4 which represents the four rows of the matrix. The **FORTRAN** variable K is used to shift J from the index of the points on the boundary line to the values of 1 through 4. To illustrate, with K=2-IY, then J and J+K become

```
J=IY-1,    J+K=IY-1+2-IY= 1
J=IY,      J+K=IY+2-IY=   2
J=IY+1,    J+K=IY+1+2-IY= 3
J=IY+2,    J+K=IY+2+2-IY= 4
```

The value of J+K represents the four columns of the data matrix. Each term of the matrix multiplication is summed into ANS(I) as I varies from 1 to 4. The result of the multiplication for each value of I is one of the coefficients of the Overhauser Functions where

```
ANS(1)=V(n,1)      ANS(3)=V(n,3)
ANS(2)=V(n,2)      ANS(4)=V(n,4), n=1,2,3,4.
```

The Overhauser Functions of the X-dependent boundary lines are calculated in the routine as

```
        DO 150 IOX=1,3,2
               CCX=V(IOX,1)
               DO 140 IEX=2,4
                      CCX=CCX*T(IOX)+V(IOX,IEX)
 140    CONTINUE
               C(IOX)=CCX
 150    CONTINUE
        CZ1=C(1)-Z1*BOX-Z2*B1X
        CZ2=C(3)-Z3*BOX-Z4*B1X
```

CZ1 and CZ2 are **FORTRAN** variables used for intermediate sums. The Overhauser Functions of the Y-dependent boundary lines as calculated in the routine as

```
        CZ=CZ1*BOY+CZ2*B1Y
        CSTUFF=0

        DO 70 IEY-2,4,2
```

```
                CCY=V(IEY,1)
                DO 60 IOY=2,4
                        CCY=CCY*T(IEY)+V(IEY,IOY)
60  CONTINUE
    CSTUFF=CSTUFF+CCY*B(IEY)
                        !CSTUFF=C(2)BOX+C(4)B1X
```

CZ and CSTUFF are, also, **FORTRAN** variables used for intermediate sums.

The Coon's Blending Functions in the X direction are computed as B1X=(3.0-2.0*TX)*TX2 and BOX=1.0-B1X. The **FORTRAN** code for TX2 is TX2=TX*TX. In the Y direction, they are calculated as B1Y=(3.0-2.0* TY)*TY2 and BOY=1.0-B1Y. The **FORTRAN** code for TY2 is TY2=TY*TY.

At this point, all the required information has been calculated so that the O-C Function can be evaluated to obtain the interpolated value of the height of the surface at a point (X,Y). The value of height, FEND, interpolated by the O-C Function is calculated as FEND= CSTUFF+CZ.

The Test of the Overhauser-Coons Bicubic Patch Routine

The accuracy of the O-C routine was tested by using a sample function FT(X,Y). The sample data array was computed at values (X,Y) on a grid of width ISAMPSP. The surface was interpolated at a data point density which kept the number of interpolated points constant. Each value of height of an interpolated point was compared with the exact value of the function at the same point (X,Y). The statistic used in evaluating the accuracy of the routine was the difference in the O-C value and the exact value at each point (X,Y). The differences at each point were summed to obtain the statistics considered in the results of the accuracy.

The test function which was used to evaluate the accuracy of the O-C procedure was

$$FT(X,Y)=127*SIN(6.283185*X/1024)*SIN(6.283185*Y/1024). \qquad (N-8)$$

The selection of a test function is arbitrary. This function was chosen because of the ease of detecting errors due to the symmetry of the X and Y coordintes.

The sample data array was determined by calculating the value of the test function, FT(X,Y), by Equation N-8, at each intersection of the grid. The following **FORTRAN** code was used to create the sample data array.

```
        DO 10 I=1,IEND
                XMIS-(I-1)*ISAMPSP
```

```
            DO 10 J=1,IEND
                  SAMPT(I,J)=FT(XMIS,(J-1)*XSAMPSP)
10  CONTINUE
```

The FORTRAN variable XMIS is the world coordinate in the X-direction of the sample data point while (J-1) *XSAMPSP is the world coordinate in the Y-direction. XSAMPSP is the floating point equivalent of ISAMPSP.

The statistic, DIF, was determined from the difference of the exact value, EXACT, and the O-C value, FEND, or DIF=EXACT-FEND. The differences squared, DIF2, and the absoluted value of the differences, DABS, was then determined. Also, two values of maximum difference was found: the maximum positive difference, DIFMXPS, and the maximum negative difference, DIFMXNG.

To evaluate the statistics, the differences were first summed. Round off error can occur when a small value is added to a large sum. To avoid round off error, the statistics were summed first for a segment, then for a line, and then for the whole picture.

The statistics for the interpolation region are summed first for each segment to calculate the segment statistics: the sum of segment differences, SGSMDF; the sum of segment differences squared, SGSMDF2; the sum of the absolute value of the segment differences, SGSMA; and the counter for the points in a segment, ICNT1. The segment data is then summed to calculate the line statistics: the sum of the line differences, SMLND; the sum of line differences squared, SMLND2; the sum of the absolute value of the line differences, SMLNA; and the counter for the points in a line, ICNTA. Then the line statistics are summed to calculate the picture statistics: the sum of the differences for the picture, SMDIF; the sum of the differences squared for the picture, SMDIF2; the sum of the absolute value of the differences, SMABS; and the counter for the total number of points interpolated in the picture, ICNT.

From the summation of the data, the average difference, the average of the absolute differences and the RMS (the standard deviation), can be calculated for the segments, the lines, and the picture. For the segment, these are as follows:

1. The standard deviation, SGRMS
   SGRMS=SQRT(SGSMDF2/ICNT1-SGAVG*SGAVG)
   where SGAVG=SGSMDF/ICNT1
2. The average of the absolute value of the differences
   SGABS=SGSMA/ICNT1.

For the line, these are as follows:

1. The standard deviation, LNRMS
   LNRMS=SQRT(SMLND2/ICNTA-LNAVG*LNAVG)
2. The average of the absolute value of the differences
   LNDABS=SMLNA/ICNTA.

The statistics of interest for the total picture are

1. The standard deviation, RMS
   RMS=SQRT(SMDIF2/ICNT-TOTAVG*TOTAVG)
   where TOTAVG=SMDIF/ICNT
2. The average of the absolute value of the differences
   DABS=SMABS/ICNT.

Note that the TOTAVG is approximately zero, on the order of 10**(-5).

The O-C routine was tested for accuracy at difference sample spacings while keeping the number of interpolated points per picture approximately the same. Two tests were performed to accomplish this, labeled "Test 1" and "Test 2". Both tests are included in LCOVRCNS. FOR (i.e., Appendix GB). The lines corresponding to the first method of testing are labeled "Test 1" and the second, "Test 2". Test 1 or Test 2 was performed by placing a D in the first character of a line corresponding to the test not desired.

Test 1 uses a **FORTRAN** loop of 40 steps where the sample spacing was varied from approximately 50 to 600. At the beginning, the sample spacing is calculated as

```
SAMPSPAC=50/1.064692          !Test 1:   1.0640929=12**1/40.
```

In the loop,

```
SAMPSPAC=SAMPSPAC*1.0640929   !TEST1.
```

The expression 12**(1/40) arises because the sample spacing varies from 50 to 600 where 600/50=12 in 40 steps for which the intervals between the steps increase with each successive step. Test 2 uses a **FORTRAN** loop of 30 steps where the sample spacing was varied from approximately 10 to 600. For Test 2, the sample sapcing is calculated as

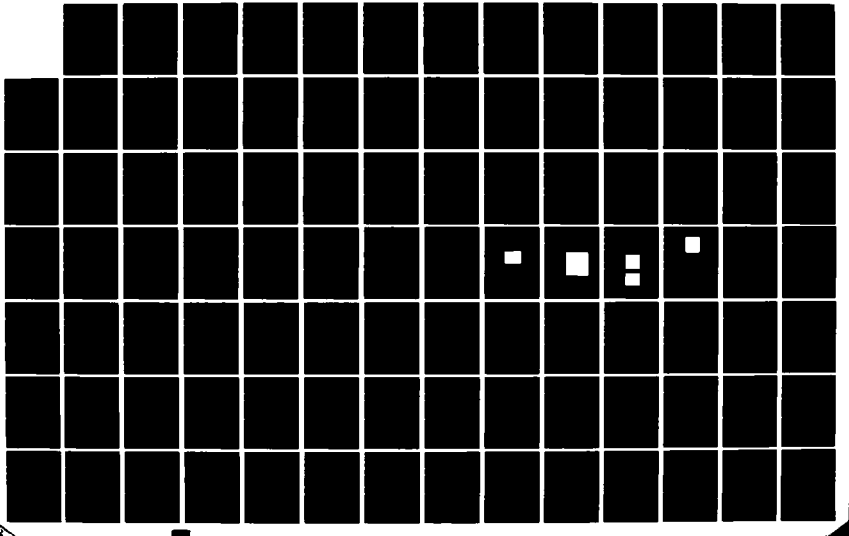
```
SAMPSPAC=10/1.1462298         !TEST 2:   1.1462298 60**(1/30)
```

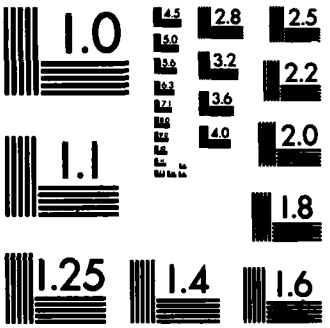
In the loop,

```
SAMPSPAC=SAMPSPAC*1.1462298   !TEST 2
```

The expression 600**(1/30) was used [illegible]
from 10 to 600, 600/10 60, in 30 steps [illegible]

AD-A122 000 REAL SCAN EVOLUTION(U) UNIVERSITY OF CENTRAL FLORIDA 4/6
ORLANDO DEPT OF ELECTRICAL ENGINEERING B W PATZ ET AL.
FEB 82 NAVTRAEQUIPC-80-D-0014-2 N61339-80-D-0014
UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

SPAC has been calculated, the integer value of the sample spacing is calculated as ISAMPSP=SAMPSPAC.

It was desired that the variation between two sample spacing increase as the value of the spacings increased. This enabled more tests to be run at smaller values of ISAMPSP where the accuracy was assumed to be better because more sample data points were used. For Test 1, the sample spacing divided by one-half the period, ISAMPSP/512, varied from 0.0957 to 1.0966. For Test 2, the sample spacing divided by one-half the period varied approximately 0.0195 to 1.0215.

The data point density, DPTDEN, was calculated so that the number of points interpolated remained constant for each test. At the beginning of the routine, the picture width, PICWIT, and the number of points to be interpolated, POINTS, were entered by the user. These values were used in the computation of DPTDEN for each step as follows

```
SQRPTS=SQRT(POINTS)    !CALCULATION OF DPTDEN ENSURES A
IPICSMP=PICWIT*DPAC    !CONSTANT NUMBER OF INTERPOLATED
IF(IPICSMP.LE.2) GOTO 400     !POINTS FOR EACH STEP
                       !REQUIRED DATA BASE NOT AVAILABLE
DENOM=IPICSMP-2
DPTDEN=SQRPTS/DENOM
```

Each time the loop is started, new values for SAMPSPAC, ISAMPSP, and DPTDEN are calculated.

Because 12 adjacent points are needed for interpolation, PICWIT must contain at least three divisions of width ISAMPSP. A check is provided such that

```
    IPICSMP=PICWIT*DPAC
    IF(IPICSMP.LE.2) GOTO 400
            ...
400 STOP
```

For a given PICWIT and COUNT, the DPTDEN may be less than one. This means that some patches may have no interior points interpolated. Since the purpose of this routine is data base reduction, the case of DPTDEN less than one is not considered. For DPTDEN less than one, some data points might be stored but not used in the interpolation of interior points. A check is provided for this case.

```
    IF(DPTDEN.LT.1)THEN
           TYPE*, `ROUTINE IS INVALID FOR DPTDEN < 1´
           GOTO 210
    ...
210 CONTINUE
```

The O-C routine was tested for

PICWIT=7200, NUMBER OF POINTS=.25 MILLION
PICWIT=2048, NUMBER OF POINTS=.25 MILLION
PICWIT=2048, NUMBER OF POINTS=1000.

The results are found in Tables N-1 through N-3. The first lists the errors corresponding to the sample spacing and the data point density: ISAMPSP, DPTDEN, DIFMXPS, DIFMXNG, RMS/127 and DAVG. Note the RMS is normalized by dividing by the maximum value of the test function which is 127. The number of points equal to .25 MILLION was chosen to get a sufficiently large sample. One run of only 1000 points was included for comparison. Since one period of the function is 1024, a picture width of 7200 includes seven periods and a width of 2048 includes two periods.

The accuracy of the interpolation is a function of the sample spacing. The value of DPTDEN, by definition in the test runs, ensures that the total number of interpolated points will remain constant for a given value of ISAMPSP. The larger the sample spacing the smaller the data base required. Use of the O-C routine allows for data base reduction but with a finite error in the interpolated values for the points. This error must be within the tolerated error of the specific application of the routine.

Note in the table of data for PICWIT=2048 and the approximate number of points interpolated equal to 1000, the value for the RMS/127 increases with the sample spacing until ISAMPSP=322. This inconsistency does not occur until the last value of ISAMPSP when PICWIT=2048 and the number of points is equal to .25 million. The inconsistency of the RMS occurs when the maximum positive and negative differences vary significantly. Since this test function is sinusiodal, it has equal positive and negative values over a period. Interpolation of a period of the function results in approximately equal maximum positive differences, DIFMXPS, and maximum negative differences, DIFMXNG. The function, however, may not be evenly interpolated over a period because the sample spacing is too large. Remember that an area the width of ISAMPSP and the length of PICWIT is not interpolated along each axis. A large sample spacing would require that a large area not be interpolated so that the sample data array will be available. Therefore, a good representation of a period would not have been interpolated. As a result, the maximum positive and negative differences are not approximately equal. Also, if the data point density is large, a period may not be interpolated evenly.

The results of the interpolation of a surface using the Overhauser-Coons Function was compared to the results using a straight line approximation. The straight line approximation assumes a straight line between given sample data points and interpolates the height along the line for the interior of the patch. To interpolate

TABLE N-1. O-C INTERPOLATION WITH PICWIT=7200*

ISMAPSP	DPTDEN	DIFMXPS	DIFMXNG	RMS/127	DAVG
49	3.4722	0.1501	0.1495	0.0004013	0.0400
53	3.7594	0.1812	0.1807	0.0005019	0.0501
56	3.9683	0.2123	0.2120	0.0005890	0.0591
60	4.2373	0.2545	0.2476	0.0007212	0.0718
64	4.5455	0.2950	0.2950	0.0008757	0.0880
68	4.8544	0.3587	0.3571	0.0010524	0.1060
72	5.1020	0.4227	0.4217	0.0012539	0.1265
77	5.4945	0.5092	0.5082	0.0015479	0.1564
82	5.8824	0.6163	0.6168	0.0018890	0.1909
87	6.2500	0.7402	0.7308	0.0022831	0.2306
93	6.6667	0.9035	0.8994	0.0028311	0.2854
98	7.0423	1.0788	1.0786	0.0033593	0.3375
105	7.5758	1.3523	1.3491	0.0042180	0.4248
112	8.0645	1.6831	1.6818	0.0052245	0.5255
119	8.6207	2.0825	2.0819	0.0064009	0.6439
126	9.0909	2.5688	2.5669	0.0077564	0.7793
135	9.8039	3.3287	3.3269	0.0097933	0.9811
143	10.4167	4.1318	4.1309	0.0118981	1.1867
152	11.1111	5.1591	5.1588	0.0146192	1.4537
162	11.9048	6.5545	6.5253	0.0180936	1.7939
173	12.8205	8.3122	8.3111	0.0224413	2.2088
184	13.5135	10.2923	10.3609	0.0278598	2.7402
196	14.7059	12.9132	12.8581	0.0337104	3.3156
208	15.6250	15.9472	15.1415	0.0409930	4.0204
221	16.6667	19.5749	19.5167	0.0497849	4.8859
236	17.8571	24.1780	24.0867	0.0612823	6.0408
251	19.2308	29.7601	29.7349	0.0738485	7.0595
267	20.8333	35.9613	35.9448	0.0903933	8.8995
284	21.7391	43.5116	43.4260	0.1068842	10.3175
302	23.8095	51.8541	51.1767	0.1302255	12.7202
322	25.0000	61.5244	61.1060	0.1509459	14.7113
342	26.3158	59.3524	56.9323	0.1774788	16.4165
364	29.4118	82.7326	82.3238	0.2172158	21.4628
388	31.2500	94.2486	94.0499	0.2507326	24.9101
413	33.3333	105.1786	99.6477	0.2873450	28.7446
439	35.7143	111.9790	111.7547	0.3306687	33.2256
467	38.4615	122.1083	120.9100	0.3834579	39.1429
497	41.6667	126.0886	125.7895	0.4452635	46.7764
529	45.4545	133.5335	132.7111	0.4727468	49.9488
563	50.0000	154.8192	154.2198	0.4934835	50.8588

*The approximate number of interpolated points is 0.25 million

TABLE N-2. O-C INTERPOLATION WITH PICWIT=2048*

ISMAPSP	DPTDEN	DIFMXPS	DIFMXNG	RMS/127	DAVG
10	2.4752	0.0045	0.0045	0.0000092	0.0008
11	2.7174	0.0055	0.0055	0.0000113	0.0010
13	3.2258	0.0077	0.0077	0.0000160	0.0014
15	3.7313	0.0104	0.0103	0.0000218	0.0020
17	4.2373	0.0134	0.0134	0.0000286	0.0026
19	4.7619	0.0171	0.0170	0.0000366	0.0034
22	5.4945	0.0232	0.0231	0.0000513	0.0048
25	6.3291	0.0309	0.0308	0.0000689	0.0094
29	7.3529	0.0430	0.0429	0.0000990	0.0143
34	8.6207	0.0616	0.0614	0.0001483	0.0143
39	10.0000	0.0840	0.0840	0.0002118	0.0205
44	11.3636	0.1146	0.1139	0.0002954	0.0291
51	13.1579	0.1640	0.1634	0.0004491	0.0447
58	15.1515	0.2328	0.2327	0.0006555	0.0656
67	17.8571	0.3440	0.3424	0.0010183	0.1026
77	20.8333	0.5130	0.5090	0.0015795	0.1600
88	23.8095	0.7678	0.7658	0.0024339	0.2471
101	27.7778	1.1898	1.1893	0.0038489	0.3928
116	33.3333	1.9046	1.8925	0.0061084	0.9893
133	38.4615	3.1464	3.1460	0.0097649	0.9893
153	45.4545	5.2805	5.2523	0.0156466	1.5673
175	55.5556	8.6817	8.6435	0.0234878	2.3688
201	62.5000	13.8902	13.1254	0.0384678	3.7019
230	83.3333	22.2807	21.8251	0.0547172	4.9317
264	100.0000	24.9628	23.2969	0.0824098	9.0863
303	125.0000	52.1643	45.6776	0.1278739	12.1058
347	166.6667	63.9731	57.7080	0.1945773	18.3542
398	166.6667	66.7295	52.5033	0.1878833	18.8707
456	250.0000	104.5176	92.5886	0.3884520	41.1434
523	500.0000	126.8260	17.8727	0.3051477	49.4079

*The approximate number of interpolated points is 0.25 million

TABLE N-3. O-C INTERPOLATION WITH PICWIT=2048*

ISMAPSP	DPTDEN	DIFMXPS	DIFMXNG	RMS/127	DAVG
64	1.0541	0.2926	0.2590	0.0008768	0.0880
68	1.1294	0.2373	0.2273	0.0006796	0.0689
72	1.2163	0.4221	0.4098	0.0012742	0.1277
77	1.3176	0.5019	0.4913	0.0015703	0.1610
82	1.4374	0.5966	0.5889	0.0019384	0.1935
87	1.5058	0.7402	0.7366	0.0022462	0.2225
93	1.5811	0.8962	0.8974	0.0029552	0.2966
98	1.7568	1.0282	1.0298	0.0035742	0.3751
105	1.8702	1.3332	1.3098	0.0043543	0.4530
112	1.9764	1.6409	1.6388	0.0051836	0.5231
119	2.1082	2.0749	2.0756	0.0070340	0.7244
126	2.2588	2.5602	2.5580	0.0079665	0.7801
135	2.4325	3.3126	3.2988	0.0102512	1.0333
143	2.6352	4.1099	3.9966	0.0124111	1.2324
152	2.8748	5.0344	4.9714	0.0151267	1.5355
162	3.1623	6.3543	6.2601	0.0188205	1.8634
173	3.5136	8.2493	8.1369	0.0222858	2.2827
184	3.5136	9.6853	9.6950	0.0290915	2.8322
196	3.9528	12.4831	10.8476	0.0340630	3.3775
208	4.5175	14.6375	14.5238	0.0394553	3.7340
221	4.5175	19.4778	17.8292	0.0502152	4.9720
236	5.2705	24.1607	22.4294	0.0589800	5.4024
251	5.2705	19.8725	18.7292	0.0737056	8.3514
267	6.3246	27.6182	25.7433	0.0850510	9.2224
284	6.3246	42.1395	40.0836	0.0982114	8.8699
302	7.9057	51.3623	45.4790	0.1274853	12.0538
322	7.9057	57.8143	36.8851	0.1290846	12.5800
342	10.5409	55.3536	54.8264	0.1844455	17.3461
364	10.5409	82.7272	58.7527	0.2102028	20.4021
388	10.5409	78.4418	53.5184	0.1974677	19.5804
413	15.8114	52.3859	42.0549	0.1592090	15.3632
439	15.8114	85.8550	69.2740	0.2895007	30.4893
467	15.8114	113.2244	105.9467	0.4381936	46.0463
497	15.8114	125.4555	125.1960	0.5024350	52.1929
529	31.6228	126.6752	24.8862	0.3201352	48.1689
563	31.6228	127.4397	79.3752	0.4397611	52.8839

*The approximate number of interpolated points is 1000.

the interior of a patch by the Straight Line Approximation, only the heights at the four corners of the patch are needed, Z1, Z2, Z3, and Z4. The data base for the straight line approximation is the same as for the O-C Function, a sample data array of the surface in the form of a grid. The definitions of the incremental distances TX and TY and the width of the patch, ISAMPSP, are the same as the O-C routine. Using these definitions, the function Z(X,Y) for the Straight Line interpolation is

$$Z(X,Y)=Z1+TX(Z2-Z1)/ISAMPSP+TY(Z3-Z1)/ISAMPSP$$
$$+(TX)(TY)[(Z4+Z3)-(Z2+Z1)]/ISAMPSP^2 \qquad (N-9)$$

The **FORTRAN** listing of the Straight Line Approximation Routine, LCSTRAIT.FOR, is in Appendix GC. All of the variables used in the O-C routine have the same meaning in the Straight Line Approximation Routine. The order of computation and the calculation of the error are also the same.

The Straight Line Approximation does not have equations for the boundary lines Cn(X) or Cn(Y). Therefore, the subroutine in the Straight Line Approximation (SLA) is different from the subroutines in the O-C routine. Subroutine PITCH of the SLA assigns the values of the array SAMPT to Z1, Z2, Z3, and Z4, according to the index of the point, as in subroutine PTCHNG. Subroutine PITCH performs addition and subtration on the height at the corners to produce

```
C1=Z2-Z1
C2=Z3-Z1
C3=Z4-Z3-Z2+Z1
```

The **FORTRAN** code for the Straight Line Approximation is

```
FEND=Z1+TX*C1/ISAMPSP+TY*C2/ISAMPSP+TX*TY*C3/ISAMP2
```

where

```
ISAMP2=ISAMPSP*ISAMPSP.
```

The Straight Line Approximation was tested in the same manner as the O-C routine.

The results of the rms (i.e., root of the mean of the squares) elevation error versus the sample spacing of the test function are plotted in Figure N-9 for PICWIT=7200 and the number of interpolated points equal to 0.25 million. The data was produced by Test 1 for 40 steps. The rms error is normalized by dividing by the maximum height of the test function, 127. The sample spacing is divided by one half the period where the period of the test function is 1024. This normalized the sample space to the Nyquist rate. Figure N-9 is a plot

TABLE N-4. O-C AND S-L INTERPOLATION COMPARISON I*

ISMAPSP/512	(O-C)RMS/127	(S-L)RMS/127
0.0957	0.0004013	0.0079639
0.1035	0.0005019	0.0093148
0.1094	0.0005890	0.0102789
0.1172	0.0007212	0.0118682
0.1250	0.0008757	0.0135461
0.1328	0.0010524	0.0153270
0.1406	0.0012539	0.0170596
0.1504	0.0015479	0.0195759
0.1602	0.0018890	0.0221940
0.1699	0.0022831	0.0249310
0.1816	0.0028311	0.0283858
0.1914	0.0033593	0.0314973
0.2051	0.0042180	0.0360472
0.2188	0.0052245	0.0408470
0.2324	0.0064009	0.0459282
0.2461	0.0077564	0.0512588
0.2637	0.0097933	0.0584865
0.2793	0.0118981	0.0651952
0.2969	0.0146192	0.0730509
0.3164	0.0180936	0.0820243
0.3379	0.0224413	0.0929381
0.3594	0.0278598	0.1038275
0.3828	0.0337104	0.1139896
0.4063	0.0409930	0.1265628
0.4316	0.0497849	0.1401669
0.4609	0.0612823	0.1563502
0.4902	0.0738485	0.1720698
0.5215	0.0903933	0.1922699
0.5547	0.1068842	0.2087045
0.5898	0.1302255	0.2330021
0.6289	0.1509459	0.2490599
0.6680	0.1774788	0.2714307
0.7190	0.2172158	0.3057964
0.7578	0.2507326	0.3288822
0.8066	0.2783450	0.3531354
0.8574	0.3306687	0.3821282
0.9121	0.3835679	0.4175627
0.9707	0.4452635	0.4586033
1.0332	0.4727468	0.4698608
1.0996	0.4934835	0.4754561

*PICWIT=7200, total interpolations=0.25 million

of RMS/127 vs. ISAMPSP/512. The values that are plotted in Figure N-9 are tabulated in Table N-4. The plot for the Straight Line Approximation shows greater normalized rms error and therefore more error than the plot for the Overhauser-Coons Function. The normalized rms as a function of the sample spacing divided by one-half the period can be estimated by extrapolating the lines connecting the plotted data to determine the slopes. The Straight Line Interpolation error may be approximated as the normalized RMS, say NRMS is

$$NRMS \sim [ISAMPSP/(PERIOD/2)]^2 \qquad (N\text{-}10)$$

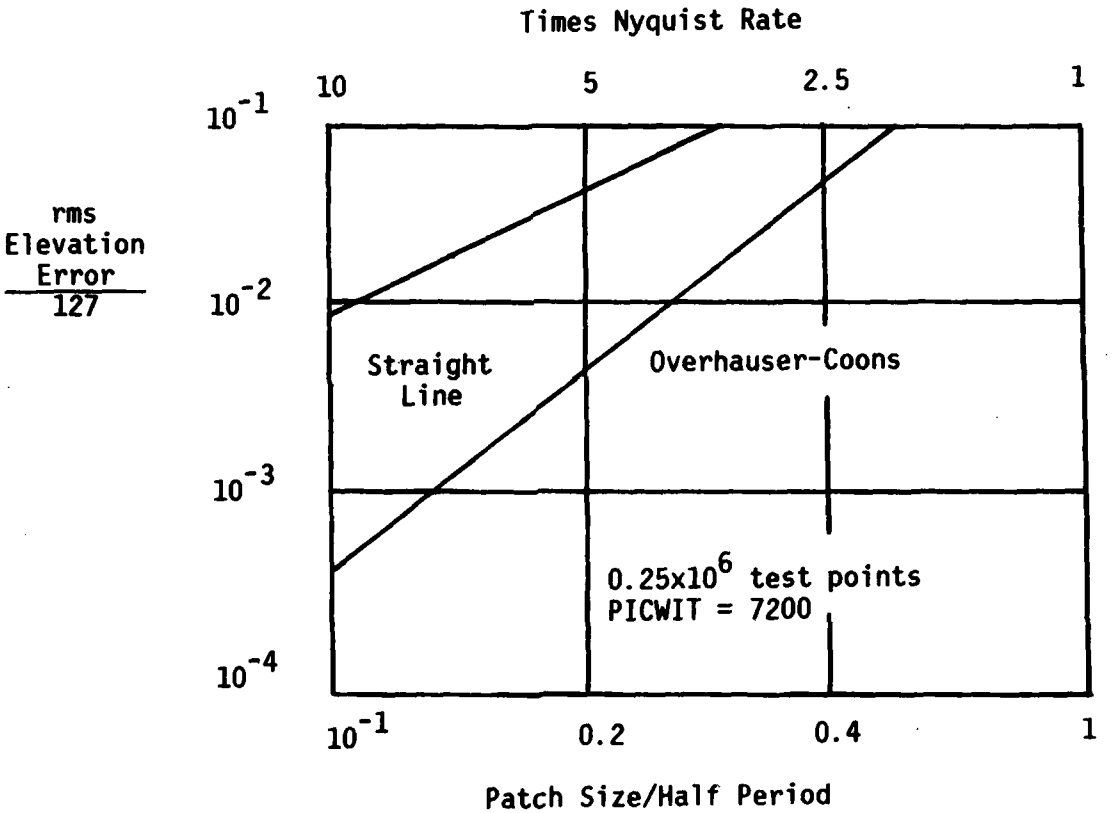


Figure N-9. Plot of RMS Divided by Maximum Height of Test Function, 127 vs. the Sample Spacing Divided by One-Half the Period, 512, for 0.25 Million Interpolated Points by Overhauser-Coons and Straight Line Approximation.

TABLE N-5. O-C AND S-L INTERPOLATION COMPARISON II*

ISMAPSP/512	(O-C)RMS/127	(S-L)RMS/127
0.0195	0.0000092	0.0003335
0.0215	0.0000113	0.0004029
0.0254	0.0000160	0.0005649
0.0293	0.0000218	0.0007538
0.0332	0.0000286	0.0009699
0.0371	0.0000366	0.0012176
0.0430	0.0000513	0.0016266
0.0488	0.0000689	0.0021259
0.0566	0.0000990	0.0028640
0.0664	0.0001483	0.0039334
0.0762	0.0002118	0.0051861
0.0859	0.0002954	0.0066749
0.0996	0.0004491	0.0089493
0.1133	0.0006555	0.0116582
0.1309	0.0010183	0.0156843
0.1504	0.0015795	0.0207622
0.1719	0.0024339	0.0270615
0.1973	0.0038489	0.0356075
0.2266	0.0061084	0.0462589
0.2598	0.0097649	0.0603879
0.2988	0.0156466	0.0779049
0.3418	0.0234878	0.0945775
0.3926	0.0384678	0.1250140
0.4492	0.0547172	0.1454254
0.5156	0.0827098	0.1793533
0.5918	0.1278739	0.2286545
0.6777	0.1945773	0.2896061
0.7773	0.1878833	0.2552577
0.8906	0.3884520	0.4268898
1.0215	0.3051477	0.3049439

*PICWIT=7200, total interpolations=0.25 million

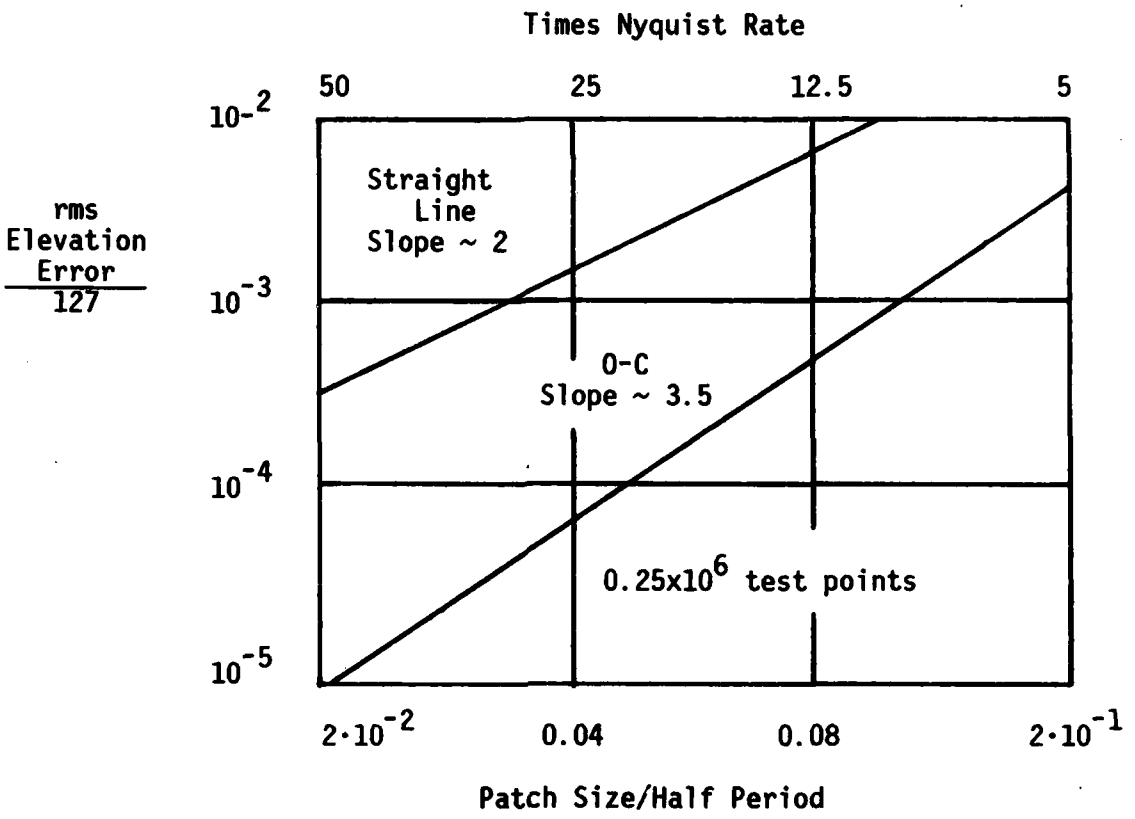


Figure N-10. Plot of RMS/127 vs. ISAMPSP/512 for PICWIT=2048.00 and Points 0.25 Million for the Overhauser-Coon's and the Straight Line Approximation.

For the Overhauser-Coons Bicubic Patch Function,

$$NRMS \sim [(ISAMPSP/(PERIOD/2)]^{3.3}.$$

Figure N-10 is a plot for a picture width of 2048 and the number of points interpolated equal to 0.25 million. The data was produced by Test 2 for 30 steps. The data is also tabulated in Table N-5. Considering a straight line connecting the plotted values, the slopes of the lines result in an equation for the normalized RMS of the Straight Line Interpolation which is the same as above. For the Overhauser-Coons Function,

$$NRMS \sim [ISAMPSP/(PERIOD/2)]^{3.5}.$$

APPENDIX O

DEVELOPMENT OF VARIABLE INCREMENTS ALONG SCAN LINES

Problem: Determine the largest increment along a scan line which still guarantees attaining fixed resolution.

Solution: Fixed resolution means the angle, α, subtended by a world point or region at the eye remains constant.

$$\alpha = \Delta r/R$$

α is the linear dimension guaranteeing this resolution.

R is the range from the eye to the point.

So Δr is the largest increment which still guarantees fixed resolution.

Problem: Determine a sequence of increments, limited by the number of bits defining the increment's precision, which maintains overall accuracy and eliminates round-off by being exact at hierarchy level boundaries.

Solution: Figure O-1 illustrates a sequence of increments from R to 2R, the boundaries between three neighboring hierarchy levels.

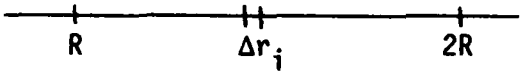


Figure O-1. Scan Line Increment Δr_i in the Hierarchy Level Bounded between R and 2R.

The sum of the increments must be R. If one normalizes the first of the p increments to approximately $1/\alpha$ then

$$R = \sum_{i=0}^{p-1} \Delta r_i \qquad (O\text{-}1)$$

but

$$\Delta r_i = \alpha R + \alpha \sum_{j=0}^{i-1} \Delta r_j \qquad (O\text{-}2)$$

so

$$R = \alpha R \sum_{i=0}^{p-1} (1 + \alpha)^i \qquad (0\text{-}3)$$

and

$$2 = (1 + \alpha)^p \qquad (0\text{-}4)$$

Hence, the normalized increments are $1, 1 + \alpha, (1 + \alpha)^2, \ldots (1 + \alpha)^{p-1}$.

Equation 0-4 yields the exact relation between α and p. Since $\alpha < 1/500$, Equation 0-4 can be accurately approximated as

$$\ln 2 \sim \alpha p \qquad (0\text{-}5)$$

Equation 0-4 shows that $p \geq \ln 2/\alpha$ if one is going to maintain the lower bound resolution, α, established by the specified display resolution accuracy divided by the number of ground points being averaged per pixel length. The initial analyses of **REAL SCAN** assume display resolution is 1/512 and 4 points per pixel area so $\alpha = 1/1024$.

It is desirable to consider

$$\Delta r_i = N_i . F_i \qquad (0\text{-}6)$$

where

N_i is the integer portion of the increment

F_i is the base 2 fraction of the increment

Since this form clearly identifies the finite precision of an increment and relates it to a computer's resolution.

The current version of **REAL SCAN** sets hierarchy levels on integers, based on the resolution. Table 0-1 illustrates the level, its range, the increment within the level, and the number of increments per level.

TABLE O-1. HIERARCHY LEVEL PARAMETERS

Level	Nearest Approach	Increment	Number of Increments	R	2R
0	1024	1	2048	0	2048
1	2048	2	1024	2048	4096
2	4096	4	1024	4096	8192
3	8192	8	1024	8192	16384
4	16384	16	1024	16384	32768
n	$1024 \cdot 2^n$	2^n	1024	$1024 \cdot 2^n$	$1024 \cdot 2^{n+1}$

The solution to the problem posed in this Appendix would yield hierarchy level parameters of the form illustrated in Table O-2. Table O-2 indicates a level O′ directly below the eye where all increments are at the resolution limit. The increment in all other levels ranges from 1 to 2 as give by Equations O-4, O-5, and O-6. The minimum number of increments is $\ln 2/\alpha$, but since Equation O-6 defines a limited precision computer, the actual number of increments will slightly exceed $\ln 2/\alpha$ for guaranteed resolution, or slightly undershoot $\ln 2/\alpha$ for simplicity with judged acceptable picture quality.

TABLE O-2. IMPROVED HIERARCHY LEVEL PARAMETERS

Level	Nearest Approach	Increment	Number of Increments	R	2R
0′	1024	1	1024	0	1024
0	1024	1 to 2	$1024 \cdot \ln 2$	1024	2048
1	2048	2 to 4	$1024 \cdot \ln 2$	2048	4096
2	4096	4 to 8	$1024 \cdot \ln 2$	4096	8192
3	8192	8 to 16	$1024 \cdot \ln 2$	8192	16384
n	$1024 \cdot 2^n$	2^n to 2^{n+1}	$1024 \cdot \ln 2$	$1024 \cdot 2^n$	$1024 \cdot 2^{n+1}$

Let us define the total number of increments as Q and allow them to be separated into k segments of q increments each, then

$$Q = kq \geq \ln 2/\alpha \tag{0-7}$$

The increment in segment i, $(1 \leq i \leq k)$ is therefore

$$(1 + \beta)^{i-1} = \Delta r_i \tag{0-8}$$

with

$$\Delta r_1 \geq 1 \text{ and } \Delta r_q \leq 2.$$

Equation 0-8 leads to the condition

$$q \sum_{i=1}^{k} \Delta r_i = q \sum_{i=1}^{k} (1+\beta)^{i-1} = 1/\alpha \tag{0-9}$$

If the condition for maintaining accuracy at hierarchy level boundaries is to be satisfied, then the following numbers must be integers: $1/\alpha$, k, and q. Hence, Equations 0-6 and 0-9 show that

$$q \cdot 2^{-n} (\Sigma N_i . F_i) \cdot 2^n = 1/\alpha \tag{0-10}$$

where $N_i . F_i \cdot 2^n$ is an integer.

For a 512*512 screen having at least 4 ground points per pixel, $1/\alpha = 2^{10}$ and Equation 0-10 shows both q and the Equation 0-10 pseudo fraction sum must be powers of 2. If q = 2, the Equation 0-7 yields

$$k \geq 2^{10-m} \ln 2 \tag{0-11}$$

Table 0-3 illustrates integer values which satisfy Equation 0-11. The first four entries in Table 0-3 indicate that one can use a few segments and therefore a few unique increments, or one can use a larger number of segments. The larger the number of segments the larger the number of bits required to represent the pseudo fraction. A resolution of about two points per pixel for 512*512 pixels can be achieved with a four-bit fraction having each fraction repeated 64 times.

The next four entries in Table 0-3 indicate that one can use a number of points per segments which is not a power of two, if the resolution is given by an integer having other factors. Hence, a four bit fraction can represent the 14 segments where each fraction is repeated 50 times to make the 700 increments totaling a normalized length of 1000, the inverse resolution.

TABLE O-3. SCAN LINE PARAMETERS ACHIEVING A SPECIFIED RESOLUTION WITH A GUARANTEED UPPER BOUND ON THE NUMBER OF TEST POINTS ALONG THE SCAN LINE

q Points per Segment	k Number of Segments	Q Total Test Points	$\ln 2/\alpha$ Resolution Test Points	α Specified Resolution
64	11	704	709.78	1/1024
16	44	704	709.78	1/1024
16	45	720	709.78	1/1024
32	23	736	709.78	1/1024
25	28	700	693.15	1/1000
50	14	700	693.15	1/1000
10	70	700	693.15	1/1000
40	17	680	693.15	1/1000
64	7	448	443.61	1/640
16	28	448	443.61	1/640
20	22	440	443.61	1/640
20	23	460	443.61	1/640
32	11	352	354.89	1/512
16	22	352	354.89	1/512
16	23	368	354.89	1/512
64	6	384	354.89	1/512

Table O-4 illustrates various finite resolution solutions to the first entry in Table O-3. The 11 segments have values ranging from 1 to 2 for 3 bit fractinal accuracy and values ranging from 67/64 to 125/64 for 6 bit fractional accuracy.

This Appendix has shown that the use of a pseudo fractional increment in ground coordiates can achieve absolute accuracy (i.e., elimination of accumulative round-off) and reduce the computatinal load about 30% compared to the current **REAL SCAN** algorithms.

TABLE O-4. SAMPLE 3 BIT, 4 BIT, AND 6 BIT SCAN LINE PSEUDO FRACTION

Exact Increment	3 Bit Scaled	4 Bit Scaled	6 Bit Scaled
1.040657	8	17	67
1.108343	9	18	71
1.180431	9	19	76
1.257207	10	20	80
1.338977	11	21	86
1.426066	11	23	91
1.518819	12	24	97
1.617604	13	26	104
1.722815	14	28	110
1.834869	15	29	117
1.954211	16	31	125

APPENDIX AA

DICOMED START-UP PROCEDURE

Read the "Operation and Programming Manual" for the DICOMED before attempting this start-up routine.

1. Turn the key on the upper left front panel of the DICOMED to the ON position. The DICOMED is now in the operating mode. The following front panel lights should be lit:

   Input: 8-bit
   Output: Normal, Log
   Resolution: High
   Filter: Neutral
   Control: Operate

If no lights have come on, check the main power switch located on the lower rear of the DICOMED chassis. Be sure it is in the ON position. If there are still no lights on, contact Mr. Jerry Diddle in N-74.

2. Loosen the two thumbscrews on the lower front panel and lower it until it stops. This exposes, from left to right, the high voltage switch, a signed three digit display, the exposure trim knob, the exposure calibration button, two test select knobs (S1 and S2), auxiliary inputs, and the prime mode button.

   A. Press the exposure calibration button. It may not light immediately (1-2 second delay). The exposure calibration works in conjunction with the exposure knob on the upper front panel. The exposure calibration for commonly used films are given in Table AA-1. The exposure calibration is the significant value. The exposure value is a suggested value. Adjust upper front panel exposure knob for desired film. Adjust lower panel exposure trim knob until the signed three-digit display shows the desired exposure trim value. The exposure is now calibrated. Press the exposure calibration button again and the light will go out.

   B. Switch the high voltage on. If the high voltage is not on, the film will not be exposed. To prolong the time between major adjustments, turn the high voltage off for ~5 minutes after every 30-40 minutes of operation.

TABLE AA-1. EXPOSURE CALIBRATION

FILM	ASA	EXPOSURE CALIBRATION TOP PANEL	EXPOSURE CALIBRATION LOWER PANEL
POLAROID			
TYPE 52	400	4.6	+.063
TYPE 59	80	10.45	+.400
ECN 5247	100	10.45	+.400
EKTACHROME 200	200	4.25	+.150
KODAK PLUS-X PAN	125	4.25	+.132

C. Now, select the type of film magazine you wish to use. The Polaroid magazine is usually mounted on the DICOMED. The magazine is located under the hood at the top of the DICOMED. If you wish to use the 35mm canister refer to page 12 of the "Operation and Programming Manual" for installation procedures. Pages 11-13 of the "Operation and Programming Manual" describe the loading of film for both types of film.

APPENDIX AB

DICOMED DRIVER ROUTINES

Appendices AC through AV presents our understanding of the function of each of the Dicomed driver routines. Each section of this appendix contains one of Jeff Rubin's subroutines. Each subroutine has a brief description of its purpose, what other subroutines call it, and what other subroutines it calls, and what system functions it uses.

Two functions have been used in these subroutines that are not standard FORTRAN functions, but VAX FORTRAN functions. First, is the used of continutation indicators. In the "VAX-11 FORTRAN IV-PLUS Language Reference Manual", page 1-7, Section 1.3.4 entitled 'continutation fields', these indicators are defined as "A continuation indicator is any character, except zero, or space, in column 6 of a FORTRAN line or any digit, except zero, after the first tab". This is used extentsively in Rubin's subroutines. Secondly, is the INCLUDE statement, which is defined as a statement that specifies "that the contents of a designated file are to be incorporated in the FORTRAN compilation directly following the INCLUDE statement". This is detailed on page 1-9, Section 1.5 of the "VAX-11 FORTRAN IV-PLUS Language Reference Manual" Jeff Rubin generated the Dicomed driver routines in 1979. There has never been any formal documentation received about the routines. This appendix defines the function of each routine.

DEFINITIONS OF SYSTEM SERVICES USED

The Dicomed routines use several system services provided by the VAX 11/780. The definitions are given here. The definitions were obtained from the "VAX11 Software Handbook". The page number after the definition refers to the "VAX11 Software Handbook".

SYS$ASSIGN=ASSIGN I/O CHANNEL:="provides a device with an I/O channel so that input/output operations can be performed on the device..." p.101

SYS$FORCEX=FORCE EXIT:=causes an exit from the specified process. p.111

SYS$SETIMR=SET TIMER:="allows a process to schedule the setting of an event flag". The time may be absolute or delta. p.116

SYS$QIO=QUEUE I/O REQUEST:="initiates an input or output op-

eration by queuing a request to a channel associated with the specific device". Here the device is the Dicomed and its device name is UZA0:. p.102
SYS$QIOW=QUEUE I/O REQUEST AND WAIT FOR EVENT FLAG:= "used when a program must wait for I/O completion". p.102
SYS$WAITFR=WAIT FOR SINGLE EVENT FLAG:="tests a specific event flag and returns immediately, if the event flag is set. Otherwise, the process is placed in a wait state until the event flag is set". p.94.

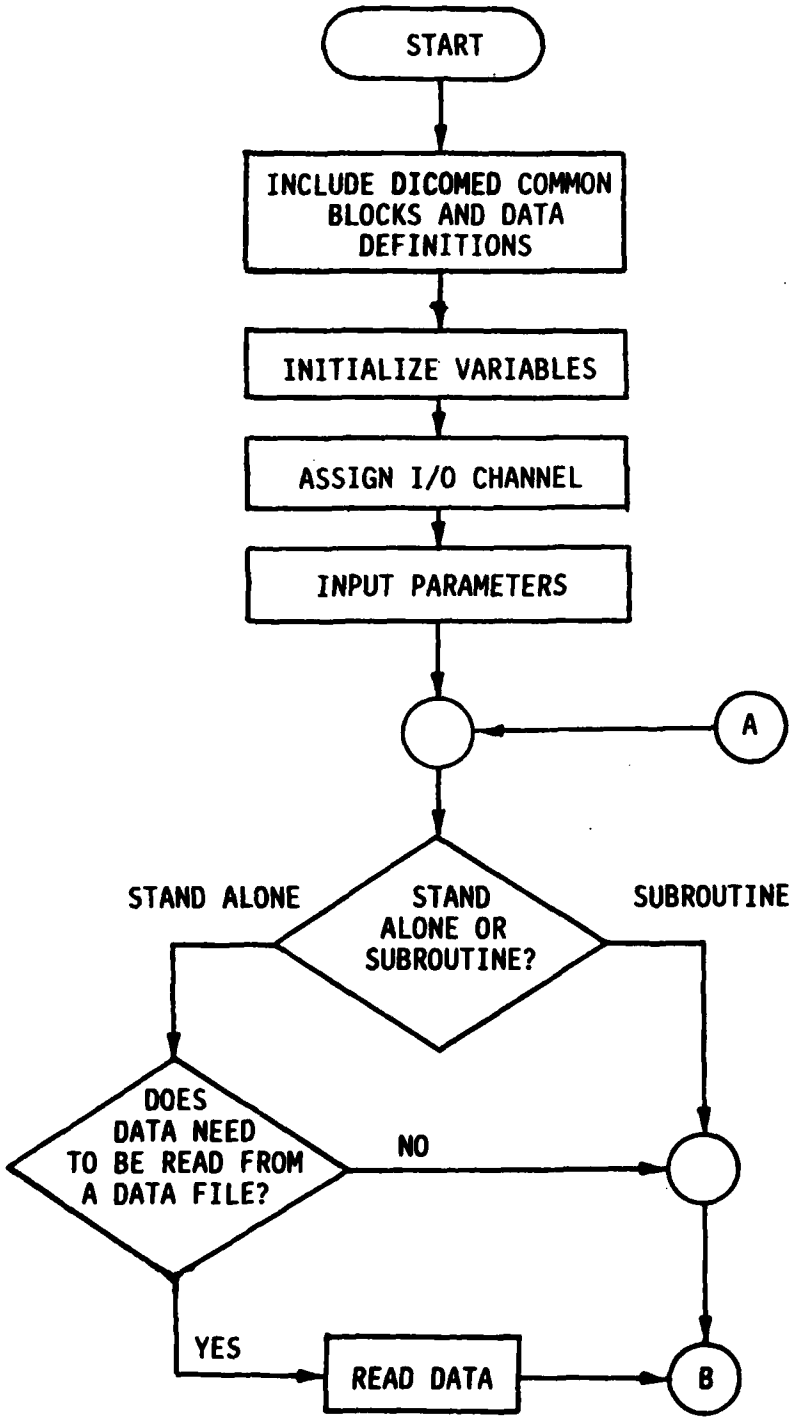


Figure AB-1a. General Flowchart of DICOMED Picture Generation Routines.

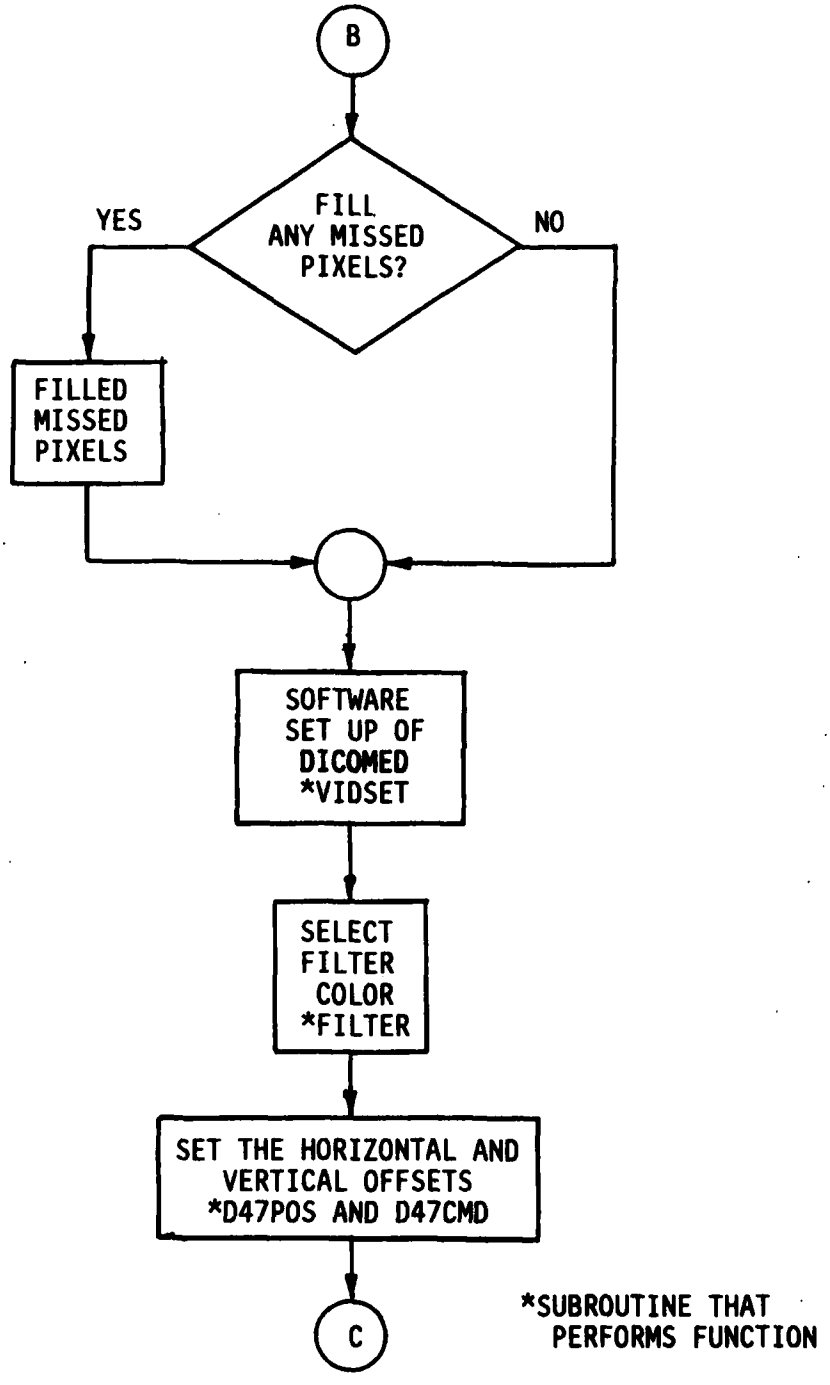


Figure AB-1b. DICOMED Picture Generation Routines.

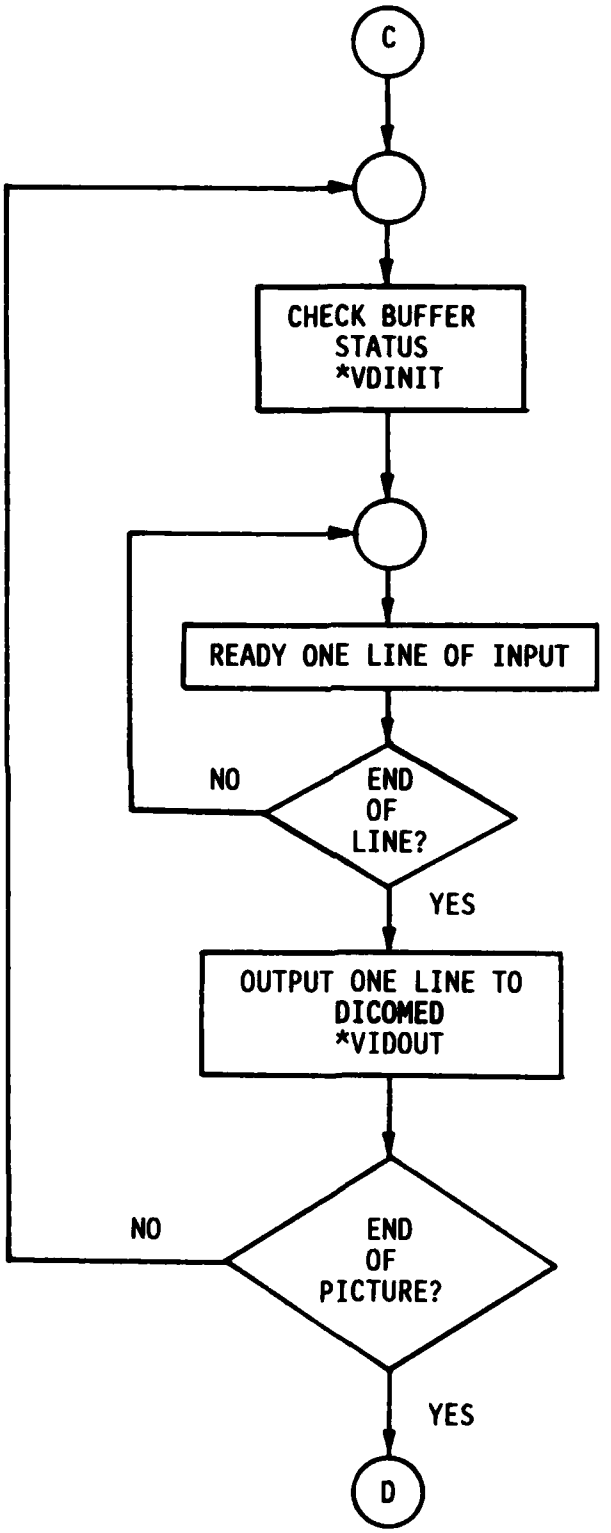


Figure AB-1c. **DICOMED** Picture Generation Routines.

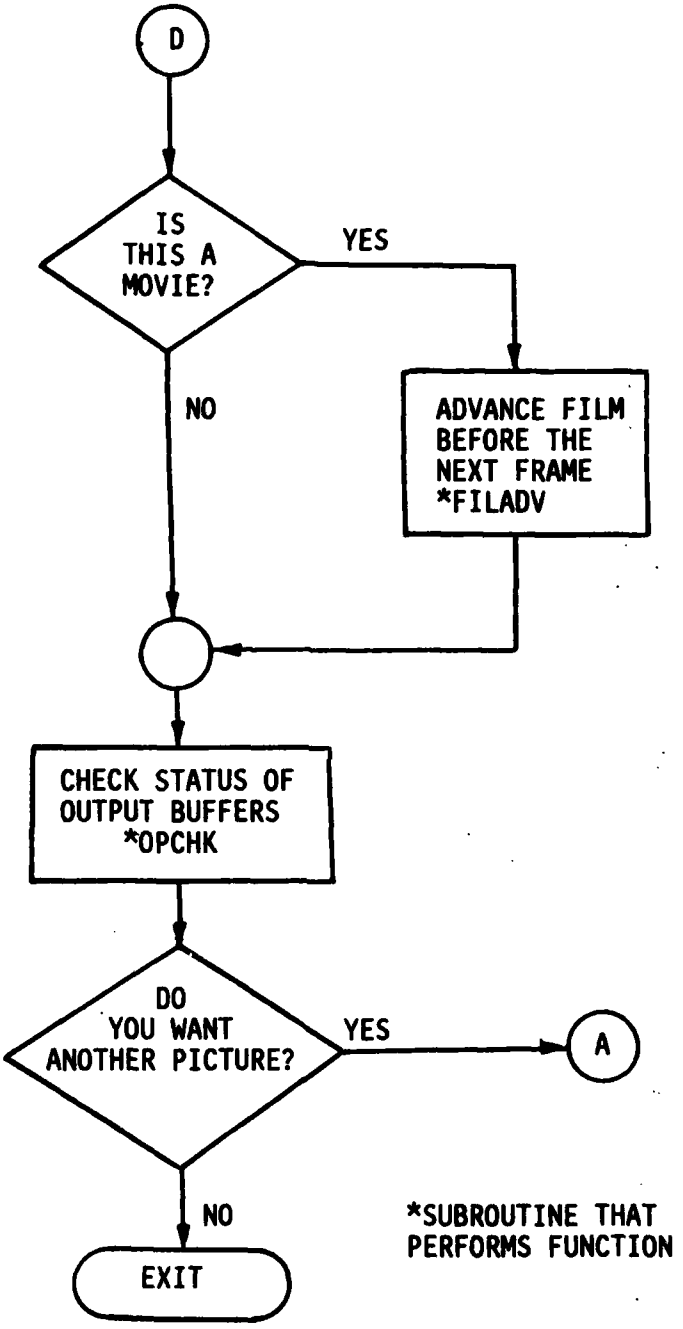


Figure AB-1d. DICOMED Picture Generation Routines.

APPENDIX AC

SUBROUTINE AST = AST.FOR

```
AST is called by:  D470UT
AST calls:         D470UT, ERR
System functions used:  none

      SUBROUTINE AST
C THIS ROUTINE IS EXECUTED WHENEVER THE DICOMED FINISHES
 OUTPUTTING
C  A LINE.
      INCLUDE 'COMMON.FOR'
      INCLUDE 'IO.FOR'
      IF(IOSB(1).NE. SSSNORMAL)CALL ERR(1)
C DO WE NEED TO OUTPUT ANOTHER COPY OF THE LINE?
      OPCNT=OPCNT-1
      IF(OPCNT .LE. 0)GO TO 10
C YES
      CALL D470UT
      RETURN
C NO, WE ARE FINISHED WITH BUF2
10    BUFRE(2)=-1
      RETURN
      END
```

APPENDIX AD

SUBROUTINE BIT10 = BIT10.FOR

BIT10 is called by: D47OUT, D47POS, D47STAT
BIT10 calls: ERR
System functions used: SYSSQIOW

```
        SUBROUTINE BIT10
        INCLUDE 'COMMON.FOR'
        INCLUDE 'IO.FOR'
        ISTAT=SYS$QIOW(,%VAL(ICHAN),%VAL(IO$SETMODE),IOSB,,,,,
        1 WRITMOD,,0,0)
10
        ISTAT=SYS$QIOW(,%VAL(ICHAN),%VAL(IO$SENSEMODE),IOSB,,,,,

        1 RT10,,NSEC,0)
        IF (IOSB(1) .EQ. SS$WASSET)  GOTO 10
        IF(IOSB (1) .NE. SS$NORMAL)CALL ERR(2)
        RETURN
        END
```

APPENDIX AE

COMMON BLOCK = COMMON.FOR

COMMON is the data common to almost all Dicomed routines. COMMON is included in: AST, BIT10, CAMERA, data manipulation routine, D47CMD, D47DCK, D47DUI, D47POS, D47STAT, FILADV, FILSEL, FILTER, OPCHK, READY, VDINIT, VIDCLS, VIDOUT, VIDSET

```
C COMMON.FOR
C COMMON AREAS USED BY CAMERA STATION ROUTINES
        IMPLICIT INTEGER*2 (A-Z)
        BYTE CODE,MODE,BUREQ,BUFRE,BUF1,BUF2,NCOL
        COMMON
 /CTRL/CODE,MODE,EDB(12),IXCSP,NCOL(2),LN1,LN2,LN3,
     1    BUFRE(2),BUREQ(2),IA(2),ITM1(8),MAGZ,MTSTS,MTBCT
        COMMON/WORK/ BUF1(4096)
        DIMENSION BUF(2048)
        EQUIVALENCE (BUF,BUF1)
        COMMON/TMP/ BUF2(8192),BUFSZ,DPCNT
        INTEGER*4 BUFSZ
        COMMON /FIXDT/ IC(54),NELE,NLIN
C THE FOLLOWING EQUIVALENCE ALLOWS US TO ACCESS BOTH BYTES
 OF
C A BYTE ARRAY AS A WORD VALUE.
        EQUIVALENCE (BUFREE,BUFRE),(BUFREQ,BUREQ)
```

APPENDIX AF

SUBROUTINE D47CMD = D47CMD.FOR

D47CMD (D47 Command) has the job of sending the data word, CMDBUF, (Command BUFfer) to the Dicomed.
D47CMD is called by: D47DCK, FILADV, FILSEL, FILTER, VIDCLS, VIDSET
D47CMD calls: BIT10, ERR, READY
System functions used: SYS$QIOW

```
      SUBROUTINE D47CMD(CMDBUF)
C THIS ROUTINE SENDS A COMMAND TO THE DICOMED
      INCLUDE 'COMMON.FOR'
      INCLUDE 'IO.FOR'
C CHECK READY
      CALL READY
C CHECK BIT 10 (I.E. DICOMED READY)
      CALL BIT10
C SET BIT 15 IN THE COMMAND BUFFER
      CMD=IOR(BIT15,CMDBUF)
C SEND THE COMMAND
      ISTAT=SYS$QIOW(,%VAL(ICHAN),%VAL(IOSWRITELBLK),IOSB,
     1 ,,CMD,%VAL(2),MI1,,NSEC,NTRIES)
C IF IOSB NOT NORMAL THEN READY WAS NEVER SET
      IF(IOSB(1).NE.SSSNORMAL)CALL ERR(1)
      RETURN
      END
```

APPENDIX AG

SUBROUTINE D47OUT = D47OUT.FOR

D47OUT sends the data buffer containing a line of data to the Dicomed.
D47OUT is called by: VIDOUT
D47OUT calls: AST, BIT10, READY
System functions used: SYSSQIO

```
      SUBROUTINE D47OUT
C THIS ROUTINE OUTPUTS A BUFFER TO THE DICOMED
      INCLUDE 'COMMON.FOR'
      EXTERNAL AST
      INCLUDE 'IO.FOR'
C CHECK THE READY BIT
      CALL READY
C CHECK BIT 10 (I.E. DICOMED READY)
      CALL BIT10
C SEND A BUFFER
      ISTAT=SYSSQIO(%VAL(1),%VAL(ICHAN),%VAL(IOSWRITELBLK)
     1 ,IOSB,AST,,BUF2,%VAL(BUFSZ),W12I,,0,0)
      RETURN
      END
```

APPENDIX AH

SUBROUTINE D47POS = D47POS.FOR

D47POS sends position data to the Dicomed to allow random positioning.
D47POS is called by: FILTER
D47POS calls: BIT10, ERR, READY
System functions used: SYS$QIOW

```
        SUBROUTINE D47POS(POSITION)
C THIS ROUTINE SENDS POSITION DATA TO THE DICOMED
        INCLUDE 'COMMON.FOR'
        INCLUDE 'IO.FOR'
C CHECK READY BIT
        CALL READY
C CHECK BIT10 (I.E. DICOMED READY)
        CALL BIT10
C SEND POSITION DATA
        ISTAT=SYS$QIOW(,%VAL(ICHAN),%VAL(IO$WRITELBLK),IOSB,
        1 ,,POSITION,%VAL(2),%T2,,0,0)
C IF IOSB NOT NORMAL, THERE WAS AN ERROR
        IF(IOSB(1).NE. SS$NORMAL)CALL ERR(1)
        RETURN
        END
```

APPENDIX AI

SUBROUTINE D47STAT = D47STAT.FOR

```
D47STAT requests the status of the Dicomed.
D47STAT is called by:   D47DCK, FILADV, FILSEL, VIDCLS
D47STAT calls:  BIT10, ERR, READY
System functions used:  SYS$QIOW

        SUBROUTINE D47STAT(STATUS)
C THIS ROUTINE READS THE STATUS OF THE DICOMED AND RETURNS
C THE VALUE IN THE PARAMETER STATUS.
        INCLUDE 'COMMON.FOR'
        INCLUDE 'IO.FOR'
        DATA    MASK/'77777'O/
        STATUS=0
C CHECK THE READY BIT
        CALL READY
C MAKE SURE BIT10 IS SET (I.E. THE DICOMED IS READY)
        CALL BIT10
C CHECK BIT 11 TO SEE IF ANYTHING IS GOING ON
        ISTAT=SYS$QIOW(,%VAL(ICHAN),%VAL(IO$SENSEMODE),
        1 IOSB,,,,,BIT11,,0,0)
        IF(IOSB(1).EQ. SS$WASSET)RETURN
        IF(IOSB(1).NE. SS$NORMAL)CALL ERR(1)
C REQUEST THE DICOMED TO SEND STATUS INFO
        ISTAT=SYS$QIOW(,%VAL(ICHAN),%VAL(IO$READLBLK),IOSB,
        1   ,,STAT,%VAL(2),RD1,,VSEC,NTRIES)
C IF IOSB IS NOT NORMAL, READY WAS NEVER SET
        IF(IOSB(1) .NE. SS$NORMAL)CALL ERR(1)
C MASK OUT BIT 15
        STATUS=IAND(STAT,MASK)
RETURN
END
```

APPENDIX AJ

DATA BLOCK = DEF.FOR

DEF assigns octal values to variables. Further explanation of most of these assignments are given in IMAGE RECORDERS, Models 046 and 047: Operation and Programming Manual, DICOMED Corporation, pages 17-18.

```
C DEF.FOR
C THESE DATA VALUES DEFINE THE DR11B FUNCTION CODES AND
C RETURN VALUES USED BY THE DICOMED.
      DATA RDY/'200'O/          !READY
      DATA BIT10/'2000'O/       !BIT 10
      DATA BIT15/'100000'O/     !BIT 15
      DATA WT1/'404'O/
      DATA WT2I/'514'O/
      DATA CLEAR/0/             !External initialize
      DATA SBBIT/6/             !Select 8-bit
      DATA LOGS/'212'O/         !Linear steps of density
      DATA RES/'16'O,'15'O,'14'O/ !Resolution (see below)
      DATA OFSET/'100'O/        !Default offset if MAGZ=1
      DATA COL/'231'O,'232'O,'233'O/ !Colors (see below)
      DATA NEU/'230'O/          !Neutral
      DATA SOI/1/               !Start Of Input
      DATA HPOS/'10'O/          !Change Horizontal POSition
      DATA WT2/'414'O/
      DATA BIT11/'4000'O/       !BIT 11
      DATA RD1/6/
      DATA FADV/'200'O/         !Film ADVance
      DATA EOFM/'32'O/          !End Of FilM
      DATA IMWT/'204'O/         !IMage waiTing
      DATA WRITMOD/2/
      DATA NORM/'201'O/         !NORMal polarity
      DATA RECQ/'204'O/         !RECord reQuest
      DATA EOI/4/               !End Of Input
      DATA READY/'200'O/        !READY
      DATA VPOS/'11'O/          !Change Vertical POSition
      DATA USEC/2/
      DATA NTRIES/5/
```

RES is defined as follows: '14'O is low; '15'O is medium; '16'O is high. COL is defined as follows: '231'O is red; '232'O is green; '232'O is blue.

APPENDIX AK

SUBROUTINE DELAY = DELAY.FOR

DELAY is wait routine. The number of seconds of delay is sent to DELAY by the calling routine.
DELAY is called by: FILADV, FILSEL, FILTER, OPCHK
DELAY calls: none
System functions used: SYSSSETIMR, SYSSWAITFR

```
      SUBROUTINE DELAY(SECONDS)
      IMPLICIT INTEGER*2 (A-Z)
      INTEGER*4 DELTADRG(2),DELTA(2),I,SYSSSETIMR
      DATA DELTADRG/ -10000000, -1/
      DATA    DELTA(2)/-1/
C COMPUTE THE DELTA TIME REQUESTED
      DELTA(1)=SECONDS*DELTADRG(1)
      I = SYSSSETIMR( %VAL(4),DELTA,,)
      IF ( .NOT. I)  GO TO 10
      CALL  SYSSWAITFR( %VAL(4))
10    CONTINUE
      RETURN
      END
```

APPENDIX AL

SUBROUTINE ERR = ERR.FOR

ERR outputs an error message as determined by the calling routine. A forced exit (SYS$FORCEX(,,)) is executed after the message is output.
ERR is called by: AST, BIT10, D47CMD, D47POS, D47STAT, FILSEL
ERR calls: none
System functions used: SYS$FORCEX

```
        SUBROUTINE   ERR (IERR)
        GO TO  (10,20, 30                )IERR
10      PRINT   810
810     FORMAT  ('0***THE DR11B IS NOT READY***')
        GO TO 100
20      PRINT  820
820     FORMAT ('0*** THE DICOMED IS NOT READY***')

        GO TO 100
30      PRINT  830
830     FORMAT('0***THERE IS A PROBLEM WITH SELECTING THE
 FILTER***')
100     CALL SYS$FORCEX(,,)
        END
```

APPENDIX A4

SUBROUTINE FILADV = FILADV.FOR

FILADV (FILm ADVance) advances 35mm film using a 4 perforation pull-down, according to American Film Academy Standards.
FILADV is called by: VIDCLS
FILADV calls: D47CMD, D47STAT, DELAY
System functions used: none

```
      SUBROUTINE FILADV
      INCLUDE 'COMMON.FOR'
      INCLUDE 'IO.FOR'
      CALL D47CMD(FADV)          !FADV='200'O
      CALL DELAY(5)
      CALL D47STAT(STATUS)
      IF (STATUS .NE. EOFM) RETURN !EOFM='32'O
10    CALL D47CMD(IMWT)          !IMWT='204'O
      CALL D47STAT(STATUS)
      IF(STATUS.NE.RECO) GO TO 10 !RECO='204'O
      RETURN
      END
```

APPENDIX AN

SUBROUTINE FILSEL = FILSEL.FOR

FILSEL (FILter SELect) selects the filter to be used via D47CMD, where COLOR is the command buffer (CMDBUF). COLOR is defined as: red='231'O, green='232'O, blue='233'O.
FILSEL is called by: FILTER
FILSEL calls: D47CMD, D47 STAT, DELAY, ERR
System functions used: none

```
      SUBROUTINE  FILSEL (COLOR)
      INCLUDE  'COMMON.FOR'
      INCLUDE 'IO.FOR'
      CALL D47CMD(COLOR)
      CALL DELAY(5)    !Wait 5 seconds
10    CALL  D47STAT (STATUS)
      IF (STATUS .EQ.0  .OR.STATUS.EQ. REDY) RETURN
      CALL   ERR(3)    !Filter selection problem
      RETURN
      END
```

APPENDIX AO

SUBROUTINE FILTER = FILTER.FOR

FILTER determines which filter has been requested and sends that data to FILSEL. This subroutine will also select the horizontal and vertical offsets at '100,100' octal, if MAGZ equals one. This is not for use with 35mm! See the section entitled Movies is the main body of this report.
FILTER is called by: data manipulation routine
FILTER calls: D47CMD, D47POS, DELAY, FILSEL
System functions used: none

```
        SUBROUTINE FILTER
        INCLUDE 'COMMON.FOR'
        INCLUDE '10.FOR'
10      IF(BUFRE(2) .NE.0 ) GO TO 15
C IF BUF2 IS NOT FREE, WAIT AND TRY AGAIN
        CALL DELAY(2)
        GO TO 10
15      CONTINUE
        IF(BUFRE0.NE.0) GO TO 10
        COLOR=NEU
        IF(NCOL(1).NE.3)GO TO 30
        I=NCOL(2)
        COLOR=  COL(I)
30      CALL FILSEL(COLOR)
        CALL D47CMD(CLEAR)
        CALL D47CMD(SOI)
        IF(MAGZ.EQ.0)RETURN
        CALL  D47CMD (VPOS)
        CALL  D47POS(OFSET)
        CALL  D47CMD(HPOS)
        CALL  D47POS(OFSET)
        RETURN
        END
```

APPENDIX AP

COMMON BLOCK 2 = IO.FOR

IO (Input/Output) is a common block for all the routines that use QIO, or system IOs.
IO is included in: AST, BIT10, CAMERA, data manipulation routine, D47CMD, D47DCK, D47DUT, D47POS, D47STAT, FILADV, FILSEL, FILTER, READY, VIDCLS, VIDSET.

```
C IO.FOR
C COMMON AREAS USED BY CAMERA STATION ROUTINES USING QIO
      COMMON/CHANNEL/ICHAN,IOSB(4),ISTAT,NSEC,NTRIES,MBXCHAN
      INTEGER*1 ICHAN,ISTAT,SYSSQIOW,SYSSQIO,SYSSCREMBX,
     1 MBXCHAN,SWCHAN
      COMMON/BITCO/RDY,BT10,WRITMOD,WI1,BIT15,WT2I,CLEAR,
     1 RES(3),NORM,LOGS,OFSET,COL(3),NEG,SOI,VPOS,HPOS,
     2 BIT11,RD1,READY,FADV,EOFM,EOI,IMNT,RECO,S8BIT,WT2
C DEFINE FUNCTION CODES AND RETURN VALUES
      PARAMETER
     1        IOSREADLBLK      ='21'X,
     2        IOSWRITELBLK     ='20'X,
     3        IOSSETMODE       ='23'X,
     4        IOSWRITEOF       ='28'X,
     5        IOSSENSEMODE     ='27'X,
     6        SSSNORMAL        ='01'X
      PARAMETER
     1        IOSMNOW ='40'X,
     2        IOSREADVBLK      ='31'X,
     3        SSSWASSET        ='09'X,
     4        IOSMWRTATTN      ='100'X
```

APPENDIX AJ

SUBROUTINE OPCHK = OPCHK.FOR

OPCHK (OutPut CHecK) checks to make sure all data has left the output buffers before exiting.
OPCHK is called by: data manipulation routine
OPCHK calls: DELAY, VIDCLS, VIDOUT
System functions used: none

```
      SUBROUTINE      OPCHK
C THIS ROUTINE TAKES SURE THAT ALL OUTPUT IS COMPLETED
C BEFORE EXITING
      INCLUDE 'COMMON1.FOR'
      GO TO 20
C UNDER CERTAIN CONDITIONS, DO AN EOF TO THE MAGTAPE
      CALL MATEF
      RETURN
C DO WE HAVE A BUFFER THAT NEEDS TO BE WRITTEN
20    IF(BUFF(2) .EQ. 0)GO TO 30
C YES, IS BUFFER2 AVAILABLE?
10    IF(BUFFE(2).NE. 0)GO TO 15
C NO, SO WAIT
      CALL DELAY(2)
      GO TO 10
C BUF2 IS AVAILABLE, SO PREPARE TO OUTPUT
15    CONTINUE
      BUFFO(2)=0
      CALL VIDOUT
      GO TO 20
C HAS THE LAST BUFFER WRITE BEEN COMPLETED?
30    IF(BUFFE(2).NE. 0)GO TO 35
C NO, SO WAIT AND TRY AGAIN
      CALL DELAY(2)
      GO TO 30
35    CALL VIDCLS
      RETURN
      END
```

APPENDIX AR

SUBROUTINE READY = READY.FOR

READY checks the dicomed status. If the status is not normal, the error message "The DR11B is not ready" is sent to the operators terminal.
READY is called by: D47CMD, D47OCK, D47OUT, D47POS, D47STAT
READY calls: ERR
System functions used: SYS$QIOW

```
        SUBROUTINE READY
        INCLUDE 'COMMON.FOR'
        INCLUDE 'TO.FOR'
10      ISTAT=SYS$QIOW(,%VAL(ICHAN),%VAL(IO$SENSEMODE),IOSB,,,,,
        1 RDY,,NSEC,0)
        IF (IOSB(1) .EQ. SS$WASSFI)     GOTO 10
        IF (IOSB(1) .NE. SS$NORMAL) CALL ERR(1)
        RETURN
        END
```

APPENDIX AS

SUBROUTINE VDINIT = VDINIT.FOR

VDINIT (VIDeo INITialize) initializes the video output. The output buffers are cleared via VIDOUT.
VDINIT is called by: data manipulation routine
VDINIT calls: VIDOUT
System funtions used: none

```
        SUBROUTINE  VDINIT
        INCLUDE  'COMMON.FOR'
C  VIDEO  O/P INITIALIZE
C
C IS  BUF1  FREE
10      IF (BUFRE(1).NE.0 )GO TO 20
C NO, IS BUF2 FREE . IF NOT, WAIT.
12      IF (BUFRE(2) .NE. 0) GO TO 15
C       CALL DELAY(1)
        GO TO 12
15      CONTINUE
C YES, CLEAR  POSSIBLE REQUEST
        BUREQ(2)=0
        CALL VIDOUT
        GO TO 10
C BUF 1 IS FREE SO RESET
20      BUFRE(1)=0
        RETURN
        END
```

APPENDIX AI

SUBROUTINE VIDCLS = VIDCLS.FOR

VIDCLS (ViDeo CLear Status) sends an EOI (End Of Input) command to the Dicomed via D47CMD. It is also capable of executing a film advance, if MAGZ equals one.
VIDCLS is called by: OPCHK
VIDCLS calls: D47CMD, D47STAT, FILADV
System functions used: none

```
      SUBROUTINE VIDCLS
      INCLUDE 'COMMON.FOR'
      INCLUDE 'IO.FOR'
      CALL D47CMD( EOI)
      CALL D47STAT(STATUS)
      IF (MAGZ .EQ. 0) GO TO 10
      CALL FILADV
10    CALL D47CMD(CLEAR)
      RETURN
      END
```

APPENDIX AU

SUBROUTINE VIDOUT = VIDOUT.FOR

VIDOUT (VIDeo OUTout) outouts a data outfer to the Diconed.
VIDOUT is called by: OPCHK, VDINIT, data manipulation routine
VIDOUT calls: D470UT
System functions used: none

```
      SUBROUTINE VIDOUT
      INCLUDE 'COMMON.FOR'
C IS BUF2 FREE
      IF(BUFRE(2).NE. 0) GO TO 10
C NO, SET BUF2 REQ
      BUREQ(2)=-1
      GO TO 50
C YES, SHOULD WE DUMP TO TAPE
10    CONTINUE
      GO TO 12
      CALL MATWR(MTSTS,BUF,NELE)
      BUFREE = '77577'O
      GO TO 50
C DO NOT DUMP TO TAPE
C MOVE BUF1 TO BUF2, IF LT 512 BYTES,DUPLICATE
12    BUFSZ=0
      BUFRE(2)=0
      OPCNT=2
      IF(NELE.GT.512) OPCNT=1
      DO  40  I= 1,NELE
      BUFSZ=BUFSZ+1
      BUF2(BUFSZ)=BUF1(I)
      IF(NELE.GT. 512) GO TO 40
      BUFSZ=BUFSZ+1
      BUF2(BUFSZ)=BUF1(I)
40    CONTINUE
C SET BUF1 FREE
      BUFRE(1)=-1
      CALL D470UT
50    RETURN
      END
```

APPENDIX AV

SUBROUTINE VIDSET = VIDSET.FOR

VIDSET (VIDeo SET-up) initializes the Dicomed settings.
VIDSET is called by: CAMERA, data manipulation routine
VIDSET calls: D47CMD
System functions used: none

```
      SUBROUTINE VIDSET
      INCLUDE 'COMMON.FOR'
      INCLUDE 'IO.FOR'
      CALL D47CMD(CLEAR)
      CALL D47CMD(SABIT)
      CALL D47CMD(RES(MODE))
      CALL D47CMD(NORM)
      CALL D47CMD(LOGS)
      RETURN
      END
```

APPENDIX BA

APPENDIX B* DESCRIPTION

Appendices BB through BC contain the routines that were used to test the capabilities of the DICOMED D47 Image Recorder.

APPENDIX BB

RANDOM POSITIONING AND

ARBITRARY POSITIONING TESTS USING RANIM.FOR

The DICOMED allows the option of arbitrary positioning. Arbitrary positioning allows picture placement anywhere in the DICOMED field. Two subroutines (D47CMD and D47POS) must be called to alter the starting position from its default location. The default location is (000,000) octal. To change the starting position:

1. D47CMD gets VPOS, the vertical offset. VPOS is defined as 11 octal in DEF.FOR.

2. D47CMD is passed the vertical offset.

3. D47CMD gets HPOS, the horizontal offset. HPOS is defined as 10 octal in DEF.FOR.

4. D47POS is passed the horizontal offset.

Steps 1 and 2 must be performed in order; likewise for steps 3 and 4. However, steps 3 and 4 may be performed prior to steps 1 and 2 with no difference in the result. RRANIM.FOR (RANdom IMage) is a test routine which allows one to place a rectangle at any position within the DICOMED field. The rectangle width and height are input along with the desired positioning, and the resolution. The lines defining the rectangle are four picture elements (pixels) wide.

The following is a listing of the Fortran code for RANIM.FOR.

```
C
C    This program is used to place a rectangle of any desired size in
C the DICOMED field.  The square and the resolution can be arbitrarily
C positioned by terminal input.  The lines on each side of the square
C are four pixels wide.
C    This program was written using Jeff Rubin's DICOMED routines and
C Rick Mitchell's revised DICOMED routines.  Copies of these routines
C are in the P8734 FORTRAN listing, entitled DICOMED PICTURE
C GENERATION TESTS.
C
C                    PROGRAM BY:      G. BECKER
C                    DATE:            11 MAR 1981
C                    PHONE:           677-7621
C                    LAST REVISION:   2 SEP 1981
C    INCLUDE THE DICOMED COMMON, I/O, AND DATA DEFINITIONS.
     INCLUDE '[DICOMED]COMMON.FOR'
     INCLUDE '[DICOMED]IO.FOR'
```

```
      INCLUDE '[DICOMED]DEF.FOR'
      INTEGER*4 SYS$ASSIGN
C
C     INITIALIZATION FOR BW POLAROID
C
      MAGZ=0
      NCOL(1)=0
      NCOL(2)=0
      BUFREE='77577'0
      BUFREQ=0
C
C     INFORMATION INPUT FROM TERMINAL BY USER
C
      NELE=512
      NLIN=512
      MODE=1
      TYPE*, 'ENTER THE PICTURE SIZE (NLIN,NELE)'
      ACCEPT*, NLIN,NELE
C*NOTE:  THE HORIZONTAL AND VERTICAL OFFSETS ARE OCTAL VALUES.
      TYPE*, 'ENTER THE HORIZONTAL AND VERTICAL OFFSET'
      READ (5,20) HOFSET, VOFSET
 20   FORMAT (203)
      TYPE*, 'ENTER RESOLUTION'
      ACCEPT*, MODE
C
c     ASSIGN THE DR11B TO A CHANNEL
C
      ISTAT=SYS$ASSIGN('UZA0:',ICHAN,,)
C
      CALL VIDSET
      CALL FILTER
      CALL D47CMD(VPOS)
      CALL D47POS(VOFSET)
      CALL D47CMD(HPOS)
      CALL D47POS(HOFSET)
      DO 36 J=1,NLIN
              DO 35 I-1,NELE
      IF (((I.LE.4).OR.(I.GT.(NELE-4))).OR.((J.LE.4).OR.
     -  (J.GT.(NLIN-4)))) THEN
                      BUF1(I)=255
           ELSE
           BUFI(I)=0
           ENDIF
 35   CONTINUE
      CALL VDINIT
      CALLVIDOUT
 36   CONTINUE
C
      CALL OPCHK
      STOP
      END
```

To use RANIM.FOR, it must be compiled and then linked with the **DICOMED** driver routines. This is accomplished by:

1) Compiling RANIM.FOR:

```
$ FOR RANIM
```

2) Link it to the **DICOMED** driver routines:

```
$ @RANIM
```

The @ executes a VAX command procedure entitled RANIM.FOR shown here:

```
$LINK/EXE=[P8734]RANIM -

DRAO:[UTILITY.DICOMED]BLKDAT,[P8734]RANIM, -

DRAO:[UTILITY.DICOMED]DICOMED/LIB
```

The result is an executable file entitled RANIM.EXE

3) Run RANIM:

To illustrate the required input parameters, a typical terminal session is shown below. Computer response is in normal type while user response is in boldface.

```
$ FOR RANIM
$ @RANIM
$ RUN RANIM
ENTER THE PICTURE SIZE (NLIN,NELE)
512
512
ENTER THE HORIZONTAL AND VERTICAL OFFSET
010010
ENTER RESOLUTION
1
FORTRAN STOP
$ RUN RANIM
ENTER THE PICTURE SIZE (NLIN,NELE)
217
401
ENTER THE HORIZONTAL AND VERTICAL OFFSET
112145
ENTER RESOLUTION
2
FORTRAN STOP
$ RUN RANIM
ENTER THE PICTURE SIZE (NLIN,NELE)
50
512
ENTER THE HORIZONTAL AND VERTICAL OFFSET
```

```
000240
ENTER RESOLUTION
3
FORTRAN STOP
$ PURGE RANIM.*
```

This session shows three separate executions, or runs, of RANIM.EXE. These three runs were all placed on the same Polaroid photograph. This photograph is shown in Figure BB-1. Table BB-1 describes which rectangles resulted from which set of inputs.

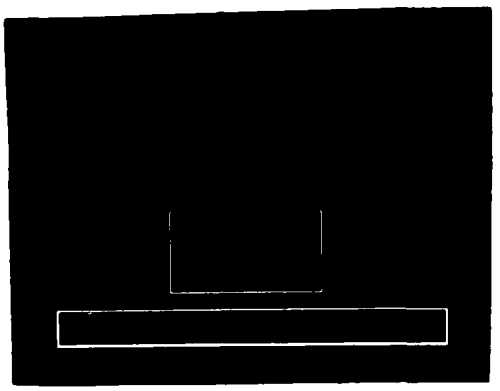


Figure BB-1. Terminal Session Photograph per Table BB-1.

TABLE BB-1. DICOMED POSITIONING IN FIGURE BB-1

RECTANGLE	RESOLUTION	DIMENSIONS H X W	OFFSETS HORIZ.	VERT.
TOP RUN 1	1=HIGH	512 X 512	010	010
CENTER RUN 2	2=MED.	217 X 401	112	145
BOTTOM RUN 3	3=LOW	50 X 512	000	240

***NOTE: Leading zeros are significant on the offset inputs illustrated in Table BB-1 for the typical terminal session shown above.

The **DICOMED** field was 'mapped' using RANIM. The map is shown in Figure BB-2. The Polaroid tab is also shown in Figure BB-2. It is the edge of the Polaroid print without 90 degree corners. This tab makes a handy reference. Holding the photograph with the tab at the top, the location (000,000) is at the top left corner, (000,377) is the bottom left, (377,000) is the top right, and (377,377) is the bottom left. Each square is 40 by 40. All locations and dimensions here are octal values.

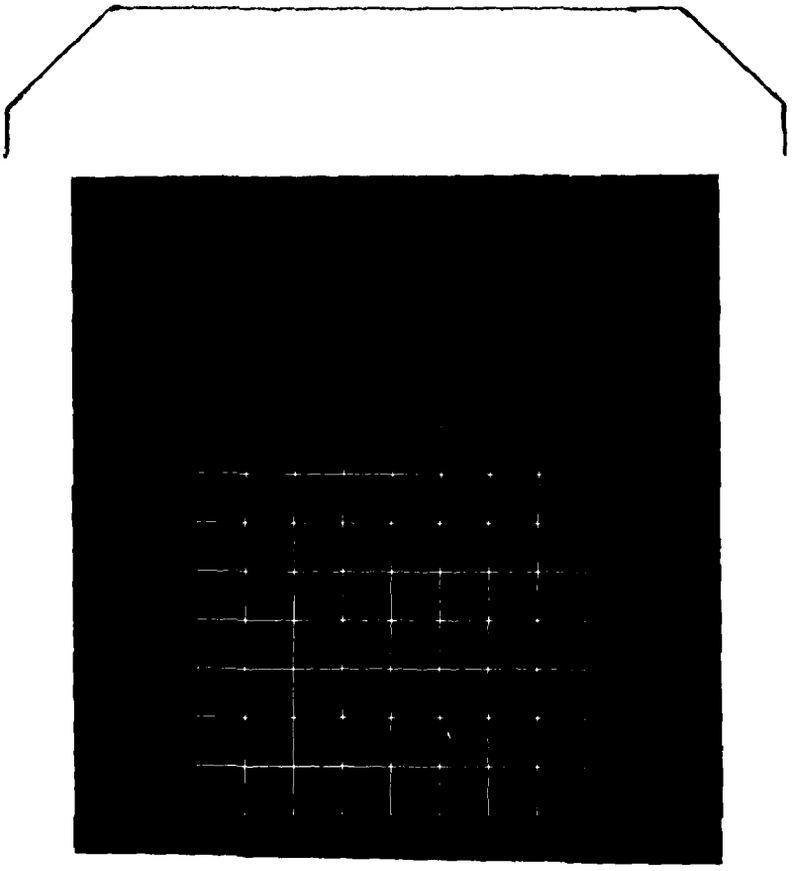


Figure BB-2. Map of the **DICOMED** Image Field.

APPENDIX BC

COLOR TESTS USING SRCAMSA

SRCAMSA is a test routine that allows the generation of various color tests. One purpose of SRCAMSA is to test exposures on different types of color film for color correctness. The second purpose of SRCAMSA is to generate different color combinations in an effort to create colors other than red, blue, and green. SRCAMSA is divided into three sections. Section one creates color bars shown in Figure BC-1. Section two creates a color triangle shown in Figure BC-2. The last section is the color gradient section. Its output is shown in Figure BC-3.

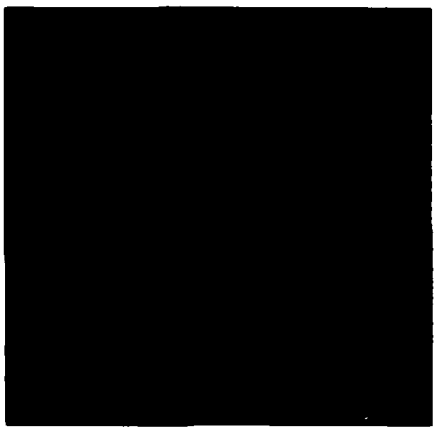


Figure BC-1. Output of the Color Bars Section of SRCAMSA.

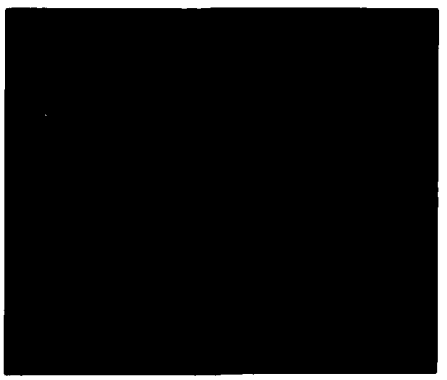


Figure BC-2. Output of the Color Triangle Section of SRCAMSA.

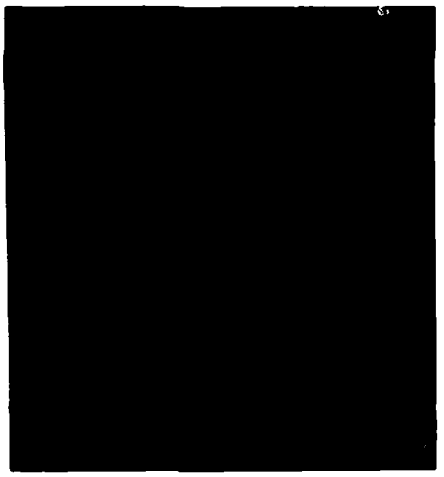


Figure BC-3. Output from the Color Gradient Section of SRCAMSA.

A listing of SRCAMSA.FOR is given here:

```
C
C       SRCAMSA.FOR
C
C       ALTERED FROM GBCAMSA.FOR BY: S. M. RICHIE
C       PHONE: 677-7621
C       DATE:  30 MAY 81
C
C       THIS IS A GENERAL PURPOSE DICOMED ROUTINE WHICH ALLOWS
C FOR THE INPUT OF:
C                1) FILM TYPE,
C                2) PICTURE SIZE,
C                3) RESOLUTION, AND
C                4) HORIZONTAL AND VERTICAL OFFSET.
C
        INCLUDE 'DRA0:[UTILITY.DICOMED]COMMON.FOR'
        INCLUDE 'DRA0:[UTILITY.DICOMED]DEF.FOR'
        INCLUDE 'DRA0:[UTILITY.DICOMED]IO.FOR'
        REAL FJCOM,DELI,DELJ,DIST
        INTEGER*4 SYSSASSIGN
C*******INITIALIZATION OF VARIABLES*******************
        HOFSET=0
        VOFSET=0
        MAGZ=0
        NCOL(1)=0
        NCOL(2)=0
C       ASSIGN DR11B TO A CHANNEL
        ISTAT=SYSSASSIGN('UZA0:',ICHAN,,)
C       IF DR11B IS NOT ASSIGNED WE CANNOT CONTINUE
        IF (.NOT.ISTAT) THEN
                PRINT 10
 10     FORMAT ('0D R 1 1 B   C A N N O T   B E   A S S I G N E D
     -  T O   A   C H A N N E L***')
                STOP
        ENDIF
C*******END INITIALIZATION*******************************
        BUFREF='77577'O
        BUFREJ=0
        TYPE*, '  '
        TYPE*, 'GENERAL DICOMED ROUTINE'
        TYPE*, '  '
C*******INPUT FROM TERMINAL*******************************
100     TYPE*,'ENTER OUTPUT SOURCE: 1=COLOR BARS'
        TYPE*,'                     2=COLOR TRIANGLE'
        TYPE*,'                     3=COLOR GRADIENT'
        TYPE*,'                     4=FORTRAN STOP'
        ACCEPT*,ISOR
        IF(ISOR.EQ.4) STOP
        TYPE*,'ENTER THE NUMBER OF PASSES OF THE SOURCE'
        ACCEPT*, IPASS
        TYPE*, 'ENTER THE PICTURE SIZE (NLIN,NELE).'
```

```
      ACCEPT*, NLIN,NELE
      NLIN2=NLIN/2
      TYPE*, 'ENTER TYPE OF FILM: POLAROID=0; 35mm=1.'
      ACCEPT*, IMOV
      TYPE*, 'ENTER DESIRED RESOLUTION: POLAROID=3; 35mm=1.'
      ACCEPT*, MODE
      IF (IMOV.EQ.1) THEN
            TYPE*, 'ENTER THE HORIZONTAL AND VERTICAL
     *              OFFSETS:'
            TYPE*, '        POLAROID=000,000; 35mm=140,145.'
            READ (5,30) HOFSET,VOFSET
30          FORMAT (03,1X,03)
      ENDIF
C*******END INPUT FROM TERMINAL************************
C     VIDSET CALL D47CMD TO:
C           1) CLEAR THE COMMAND BUFFER,
C           2) SET THE DATA SIZE TO 8 BITS,
C           3) SET THE RESOLUTION,
C           4) SET THE POLARITY TO NORMAL, AND
C           5) SET LINEAR STEPS IN DENSITY.
C     THE HORIZONTAL AND VERTICAL OFFSETS ARE GIVEN TO
C     THE DICOMED.
C
      GOTO(110,120,130) ISOR
      TYPE*,'ERROR IN SOURCE FLAG'
      GOTO 100
C
C
110   TYPE*,'WORKING ON COLOR BARS'
      DO 200 K=1,IPASS
C
      CALL VIDSET
C
      NCOL(1)=3
      NCOL(2)=1
C
      CALL FILTER
      CALL D47CMD(HPOS)
      CALL D47POS(HOFSET)
      CALL D47CMD(VPOS)
      CALL D47POS(VOFSET)
C     PAINT RED FILTER ACTIVITY ON COLUMNS 1-170
      DO 20 I = 1,NLIN
      CALL VDINIT
      DO 21 J=1,512
      BUF1(J)=0
      IF (J.LE.170) BUF1(J)=-1
21    CONTINUE
      CALL VIDOUT
20    CONTINUE
      CALL OPCHK
```

```
C       PAINT BLUE FILTER ACTIVITY ON COLUMNS 171-340
        CALL VIDSET
        NCOL(2)=3
        CALL FILTER
        CALL D47CMD(HPOS)
        CALL D47POS(HOFSET)
        CALL D47CMD(VPOS)
        CALL D47POS(VOFSET)
        DO 32 I=1,NLIN
        CALL VDINIT
        DO 31 J=1,512
        BUF1(J)=0
        IF (J.GT.170.AND.J.LE.340) BUF1(J)=-1
 31     CONTINUE
        CALL VIDOUT
 32     CONTINUE
        CALL DPCHK
C
C       PAINT GREEN FILTER ACTIVITY ON COLUMNS 341-512
        CALL VIDSET
        NCOL(2)=2
        CALL FILTER
        CALL D47CMD(HPOS)
        CALL D47POS(HOFSET)
        CALL D47CMD(VPOS)
        CALL D47POS(VOFSET)
        DO 42 I=1,NLIN
        CALL VDINIT
        DO 41 J=1,512
        BUF1(J)=0
        IF (J.GT.340) BUF1(J)=-1

 41     CONTINUE
        CALL VIDOUT
 42     CONTINUE
        CALL DPCHK
        TYPE*,'PASS',K,'COMPLETED'
200     CONTINUE
        GOTO 500
C
120     TYPE*, '    '
        TYPE*, 'PAINT RED? ... 1=YES,0=NO'
        ACCEPT*,  IPRED
        TYPE*, 'PAINT BLUE? ... 1=YES,0=NO'
        ACCEPT*,  IPBLUE
        TYPE*, 'PAINT GREEN? ... 1=YES,0=NO'
        ACCEPT*,  IPGRN
        TYPE*, 'COLOR:MAX ON CORNER=1,MIN ON CORNER=0'
        ACCEPT*,  ICORN
C
        TYPE*,'WORKING ON COLOR TRIANGLE'
```

```
      DO 9999 IREPET = 1,IPASS
C
      IF(IPRED.EQ.1)  THEN
C     PAINT RED STUFF
      CALL VIDSET
      NCOL(1) = 3
      NCOL(2) = 1
      CALL FILTER
      CALL D47CMD(HPOS)
      CALL D47POS(HOFSET)
      CALL D47CMD(VPOS)
      CALL D47POS(VOFSET)
           IREF = 1
           JREF = 1
      FJLOW = 0.0
      DO 50 I = 1,NLIN
      CALL VDINIT
      FJLOW = FJLOW + (256./445.)
      JLOW = 1 + FJLOW
      JHIGH = 513 - JLOW
      IF(I.GT.445)  JLOW = 10000
           DO 51 J = 1,512
           IF(J.LT.JLOW)  GO TO 52
           IF(J.GT.JHIGH) GO TO 52
                DELI =  I - IREF
                DELJ = J - JREF
                DIST = SQRT(DELI*DELI + DELJ*DELJ)
                IDIST = DIST
                COLOR = IDIST/2
                IF(ICORN.EQ.1) COLOR = 256 - IDIST/2
                IF(COLOR.LT.0) COLOR = 0
                IF(COLOR.GT.255) COLOR=255
                IF(COLOR.LE.127) BUF1(J) = COLOR
                IF(COLOR.GT.127) BUF1(J) = COLOR - 256
                GO TO 51
52    BUF1(J) = 0
51    CONTINUE
      CALL VIDOUT
50    CONTINUE
      CALL OPCHK
     ENDIF
C
     IF(IPBLUE.EQ.1)  THEN
C     PAINT BLUE STUFF
      CALL VIDSET
      NCOL(1) = 3
      NCOL(2) = 3
      CALL FILTER
      CALL D47CMD(HPOS)
      CALL D47POS(HOFSET)
      CALL D47CMD(VPOS)
```

```
      CALL D47POS(VOFSET)
           IREF = 1
           JREF = 512
      FJLOW = 0.0
      DO 60 I = 1,NLIN
      CALL VDINIT
      FJLOW = FJLOW + (256./445.)
      JLOW = 1. + FJLOW
      JHIGH = 513 - JLOW
      IF(I.GT.445) JLOW = 10000
           DO 61 J = 1,512
           IF(J.LT.JLOW)  GO TO 62
           IF(J.GT.JHIGH) GO TO 62
           DELI = I - IREF
           DELJ = J - JREF
           DIST = SQRT(DELI*DELI+DELJ*DELJ)
           IDIST = DIST
           COLOR = IDIST/2
          IF(ICORN.EQ.1) COLOR = 256 - IDIST/2
           IF(COLOR.LT.0) COLOR=0
           IF(COLOR.GT.255) COLOR=255
           IF(COLOR.LE.127) BUF1(J)=COLOR
           IF(COLOR.GT.127) BUF1(J)=COLOR - 256
           GO TO 61
62         BUF1(J) = 0
61    CONTINUE
      CALL VIDOUT
60    CONTINUE
      CALL DFCHK
      ENDIF
C
     IF(IPGRN.EQ.1) THEN
C     PAINT GREEN STUFF
C
      CALL VIDSET
     NCOL(1) = 3
      NCOL(2) = 2
      CALL FILTER
      CALL D47CMD(HPOS)
      CALL D47POS(HOFSET)
      CALL D47CMD(VPOS)
      CALL D47POS(VOFSET)
           IREF = 445
           JREF = 256
           FJLOW = 0.0
      DO 70 I = 1,NLIN
      CALL VDINIT
      FJLOW = FJLOW + (256./445.)
      JLOW = 1. + FJLOW
      JHIGH = 513 - JLOW
      IF(I.GT.445)  JLOW = 10000
```

```
        DO 71 J = 1, 512
        IF(J.LT.JLOW)  GO TO 72
        IF(J.GT.JHIGH)  GO TO 72
        DELI = I - IREF
        DELJ = J - JREF
        DIST = SQRT(DELI*DELI + DELJ*DELJ)
        IDIST = DIST
        COLOR = IDIST/2
       IF(ICORN.EQ.1) COLOR = 256 - IDIST/2
        IF(COLOR.LT.0) COLOR = 0
        IF(COLOR.GT.255) COLOR=255
        IF(COLOR.LT.127) BUF1(J) = COLOR
        IF(COLOR.GT.127) BUF1(J) = COLOR - 256
        GOTO 71
72      BUF1(J) = 0
71      CONTINUE
        CALL VIDOUT
70      CONTINUE
        CALL OPCHK
       ENDIF
C
        TYPE*,'PASS',IREPET,'COMPLETED'
9999    CONTINUE
        GOTO 500
130     TYPE*,  'INITLIZE COLOR:  NEUTRAL=0, COLOR=3'
        ACCEPT*, NCOL(1)
        TYPE*,  'ENTER COLOR:  NEU=0, RED=1, GREEN=2, BLUE=3'
        ACCEPT*, NCOL(2)
        TYPE*,  'ENTER MAX,MIN COLOR LIMITS'
        ACCEPT*, IMX,IMN
        TYPE*,'ENTER VARIATION AXIS: 0=X,1=Y'
        ACCEPT*,IAV
        TYPE*,'WORKING ON COLOR GRADIENT'
C
        DO 8888 K=1,IPASS
C
        CALL VIDSET
C
C
        CALL FILTER
        CALL D47CMD(HPOS)
        CALL D47POS(HOFSET)
        CALL D47CMD(VPOS)
        CALL D47POS(VOFSET)
C
        DO 7775 I=1,NLIN
        CALL VDINIT
        DO 7777 J=1,NELE
        IVATST=J
        IF(IAV.EQ.0) IVATST=I
        IR=((IMN-IMX)/511.)*(IVATST-1) + IMX
```

```
      BUF1(J)=IR-256*(IR/128)
7777  CONTINUE
      CALL VIDOUT
7775  CONTINUE
      TYPE*,'PASS',K,'COMPLETED'
      CALL OPC4K
8888  CONTINUE
      GOTO 500
C
500   IF (IMOV.EQ.1) THEN
            TYPE*, 'DO YOU NEED A FILM ADVANCE? YES=1'
            ACCEPT*, IADV
            IF (IADV.EQ.1) CALL FILADV
      ENDIF
      IF(ISOR.EQ.1) THEN
      TYPE*,'INPUT DATA FOR COLOR BARS FOLLOWS:'
      TYPE*,'NUMBER OF PASSES COMPLETED IS',IPASS
      TYPE*,'NLIN=',NLIN
      TYPE*,'NELE=',NELE
      TYPE*,'FILM TYPE (IMOV)=',IMOV
      TYPE*,'RESOLUTION (MODE)=',MODE
      TYPE*,'HORZ OFFSET=',HOFSET
      TYPE*,'VERT OFFSET=',VOFSET
      WRITE(15,*) 'INPUT DATA FOR COLOR BARS FOLLOWS:'
      WRITE(15,*) 'NUMBER OF PASSES COMPLETED IS',IPASS
      WRITE(15,*) 'NLIN=',NLIN
      WRITE(15,*) 'NELE=',NELE
      WRITE(15,*) 'FILM TYPE (IMOV)=',IMOV
      WRITE(15,*) 'RESOLUTION (MODE)=',MODE
      WRITE(15,*) 'HORZ OFFSET=',HOFSET
      WRITE(15,*) 'VERT OFFSET=',VOFSET
      ENDIF
      IF(ISOR.EQ.2) THEN
      TYPE*,'INPUT DATA FOR COLOR TRIANGLE FOLLOWS:'
      TYPE*,'NUMBER OF PASSES COMPLETED IS',IPASS
      TYPE*,'NLIN=',NLIN
      TYPE*,'NELE=',NELE
      TYPE*,'FILM TYPE (IMOV)=',IMOV
      TYPE*,'RESOLUTION (MODE)=',MODE
      TYPE*,'HORZ OFFSET=',HOFSET
      TYPE*,'VERT OFFSET=',VOFSET
      TYPE*,'RED OUTPUT (IPRED)=',IPRED
      TYPE*,'BLUE OUTPUT (IPBLUE)=',IPBLUE
      TYPE*,'GREEN OUTPUT (IPGRN)=',IPGRN
      TYPE*,'MAX OR MIN ON CORNER (ICORN)=',ICORN
      WRITE(15,*) 'INPUT DATA FOR COLOR BARS FOLLOWS:'
      WRITE(15,*) 'NUMBER OF PASSES COMPLETED IS',IPASS
      WRITE(15,*) 'NLIN=',NLIN
      WRITE(15,*) 'NELE=',NELE
      WRITE(15,*) 'FILM TYPE (IMOV)=',IMOV
      WRITE(15,*) 'RESOLUTION (MODE)=',MODE
```

```
      WRITE(15,*) 'HORZ OFFSET=',HOFSET
      WRITE(15,*) 'VERT OFFSET=',VOFSET
      WRITE(15,*) 'RED OUTPUT (IPRED)=',IPRED
      WRITE(15,*) 'BLUE OUTPUT (IPBLUE)=',IPBLUE
      WRITE(15,*) 'GREEN OUTPUT (IPGRN)=',IPGRN
      WRITE(15,*) 'MAX OR MIN ON CORNER (ICORN)=',ICORN
      ENDIF
      IF(ISOR.EQ.3) THEN
      TYPE*,'INPUT DATA FOR COLOR GRADIENT FOLLOWS:'
      TYPE*,'NUMBER OF PASSES COMPLETED IS',IPASS
      TYPE*,'NLIN=',NLIN
      TYPE*,'NELE=',NELE
      TYPE*,'FILM TYPE (IMOV)=',IMOV
      TYPE*,'RESOLUTION (MODE)=',MODE
      TYPE*,'HORZ OFFSET=',HOFSET
      TYPE*,'VERT OFFSET=',VOFSET
      TYPE*,'COLOR FOR GRADIENT (NCOL(1),NCOL(2))=',
     *      NCOL(1),NCOL(2)
      TYPE*,'MAX,MIN COLOR LIMITS =',IMX,IMN
      WRITE(15,*) 'INPUT DATA FOR COLOR GRADIENT FOLLOWS:'
      WRITE(15,*) 'NUMBER OF PASSES COMPLETED IS',IPASS
      WRITE(15,*) 'NLIN=',NLIN
      WRITE(15,*) 'NELE=',NELE
      WRITE(15,*) 'FILM TYPE (IMOV)=',IMOV
      WRITE(15,*) 'RESOLUTION (MODE)=',MODE
      WRITE(15,*) 'HORZ OFFSET=',HOFSET
      WRITE(15,*) 'VERT OFFSET=',VOFSET
      WRITE(15,*) 'COLOR FOR GRADIENT (NCOL(1),NCOL(2))=',
     *      NCOL(1),NCOL(2)
      WRITE(15,*) 'MAX,MIN COLOR LIMITS =',IMX,IMN
      ENDIF
      TYPE*, '   '
      TYPE*, '   '
      TYPE*,   '   '
      WRITE(15,*) '   '
      WRITE(15,*) '   '
      WRITE(15,*)   '   '
      GOTO 100
 40   STOP
      END
```

To use SRCAMSA.FOR:

1) Compile SRCAMSA.FOR:

```
$ FOR SRCAMSA
```

2) Link SRCAMSA to the Dicomed routines:

```
$ @SRCAMSA
```

The @ executes a VAX command file entitled SRCAMSA.COM:

```
$LINK/EXE=[P8734]SRCAMSA -
DIC:BLKDAT,[P8734]SRCAMSA, -
DIC:DICOMED/LIB
```

3) Run SRCAMSA:

```
$RUN SRCAMSA
```

APPENDIX CA

APPENDIX C* DESCRIPTION

Appendices CA through CE contain the four picture generation routines described in Section VI of the main report.

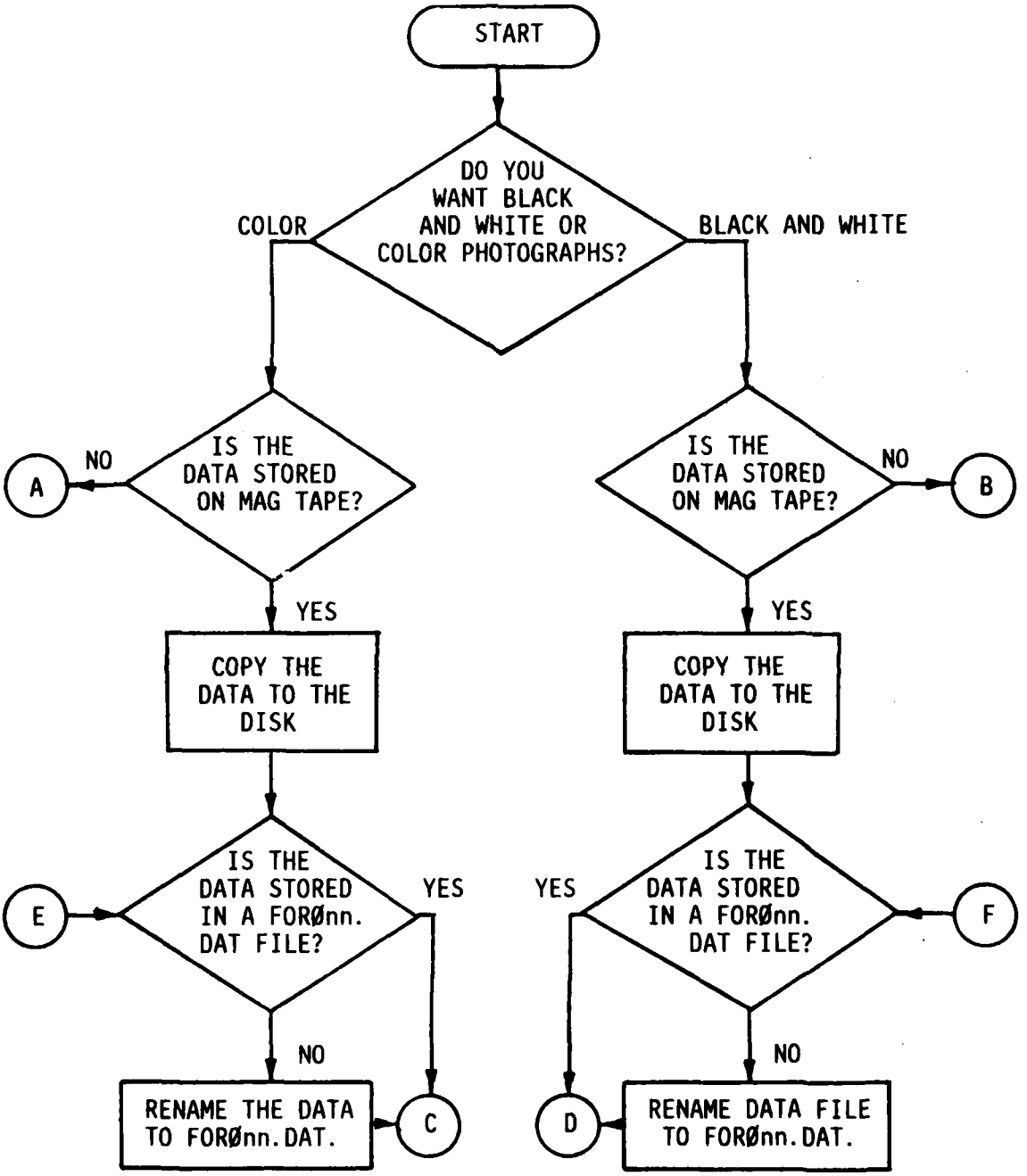


Figure CA-1a. Flowchart for DICOMED Appendix.

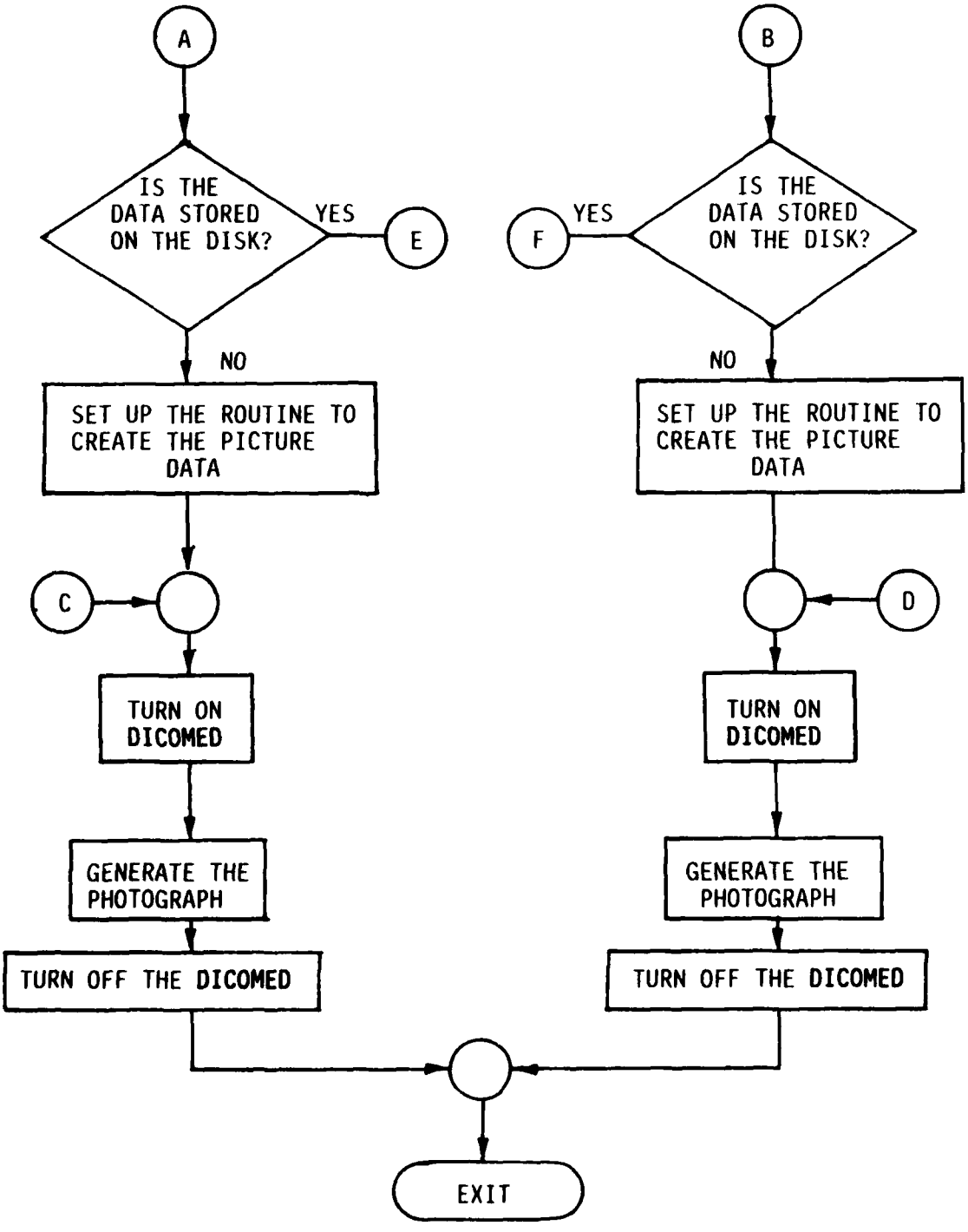


Figure CA-1b. Flowchart for DICOMED Appendix.

APPENDIX C4

DICBWSA.FOR

```
C       DICBWSA.FOR
C       DICOMED Black and White Stand Alone
C
C       PROGRAM BY:     G. BECKER
C       PHONE:          577-7621
C       DATE:           6 AUG 1981
C
C THIS IS A GENERAL BLACK AND WHITE STAND ALONE DICOMED ROUTINE
C FOR USE WITH POLAROID OR 35mm BLACK AND WHITE FILM.
C
C THE REQUIREMENTS OF THE PROGRAM ARE:
C               1) THE DATA HAS BEEN EXTERNALLY CREATED AND WILL
C BE READ INTO THIS PROGRAM,
C               2) THE FILM IS B/W.
C
        INCLUDE 'DRA0:[UTILITY.DICOMED]COMMON.FOR'
        INCLUDE 'DRA0:[UTILITY.DICOMED]DEF.FOR'
        INCLUDE 'DRA0:[UTILITY.DICOMED]IO.FOR'
        INTEGER*1 SYS$ASSIGN
        INTEGER*2 ITEMBUF(512,512),INI2(512,256)
        BYTE ICHT(512,512)
        EQUIVALENCE (INT2,ICHT)
C*******INITIALIZATION OF VARIABLES*********************
        HOFSET=0
        VOFSET=0
        MAGZ=0
        NCOL(1)=0    !BLACK AND WHITE PART 1
        NCOL(2)=0    !BLACK AND WHITE PART 2
        NLIN=512
        NELE=512
        NGIN2=256
        BUFREF='7577'O
        BUFREQ=0
C       ASSIGN DR11W TO A CHANNEL
        ISTAT=SYS$ASSIGN('DZA0:',ICHAN,,)
C       IF DR11W IS NOT ASSIGNED WE CANNOT CONTINUE
        IF (.NOT.ISTAT) THEN
                PRINT 10
```

```
 10     FORMAT ('OD R 1 1 B   C A N N O T   B E   A S S I G N E D
     -  T O   A   C H A N N E L***')
            GO TO 40
        ENDIF
C*******END INITIALIZATION*******************************
        TYPE*, '  '
    TYPE*, 'GENERAL STAND ALONE BLACK AND WHITE DICOMED ROUTINE'
        TYPE*, '  '
C*******INPUT FROM TERMINAL*****************************
        TYPE*, 'THE PICTURE DIMENSIONS ARE SET AT 512,512'
        TYPE*, 'ENTER TYPE OF FILM: POLAROID=0; 35mm=1.'
        ACCEPT*, IMOV
        TYPE*, 'DO YOU WISH TO FILTER THE DATA? (YES=1)'
        ACCEPT*, IFLT
        IF (IMOV.EQ.1) THEN
                TYPE*, 'THE HORIZONTAL OFFSET IS 140 OCTAL.'
                HOFSET=96                !140 OCTAL
                TYPE*, 'THE VERTICAL OFFSET IS 145 OCTAL.'
                VOFSET=101               !145 OCTAL
                TYPE*, 'THE RESOLUTION IS SET AT 1.'
                MODE=1
        ELSE
                TYPE*, 'THE RESOLUTION IS SET AT 3.'
                MODE=3
        ENDIF
 20     TYPE*,'  IS A READ REQUIRED?(YES=1)'
        ACCEPT*, IRD
        IF (IRD.EQ.1) THEN
          TYPE*, ' ENTER THE FILE YOU WANT READ ITEMBUF '
          ACCEPT*, IFILE
          TYPE*, '   READING ITEMBUF FROM FILE  ', IFILE
          READ (IFILE,1000) ((ITEMBUF(I,J),I=1,NELE),J=1,NLIN)
        IF (IFLT.EQ.1) THEN
          TYPE*, '   READING INT2 FROM FILE  ', IFILE
          READ (IFILE,1000) ((INT2(I,J),I=1,NELE),J=1,NLIN2)
        ENDIF
 1000   FORMAT (66A2)
        ENDIF
C*******END INPUT FROM TERMINAL**************************
        IF (IFLT.EQ.1) THEN
C       SCENE IS FILTERED TO FILL ANY MISSED PIXELS
                TYPE*,'FILLING ANY MISSED PIXELS'
                DO 111 IYBS=NLIN,1,-1
                DO 222 IXBS=NELE,1,-1
                IF(ICNT(IXBS,IYBS).NE.0) GOTO 220
                JCOUNT=0
                JSUM=0
                DO 300 K=IYBS-1,IYBS+1
                        IF((K.LT.1).OR.(K.GT.NLIN)) GOTO 300
                DO 440 L=IXBS-1,IXBS+1
                        IF((L.LT.1).OR.(L.GT.NELE)) GOTO 440
```

```
                        IF(ICNT(L,K).EQ.0) GO TO 440
                                JCOUNT=JCOUNT+1
                                JSUM=JSUM+ITEMBUF(L,K)
 440            CONTINUE
 300            CONTINUE
                IF(JCOUNT.EQ.0) THEN
                TYPE*,'MISSED PIXEL AT IXBS,IYBS=',IXBS,IYBS
                 ELSE
                ITEMBUF(IXBS,IYBS)=JSUM/JCOUNT
                 ENDIF
                ICNT(IXBS,IYBS)=-1
 220            CONTINUE
 222            CONTINUE
 111            CONTINUE
        ENDIF
**************DATA IS SENT TO THE DICOMED**********
        TYPE*,'  SENDING DATA TO THE DICOMED'
C       VIDSET CALLS D47CMD TO:
C               1) CLEAR THE COMMAND BUFFER,
C               2) SET THE DATA SIZE TO 8 BITS,
C               3) SET THE RESOLUTION,
C               4) SET THE POLARITY TO NORMAL, AND
C               5) SET LINEAR STEPS IN DENSITY.
        CALL VIDSET
C       FILTER SELECTS THE FILTER TO BE USED VIA FILSEL
        CALL FILTER
C THE HORIZONTAL AND VERTICAL OFFSETS ARE GIVEN TO THE
C DICOMED.
        CALL D47CMD(HPOS)
        CALL D47POS(HOFSET)
        CALL D47CMD(VPOS)
        CALL D47POS(VOFSET)
        DO 444 IXBS=NLIN,1,-1
C       VDINIT CHECKS THE STATUS OF THE DICOMED BUFFERS.
        CALL VDINIT
                DO 555 IYBS=NELE,1,-1
                 ISHIFT=256*(ITEMBUF(IXBS,IYBS)/128)
                 BUF1(IYBS)=ITEMBUF(IXBS,IYBS)-ISHIFT
 555            CONTINUE
        CALL VIDOUT
 444    CONTINUE
        IF (IMOV.EQ.1) THEN
                CALL FILADV
        ENDIF
        TYPE*, 'THE END'
        CALL ORCHK
        TYPE*,'D) YOU HAVE TO DO ANOTHER PICTURE?(YES=1)'
        ACCEPT*, IAN
        IF (IAN.EQ.1) GO TO 20
 40     STOP
        END
```

APPENDIX CC

DICCOLSA.FOR

```
C       DICCOLSA.FOR
C       DIComed COLor Stand Alone
C
C       PROGRAM BY:     G. BECKER
C       PHONE:          677-7621
C       DATE:           6 AUG 1981
C
C       THIS IS A GENERAL DICOMED COLOR STAND ALONE ROUTINE FOR
C       USE WITH POLAROID OR 35mm FILM.
C
C       THE REQUIREMENTS OF THE PROGRAM ARE:
C               1) THE DATA HAS BEEN EXTERNALLY CREATED AND WILL
C BE READ INTO THIS PROGRAM AS GREEN,RED,BLUE,ICNT().
C               2) THE FILM IS ASSUMED TO BE COLOR.
C
        INCLUDE 'DRA0:[UTILITY.DICOMED]COMMON.FOR'
        INCLUDE 'DRA0:[UTILITY.DICOMED]DEF.FOR'
        INCLUDE 'DRA0:[UTILITY.DICOMED]IO.FOR'
        INTEGER*2 ITEMBUF(512,512,3),INT2(512,256)
        BYTE NEUTRL(512,512),ICNT(512,512)
        EQUIVALENCE (ICNT,INT2)
        DIMENSION  JSUM(3)
        INTEGER*4 SYSSASSIGN
C*******INITIALIZATION OF VARIABLES*********************
        HOFSET=0
        VOFSET=0
        MAGZ=0
        NCOL(1)=0    !BLACK AND WHITE PART 1
        NCOL(2)=0    !BLACK AND WHITE PART 2
        NLIN=512
        NELE=512
        NLIN2=256
        BUFREF='7757'O
        BUFREQ=0
C       ASSIGN DR11B TO A CHANNEL
        ISTAT=SYSSASSIGN('UZA0:',ICHAN,,)
C       IF DR11B IS NOT ASSIGNED WE CANNOT CONTINUE
        IF (.NOT.ISTAT) THEN
```

```
                PRINT 10
 10     FORMAT ('0D R I I H   C A N N O T  B E  A S S I G N E D
     -  T O  A  C H A N N E L (***')
                GO TO 40
        ENDIF
C*******END INITIALIZATION*******************************
        TYPE*, '  '
        TYPE*, 'GENERAL STAND ALONE COLOR DICOMED ROUTINE'
        TYPE*, '  '
C*******INPUT FROM TERMINAL*****************************
        TYPE*, 'THE PICTURE DIMENSIONS ARE SET AT 512,512'
        TYPE*, 'ENTER TYPE OF FILM: POLAROID=0; 35mm=1.'
        ACCEPT*, IMOV
        TYPE*, 'DO YOU WISH TO FILTER THE DATA? (YES=1)'
        ACCEPT*, IFLT
        IF (IMOV.EQ.1) THEN
                TYPE*, 'THE HORIZONTAL OFFSET IS 140 OCTAL.'
                HOFSET=96                !140 OCTAL
                TYPE*, 'THE VERTICAL OFFSET IS 145 OCTAL.'
                VOFSET=101               !145 OCTAL
                TYPE*, 'THE RESOLUTION IS SET AT 1.'
                MODE=1
        ELSE
                TYPE*, 'THE RESOLUTION IS SET AT 3.'
                MODE=3
        ENDIF
        TYPE*, 'ENTER THE NUMBER OF DICOMED PASSES'
        ACCEPT *, IPASS
 20     TYPE*,'  IS A READ REQUIRED?(YES=1)'
        ACCEPT*, IRD
        IF (IRD.EQ.1) THEN
                TYPE*, ' ENTER THE FILE YOU WANT READ ITEMBUF '
                ACCEPT*, IFILE
                TYPE*, '   READING ITEMBUF FROM FILE  ', IFILE
        READ (IFILE,1000) (((ITEMBUF(I,J,K),I=1,NELE),J=1,
     -  NLIN),K=1,3)
        IF (IFLT.EQ.1) THEN
                TYPE*, '   READING INT2 FROM FILE  ', IFILE
                READ (IFILE,1000) ((INT2(I,J),I=1,NELE),
     *          J=1,NLIN2)
        ENDIF
 1000   FORMAT (66A2)
        ENDIF
C*******END INPUT FROM TERMINAL**************************
        IF(IFLT.EQ.1) THEN
C SCENE IS FILTERED TO FILL ANY MISSED PIXELS
C NOTE: THIS DATA WAS FILTERED JUST BEFORE BEING WRITTEN TO
C FILE
C BY THE PICTURE CREATE ROUTINE. HENCE ICNT() STILL CONTAINS
C THE PIXEL'S SUM BUT ITEMBUF HAS A SCALED VALUE READY FOR
C DISPLAY.
```

```
          TYPE*,'FILLING ANY MISSED PIXELS'
          DO 111 IYBS=1,NLIN
          DO 222 IXBS=1,NELE
          JCOUNT=0
            DO  6325  KVS=1,3
          JSUM(KVS)=0
6325      CONTINUE
          DO 300 K=IYBS-1,IYBS+1
                  IF((K.LT.1).OR.(K.GT.NLIN)) GOTO 300
          DO 440 L=IXBS-1,IXBS+1
                  IF((L.LT.1).OR.(L.GT.NELE)) GOTO 440
                  IF(ICNT(L,K).EQ.0) GO TO 440
                          JCOUNT=JCOUNT+1
            DO 5471  KVS=1,3
          JSUM(KVS)=JSUM(KVS)+ITEMBUF(L,K,KVS)
5471      CONTINUE
            IF(ICNT(L,K).LT.-1) IFG=1
C
C   TEST ICNT.LT.-1   SINCE THE MISSED PIXEL TAG IS -1
C         JUST ABOVE STATEMENT 220.
C
440       CONTINUE
300       CONTINUE
          IF(JCOUNT.EQ.0) THEN
          TYPE*,'MISSED PIXEL AT IXBS,IYBS=',IXBS,IYBS
           ELSE
            DO 4762 KVS=1,3
          ITEMBUF(IXBS,IYBS,KVS)=JSUM(KVS)/JCOUNT
4762      CONTINUE
           ENDIF
          ICNT(IXBS,IYBS)=-1
 220      CONTINUE
          IR=8
          IF(ITEMBUF(IXBS,IYBS,3).GT.0) THEN
          IF(ITEMBUF(IXBS,IYBS,3).GE.3*ITEMBUF(IXBS,IYBS,1)) THEN
              IR=ITEMBUF(IXBS,IYBS,3)    ! SKY

          ELSE IF(ITEMBUF(IXBS,IYBS,2).LT.ITEMBUF(IXBS,IYBS,1)/2)
         *     THEN
              IR=128-(ITEMBUF(IXBS,IYBS,3)/4)      ! LAKE
              ENDIF
          ENDIF
          ISHIFT=256*(IR/128)
          NEUTRL(IXBS,IYBS)=IR-ISHIFT
222       CONTINUE
111       CONTINUE
          ENDIF
**********DATA IS SENT TO THE DICOMED**********************
          DO 333 KKI=1,TPASS
          TYPE*,'  SENDING NEUTRAL DATA TO THE DICOMED'
C         VIDSET CALL 0470(*) TO:
```

```
C                1) CLEAR THE COMMAND BUFFER,
C                2) SET THE DATA SIZE TO 8 BITS,
C                3) SET THE RESOLUTION,
C                4) SET THE POLARITY TO NORMAL, AND
C                5) SET LINEAR STEPS IN DENSITY.
         CALL VIDSET
C        FILTER SELECTS THE FILTER TO BE USED VIA FILSEL
         CALL FILTER
C THE HORIZONTAL AND VERTICAL OFFSETS ARE GIVEN TO THE
C DICOMED.
         CALL D47CMD(HPOS)
         CALL D47POS(HOFSET)
         CALL D47CMD(VPOS)
         CALL D47POS(VOFSET)
           DO 1444 IXBS=NLIN,1,-1
                 CALL VDINIT
           DO 1555 IYBS=NELE,1,-1
         BUF1(IYBS)=NEUTRL(IXBS,IYBS)
1555     CONTINUE
                 CALL VIDOUT
1444     CONTINUE
         CALL OPCHK
         NCOL(1)=3
           DO  666 KVS=1,3
         TYPE*,'SENDING GREEN(1),RED(2),BLUE(3)=',KVS
                 CALL VIDSET
C RED= 1;  GREEN=2; BLUE=3
           IF(KVS.EQ.1) THEN
                 NCOL(2)=2
           ELSE
                 NCOL(2)=2*KVS-3
           ENDIF
                 CALL FILTER
                 CALL D47CMD(HPOS)
                 CALL D47POS(HOFSET)
                 CALL D47CMD(VPOS)
                 CALL D47POS(VOFSET)
         DO 444 IXBS=NLIN,1,-1
C        VDINIT CHECKS THE STATUS OF THE DICOMED BUFFERS.
         CALL VDINIT
                 DO 555 IYBS=NELE,1,-1
          IR=ITEMBUF(IXBS,IYBS,KVS)
          IF(KVS.EQ.3) THEN
          IF(IR.GT.4*ITEMBUF(IXBS,IYBS,1)) IR=260-IR+IR/4
          IF(IR.GT.255) IR=255
          ENDIF
C THIS MAKES THE SKY A CONSTANT BLUE, BUT WHITE FADES .
                          ISHIFT=256*(IR/128)
                          BUF1(IYBS)=IR-ISHIFT
 555             CONTINUE
         CALL VIDOUT
```

```
 444     CONTINUE
         CALL OPCHK
 666     CONTINUE
         TYPE*,'PASS',KKI,'COMPLETED'
C      SET THE FILTER BACK TO NEUTRAL.
         NCOL(1)=0
         NCOL(2)=0
 333     CONTINUE
         IF (IMOV.EQ.1) THEN
                 CALL FILADV
         ENDIF
         TYPE*, 'THE END'
         CALL OPCHK
         TYPE*,'D) YOU WANT TO DO ANOTHER PICTURE?(YES=1)'
         ACCEPT*, IAN
         IF (IAN.EQ.1) GO TO 20
 40      STOP
         END
```

APPENDIX CD

SDICRW.FOR

```
C       SDICRW.FOR
C       Subroutine DICOMED Black and White
C
C       PROGRAM BY:     G. BECKER
C       PHONE:          677-7621
C       DATE:           7 AUG 1981
C
C       THIS IS A GENERAL BLACK AND WHITE DICOMED SUBROUTINE
C       FOR USE WITH POLAROID OR 35mm BLACK AND WHITE FILM.
C       THE PARAMETERS THAT MUST BE PASSED FROM THE CALLING
C       ROUTINE ARE:
C               1) IMOV:=FILM SELECTION (POLAROID=0,35mm=1)
C               2) IFLT:=FILL ANY MISSED PIXELS (YES=1,NO=0)
C               3) IAN:=ANOTHER PICTURE OF THE SAME DATA (YES=1)
C
C NOTE: THE DATA MUST BE GENERATED IN THE CALLING ROUTINE AND
C       PASSED VIA ITEMBUF,INT2,ICNT
C
        SUBROUTINE SDICRW(IMOV,IFLT,IAN)
        INCLUDE 'DRA0:[UTILITY.DICOMED]COMMON.FOR'
        INCLUDE 'DRA0:[UTILITY.DICOMED]DEF.FOR'
        INCLUDE 'DRA0:[UTILITY.DICOMED]IO.FOR'
        INTEGER*4 SYSSASSIGN
C       ITEMBUF,INT2,ICNT MUST BE DIMENSIONED AND PASSED TO THIS
C       SUBROUTINE.
        INTEGER*2 ITEMBUF(512,512),INT2(512,256)
        BYTE ICNT(512,512)
        EQUIVALENCE (ICNT,INT2)
        COMMON ITEMBUF,ICNT
C*******INITIALIZATION OF VARIABLES*******************
        HOFSET=0
        VOFSET=0
        MAGZ=0
        NCOL(1)=0    !BLACK AND WHITE PART 1
        NCOL(2)=0    !BLACK AND WHITE PART 2
        NLIN=512
        NELE=512
        NLIN2=256
```

```
      BUFREE='77577')
      BUFREQ=0
C     ASSIGN DR11B TO A CHANNEL
      ISTAT=SYS$ASSIGN('UZA0:',ICHAN,,)
C     IF DR11B IS NOT ASSIGNED WE CANNOT CONTINUE
      IF (.NOT.ISTAT) THEN
            PRINT 10
 10   FORMAT ('0D R 1 1 B   C A N N O T  B E  A S S I G N E D
     - T O  A  C H A N N E L***')
            GO TO 40
      ENDIF
C*******END INITIALIZATION*****************************
      TYPE*, '  '
      TYPE*, 'GENERAL BLACK AND WHITE DICOMED SUBROUTINE'
      TYPE*, '  '
C*******INPUT FROM TERMINAL*****************************
      TYPE*, 'THE PICTURE DIMENSIONS ARE SET AT 512,512'
      TYPE*, 'TYPE OF FILM: POLAROID=0; 35mm=1;',IMOV
      TYPE*, 'FILTER THE DATA? (YES=1);',IFLT
      IF (IMOV.EQ.1) THEN
            TYPE*, 'THE HORIZONTAL OFFSET IS 140 OCTAL.'
            HOFSET=96                  !140 OCTAL
            TYPE*, 'THE VERTICAL OFFSET IS 145 OCTAL.'
            VOFSET=101                 !145 OCTAL
            TYPE*, 'THE RESOLUTION IS SET AT 1.'
            MODE=1
      ELSE
            TYPE*, 'THE RESOLUTION IS SET AT 3.'
            MODE=3
      ENDIF
C*******END INPUT FROM TERMINAL*************************
 20   IF (IFLT.EQ.1) THEN
C     SCENE IS FILTERED TO FILL ANY MISSED PIXELS
            TYPE*,'FILLING ANY MISSED PIXELS'
            DO 111 IYBS=1,NLIN
            DO 222 IXBS=1,NELE
            IF(ICNT(IXBS,IYBS).GE.0) GOTO 220
            JCOUNT=0
            JSUM=0
            DO 300 K=IYBS-1,IYBS+1
                  IF((K.LT.1).OR.(K.GT.NLIN)) GOTO 300
            DO 440 L=IXBS-1,IXBS+1
                  IF((L.LT.1).OR.(L.GT.NELE)) GOTO 440
                  IF(ICNT(L,K).EQ.0) GO TO 440
                        JCOUNT=JCOUNT+1
                        JSUM=JSUM+ITEMBUF(L,K)
 440        CONTINUE
 300        CONTINUE
            IF(JCOUNT.EQ.0) THEN
            TYPE*,'MISSED PIXEL AT IXBS,IYBS=',IXBS,IYBS
             ELSE
```

```
                ITEMBUF(IXBS,IYBS)=JSUM/JCOUNT
                 ENDIF
                ICNT(IXBS,IYBS)=-1
220             CONTINUE
222             CONTINUE
111             CONTINUE
        ENDIF
**************DATA IS SENT TO THE DICOMED**********
        TYPE*,'  SENDING DATA TO THE DICOMED'
C       VIDSET CALL D47CMD TO:
C               1) CLEAR THE COMMAND BUFFER,
C               2) SET THE DATA SIZE TO 8 BITS,
C               3) SET THE RESOLUTION,
C               4) SET THE POLARITY TO NORMAL, AND
C               5) SET LINEAR STEPS IN DENSITY.
        CALL VIDSET
C       FILTER SELECTS THE FILTER TO BE USED VIA FILSEL
        CALL FILTER
C THE HORIZONTAL AND VERTICAL OFFSETS ARE GIVEN TO THE
C DICOMED.
        CALL D47CMD(HPOS)
        CALL D47POS(HOFSET)
        CALL D47CMD(VPOS)
        CALL D47POS(VOFSET)
        DO 444 IXBS=NLIN,1,-1
C       VDINIT CHECKS THE STATUS OF THE DICOMED BUFFERS.
        CALL VDINIT
                DO 555 IYBS=NELE,1,-1
                        ISHIFT=256*(ITEMBUF(IXBS,IYBS)/128)
                        BUF1(IYBS)=ITEMBUF(IXBS,IYBS)-ISHIFT
555             CONTINUE
        CALL VIDOUT
444     CONTINUE
        IF (JMOV.EQ.1) THEN
                CALL FILADV
        ENDIF
        TYPE*, 'THE END'
        CALL OPCHR
        TYPE*,'ANOTHER PICTURE?(YES=1)',IAN
        IF (IAN.EQ.1) GO TO 20
40      RETURN
        END
```

APPENDIX CE

SDICCOL.FOR

```
C       SDICCOL.FOR
C       Subroutine DIComed COLor
C
C       PROGRAM BY:     G. BECKER
C       PHONE:          677-7621
C       DATE:           7 AUG 1981
C
C       THIS IS A GENERAL DICOMED COLOR SUBROUTINE FOR
C       USE WITH POLAROID OR 35mm FILM.
C       THE PARAMETERS THAT MUST BE PASSED FROM THE CALLING
C       ROUTINE ARE:
C               1) IMOV:=FILM SELECTION (POLAROID=0,35mm=1)
C               2) IFLT:=FILL ANY MISSED PIXELS (YES=1,NO=0)
C               3) IPASS:=THE NUMBER OF DICOMED PASSES
C               4) IAN:=ANOTHER PICTURE OF THE SAME DATA (YES=1).
C NOTE: THE DATA MUST BE GENERATED IN THE CALLING ROUTINE AND
C       PASSED VIA ITEMBUF,NEUTRL,INT2,ICNT
C
        SUBROUTINE SDICCOL(IMOV,IFLT,IPASS,IAN)
        INCLUDE 'DRA0:[UTILITY.DICOMED]COMMON.FOR'
        INCLUDE 'DRA0:[UTILITY.DICOMED]DEF.FOR'
        INCLUDE 'DRA0:[UTILITY.DICOMED]IO.FOR'
C       ITEMBUF,INT2,NEUTRL,ICNT MUST BE DIMENSIONED AND PASSED
C       TO THIS SUBROUTINE.
        INCLUDE '[PP734]DICBUF.FOR'
        DIMENSION  JSUM(3)
        INTEGER*4 SYS$ASSIGN
C*******INITIALIZATION OF VARIABLES*********************
        HOFSET=0
        VOFSET=0
        MAGZ=0
        NCOL(1)=0    !BLACK AND WHITE PART 1
        NCOL(2)=0    !BLACK AND WHITE PART 2
        NLIN=512
        NELE=512
        NLIN2=256
        BUFREE='77577'O
        BUFREQ=0
```

```
C       ASSIGN DR11B TO A CHANNEL
        ISTAT=SYS$ASSIGN('UZA0:',ICHAN,,)
C       IF DR11B IS NOT ASSIGNED WE CANNOT CONTINUE
        IF (.NOT.ISTAT) THEN
                PRINT 10
 10     FORMAT ('0D R 1 1 B   C A N N O T  B E  A S S I G N E D
     -  T O  A  C H A N N E L***')
                GO TO 40
        ENDIF
C*******END INITIALIZATION*******************************
        TYPE*, '  '
        TYPE*, 'GENERAL COLOR DICOMED SUBROUTINE'
        TYPE*, '  '
C*******INPUT FROM TERMINAL*******************************
        TYPE*, 'THE PICTURE DIMENSIONS ARE SET AT 512,512'
        TYPE*, 'TYPE OF FILM: POLAROID=0; 35mm=1;',IMOV
        TYPE*, 'FILTER THE DATA? (YES=1)',IFLT
        IF (IMOV.EQ.1) THEN
                TYPE*, 'THE HORIZONTAL OFFSET IS 140 OCTAL.'
                HOFSET=96                !140 OCTAL
                TYPE*, 'THE VERTICAL OFFSET IS 145 OCTAL.'
                VOFSET=101               !145 OCTAL
                TYPE*, 'THE RESOLUTION IS SET AT 1.'
                MODE=1
        ELSE
                TYPE*, 'THE RESOLUTION IS SET AT 3.'
                MODE=3
        ENDIF
        TYPE*, 'THE NUMBER OF DICOMED PASSES=',IPASS
C*******END INPUT FROM TERMINAL***********************
 20     IF(IFLT.EQ.1) THEN
C SCENE IS FILTERED TO FILL ANY MISSED PIXELS
CNOTE: THIS DATA WAS FILTERED JUST BEFORE BEING WRITTEN TO
C FILE BY THE PICTURE CREATE ROUTINE. HENCE ICNT()
C STILL CONTAINS THE PIXEL'S SUM BUT ITEMBUF HAS A SCALED
C VALUE READY FOR DISPLAY.
        TYPE*,'FILLING ANY MISSED PIXELS'
        DO 111 IYBS=1,NLIN
        DO 222 IXBS=1,NELE
        JCOUNT=0
          DO  6325  KVS=1,3
        JSUM(KVS)=0
6325    CONTINUE
        DO 300 K=IYBS-1,IYBS+1
                IF((K.LT.1).OR.(K.GT.NLIN)) GOTO 300
        DO 440 L=IXBS-1,IXBS+1
                IF((L.LT.1).OR.(L.GT.NELE)) GOTO 440
                IF(ICNT(L,K).EQ.0) GO TO 440
                        JCOUNT=JCOUNT+1
          DO 5471  KVS=1,3
        JSUM(KVS)=JSUM(KVS)+ITEMBUF(L,K,KVS)
```

```
5471     CONTINUE
           IF(ICNT(L,K).LT.-1) IFG=1
C  TEST ICNT.LT.-1   SINCE THE MISSED PIXEL TAG IS -1
C        JUST ABOVE STATEMENT 220.
440      CONTINUE
300      CONTINUE
         IF(JCOUNT.EQ.0) THEN
         TYPE*,'MISSED PIXEL AT IXBS,IYBS=',IXBS,IYBS
          ELSE
           DO 4762 KVS=1,3
         ITEMBUF(IXBS,IYBS,KVS)=JSUM(KVS)/JCOUNT
4762     CONTINUE
          ENDIF
         ICNT(IXBS,IYBS)=-1
 220     CONTINUE
         IR=8
         IF(ITEMBUF(IXBS,IYBS,3).GT.0) THEN
             IF(ITEMBUF(IXBS,IYBS,3).GE.3*ITEMBUF(IXBS,IYBS,1))
     *       THEN
             IR=ITEMBUF(IXBS,IYBS,3)    ! SKY

         ELSE IF(ITEMBUF(IXBS,IYBS,2).LT.ITEMBUF(IXBS,IYBS,1)/2)
     *       THEN
             IR=128-(ITEMBUF(IXBS,IYBS,3)/4)      ! LAKE
             ENDIF
         ENDIF
         ISHIFT=256*(IR/128)
         NEUTRL(IXBS,IYBS)=IR-ISHIFT
222      CONTINUE
111      CONTINUE
         ENDIF
**********DATA IS SENT TO THE DICOMED**********
         DO 333 KKI=1,IPASS
         TYPE*,'  SENDING NEUTRAL DATA TO THE DICOMED'
C        VIDSET CALL D47CMD TO:
C                1) CLEAR THE COMMAND BUFFER,
C                2) SET THE DATA SIZE TO 8 BITS,
C                3) SET THE RESOLUTION,
C                4) SET THE POLARITY TO NORMAL, AND
C                5) SET LINEAR STEPS IN DENSITY.
         CALL VIDSET
C        FILTER SELECTS THE FILTER TO BE USED VIA FILSEL
         CALL FILTER
C THE HORI. AND VERT. OFFSETS ARE GIVEN TO THE DICOMED
         CALL D47CMD(HPOS)
         CALL D47POS(HOFSET)
         CALL D47CMD(VPOS)
         CALL D47POS(VOFSET)
           DO 1444 IXBS=NLIN,1,-1
              CALL VDINIT
           DO 1555 IYBS=NELE,1,-1
```

```
      BUF1(IYBS)=NEUTRL(IXBS,IYBS)
1555  CONTINUE
            CALL VIDOUT
1444  CONTINUE
      CALL OPCHK
      NCOL(1)=3
        DO  666 KVS=1,3
      TYPE*,'SENDING GREEN(1),RED(2),BLUE(3)=',KVS
            CALL VIDSET
C RED= 1;  GREEN=2; BLUE=3
        IF(KVS.EQ.1) THEN
            NCOL(2)=2
        ELSE
            NCOL(2)=2*KVS-3
        ENDIF
            CALL FILTER
            CALL D47CMD(HPOS)
            CALL D47POS(HOFSET)
            CALL D47CMD(VPOS)
            CALL D47POS(VOFSET)
      DO 444 IXBS=NLIN,1,-1
C     VDINIT CHECKS THE STATUS OF THE DICOMED BUFFERS.
      CALL VDINIT
            DO 555 IYBS=NELE,1,-1
       IR=ITEMBUF(IXBS,IYBS,KVS)
       IF(KVS.EQ.3) THEN
       IF(IR.GT.4*ITEMBUF(IXBS,IYBS,1)) IR=415-IR/4
       IF(IR.GT.255) IR=255
       ENDIF
C THIS MAKES THE SKY A CONSTANT BLUE, BUT WHITE FADES .
                    ISHIFT=256*(IR/128)
                    BUF1(IYBS)=IR-ISHIFT
 555            CONTINUE
      CALL VIDOUT
 444  CONTINUE
      CALL OPCHK
 666  CONTINUE
      TYPE*,'PASS',KKI,'COMPLETED'
C     SET THE FILTER BACK TO NEUTRAL.
      NCOL(1)=0
      NCOL(2)=0
333   CONTINUE
      IF (IMOV.EQ.1) THEN
            CALL FILADV
      ENDIF
      TYPE*, 'THE END'
      CALL OPCHK
      TYPE*,'ANOTHER PICTURE?(YES=1)',IAN
      IF (IAN.EQ.1) GO TO 20
 40   RETURN
      END
```

APPENDIX DA

ROUTINES FOR TEXT GENERATION

Appendices DH through DG are the routines that are used to generate and test the REALSCAN text generation routines. Appendix DH contains the character set used to write parametric information on images generated from the EGLDATA data base. Appendix DI contains alternate character constructions for the character set of Appendix DH.

APPENDIX D8

DATACHAR.FOR

```
C       DATACHAR.FOR
C
C       DATACHAR.FOR CONTAINS THE DATA STATEMENTS FOR 'ICH' AND
C       'ASCH'.  ICH(7,9,51) DEFINES A CHARACTER SET WITH 51
C       DIFFERENT  7 X 9  ELEMENT CHARACTERS. AN ICH ELEMENT
C       OF "1" DESIGNATES AN ILLUMINATED ELEMENT AND "0"
C       DESIGNATES A SUPPRESSED ELEMENT.  ASCH(17,39) CONTAINS
C       THE  17 X 39 CHARACTER SPACES AVAILABLE FOR THE TEXT.
C       IT IS INITIALIZED WITH THE DECIMAL ASCII FOR EACH
C       CHARACTER IN THE DESIRED TEXT. A "0" DESIGNATES NO
C       CHARACTER PRESENT.
C
        DATA ICH1/
     + 0,0,0,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,
     + 0,0,0,-1,0,0,0,-1,-1,-1,-1,-1,-1,-1,0,0,0,-1,0,0,0,
     + 0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,0,0,0,0,0,0,0,
     , 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     , 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-1,-1,0,0,
     , 0,0,0,-1,-1,0,0,0,0,0,-1,-1,0,0,0,0,-1,-1,0,0,0,
     - 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     - 0,0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-1,0,0,0,0,0,0,0,
     - 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     . 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     . 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     . 0,0,0,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,-1,0,0,0,0,0,0,
     / 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-1,0,0,0,0,0,-1,0,
     / 0,0,0,0,0,-1,0,0,0,0,0,-1,0,0,0,0,0,-1,0,0,0,0,
     / 0,-1,0,0,0,0,0,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
C    DUE TO COMPILATION ERRORS 0 COULD NOT BE USED TO REPRESENT
C    ZERO.  A DASH (-) WAS USED IN ITS PLACE.
     - 0,0,-1,-1,-1,0,0,0,-1,0,0,0,-1,0,-1,0,0,0,0,0,0,-1,
     - -1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,
     - -1,0,0,0,0,0,-1,0,-1,0,0,0,-1,0,0,0,-1,-1,-1,0,0,
     1 0,0,0,-1,0,0,0,0,0,-1,-1,0,0,0,0,-1,0,-1,0,0,0,
     1 0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,
     1 0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,-1,-1,-1,-1,-1,-1,-1,
     2 0,-1,-1,-1,-1,-1,0,-1,0,0,0,0,0,-1,0,0,0,0,0,0,-1,
     2 0,0,0,0,0,0,-1,0,0,0,0,0,-1,0,0,0,0,0,-1,0,0,0,
```

```
     2 0,0,-1,0,0,0,0,0,-1,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-1,
     3 -1,-1,-1,-1,-1,-1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,
     3 0,0,0,0,0,0,-1,0,0,-1,-1,-1,-1,0,0,0,0,0,0,0,-1,
     3 0,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,0,-1,-1,-1,-1,-1,0,
     4 -1,0,0,0,0,0,0,-1,0,0,0,-1,0,0,-1,0,0,0,-1,0,0,
     4 -1,0,0,0,-1,0,0,-1,-1,-1,-1,-1,-1,-1,0,0,0,0,-1,0,0,
     4 0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,
     5 -1,-1,-1,-1,-1,-1,-1,-1,0,0,0,0,0,0,-1,0,0,0,0,0,0,
     5 -1,0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,0,0,0,0,0,0,0,-1,
     5 0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-1,0,
     6 0,0,0,-1,-1,0,0,0,0,-1,0,0,0,0,0,-1,0,0,0,0,0,
     6 -1,0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,0,-1,0,0,0,0,0,-1,
     6 -1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,0,-1,-1,-1,-1,-1,0,
     7 -1,-1,-1,-1,-1,-1,-1,-1,0,0,0,0,0,-1,0,0,0,0,0,0,-1,
     7 0,0,0,0,0,0,-1,0,0,0,0,0,-1,0,0,0,0,0,-1,0,0,
     7 0,0,0,-1,0,0,0,0,0,-1,0,0,0,0,0,-1,0,0,0,0,0,
     8 0,0,-1,-1,-1,0,0,0,-1,0,0,0,-1,0,-1,0,0,0,0,0,-1,
     8 0,-1,0,0,0,-1,0,0,0,-1,-1,-1,0,0,0,-1,0,0,0,-1,0,
     8 -1,0,0,0,0,0,-1,0,-1,0,0,0,-1,0,0,0,-1,-1,-1,0,0,
     9 0,-1,-1,-1,-1,-1,0,-1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,
     9 -1,0,0,0,0,0,-1,0,-1,-1,-1,-1,-1,-1,0,0,0,0,0,-1,0,
     9 0,0,0,0,-1,0,0,0,0,0,-1,0,0,0,0,-1,-1,0,0,0,0,
     : 0,0,0,0,0,0,0,0,0,0,0,-1,0,0,0,0,0,-1,-1,-1,0,0,
     : 0,0,0,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-1,0,0,0,
     : 0,0,-1,-1,-1,0,0,0,0,0,-1,0,0,0,0,0,0,0,0,0,0,
     ; 0,0,0,0,0,0,0,0,0,0,0,-1,0,0,0,0,0,-1,-1,-1,0,0,
     ; 0,0,0,-1,0,0,0,0,0,0,0,0,0,0,0,0,0,-1,-1,0,0,
     ; 0,0,0,-1,-1,0,0,0,0,0,-1,-1,0,0,0,0,-1,-1,0,0,0,
     < 0,0,0,0,0,-1,0,0,0,0,0,-1,0,0,0,0,0,-1,0,0,0,
     < 0,0,-1,0,0,0,0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,0,
     < 0,0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,0,-1,0,
     = 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     = -1,-1,-1,-1,-1,-1,-1,0,0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-1,
     = 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
     > 0,-1,0,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,
     > 0,0,0,0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,
     > 0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,0,0,-1,0,0,0,0,0,
     ? 0,0,0,-1,-1,-1,0,0,0,-1,0,0,0,-1,0,0,0,0,0,0,0,-1,0,
     ? 0,0,0,0,-1,-1,0,0,0,0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,
     ? 0,0,0,0,0,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,
     @ 0,-1,-1,-1,-1,-1,-1,0,-1,0,0,0,0,0,0,-1,-1,0,-1,-1,-1,0,-1,
     @ -1,0,-1,0,-1,0,-1,0,-1,-1,0,-1,0,-1,0,-1,0,-1,-1,0,-1,-1,-1,-1,0,
     @ -1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,-1,-1,-1,-1,-1,-1,0/
C
C
C
C
      DATA ICH2/
     A 0,0,0,-1,0,0,0,0,0,-1,0,-1,0,0,0,-1,0,0,0,0,-1,0,
     A -1,0,0,0,0,0,0,-1,-1,0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-1,-1,
     A -1,0,0,0,0,0,0,0,-1,-1,0,0,0,0,0,0,-1,-1,0,0,0,0,0,0,-1,
```

```
B -1,-1,-1,-1,-1,-1,0,-1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,
B -1,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-1,0,-1,0,0,0,0,0,-1,
B -1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-1,0,
C 0,0,-1,-1,-1,-1,0,0,-1,0,0,0,0,-1,-1,0,0,0,0,0,0,
C -1,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,0,0,0,
C -1,0,0,0,0,0,0,0,0,-1,0,0,0,0,-1,0,0,-1,-1,-1,-1,0,
D -1,-1,-1,-1,-1,-1,0,0,-1,0,0,0,0,-1,0,-1,0,0,0,0,0,-1,
D -1,0,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,
D -1,0,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,0,-1,-1,-1,-1,-1,0,0,
E -1,-1,-1,-1,-1,-1,-1,-1,-1,0,0,0,0,0,0,-1,0,0,0,0,0,0,
E -1,0,0,0,0,0,0,0,-1,-1,-1,-1,-1,0,0,-1,0,0,0,0,0,0,
E -1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-1,
F -1,-1,-1,-1,-1,-1,-1,-1,-1,0,0,0,0,0,0,-1,0,0,0,0,0,0,
F -1,0,0,0,0,0,0,0,-1,-1,-1,-1,-1,0,0,-1,0,0,0,0,0,0,
F -1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,0,0,0,
G 0,0,-1,-1,-1,-1,-1,0,0,-1,0,0,0,0,-1,-1,0,0,0,0,0,0,
G -1,0,0,0,0,0,0,0,-1,0,0,-1,-1,-1,-1,-1,-1,0,0,0,0,0,-1,
G -1,0,0,0,0,0,0,-1,0,-1,0,0,0,-1,0,0,0,-1,-1,-1,0,0,
H -1,0,0,0,0,0,0,-1,-1,0,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,
H -1,0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,0,0,0,0,0,0,-1,-1,
H 0,0,0,0,0,0,-1,-1,0,0,0,0,0,0,-1,-1,0,0,0,0,0,0,-1,
I 0,-1,-1,-1,-1,-1,-1,0,0,0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,
I 0,0,0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,
I 0,0,0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,0,-1,-1,-1,-1,-1,0,
J 0,0,0,-1,-1,-1,-1,-1,0,0,0,0,0,-1,0,0,0,0,0,0,0,-1,0,0,
J 0,0,0,0,0,-1,0,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,0,-1,0,0,
J 0,0,0,0,0,-1,0,0,-1,0,0,0,0,-1,0,0,0,0,-1,-1,-1,0,0,0,
K -1,0,0,0,0,0,0,-1,-1,0,0,0,0,0,0,-1,0,-1,0,0,0,-1,0,0,
K -1,0,0,0,-1,0,0,0,0,-1,0,0,-1,0,0,0,0,0,-1,-1,0,-1,0,0,0,
K -1,0,0,0,0,-1,0,0,0,-1,0,0,0,0,0,-1,0,0,-1,0,0,0,0,0,-1,
L -1,0,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,0,0,0,0,
L -1,0,0,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,
L 0,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-1,
M -1,0,0,0,0,0,0,-1,-1,-1,0,0,0,0,-1,-1,-1,0,-1,0,-1,0,
M -1,-1,0,0,0,-1,0,0,0,-1,-1,0,0,0,0,0,0,0,-1,-1,0,0,0,0,0,0,-1,
N -1,0,0,0,0,0,0,0,-1,-1,0,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,
O -1,0,0,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,-1,-1,0,0,0,0,-1,
O -1,0,-1,0,0,0,0,-1,-1,0,0,-1,0,0,0,-1,-1,0,0,0,0,-1,0,-1,
P -1,0,0,0,0,0,0,-1,-1,-1,0,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,
P -1,-1,-1,-1,-1,-1,-1,0,-1,0,0,0,0,0,0,-1,-1,0,0,0,0,0,
P -1,-1,0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-1,-1,0,-1,0,0,0,0,0,0,0,
P -1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,0,0,0,
Q 0,0,-1,-1,-1,-1,0,0,0,-1,0,0,0,0,-1,0,-1,0,0,0,0,0,
Q -1,-1,0,0,0,0,0,0,-1,-1,0,0,0,0,0,0,-1,-1,0,0,0,0,0,0,-1,
Q -1,0,0,0,0,-1,0,0,-1,0,-1,0,0,0,0,-1,0,0,0,-1,-1,-1,0,-1,
R -1,-1,-1,-1,-1,-1,-1,0,-1,0,0,0,0,0,0,-1,-1,0,0,0,0,0,0,
R -1,0,0,0,0,-1,0,0,-1,0,0,0,0,-1,0,-1,0,0,0,0,0,-1,
```

```
S 0,-1,-1,-1,-1,-1,-1,-1,0,0,0,0,0,0,-1,0,0,0,0,0,0,
S -1,0,0,0,0,0,0,0,-1,-1,-1,-1,-1,0,0,0,0,0,0,0,-1,
S 0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-1,0,
T -1,-1,-1,-1,-1,-1,-1,0,0,0,-1,0,0,0,0,0,0,-1,0,0,
T 0,0,0,0,-1,0,0,0 0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,
T 0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,
U -1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,
U -1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,
U -1,0,0,0,0,0,-1,0,-1,0,0,0,-1,-1,0,0,-1,-1,-1,0,-1,
V -1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,-1,-1,0,0,0,-1,-1,
V 0,-1,0,0,0,-1,0,0,-1,-1,0,-1,-1,0,0,0,-1,0,-1,0,0,0,
V 0,-1,-1,-1,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,
W -1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,
W -1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,
W -1,0,0,-1,0,0,-1,-1,0,-1,0,-1,0,-1,0,-1,0,0,0,-1,0,
X -1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,0,-1,0,0,0,-1,0,
X 0,0,-1,0,-1,0,0,0,0,0,-1,0,0,0,0,0,-1,0,-1,0,0,
X 0,-1,0,0,0,-1,0,-1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,
Y -1,0,0,0,0,0,-1,-1,0,0,0,0,0,-1,0,-1,0,0,0,-1,0,
Y 0,0,-1,0,-1,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,
Y 0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,
Z -1,-1,-1,-1,-1,-1,-1,0,0,0,0,0,0,-1,0,0,0,0,0,-1,0,
Z 0,0,0,0,-1,0,0,0,0,0,-1,0,0,0,0,0,-1,0,0,0,0,
Z 0,-1,0,0,0,0,0,-1,0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-1,
[ 0,-1,-1,-1,-1,-1,0,0,-1,0,0,0,0,0,0,-1,0,0,0,0,0,
[ 0,-1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,0,0,
[ 0,-1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,-1,-1,-1,-1,0,
  0,0,0,0,0,0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,0,0,
  0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,
  0,0,0,0,0,-1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,0,
] 0,-1,-1,-1,-1,-1,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,
] 0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,
] 0,0,0,0,0,-1,0,0,0,0,0,0,-1,0,0,-1,-1,-1,-1,-1,0/
C
C
C
C
      DATA ASCH/
*     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
*     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
*     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
*     0,0,88,61,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
*     0,0,89,61,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
*     0,0,65,61,0,0,46,0,0,0,0,0,0,0,0,0,0,0,0,0,
*     0,0,66,61,0,0,46,0,0,0,0,0,0,0,0,0,0,0,0,0,
*     0,0,67,61,0,0,46,0,0,0,0,0,0,0,0,0,0,0,0,0,
*     0,0,78,67,82,61,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
*     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
*     0,0,0,0,0,84,69,82,82,65,73,78,0,0,0,0,0,0,
*     0,0,83,73,90,69,61,0,0,0,0,0,0,0,0,0,0,0,
*     0,0,84,39,80,69,61,0,0,0,0,0,0,0,0,0,0,0,
```

```
*   0,0,72,73,82,90,61,0,0,0,0,0,0,0,0,0,0,
*   0,0,86,69,82,84,61,0,0,0,0,0,0,0,0,0,0,
*   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
*   0,0,0,81,82,69,69,83,47,82,79,67,75,83,0,0,0,
*   0,0,83,73,90,69,61,0,0,0,0,0,0,0,0,0,0,
*   0,0,84,89,80,69,61,0,0,0,0,0,0,0,0,0,0,
*   0,0,72,79,82,90,61,0,0,0,0,0,0,0,0,0,0,
*   0,0,86,69,82,84,61,0,0,0,0,0,0,0,0,0,0,
*   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
*   0,0,71,82,65,83,83,47,76,69,65,86,69,83,0,0,0,
*   0,0,83,73,90,69,61,0,0,0,0,0,0,0,0,0,0,
*   0,0,84,89,80,69,61,0,0,0,0,0,0,0,0,0,0,
*   0,0,72,79,82,90,61,0,0,0,0,0,0,0,0,0,0,
*   0,0,86,69,82,84,61,0,0,0,0,0,0,0,0,0,0,
*   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
*   0,0,0,0,0,0,76,65,75,69,0,0,0,0,0,0,0,
*   0,0,83,67,65,76,69,61,0,0,0,0,0,0,0,0,0,
*   0,0,65,88,73,83,0,83,84,82,61,0,0,0,0,0,0,
*   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
*   0,0,0,0,0,70,73,69,76,68,83,0,0,0,0,0,0,
*   0,0,83,75,79,80,69,61,0,0,0,0,0,0,0,0,0,
*   0,0,83,73,90,69,61,0,0,0,0,0,0,0,0,0,0,
*   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
*   0,83,85,78,0,83,72,65,68,79,87,58,0,0,0,0,0,
*   0,0,70,73,82,83,84,58,0,0,0,0,0,0,0,0,0,
*   0,0,83,69,67,79,78,68,58,0,0,0,0,0,0,0,0/
```

APPENDIX DC

DICCHAR.FOR

```
C       DICCHAR.FOR
C
C       THIS ROUTINE IS THE CHARACTER GENERATOR FOR THE DICOMED.
C       THE ARRAY ITEXT IS GENERATED FROM THE ARRAY ASCH USING
C       THE INPUT CHARACTER STRING VARIABLES AND THOSE STRINGS
C       GENERATED BY CHCONV.  ITEXT IS THEN READY FOR OUTPUT
C       TO THE DICOMED BY A SEPARATE DICOMED CONTROL ROUTINE.
C
C       CALLING ROUTINE:  UNSPECIFIED DICOMED CONTROL ROUTINE
C
C       ROUTINES CALLED:  SUBROUTINE CHCONV IN CHCONV.FOR
C
C       INPUT VARIABLES:  A - SUN VECTOR X COMPONENT
C                         B - SUN VECTOR Y COMPONENT
C                         C - SUN VECTOR Z COMPONENT
C                         ISNT - SIGN FLAG FROM SUB. CHCONV
C                         IX - X POSITION OF SCENE CENTER
C                         JY - Y POSITION OF SCENE CENTER
C                         NCR - DATABASE COORDINATE SCALE FACTOR
C
C       INPUT ARRAYS:     BCHTST(9) - CHAR. CONV. RETURN ARRAY
C                         FLOSL(8) - DATABASE SCALE FACTOR
C                         FLOSZ(9) -          "
C                         GRAHOR(9) -         "
C                         GRASZ(9) -          "
C                         GRATP(9) -          "
C                         GRAVER(9) -         "
C                         LAKAS(5) -          "
C                         LAKSC(8) -          "
C                         MONTH(3) - MONTH INPUT ARRAY
C                         ROCHOR(9) - DATABASE SCALE FACTOR
C                         ROCSZ(9) -          "
C                         ROCTP(9) -          "
C                         ROCVER(9) -         "
C                         SUNET(8) -          "
C                         SUNSD(7) -          "
C                         SUNSH(3) - SUN SHADOW?-YES OR NO
C                         TERHOR(9) - DATABASE SCALE FACTOR
```

```
C                     TERSZ(9) -          "
C                     TERTP(9) -          "
C                     TERVER(9) -         "
C                     TITLE(15) - TITLE CHARACTER STRING
C
C     OUTPUT VARIABLES: IFGT - FLAG FOR REAL OR INTEGER CONV.
C                       ITST - INTEGER VARIABLE FOR CONVERSION
C                       RTST - REAL VARIABLE FOR CONVERSION
C
C     OUTPUT ARRAYS:    ITEXT - DICOMED TEXT ARRAY
C
C     LOCAL VARIABLES:  ICHDEX - ASCH ARRAY CODE FOR CHARACTER
C                       IND - POINTER TO FIRST NON-ZERO CHAR.
C                       K - DO LOOP COUNTER
C                       K1 -      "
C                       K2 -      "
C                       K3 -      "
C                       K3DEX - CHARACTER PLACEMENT POINTER
C                       K4 - DO LOOP COUNTER
C                       K4DEX - CHARACTER PLACEMENT POINTER
C                       KK - CHARACTER PLACEMENT COUNTER
C
C     LOCAL ARRAYS:     ASCH(17,39) - TEXT CHARACTER ARRAY
C                       CHTST(9) - CONVERSION CHARACTER ARRAY
C                       ICH(7,9,51) - CHARACTER SET ARRAY
C                       ICH1(7,9,22) - PARTIAL CHAR SET ARRAY
C                       ICH2(7,9,29) - PARTIAL CHAR SET ARRAY
C                       NCRCH(9) - NCR CONVERSION CHAR. ARRAY
C                       SUNA(9) - "A" CONVERSION CHAR. ARRAY
C                       SUNB(9) - "B" CONVERSION CHAR. ARRAY
C                       SUNC(9) - "C" CONVERSION CHAR. ARRAY
C                       XPOS(9) - "IX" CONVERSION CHAR. ARRAY
C                       YPOS(9) - "JY" CONVERSION CHAR. ARRAY
C                       CHDATE*9 - DATE CHARACTER STRING
C                       CHTIME*8 - FULL TIME CHARACTER STRING
C                       CHTIM*5 - HRS:MIN CHARACTER STRING
C
C     NOTE: THE FOLLOWING VARIABLES AND ARRAYS ARE NOT USED
C           BY THIS ROUTINE BUT EXIST DUE THE USE OF COMMON
C           BLOCKS WHICH HAVE VARIABLES OR ARRAYS NECESSARY
C           TO SUBROUTINE DICCHAR:
C                              DIVSKY
C                              ICOLOR
C                              IFIR
C                              SKYSCL
C                              SIARG
C                              IPP(2)
C                              IRV(3)
C
C     FOR MORE INFORMATION SEE:
C                       SAM M. RICHIE
```

```
C                       PH: 677-7621
C
C
C       DATE LAST REVISION: 1 SEP 81
C
        SUBROUTINE DICCHAR
C
        CHARACTER*1 XPOS(9),YPOS(9),SUNA(9),SUNB(9),
     *  SUNC(9),ICRCH(9),TERSZ(9),TERTP(9),TERHOR(9),TERVER(9),
     *  ROCSZ(9),ROCTP(9),ROCHOR(9),ROCVER(9),GRASZ(9),GRATP(9),
     *  GRAHOR(9),GRAVER(9),LAKSC(8),LAKAS(5),FLDSL(8),FLDSZ(9),
     *  SUNSH(3),SUNFT(8),SUNSD(7),CHISI(9),TITLE(15)
C
        CHARACTER CHDATE*9,CHTIME*8,CHTIM*5
C
C
C
        BYTE ASCH(17,39),ITEXT(170,512),BCHTST(9),ICH(7,9,51)
        BYTE ICH1(7,9,22),ICH2(7,9,29)
        BYTE ITXTFLAG
        INTEGER*2 NCR,ICOLOR,IFIR,IRV(3)
C
C
C
C
C
C
        EQUIVALENCE (ASCH(7,4),XPOS(1)),
     *  (ASCH(7,5),YPOS(1)),(ASCH(8,6),SUNA(1)),
     *  (ASCH(8,7),SUNB(1)),(ASCH(8,8),SUNC(1)),
     *  (ASCH(8,9),ICRCH(1)),(ASCH(9,12),TERSZ(1)),
     *  (ASCH(9,13),TERTP(1)),(ASCH(9,14),TERHOR(1)),
     *  (ASCH(9,15),TERVER(1)),(ASCH(9,18),ROCSZ(1)),
     *  (ASCH(9,19),ROCTP(1)),(ASCH(9,20),ROCHOR(1)),
     *  (ASCH(9,21),ROCVER(1)),(ASCH(9,24),GRASZ(1)),
     *  (ASCH(9,25),GRATP(1)),(ASCH(9,26),GRAHOR(1)),
     *  (ASCH(9,27),GRAVER(1)),(ASCH(10,30),LAKSC(1)),
     *  (ASCH(13,31),LAKAS(1)),(ASCH(10,34),FLDSL(1)),
     *  (ASCH(9,35),FLDSZ(1)),(ASCH(14,37),SUNSH(1)),
     *  (ASCH(10,38),SUNFT(1)),(ASCH(11,39),SUNSD(1)),
     *  (ASCH(2,1),TITLE(1)),(ASCH(2,2),CHDATE),
     *  (ASCH(12,2),CHTIM)
C
        EQUIVALENCE (BCHTST,CHTST),(ICH,ICH1),(ICH(1,1,23),ICH2)
C
        COMMON/CHR2/ ASCH,ITXTFLAG
        COMMON/SUN/A,B,C,SMARG,SKYSCL,DIVSKY
        COMMON/COLOR/ICOLOR,IFIR,NCR,IPP(2),IRV,IX,JY
        COMMON/CHR3/ ITEXT
C
        COMMON/CONV/ IEGT,ITST,RTST,BCHIST,ISHT
C
C
```

```
      DATA IEND3/39/
C
      COMMON/DATABASE/N1024,IOFF1(3,2),IOFF2(3,2),SCALE(3,2),
     *  IBRFLD(3),SHOVA,COVMARG,IDOV2,KSI,KSI4,KSI5,KSI7,KSI8,
     *  IDENSITY(3,2),SR4OVA,SR7OVA,DNO1(3,2),W1,W,IPIL,LSCAL,
     *  INAVSCAL
C
C
      INCLUDE 'DATACHAR.FOR'
C
C
      IFGT=0
      ITST=IX
      CALL CHCONV
      IF(ISNT.EQ.1) ASCH(6,4)=45
      IF(ISNT.EQ.0) ASCH(6,4)=0
      DO 10 K=1,9
            IND=K
            IF(BCHTST(IND).NE.48) GOTO 11
10    CONTINUE
11    KK=0
      DO 12 K=IND,9
            KK=KK+1
12    XPOS(KK)=CHTST(K)
      DO 13 K=KK+1,9
13    XPOS(K)=SUNA(10)  ! SUNA(10) IS '0' CONSTANT
      ITST=IY
      CALL CHCONV
      IF(ISNT.EQ.1) ASCH(6,5)=45
      IF(ISNT.EQ.0) ASCH(6,5)=0
      DO 20 K=1,9
            IND=K
            IF(BCHTST(IND).NE.48) GOTO 21
20    CONTINUE
21    KK=0
      DO 22 K=IND,9
            KK=KK+1
22    YPOS(KK)=CHTST(K)
      DO 23 K=KK+1,9
23    YPOS(K)=SUNA(10)  ! SUNA(10) IS '0' CONSTANT
      ITST=NCR
      CALL CHCONV
      DO 30 K=1,9
            IND=K
            IF(BCHTST(IND).NE.48) GOTO 31
30    CONTINUE
31    KK=0
      DO 32 K=IND,9
            KK=KK+1
32    NCRCH(KK)=CHTST(K)
      DO 33 K=KK+1,9
```

```
33      NCRCH(K)=SUNA(10)  ! SUNA(10) IS "0" CONSTANT
        IFGT=1
        RTST=A
        CALL CHCOUV
        IF(ISNT.EQ.1) ASCH(6,6)=45
        IF(ISNT.EQ.0) ASCH(6,6)=0
        DO 40 K=7,1,-1
                IND=K
                IF(RCHTST(K).NE.48) GOTO 41
40      CONTINUE
41      KK=0
        DO 42 K=1,IND
                KK=KK+1
42      SUNA(K)=CHTST(K)
        DO 43 K=KK+1,9
43      SUNA(K)=SUNA(10)  ! SUNA(10) IS "0" CONSTANT
        RTST=B
        CALL CHCOUV
        IF(ISNT.EQ.1) ASCH(6,7)=45
        IF(ISNT.EQ.0) ASCH(6,7)=0
        DO 50 K=7,1,-1
                IND=K
                IF(RCHTST(IND).NE.48) GOTO 51
50      CONTINUE
51      KK=0
        DO 52 K=1,IND
                KK=KK+1
52      SUNA(K)=CHTST(K)
        DO 53 K=KK+1,9
53      SUNB(K)=SUNA(10)  ! SUNA(10) IS "0" CONSTANT
        RTST=C
        CALL CHCOUV
        IF(ISNT.EQ.1) ASCH(6,8)=45
        IF(ISNT.EQ.0) ASCH(6,8)=0
        DO 60 K=7,1,-1
                IND=K
                IF(RCHTST(IND).NE.48) GOTO 61
60      CONTINUE
61      KK=0
        DO 62 K=1,IND
                KK=KK+1
62      SUNC(K)=CHTST(K)
        DO 63 K=KK+1,9
63      SUNC(K)=SUNA(10)  ! SUNA(10) IS "0" CONSTANT
C
        IFGT=0
        ITST=LSCAL
        CALL  CHCOUV
          DO 70 K=1,9
        IND=K
        IF(RCHTST(IND).NE.48) GOTO 71
```

```
 70      CONTINUE
 71      KK=0
         DO  72 K=IND,9
           KK=KK+1
 72      LAKSC(KK)=CHTST(K)
           DO 73 K=KK+1,9
 73      LAKSC(KK)=SUNA(10)
         ITST=I*AVSCAL
         CALL  CHCONV
           DO 80 K=1,9
         IND=K
         IF(BCHTST(IND).NE.48)  GOTO 81
 80      CONTINUE
 81      KK=0
           DO 82 K=IND,9
         KK=KK+1
 82      LAKAS(KK)=CHTST(K)
           DO 83 K=KK+1,9
 83      LAKAS(KK)=SUNA(10)
C
         IF(SUNSH(1).EQ.'N') THEN
                 DO 600 K2=38,39
                 DO 600 K1=3,17
                          ASCH(K1,K2)=0
600              CONTINUE
         ENDIF
C
C        CALL TO SYSTEM SUBROUTINES FOR DATE AND TIME
C
         CALL DATE(CHDATE)
         CALL TIME(CHTIME)
         CHTIM(1:5)=CHTIME(1:5)
C
C
C        FILL ITEXT WITH CHARACTERS POINTED BY ASCH
         IEND=39
         DO 100 K1=1,IEND3
         DO 200 K2=1,17
                  IF(ASCH(K2,K1).EQ.32) ASCH(K2,K1)=0
                  K3DEX=(K1-1)*13 + 2
                  K4DEX=(K2-1)*10
                  ICHDEX=ASCH(K2,K1)-42
                  DO 300 K3=1,9
                  DO 300 K4=1,7
                     IF(ASCH(K2,K1).EQ.0) THEN
                        ITEXT(K4+K4DEX,K3+K3DEX)=0
                     ELSE
                        ITEXT(K4+K4DEX,K3+K3DEX)=ICH(K4,K3,ICHDEX)
                     ENDIF
300      CONTINUE
200      CONTINUE
```

```
100     CONTINUE
        IEND3=0
C
        RETURN
C
        END
```

APPENDIX D

CHCONV.FOR

```
C
C  CHCONV.FOR
C
C  THIS SUBROUTINE CONVERTS FROM A SIGNED INTEGER VARIABLE
C  OR A REAL VARIABLE (MAGNITUDE LESS THAN 1) TO A CHARACTER
C  STRING WHICH REPRESENTS THE NUMERIC VALUE OF THE VARIABLE
C
C  CALLING ROUTINE:      SUBROUTINE DICCHAR IN DICCHAR.FOR
C
C  ROUTINES CALLED:      NONE
C
C  INPUT VARIABLES:      IFST - FLAG FOR REAL OR INTEGER CONVERT
C                        IIST - INTEGER VARIABLE FOR CONVERSION
C                        RTST - REAL VARIABLE FOR CONVERSION
C
C  INPUT ARRAYS:         NONE
C
C  OUTPUT VARIABLE:      NCHFST - ASCII STRING FROM CONVERSION
C                        ISNF - NEGATIVE VALUE FLAG FOR INPUT
C
C  OUTPUT ARRAYS:        NCHTST(9) - CHAR STRING FROM CONVERSION
C
C  LOCAL VARIABLES:      I - LOOP COUNTER
C                        I1 - INTERMEDIATE VARIABLE USED
C                             IN CONVERSION
C                        I2 - INTERMEDIATE VARIABLE USED
C                             IN CONVERSION
C                        N1 - NUMERIC VALUE FOR ASCII DIGIT
C
C  LOCAL ARRAYS:         NONE
C
C       FOR MORE INFORMATION, SEE:
C                        JAY J. RICHTER
C                        PH: 677-7621
C
C       DATE LAST REVISED: 14 APR 81
C
      SUBROUTINE CHCONV
      BYTE NCHTST(9)
```

```
      COMMON/CONV/ IFGT,ITST,RTST,BCHIST,ISNT
C
C
C     IFGT=1 MEANS REAL CONVERT,IFGT=0 MEANS INTEGER CONVERT
C
C
C
C     ISNT=1 MEANS NEGATIVE VALUE OF VARIABLE
C     ISNT=0 MEANS POSITIVE VALUE OF VARIABLE
C
C
      ISNT=0
      DO 30 I=1,9
30    BCHTST(I)=0
      IF(IFGT.EQ.1) GOTO 100
C
C     INTEGER CONVERT 9 DIGITS CAPACITY
C
      IF(ITST.LT.0) ISNT=1
      ITST=IABS(ITST)
      IR1=ITST
      IR2=ITST
      DO 10 I=1,9
          IR1=IR1/10
          NUM=IR2-IR1*10
          IR2=IR2/10
          BCHTST(10-I)=NUM+48 ! 48 ADDED TO GET ASCII CODE
10    CONTINUE
      GOTO 20
C
C
C
C     REAL CONVERT FOR |RTST|<1    7 DIGIT CAPACITY
C
100   IF(RTST.LT.0) ISNT=1
      RTST=ABS(RTST)
      IR1=0
      DO 20 I=1,7
          RTST=RTST*10
          IR1=IR1*10
          NUM=RTST-IR1
          BCHTST(I)=NUM+48 ! 48 ADDED TO GET ASCII CODE
              IR1=RTST
20    CONTINUE
C
C
      RETURN
      END
```

APPENDIX DE

DICCAMMAN.FOR

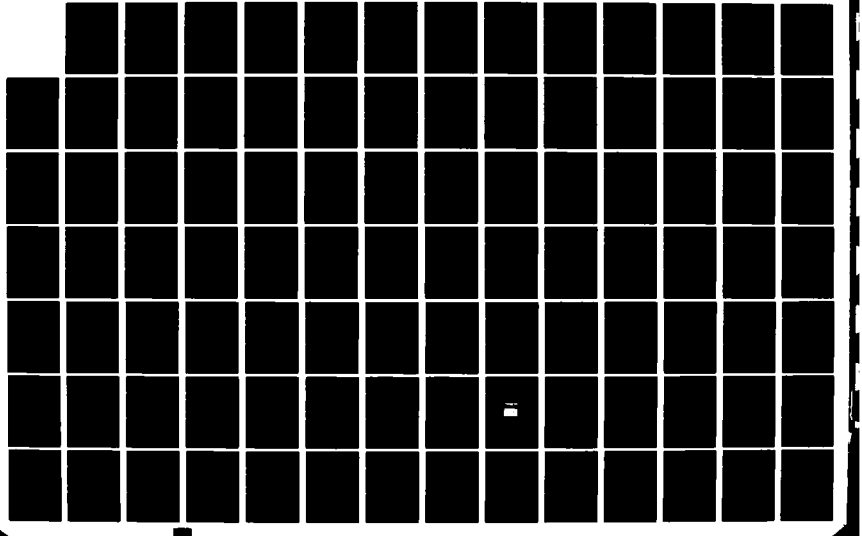
```
C       DICCAMMAN.FOR
C
C  THIS IS A GENERAL PURPOSE DICOMED ROUTINE WHICH ALLOWS FOR
C  THE INPUT OF:
C                   1) FILM TYPE,
C                   2) PICTURE SIZE,
C                   3) RESOLUTION, AND
C                   4) HORIZONTAL AND VERTICAL OFFSET.
C
C       THE REQUIREMENTS OF THE PROGRAM ARE:
C          1) THE DATA HAS BEEN EXTERNALLY CREATED AND WILL BE
C             READ INTO THIS PROGRAM AS GREEN,RED,BLUE,ICNT().
C          2) THE FILE IS ASSIGNED TO DF CO:DF.
C
C
C
C       SUBROUTINE DRIVECCAM
C       INCLUDE 'SRA0:[UTILITY.DICOMED]CCAMDA.FOR'
C       INCLUDE 'SRA0:[UTILITY.DICOMED]CCAMEQU.FOR'
C       INCLUDE 'SRA0:[UTILITY.DICOMED]DF.FOR'
C       INCLUDE '($RMSDEF.FOR)'
C       INCLUDE '($IODEF.FOR)'
C       BYTE IMAGE(0:2,0:511),IY-X1(17*,512),IFXDELAY,ASCII(17,39)
C       DIMENSION  JSUM(3)
C       INTEGER*4 SYSSASSIGN
C       INTEGER*2 IMAGE(512,512,3)
C       COMMON/DICCAM/IMAGE
C       COMMON/CHR3/ IPIX(
C       COMMON/CHR2/ ASCII,IFXDELAY
C*******INITIALIZATION OF VARIABLES*****************************
C       XOFSET=0
C       YOFSET=0
C       MAG=0
C       XCOL(1)=0
C       XCOL(2)=0
C       ASSIGN DR11W TO '2 (NAME OF
C       ISTAT=SYSSASSIGN('DRA0:',ICHAN,,)
C       IF DR11W IS NOT ASSIGNED SEE CAMERA OPERATOR
C       IF (.NOT.ISTAT) PAUSE
```

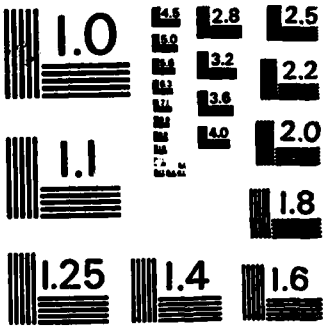
```
                PRINT 10
 10             FORMAT ('00 R 1 1 B  C A N N O T  B E  A S S I
     - G N E D  T O  A  C H A N N E L***')
                GO TO 40
        ENDIF
C*******END INITIALIZATION*****************************************
C
        BUFREE='77577'O
        BJFREQ=0
        TYPE*, ' '
C*******INPUT FROM TERMINAL*****************************************
        NLIN=512
        NLIN2=NLIN/2
        TYPE*, 'ENTER TYPE OF FILM: POLAROID=0; 35mm=1.'
        ACCEPT*, IMOV
        IF(IMOV.EQ.0) THEN
                MODE=3
                VOFSET=0
                HOVSET=0
                IF(ITXTFLAG.EQ.0) NELE=512
                IF(ITXTFLAG.EQ.1) NELE=682
        ELSE
                VOFSET=101
                IF(ITXTFLAG.EQ.0) THEN
                        NELE=512
                        MODE=1
                        HOFSET=96
                ELSE
                        NELE=682
                        MODE=2
                        HOFSET=85
                ENDIF
        ENDIF

        TYPE*,'ENTER THE NUMBER OF DICOMED PASSES'
        ACCEPT*, IPASS
C*******END INPUT FROM TERMINAL*************************************
C SCENE IS FILTERED TO FILL ANY MISSED PIXELS
C
C  NOTE: THIS DATA WAS FILTERED JUST BEFORE BEING WRITTEN TO
C        FILE BY THE PICTURE CREATE ROUTINE. HENCE ICNT()
C        STILL CONTAINS THE PIXEL'S SUM BUT IBUFX HAS A SCALED
C        VALUE READY FOR DISPLAY.
C
        IF(ITXTFLAG.EQ.0) GOTO 110
        TYPE*,'CREATING TEXT ARRAY'
        CALL DICCHAR
110     DO 111 IYBS=1,512
        DO 222 IXBS=1,512
        IR=8
        IF(IBUFX(IXBS,IYBS,3).GT.0) THEN
```

AD-A122 000 REAL SCAN EVOLUTION(U) UNIVERSITY OF CENTRAL FLORIDA 5/6
ORLANDO DEPT OF ELECTRICAL ENGINEERING B W PATZ ET AL.
FEB 82 NAVTRAEQUIPC-80-D-0014-2 N61339-80-D-0014

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

```
            IF(IBUFX(IXBS,IYBS,3).GE.3*IBUFX(IXBS,IYBS,1)) THEN
                 IR=IBUFX(IXBS,IYBS,3)    ! SKY

            ELSE IF(IBUFX(IXBS,IYBS,2).LT.IBUFX(IXBS,IYBS,1)/2)
     *           THEN
                 IR=32-(IBUFX(IXBS,IYBS,3)/8)      ! LAKE
            ENDIF
         ENDIF
         ISHIFT=256*(IR/128)
         NEUTRL(IXBS,IYBS)=IR-ISHIFT
222      CONTINUE
111      CONTINUE
         IF(ITXTFLAG.EQ.0) GOTO 112
                 DO 114 KSR=1,512
                 DO 114 KSM=1,170
                 NEUTRL(KSM+512,513-KSR)=ITEXT(KSM,KSR)
114              CONTINUE
112      DO 333 KK1=1,IPASS
         TYPE*,'  SENDING NEUTRAL DATA TO THE DICOMED'
C        VIDSET CALL D47CMD TO:
C                1) CLEAR THE COMMAND BUFFER,
C                2) SET THE DATA SIZE TO 8 BITS,
C                3) SET THE RESOLUTION,
C                4) SET THE POLARITY TO NORMAL, AND
C                5) SET LINEAR STEPS IN DENSITY.
         CALL VIDSET
C        FILTER SELECTS THE FILTER TO BE USED VIA FILSEL
         CALL FILTER
C        THE HORIZONTAL AND VERTICAL OFFSETS ARE GIVEN TO
C        THE DICOMED.
         CALL D47CMD(HPOS)
         CALL D47POS(HOFSET)
         CALL D47CMD(VPOS)
         CALL D47POS(VOFSET)
           DO 1444 IYBS=512,1,-1
                 CALL VDIWIT
C                IYBSS=0
           DO 1555 IXBS=1,512+ITXTFLAG*175
C                IYBSS=IYBSS+1
         BUF1(IXBS)=NEUTRL(IXBS,IYBS)
1555     CONTINUE
                 CALL VIDWIT
1444     CONTINUE
         CALL OPCHK
         NCOL(1)=3
           DO  666 KVS=1,3
         TYPE*,'SENDING GREEN(1),RED(2),BLUE(3)=',KVS
                 CALL VIDSET
C RED= 1;  GREEN=2; BLUE=3
           IF(KVS.EQ.1) THEN
                 NCOL(2)=2
```

```
          ELSE
                NCOL(2)=2*KVS-3
          ENDIF
                CALL FILTER
                CALL D47CMD(HPOS)
                CALL D47POS(HOFSET)
                CALL D47CMD(VPOS)
                CALL D47POS(VOFSET)
        DO 444 IYBS=512,1,-1
C       VOINIT CHECKS THE STATUS OF THE DICOMED BUFFERS.
        CALL VOINIT
C               IYBSS=0
                DO 555 IXBS=1,512
C               IYBSS=IYBSS+1
         IR=IBUFX(IXBS,IYBS,KVS)
         IF(KVS.EQ.3) THEN
         IF(IR.GT.4*IBUFX(IXBS,IYBS,1)) IR=415-IR/4
         IF(IR.GT.255) IR=255
         ENDIF
C THIS MAKES THE SKY A CONSTANT BLUE, BUT WHITE FADES .
                        ISHIFT=256*(IR/128)
                        BUF1(IXBS)=IR-ISHIFT
 555            CONTINUE
        CALL VIDOUT
 444    CONTINUE
        CALL OPCHK
 666    CONTINUE
        TYPE*,'PASS',KKI,'COMPLETED'
C     SET THE FILTER BACK TO NEUTRAL.
        NCOL(1)=0
        NCOL(2)=0
333     CONTINUE
        IF (IMOV.EQ.1) THEN
                TYPE*, 'DO YOU NEED A FILM ADVANCE? YES=1'
                ACCEPT*, IADV
                IF (IADV.EQ.1) CALL FILADV
        ENDIF
        TYPE*, 'THE END'
        CALL OPCHK
 40     RETURN
        END
```

APPENDIX DF

EGLPICMAN.FOR

```
C       EGLPICMAN.FOR
C
C       THIS ROUTINE ALLOWS FOR THE GENERATION OF A 'BIRD'S-EYE
C          VIEW' OF THE DATA BASE.  IT AUTOMATICALLY ALLOWS THE
C          PICTURE TO BE SENT TO THE DICOMED, THEREFORE SAVE THE
C          WRITE AND READ TIMES INCURRED IN THE PAST.
C
C
C DECLARATIONS FOR PICTURE GENERATION
C
C
C
```

```
      INTEGER*2 JDAT1(512,256),JDAT2(512,256),JDAT3(512,256)
      EQUIVALENCE (I,IPP(1)),(J,IPP(2))
      EQUIVALENCE (JDAT1,IDAT),(JDAT2,IDAT(1,1,2)),
     *  (JDAT3,IDAT(1,1,3))
C
C
C
C
      EQUIVALENCE (ASCH(7,4),XPOS(1)),
     *  (ASCH(7,5),YPOS(1)),(ASCH(8,6),SUNA(1)),
     *  (ASCH(8,7),SUNB(1)),(ASCH(8,8),SUNC(1)),
     *  (ASCH(8,9),NCHCH(1)),(ASCH(9,12),TERSZ(1)),
     *  (ASCH(9,13),TERTP(1)),(ASCH(9,14),TERHOR(1)),
     *  (ASCH(9,15),TERVER(1)),(ASCH(9,18),ROCSZ(1)),
     *  (ASCH(9,19),ROCTP(1)),(ASCH(9,20),ROCHOR(1)),
     *  (ASCH(9,21),ROCVER(1)),(ASCH(9,24),GRASZ(1)),
     *  (ASCH(9,25),GRATP(1)),(ASCH(9,26),GRAHOR(1)),
     *  (ASCH(9,27),GRAVER(1)),(ASCH(10,30),LAKSC(1)),
     *  (ASCH(10,31),LAKAS(1)),(ASCH(10,34),FLDSL(1)),
     *  (ASCH(9,35),FLDSZ(1)),(ASCH(14,37),SUNSH(1)),
     *  (ASCH(10,38),SUNFT(1)),(ASCH(11,39),SUNSD(1)),
     *  (ASCH(2,1),TITLE(1)),(ASCH(2,2),CHDATE),
     *  (ASCH(12,2),CHTIM)
C
C
C
C
C    MAKE 4 PATCHES
C
C        PROGRAM BY:      DR B. W. PATZ AND G. BECKER
C        PHONE:           677-7621
C        DATE:            6 JUN 1981
C
C
C
          JIIJ=99917
          DO 1668 I=1,3
          DO 1668 J=1,2
          X=RAN(JIIJ)
          IOFF1(I,J)=JIIJ
          X=RAN(JIIJ)
          IOFF2(I,J)=JIIJ
1668      CONTINUE
                  TYPE*,'READING NOISE'
                  READ(40,1000) ((JDAT1(I,J),I=1,512),J=1,256)
                  READ(41,1000) ((JDAT2(I,J),I=1,512),J=1,256)
                  READ(42,1000) ((JDAT3(I,J),I=1,512),J=1,256)
                  TYPE*,'DONE READING NOISE'
C
C-------------------------------------------------------------
C
```

```
C--------------------INPUT TEXT PARAMETERS--------------------
C
      TYPE*,'WOULD YOU LIKE THE IMAGE PARAMETER TEXT
     *  GENERATED?'
      TYPE*,' YES=1,NO=0'
      ACCEPT*,ITXTFLAG
      IF(ITXTFLAG.EQ.0) GOTO 20
      TYPE*,'ANSWER THE FOLLOWING QUESTIONS WITH A CHARACTER'
      TYPE*,' STRING OF LENGTH SPECIFIED BY (LENGTH)'
      TYPE*,'  '
      TYPE*,'ENTER TITLE (15)'
      ACCEPT 6,TITLE
      TYPE*,'ENTER TERRAIN NOISE PARAMETERS'
      TYPE*,'    SIZES (9)'
      ACCEPT 2, TERSZ
      TYPE*,'    TYPES (9)'
      ACCEPT 2, TERTP
      TYPE*,'    HORIZONTAL SCALES (9)'
      ACCEPT 2, TERHOR
      TYPE*,'    VERTICAL SCALES (9)'
      ACCEPT 2, TERVER
      TYPE*,'ENTER TREES/ROCKS NOISE PARAMETERS'
      TYPE*,'    SIZES (9)'
      ACCEPT 2, ROCSZ
      TYPE*,'    TYPES (9)'
      ACCEPT 2, ROCTP
      TYPE*,'    HORIZONTAL SCALES (9)'
      ACCEPT 2, ROCHOR
      TYPE*,'    VERTICAL SCALES (9)'
      ACCEPT 2, ROCVER
      TYPE*,'ENTER GRASS/LEAVES NOISE PARAMETERS'
      TYPE*,'    SIZES (9)'
      ACCEPT 2, GRASZ
      TYPE*,'    TYPES (9)'
      ACCEPT 2, GRATP
      TYPE*,'    HORIZONTAL SCALES (9)'
      ACCEPT 2, GRAHOR
      TYPE*,'    VERTICAL SCALES (9)'
      ACCEPT 2, GRAVER
      TYPE*,'ENTER FIELD SLOPE (8)'
      ACCEPT 3, FLDSL
      TYPE*,'ENTER FIELDS SIZE (9)'
      ACCEPT 2, FLDSZ
      TYPE*,'IS SUN SHADOW PRESENT IN THE DATA BASE? YES/NO'
      ACCEPT 1, SUNSH
      IF(SUNSH(1).EQ.'N') GOTO 20
      TYPE*,'ENTER SUN SHADOW FIRST FACTORS (8)'
      ACCEPT 3, SUNFT
      TYPE*,'ENTER SUN SHADOW SECOND FACTORS (7)'
      ACCEPT 5, SUNSO
C
```

```
C-----------------------------------------------------------------
C
1       FORMAT(3A1)
2       FORMAT(9A1)
3       FORMAT(9A1)
4       FORMAT(5A1)
5       FORMAT(7A1)
6       FORMAT(15A1)
C
C
C---------------INPUT SCENE PARAMETERS--------------------
C
C
 20     TYPE*,'TYPE XOFF-SET TO SCENE CENTER'
        ACCEPT*,IX
        TYPE*,'TYPE Y-OFFSET TO SCENE CENTER'
        ACCEPT*,JY
        TYPE*,'TYPE SAMPLE POINT SPACING,AN INTEGER'
        ACCEPT*,NCR
        TYPE*,'Type the polar sun angle from the Z axis
     *       (DEGREES)'
        ACCEPT*,TZDEG
        TZRAD=TZDEG/57.3
        SINZ=SIN(TZRAD)
        TYPE*,'Type the cylindrical sun angle from the X axis
     *       toward the Y axis (DEGREES)'
        ACCEPT*, TXDEG
C
C
        TXRAD=TXDEG/57.3
        C=COS(TZRAD)
        A=SINZ*COS(TXRAD)
        B=SINZ*SIN(TXRAD)
C
C         W1=W1*3
C         W =W *3
C
        SMARG=SQRT(A*A+B*B)
        SBOVA=B/A
        COVMARG=C/SMARG
        IDOV2=IDENSITY(2,2)/2
        KSI=SCALE(2,2)*IDENSITY(2,2)
        KSI4=3*IDENSITY(2,2)
        KSI5=4*IDENSITY(2,2)
        KSI7=5*IDENSITY(2,2)
        KSI8=6*IDENSITY(2,2)
C
        IBRFLD(1)=20000000
        IBRFLD(2)=60*SCALE(2,2) ! THIS SETS REACH BOUNDRY AT
C                                 ABOUT 90*2.5=210 UNITS OR 21
C                                 FEET.
```

```
      IBRFLD(3)=IBRFLD(2)-150
C
      DO 11 I=1,3
      DO 11 J=1,2
      X=IDENSITY(I,J)
11    DND1(I,J)=1./X
      S840VA=S80VA*3
      S870VA=S80VA*5
      TYPE *, 'X,Y=',IX,JY,' SCALE=',NCR,' NOISE?=',INOIS
              K11=-256
              K22=-256
            DO 100 J1=1,512
               J=(J1+K22)*NCR+JY
               DO 100 I1=1,512
                  I=(I1+K11)*NCR+IX
                  CALL ZRDAT5(*116)
116                     IF (IFIR.EQ.2) THEN
                        CALL ZRDAT5IR(*117)
                        ELSE IF (IFIR.EQ.5) THEN
                                 CALL ZRDAT55IR(*117)
                        ELSE
                                 TYPE *,'IFIR=',IFIR
                        ENDIF
 117                    CONTINUE
                        DO 153 KJX=1,3
                        IBUFX(I1,J1,KJX)=IRV(KJX)
 153                    CONTINUE
100           CONTINUE
1000  FORMAT(66A2)
      TYPE*,'STOP=1,CONTINUE=0'
      ACCEPT*,MI
      IF(MI.EQ.1) STOP
1734  TYPE*, 'SHALL WE MAKE A PICTURE? (YES=1)'
      ACCEPT *, IYES
      IF (IYES.NE.1) GO TO 373
      TYPE*, ' '
      TYPE*, 'DATA WILL BE OUTPUT TO THE DICOMED'
      CALL BPDICCAM
 373  TYPE*,'DO YOU WANT TO DO ANOTHER PICTURE?(YES=1)'
      ACCEPT*, IAN
      IF (IAN.EQ.1) GO TO 20
      STOP
 40   STOP
      END
```

APPENDIX DG

EGLDATA.FOR

```
C
C       EGLDATA.FOR
C
C
C
C
        SUBROUTINE ZRDATA(*)
        INCLUDE '4PCOM.FOR'
        INCLUDE '8PNOISE.FOR'
        COMMON/SUN/A,B,C,SMARG,SKYSCL,DIVSKY
        COMMON/DATABASE/N1024,IOFF1(3,2),IOFF2(3,2),SCALE(3,2),
     *  IBRFLD(3),SROVA,COVMARG,IDOV2,KSI,KSI4,KSI5,KSI7,KSI8,
     *  IDENSITY(3,2),SB4OVA,SB7OVA,DNO1(3,2),W1,W,IPIL,LSCAL,
     *  INAVSCAL
        INTEGER*2 ISAT,IDD,IFIR
        INTEGER*2 INDEXX,INDEXY,
     *  ISAM,INX,INY,KL1,KL2,KL3,KH1,KH2,KH3,IBL
        DIMENSION IOX(3,2),IOY(3,2),IYDEY(3,2),IXDEX(3,2)
        DIMENSION GGET(4),PARTIDA(2)
        DIMENSION  IDDX(3,2),IDDY(3,2),PARX(3,2),PARY(3,2)
        DIMENSION ISAV1(3,2),ISAV2(3,2),ISAV3(3,2),ISAV4(3,2)
        DIMENSION IVSAV(3,2,4)
        EQUIVALENCE (IVSAV,ISAV1),(IVSAV(1,1,2),ISAV2),
     *  (IVSAV(1,1,3),ISAV3),(IVSAV(1,1,4),ISAV4)
        EQUIVALENCE (GGET,X1X),(GGET(2),Y1Y),(GGET(3),X2X),
     *  (GGET(4),Y2Y),(PARTIDA,PARTIDATX),(PARTIDA(2),PARTIDATY)
        COMMON /GRND/ INX,INY
C
C BUILDINGS WITH ROADS AND 2 NOISE TYPES AND MULTIPLE RETURN
C
C IP := NADIR CENTRIC COORDINATES
C IPP := WORLD CENTRIC COORDINATES
C IPP = IP + IF
C IF IPP<0 ;IPNP=IPP
C  ELSE      IPNP=IPP+1
C
C
C       CALLING ROUTINE:  SPECIFIED DIGUED CONTROL ROUTINE
C
C ZRDAT5.FOR
```

```
C CREATED BY PHILIP GATT AND DR. B. W. PATZ
C 10/20/80
C SIN(X)*SIN(G(Y)) + NOISE AND LAKES
C G(Y) IS SMALL FOR Y < 3000

      ENTRY ZRDAT5(*)
      DATA C/.852870/,B/-.173648/,A/.492404/
      DATA SCALE/.03,.5,8.,.12,2.5,31/
      DATA W1/2.849003E-6/,W/8.950406E-6/    !W1=1/117000
      DATA IDENSITY/1,7,733,3,31,2483/
      DATA LSCAL/2/ , IWAVSCAL/47/
C IDENSITY(ITYPE,IVARTN) IS THE   WORLD POINTS WHICH MAP TO ONE
C NOISE POINT
C
C     SUN VECTOR FROM GROUND POINT TO SUN (X,A;Y,B)
C
C FOR STARTING XE LOCATION USE 2000
C YE SHOULD BE TESTED AT -10350
C     A=.866
C     B=-.25
C     C=.4330125
C     SEE ZRDATA4 FOR (ABC) DATA STATEMENT

C IDENSITY(3,2) AND SCALE(3,2) ARE LOCAL VARIABLES TO ZRDATA.
C IDENSITY(3,2) DEFINES THE SPREAD TO BE WITH EACH NOISE DATA
C FILE.
C JDAT1 IS SET UP TO HAVE AND AVERAGE PERIOD OF ABOUT 5 UNITS.
C JDAT1: POLAR FILTER 3 UNIFORM
C  JDAT2: POLAR FILTER 11 CUSP F=1
C   JDAT3: POLAR FILTER 25 PARABOLA F=.5
C  PERIOD = 1.8*FILTER
C JDAT2 HAS THE SAME PERIOD BUT HAS BEEN ANDED TO GIVE REGIONS
C WHERE NO VARIATION OCCURS BY THE ROUTINE CALLED PATCHES,THE
C NO VARIATION REGION HAS THE JDAT2 VALUE SET TO ZERO.
C  JDAT2 RANGES
C  FROM -128 TO 127 AS DO ALL THE BYTE JDAT* FILES.
C JDAT3 IS A TEXTURE NOISE FILE HAVING A NOISE PERIOD BETWEEN
C 3 AND 4.
C IF WE SCALE 1000 UNITS TO BE 100 FEET AND WE ASSUME THE
C FOLLOWING
C     VARIATIONS:
C
C     TERRAIN:  3000+ UNITS VARIATION = 300 FT. FOR ELEVATION
C                                       TO REPRESENT RUTS AND
C                                       GULLIES AND NORMAL
C                                       VARIATIONS
C     TREES:  256 UNITS VARIATIONS = 25.6 FT. FOR CLOSE PACKED
C                                       TREES ~ 60 FT INSIDE
C                                       FOREST.
C     BRANCHES, ETC.:  30 UNITS = 3 FT. BRANCH VARIATIONS.
C     GROUND:  5 UNITS VARIATION = 6 IN. FOR FIELDS.
```

```
C
C THEREFORE, THE ELEVATION SCALES BECOME:
C                SCALE(1) = 0.03 ; .03*256=~7.5>6''
C                SCALE(2) = 2.5 ; 2.5*256=~630 ~ 63'
C                SCALE(3) = 59 ; 59*256=~15000 ~ 1500'
C
C IF GROUND SWELLS ARE ABOUT 5 MILES APART THEN FOR THE SINE
C WAVE:
C                W=PI/117,000.
C IF WE LET THE AMPLITUDE OF ROLLING HILLS BE 12,000 UNITS, WITH
C A 5,500 UNIT OFFSET, THEN THE HILL PEAKS ARE ABOUT 1,750 FT.
C IF TERRAIN VARIES FROM HIGH TO LOW IN ABOUT ONE-FIFTH TO
C       ONE-TWENTIETH OF THE PEAK ELEVATION, THEN:
C            IDENSITY(3) = (3000*10)/3 = 10,000 SAY 9973 (PRIME).
C IF TREES VARY FROM HIGH TO LOW IN ABOUT 30 FT. OR 300 UNITS,
C THEN:
C                IDENSITY(2) = 300/3 = 100 SAY 111 (PRIME)
C IF THE BRANCH DIMENSIONS ARE ABOUT ONE-HALF TO ONE-FIFTH THEIR
C LENGTH, THEN
C                IDENSITY(1) = (30/3)/3 = 3
C IDENSITY(4) FOR GRASS IS 1 AND THEREFORE NO IDENSITY(4) IS
C NEEDED.
C FINALLY, TO GIVE EFFECTS OF LEAVES, ETC. A NOISE TEXTURE AT
C       FREQUENCY = 3 UNITS IS ADDED TO THE FIELDS AND TREES.
C COMPUTE THE HEIGHT
C
C--------------------CALCULATE TRIG HEIGHT ----------------
        IPP2=IPP(2)
        XNUM=1+(W1*IPP2)**2
        DENOM=1./(1.+ABS(W1*IPP2))
        WIPP1=W*IPP(1)
        SWIPP=13500*SIN(WIPP1)
        H=(XNUM*DENOM -.85)*2*3.14159*.6
        SNHH=SIN(H)
        XZ=SWIPP*SNHH+5500
C-----------------------------------------------------------
C
        FIELD=1.
        IFIELD=0
C               IBENF=0
C       DO 1978 IBEN1=1,3
C       DO 1978 IBEN2=1,2
C         IF(IDDX(IBEN1,IBEN2).GT.IDENSITY(IBEN1,IBEN2)) IBENF=1
C         IF(-IDDX(IBEN1,IBEN2).GT.IDENSITY(IBEN1,IBEN2))IBENF=1
C         IF(IDDY(IBEN1,IBEN2).GT.IDENSITY(IBEN1,IBEN2)) IBENF=1
C         IF(-IDDY(IBEN1,IBEN2).GT.IDENSITY(IBEN1,IBEN2))IBENF=1
C1978   CONTINUE
C       IF(IBENF.EQ.1)  THEN
C         TYPE*,'ENTRY'
C       TYPE*,'IDDX,IDDY=',IDDX,IDDY,' IDENSITY=',IDENSITY
C         TYPE*,',IBENF:IPP=',IPP,'  IIX,IIY=',IIX,IIY
```

```
C       ENDIF
        DO 1111 ITYPE=3,1,-1
C         IDDX(ITYPE,1)=0
C         IDDX(ITYPE,2)=0
          IDDY(ITYPE,1)=0
C         IDDY(ITYPE,2)=0
        DO 1221 IVARTN=2,1,-1
        INT1=1022*IDENSITY(ITYPE,IVARTN) ! MODULO VALUE FOR
                                           NOISE ARRAY
        INT2=512*IDENSITY(ITYPE,IVARTN)
C
C--------------------CALCULATE NOISE INDEX VARIABLE----------
                IIX=IABS(MOD((IPP(1)-IOFF1(ITYPE,IVARTN)),INT1))
                IIY=IABS(MOD((IPP(2)-IOFF2(ITYPE,IVARTN)),INT1))
C       IF(IPP(1).EQ.-163750) THEN
C         TYPE*,'IIX,IIY=',IIX,IIY
C         TYPE*,'BEGIN: IDDX=',IDDX
C         TYPE*,'BEGIN: IDDY=',IDDY
C       ENDIF
        IF(IIX .LT. INT2) THEN          ! 1
                IQX(ITYPE,IVARTN)= 1+IIX/IDENSITY(ITYPE,IVARTN)
                IXDEX(ITYPE,IVARTN)=IQX(ITYPE,IVARTN)+1
                IF(IQX(ITYPE,IVARTN).EQ.512) IXDEX(ITYPE,IVARTN)
     *             =511
           ELSE                                         ! 1
                IQX(ITYPE,IVARTN)= 1023-IIX/IDENSITY(ITYPE,
     *          IVARTN)
                IXDEX(ITYPE,IVARTN)=IQX(ITYPE,IVARTN)-1
           ENDIF                                        ! 1
                IDDX(ITYPE,IVARTN)=MOD(IIX,IDENSITY(ITYPE,
     *          IVARTN))
                IDDY(ITYPE,IVARTN)=MOD(IIY,IDENSITY(ITYPE,
     *          IVARTN))
C       IF(IPP(1).EQ.-163750) THEN
C         TYPE*,'SOURCE:IDDY=',IDDY
C         TYPE*,'SOURCE: ITYPE,IVARTN=',ITYPE,IVARTN
C         TYPE*,'SOURCE: IIX,IIY=',IIX,IIY
C         TYPE*,'SOURCE: IDENSITY=',IDENSITY
C       ENDIF
        IF( IIY .LT. INT2) THEN         ! 1
                IQY(ITYPE,IVARTN)= 1+IIY/IDENSITY(ITYPE,IVARTN)
                IYDEY(ITYPE,IVARTN)=IQY(ITYPE,IVARTN)+1
                IF(IQY(ITYPE,IVARTN).EQ.512) IYDEY(ITYPE,IVARTN)
     *          =511
           ELSE                                 ! 1
                IQY(ITYPE,IVARTN)= 1023 - IIY/IDENSITY(ITYPE,
     *          IVARTN)
                IYDEY(ITYPE,IVARTN)=IQY(ITYPE,IVARTN)-1
           ENDIF                                ! 1
C-------------------------------------------------------------
C
```

```
            IF(IDDX(ITYPE,IVARTN).LT.0) THEN
              TYPE*,'IDDX<0=',IDDX,'  AFTER IXDEX,IYDEY'
              TYPE*,'IDDX<0,IPP=',IPP,'  IIX,IIY=',IIX,IIY
              TYPE*,'INT1,INT2=',INT1,INT2
            ENDIF
            IF(IDDY(ITYPE,IVARTN).LT.0) THEN
              TYPE*,'IDDY<0=',IDDY,'  AFTER IXDEX, IYDEY'
              TYPE*,'IDDY<0,IPP=',IPP,'  IXX,IIY=',IIX,IIY
              TYPE*,'INT1,INT2=',INT1,INT2
            ENDIF
            ILP=ITYPE
C           IF(IPP(1).EQ.-163750)  THEN
C           TYPE*,'IDDY AT DX,Y=',IDDY
C             TYPE*,'ATDX,Y: ITYPE,IVARTN=',ITYPE,IVARTN
C           ENDIF
            IF (ITYPE.EQ.2) THEN ! 2
            INTX1=INT1*25
            INTX2=INT2*25
            INTX3=25*IDENSITY(2,2)
            IIXX=IABS(MOD((IPP(1)-567397),INTX1))
            IIYY=IABS(MOD((IPP(2)-31797143),INTX1))
            IF (IIXX.LT.INTX2) THEN ! 6
                    IQXX=1+IIXX/INTX3
            ELSE ! 6
                    IQXX=1023-IIXX/INTX3
            ENDIF ! 6
            IF (IIYY.LT.INTX2) THEN ! 7
                    IQYY=1+IIYY/INTX3
            ELSE ! 7
                    IQYY=1023-IIYY/INTX3
            ENDIF ! 7
            IJERD=IDAT(IQXX,IQYY,2)
            IF((IJERD.LT.77).OR.(IABS(ISAV2(3,2)).GT.31).OR.
        *   (IABS(ISAV3(3,2)).GT.31)) THEN ! 8 CALCULATE TREES
C                                          (400=IDENSITY(3)/SCALE(3)
            IFIELD=0
            FIELD=1.
            GOTO  161
              ELSE ! 8  FIELD, NOT TREES
C IF NOT SATISFIED THEN WE CALCULATE A FIELD, SO KZ GETS NO
C CHANGE.
C
            IFIELD=1
            FIELD=0.33
            IVARSIR=IVARTN
            GOTO 1222
            ENDIF  !8  FIELD/FORREST=TREES
            ELSE              ! 2  TEST FOR FIELDS
 161        ISAV1(ITYPE,IVARTN)=IDAT(IQX(ITYPE,IVARTN),IQY(ITYPE,
        *   IVARTN),ITYPE)
            ISAV2(ITYPE,IVARTN)=IDAT(IXDEX(ITYPE,IVARTN),IQY(ITYPE,
```

```
     *   IVARTN),ITYPE)-ISAV1(ITYPE,IVARTN)

         ISAV3(ITYPE,IVARTN)=IDAT(IDX(ITYPE,IVARTN),IYDEY(ITYPE,
     *   IVARTN),ITYPE)-ISAV1(ITYPE,IVARTN)

         ISAV4(ITYPE,IVARTN)=IDAT(IXDEX(ITYPE,IVARTN),IYDEY(ITYPE
     *   ,IVARTN),ITYPE)-ISAV3(ITYPE,IVARTN)-ISAV2(ITYPE,IVARTN)-
     *   ISAV1(ITYPE,IVARTN)

C        DNO1(ITYPE,IVARTN)=1./IDENSITY(ITYPE,IVARTN)
C        DNO1 IS SET UP IN BPNEWIC DURING INITIALIZATION

         DELZ=ISAV1(ITYPE,IVARTN)+((IDDX(ITYPE,IVARTN)*ISAV2
     *   (ITYPE,IVARTN)+IDDY(ITYPE,IVARTN)*ISAV3(ITYPE,IVARTN))
     *   +DNO1(ITYPE,IVARTN)*IDDX(ITYPE,IVARTN)*IDDY(ITYPE,
     *   IVARTN)*ISAV4(ITYPE,IVARTN))*DNO1(ITYPE,IVARTN)

         DELZ=DELZ*SCALE(ITYPE,IVARTN)
         IF(IDDX(ITYPE,IVARTN).LT.0) THEN
           TYPE*,'IDDX<0=',IDDX,'  AT DELZ'
           TYPE*,'IDDX<0,IPP=',IPP,'  IIX,IIY=',IIX,IIY
           TYPE*,'INT1,INT2=',INT1,INT2
         ENDIF
         IF(IDDY(ITYPE,IVARTN).LT.0) THEN
           TYPE*,'IDDY<0=',IDDY ,'  AT DELZ'
           TYPE*,'IDDY<0,IPP=',IPP,'  IIX,IIY=',IIX,IIY
           TYPE*,'INT1,INT2=',INT1,INT2
         ENDIF
C
C
C EDGE OF FORREST BASED ON IJERD
         IF(ITYPE.EQ.2) THEN
           IF(IJERD.LT.77) THEN
             IF(IABS(ISAV2(3,2)).LE.31) THEN
                 IF(IABS(ISAV3(3,2)).LE.31) THEN
                   ILOW =-IDENSITY(2,2)
                   IHIGH =-ILOW
                   IINCRT = IHIGH+IHIGH
           DO  1557  IKRD = ILOW,IHIGH,IINCRT
         IIX1=IABS(MOD((IPP(1)-567397+IKRD),INTX1))
           IF(IIX1.LT.INTX2) THEN
                 IOXX1= 1+IIX1/INTX3
         ELSE
                 IOXX1= 1023-IIX1/INTX3
         ENDIF
           DO  1557  JKRD = ILOW,IHIGH,IINCRT
         IIY1=IABS(MOD((IPP(2)-31797143+JKRD),INTX1))
         IF(IIY1.LT.INTX2) THEN
                 IOYY1= 1+IIY1/INTX3
         ELSE
                 IOYY1 = 1023-IIY1/INTX3
```

```
            ENDIF
            IERDTST=IDAT(IQXX1,IQYY1,2)
            IF(IERDTST.GE.77) THEN  ! FIELD NEARBY
              IT1X=IABS(IIX1-IIXX)
              JT1Y=IABS(IIY1-IIYY)
              XX=IDENSITY(2,2)
                IF(IT1X.LT.JT1Y) THEN
                    X=I11X/XX
                ELSE
                    X=JT1Y/XX
                ENDIF
            ENDIF
1557        CONTINUE
            DELZ=X*DELZ
                    ENDIF
                 ENDIF
                ENDIF
            ENDIF
            XZ=XZ+DELZ*FIELD
            ENDIF ! 2
1221        CONTINUE
1222                IPIL=ITYPE
            IF(XZ.LT.IBRFLD(ITYPE)) THEN ! 9
C                   IBRFLD(3)=70*SCALE(2)-100; BEACH      IPIL=3
C                   IBRFLD(2)=70*SCALE(2); ROCKS AT BEACH
C                                                NEUTRAL=IR/2   IPIL=2
C                   IBRFLD(1)=20000000; FIELDS AND FOREST    IPIL=1
                    GO TO 101
            ENDIF ! 9
C           IF(IPP(1).EQ.-163750) THEN
C             TYPE*,'IDDY BEFOR 1111=',IDDY
C             TYPE*,'BEFORE 1111: ITYPE,IVARIN=',ITYPE,IVARIN
C           ENDIF
1111        CONTINUE
101         CONTINUE
            IF(IPIL.LT.0)THEN ! 10
            TYPE*,'END ITYPE LOOP 16800'
            TYPE*,'ISAV1=',ISAV1
            TYPE*,'ISAV2,DZ/DX=',ISAV2
            TYPE*,'ISAV3,DZ/DY=',ISAV3
            TYPE*,'IQX,IXDX=',IQX,IXDEX
            TYPE*,'IQY,IYDEY=',IQY,IYDEY
            TYPE*,'IDDX,IDDY=',IDDX,IDDY
            TYPE*,'IVSAV(ITYPE=3,SAV=4)=',IVSAV
            ENDIF ! 10
            IZ=XZ
            IFIR=2
            IF(IZ.LT.-50) THEN ! 11
                    IFIR=5
1212                        IZ=-50
            ENDIF ! 11
```

```
      IP(3)=IZ
            IBENF=0
      DO 1098 IBEN1=1,3
      DO 1098 IBEN2=1,2
        IF(IDDX(IBEN1,IBEN2).GT.IDENSITY(IBEN1,IBEN2)) IBENF=1
        IF(-IDDX(IBEN1,IBEN2).GT.IDENSITY(IBEN1,IBEN2))IBENF=1
        IF(IDDY(IBEN1,IBEN2).GT.IDENSITY(IBEN1,IBEN2)) IBENF=1
        IF(-IDDY(IBEN1,IBEN2).GT.IDENSITY(IBEN1,IBEN2))IBENF=1
1098  CONTINUE
      IF(IBENF.EQ.1)  THEN
      TYPE*,'IDDX,IDDY=',IDDX,IDDY,' IDENSITY=',IDENSITY
        TYPE*,',IBENF;IPP=',IPP,'  IIX,IIY=',IIX,IIY
      ENDIF
      RETURN 1
      ENTRY ZRDATSIR(*)
C COMPUTE THE REFLECTANCE
      DO 2211 ITYPE=IPIL,3
      ISTRT=1
      IF(ITYPE.EQ.2) ISTRT=IVARSTR
      DO 2211 IVARTN=ISTRT,2
      IF((ITYPE.EQ.2).AND.(IFIELD.EQ.1)) THEN
        CVER=0
        GOTO 2210
      ELSE
         CVER=SCALE(ITYPE,IVARTN)
      ENDIF
      XTEM=ISAV4(ITYPE,IVARTN)*DNO1(ITYPE,IVARTN)
      PARTIDATX=(ISAV2(ITYPE,IVARTN)+IDDY(ITYPE,IVARTN)*XTEM)
     *  *DNO1(ITYPE,IVARTN)
      PARTIDATY=(ISAV3(ITYPE,IVARTN)+IDDX(ITYPE,IVARTN)*XTEM)
     *  *DNO1(ITYPE,IVARTN)
      X1X=ABS(PARTIDATX)
      Y1Y=ABS(PARTIDATY)
C
C     THIS COMPUTES THE PARTIALS FOR THE FIELDS IF IFIELD=1
C     AND OTHER REGIONS IF IFIELD=0
C
            IF(ITYPE.NE.3)  THEN
            IMULL=ITYPE*IDENSITY(ITYPE,IVARTN)+2
                  IF(NCR.GT.IMULL) THEN
                    PARTIDATX=PARTIDATX*IMULL/NCR
                    PARTIDATY=PARTIDATY*IMULL/NCR
                  ENDIF
            ENDIF
      CVER1 =IPP(1)-IOFF1(ITYPE,IVARTN)
      CVER2= IPP(2)-IOFF2(ITYPE,IVARTN)
2210  CONTINUE
      PARX(ITYPE,IVARTN)=PARTIDATX*SIGN(CVER,CVER1)
      PARY(ITYPE,IVARTN)=PARTIDATY*SIGN(CVER,CVER2)
2211  CONTINUE
      PARTIDATX=0
```

```
      PARTIDATY=0
      DO 2215 I2=1,2
      DO 2215 IPAR=IPIL,3
              PARTIDATX=PARTIDATX+PARX(IPAR,I2)
              PARTIDATY=PARTIDATY+PARY(IPAR,I2)
2215  CONTINUE
      PARTH1=2*.6*2*3.14159*W1*W1*IPP2*DENOM
      PARTH2=XNUM*DENOM**2*W1*.6*2*3.14159
      IF(IPP2 .GT. 0) THEN
              PARTHY=PARTH1-PARTH2
      ELSE IF (IPP2 .LT. 0) THEN
              PARTHY=PARTH1 + PARTH2
      ELSE
              PARTHY=0
      ENDIF
 2227 PARTRIGX=13500*W*COS(WIPP1)*SNHH
      PARTRIGY=SWIPP*COS(H)*PARTHY
      PARTZX= PARTRIGX + PARTIDATX
      PARTZY=  PARTRIGY + PARTIDATY
      REMAG=SQRT(1 + PARTZX**2 + PARTZY**2)
      IR=255*(-A*PARTZX-B*PARTZY+C)/REMAG
      X1X=ABS(PARTIDATX)
      Y1Y=ABS(PARTIDATY)
      X2X=ABS(PARTRIGX)*10
      Y2Y=ABS(PARTRIGY)*10
      IF(IR.LT.3)IR=3
C     SUN SHADOWING SHOULD BE BASED AS MUCH AS POSSIBLE ON
C     MAJOR FEATURE EXTRATION FOR AN APPROXIMATION OR ON EXACT
C     RAYTRACING
C
C     IQX(3,2),IQY(3,2)  ARE INDEX POINTERS FOR TREE FEATURES
C     IXDEX(2),IYDEY(2) ARE DIRECTION ON DATA BASE
C
C-----------------------------------------------------------------
C     IF IR.GT.400 THEN SUN SHADOWING IS BYPASSED
C     IF IR.GT.40 THEN SUN SHADOW IS CALCULATED
C
      IF (IR.GT.400)  THEN
C     THIS CODE VALID FOR ABS(A).GE.ABS(B)
      I4X=IQX(2,2)+4
      I4X1=I4X+1
         IF(I4X.GE.512) I4X=1023-I4X
         IF(I4X1.GE.512) I4X1=1023-I4X1
      I4YF=(IQY(2,2)*IDENSITY(2,2)+SB40VA*IDDY(2,2))/
     *  IDENSITY(2,2)
C
C=================================================================
C SHOULD IT BE SB40VA+IDDY ,OR SHOULD IT BE SB40VA*IDDY ?
C=================================================================
C
      I4Y=I4YF
```

```
          I4Y1=I4Y+1
             IF(I4Y.GE.512) I4Y=1023-I4Y
             IF(I4Y1.GE.512) I4Y1=1023-I4Y1
          IY4J=IDDY(2,2)+SBOVA*(KSI4-IDDX(2,2))
          IDELTAY=IY4J-I4YF*IDENSITY(2,2)
          INOW=IDAT(I4X,I4Y,2)
          INOW1=IDAT(I4X,I4Y1,2)
          IZSHD=SCALE(2,2)*(INOW+(INOW1-INOW)*DNO1(2,2)*IDELTAY)-
      *   XZ
          KIDDTES=COVMARG*(KSI5-IDDX(2,2))+.5
             IF(IZSHD.LT.KIDDTES) THEN                !3
C    NO SUN SHADOWING BY THIS HILL, TEST FOR NEXT CONDITION.
                   IF(IDOV2.GT.IDELTAY)THEN
C                         USE IDELTAZ/IDELTAX AT (I4X,I4Y) POINT
                          IZSHD=IZSHD+KSI*(IDAT(I4X1,I4Y,2)-INOW)
                   ELSE
C                         USE IDELTAZ/IDELTAX AT (I4X,I4Y1)
                          IZSHD=IZSHD+KSI*(IDAT(I4X1,I4Y1,2)-
      *                        INOW1)
                   ENDIF
             IF(IZSHD.LT.KIDDTES) THEN                !2
C                  NO SUN SHADOWING YET, BY THESE TWO TESTS
                   I4X=IQX(2,2)+7           !NO TEST 7 UNITS OVER
                   I4X1=I4X+1
                      IF(I4X.GE.512) I4X=1023-I4X
                      IF(I4X1.GE.512) I4X1=1023-I4X1
                   I4YF=IQY(2,2)+SBTOVA
C
C  SEE LINE10300 ABOVE
C
                   I4Y=I4YF
                   I4Y1=I4Y+1
                      IF(I4Y.GE.512) I4Y=1023-I4Y
                      IF(I4Y1.GE.512) I4Y1=1023-I4Y1
                   IY4J=IDDY(2,2)+SBOVA*(KSI7-IDDX(2,2))+.5
                   IDELTAY=IY4J-I4YF*IDENSITY(2,2)
                   INOW=IDAT(I4X,I4Y1,2)
                   INOW1=IDAT(I4X,I4Y1,2)
                   IZSHD=SCALE(2,2)*(INOW+(INOW1-INOW)*DNO1(2,2)*
      *                 IDELTAY)-XZ
                   KIDDTES=COVMARG*(KSI8-IDDX(2,2))+.5
             IF(IZSHD.LT.KIDDTES)THEN                 !1
C                  STILL NO SUN SHADOWING, TEST FOR LAST CONDITION
                      IF(IDOV2.GT.IDELTAY) THEN
C                           USE IDELTAZ/IDELTAX FOR (I4X,I4Y)
                            IZSHD=IZSHD+KSI*(IDAT(I4X1,I4Y,2)-INOW)
                      ELSE     !USE IDELTAZ/IDELTAX AT (I4X,I4Y1)
                            IZSHD=IZSHD+KSI*(IDAT(I4X1,I4Y1,2)-
                                 INOW1)
                      ENDIF
          IF(IZSHD.LT.((KSI8-IDDX(2,2))*COVMARG)) THEN    !0
```

```
C                   NO SUN SHADOWING BY THIS APPROXIMATION
                    GO TO 12435
            ENDIF                   !0
            ENDIF                   !1
            ENDIF                   !2
            ENDIF                   !3
C           SUN SHADOWING
C
            IR=35+IR/8
12435       CONTINUE
            ENDIF
C           END SUN SHADOWING AND SUN ANGLE APPROXIMATIONS
C
            IF(IPIL.EQ.1)  THEN     !TREES AND SHRUBS
C           NEED SPECKLE ON OUTLINE TO MAKE PATTERN
C           USE IDAT(3)
            IIX=IABS(MOD(IPP(1),1022))
            IIY=IABS(MOD(IPP(2),1022))
                    IF(IIX.LT.512)THEN
                            IQXX=1+IIX
                    ELSE
                            IQXX=1023-IIX
                    ENDIF
                    IF(IIY.LT.512)THEN
                            IQYY=1+IIY
                    ELSE
                            IQYY=1023-IIY
                    ENDIF
                            NNV=IR/3/NCR
                    IF(IDAT(IQXX,IQYY,1).GE.0)THEN
                            IR=IR+NNV
                            IF(IR.GT.255)IR=255
                    ELSE
                            IR=IR-NNV
                    ENDIF
C   SPECKLE TEXTURE FOR TREES AND FINE DETAIL; NOISE FILTER=3
            IF (IFIELD.EQ.1)THEN
C           FIELDS=GOLD; NEEDS NEUTRAL APPROX. IR/4
                    IRV(1)=IR
                    IRV(2)=IR
                    IRV(3)=0
            ELSE
C           TREES=GREEN/BROWN; NEEDS NEUTRAL APPROX. IR/4
                    IRV(1)=IR
                    IRV(2)=(2*IR+2)/3
                    IRV(3)=-IRV(2)+1+IR
            ENDIF
            ELSE IF(IPIL.EQ.2)  THEN        !ROCKS AND TREES
               IRV(1)=IR
               IRV(2)=(IR+1)/2
               IRV(3)=IRV(2)
```

```
      ELSE                                    ! SHORE LINE
         IRV(1)=IR                            !IPIL=3 !GREEN
         IRV(2)=IRV(1)                                !RED
         IRV(3)=IRV(1)                        !BLUE
      ENDIF

      RETURN 1
      ENTRY ZRDAT55IR(*)
C     COMPUTE SMALL WAVES ON LAKE
C
C     IF(IDAT(IQX(2),IQY(2),2).GT.-32) THEN
C     CODE NEGLECTS PARTIALS OF IDAT COMPARED TO TRIG TERMS
C           PARTZX=3*3.14159/24*(COS(IPH+(3*IPP(1)+4*IPP(2))
C     *           *2*3.14159/120)+COS(IPH+(3*IPP(1)+4*
C     *           IPP(2))*2*3.14159/60))*EXP(-IP(2)*.0001)
C           PARTZY=1.33333*PARTZX
            IIX=IABS(MOD(((IPP(1)+(IPP(2)/3))/IWAVSCAL),
     *          1022*LSCAL))
            IIY=IABS(MOD((IPP(2)-(IPP(1)/3)),1022*LSCAL))
             IF(IIX.LT.512*LSCAL) THEN
                   IAX=1+IIX/LSCAL   !IQX
                   IBX=1+IAX   !IXDEX
                   IF(IAX.EQ.512) IBX=511
             ELSE
                   IAX=1023-IIX/LSCAL
                   IBX=IAX-1
             ENDIF
               IF(IIY.LT.512*LSCAL) THEN
                   IAY=1+IIY/LSCAL          !IQY
                   IBY=IAY+1          !IYDEY
                   IF(IAY.EQ.512)  IBY=511
               ELSE
                   IAY=1023-IIY/LSCAL
                   IBY=IAY-1
               ENDIF
            IS1=IDAT(IAX,IAY,3)
            IS2=IDAT(IBX,IAY,3) -IS1
            IS3=IDAT(IAX,IBY,3) -IS1
            X1X=ABS(PARTZX)
            Y1Y=ABS(PARTZY)
        IF(NCR.GT.LSCAL) THEN
            PARTZX=IS2/NCR
            PARTZY=IS3/NCR
      ELSE
            PARTZX=IS2
            PARTZY=IS3
        ENDIF
      PARTZX=DN01(2,2)*PARTZX*3
      PARTZY=DN01(2,2)*PARTZY*3
            REMAG=SQRT(1+PARTZX**2+PARTZY**2)
C     REMAG=SQRT(1 + PARTZX**2 + PARTZY**2)
```

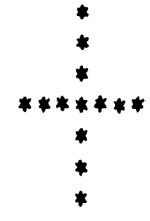
```
      IR=255*(-A*PARTZX -B*PARTZY + C)/REMAG
      IIX=IABS(MOD((IPP(1)),1022))
      IIY=IABS(MOD((IPP(2)),1022))
            IF(IIX.LT.512)THEN
                  IQXX=1+IIX
            ELSE
                  IQXX=1023-IIX
            ENDIF
            IF(IIY.LT.512)THEN
                  IQYY=1+IIY
            ELSE
                  IQYY=1023-IIY
            ENDIF
      IF(IR.LT.120) THEN
                  NNV=40/NCR
            ELSE
                  NNV=IR/(3*NCR)
            ENDIF
            IF(IDAT(IQXX,IQYY,1).GE.0)THEN
                  IR=IR+NNV
            ELSE
                  IR=IR-NNV
            ENDIF
      IF(IR .LT.3) IR=3
      IF(IR .GT. 255) IR=255
      IRV(1)=2*IR/3
      IRV(2)=0
      IRV(3)=IR
      RETURN 1
C
C CREATED BY PHILIP GATT AND DR. J. W. PATZ
C 10/20/80
C SIN(X)*SIN(G(Y)) + NOISE
C G(Y) IS SMALL FOR Y < 3000

      ENTRY ZRDAT6(*)
C IDENSITY(1) IS THE   WORLD POINTS WHICH MAP TO ONE NOISE POINT
C     A=.866
C     B=-.25
C     C=.4330125
      A=.866
      B=0
      C=.5
      RETURN1
      END
```

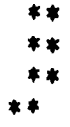
APPENDIX OH

TEXT CHARACTER SET

ICH(-,-,1)=



ICH(-,-,2)=



ICH(-,-,3)=



ICH(-,-,4)=



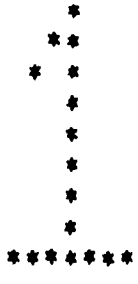
ICH(-,-,5)=



ICH(-,-,6)=



ICH(-,-,7)=



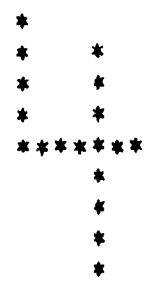
ICH(-,-,8)=



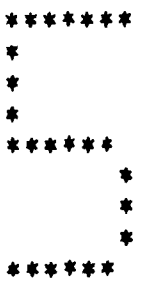
ICH(-,-,9)=



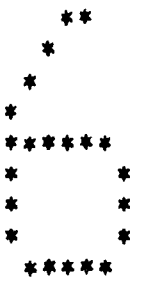
ICH(-,-,10)=



ICH(-,-,11)=



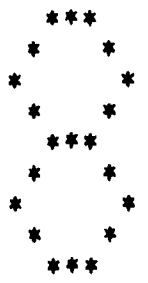
ICH(-,-,12)=



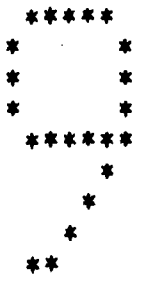
ICH(-,-,13)=



ICH(-,-,14)=



ICH(-,-,15)=



ICH(-,-,16)=



ICH(-,-,17)=



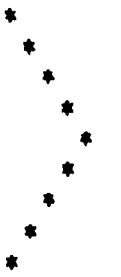
ICH(-,-,18)=



ICH(-,-,19)=



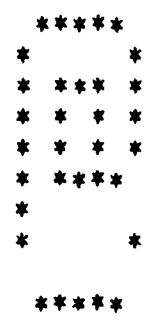
ICH(-,-,20)=



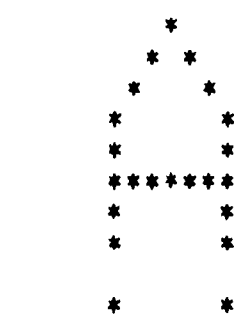
ICH(-,-,21)=



ICH(-,-,22)=



ICH(-,-,23)=



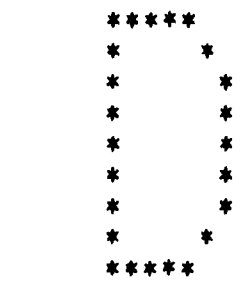
ICH(-,-,24)=



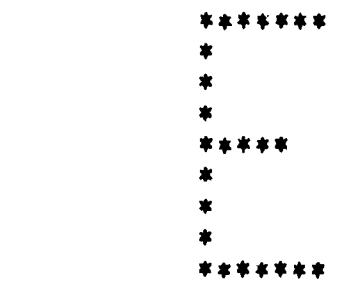
ICH(-,-,25)=



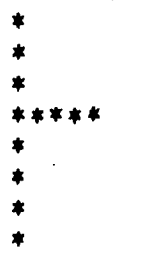
ICH(-,-,26)=



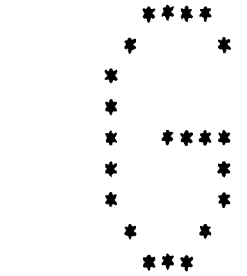
ICH(-,-,27)=



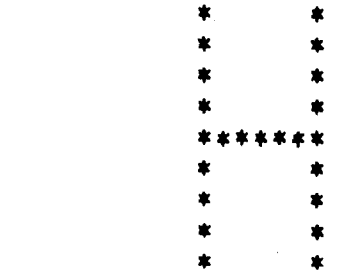
ICH(-,-,28)=



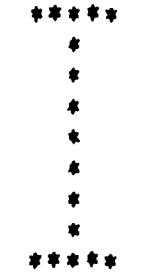
ICH(-,-,29)=



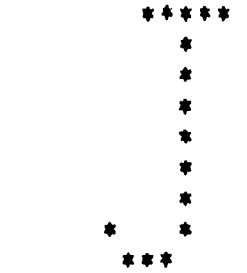
ICH(-,-,30)=



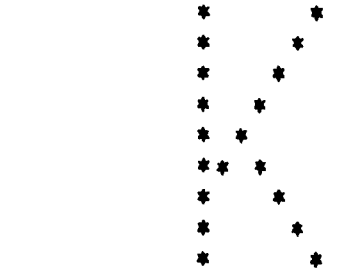
ICH(-,-,31)=



ICH(-,-,32)=



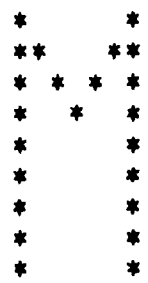
ICH(-,-,33)=



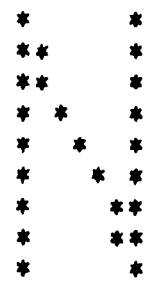
ICH(-,-,34)=



ICH(-,-,35)=



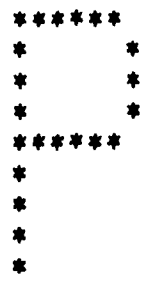
ICH(-,-,36)=



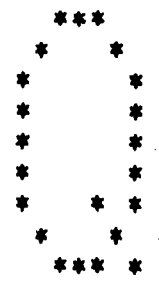
ICH(-,-,37)=



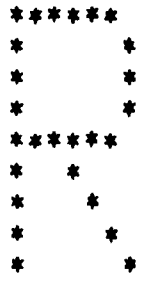
ICH(-,-,38)=



ICH(-,-,39)=



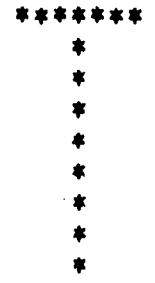
ICH(-,-,40)=



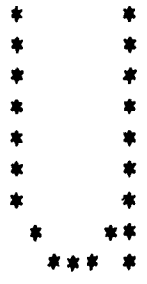
ICH(-,-,41)=



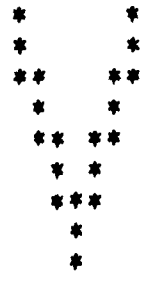
ICH(-,-,42)=



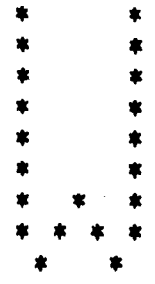
ICH(-,-,43)=



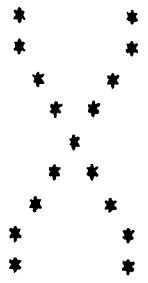
ICH(-,-,44)=



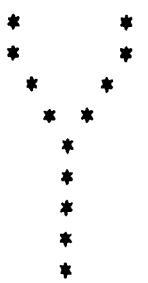
ICH(-,-,45)=



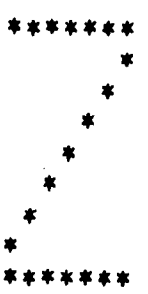
ICH(-,-,46)=



ICH(-,-,47)=



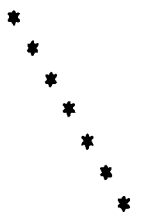
ICH(-,-,48)=



ICH(-,-,49)=



ICH(-,-,50)=

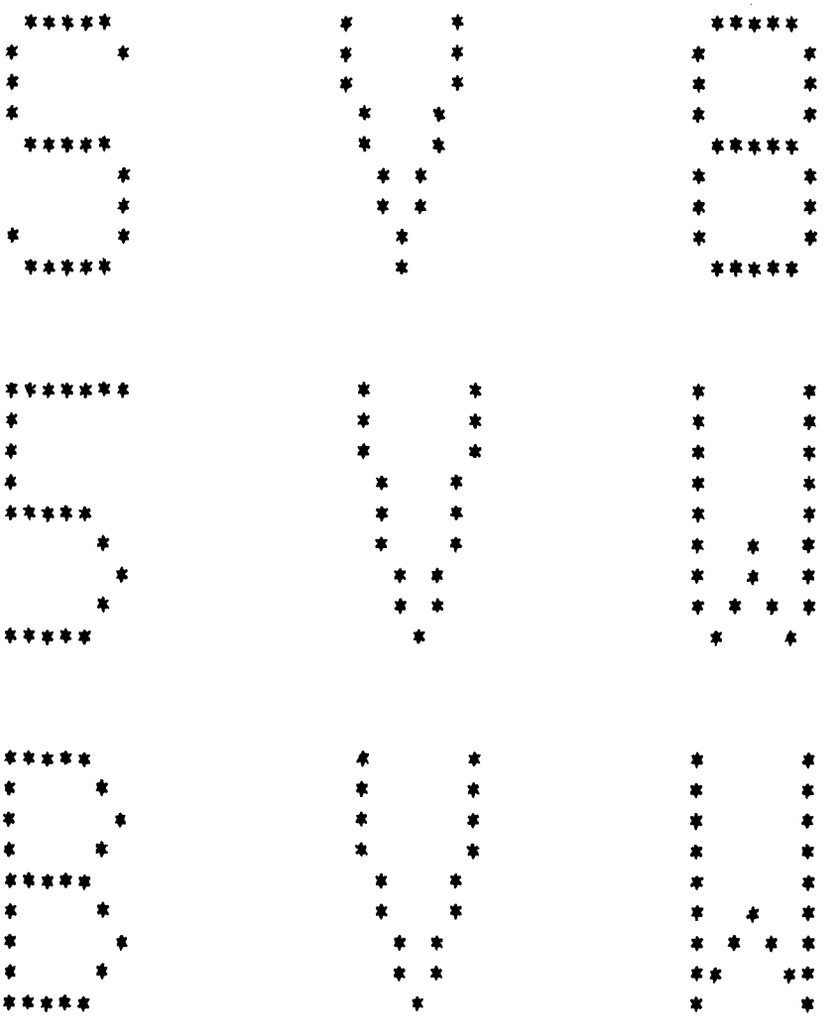


ICH(-,-,51)=



APPENDIX DI

ALTERNATE CHARACTER CONSTRUCTIONS



APPENDIX EA

FIRST GENERATION REALSCAN SOFTWARE

Appendices EB through EL contain the routines that are considered the first generation of REALSCAN.

APPENDIX EB

PGBUF.FOR

```
C       PGBUF.FOR APRIL 22 1981

        INTEGER*2 ITEMBUF(512,512)
        COMMON/BUF/ ITEMBUF
```

APPENDIX EC

PGCOMDAT.FOR

```
C       PGCOMDAT.FOR
C
C       THIS IS A COMMON BLOCK  FOR THE ROUTINES WHICH GENERATE
C       A PICTURE OF A 3 DIMENSIONAL SCENE
C       THESE ROUTINES ARE MAIN3D,SLCREATE,VIS,SCAN,ZRDATA,PROJ4
C       FINFIL, GETSCENE
C       LATEST REVISION DATE:   APRIL 22 1981

        DIMENSION IE(3),IOUT1(256),IF(2),IP(3)

        BYTE ICNT(512,512),IDAT(512,512,3)

        INTEGER*2 IOUT2(40,40),IXBS,IYBS,ISCRN(2),NCR,INT2(512,
     *  256),JDAT1(512,256),JDAT2(512,256),JDAT3(512,256)
        COMMON/VISIB/IPP(2),IPNP(2),NPN(2),IQ(2),LOW(3),NCR,
     *  ICODE,IOTHR,ICASE,INC,INCIO,IEEE,IXY2,IDXY,JXY2
        COMMON/INTG/IXP,IYP,IZP,IXE,IYE,IZE,IROT(3,3),ISCALE
     1  ,IXH,IYH,IZH,IXBS,IYBS,IXPINC,IYPINC,IZPINC,IRBW
     2  ,IDIV1,ICOMD,IFILTER1,IFILTER2,IRUN,IOUT1,IFIR,NC
     3  ,NPL,NX,NY,LX,LY,IAXIS,NPIX,IUS,IVS,IWS,LINE
     4  ,IFILDIM,IFLDMD2
        COMMON/REAL/ANG,CW(4,3),PITCH,BANK,HEADING,ALPHAV,
     1  SUMANG,SQRA,AA(4,2),ROT(3,3),PROJ(4),CANG(4),
     2  CMAG(4),COR(4,3),ANGFACTOR,XH,YH,ZH
        COMMON/FLAGS/ NADIR,LASTPI,IFIRST,IFINISHED,ICAS,KSL,
     1  IRESOL,IDATBAS,ISKY,ISTART,IEND

        COMMON/CONSTS/ICRASH,JCONSTX,JCONSTY,ICONST,KCOUNT,
     1  IPOWER,IXSO,IYSO,HORIZONLIM
        COMMON/BUF2/ ICNT,IDAT,IOUT2

        EQUIVALENCE (IE(1),IXE),(IE(2),IYE),(IE(3),IZE),
     *              (IP(1),IXP),(IP(2),IYP),(IP(3),IZP),
     *              (IRBW,IR),(ICL,IRESOL),(IF(1),IXH),
     *              (IF(2),IYH),(INT2,ICNT),(JDAT1,IDAT),
     *              (JDAT2,IDAT(1,1,2)),(JDAT3,IDAT(1,1,3)),
     *              (IXBS,ISCRN(1)),(IYBS,ISCRN(2))
```

APPENDIX ED

PGDATA.FOR

```
C                    ****************************************
C                    *                                      *
C                    *          SUBROUTINE ZRATA            *
C                    *                                      *
C                    *             DATA BASE  1             *
C                    *                                      *
C                    ****************************************
C
C            LATEST REVISION DATE:     SEPTEMBER 30,1980
C            PROGRAMMED BY:  GERALD L. BECKER
C                    PHONE:  ORLANDO:  677-7621
C                            FORT MYERS: 813/995-7930
C<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C  THIS IS TO BE USED WITH THE DICOMED!!!!!!!!
C<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C            THIS SUBROUTINE IS CALLED BY SUBROUTINE 'SCAN'.
C            THIS SUBROUTINE DEALS WITH DATA BASE 1.  DATA BASE 1
C      IS MADE UP OF BLOCKS WITH 3 POSSIBLE ALTITUDES.  POSSIBLE
C      ALTITUDES ARE HIGH (512), LOW (-128), AND GROUND (0).
C           ALL CALCULATIONS IN THIS SUBROUTINE ARE MADE WITH
C      INTEGER MATH.
C        1   2   3   4
C      ****************M2      THE ENTIRE DATA BASE 1 IS COVERED
C      *   *   *   *   *  WITH DATA BLOCKS AS SHOWN TO THE LEFT.
C      * 1 * 8 * 8 * 2 *1 BLOCKS OF TYPE 1 DATA HAVE A HEIGHT OF
C J=  *  1*  2*  3*  4*  512.  BLOCKS OF TYPE 2 DATA HAVE A
C      ****************  HEIGHT OF -128.  THE REFLECTANCE IS A
C      *   *   *   *   *  FUNCTION OF POSITION WITHIN THE FRAME.
C      * 8 * 2 * 1 * 8 *2 BLOCKS OF TYPE 4 DATA HAVE ZERO HEIGHT
C J=  *  5*  6*  7*  8*  AND A REFLECIANCE OF 4.
C      ****************  DEFINITIONS:
C      *   *   *  X*   *  CLASS: DIFFERENT LEVELS OF DETAIL OR
C      * 1 * 8 * 8 * 2 *3 RESOLUTION.
C      *   *   *   *   *  FRAME COORDINATES: M1 AND M2 VALUES
C      ****************  SPECIFYING THE BLOCK BEING USED.  EX-
C      *   *   *   *   *  AMPLE: POINT X HAS FRAME COORDINATES
C      * 8 * 1 * 2 * 8 *4 M1=3, M2=3.  FRAME COORDINATES ARE
C      *   *   *   *   *  CLASS DEPENDENT. M1=MB1.
```

```
C     ******************  ARRAY INDICES: POSITION WITHIN A GIVEN
C                         FRAME.
C
        SUBROUTINE ZRDATA(*)
        INCLUDE 'PGCOMDAT.FOR'
        INTEGER*2 IDENSITY(3),ISAL,IDD
        INTEGER*2 INDEXX,INDEXY,IPI(10),IOFF1(3),IOFF2(3),
     *  ISAM,INX,INY,KL1,KL2,KL3,KH1,KH2,KH3,IBL
        DIMENSION SCALE(3),IQX(3),IQY(3),IYDEY(3),IXDEX(3),
     *            DNO1(3)
        INTEGER*2 ISAV1(3),ISAV2(3),ISAV3(3),ISAV4(3)
        COMMON /GRND/ INX,INY
        DATA ISAM /1/,IDENSITY/4,8,32/,ISAT /128/
        DATA SCSAM /.9/,N1024/1024/
C BUILDINGS WITH ROADS AND 2 NOISE TYPES AND MULTIPLE RETURN
C
C IP := NADIR CENTRIC COORDINATES
C IPP := WORLD CENTRIC COORDINATES
C IPP = IP + IE
C IF IPP<0 ;IPNP=IPP
C  ELSE     IPNP=IPP+1
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
        NPHIL=N1024*NCR
              MIX=MOD(IPNP(1),(NPHIL))
              MJY=MOD(IPNP(2),(NPHIL))
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C     ARRAY INDICES FOR ALL CLASSES ARE CALCULATED.
C
C             THE RANGE OF INDEXX,INDEXY IS ALWAYS 1<=1024.
C       THE GRID BOUNDARIES OCCUR ON IDD LINES.  THE IDEA OF
C       INDEXX,INDEXY COORDINATES IS TO MATCH DATA COMPRESSION
C       INFORMATION AS IT WOULD BE ACCESSED IN A HIERARCHIAL
C       DATABASE.  TO DEFINE BOUNDARIES ONE NEEDS, IDD, MB1,MB2,
C       THE DISPLACEMENT FROM IDD IN THE COORDINATE FRAME.  ALL
C       DISPLACEMENTS ARE POSITIVE MODULAR ARITHMETIC.
              IF (IPP(1).GE.0) THEN
                   INDEXX=(MIX/NCR)+1
              ELSE
                   INDEXX=(MIX/NCR)+1024
              ENDIF
              IF (IPP(2).GE.0) THEN
                   INDEXY=(MJY/NCR)+1
              ELSE
                   INDEXY=(MJY/NCR)+1024
              ENDIF
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
```

```
C     BLOCK ASSIGNMENTS ARE DEPENDANT ON WHETHER OR NOT THE TEST
C     POINT IS WITHIN THE VALNCR RANGE.
C
                IDD=1024/NCR
        IF(ICL.GT.9) THEN
                TYPE*,'ERROR ICL .GT. 9 ; ZRDATA 9400'
                STOP
        ENDIF
        IBL=IDD-1
        INX=IMOD(INDEXX-1,IDD)
        INY=IMOD(INDEXY-1,IDD)
C THIS ROUTINE ASSUMES NCR=2**(ICL-1)
        KL1=IMOD(64/NCR,IDD)
        KL2=IMOD(128/NCR,IDD)
        KL3=IMOD(160/NCR,IDD)
        KH1=IMOD((1024-64)/NCR,IDD)-1
        KH2=IMOD((1024-128)/NCR,IDD)-1
        KH3=IMOD((1024-160)/NCR,IDD)-1
C       KL1 THROUGH KL3 SET THE LOWER BOUNDS
C       KH1 THROUGH KH3 SET THE UPPER BOUNDS
                MB1=IPNP(1)/1024
                MB2=IPNP(2)/1024
                IREM=MOD(MB1,4)
                JREM=MOD(MB2,4)
                IF (IPP(1).LT.0) IREM=IREM+3
                IF (IPP(2).LT.0) JREM=JREM+3
                J=JREM+(4*IREM)+1
D       IF(IPP(2).GT.1000) TYPE*,'IP1,IP2,J=',IPP(1),IPP(2),J
D       IF(IPP(2).GT.1024) TYPE*,'INDEXY,MOD=',INDEXY,IPT(5)
        GOTO (1010,1020,1020,1040,8,1060,1070,8,1090,1100,1100,
     *       1120,1130,1140,1150,1160),J
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C     IP(3) AND IR ASSIGNMENTS ARE MADE.  THIS ASSIGNMENT IS
C     DEPENDANT ON THE FRAME INDICES.
C
1010    CONTINUE
        CALL GND(KL2,KL2,IBL,IBL,*9,*8,*8)
        IF(INX.LT.KL3) GOTO 8
        IP(3)=512
        CALL CRSIFA(KL3,KL2,IBL,IBL,*6,*4,*5)
        GOTO 7
1020    CONTINUE
        IF(INX.LT.KL2) GOTO 9
        GOTO 8
1040    CONTINUE
        CALL GND(KL2,0,IBL,KH1,*9,*8,*8)
        IF(INX.LT.KL3) GOTO 8
        IP(3)=-128
        CALL  CRSIFA(KL3,0,IBL,KH1,*6,*4,*5)
```

```
      GOTO 3
1060  CONTINUE
      CALL GND(KL1,0,IBL,KH1,*8,*8,*8)
      IF(INX.LT.KL1) GOTO 8
      IF (INY.GT.KH1) GOTO 8
      IP(3)=-128
      CALL CRSIFA(KL1,0,IBL,KH1,*6,*4,*5)
      GOTO 3
1070  CONTINUE
      CALL GND(0,0,IBL,IBL,*8,*8,*8)
      IP(3)=512
      CALL CRSIFA(0,0,IBL,IBL,*6,*4,*5)
      GOTO 7
1090  CONTINUE
      CALL GND(KL1,KL1,KH2,KH1,*8,*9,*8)
      IF (INX.GT.KH3) GOTO 8
      IP(3)=512
      CALL  CRSIFA(KL1,KL1,KH3,KH1,*6,*4,*5)
      GOTO 7
1100  CONTINUE
      IF(INX.GT.KH2) GOTO 9
      GOTO 8
1120  CONTINUE
      IF(INX.GT.KH2) GOTO 9
      IF(INX.GT.KH3) GOTO 8
      IP(3)=-128
      CALL CRSIFA(0,0,KH3,IBL,*6,*4,*5)
      GOTO 3
1130  CONTINUE
      CALL GND (KL2,0,KH2,IBL,*9,*9,*8)
      GOTO 8
 1140 CONTINUE
D     TYPE*,'KL2,KH2,KH1=',KL2,KH2,KH1
          CALL GND(KL2,0,KH2,KH1,*9,*9,*8)
      IF(INX.GT.KH3) GOTO 8
      IF(INX.LT.KL3) GOTO 8
      IP(3)=512
D     TYPE*,'1140: IPT=',IPT
      CALL CRSIFA(KL3,0,KH3,KH1,*6,*4,*5)

      GOTO 7
1150  CONTINUE
      CALL GND(KL2,KL1,KH2,KH1,*9,*9,*8)
      IF(INX.LT.KL3) GOTO 8
      IF(INX.GT.KH3) GOTO 8
      IP(3)=-128
      CALL CRSIFA(KL3,0,KH3,KH1,*6,*4,*5)
      GOTO 3
1160  CONTINUE
      IF(INX.LT.KL2) GOTO 9
      IF(INX.GT.KH2) GOTO 9
```

```
          GOTO 8
C         BOTTOM OF HOLE
 3        IR=0
          GOTO 99
  4       CONTINUE
C         SIDES;X CONSTANT,Y VARIES
          IF(INX.LT.IBL/2) THEN
          IR=158
          ELSE
          IR=98
          ENDIF
          IF(IP(3).EQ.-128)IP(3)=0
                GO TO 99
  5       CONTINUE
C         FRONT AND BACK FACES;Y CONSTANT,X VARIES
D         TYPE*,' "5" '
          IF(INY.LT.IBL/2) THEN
          IR=76
          ELSE
          IR=180
          ENDIF
          IF(IP(3).EQ.-128) THEN
          IP(3)=0
          IR=(IR-127)*.75+127
          ENDIF
        GO TO 99
C         CORNER
  6       CONTINUE
D         TYPE*,' "6"'
          IR=127
          GO TO 99
 7      CONTINUE
C
C     ROOF TEXTURE
C
          IFIR=4
          RETURN 1
C GROUND TEXTURE
  8       CONTINUE
D         IF(IPP(2).GT.999) TYPE*,'"8",KL,IPT=',KL1,KL2,KL3,IPT
D     *           ,'KH=',KH1,KH2,KH3
        IP(3)=0
          IFIR=3
          RETURN 1
C ROAD
  9       CONTINUE
D         TYPE*,' "9" ,INDEXX,INDEXY=',INDEXX,INDEXY
          IR=70
          IP(3)=0
          IF(INX.LT.(8/NCR)) IR=200
          IF(INX.GT.(IDD-8/NCR)) IR=200
```

```
      IF(INX.LT.(4/NCR)) IR=150
      IF(INX.GT.(IDD-4/NCR)) IR=150
      IF(NCR.EQ.8) THEN
              IF(INX.EQ.0) IR=167
              IF(INX.EQ.IDD) IR=167
      ENDIF
D     TYPE*,'IR9=',IR
99    IFIR=1
      RETURN 1
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
      ENTRY  ZRROF(*)
      P11=90./72.
      IR=(P11*(SIN(.092*IPP(1)+.04*IPP(2)-1.)+3.*SIN(
     *  .26*IPP(1)-.103*IPP(2)+.5)+5.*SIN(.432*IPP(1)+
     *  .197*IPP(2)+.8))*(SIN(.096*IPP(2)-.037*IPP(1)-.96)+
     *  3.*SIN(.253*IPP(2)+.096*IPP(1)+.45)+4.*SIN(.385*
     *  IPP(2)-.186*IPP(1))))
C ALPHAV =-4.5/IHORIZON(ICASE) AND IS SET UP IN VIS
      IR=165+IR*EXP(ALPHAV*IP(ICASE))
      IF(IP(3) .EQ. -128) IR=0
      IF(IR.GT.255) TYPE*,'950/2 ZRDATA IR,IP=',IR,IP
      RETURN 1
C
C     GROUND COMPUTATION
C
      ENTRY ZRGND(*)
C      NOISE ON GROUND
      INDX= ABS(MOD((IPP(2)-IPP(1)/32),2044))
      INDY= ABS(MOD((IPP(1)-IPP(2)/128),2044))
      IF(INDX .LT. 1024) THEN
              INDX= 1 + INDX/2
      ELSE
              INDX= 1023 - INDX/2
      ENDIF
      IF(INDY .LT. 1024) THEN
              INDY= 1 + INDY/2
      ELSE
              INDY= 1023 - INDY/2
      ENDIF
C     IR=127+(SCSAV*(IDAT(INDX,INDY,1)+128)-127)*EXP(-.00005*IP(2))
      IR=127
      RETURN 1
C                 ****************************************
C                 *                                      *
C                 *        SUBROUTINE ZRDATA2            *
C                 *                                      *
C                 *           DATA BASE  2               *
C                 *                                      *
C                 ****************************************
```

```
C
C            LATEST REVISION DATE:      30 SEP 1980
C            PROGRAMMED BY:  GERALD L. BECKER
C                    PHONE:  ORLANDO:  677-7621
C                            FORT MYERS:  813/995-7930
C
C<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
C   THIS PROGRAM IS TO BE USED WITH THE DICOMED!!!!!!!!!!!!!!!!!
C<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
C            THIS SUBROUTINE IS CALLED BY SUBROUTINE 'SCAN'.
C           THIS SUBROUTINE DEALS WITH DATA BASE   2.
C      DATA BASE   2 IS A SIN FUNCTION IN BOTH THE X AND Y
C      DIRECTIONS.  THIS GIVES THE EFFECT OF ROLLING HILLS IN A
C      3-D SCENE.
C
        ENTRY ZRDAT2(*)
C++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C      THESE EQUATIONS ARE USED TO DEFINE ALTITUDE (IP(3)) AND
C      REFLECTANCE
        IP(3)= 512*SIN(6.283185*(IPP(1)+IPP(2)))/2048
        IR=IP(3)/4 + 127
        IFIR=1
        RETURN 1

C SUBROUTINE ZRDAT3 SINX*SINY WITH SUN SHADING
C PHILIP GATT
C 9/30/80

C      FOR A FULL DESCRIPTION OF THE MATH SEE THE MEMO FROM
C      DR. B. W. PATZ DATED 9/21/80 ON SUN SHADING
C      NOTE A,C DEFINE THE SUN ANGLE

        ENTRY ZRDAT3(*)
        W1=6.283185/2048
        W2=W1
        A=.5
        C=.8660254
        R=255

        HEIGHT=512*SIN(W1*IPP(1))*SIN(W2*IPP(2))
        IP(3)=HEIGHT+SIGN(.5,HEIGHT)

        IR=(-A*512*W1*COS(W1*IPP(1))*SIN(W2*IPP(2)) + C)*
     *  R/SQRT(1+262144*(W1**2*COS(W1*IPP(1))**2*SIN(W2*
     *  IPP(2))**2 + W2**2*SIN(W1*IPP(1))**2*COS(W2*IPP(2))**2))
        IF(IR.LT.0) IR=0
        IF(IR.GT.255) THEN
                TYPE*,'IR .GT.255 =',IR
                IR=255
```

```
      ENDIF
      IFIR=1
      RETURN 1
C ZRDAT4.FOR
C CREATED BY PHILIP GATT AND DR. B. W. PATZ
C 10/20/80
C SIN(X)*SIN(G(Y)) + NOISE
C G(Y) IS SMALL FOR Y < 3000
      DATA A/.852R7/,B/-.173648/C/.492404/
      ENTRY ZRDAT4(*)
C IDENSITY(1) IS THE   WORLD POINTS WHICH MAP TO ONE NOISE POINT
C G(Y) IS SMALL FOR Y < 3000

C COMPUTE THE HEIGHT
      IPP2=IPP(2)+4500
      IDENSITY(2)=16
      XZ=0
      DO 1221 ILOP=1,3
        IIX=ABS(MOD((IPP(1)-IOFF1(ILOP)),1022*IDENSITY(ILOP)))
        IIY=ABS(MOD((IPP(2)-IOFF2(ILOP)),1022*IDENSITY(ILOP)))
      IF(IIX .LT. (512*IDENSITY(ILOP))) THEN          ! 1
            IQX(ILOP)= 1 + IIX/IDENSITY(ILOP)
            IXDEX(ILOP)=IQX(ILOP)+1
            IF(IQX(ILOP).EQ.512) IXDEX(ILOP)=511
      ELSE                                            ! 1
            IQX(ILOP)= 1023 - IIX/IDENSITY(ILOP)
            IXDEX(ILOP)=IQX(ILOP)-1
      ENDIF                                           ! 1
            IODX=MOD(IIX,IDENSITY(ILOP))
            IODY=MOD(IIY,IDENSITY(ILOP))
      IF( IIY .LT. (512*IDENSITY(ILOP))) THEN         ! 1
            IQY(ILOP)= 1 + IIY/IDENSITY(ILOP)
            IYDEY(ILOP)=IQY(ILOP)+1
            IF(IQY(ILOP).EQ.512) IYDEY(ILOP)=511
      ELSE                                  ! 1
            IQY(ILOP)= 1023 - IIY/IDENSITY(ILOP)
            IYDEY(ILOP)=IQY(ILOP)-1
      ENDIF                                 ! 1
      ISAV1(ILOP)=IDAT(IQX(ILOP),IQY(ILOP),ILOP)
      ISAV2(ILOP)=IDAT(IXDEX(ILOP),IQY(ILOP),ILOP)-ISAV1(ILOP)
      ISAV3(ILOP)=IDAT(IQX(ILOP),IYDEY(ILOP),ILOP)-ISAV1(ILOP)
      ISAV4(ILOP)=IDAT(IXDEX(ILOP),IYDEY(ILOP),ILOP)-
     *            ISAV3(ILOP)-ISAV2(ILOP)-ISAV1(ILOP)
      DNO1(ILOP)=1./IDENSITY(ILOP)
      DELZ=ISAV1(ILOP)+((IODX*ISAV2(ILOP)+IODY*ISAV3(ILOP))
     *  +IODX*IODY*ISAV4(ILOP)*DNO1(ILOP))*DNO1(ILOP)
      IF(ILOP.EQ.2) THEN
            IF(IPP2.GT.2*IPP(1)+250) THEN
                  FACT=6
            ELSE IF(IPP2.GT.2*IPP(1)+150) THEN
                  FACT=1.+(IPP2-150-2*IPP(1))/20
```

```
                ELSE
                        FACT=1
                ENDIF
        ELSE
                FACT=SCALE(ILOP)
        ENDIF
        IF(ILOP.EQ.2) THEN
        XZ=XZ+(DELZ+128)*FACT
        ELSE
        XZ=XZ+DELZ*FACT
        ENDIF
1221    CONTINUE
        IZ=XZ
C COMPUTE THE REFLECTANCE
        PARTX=0
        PARTY=0
        DO 3311 ILOP=1,3
        XITEM=ISAV4(ILOP)*DNO1(ILOP)
        PARTIDATX=(ISAV2(ILOP)+IDDY*XITEM)*DNO1(ILOP)
        PARTIDATY=(ISAV3(ILOP)+IDDX*XITEM)*DNO1(ILOP)
        IF(ILOP .EQ.1) THEN
                XMV=EXP(-IP(2)*.00006)
                PARTIDATX=PARTIDATX*XMV
                PARTIDATY=PARTIDATY*XMV
        ENDIF
        IF(ILOP.EQ.2) THEN
                CVER=FACT
        ELSE
                CVER=SCALE(ILOP)
        ENDIF
        CVER1 =IPP(1)-IOFF1(ILOP)
        CVER2= IPP(2)-IOFF2(ILOP)
        PARTX=PARTX+PARTIDATX*SIGN(CVER,CVER1)
        PARTY=PARTY+PARTIDATY*SIGN(CVER,CVER2)
3311    CONTINUE
        REMAG=SQRT(1 + PARTX**2 + PARTY**2)
        IR=255*(-A*PARTX-B*PARTY+C)/REMAG
        IF(IR .LT.0) IR=0
        IF(IR .GT. 255) THEN
                TYPE*,'ERROR IR .GT. 255 =',IR
                IR=255
        ENDIF
        IP(3)=IZ
        IFIR=1
        RETURN 1
C ZRDAT5.FOR
C CREATED BY PHILIP GATT AND DR. B. N. PATZ
C 10/20/80
C SIN(X)*SIN(G(Y)) + NOISE AND LAKES
C G(Y) IS SMALL FOR Y < 3000
```

```
      ENTRY ZRDAT5(*)
      DATA IOFF1,IOFF2 /127,69,0,13,243,0/
      DATA SCALE /.4,1.6,3./
C     IDENSITY(ILOP) IS THE WORLD POINTS WHICH MAP TO ONE
C     NOISE POINT
C
C     SUN VECTOR FROM GROUND POINT TO SUN (X,A;Y,B)
C
C FOR STARTING XE LOCATION USE 2000
C XE SHOULD BE TESTED AT -10350
C     A=.866
C     B=-.25
C     C=.4330125
C     SEE ZRDATA4 FOR (ABC) DATA STATEMENT


C COMPUTE THE HEIGHT
      IPP2=IPP(2)-16500
      W1=1./3000.
      W=2*3.14159*W1*.5
      XNUM=1+(W1*IPP2)**2
      DENOM=1./(1.+ABS(W1*IPP2))
      H=(XNUM*DENOM -.85)*2*3.14159*.6
      IHEIGHT=1000*(SIN(W*IPP(1))*SIN(H))
      XZ=0
      IPIL=1            !  TREES AND SHRUBS
      IF(IHEIGHT.LT.12) IPIL=2        !  ROCKS AND TREES
      IF(IHEIGHT.LT.-12) IPIL=3       !  LAKE BANK (TERRAIN)
      DO 1111 ILOP=IPIL,3
        IIX=ABS(MOD((IPP(1)-IOFF1(ILOP)),1022*IDENSITY(ILOP)))
        IIY=ABS(MOD((IPP(2)-IOFF2(ILOP)),1022*IDENSITY(ILOP)))
      IF(IIX .LT. (512*IDENSITY(ILOP))) THEN          ! 1
              IQX(ILOP)= 1 + IIX/IDENSITY(ILOP)
              IXDEX(ILOP)=IQX(ILOP)+1
              IF(IQX(ILOP).EQ.512) IXDEX(ILOP)=511
      ELSE                                            ! 1
              IQX(ILOP)= 1023 - IIX/IDENSITY(ILOP)
              IXDEX(ILOP)=IQX(ILOP)-1
      ENDIF                                           ! 1
              IODX=MOD(IIX,IDENSITY(ILOP))
              IODY=MOD(IIY,IDENSITY(ILOP))
      IF( IIY .LT. (512*IDENSITY(ILOP))) THEN         ! 1
              IQY(ILOP)= 1 + IIY/IDENSITY(ILOP)
              IYDEY(ILOP)=IQY(ILOP)+1
              IF(IQY(ILOP).EQ.512) IYDEY(ILOP)=511
      ELSE                                            ! 1
              IQY(ILOP)= 1023 - IIY/IDENSITY(ILOP)
              IYDEY(ILOP)=IQY(ILOP)-1
      ENDIF                                           ! 1
      ISAV1(ILOP)=IDAT(IQX(ILOP),IQY(ILOP),ILOP)
      ISAV2(ILOP)=IDAT(IXDEX(ILOP),IQY(ILOP),ILOP)-ISAV1(ILOP)
```

```
      ISAV3(ILOP)=IDAT(IQX(ILOP),IYDEY(ILOP),ILOP)-ISAV1(ILOP)
      ISAV4(ILOP)=IDAT(IXDEX(ILOP),IYDEY(ILOP),ILOP)
     *            -ISAV3(ILOP)-ISAV2(ILOP)-ISAV1(ILOP)
      DNO1(ILOP)=1./IDENSITY(ILOP)
      DELZ=ISAV1(ILOP)+((IDDX*ISAV2(ILOP)+IDDY*ISAV3(ILOP))
     *  +IDDX*IDDY*ISAV4(ILOP)*DNO1(ILOP))*DNO1(ILOP)
      XZ=XZ+DELZ*SCALE(ILOP)
1111  CONTINUE
      IZ=XZ+IHEIGHT
      IFIR=2
      IF(IZ.LT.-50) THEN
              IFIR=5
              IF(IDAT(IQX(2),IQY(2),2).GT.0) THEN
                      IPH=IDAT(IQX(3),IQY(3),3)/31
              IZ=(2.5*SIN((3*IPP(1)+4*IPP(2))*2*3.14159/120+
     *           IPH)+1.25*SIN((3*IPP(1)+4*IPP(2))*2*3.14159/
     *           60+IPH))*IDAT(IQX(2),IQY(2),2)/127-50
              ELSE
1212                  IZ=-50
              ENDIF
      ENDIF
      IP(3)=IZ+70
      RETURN 1
      ENTRY ZRDAT5IR(*)
C COMPUTE THE REFLECTANCE
      PARTX=0
      PARTY=0
      DO 2211 ILOP=IPIL,3
      XTEM=ISAV4(ILOP)*DNO1(ILOP)
      PARTIDATX=(ISAV2(ILOP)+IDDY*XTEM)*DNO1(ILOP)
      PARTIDATY=(ISAV3(ILOP)+IDDX*XTEM)*DNO1(ILOP)
      IF(ILOP .EQ.1) THEN
              XMV=EXP(-IP(2)*.00005)
              PARTIDATX=PARTIDATX*XMV
              PARTIDATY=PARTIDATY*XMV
      ENDIF
       CVER=SCALE(ILOP)
      CVER1 =IPP(1)-IOFF1(ILOP)
      CVER2= IPP(2)-IOFF2(ILOP)
      PARTX=PARTX+PARTIDATX*SIGN(CVER,CVER1)
      PARTY=PARTY+PARTIDATY*SIGN(CVER,CVER2)
2211  CONTINUE
      PARTIDATX=PARTX
      PARTIDATY=PARTY
      PARTH1=2*.6*2*3.14159*W1*W1*IPP2*DENOM
      PARTH2=XNUM*DENOM**2*W1*.6*2*3.14159
      IF(IPP2 .GT. 0) THEN
              PARTHY=PARTH1-PARTH2
      ELSE IF (IPP2 .LT. 0) THEN
              PARTHY=PARTH1 + PARTH2
      ELSE
```

```
                PARTHY=0
        ENDIF
2227    PARTRIGX=1000*W*COS(W*IPP(1))*SIN(H)
        PARTRIGY=1000*SIN(W*IPP(1))*COS(H)*PARTHY
        PARTZX=PARTRIGX + PARTIDATX
        PARTZY=PARTRIGY + PARTIDATY
        REMAG=SQRT(1 + PARTZX**2 + PARTZY**2)
        IR=253*(-A*PARTZX-B*PARTZY+C)/REMAG
D       IF(IP(2).GT.500)THEN
C       TYPE*,'TRIGX,Y,DATX,Y=',PARTRIGX,PARTRIGY,PARTIDATX,
C       PARTIDATY
D       TYPE *,'IP(3)=',IP
C       TYPE*,'PARTH1,PARTH2=',PARTH1,PARTH2
C       TYPE*,'PARTZX,PARTZY=',PARTZX,PARTZY
D       TYPE*,'IR=',IR
C       TYPE*,' '
D       ENDIF
        IF(IR .LT.0) IR=0
        IF(IR .GT. 255) THEN
                TYPE*,'ERROR IR .GT. 255 =',IR
                IR=255
        ENDIF
        RETURN 1
        ENTRY ZRDAT551R(*)
C       COMPUTE SMALL WAVES ON LAKE
C
        IF(IDAT(IQX(2),IQY(2),2).GT.-32) THEN
C       CODE NEGLECTS PARTIALS OF IDAT COMPARED TO TRIG TERMS
                PARTZX=3*3.14159/24*(COS(IPH+(3*IPP(1)+4*IPP(2))
     *                  *2*3.14159/120)+COS(IPH+(3*IPP(1)+4*
     *                  IPP(2))*2*3.14159/60))*EXP(-IP(2)*.0001)
                PARTZY=1.33333*PARTZX
        ELSE
                PARTZX=0
                PARTZY=0
        ENDIF
        REMAG=SQRT(1 + PARTZX**2 + PARTZY**2)
        IR=220*(-A*PARTZX -B*PARTZY + C)/REMAG
        IF(IR .LT.0) IR=0
        IF(IR .GT. 255) THEN
                TYPE*,'ERROR IR .GT. 255 =',IR
                IR=255
        ENDIF
        RETURN 1
C ZRDAT4.FOR
C CREATED BY PHILIP GATT AND DR. B. W. PATZ
C 10/20/80
C SIN(X)*SIN(G(Y)) + NOISE
C G(Y) IS SMALL FOR Y < 3000

        ENTRY ZRDAT6(*)
```

```
C IDENSITY(1) IS THE   WORLD POINTS WHICH MAP TO ONE NOISE POINT
C       A=.866
C       B=-.25
C       C=.4330125
        A=.866
        B=0
        C=.5
C COMPUTE THE HEIGHT
        W1=1./3000.
        W=2*3.14159*W1*.7
        IPP2=IPP(2)-4400
        XNUM=1+(W1*IPP2)**2
        DENOM=1./(1.+ABS(W1*IPP2))
        H=(XNUM*DENOM -1)*2*3.14159*.6
        IIX=ABS(MOD(IPP(1),(1022*IDENSITY(1))))
        IIY=ABS(MOD(IPP(2),(1022*IDENSITY(1))))
        IF(IIX .LT. (512*IDENSITY(1))) THEN                    ! 1
                IQX(1)= 1 + IIX/IDENSITY(1)
                IXDEX(1)=IQX(1)+1
                IF(IQX(1).EQ.512) IXDEX(1)=511
        ELSE                                                   ! 1
                IQX(1)= 1023 - IIX/IDENSITY(1)
                IXDEX(1)=IQX(1)-1
        ENDIF                                                  ! 1
                IDDX=MOD(IIX,IDENSITY(1))
                IDDY=MOD(IIY,IDENSITY(1))
        IF( IIY .LT. (512*IDENSITY(1))) THEN                   ! 1
                IQY(1)= 1 + IIY/IDENSITY(1)
                IYDEY(1)=IQY(1)+1
                IF(IQY(1).EQ.512) IYDEY(1)=511
        ELSE                                                   ! 1
                IQY(1)= 1023 - IIY/IDENSITY(1)
                IYDEY(1)=IQY(1)-1
        ENDIF                                                  ! 1
        ISAV1(1)=IDAT(IQX(1),IQY(1),1)*1.0
        ISAV2(1)=IDAT(IXDEX(1),IQY(1),1)-ISAV1(1)
        ISAV3(1)=IDAT(IQX(1),IYDEY(1),1)-ISAV1(1)
        ISAV4(1)=IDAT(IXDEX(1),IYDEY(1),1)-ISAV3(1)
     *  -ISAV2(1)-ISAV1(1)
        DNO1(1)=1./IDENSITY(1)
        DELZ=ISAV1(1)+((IDDX*ISAV2(1)+IDDY*ISAV3(1))
     *  +IDDX*IDDY*ISAV4(1)*DNO1(1))*DNO1(1)
        IP(3)=512*(SIN(W*IPP(1))*SIN(H)) + DELZ
C       RETURN 1
C       ENTRY ZRDAT4IR
C COMPUTE THE REFLECTANCE
        XTEM=ISAV4(1)*DNO1(1)
        PARTIDATX=(ISAV2(1)+IDDY*XTEM)*DNO1(1)*ISIGN(1,IPP(1))
        PARTIDATY=(ISAV3(1)+IDDX*XTEM)*DNO1(1)*ISIGN(1,IPP(2))
        PARTH1=2*.6*2*3.14159*W1*W1*IPP2*DENOM
        PARTH2=XNUM*DENOM**2*W1*.6*2*3.14159
```

```
      IF(IPP(2) .GT. 0) THEN
              PARTHY=PARTH1-PARTH2
      ELSE IF (IPP(2) .LT. 0) THEN
              PARTHY=PARTH1 + PARTH2
      ELSE
              PARTHY=0
      ENDIF
      PARTRIGX=512*W*COS(W*IPP(1))*SIN(H)
      PARTRIGY=512*SIN(W*IPP(1))*COS(H)*PARTHY
      PARTZX=PARTRIGX + PARTIDAIX
      PARTZY=PARTRIGY + PARTIDAIY
      REMAG=SQRT(1 + PARTZX**2 + PARTZY**2)


      IR=255*(-A*PARTZX -B*PARTZY + C)/REMAG
      IF(IR .LT.0) IR=0
      IF(IR .GT. 255) THEN
              TYPE*,'ERROR IR .GT. 255 =',IR
              IR=255
      ENDIF
      IFIR=1
      RETURN 1
      END
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C
      SUBROUTINE CRSIFA(LX,LY,HX,HY,*,*,*)
C
      INTEGER*2 INX,INY,LX,LY,HX,HY
C
      COMMON /GRND/ INX,INY
C     CORNER TEST
C
      IF(((INX .EQ.LX) .OR. (INX .EQ. HX)) .AND.
     *    ((INY.EQ.LY) .OR. (INY .EQ. HY)))  RETURN 1
C
C-------------------------------------------------------
C
C     SIDE TEST
C
      IF((INX .EQ.LX  ) .OR. (INX .EQ. HX))  RETURN 2
C
C-------------------------------------------------------
C
C     FACE TEST
C
      IF((INY .EQ.LY) .OR. (INY .EQ. HY))  RETURN 3
C
C-------------------------------------------------------
C
      RETURN
```

```
      END
C
C
C=====================================================================
C
C
      SUBROUTINE GND(LX,LY,HX,HY,*,*,*)
C
C
      INTEGER*2 INX,INY,LX,LY,HX,HY
C
      COMMON /GRND/ INX,INY
C
C
      IF(INX .LT. LX) RETURN 1
      IF(INX .GT. HX) RETURN 2
      IF(INY .LT. LY) RETURN 3
      IF(INY .GT. HY) RETURN 3
C
      RETURN
      END
```

APPENDIX EE

PGFILTER.FOR

```
C       PGFILTER.FOR
C
        SUBROUTINE FINFIL
        INCLUDE'PGCOMDAT.FOR'
        INCLUDE'PGBUF.FOR'

C
100     TYPE*,'WE ARE IN THE FILTER STAGE'
        IFILTER1=1
        DO 200 J=1,IFILDIM
        DO 200 I=1,IFILDIM
        KDIV=ICNT(I,J)
        IF (KDIV.EQ.0) KDIV=1
        IF(KDIV.GT.37) TYPE*,'FINFIL ICNT=',KDIV,'AT',I,J
        ITEMBUF(I,J)=ITEMBUF(I,J)/KDIV
200     CONTINUE
        TYPE*,'BUFFER IS FILTERED BY FILTER  1'
        RETURN
C
C
C
        ENTRY FILT2
        TYPE*,' READ OF FILE  22 REQUIRED? (YES=1)'
        ACCEPT*,IREAD2
        IF(IREAD2.EQ.1) THEN
                TYPE*,'READING FILE  21 TO BUFFER'
                READ(21,*) ITEMBUF
                TYPE*,'READING FILE  31 TO ICNT'
                READ(31,*) INT2
        ELSE IF(IFILTER2.EQ.1) THEN
                TYPE*,'ERROR--MODE COMPLETED: CHECK STATUS'
                RETURN
        ELSE IF(IFILTER1.EQ.0) THEN
                TYPE*,'NO OPERATIONAL FILE FOUND: CHECK STATUS'
                RETURN
        ENDIF
        IFILTER2=1
        DO 111 IYBS=512,1,-1
```

```
      DO 222 IXBS=1,512
      IF(ICNT(IXBS,IYBS).NE.0) GOTO 222
      JCOUNT=0
      JSUM=0
      DO 300 K=IYBS+1,IYBS-1,-1
      DO 444 L=IXBS-1,IXBS+1
            IF((L.LT.1).OR.(L.GT.512)) GOTO 444
            IF((K.LT.1).OR.(K.GT.512)) GOTO 300
            IF(ICNT(K,L).EQ.0) GO TO 444
                  JCOUNT=JCOUNT+1
                  JSUM=JSUM+IIEMBUF(K,L)
444   CONTINUE
300   CONTINUE
      IF(JCOUNT.EQ.0) JCOUNT=1
      IIEMBUF(IXBS,IYBS)=JSUM/JCOUNT
      ICNT(IXBS,IYBS)=-1
222   CONTINUE
      TYPE*,'FIL 2 LINE',IYBS
111   CONTINUE
445   RETURN
C
C
      ENTRY CLEAR
      DO 500 J=1,512
      DO 500 I=1,512
      ICNT(I,J)=0
      IIEMBUF(I,J)=0
500   CONTINUE
      TYPE*,'BUFFERED CLEARED'
      RETURN
      END
```

APPENDIX EE

PGLOGLOAD.FOR

```
      SUBROUTINE LOGLOAD
      INCLUDE'PGCOMDAT.FOR'
      INCLUDE'PGBUF.FOR'
      KCOUNT=KCOUNT+1
      IF((IXBS.NE.IXSO).OR.(IYBS.NE.IYSO)) THEN
              IPOWER=2
              KCOUNT=1
              IXSO=IXBS
              IYSO=IYBS
              GOTO 1000
      ELSE IF(IPOWER.EQ.KCOUNT) THEN
              IPOWER=2*IPOWER
1000          GOTO(50,10,20,30,40),IFIR ! GET REFLECTANCE IR
10            CALL ZRDAT5IR(*50)
20            CALL ZRGND(*50)
30            CALL ZRROF(*50)
40            CALL ZRDAT55IR(*50)
50            CONTINUE
      IF((IXBS.GT.IFILDIM).OR.(IXBS.LE.0)) THEN
              TYPE*,'IXBS=',IXBS,'LOGLD'
              RETURN
      ELSE IF((IYBS.GT.IFILDIM).OR.(IXBS.LE.0))THEN
              TYPE*,'IYBS=',IYBS,'LOGLD'
              RETURN
      ENDIF
      IF((IR.GT.255).OR.(IR.LT.0))THEN
              TYPE*,'LOGLOAD 2150 IR=',IR
              RETURN
      ENDIF
              ITEMBUF(IYBS,IXBS)=ITEMBUF(IYBS,IXBS)+IR
              ICNT(IYBS,IXBS)=ICNT(IYBS,IXBS)+1
      IF(ICNT(IYBS,IXBS).LT.0) TYPE*,'PGLOGLOAD ICNT',
                               ICNT(IYBS,IXBS)
      IF(ICNT(IYBS,IXBS).GT.37) TYPE*,'PGLOGLOAD ICNT',
                                ICNT(IYBS,IXBS)
      IF(ICNT(IYBS,IXBS).EQ.126) TYPE*,'WARNING ICNT .GT.126'
      ENDIF
      RETURN
      END
```

APPENDIX EG

PGMAIN.FOR

```
C
C       PGMAIN.FOR
C
        INCLUDE 'PGCOMDAT.FOR'
        INCLUDE 'PGBUF.FOR'
        DATA IRUN,IFILTER1,IFILTER2 /0,0,0/
C IFILDIM IS THE FILE DIMENSION AND THE SCREEN DIMENSION + 1
C IF WE WHISH TO HAVE A 2*SCREEN OF DATA PER LINE THEN IFILDIM=
C 1024 AND ANGLEFACT=2 ?? SINCE ANGLEFACT IS A FUNCTION OF NX
      TYPE*,'WHAT DIMENSION DO YOU WANT FOR THE SCENE NORM=512'
        ACCEPT*,IFILDIM
        IFLDM02=IFILDIM/2
5       TYPE*,'   ENTER COMMAND BY NUMBER:  '
        TYPE*,'         (1) INITIALIZE SCENE AND RUN MAIN (NOTE:'
        TYPE*,'             FILTER 1 EXECUTED AUTOMATICALLY)'
C       TYPE*,'         (2) ANALYSIZE SCENE'
        TYPE*,'         (3) RUN DICOMED PICTURE OF BUFFER'
        TYPE*,'         (4) WRITE BUFFER'
        TYPE*,'         (5) WRITE ICNT'
        TYPE*,'         (6) CHECK STATUS'
        TYPE*,'         (7) EXECUTE FILTER  2'
        TYPE*,'         (8) STOP'
        TYPE*,'         (9) READ ITEMBUF AND INT2'
        ACCEPT*, ICOMAND
        GOTO (11,6,33,44,55,66,77,88,99) ICOMAND
11      CALL RUN
        GOTO 5
6       TYPE*,'   ERROR IN COMMAND CODE--TRY AGAIN'
        GOTO 5
C22     CALL ANSCN3D
C       GOTO 5
33      CONTINUE
        TYPE*,'READ REQUIRED ? YES=1'
        ACCEPT*,IYES
        IF(IYES .EQ. 1) CALL READ
        CALL PGCAM
        GOTO 5
44      CONTINUE
```

```
      TYPE*,' WHICH FILE DO YOU WANT TO WRITE ITEMBUF INTO
     *  (20:29)'
        ACCEPT*,IFILE
        TYPE*,' WRITING ITEMBUF INTO FILE ',IFILE
        WRITE(IFILE,1000) ((ITEMBUF(I,J),I=1,IFILDIM),J=1,
        IFILDIM)
        GOTO 5
55      CONTINUE
      TYPE*,' WHICH FILE DO YOU WANT TO WRITE ICNT INTO (30:39)'
        ACCEPT*,IFILE
        TYPE*,' WRITING ICNT INTO FILE ',IFILE
        WRITE(IFILE,1000) ((INT2(I,J),I=1,IFILDIM),J=1,IFLDM02)
        GOTO 5
66      TYPE*,'  '
        TYPE*,'  '
        TYPE*,'    STATUS VARIABLE CHECK'
        TYPE*,'  '
    TYPE*,'IRUN=',IRUN,'IFILTER1=',IFILIER1,'IFILTER2=',IFILTER2
        TYPE*,'  '
        TYPE*,'  '
        GOTO 5
77      CALL FILT2
        GOTO5
88      STOP
99      CALL READ
        GO TO 5
1000    FORMAT(66A2)
        END
        SUBROUTINE READ
        INCLUDE'PGBUF.FOR'
        INCLUDE 'PGCOMDAT.FOR'
    TYPE*,' WHICH FILE DO YOU WANT TO READ ITEMBUF FROM (20:29)'
        ACCEPT*,IFILE
        TYPE*,' READING ITEMBUF FROM FILE ',IFILE
      READ(IFILE,1000) ((ITEMBUF(I,J),I=1,IFILDIM),J=1,IFILDIM)
        DO 100 I=1,500,100
        DO 100 J=1,500,100
        TYPE*,'ITEMBUF(',J,',',I,') =',ITEMBUF(I,J)
100     CONTINUE
        TYPE*,'READING INT2 FROM FILE  ',IFILE
        TYPE*,' INT2 IS EQUIVALENCED TO ICNT'
        READ (IFILE,1000) ((INT2(I,J),I=1,IFILDIM),J=1,IFLDM02)
        DO 101 I=1,500,100
        DO 101 J=1,500,100
        TYPE*,'ICNT(',J,',',I,')=',ICNT(J,I)
 101    CONTINUE
        TYPE*,'DO YOU WANT TO SCALE ITEMBUF YES=1'
        ACCEPT*,IYES
        MIN8=1000
        IF(IYES.NE.1) GOTO 400
        MAX8=-1000
```

```
      TYPE*,'INPUT SCALE MIN,MAX  NORM=0,255'
      ACCEPT*,SCMIN,SCMAX
      DO 200  I=1,IFILDIM
      DO 200  J=1,IFILDIM

              IF(ITEMBUF(I,J).GE.MINB) MINB=ITEMBUF(I,J)
              IF(ITEMBUF(I,J).LE.MAXB) MAXB=ITEMBUF(I,J)
200   CONTINUE
      SCALE=(SCMAX-SCMIN)/(MAXB-MINB)
      DO 300 I=1,IFILDIM
      DO 300 J=1,IFILDIM
              ITEMBUF(I,J)=(ITEMBUF(I,J)-MINB)*SCALE+SCMIN
300   CONTINUE
      RETURN
400   CONTINUE
     TYPE*,'DO YOU WANT TO ELIMINATE ALL VALUES <0,>255 YES=1'
      ACCEPT*,IYES
      IF(IYES .NE. 1) RETURN
      JCOUNT=0
      LCOUNT=0
              DO 500 I=1,IFILDIM
              DO 500 J=1,IFILDIM
              IF(ITEMBUF(I,J).GT.255)THEN
                      JCOUNT=JCOUNT+1
                      ITEMBUF(I,J)=255
              ENDIF
              IF(ITEMBUF(I,J).LT.0)THEN
                      LCOUNT=LCOUNT+1
                      ITEMBUF(I,J)=0
              ENDIF
500   CONTINUE
              TYPE*,'  BUF .GT. 255 =',JCOUNT
              TYPE*,'  BUF .LT. 0 =',LCOUNT
      RETURN
1000  FORMAT(66A2)
      END
      SUBROUTINE INPUT
      INCLUDE 'PGCOMDAT.FOR'
109   TYPE*,'INPUT DATABASE CHOICE 1=ZRDATA,2=ZRDAT2,3=ZRDAT3'
      TYPE*,'ZRDAT4=4,ZRDAT5=5'
      ACCEPT*,IDATBAS
      IF((IDATBAS.LT.1).OR.(IDATBAS.GT.5)) GOTO 109
      TYPE*,'INPUT STARTING SCANLINE  ,ENDING  '
      TYPE*,'INPUT A FACTOR TO DIVIDE THE EYE COORDS BY'
      TYPE*,'INPUT SCALE FACTOR FOR ANGLE BETWEEN SCAN LINES'
      TYPE*,'INPUT ICAS=1,2,3 TO SELECT A ROTATION MATRIX'
      TYPE*,'APROXIMATION'
      TYPE*,'INPUT ISCALE FOR THE ROTATION MATRIX'
      ACCEPT*,ISTART,IEND,IDIV1,ANGFACTOR,ICAS,ISCALE
      TYPE*,'INPUT THE EYE LOCATION XE,YE,ZE'
      TYPE*,'INPUT THE SCREEN CENTER IUS,IVS,IWS'
```

```
      TYPE*,'INPUT THE SCREEN DIMENSIONS (LENGTH) LX,LY'
      ACCEPT*,IXE,IYE,IZE,IUS,IVS,IWS,LX,LY
      TYPE*,'INPUT PITCH,BANK,HEADING IN DEGREES'
      ACCEPT*,PITCH,BANK,HEADING
      IF((IDATBAS .EQ.4).OR.(IDATBAS .EQ.5).OR.(IDATBAS.EQ.1))
     * THEN
      TYPE*,'DO YOU WANT TO READ IN TEXTURE NOISE JDAT1?YES=1'
      ACCEPT*,IYES
      IF(IYES.EQ.1) THEN
              TYPE*,' WHICH FILE DO YOU WANT TO READ NOISE'
              TYPE*,' FROM (40:49)'
              ACCEPT*,IFILE
              TYPE*,' READING JDAT1 FROM FILE ',IFILE
              READ(IFILE,5005) ((JDAT1(I,J),I=1,512),J=1,256)
      ENDIF
      TYPE*,'DO YOU WANT TO READ IN TEXTURE NOISE JDAT2,YES=1'
      ACCEPT*,IYES
      IF(IYES.EQ.1) THEN
              TYPE*,' WHICH FILE DO YOU WANT TO READ NOISE'
              TYPE*,' FROM (40:49)'
              ACCEPT*,IFILE
              TYPE*,' READING JDAT2 FROM FILE ',IFILE
              READ(IFILE,5005) ((JDAT2(I,J),I=1,512),J=1,256)
      ENDIF
      TYPE*,'DO YOU WANT TO READ IN TEXTURE NOISE JDAT3,YES=1'
      ACCEPT*,IYES
      IF(IYES.EQ.1) THEN
              TYPE*,' WHICH FILE DO YOU WANT TO READ NOISE'
              TYPE*,' FROM (40:49)'
              ACCEPT*,IFILE
              TYPE*,' READING JDAT3 FROM FILE ',IFILE
              READ(IFILE,5005) ((JDAT3(I,J),I=1,512),J=1,256)
      ENDIF
      ENDIF
5005  FORMAT(66A2)
      RETURN

      ENTRY OUTPUT
      WRITE(10,*),'ROT(3,3),PROJ(4),NADIR=',ROT,PROJ,NADIR
      WRITE(10,*),'COR(4,3),AA(4,2)=',COR,AA
      WRITE(10,*),'CW(4,3)=',CW
      WRITE(10,*),'ANGLE BETWEENE SCANS=',ANG
      WRITE(10,*),'IXE,IYE,IZE=',IXE,IYE,IZE
      WRITE(10,*),'SCREEN CENTER LOCATION IN EYE COORDS',
     *             IUS,IVS,IWS
      WRITE(10,*),'SCREEN DIMENSIONS LX,LY,NX,NY=',LX,LY,NX,NY
      WRITE(10,*),'PITCH,BANK,HEADING=',PITCH,BANK,HEADING
      WRITE(10,*),'ICAS,IDIV1=',ICAS,IDIV1
      WRITE(10,*),'ISCALE,ANGFACT=',ISCALE,ANGFACTOR
      WRITE(10,*),'ICONST,JCONSTX,JCONSTY=',ICONST,JCONSTX,
     *            JCONSTY
```

```
      WRITE(10,*),'CMAG(4),CANG(4)=',CMAG,CANG
      TYPE*,'ROT(4,3),PROJ(4),NADIR=',ROT,PROJ,NADIR
      TYPE*,'COR(4,3),AA(4,2)=',COR,AA
      TYPE*,'CW(4,3)=',CW
      TYPE*,'ANGLE BETWEENE SCANS=',ANG
      TYPE*,'IXE,IYE,IZE=',IXE,IYE,IZE
      TYPE*,'SCREEN CENTER LOCATION IN EYE COORDS',IUS,IVS,IWS
      TYPE*,'SCREEN DIMENSIONS LX,LY,NX,NY=',LX,LY,NX,NY
      TYPE*,'PITCH,BANK,HEADING=',PITCH,BANK,HEADING
      TYPE*,'ICAS,IDIV1=',ICAS,IDIV1
      TYPE*,'ISCALE,ANGFACT=',ISCALE,ANGFACTOR
      TYPE*,'ICONST,JCONSTX,JCONSTY=',ICONST,JCONSTX,JCONSTY
      TYPE*,'CMAG(4),CANG(4)=',CMAG,CANG
      RETURN

      ENTRY TESTPR
C THIS ROUTINE WILL COMPUTE THE EXACT SCREEN COORDINATES OF A
C WORLD PT.
103   TYPE*,' DO YOU WANT TO TEST THE PROJECTION PROCESSOR ?'
      TYPE*,'NO=0'
      ACCEPT*,IANSPROJ5
      IF(IANSPROJ5 .NE.0) THEN
              TYPE*,'INPUT X,Y,Z IN NADIR CENTRIC COORDS'
              ACCEPT*,IXP,IYP,IZP
              UPOINT=(ROT(1,1)*IXP+ROT(1,2)*IYP-ROT(1,3)*
     *              (IZE-IZP))
              VVV=(ROT(2,1)*IXP+ROT(2,2)*IYP-ROT(2,3)*
     *             (IZE-IZP))
              WPOINT=(ROT(3,1)*IXP+ROT(3,2)*IYP-ROT(3,3)*
     *              (IZE-IZP))
              XS=ICONST*UPOINT/VVV -JCONSTX
              YS=-ICONST*WPOINT/VVV -JCONSTY
              IXBS=(2*(XS+1) +NX)/2
              IYBS=(2*(YS+1) +NY)/2
              TYPE*,'PROJ5 TEST X,Y=',IYBS,IXBS,'IP=',IXP,
     *             IYP,IZP
              GO TO 103
      ENDIF
      RETURN
      END


      SUBROUTINE RUN
C THIS ROUTINE WILL CONTROL THE SEQUENCING OF THE REAL SCAN
C ALGORITHMS.

      INCLUDE 'PGCOMDAT.FOR'
      INCLUDE 'PGBUF.FOR'

      CALL CLEAR
      CALL INPUT
```

```
      CALL GETSCENE
      CALL OUTPUT
      CALL TESTPR

      DO 140 LINE=1,IEND+1
      CALL SLCREAT
      IF(LINE.GT.IEND) IFINISHED=1
      IF (IFINISHED .EQ. 1) THEN
            TYPE*,'THATS ALL FOLKS'
            WRITE(10,*),'FIRST SCAN LN=',ISTART,'LAST=',LINE
            TYPE*,'FIRST SCAN LN=',ISTART,'LAST=',LINE
            CALL FINFIL
            RETURN
      ENDIF
      IF(IFIRST .EQ. 1) IFIRST=0
      IF(LINE .LT.ISTART) GOTO 140
      CALL VIS
      CALL SCAN(*160)          ! RETURN TO 160 IF HORIZON LIMIT
C                                IS PASSED
      IHEIGHT=IZP
      IF(ICODE.GT.NCR) IZP=IZP-ICODE
      CALL PROJECTION(*130)    ! RETURN TO 130 IF SCREEN
C                                BOUNDARY IS PASSED
      CALL LOGLOAD
60    LOW(1)=IXP
      LOW(2)=IYP
C INCREMENT UP A WALL
      IF (ICODE .GT. NCR) THEN
        IXPINC=0
        IYPINC=0
        IZPINC=NCR
70      IZP=IZP+NCR
        CALL INCREMENT(*130,*200)     ! 130 IF END OF SCREEN
D       IF((IXBS.GT.IFILDIM).OR.(IXBS.LE.0)) TYPE*,'IXBS=',
     *    IXBS,'RUN'
D       IF((IYBS.GT.IFILDIM).OR.(IYBS.LE.0)) TYPE*,'IYBS=',
     *    IYBS,'RUN'
        ITEMBUF(IYBS,IXBS)=ITEMBUF(IYBS,IXBS)+IR
        ICNT(IYBS,IXBS)=ICNT(IYBS,IXBS)+1
        IF((IZP+NCR) .LE. IHEIGHT) GOTO 70
      ENDIF
      LOW(3)=IZP
      CALL SCAN(*160)          !160 IF HORIZON LIMIT IS PASSED
      IHEIGHT=IZP
      IF(ICODE .GT.NCR) IZP=IZP-ICODE
      IXPINC=IXP-LOW(1)
      IYPINC=IYP-LOW(2)
      IZPINC=IZP-LOW(3)
      CALL INCREMENT(*130,*200)       !130 IF END OF SCREEN
      CALL LOGLOAD
      GOTO 60
```

```
130     CONTINUE
D       TYPE*,LINE,IXBS,IYBS,IP,SUMANG
        ISKY=0
140     CONTINUE
        RETURN
C PAINT THE SKY BACKDROP
160     CONTINUE
        IP(3)=IE(3)-NCR+(IF(ICASE)*(LOW(3)-IE(3))+LOW(ICASE)/2)/
     *        LOW(ICASE)
        ISKY=1
        X=250*EXP(-IZP/45000.)
        Y=EXP(-NCR/45000.)
        IR=X*Y
        IZPINC=IZP-LOW(3)
        IXPINC=IP(1)-LOW(1)
        IYPINC=IP(2)-LOW(2)
        CALL INCREMENT(*130,*200)
        ITEMBUF(IYBS,IXBS)=ITEMBUF(IYBS,IXBS)+IR
D       IF((IXBS.GT.IFILDIM).OR.(IXBS.LE.0)) TYPE*,'IXBS=',IXBS,
     *    'RUN 7300'
D       IF((IYBS.GT.IFILDIM).OR.(IYBS.LE.0)) TYPE*,'IYBS=',IYBS,
     *    'RUN 7300'

        ICNT(IYBS,IXBS)=ICNT(IYBS,IXBS)+1
                INCSKY=NCR
        IXPINC=0
        IYPINC=0
        IZPINC=INCSKY
170     IZP=IZP+INCSKY
        CALL INCREMENT(*130,*200)     !130 IF END OF THE SCREEN
D       IF((IXBS.GT.IFILDIM).OR.(IXBS.LE.0)) TYPE*,'IXBS=',IXBS,
     *    'RUN 8400'
D       IF((IYBS.GT.IFILDIM).OR.(IYBS.LE.0)) TYPE*,'IYBS=',IYBS,
     *    'RUN 8400'
        IF((IXBS.GT.IFILDIM).OR.(IYBS.GT.IFILDIM).OR.(IXBS.LE.0)
     *  .OR.(IYBS.LE.0))THEN
           TYPE*,'ERROR COORDS OUT OF RANGE "SKY"',IXBS,IYBS
        ELSE
           ITEMBUF(IYBS,IXBS)=ITEMBUF(IYBS,IXBS)+IR
           ICNT(IYBS,IXBS)=ICNT(IYBS,IXBS)+1
        ENDIF
        X=X*Y
        IR=X
        GOTO 170
200     RETURN
        END
```

APPENDIX EH

PGPROJ.FOR

```
C
C       PGPROJ.FOR
C
C MEMORY EXTENSIVE
        SUBROUTINE PROJECTION(*)


C
C       PHILIP GATT (305-896-4741)
C       8/27/80
C
C       THIS ROUTINE COMPUTES THE SCREEN LOCATION OF A VISIBLE
C       POINT
C
C  INPUT VARIABLES ARE
C  (IVSORIZ(3),IPOINT(3),IPINC(3),IEYE(3),IROT,IROT(3,3))
C
C       IHORIZ = POSITION OF HORIZON;WORLD COORDINATES
C       IP = POSITION OF VISIBLE POINT,WORLD COORDINATES
C       IPINC=INCREMENTAL DISTANCE BETWEEN SUCCESSIVE POINTS
C             IN WORLD COORDINATES
C       IE = POSITION OF EYE;WORLD COORDINATES
C       IROT = ROTATION MATRIX DUE TO EQUATIONS OF MOTION
C              FROM THE INITIAL TO THE PRESENT ORIENTATION
C
C  OUTPUT VARIABLES ARE (IXBS,IYBS)
C
C       IXBS,IYBS = SCREEN BUFFER CO-ORDINATES OF THE VISIBLE
C                   POINT
C
C  CONSTANTS ARE (IVS,L,N,ISCALE,ICONST)
C
C       IVS = DISTANCE FROM EYE TO SCREEN
C       L = LENGTH OF THE SCREEN
C       N = DIMENSION OF THE SCREEN IN PIXELS
C       ISCALE = SCALE FACTOR FOR IROT
C       CONST=IVS*N
C
```

```
C  KEY VARIABLES ARE (IXS,IYS,INCXS,INCYS,JERRX,JERRY,IRESOL)
C
C       IXS,IYS = SCREEN CO-ORDINATES OF A VISIBLE POINT
C       INCXS,INCYS = +1 -1 OR 0, WHICH INDICATE THE DIRECTION
C                     IN WHICH THE SCREEN COORDINATES IXS,IYS
C                     ARE INCREMENTED/DECREMENTED
C       JERRX,JERRY = THE ERROR BETWEEN THE INTEGER SCREEN
C                     COORDINATE AND THE REAL SCREEN COORDINATE
C                     PROJECTED BACK TO THE VISIBLE POINT
C       IRESOL = THE RESOLUTION BETWEEN SUCCESSIVE POINTS
C
C       THIS ROUTINE USES FOUR CO-ORDINATE SYSTEMS
C               1) THE WORLD CO-ORDINATES X,Y,Z
C               2) THE EYE CO-ORDINATES U,V,W
C               3) THE SCREEN CO-ORDINATES IXS,IYS
C               4) THE SCREEN BUFFER CO-ORDINATES IXBS,IYBS
C
C       THE SCREEN IS IN THE U,W PLANE, AND THE V AXIS PASSES
C       THRU THE CENTER OF THE SCREEN. THE SCREEN HAS X AND Y
C       AXIS WITH THE ORIGIN IN THE CENTER. THE X AXIS IS IN THE
C       U DIRECTION THE Y AXIS IS IN THE -W DIRECTION.SEE FIG 1A
C
C       THE SCREEN BUFFER HAS THE SAME ORIENTATION AS THE SCREEN
C       BUT IT'S ORIGIN IS IN THE UPPER LEFT HAND CORNER. SEE
C       FIG 1B
C
C
C (1,1)                    (1,N)   (-N/2,-N/2)        ((N-1)/2,-N/2)
C       ******************> XBS          :----------------:
C       *                    :           :                :
C       *                    :           :                :
C       *                    :           :                :
C       *                    :           :                :
C       *                    :           :           **********> XS
C       *                    :           :           *
C       *                    :           :           *        :
C       *                    :           :           *        :
C       *--------------------:           :-----------*--------:
C       *                                            *
C       * YBS                                        * YS
C
C    (N,1)                  (N,N)  ((N-1)/2,-N/2) ((N-1)/2,(N-1)/2))
C
C            FIG 1A                                  FIG 1B
C
C       FOR A FULL DESCRIPTION OF THE MATHMATICS SEE THE MEMO-
C       RANDUMS DATED: 28 NOV 79; 1 FEB 80; 13 FEB 80; 30 APRIL
C       80; 1 MAY 80 PERTAINING TO THE PROJECTION PROCESSOR

        INCLUDE 'PGCOMDAT.FOR'
```

```
      IDIVNCR=IDIV1*NCR
      IDLORESOL=IRESOL
C TRANSLATE HORIZON
      IZHT=IZH-IZE
C TRANSLATE POINT
      IZPT=IZP-IZE

C ROTATE NADIR
      IUNADIR=IROT(3,1)*(-IZE)
      IVNADIR=IROT(3,2)*(-IZE)
      IWNADIR=IROT(3,3)*(-IZE)
C ROTATE HORIZON
      UHORIZ=IROT(1,1)*XH+IROT(2,1)*YH+IROT(3,1)*IZHT
      WHORIZ=IROT(1,3)*XH+IROT(2,3)*YH+IROT(3,3)*IZHT
      VHORIZ=IROT(1,2)*XH+IROT(2,2)*YH+IROT(3,2)*IZHT
C ROTATE FIRST POINT
   IUPOINT=(IROT(1,1)*IXP+IROT(2,1)*IYP+IROT(3,1)*IZPT)/IDIVNCR
   IV=((IROT(1,2)*IXP+IROT(2,2)*IYP+IROT(3,2)*IZPT))/IDIVNCR
   IWPOINT=(IROT(1,3)*IXP+IROT(2,3)*IYP+IROT(3,3)*IZPT)/IDIVNCR

C COMPUTE SIGN FOR INCREMENTAL POINTS
      SGN=SIGN(1.,VNADIR)*SIGN(1.,VHORIZ)
      DELX=IVNADIR*UHORIZ-VHORIZ*IUNADIR
      DELY=VHORIZ*IWNADIR-IVNADIR*WHORIZ
      ABSX=ABS(DELX)
      ABSY=ABS(DELY)
      IF(2*ABSY.LT.ABSX/NX) THEN
            INCYS=0
            INCXS=SIGN(1.,DELX)*SGN
      ELSE IF(2*ABSX.LT.ABSY/NY) THEN
            INCXS=0
            INCYS=SIGN(1.,DELY)*SGN
      ELSE
            INCXS=SIGN(1.,DELX)*SGN
            INCYS=SIGN(1.,DELY)*SGN
      ENDIF
C COMPUTE FIRST POINT
      IXS= ICONST*IUPOINT/IV -JCONSTX
      IYS=-ICONST*IVPOINT/IV +JCONSTY
      IXBS=(2*IXS+NX+2)/2
      IYBS=(2*IYS+NY+2)/2
      JIX=IXS+JCONSTX
      JIY=IYS-JCONSTY

C COMPUTE ERROR OF FIRST  POINT
      JERRU=ICONST*IUPOINT-JIX*IV
      JERRW=-ICONST*IWPOINT-JIY*IV
      INCX=ISIGN(1,JERRU)
      INCY=ISIGN(1,JERRW)

C NEXT PIXEL TEST FOR FIRST POINT
```

```
      IF ((2*JERRU.GE.IV) .OR. (2*JERRU.LT.-IV)) THEN
            JIX=JIX+INCX
            IXBS=IXBS+INCX
            JERRU=JERRU-IV*INCX
      ENDIF
      IF ((2*JERRW.GE.IV) .OR. (2*JERRW.LT.-IV)) THEN
            JIY=JIY+INCY
            IYBS=IYBS+INCY
            JERRW=JERRW-IV*INCY
      ENDIF

C CHECK FOR CRASH
C
D     TYPE*,'FIRST SCREEN COORDINATE=',IXBS,IYBS
100   IF(ICRASH .GE. IV) THEN
        WRITE (6,*) '"ERROR" POINT IS BETWEEN EYE AND SCREEN'
        TYPE*,'ICRASH,IV,IP,IXBS,IYBS=',ICRASH,IV,IP,IXBS,IYBS
        TYPE*,'ISCRN,IAXIS,NPIX=',ISCRN,IAXIS,NPIX
        RETURN2
      ENDIF

C THIS CALL IS A TEST USING A STRAIGHT DIVIDE TO TEST THE SCREEN
C CO-ORDINATES OF A NADIR CENTRIC WORLD POINT IP
C
D     IF(ISKY.NE.1) THEN
D       UPOINT=(ROT(1,1)*IXP+ROT(1,2)*IYP-ROT(1,3)*(IZE-IZP))
D       VL=((ROT(2,1)*IXP+ROT(2,2)*IYP-ROT(2,3)*(IZE-IZP)))
D       WPOINT=(ROT(3,1)*IXP+ROT(3,2)*IYP-ROT(3,3)*(IZE-IZP))
D       XS= ICONST*UPOINT/VL -JCONSTX
D       YS=-ICONST*WPOINT/VL +JCONSTY
D       IYYS=YS+SIGN(.5,YS)
D       IXXS=XS+SIGN(.5,XS)
D       IXXBS=IXXS+NX/2+1
D       IYYBS=IYYS+NY/2+1
D       IF((ABS(IXBS-IXXBS).GT.1).OR.(ABS(IYBS-IYYBS).GT.1))THEN
D             TYPE*,'ERROR IN SCREEN CO-ORDINATES.'
D             TYPE*,'SHOULD BE',IXXBS,IYYBS,'IS',IXBS,IYBS
D             TYPE*,'INCX,Y,JERRU,W,IV,IP=',INCXS,INCYS,JERRU,
     *                JERRW,IV,IP
D       ENDIF
D       ENDIF
C OUTPUT SCREEN BUFFER CO-ORDINATES FOR VISIBLE P
        IF(NPIX.EQ.0) THEN
              IF(ISCRN(IAXIS).GE.NPIX) RETURN1
        ELSE
              IF(ISCRN(IAXIS).GE.NPIX) RETURN1
        ENDIF

        RETURN

C=============================================================
```

```
      ENTRY INCREMENT(*)
C=====================================================
      IF(IFINISHED.EQ.1) THEN
            RETURN
      ENDIF
C CHECK FOR CHANGE IN RESOLUTION
      IF (IRESOL .NE.IOLDRESOL) THEN
            NEWRF=2**(IRESOL-IOLDRESOL)
            NU2=NEWRF/2
            JERRU=(JERRU+NU2)/NEWRF
            JERRW=(JERRW+NU2)/NEWRF
            IV=(IV+NU2)/NEWRF
            ICRASH=(ICRASH+NU2)/NEWRF
            IOLDRESOL=IRESOL
      ENDIF

      IDIO=IDIV1*NCR
C ROTATE INCREMENTAL DISTANCE FOR NEXT POINT
      IUPINC=(IROT(1,1)*IXPINC+IROT(2,1)*IYPINC+IROT(3,1)*
     *          IZPINC)/IDIO
      IVPINC=(IROT(1,2)*IXPINC+IROT(2,2)*IYPINC+IROT(3,2)*
     *          IZPINC)/IDIO
      IWPINC=(IROT(1,3)*IXPINC+IROT(2,3)*IYPINC+IROT(3,3)*
     *          IZPINC)/IDIO

C UPDATE V CO-ORDINATE
      IV=IV+IVPINC

C COMPUTE ERROR FOR NEXT POINT
      JERRU=JERRU+ICONST*IUPINC-JIX*IVPINC
      JERRW=JERRW-ICONST*IWPINC-JIY*IVPINC

C NEXT PIXEL TEST FOR NEXT POINT
      IF (INCXS .EQ. -1) THEN                              !1
            ITEST=-IV-JERRU-JERRU
            IF (ITEST.GT.0) THEN                  !2
                  JERRU=JERRU+IV
                  JIX=JIX-1
                  IXBS=IXBS-1
            ENDIF                                          !2
      ELSE IF (INCXS .EQ. 1 )THEN                          !1
            ITEST=IV-JERRU-JERRU
            IF (ITEST.LE.0) THEN      !3
                  JERRU=JERRU -IV
                  JIX=JIX+1
                  IXBS=IXBS+1
            ENDIF                                          !3
      ENDIF                                                !1
      IF (INCYS .EQ. -1) THEN                              !4
            ITEST=-IV-JERRW-JERRW
            IF (ITEST.GT.0) THEN                  !5
```

```
                JERRw=JERRW+IV
                JIY=JIY-1
                IYBS=IYBS-1
        ENDIF                                           !5
ELSE IF (IYCYS .EQ. 1 )THEN                             !4
        ITEST=IV-JERRw-JERRw
        IF (ITEST.LT.0) THEN    !6
                JERRw=JERRW -IV
                JIY=JIY+1
                IYBS=IYBS+1
        ENDIF                                           !6
ENDIF                                                   !4

GO TO 100
END
```

APPENDIX EI

SCAN.FOR

```
      SUBROUTINE SCAN(*)
      DATA N1024/1024/
      INCLUDE 'PGCOMDAT.FOR'

C THIS ROUTINE INCREMENTS ALONG THE SCAN LINE VIA A BRESENHAM
C TYPE LINE DRAWING ALGORITHM, VISIBILITY TESTS ARE PERFORMED
C FOR EACH POINT.

      KILL=0
100           IF(IEEE.GT.0)THEN
                  IP(IOTHR)=IP(IOTHR)+INCIO*NCR
                  IEEE=IEEE-IXY2
              ENDIF
              IP(ICASE)=IP(ICASE)+INC*NCR
              IEEE=IEEE+JXY2
110           DO 130 NI=1,2
              IPP(NT)=IP(NT)+IE(NI)
              IF(IPP(NT).LT.0)THEN
                  IPNP(NT)=IPP(NT)+1
                  IQ(NT)=IPNP(NI)/1024-1
                  IF(ICL.EQ.1)GOTO 130
                  DO 120 ITI=2,ICL
                      IQ(NT)=((IQ(NI)+1)/2)-1
120               CONTINUE
              ELSE
                  IPNP(NT)=IPP(NT)
                  NPHIL=N1024*NCR
                  IQ(NT)=IPNP(NI)/(NPHIL)
              ENDIF
130           CONTINUE
      IF((IABS(IQ(1)-NPN(1)).GT.1).OR.(IABS(IQ(2)-NPN(2))
     *  .GT.1))THEN
                  IF(KILL.EQ.1)THEN
                      TYPE*,'INVALID CLASS CHANGE'
                      STOP
                  ENDIF
                                KILL=1
              DO 140 NT=1,2
```

```
                IF(IE(NT).LT.0)THEN
                        NPN(NI)=((NPN(NT)+1)/2)-1
                ELSE
                        NPN(NI)=NPN(NI)/2
                ENDIF
140             CONTINUE
                LSI=NCR
                NCR=NCR*2
                ICL=ICL+1
                IP2=IPP(IOTHR)
                IP1=IPP(ICASE)
                IF(ICL.EQ.2)THEN
                  IF(IP2.EQ.(IP2/2)*2)THEN
                        IP2=IP2 -INCIO
                  ENDIF
                  IF(IP1.EQ.(IP1/2)*2)THEN
                        IP1=IP1+INC
                  ENDIF
                ELSE
                  IBM=((IP2-LSI*INCIO)/NCR)*NCR
                  IPPP1=IBM+LSI*INCIO
                  IPPP2=IBM-LSI*INCIO
                  IF((IABS(IPPP1-IP2)).GT.(IABS(IPPP2-IP2)))THEN
                     IP2=IPPP2
                  ELSE
                     IP2=IPPP1
                  ENDIF
                  ICBM=((IP1+LSI*INC)/NCR)*NCR
                  IPP1=ICBM+LSI*INC
                  IPP2=ICBM-LSI*INC
                  IF((IABS(IPP1-IP1)).GT.(IABS(IPP2-IP1)))THEN
                        IP1=IPP2
                  ELSE
                        IP1=IPP1
                  ENDIF
                ENDIF
                IP1=IP1-IE(ICASE)
                IP2=IP2-IE(IOTHR)
                IEEE= -IDXY +(IP1*JXY2*INC-IP2*IXY2*INCIO)/NCR
                IF(IEEE.GT. 0)THEN
                        IP2=IP2+INCIO*NCR
                        IEEE=IEEE-IXY2
                ENDIF
                IF(IEEE.LT.-IXY2)THEN
                        TYPE*,'IEEE .LT.-IXY2 =',IEEE,'IP1,IP2='
     *                        ,IP1,IP2
                        WRITE(10,*)'IEEE .LT.-IXY2 =',IEEE,'Ip1,
     *                             IP2=',IP1,IP2
                        IEEE=IEEE+IXY2
                        IP2=IP2-INCIO*NCR
                ENDIF
```

```
                IP(ICASE)=IP1
                IP(IOTHR)=IP2
                IEEE=IEEE+JXY2
                GOTO 110
        ENDIF


C THIS IS A TEST WHICH VERIFIES THE IOTHR POINT GIVEN AN ICASE
C POINT
D       LSI=NCR/2
D       OTHER=1.*JXY2/IXY2*IP(ICASE)*INC*INCID +IE(IOTHR)
D       ITEST=OTHER +SIGN(.5,OTHER)
D       N=(ITEST-LSI)/NCR
D       ITEST=N*NCR-LSI
D       TEST1=ABS(ITEST-OTHER)
D       TEST2=ABS(ITEST-OTHER+NCR)
D       IF(TEST2 .LT. TEST1) THEN
D               ITEST=ITEST+NCR
D               IF(TEST2 .GT. ABS(ITEST-OTHER +NCR)) THEN
D                       ITEST=ITEST+NCR
D               ENDIF
D       ENDIF
D       IF(ITEST.NE.IPP(IOTHR))THEN
D          IF(ABS(1.*(ITEST+IPP(IOTHR))/2 -OTHER).GT..001) THEN
D             TYPE*,'LSI,OTHER=',LSI,OTHER
D             TYPE*,'ERROR IPP(OTHR)=',IPP(IOTHR),'SHOULD BE',
     *                ITEST
D             TYPE*,'IP=',IP,'N=',N,'NCR=',NCR
D             TYPE*,'IEEE=',IEEE,'IP1=',IP1,'IP2=',IP2
D             WRITE(10,*)'LSI,OTHER=',LSI,OTHER
D             WRITE(10,*)'ERROR IPP(OTHR)=',IPP(IOTHR),
     *                   'SHOULD BE',ITEST
D             WRITE(10,*)'IP=',IP,'N=',N,'NCR=',NCR
D             WRITE(10,*)'IEEE=',IEEE,'IP1=',IP1,'IP2=',IP2
D          ENDIF
D       ENDIF
!
        KILL=0
!
! LVVP=ALTITUDE OF PREVIOUS NON-VISIBLE TEST POINT SUPPLIED
! BY ZRDATA.
!
        LVVP=IP(3)
!
!ZRDATA IS A SUBROUTINE WHICH GIVEN A DATA BASE POINT CALCULATES
!THE ALTITUDE AND REFLECTANCE OF THE CORRESPONDING TEST POINT.
!
        GOTO (1001,1002,1003,1004,1005),IDATBAS
1001    CALL ZRDATA(1010)
1002    CALL ZRDAT2(1010)
1003    CALL ZRDAT3(1010)
```

```
1004     CALL ZRDAT4(1010)
1005     CALL ZRDAT5(1010)
1010     CONTINUE
!
!
! NC=A PROPER TEST LINE COORDINATE SUPPLIED BY ZRDATA.
!
! VISIBILITY TEST INTRODUCED    (SEE STATEMENT OF WORK)
!
         KK=(IE(3)-LOW(3))*IP(ICASE)
         JJ=LOW(ICASE)*(IE(3)-IP(3))
         IF(INC.GT.0)THEN
                  IF(IP(ICASE).GT.IF(ICASE))GOTO 400
                  IF(JJ.GE.KK)GOTO 100
         ELSE
                  IF(IP(ICASE).LT.IF(ICASE))GOTO 400
                  IF(JJ.LE.KK)GOTO 100
         ENDIF
!
! ROUTINE HAS FOUND A VISIBLE TEST POINT .
!
!
         IF(IP(3).LE.LNVP) THEN
                  ICODE=1
         ELSE
                  KIE=((IE(3)-LOW(3))*IP(ICASE))/LOW(ICASE)
                  KZCALC=IE(3)-KIE
                  IF(IE(3).GT.KIE) KZCALC=KZCALC+1
                  ICODE=IP(3)-KZCALC
                  IF(ICODE.LE.0) ICODE=1
         ENDIF
         RETURN
400      CONTINUE          !PAST HORIZON LIMIT
         RETURN1
       END
```

APPENDIX EJ

PGSCENE.FOR

```
C
C       PGSCENE.FOR
C PROGRAMED BY PHILIP GATT 896-4741
C LATEST REVISION DATE : MARCH 26 1981
C FOR A FULL DESCRIPTION OF THE MATHEMATICS SEE THE MEMO FROM
C DR. B.W.PATZ DATED 5/30/80
C THIS PROGRAM SETS UP THE FOLLOWING SCENE CONSTANTS:
C       1. THE ROTATION MATRIX.  ROT
C       2. THE LOCATION OF THE SCREEN CORNERS IN EYE CENTRIC
C          WORLD COORDINTAES.  COR(4,3)
C       3. THE LOCATION OF THE SCREEN CORNERS WHEN PROJECTED
C          TO THE GROUND.  CW(4,2)
C       4. A VECTOR WHICH INDICATES WHETHER OR NOT THE SCREEN
C          CORNERS PROJECT OT THE GROUND.  PROJ(4)
C          PROJ(*) = 0 => PROJECTS TO THE HORIZON LIMIT
C          PROJ(*) = 1 => PROJECTS TO THE GROUND
C       5. THE LINES BETWEEN THE FOUR CORNERS ON THE GROUND
C          AA(4,2)
C       6. THE ANGLE BETWEEN SCAN LINES, SUCH THAT AT LEAST TWO
C          SCAN LINES CUT THROUGH EACH PIXEL.  ANG


       SUBROUTINE GETSCENE

       INCLUDE 'PGCOMDAT.FOR'
       DIMENSION CPROJ(4),CROSS(4)
C
       HORIZONLIM=45000.
       NPL=IFILDIM-1
       NX=LX*NPL
       NY=LY*NPL
C COMPUTE SINES AND COSINES
       PITCH1=PITCH*3.141592654/180
       HEADING1=HEADING*3.141592654/180
       BANK1=BANK*3.141592654/180
       SH=SIN(HEADING1)
       SP=SIN(PITCH1)
       SB=SIN(BANK1)
```

```
          CH=COS(HEADING1)
          CP=COS(PITCH1)
          CB=COS(BANK1)
C         COMPUTE THE ROTATION MATRIX
          ROT(1,1)=CB*CH-SH*SB*SP
          ROT(1,2)=-SH*CB-SB*SP*CH
          ROT(1,3)=-SB*CP
          ROT(2,1)=CP*SH
          ROT(2,2)=CP*CH
          ROT(2,3)=-SP
          ROT(3,1)=SB*CH+CB*SP*SH
          ROT(3,2)=-SH*SB+CB*SP*CH
          ROT(3,3)=CB*CP

C CALCULATE THE SCREEN CORNERS IN EYE CENTRIC WORLD COORDINATES
C COR(4,3)
          DO 100 I=1,3
          A1=IUS*ROT(1,I)
          A2=IWS*ROT(3,I)
          A3=IVS*ROT(2,I)
          A4=GX*ROT(1,I)/2
          A5=GY*ROT(3,I)/2
          X=A1+A2+A3
          Y1=A4+A5
          Y2=A4-A5
          COR(1,I)=X+Y1
          COR(2,I)=X-Y2
          COR(3,I)=X-Y1
          COR(4,I)=X+Y2
100       CONTINUE
C COMPUTE THE DISTANCE FORM THE NADIR TO THE CORNERS WHEN
C DROPPED TO THE FLAT EARTH CMAG(4)
          DO 200 I=1,4
          CMAG(I)=0
          DO 150 J=1,2
          X=COR(I,J)
          CMAG(I)=CMAG(I) + X*X
150       CONTINUE
          CMAG(I)=SQRT(CMAG(I))
200       CONTINUE

C COMPUTE THE CROSS PRODUCTS OF THE SCREEN CORNERS CROSS(4)
          CROSS(1)=COR(4,1)*COR(1,2)-COR(4,2)*COR(1,1)
          DO 300 I=2,4
          K=I-1
          CROSS(I)=COR(K,1)*COR(I,2)-COR(K,2)*COR(I,1)
300       CONTINUE

C COMPUTE THE ANGLE TO EACH SCREEN CORNER FROM THE NADIR CANG(3)
          DO 350 I=1,4
          CANG(I)=ATAN2(COR(I,2),COR(I,1))
```

```
350     CONTINUE
        DO 375 I=1,3
        CANG(I)=CANG(I)-CANG(4)
        IF(CANG(I).LT.0) CANG(I)=CANG(I)+2*3.141592654
        IF(CANG(I).LT.0) TYPE *,'ERROR IN CANG(I)=',CANG(I)
375     CONTINUE
C COMPUTE ANGLE BETWEEN SUCCESSIVE SCAN LINES
        ANG=ANGFACTOR*(LX*LX + LY*LY)/(3*CMAG(1)*CMAG(1)*NX)
C COMPUTE WHERE IS THE NADIR
        IF(CROSS(4) .LT. 0) THEN
                NADIR=0          !NADIR IS OUTSIDE
        ELSE
                NADIR=1          !NADIR IS INSIDE
        ENDIF

C PROJECT THE SCREEN CORNERS TO THE FLAT EARTH
C CW(I,*) IS THE NADIR CENTRIC COORDINATES OF THE INTERCEPT OF
C THE LINE FROM THE EYE THROUGH THE CORNER (COR(I,*)) ONTO THE
C GROUND PLANE. IF SUCH AN INTERCEPT OCCURS THEN PROJ(I)=1 ELSE
C PROJ(I)=0 IF PROJ(1)=0 THEN CW(1,*) HAS A LENGTH CORESPONDING
C TO THE HORIZON LIMIT.
        DO 400 I=1,4
        IF(COR(I,3) .GE. 0)THEN
                PROJ(I)=0   !THE CORNER PROJECTS TO THE HORIZON

                SCALE=HORIZONLIM/CMAG(I)
                CW(I,1)=SCALE*COR(I,1)
                CW(I,2)=SCALE*COR(I,2)
        ELSE
                PROJ(I)=1 !THE CORNER PROJECTS TO THE FLAT EARTH
                CPROJ(I)=IZE/COR(I,3)
                CW(I,1)=-COR(I,1)*CPROJ(I)
                CW(I,2)=-COR(I,2)*CPROJ(I)
        ENDIF
400     CONTINUE
                DO 500 I=1,2
                AA(1,I)=CW(1,I)-CW(4,I)
                AA(2,I)=CW(2,I)-CW(1,I)
                AA(3,I)=CW(3,I)-CW(2,I)
                AA(4,I)=CW(4,I)-CW(3,I)
                IF (NADIR .EQ. 0) THEN
                        AA(4,I)=-AA(4,I)
                ENDIF
500             CONTINUE
        DO 600 I=1,3
        DO 600 J=1,3
        IROT(J,I)=ROT(I,J)*ISCALE
600     CONTINUE
        SQRA=ANG**2
        ICRASH=IVS*ISCALE/IDIV1
        ICOUNT=IVS*APL
```

```
      JCONSTX=IOS*NPL
      JCONSTY=IAS*NPL
      IFIRST=1
      IFINISHED=0
      IAXIS=1
      NPTX=NX+1
      SUMANG=0
      KSL=1
C KSL IS THE CORNER COUNTER 1 THRU 4
      IASO=0
      IYSO=0
      KCOUNT=1
      IPOWER=1
      NC=0
      IRUN=1
      IFILTER1=0
      IFILTER2=0
      RETURN
      END
```

APPENDIX EK

PGSL.FOR

```
C       PGSL.FOR
C
C PROGRAMED BY PHILIP GATT 896-4741
C LATEST REVISION DATE MARCH 6 1981
C THIS ROUTINE COMPUTES THE SCAN LINES FOR A SCENE ONE AT A TIME
C THE SCREEN CORNERS ARE NUMBERED COUNTERCLOCKWISE 1-4 STARTING
C AT THE UPPER RIGHT HAND CORNER
C
C OUTPUT VARIABLES ARE (XH,YH,UPPER,RLOWER,IFINISHED)
C   IXH,IYH - DEFINE THE LOCATION OF THE LOWEST RESOLUTION RANGE
C   IUPPER  - DEFINES THE UPPER BOUND FOR A PARTICULAR SCAN LINE
C   LOWER   - DEFINES THE LOWER BOUND FOR A PARTICULAR SCAN LINE
C   IFINISHED - DEFINES WHEN THE LAST LINE HAS BEEN COMPLETED
C INTERNAL VARIABLES USED FOR INDICES ARE:
C       KSL     IDENTIFIES THE CORNER NOUMBER
C       ISL     IDENTIFIES THE X OR Y COMPONENT TO BE TESTED
C       LSL     IS A SIGN MULTIPLIER.  LSL HAS THE EFFECT OF
C               ABSOLUTE VALUE
C
C FOR A DESCRIPTION OF THE MATHEMATICS SEE THE MEMO FROM
C DR. H. W. PATZ DATED 5-30-80

      SUBROUTINE SLCREAT
      INCLUDE 'PGCOMDAT.FOR'
      DIMENSION RLOWER(3)
C T IS A FUNCTION WHICH COMPUTES THE INTERCEPT OF THE SCAN LINE
C WITH THE SCREEN FOOTPRINT ON THE GROUND.
      T(ANG,X,Y,A,B)=ANG*((X**2+Y**2)/(X**2-Y*A))*(1+ANG*
     *                (A*X+B*Y)/(B*X-A*Y))
C
      IF (IFIRST .EQ.1) THEN
              IF (NADIR.EQ.0)THEN      !NADIR IS OUTSIDE
                      RLOWER(1)=CN(4,1)
                      RLOWER(2)=CN(4,2)
              ELSE
                      RLOWER(1)=0
                      RLOWER(2)=0
              ENDIF
```

```
C CW(I,*) IS THE NADIR CENTRIC COORDINATES OF THE INTERCEPT OF
C THE LINE FROM THE EYE THROUGH THE CORNER (COR(I,*)) ONTO THE
C GROUND PLANE.  IF SUCH AN INTERCEPT OCCURS THEN PROJ(I)=1 ELSE
C PROJ(I)=0. IF PROJ(I)=0 THEN CW(I,*) HAS A LENGTH CORESPONDING
C TO THE HORIZON LIMIT
C PROJECT THE RADIUS THROUGH CW4
                SCALE=HORIZONLIM/CMAG(4)
                XH=COR(4,1)*SCALE
                YH=COR(4,2)*SCALE
        ELSE
                IF (NADIR .EQ.0) THEN
                   TL=T(ANG,RLOWER(1),RLOWER(2),AA(4,1),AA(4,2))
                ELSE
                   TL=0
                ENDIF

                RLOWER(1)=RLOWER(1)+AA(4,1)*TL
                RLOWER(2)=RLOWER(2)+AA(4,2)*TL
C CORNER CROSS TEST
        SUMANG=SUMANG+ANG
        IF(CANG(KSL) .LT. SUMANG) THEN
                TYPE*,'CROSSED CORNER ',KSL,'AT LINE ',LINE
                TYPE*,'WITH LAST POINT,SCREEN AND WORLD COORDS='
     *                 ,IXBS,IYBS,IP
                IF (KSL .EQ. (3+NADIR)) THEN   !NADIR=1,0/IN,OUT
                                IFINISHED=1
                                TYPE*,'LAST LINE IS COMPLETED'
                                RETURN
                ENDIF
                KSL=KSL+1
                GOTO(100,200,300,400),KSL
C  NPIX IS INITIALIZED INSUBR INPUT IN PROG PGMAIN
100             TYPE*,'ERROR KSL =1 =',KSL
                GO TO 500
200             IAXIS=2
                NPIX=0
                GOTO 500
300             IAXIS=1
                NPIX=0
                GOTO 500
400             IAXIS=2
                NPIX=NY+1
500     CONTINUE
        ENDIF

C ICAS SELECTS ONE OF THREE ROTATION MATRIX APROXIMATIONS
                IF (ICAS.EQ.1)THEN
                        X=XH - YH*ANG
                        Y=XH*ANG + YH*(1-SQRA)
                ELSE IF (ICAS .EQ. 2) THEN
                        X=XH*(1-SQRA) - YH*ANG
```

```
                Y=XH*ANG +YH
        ELSE IF (ICAS .EQ. 3) THEN
                X=XH*(1-SQRA/2) - YH*ANG
                Y=XH*ANG +YH*(1-SQRA/2)
        ELSE
                TYPE*,'ERROR ICAS IS NOT 1,2,3 BUT IS',
     *                 ICAS
        ENDIF

    XH=X
    YH=Y
    ENDIF
    IXH=XH+SIGN(.5,XH)
    IYH=YH+SIGN(.5,YH)
    LOW(1)=RLOWER(1) + SIGN(.5,RLOWER(1))
    LOW(2)=RLOWER(2) + SIGN(.5,RLOWER(2))
    LOW(3)=0
    RETURN
    END
```

APPENDIX E1

PGVIS.FOR

```
C       PGVIS.FOR
        SUBROUTINE VIS

C THIS SUBROUTINE WILL SET UP THE BRESENHAM ALGORITHM IN SCAN.
C THE OUTPUT VARIABLES ARE:
C       IEEE    THE ERROR TERM FOR THE BRESENHAM ALGORITHM
C       IXY2    THE DISTANCE TO THE HORIZON ALONG THE ICASE AXIS
C       IDXY    ONE HALF OF IXY2
C       JXY2    THE DISTANCE TO THE HORIZON ALONG THE IOTHR AXIS
C
        INCLUDE 'PGCOMDAT.FOR'

        NCR=1
C FOR A GOOD DESCRIPTION OF IOTHR AND ICASE SEE SUBROUTINE SCAN.
        IF(IABS(IYH).GT.IABS(IXH))THEN
                ICASE=2
                IOTHR=1
        ELSE
                ICASE=1
                IOTHR=2
        ENDIF
C
C DETERMINING ICASE AND IOTHR FOR A GIVEN SCAN LINE USING
C BRESENHAM'S ALGORITHM WHICH USES A SCALING FACTOR OF TWO.
C (SEE STATEMENT OF WORK)
        ALPHAV=-4.5/IF(ICASE)
        IXY2=IF(ICASE)
        JXY2=IF(IOTHR)
        IF(IXY2.LT.0)THEN
                IXY2=-IXY2
                INC=-1
        ELSE
                INC=1
        ENDIF
        IF(JXY2.LT.0)THEN
                JXY2=-JXY2
                INCIO=-1
        ELSE
```

```
                INCIO=1
        ENDIF
        IDXY=IXY2/2
        IEEE=JXY2-IDXY
        DO 100 NT=1,2
                IP(NT)=0
                IPP(NT)=IE(NT)
                IPNP(NT)=IE(NT)
                IF(IE(NT) .LT.0)IPNP(NT)=IPNP(NT)+1
                NPN(NT)=IPNP(NT)/1024
                IF(IE(NT).LT.0) NPN(NI)=NPN(NT)-1
                IQ(NT)=NPN(NT)
100     CONTINUE
        ICL=1
        IF(IDATBAS.EQ.1) CALL ZRDATA
        IF(IDATBAS .EQ.2) CALL ZRDAT2
        IF(IDATBAS .EQ.3) CALL ZRDAT3
        IF(IDATBAS .EQ. 4) CALL ZRDAT4
        IF(IDATBAS .EQ. 5) CALL ZRDAT5
C
C IF THE VIEWER IS WITHIN THE LEVEL SURFACE NO VISIBLE POINTS
C WILL BE TRANSMITTED BY SCAN.  I.E. IF A PLANE FLEW INTO THE
C SIDE OF A MOUNTAIN OR INTO THE OCEAN, ETC.  SEE STATEMENT
C OF WORK DONE BY TERRY TANZEY.
C
        IF(IF(3).LT.IP(3))THEN
                TYPE*,'EYE IS INSIDE THE DATA BASE'
                TYPE*,'AT THE WORLD POINT',IPP,IP(3)
                STOP
        ENDIF
        RETURN
        END
```

APPENDIX EM

TYPICAL USE OF REAL SCAN PROGRAMS

The routines are compiled with the aid of a command file called PGFOR.COM, which compiles all the routines for the user by typing @PGFOR. A listing of PGFOR.COM is given below:

```
$ON ERROR THEN CONTINUE

$FOR PGSCAN

$FOR/DLINES PGPROJ

$FOR PGSL

$FOR PGSCENE

$FOR PGVIS

$FOR PGDATA

$FOR PGFILTER

$FOR/DLINES PGMAIN

$FOR PGLOGLOAD

$FOR PGCAM

$PURGE *.OBJ

$EXIT
```

The FORTRAN files are linked together by the command file PGMAIN.COM. The user can link these files by typing @PGMAIN. This file is listed below:

```
$LINK-

PGMAIN,PGSCENE,PGSL,PGPROJ,PGFILTER,PGDATA,PGVIS,PGSCAN,PGLOGLOAD,PGCAM,-

DIC:BLKDAT,DIC:DICOMED/LIB
```

The user can create a picture by typing RUN PGMAIN and by typing in the required input information of Table 1. A sample picture is given in Figure EM-1.

TABLE EM-1. Input Information For The Sample Scene.

PROMPT	RESPONSE
INPUT SCREEN SIZE	512
ENTER COMMAND	1 (RUN SCENE)
DATABASE CHOICE	1
STARTING AND ENDING SCAN LINE	1,2000
PROJECTION DIVISION FACTOR	3
A FACTOR FOR THE SCAN LINE ANGLE	1
ICASE (ROTATION MATRIX APPROXIMATION)	3
SCALE FACTOR FOR THE ROTATION MATRIX	2 500
THE EYE LOCATION	0,200,800
THE SCREEN CENTER	0,1,0
THE SCREEN DIMENSION (LX,LY)	1,1
PITCH, BANK, AND HEADING IN DEGREES	5,0,0
ENTER COMMAND	3 (RUN DICOMED)
ENTER COMMAND	6 (STOP)

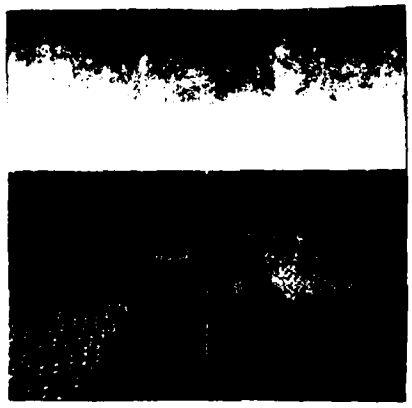


Figure EM-1. Sample Picture.

The input information for this picture is listed in Table EM-1.

After all the input information has been entered, the routines will create the scene data. This data can then be written to a file or a Polaroid picture can be created.

APPENDIX FA

SECOND GENERATION REALSCAN ROUTINES

Appendix FB through FM contain the routines that are considered the second generation of REALSCAN.

APPENDIX F6

PZBUF.FOR

```
C       PZBUF.FOR

        INTEGER*2 ITEMBUF(3,512,512),INT2(512,256)
        BYTE ICNI(512,512)
        COMMON/BUF/ ITEMBUF,ICNI
        EQUIVALENCE (ICNI,INT2)
```

APPENDIX FC

PZCOMDAT.FOR

```
C
C       PZCOMDAT.FOR
C
C       THIS IS A COMMON BLOCK  FOR THE ROUTINES WHICH GENERATE
C       A COLOR PICTURE OF A 3 DIMENSIONAL SCENE


        INTEGER*2 ISCRN(2),ICOLOR,IRV(3)

        COMMON/POINTS/IPP(2),IE(3),IHORIZ(2),IP(3),IOTHR,ICASE
        COMMON/INTG/ISCRN,IXPINC,IYPINC,IZPINC,
     *  IFILDIM,IRV,NCR,NX,NY
        COMMON/REAL1/ALPHAV,SUMANG,HORIZONLIM,XBS,YBS
        COMMON/FLAGS/ IDATBAS,IFIR,ICOLOR,LINE

        EQUIVALENCE (IR,IRV(1))

C IPP IS THE DATA POINT IN WORLD COORDINATES,
C IP IS THE SCALED DATA POINT IN EYE CENTRIC WORLD COORDINATES
C (12 BITS), LOWN IS THE SCALED LAST VISIBLE DATA POINT IN EYE
C CENTRIC WORLD COORDINATES (12 BITS), IHORIZ IS THE END OF
C THE SCAN LINE IN EYE CENTRIC WORLD COORDINATES, IE IS THE
C LOCATION OF THE OBSERVER IN WORLD COORDINATES.
```

APPENDIX FD

PZDATA3.FOR

```
C       PZDATA3.FOR
C                    ****************************************
C                    *                                      *
C                    *         SUBROUTINE ZRATA             *
C                    *                                      *
C                    *            DATA BASE  1              *
C                    *                                      *
C                    ****************************************
C
C            LATEST REVISION DATE:     SEPTEMBER 30,1980
C            PROGRAMMED BY:  GERALD L. BECKER
C<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C  THIS IS TO BE USED WITH THE DICOMED!!!!!!!!
C<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
C            THIS SUBROUTINE IS CALLED BY SUBROUTINE 'SCAN'.
C            THIS SUBROUTINE DEALS WITH DATA BASE 1.  DATA BASE 1
C      IS MADE UP OF BLOCKS WITH 3 POSSIBLE ALTITUDES.  POSSIBLE
C      ALTITUDES ARE HIGH (512), LOW (-128), AND GROUND (0).
C      ALL CALCULATIONS IN ZRDATA ARE MADE WITH INTEGER MATH.
C        1   2   3   4
C      ****************M2      THE ENTIRE DATA BASE 1 IS COVERED
C      *   *   *   *   *  WITH DATA BLOCKS AS SHOWN TO THE LEFT.
C      * 1 * 8 * 8 * 2 *1 BLOCKS OF TYPE 1 DATA HAVE A HEIGHT OF
C J=  *  1*  2*  3*  4*  512.  BLOCKS OF TYPE 2 DATA HAVE A
C      **************** HEIGHT OF -128.  THE REFLECTANCE IS A
C      *   *   *   *   *  FUNCTION OF POSITION WITHIN THE FRAME.
C      * 8 * 2 * 1 * 8 *2 BLOCKS OF TYPE 4 DATA HAVE ZERO HEIGHT
C J=  *  5*  6*  7*  8*  AND A REFLECTANCE OF 4.
C      **************** DEFINITIONS:
C      *   *   *  X*   *  CLASS: DIFFERENT LEVELS OF DETAIL OR
C      * 1 * 8 * 8 * 2 *3 RESOLUTION.
C      *   *   *   *   *  FRAME COORDINATES: M1 AND M2 VALUES
C      **************** SPECIFYING THE BLOCK BEING USED.  EX-
C      *   *   *   *   *  AMPLE: POINT X HAS FRAME COORDINATES
C      * 8 * 1 * 2 * 8 *4 M1=3, M2=3.  FRAME COORDINATES ARE
C      *   *   *   *   *  CLASS DEPENDENT.
C      **************** ARRAY INDICES: POSITION WITHIN A GIVEN
C                        FRAME.
```

```
C      M1=MB1
C
         SUBROUTINE ZRDATA(*)
         INCLUDE 'PZCONDAT.FOR'
         COMMON/DATABASE/N1024,IOFF1(3),IOFF2(3),SCALE(3),
      *  IBRFLD(3),W1,SB0VA,COVMAG,IDOV2,KSI,KSI4,KSI5,KSI7,KSI8,
      *  IDENSITY(3),SB4OVA,SB7OVA,DNO1(3),W,MAXHEIGHT
         COMMON /GRND/ INX,INY
         INTEGER*2 INDEXX,INDEXY,IDD,NCR1
      *  ,INX,INY,KL1,KL2,KL3,KH1,KH2,KH3,IBL
         DIMENSION IPNP(2)
         DATA SCSAM /.9/
C MAXHEIGHT MUST BE SET FOR THE SCAN LINE INITILIZATION
         MAXHEIGHT=512
         NCR1=NCR
C BUILDINGS WITH ROADS AND 2 NOISE TYPES AND MULTIPLE RETURN
C
C IP := NADIR CENTRIC COORDINATES
C IPP := WORLD CENTRIC COORDINATES
C IPP = IP + IE
C IF IPP<0 ;IPNP=IPP+1
C  ELSE      IPNP=IPP
         DO 123 NN=1,2
         IPNP(NN)=IPP(NN)
         IF(IPP(NN).LT.0)IPNP(NN)=IPNP(NN)+1
123      CONTINUE
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
         NPHIL=N1024*NCR1
                  MIX=MOD(IPNP(1),(NPHIL))
                  MJY=MOD(IPNP(2),(NPHIL))
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C      ARRAY INDICES FOR ALL CLASSES ARE CALCULATED.
C
C                  THE RANGE OF INDEXX,INDEXY IS ALWAYS 1<=1024.
C        THE GRID BOUNDARIES OCCUR ON IDD LINES.  THE IDEA OF
C        INDEXX, INDEXY COORDINATES IS TO MATCH DATA COMPRESSION
C        INFORMATION AS IT WOULD BE ACCESSED IN A HIERARCHIAL
C        DATABASE.  TO DEFINE BOUNDARIES ONE NEEDS, IDD, MB1,MB2,
C        THE DISPLACEMENT FROM IDD IN THE COORDINATE FRAME.  ALL
C        DISPLACEMENTS ARE POSITIVE MODULAR ARITHMETIC.
                  IF (IPP(1).GE.0) THEN
                       INDEXX=(MIX/NCR1)+1
                  ELSE
                       INDEXX=(MIX/NCR1)+1024
                  ENDIF
                  IF (IPP(2).GE.0) THEN
                       INDEXY=(MJY/NCR1)+1
                  ELSE
                       INDEXY=(MJY/NCR1)+1024
```

```
                ENDIF
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C       BLOCK ASSIGNMENTS ARE DEPENDANT ON WHETHER OR NOT THE TEST
C       POINT IS WITHIN THE VALNCR1 RANGE.
C
                IDD=1024/NCR1
          IF(ICL.GT.9) THEN
                TYPE*,'ERROR ICL .GT. 9 ; ZRDATA 9400'
                STOP
          ENDIF
          IBL=IDD-1
          INX=IMOD(INDEXX-1,IDD)
          INY=IMOD(INDEXY-1,IDD)
C THIS ROUTINE ASSUMES NCR1=2**(ICL-1)
          KL1=IMOD(64/NCR1,IDD)
          KL2=IMOD(128/NCR1,IDD)
          KL3=IMOD(160/NCR1,IDD)
          KH1=IMOD((1024-64)/NCR1,IDD)-1
          KH2=IMOD((1024-128)/NCR1,IDD)-1
          KH3=IMOD((1024-160)/NCR1,IDD)-1
C         KL1 THROUGH KL3 SET THE LOWER BOUNDS
C         KH1 THROUGH KH3 SET THE UPPER BOUNDS
                MB1=IPNP(1)/1024
                MB2=IPNP(2)/1024
                IREM=MOD(MB1,4)
                JREM=MOD(MB2,4)
                IF (IPP(1).LT.0) IREM=IREM+3
                IF (IPP(2).LT.0) JREM=JREM+3
                J=JREM+(4*IREM)+1
          GOTO (1010,1020,1020,1040,8,1060,1070,8,1090,1100,1100,
     *         1120,1130,1140,1150,1160),J
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C       IP(3) AND IR ASSIGNMENTS ARE MADE.  THIS ASSIGNMENT IS
C       DEPENDENT ON THE FRAME INDICES.
C
1010      CONTINUE
          CALL GTD(KL2,KL2,IBL,IBL,*9,*8,*8)
          IF(INX.LT.KL3) GOTO 8
          IP(3)=512
          CALL CRSIFA(KL3,KL2,IBL,IBL,*6,*4,*5)
          GOTO 7
1020      CONTINUE
          IF(INX.LT.KL2) GOTO 9
          GOTO 8
1040      CONTINUE
          CALL GTD(KL2,0,IBL,KH1,*9,*6,*8)
          IF(INX.LT.KL3) GOTO 8
          IP(3)=-128
          CALL  CRSIFA(KL3,0,IBL,KH1,*6,*4,*5)
```

```
      GOTO 3
1060  CONTINUE
      CALL GND(KL1,0,IBL,KH1,*8,*6,*8)
      IF(INX.LT.KL1) GOTO 8
      IF (INY.GT.KH1) GOTO 8
      IP(3)=-128
      CALL CRSIFA(KL1,0,IBL,KH1,*6,*4,*5)
      GOTO 3
1070  CONTINUE
      CALL GND(0,0,IBL,IBL,*8,*8,*8)
      IP(3)=512
      CALL CRSIFA(0,0,IBL,IBL,*6,*4,*5)
      GOTO 7
1090  CONTINUE
      CALL GND(KL1,KL1,KH2,KH1,*8,*9,*8)
      IF (INX.GT.KH3) GOTO 8
      IP(3)=512
      CALL  CRSIFA(KL1,KL1,KH3,KH1,*6,*4,*5)
      GOTO 7
1100  CONTINUE
      IF(INX.GT.KH2) GOTO 9
      GOTO 8
1120  CONTINUE
      IF(INX.GT.KH2) GOTO 9
      IF(INX.GT.KH3) GOTO 8
      IP(3)=-128
      CALL CRSIFA(0,0,KH3,IBL,*6,*4,*5)
      GOTO 3
1130  CONTINUE
      CALL GND (KL2,0,KH2,IBL,*9,*9,*8)
      GOTO 8
 1140 CONTINUE
D     TYPE*,'KL2,KH2,KH1=',KL2,KH2,KH1
          CALL GND(KL2,0,KH2,KH1,*9,*9,*8)
      IF(INX.GT.KH3) GOTO 8
      IF(INX.LT.KL3) GOTO 8
      IP(3)=512
      CALL CRSIFA(KL3,0,KH3,KH1,*6,*4,*5)
      GOTO 7
1150  CONTINUE
      CALL GND(KL2,KL1,KH2,KH1,*9,*9,*8)
      IF(INX.LT.KL3) GOTO 8
      IF(INX.GT.KH3) GOTO 8
      IP(3)=-128
      CALL CRSIFA(KL3,KL1,KH3,KH1,*6,*4,*5)
      GOTO 3
1160  CONTINUE
      IF(INX.LT.KL2) GOTO 9
      IF(INX.GT.KH2) GOTO 9
      GOTO 8
  3   IR=0  !  BOTTOM OF HOLE
```

```
            GOTO 99
    4       CONTINUE
C           SIDES;X CONSTANT,Y VARIES
            IF(INX.LT.IBL/2) THEN
            IR=158
            ELSE
            IR=98
            ENDIF
            IF(IP(3).EQ.-128)IP(3)=0
                  GO TO 99
    5       CONTINUE
C           FRONT AND BACK FACES;Y CONSTANT,X VARIES
D           TYPE*,' "5" '
            IF(INY.LT.IBL/2) THEN
            IR=76
            ELSE
            IR=180
            ENDIF
            IF(IP(3).EQ.-128) THEN
            IP(3)=0
            IR=(IR-127)*.75+127
            ENDIF
          GO TO 99
C           CORNER
    6       CONTINUE
D           TYPE*,' "6"'
            IR=127
            GO TO 99
   7      CONTINUE
C       ROOF TEXTURE
            IFIR=4
            RETURN 1
C GROUND TEXTURE
    8       CONTINUE
          IP(3)=0
            IFIR=3
            RETURN 1
C ROAD
    9       CONTINUE
D           TYPE*,' "9" ,INDEXX,INDEXY=',INDEXX,INDEXY
            IR=70
            IP(3)=0
            IF(INX.LT.(8/NCR1)) IR=200
            IF(INX.GT.(IDD-8/NCR1)) IR=200
            IF(INX.LT.(4/NCR1)) IR=150
            IF(INX.GT.(IDD-4/NCR1)) IR=150
            IF(NCR1.EQ.8) THEN
                    IF(INX.EQ.0) IR=167
                    IF(INX.EQ.IDD) IR=167
            ENDIF
D           TYPE*,'IR9=',IR
```

```
99      IFIR=1
        RETURN 1
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
        ENTRY  ZRROF(*)
        P11=90./72.
        IR=(P11*(SIN(.092*IPP(1)+.04*IPP(2)-1.)+3.*SIN(
     *  .26*IPP(1)-.103*IPP(2)+.5)+5.*SIN(.432*IPP(1)+
     *  .197*IPP(2)+.8))*(SIN(.086*IPP(2)-.037*IPP(1)-.96)+
     *  3.*SIN(.253*IPP(2)+.096*IPP(1)+.45)+4.*SIN(.385*
     *  IPP(2)-.166*IPP(1))))
C ALPHAV =-4.5/(HORIZON(ICASE) AND IS SET UP IN VIS
        TERM=EXP(ALPHAV*(IPP(ICASE)-IE(ICASE)))
        IR=165+IR*TERM
        IF(IP(3) .EQ. -128) IR=0
        IF(IR.GT.255) TYPE*,'950/2 ZRDATA IR,IP=',IR,IP
        RETURN 1
C
C     GROUND COMPUTATION
C
        ENTRY ZRGND(*)
C        NOISE ON GROUND
CC      INDX= ABS(MOD((IPP(2)-IPP(1)/32),2044))
CC      INDY= ABS(MOD((IPP(1)-IPP(2)/128),2044))
CC      IF(INDX .LT. 1024) THEN
CC              INDX= 1 + INDX/2
CC      ELSE
CC              INDX= 1023 - INDX/2
CC      ENDIF
CC      IF(INDY .LT. 1024) THEN
CC              INDY= 1 + INDY/2
CC      ELSE
CC              INDY= 1023 - INDY/2
CC      ENDIF
CC      IR=127+(SCSAM*(IDAT(INDX,INDY,1)+128)-127)*
CC      EXP(-.00005*IP(2))
        IR=127
        RETURN 1
        END
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
        SUBROUTINE CRSIFA(LX,LY,HX,HY,*,*,*)
C
        INTEGER*2 INX,INY,LX,LY,HX,HY
        COMMON /GRND/ INX,INY
C       CORNER TEST
        IF(((INX .EQ.LX) .OR. (INX .EQ. HX)) .AND.
     *    ((INY.EQ.LY) .OR. (INY .EQ. HY)))  RETURN 1
C
C-----------------------------------------------------------
```

```
C
C       SIDE TEST
C
        IF((INX .EQ.LX  ) .OR. (INX .EQ. HX))  RETURN 2
C
C-------------------------------------------------------------
C
C       FACE TEST
C
        IF((INY .EQ.LY) .OR. (INY .EQ. HY))  RETURN 3
C
C-------------------------------------------------------------
C
        RETURN
        END
C
C
C=============================================================
C
C
        SUBROUTINE GND(LX,LY,HX,HY,*,*,*)
C
C
        INTEGER*2 INX,INY,LX,LY,HX,HY
C
        COMMON /GRND/ INX,INY
C
C
        IF(INX .LT. LX) RETURN 1
        IF(INX .GT. HX) RETURN 2
        IF(INY .LT. LY) RETURN 3
        IF(INY .GT. HY) RETURN 3
C
        RETURN
        END
```

APPENDIX FE

PZFILTER.FOR

```
C       PZFILTER.FOR

        SUBROUTINE FINFIL
        INCLUDE 'PZCOMDAT.FOR'
        INCLUDE'PZBUF.FOR'

100     TYPE*,'WE ARE IN THE FILTER STAGE'
        DO 300 J=1,IFILDIM
        DO 300 I=1,IFILDIM
        KDIV=ICNT(I,J)
        IF(KDIV.NE.0) THEN
                IF(KDIV.LT.0) KDIV=KDIV+128
                DO 200 KVS=1,ICOLOR
                ITEMBUF(KVS,I,J)=(ITEMBUF(KVS,I,J)+KDIV/2)/KDIV
200             CONTINUE
        ENDIF
300     CONTINUE
        RETURN

        ENTRY CLEAR
        DO 400 KVS=1,ICOLOR
        DO 400 I=1,IFILDIM
        DO 400 I=1,IFILDIM
        IF(KVS.EQ.1)ICNT(I,J)=0
        ITEMBUF(KVS,I,J)=0
400     CONTINUE
        RETURN
        END
```

APPENDIX FF

PZLOGLOAD.FOR

```
C PZLOGLOAD

      SUBROUTINE LOGLOAD
      INCLUDE 'PZBUF.FOR'
      INCLUDE 'PZCOMDAT.FOR'
      COMMON/LOAD/IXSO,IYSO   !COMMON WITH GETSCENE

      KCOUNT=KCOUNT+1
      IF((ISCRN(1).NE.IXSO).OR.(ISCRN(2).NE.IYSO)) THEN
              IPOWER=2
              KCOUNT=1
              IXSO=ISCRN(1)
              IYSO=ISCRN(2)
              GOTO 100
      ELSE IF(IPOWER.EQ.KCOUNT) THEN
              IPOWER=2*IPOWER
100           GOTO(600,200,300,400,500),IFIR ! GET REFLECTANCE
200           CALL COLORIR(*600)
300           CALL ZRGND(*600)
400           CALL ZRROF(*600)
500           CALL COLORIIR(*600)
600           CONTINUE
      IF(ICNT(ISCRN(2),ISCRN(1)) .GT.126 ) RETURN
      IF((ISCRN(1).GE.(IFILDIM + 1)).OR.(ISCRN(1).LE.0)) THEN
              RETURN
      ELSE IF((ISCRN(2).GE.(IFILDIM+1)).OR.(ISCRN(2).LE.0))
     *  THEN
              RETURN
      ENDIF
      DO 700 KVS=1,ICOLOR
      ITEMBUF(KVS,ISCRN(2),ISCRN(1))=ITEMBUF(KVS,ISCRN(2),
     *  ISCRN(1))+IRV(KVS)
700   CONTINUE
      ICNT(ISCRN(2),ISCRN(1))=ICNT(ISCRN(2),ISCRN(1))+1
      ENDIF
      RETURN
      END
```

APPENDIX FG

PZMAIN.FOR

```
C       PZMAIN.FOR
C
        INCLUDE 'PZCOMDAT.FOR'
        INCLUDE 'PZBUF.FOR'
C IFILDIM IS THE FILE DIMENSION AND THE SCREEN DIMENSION + 1
C IF WE WISH TO HAVE A 2*SCREEN OF DATA PER LINE THEN IFILDIM =
C 1024 AND ANGLEFACT=1 ?? SINCE ANGLEFACT IS A FUNCTION OF NX.
100     TYPE*,' DO YOU WANT COLOR (3) OR BLACK AND WHITE (1) ?'
        ACCEPT*,ICOLOR
        IF((ICOLOR.EQ.1).OR.(ICOLOR.EQ.3)) THEN
        ELSE
                TYPE*,'ICOLOR=',ICOLOR
                GOTO 100
        ENDIF
200     TYPE*,'   ENTER COMMAND BY NUMBER:  '
        TYPE*,'         (1) INITIALIZE SCENE AND RUN MAIN (NOTE:'
        TYPE*,'             FILTER 1 EXECUTED AUTOMATICALLY)'
        TYPE*,'         (2) RUN DICOMED PICTURE OF BUFFER'
        TYPE*,'         (3) WRITE BUFFER AND ICNT'
        TYPE*,'         (4) STOP'
        ACCEPT*, ICOMAND
        GOTO (300,400,500,600) ICOMAND
300     CALL RUN
        GOTO 200
400     CONTINUE
        CALL SDICSW(0,0,0)
        GOTO 200
500     CONTINUE
        TYPE*,' WHICH FILE DO YOU WANT TO WRITE ITEMBUF AND'
        TYPE*,' ICNT INTO ?'
        ACCEPT*,IFILE
        TYPE*,' WRITING ITEMBUF INTO FILE ',IFILE
        WRITE(IFILE,700) (((ITEMBUF(K,I,J),I=1,IFILDIM),J=1,
     *  IFILDIM),K=1,ICOLOR)
        TYPE*,' WRITING ICNT INTO FILE ',IFILE
        WRITE(IFILE,800) ((ICNT(J,I),J=1,IFILDIM),I=1,IFILDIM)
        GOTO 200
600     STOP
```

```
600     FORMAT(132A1)
700     FORMAT(6642)
        END

        SUBROUTINE RUN
C THIS ROUTINE WILL CONTROL THE SEQUENCING OF THE REAL SCAN
C ALGORITHMS.

        INCLUDE 'PZCOMDAT.FOR'
        INCLUDE 'PZBUF.FOR'
        COMMON/SUN/A,B,C,SMARG,SKYSCL,DIVSKY
        COMMON/MAIN/ISTART,IEND
        COMMON/VIS/DOWN(3),IHIEGHT

C CLEAR MAKES THE ARRAY'S ITEMBUF AND ICNT=0
        CALL CLEAR

C  GETSCENE INITIALIZES MOST PICTURE PARAMETERS
        CALL GETSCENE
        DO 400 LINE=1,IEND+1

C  SLINIT INITIALIZES EACH SCAN LINE
        CALL SLINIT(*900)       !900 IF FINISHED 50 IF NADIR = 0
        IF(LINE.GT.IEND) GOTO 900
        IF(LINE .LT.ISTART) GOTO 400

C  SCAN DETERMINES THE VERY FIRST VISIBLE POINT.
        CALL SCAN(*500)         ! RETURN TO 500 IF HORIZON LIMIT
C                                 IS PASSED

C  PROJ PLACES THE FIRST VISIBLE POINT ON THE SCREEN.

        CALL PROJECTION(*300)   ! RETURN TO 300 IF SCREEN
C                                 BOUNDRY IS PASSED

C LOGLOAD ACCUMULTES THE REFLECTANCE DATA IN ITEMBUF

        CALL LOGLOAD

100     IF(IP(3)+2/2 .LE. IHIEGHT) THEN !INCREMENT UP A VERTICAL
C                                        SURFACE
                IXPINC=0
                IYPINC=0
                IZPINC=4/2
                IP(3)=IP(3)+4/2
                CALL INCREMENT(*300)    ! 300 IF END OF SCREEN
                IF(ICNT(ISCRN(2),ISCRN(1)) .GT.126) GOTO 100
                DO 200 KVS=1,ICOLOR
        ITEMBUF(KVS,ISCRN(2),ISCRN(1))=ITEMBUF(KVS,ISCRN(2),
     *  ISCRN(1))+IRV(KVS)
200             CONTINUE
```

```
                ICNT(ISCRN(2),ISCRN(1))=ICNT(ISCRN(2),ISCRN(1))
     *          +1
                GO TO 100
           ENDIF
           LOWN(1) = IP(1)
           LOWN(2) = IP(2)
           LOWN(3) = IP(3)
           CALL SCAN(*500)          !500 IF HORIZON LIMIT IS PASSED
           IXPINC = IP(1)-LOWN(1)
           IYPINC = IP(2)-LOWN(2)
           IZPINC = IP(3)-LOWN(3)
           CALL INCREMENT(*300)     !300 IF END OF SCREEN
           CALL LOGLOAD
           GOTO 100
300        CONTINUE
D          TYPE*,LINE,ISCRN,XBS,YBS
400        CONTINUE
           RETURN

C PAINT THE SKY BACKDROP
500        CONTINUE
           IP(3)=4*IHORIZ(ICASE)*LOWN(3)/(NCR*LOWN(ICASE))
           X=(7+(A*IP(1)+B*IP(2))*DIVSKY)*
     *          31.9*EXP(-(IP(3)*NCR+IE(3)*4)*SKYSCL/NCR)
           Y=EXP(-1*SKYSCL)
           IR=X*Y
           IF(ICOLOR .EQ.3) THEN
                IRV(1)=0
                IRV(2)=0
                IRV(3)=IR
           ENDIF
           IXPINC=IP(1)-LOWN(1)
           IYPINC=IP(2)-LOWN(2)
           IZPINC=IP(3)-LOWN(3)
           CALL INCREMENT(*300)
           DO 600 KVS=1,ICOLOR
           ITEMBUF(KVS,ISCRN(2),ISCRN(1))=ITEMBUF(KVS,ISCRN(2),
     *     ISCRN(1))+IRV(KVS)
600        CONTINUE
           IF(ICNT(ISCRN(2),ISCRN(1)).LT.0) THEN
              ICNT(ISCRN(2),ISCRN(1))=ICNT(ISCRN(2),ISCRN(1))+1
           ELSE
              ICNT(ISCRN(2),ISCRN(1))=ICNT(ISCRN(2),ISCRN(1))-127
           ENDIF
           IXPINC=0
           IYPINC=0
           IZPINC=4/2
700        IP(3)=IP(3)+4/2
           CALL INCREMENT(*300)     !300 IF END OF THE SCREEN
           IF((ISCRN(1).GE.(IFILDIM + 1)).OR.(ISCRN(1).LE.0)) THEN
                GOTO 850
```

```
      ELSE IF((ISCRN(2).GE.(IFILDIM+1)).OR.(ISCRN(2).LE.0))
     *  THEN
              GOTO 850
      ENDIF
      DO 800 KVS=1,ICOLOR
      ITEMBUF(KVS,ISCRN(2),ISCRN(1))=ITEMBUF(KVS,ISCRN(2),
     *  ISCRN(1))+IRV(KVS)
800   CONTINUE
      IF(ICNT(ISCRN(2),ISCRN(1)).LT.0) THEN
          ICNT(ISCRN(2),ISCRN(1))=ICNT(ISCRN(2),ISCRN(1))+1
      ELSE
          ICNT(ISCRN(2),ISCRN(1))=ICNT(ISCRN(2),ISCRN(1))-127
      ENDIF
850   X=X*Y
      IR=X
      IF(ICOLOR .EQ. 3) THEN
              IRV(1)=0
              IRV(3)=IR
      ENDIF
      GOTO 700

900   TYPE*,'TH-TH-TH-TH-THATS ALL FOLKS'
      WRITE(10,*),'FIRST SCAN LN=',ISTART,'LAST=',LINE
      TYPE*,'FIRST SCAN LN=',ISTART,'LAST=',LINE
      CALL FINFIL
      RETURN
      END
```

APPENDIX FH

PZMULT.FOR

```
C PZMULT.FOR
C CREATED BY PHILIP GAIT ON JUNE 25 1981
C THIS IS A MULTIPLY ROUTINE WHICH NEGLECTS OVERFLOW
C AND IS USED BY THE PROJECTION PROCESSOR
C THIS ROUTINE MUST BE COMPILED WITH THE QUALIFIER
C /CHECK=NOOVERFLOW

      SUBROUTINE MULT(ICON,IUP,IWP,IVP,JX,JY,IRESU,IRESW)
      IRESU = ICON*IUP-JX*IVP
      IRESW = -ICON*IWP-JY*IVP
      RETURN
      END
```

APPENDIX F1

PZNOISE.FOR

```
C       PZNOISE.FOR
C       THIS IS A COMMON BLOCK FOR THE NOISE DATA

        BYTE IDAT(512,512,3)
        COMMON/NOISE/IDAT
```

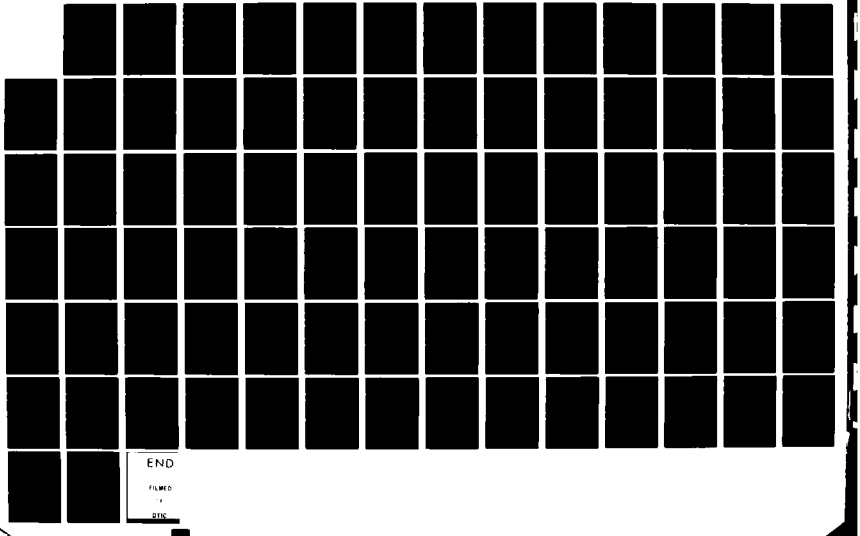
APPENDIX FJ

PZPROJ.FOR

```
C       PZPROJ.FOR
C
C MEMORY EXTENSIVE
      SUBROUTINE PROJECTION(*)
C
C       PHILIP GATT (305-896-4741)
C       THIS ROUTINE COMPUTES THE SCREEN LOCATION OF A VISIBLE
C       POINT.
C       INPUT VARIABLES ARE: IP(3),IPINC(3),IE(3),IROT(3,3)
C               IP = POSITION OF VISIBLE POINT IN EYE CENTRIC
C                    WORLD COORDS SCALED TO APPROXIMATLY 12
C                    BITS AND IN NCR UNITS.
C               IPINC=INCREMENTAL DISTANCE BETWEEN SUCCESSIVE
C                    POINTS SCALED THE SAME AS IP.
C               IE = POSITION OF EYE; WORLD COORDINATES
C               IROT = ROTATION MATRIX DUE TO EQUATIONS OF
C                    MOTION FROM THE INITIAL TO THE PRESENT
C                    ORIENTATION SCALED BY ISCALE : USUALLY
C                    4096.
C       OUTPUT VARIABLES ARE: ISCRN(2)
C               ISCRN = SCREEN BUFFER CO-ORDINATES OF THE
C               VISIBLE POINT.
C
C       CONSTANTS ARE: ICONST,JCONSTX,JCONSTY (INITIALIZED IN
C       PZSCENE)
C               ICONST=NPL*IVS
C               JCONST=NPL*IUS
C               JCONSTY=NPL*IWS
C               WHERE NPL= PIXELS PER UNIT LENGTH, AND THE
C               SCREEN CENTER IS AT (IUS,IVS,IWS) IN EYE COORDS
C       KEY VARIABLES ARE (IXS,IYS,JERRU,JERRW)
C               IXS,IYS = SCREEN CO-ORDINATES OF A VISIBLE POINT
C               JERRU,JERRW = THE SCALED ERROR BETWEEN THE
C                             INTEGER SCREEN CO-ORDINATE AND THE
C                             REAL SCREEN CO-ORDINATE PROJECTED
C                             BACK TO THE VISABLE POINT.
C       THIS ROUTINE USES FOUR CO-ORDINATE SYSTEMS
C               1) THE WORLD CO-ORDINATES X,Y,Z
```

AD-A122 000 REAL SCAN EVOLUTION(U) UNIVERSITY OF CENTRAL FLORIDA 1/6
ORLANDO DEPT OF ELECTRICAL ENGINEERING B W PATZ ET AL.
FEB 82 NAVTRAEQUIPC-80-D-D014-2 N61339-80-D-0014

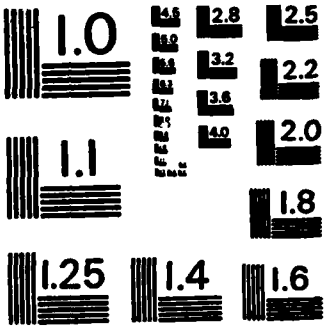
UNCLASSIFIED F/G 9/2 NL



END

FILMED

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

```
C                2) THE EYE CO-ORDINATES U,V,W
C                3) THE SCREEN CO-ORDINATES IXS,IYS
C                4) THE SCREEN BUFFER CO-ORDINATES ISCRN(1),
C                   ISCRN(2)
C        THE SCREEN IS IN THE U,W PLANE, AND IS CENTERED AT
C        (IUS,IVS,IWS).  IT HAS X AND Y AXES WITH THE ORIGIN
C        IN THE CENTER.
C        THE X AXIS IS PARALLEL TO THE U AXIS OF THE EYE, WHILE
C        THE Y AXIS IS PARALLEL TO THE -W AXIS OF THE EYE.  SEE
C        FIG. 1A
C        THE SCREEN BUFFER HAS THE SAME ORIENTATION AS THE SCREEN
C        BUT IT'S ORIGIN IS IN THE UPPER LEFT HAND CORNER. SEE
C        FIG. 1B

         INCLUDE 'PZCOMDAT.FOR'
         COMMON/PROJ/JCONSTX,JCONSTY,ICONST,IROT(3,3),ROI(3,3)
         COMMON/PROJSCAN/JERRU,JERRW,IVPOINT,ICRASH
         COMMON/PROJ1/NPIX,IAXIS

C ROTATE FIRST POINT
         IUPOINT=IROT(1,1)*IP(1)+IROI(2,1)*IP(2)+IROT(3,1)*Ip(3)
         IVPOINT=IROT(1,2)*IP(1)+IROI(2,2)*IP(2)+IROT(3,2)*Ip(3)
         IWPOINT=IROT(1,3)*IP(1)+IROI(2,3)*IP(2)+IROT(3,3)*Ip(3)
C COMPUTE SCREEN COORDINATES FOR THE FIRST POINT
         IXS = 1.*ICONST*IUPOINT/IVPOINT
         IYS =-1.*ICONST*IWPOINT/IVPOINT
         ISCRN(1)=(2*(IXS-JCONSTX)+NX+2)/2
         ISCRN(2)=(2*(IYS+JCONSTY)+NY+2)/2
C COMPUTE ERROR OF FIRST POINT
         CALL MULT(ICONST,IUPOINT,IWPOINT,IVPOINT,IXS,IYS,JERRU,
     *   JERRW)
C NEXT PIXEL TEST FOR FIRST POINT
100      IF ((2*JERRU.GE.IVPOINT).OR.(2*JERRU.LT.-IVPOINT)) THEN
                  INCX=ISIGN(1,JERRU)
                  IXS=IXS+INCX
                  ISCRN(1)=ISCRN(1)+INCX
                  JERRU=JERRU-IVPOINT*INCX
         ENDIF
         IF ((2*JERRW.GE.IVPOINT).OR.(2*JERRW.LT.-IVPOINT)) THEN
                  INCY=ISIGN(1,JERRW)
                  IYS=IYS+INCY
                  ISCRN(2)=ISCRN(2)+INCY
                  JERRW=JERRW-IVPOINT*INCY
         ENDIF
C CHECK FOR CRASH
         IF(ICRASH .GE. IVPOINT) THEN
                  WRITE (6,*) '                "ERROR"'
                  WRITE (6,*) ' POINT IS BETWEEN EYE AND SCREEN '
                  RETURN2
         ENDIF
```

```
C TEST SCREEN COORDINATE BY USING A DIVIDE
      CALL TESTPR

C OUTPUT SCREEN BUFFER CO-ORDINATES FOR VISIBLE POINT
      IF(NPIX.EQ.0) THEN
            IF((ISCRN(1).LE.0).OR.(ISCRN(2).LE.0)) RETURN1
      ELSE
            IF(ISCRN(IAXIS).GE.NPIX) RETURN1
      ENDIF

      RETURN

C=================================================
      ENTRY INCREMENT(*)
C=================================================

C ROTATE INCREMENTAL DISTANCE FOR NEXT POINT
      IUPINC = IROT(1,1)*IXPINC+IROT(2,1)*IYPINC+IROT(3,1)
     *  *IZPINC
      IVPINC = IROT(1,2)*IXPINC+IROT(2,2)*IYPINC+IROT(3,2)
     *  *IZPINC
      IWPINC = IROT(1,3)*IXPINC+IROT(2,3)*IYPINC+IROT(3,3)
     *  *IZPINC

C UPDATE V CO-ORDINATE
      IVPOINT=IVPOINT+IVPINC

C COMPUTE ERROR FOR NEXT POINT
C MULT IS A MULTIPLY ROUTINE WHICH NEGLECTS OVERFLOW
C MULT MUST BE COMPILED AS FOR/CHECK=NOOVERFLOW BPMULT
      CALL MULT(ICONST,IUPINC,IWPINC,IVPINC,IXS,IYS,IRESULTU,
     *  IRESULTW)
D     IF(ABS(IRESULTU) .GT. IVPOINT/2) THEN
D           TYPE*,'ERROR NCR TO LARGE U =',NCR,'AT',ISCRN
D           TYPE*,'IPP=',IPP
D     ENDIF
D     IF(ABS(IRESULTW) .GT. IVPOINT/2) THEN
D           TYPE*,'ERROR NCR TO LARGE W =',NCR,'AT',ISCRN
D           TYPE*,'AT IPP=',IPP
D     ENDIF
      JERRU = JERRU+IRESULTU
      JERRW = JERRW+IRESULTW

      GOTO 100

      END
```

APPENDIX FK

PZSCAN.FOR

```
C       PZSCAN.FOR
C THIS ROUTINE DETERMINES THE SCANLINE SEQUENCE OF POINTS AND
C CALLS THE DATA BASE TO DETERMINE HEIGHT, AND PERFORMS THE
C VISIBILITY TEST.
C INPUT VARIBLES (IPP(2).LOWN(3),EEEE,JXY2,IXY2)
C  IPP IS THE DATA POINT IN WORLD COORDINATES
C  LOWN IS THE LAST VISIBLE POINT SCALED TO APPROXIMATLY 12 BITS
C  EEEE IS THE BRESENHAM ERROR TERM
C  IXY2 IS THE LENGTH OF THE FAST AXIS SCALED TO 12 BITS
C  JXY2 IS THE LENGTH OF THE SLOW AXIS SCALED TO 12 BITS
C OUTPUT VARIABLES (IP(3),IHEIGHT)
C       IP IS THE NEW DATA POINT IN EYE CENTRIC WORLD
C       COORDINATES SCALED TO 12 BITS
C       IHEIGHT IS THE EYE CENTRIC HIEGHT OF THE ENVIROMENT AT
C       IP(1),IP(2)

        SUBROUTINE SCAN(*)

        INCLUDE 'PZCOMDAT.FOR'
        COMMON/PROJSCAN/JERRU,JERRW,IV,ICRASH
        COMMON/SCAN/INC,INCIO,INC4,INC4IO,INCR,INCRIO,
     *  INHORIZ(2),IEEE,NCRL
        COMMON/VIS/LOWN(3),IHIEGHT
        COMMON/SLSTAR/NCRLIM,XLN2,IHGT,IXY2,JXY2,IDXY,DELTA
100     CONTINUE
              IPP(ICASE) = IPP(ICASE)+INCR
              IF(IPP(ICASE)*INC.GE.INHORIZ(ICASE)*INC) RETURN1
              IP(ICASE) = IP(ICASE)+INC4
              IEEE = IEEE+JXY2
              IF(IEEE.GT.0)THEN
                    IPP(IOTHR) = IPP(IOTHR)+INCRIO
                    IP(IOTHR) = IP(IOTHR)+INC4IO
                    IEEE = IEEE-IXY2
                ENDIF
C IF IP(ICASE) EXCEEDS THE VALUE OF 4*NCRLIM THEN IT IS TIME TO
C GO TO THE NEXT HIERARCHY OF DATA.  THIS MEANS WE INCREASE THE
C FAKE EXPONENT "ICL" BY 1 AND ALL NORMALIZED VALUES CHANGE BY
C A FACTOR OF 2.
```

```
      IF(4*NCRL.LE.IP(ICASE)*INC) THEN
              NCRL=1.*NCRL/DELTA !THERORETICALY DELTA = 1.414
              NCR = NCR*2
              ICL = ICL+1
              INCR = INCR*2
              INCRIO = INCRIO*2
              JERRU = JERRU/2
              JERRW = JERRW/2
              IV = IV/2
              ICRASH = ICRASH/2
              DO 200 I = 1,3
                      IP(I) = IP(I)/2
                      DOWN(I) = DOWN(I)/2
200           CONTINUE
      IEEE = -IDXY+(IP(ICASE)*JXY2*INC-IP(IOTHR)*IXY2*INCIO)/4
              IF(IEEE.GE.0) THEN
              TYPE*,'IEEE >0 SCAN, IPP=',IPP,'IP=',IP,
     *        'IEEE=',IEEE
                      IEEE = IEEE-IXY2
                      IPP(IOTHR) = IPP(IOTHR)+INCRIO
                      IP(IOTHR) = IP(IOTHR)+INC4IO
              ENDIF
              IF(IEEE.LT.-IXY2) THEN
              TYPE*,'IEEE<-IXY2 SCAN, IPP=',IPP,'IP=',IP,
     *        'IEEE=',IEEE
                      IEEE = IEEE+IXY2
                      IPP(IOTHR) = IPP(IOTHR)-INCRIO
                      IP(IOTHR) = IP(IOTHR)-INC4IO
              ENDIF
      ENDIF
D     OTHER = 1.*IHORIZ(IOTHR)*IP(ICASE)/IHORIZ(ICASE)
D     TEST = ABS(ABS(IP(IOTHR))-ABS(OTHER))
D     IF(TEST.GT.(4.*NCR)) THEN
D             TYPE *,'SCAN ERROR IP SHOULD BE',IP(ICASE),OTHER
D             TYPE*,'BUT IS ',IP,'AND NCR=',NCR
D     ENDIF

      GOTO (1001,1005),IDATBAS
1001  CALL ZRDATA(1010)
1005  CALL COLORDATA(1010)
1010  CONTINUE

C  ZRDAT* RETURNS IP(3), THE LOCAL HEIGHT OF THE WORLD POINT.
C   SINCE WE ARE WORKING IN NORMALIZED COORDINATES, IP(3)
C   MUST BE CONVERTED TO IP(3), WHICH IS IN EYE CENTRIC
C   COORD'S AND IN "NCR" UNITS.

      LASTIP=IHIEGHT
      IHIEGHT = IP(3)-IE(3)
      IHIEGHT = (IHIEGHT+ISIGN(NCR/2,IHIEGHT))*4/NCR
```

```
KK = LOWN(3)*IP(ICASE)
JJ = LOWN(ICASE)*IHIEGHT
IF(INC.GT.0)THEN
        IF(JJ.LT.KK)GOTO 100     !NOT VISIBLE
ELSE
        IF(JJ.GT.KK)GOTO 100     !NOT VISIBLE
ENDIF

IF(IHIEGHT.LE.LASTIP) THEN
        IP(3)=IHIEGHT
ELSE
        RIP = 1.*KK/LOWN(ICASE)
        IP(3) = (RIP + SIGN(.5,RIP))
ENDIF
RETURN
END
```

APPENDIX FL

PZSCENE.FOR

```
C       PZSCENE.FOR
C
C PROGRAMED BY PHILIP GATT 896-4741
C LATEST REVISOIN DATE :JUNE 10 1981 BY P. GATT
C THIS ROUTINE INITIALIZES MOST SCENE PARAMATERS AND HANDLES ALL
C INPUT AND OUTPUT DATA.
C THIS PROGRAM SETS UP THE FOLLOWING SCENE CONSTANTS:
C   1. THE ROTATION MATRIX.  ROT
C   2. THE LOCATION OF THE SCREEN CORNERS IN EYE CENTRIC
C       WORLD COORDINTAES.  COR(4,3)
C   3. THE DISTANCE FROM THE NADIR TO THE DROPPED CORNERS.
C       CMAG(4)
C   4. THE CROSS PRODUCT OF THE FOUR CORNERS.  CROSS(4)
C   5. THE ANGLE OF EACH CORNER RELATIVE TO CORNER  4
C       CANG(3)
C   6. THE ANGLE BETWEEN SCAN LINES, SUCH THAT AT LEAST TWO
C       SCAN LINES CUT THROUGH EACH PIXEL.  ANG
C   9. THE LINES BETWEEN THE FOUR CORNERS ON THE GROUND AA(4,2)
C   10. CONSTANTS FOR THE FOLOWING ROUTINES : PGPROJ, PGSL,
C                       PGLOGLOAD
C
      SUBROUTINE GETSCENE

      INCLUDE 'PZCOMDAT.FOR'
      INCLUDE 'PZNOISE.FOR'

      COMMON/DATABASE/N1024,IOFF1(3,2),IOFF2(3,2),SCALE(3,2),
     *  IBRFLD(3),SROVA,COVMAG,IDOV2,KSI,KSI4,KSI5,KSI7,KSI8,
     *  IDENSITY(3,2),SR4OVA,SR7OVA,DNOI(3,2),W1,W,IPIL,LSCAL,
     *  IWAVSCAL,MAXHEIGHT
      COMMON/REAL/ANG,SQRA,AA(4,3),CANG(4),CMAG(4),CORUSE(4,3)
      COMMON/SL/NADIR,KSL,IFIRSI,NEXTCOR
      COMMON/PROJ/JCONSTX,JCONSTY,ICONST,IROT(3,3),ROI(3,3)
      COMMON/PROJ1/NPIX,IAXIS
      COMMON/LOAD/IXSO,IYSO
      COMMON/SLSTAR/NCRLIM,XLN2,IHGT,IXY2,JXY2,IDXY,DELTA
      COMMON/SUN/A,B,C,SMARG,SKYSCL,DIVSKY
```

```
      COMMON/MAIN/ISTART,IEND

      DIMENSION CROSS(4),COR(4,3)

      DATA N1024/1024/,IFILDIM/512/
C     DATA IBITSCALE/4096/
C     DATA NCRLIM/1414/
      DATA IDXY/32000/

      DATA SCALE/.03,.5,8.,.12,2.5,31/
      DATA W1/2.849003E-6/,W/8.950406E-6/    !W1=1/117000
      DATA IDENSITY/1,7,733,3,31,2483/
      DATA LSCAL/2/ , IWAVSCAL/47/
      TYPE*,'INPUT IBITSCALE; NCRLIM, DELTA'
      ACCEPT*,IBITSCALE,NCRLIM,DELTA

      IXY2=2*IDXY
      XLN2=1./ALOG(2.)
C
      JIIJ=99917
      DO 1668 I=1,3
      DO 1668 J=1,2
      X=RAN(JIIJ)
      IOFF1(I,J)=JIIJ
      X=RAN(JIIJ)
      IOFF2(I,J)=JIIJ
1668  CONTINUE
      TYPE*,'Type the polar sun angle from the Z axis
     * (DEGREES)'
      ACCEPT*,TZDEG
      TZRAD=TZDEG/57.3
      SINZ=SIN(TZRAD)
      TYPE*,'Type the cylindrical sun angle from the X axis to
     * ward the Y axis (DEGREES)'
      ACCEPT*, TXDEG
C
C
      TXRAD=TXDEG/57.3
      C=COS(TZRAD)
      A=SINZ*COS(TXRAD)
      B=SINZ*SIN(TXRAD)
C
C       W1=W1*3
C       W =W *3
C
      SMARG=SQRT(A*A+B*B)
      SBOVA=B/A
      COVMARG=C/SMARG
      IDOV2=IDENSITY(2,2)/2
      KSI=SCALE(2,2)*IDENSITY(2,2)
      KSI4=3*IDENSITY(2,2)
```

```
      KSI5=4*IDENSITY(2,2)
      KSI7=5*IDENSITY(2,2)
      KSI8=6*IDENSITY(2,2)
      IBRFLD(1)=20000000
      IBRFLD(2)=60*SCALE(2,2)  ! THIS SETS BEACH BOUNDRY AT
C     ABOUT 90*2.5=210 UNITS OR 21 FEET.
      IBRFLD(3)=IBRFLD(2)-150
      DO 311 I=1,3
      DO 311 J=1,2
      X=IDENSITY(I,J)
311   DNO1(I,J)=1./X
      SB4OVA=SBOVA*3
      SB7OVA=SBOVA*5
300   TYPE*,'INPUT DATABASE CHOICE 1=BUILDING(Bw)  2=COLORDAT'
      ACCEPT*,IDATBAS
      IF((IDATBAS.LT.1).OR.(IDATBAS.GT.2)) GOTO 300
      TYPE*,'INPUT STARTING SCANLINE  ,ENDING  '
      TYPE*,'INPUT SCALE FACTOR FOR ANGLE BETWEEN SCAN LINES'
      ACCEPT*,ISTART,IEND,ANGFACTOR
      TYPE*,'INPUT THE EYE LOCATION XE,YE,ZE'
      TYPE*,'INPUT THE SCREEN CENTER IUS,IVS,IWS'
      TYPE*,'INPUT THE SCREEN DIMENSIONS (PIXELS) NX,NY'
      TYPE*,'INPUT THE SCREEN LENGTH ALONG THE X AXIS NOTE'
      TYPE*,'Y LENGTH IS COMPUTED SINCE WE WANT SQUARE PIXELS'
      ACCEPT*,IE,IUS,IVS,IWS,NX,NY,RLX
      TYPE*,'INPUT PITCH,BANK,HEADING IN DEGREES'
      ACCEPT*,PITCH,BANK,HEADING
      DO 400 K = 1,3
      TYPE*,'DO YOU WANT TO READ IN TEXTURE NOISE IDAT',K,'?
     *  YES=1'
      ACCEPT*,IYES
      IF(IYES.EQ.1) THEN
              TYPE*,' WHICH FILE DO YOU WANT TO READ NOISE
     *        FROM (40:49)'
              ACCEPT*,IFILE
              TYPE*,' READING IDAT',K,'FROM FILE ',IFILE
              READ(IFILE,500) ((IDAT(I,J,K),I=1,512),J=1,512)
      ENDIF
400   CONTINUE
500   FORMAT(132A1)
      IPP(2)=IE(2)
      IPP(1)=IE(1)
      NCR=1
      IF(IDATBAS .EQ. 1) CALL ZRDATA
      IF(IDATBAS .EQ. 2) CALL COLORDATA
C IF THE VIEWER IS WITHIN THE LEVEL SURFACE NO VISIBLE POINTS
C WILL BE TRANSMITTED BY SCAN.  I.E. IF A PLANE FLEW INTO THE
C SIDE OF A MOUNTAIN OR INTO THE OCEAN, ETC.  SEE STATEMENT OF
C WORK DONE BY TERRY TANZEY.
      TYPE*,'ELEVATION UNDER THE EYE IS',IP(3)
      WRITE(10,*)'ELEVATION UNDER THE EYE IS',IP(3)
```

```
      IF(IE(3).LT.IP(3))THEN
              TYPE*,'EYE IS INSIDE THE DATA BASE'
              TYPE*,'AT THE WORLD POINT',IPP,IP(3)
              STOP
      ENDIF

C MAXHEIGHT SOULD BE RETURNED FROM DATABASE ON THE FIRST CALL
      IHGT=MAXHEIGHT-IE(3)
      TYPE*,'MZXHEIGHT,IE(3),IHGT=',MAXHEIGHT,IE(3),IHGT
      HORIZONLIM=1200*IE(3)**.5
      TYPE*,'BPSCENE HORIZONLIM=',HORIZONLIM
      NPL=NX/RLX
      RLY=NY*RLX/NX

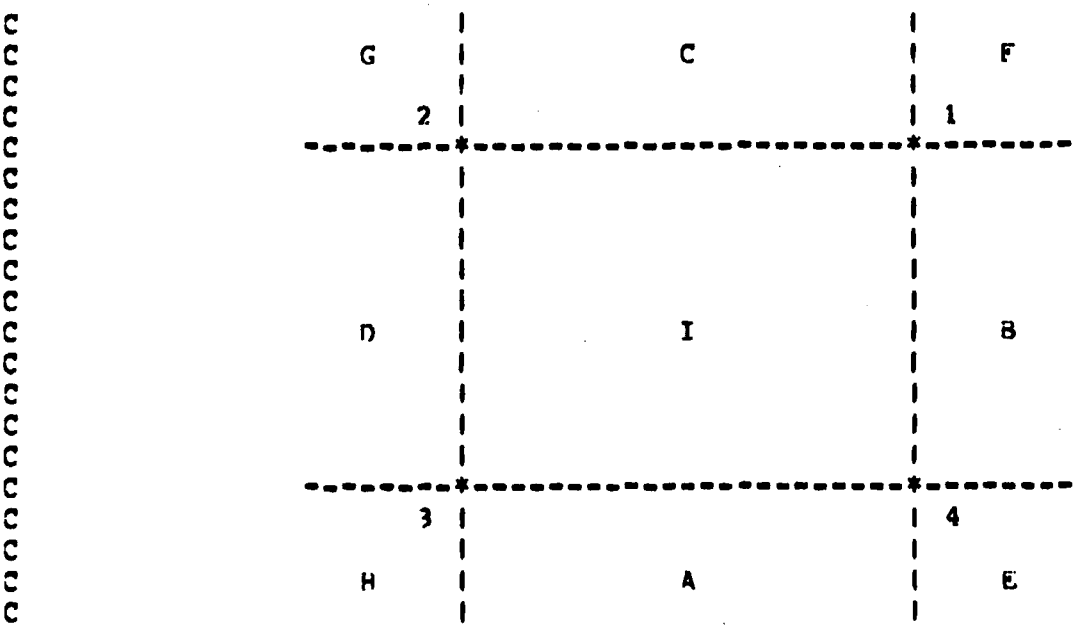
C COMPUTE SINES AND COSINES
      PITCH1=PITCH*3.141592654/180
      HEADING1=HEADING*3.141592654/180
      BANK1=BANK*3.141592654/180

      SH=SIN(HEADING1)
      SP=SIN(PITCH1)
      SB=SIN(BANK1)
      CH=COS(HEADING1)
      CP=COS(PITCH1)
      CB=COS(BANK1)

C COMPUTE THE ROTATION MATRIX
      ROT(1,1)=CB*CH-SH*SB*SP
      ROT(1,2)=-SH*CB-SB*SP*CH
      ROT(1,3)=-SB*CP
      ROT(2,1)=CP*SH
      ROT(2,2)=CP*CH
      ROT(2,3)=-SP
      ROT(3,1)=SB*CH+CB*SP*SH
      ROT(3,2)=-SH*SB+CB*SP*CH
      ROT(3,3)=CB*CP

C CALCULATE THE SCREEN CORNERS IN EYE CENTRIC
C WORLD COORDINATES COR(4,3)
      DO 600 I=1,3
      A1=IUS*ROT(1,I)
      A2=IWS*ROT(3,I)
      A3=IVS*ROT(2,I)
      A4=RLX*ROT(1,I)/2
      A5=RLY*ROT(3,I)/2
      X=A1+A2+A3
      Y1=A4+A5
      Y2=A4-A5
      COR(1,I)=X+Y1
      COR(2,I)=X-Y2
      COR(3,I)=X-Y1
```

```
        COR(4,I)=X+Y2
600     CONTINUE
C COMPUTE THE CROSS PRODUCTS OF THE SCREEN CORNERS CROSS(4)
        CROSS(1)=COR(4,1)*COR(1,2)-COR(4,2)*COR(1,1)
        DO 900 I=2,4
        K=I-1
        CROSS(I)=COR(K,1)*COR(I,2)-COR(K,2)*COR(I,1)
900     CONTINUE
C COMPUTE WHERE IS THE NADIR AND RENUMBER THE CORNERS.
```




```
C       FIGURE 1 NADIR LOCATION WITH RESPECT TO THE FOUR
C                DROPPED SCREEN CORNERS.  THERE ARE 9 CASES.
C                EACH CASE IS LABELED WITH A LETTER (A-I).

        KFLAG=9
        NADIR = 1                                       !CASE I
        KSL = 1
        NPIX = NX
        IAXSIS = 1
        IF(CROSS(1) .LT. 0) THEN                        !CASE B
                DO 1 I = 1,4
                DO 1 J = 1,3
1               CORUSF(I,J) = COR((JMOD(I,4) + 1),J)
                KSL = 2
                NPIX = 0
                IAXIS = 2
```

```
                NADIR = 0
                IF((CROSS(2) .EQ.0).OR.(CROSS(4).EQ.0)) GO TO 25
                IF(CROSS(4) .LT.0) THEN                 !CASE E
                        KFLAG=5
                        NADIR = -1
                        GO TO 25
                ENDIF
        KFLAG=2
        ENDIF
        IF(CROSS(2) .LT. 0) THEN                        !CASE C
                DO 2 I = 1,4
                DO 2 J = 1,3
2               CORUSE(I,J) = COR((JMOD((I+1),4) + 1),J)
                KSL = 3
                NPIX = 0
                IAXIS = 1
                NADIR = 0
                IF((CROSS(1) .EQ.0).OR.(CROSS(3).EQ.0)) GO TO 25
                IF(CROSS(1) .LT.0) THEN                 !CASE F
                        KFLAG=6
                        NADIR = -1
                        GO TO 25
                ENDIF
                KFLAG=3
        ENDIF
        IF(CROSS(3) .LT. 0) THEN                        !CASE D
                DO 3 I = 1,4
                DO 3 J = 1,3
3               CORUSE(I,J) = COR((JMOD((I+2),4) + 1),J)
                KSL = 4
                NPIX = NY
                IAXIS = 2
                NADIR = 0
                IF((CROSS(2) .EQ.0).OR.(CROSS(4).EQ.0)) GO TO 25
                IF(CROSS(2) .LT.0) THEN                 !CASE G
                        KFLAG=7
                        NADIR = -1
                        GO TO 25
                ENDIF
                KFLAG=4
        ENDIF
        IF(CROSS(4) .LT. 0) THEN                        !CASE A
                KSL = 1
                DO 4 I = 1,4
                DO 4 J = 1,3
4               CORUSE(I,J) = COR(I,J)
                NPIX = NX
                IAXIS = 1
                NADIR = 0
                IF((CROSS(3) .EQ.0).OR.(CROSS(1).EQ.0)) GO TO 25
                IF(CROSS(3) .LT.0) THEN                 !CASE H
```

```
                        KFLAG=8
                        NADIR = -1
                        GO TO 25
                    ENDIF
                    KFLAG=1
                ENDIF
                GO TO (11,12,13,14,15,16,17,18,19),KFLAG
11              TYPE*,'CASE = A'
                WRITE(10,*),'CASE = A'
                GOTO 20
12              TYPE*,'CASE = B'
                WRITE(10,*),'CASE = B'
                GOTO 20
13              TYPE*,'CASE = C'
                WRITE(10,*),'CASE = C'
                GOTO 20
14              TYPE*,'CASE = D'
                WRITE(10,*),'CASE = D'
                GOTO 20
15              TYPE*,'CASE = E'
                WRITE(10,*),'CASE = E'
                GOTO 20
16              TYPE*,'CASE = F'
                WRITE(10,*),'CASE = F'
                GOTO 20
17              TYPE*,'CASE = G'
                WRITE(10,*),'CASE = G'
                GOTO 20
18              TYPE*,'CASE = H'
                WRITE(10,*),'CASE = H'
                GOTO 20
19              TYPE*,'CASE = I'
                WRITE(10,*),'CASE = I'
20              CONTINUE
25              NEXTCOR = 1

C COMPUTE THE DISTANCE FORM THE NADIR TO THE CORNERS WHEN
C DROPPED TO THE FLAT EARTH CMAG(4)
                DO 800 I=1,4
                CMAG(I)=0
                DO 700 J=1,2
                X=CORUSE(I,J)
                CMAG(I)=CMAG(I) + X*X
700             CONTINUE
                CMAG(I)=SQRT(CMAG(I))
800             CONTINUE

C COMPUTE THE ANGLE TO EACH SCREEN CORNER FROM THE NADIR CANG(3)
                DO 1000 I=1,4
                CANG(I)=ATAN2(CORUSE(I,2),CORUSE(I,1))
1000            CONTINUE
```

```
      DO 1100 I=1,3
      CANG(I)=CANG(I)-CANG(4)
      IF(CANG(I).LT.0) CANG(I)=CANG(I)+2*3.141592654
      IF(CANG(I).LT.0) TYPE *,'ERROR IN CANG(I)=',CANG(I)
1100  CONTINUE

      IF(NADIR.EQ.0) CANG(4) = 2*3.14159
C COMPUTE ANGLE BETWEEN SUCCESSIVE SCAN LINES
C     ANG=ANGFACTOR*(RLX*RLX + RLY*RLY)/(3*CMAG(1)*CMAG(1)*NX)
      ANG=((CORUSE(1,1)-CORUSE(2,1))**2+(CORUSE(1,2)-
     *  CORUSE(2,2))**2)/(2*2*NX*NX*CMAG(1)*CMAG(1))
      ANG=SQRT(ANG)*ANGFACTOR

      DO 1300 I=1,3
      AA(1,I) = CORUSE(4,I)-CORUSE(1,I)
      AA(2,I) = CORUSE(1,I)-CORUSE(2,I)
      AA(3,I) = CORUSE(2,I)-CORUSE(3,I)
      AA(4,I) = CORUSE(3,I)-CORUSE(4,I)
1300  CONTINUE

C PROJ CONSTANTS
      DO 1400 I=1,3
      DO 1400 J=1,3
      IROT(J,I)=ROT(I,J)*IBITSCALE
1400  CONTINUE
      ICRASH=IVS*IBITSCALE
      ICONST=IVS*NPL
      JCONSTX=IUS*NPL
      JCONS1Y=IWS*NPL

C SL CONSTANTS
      SQRA=ANG**2
      IFIRST=1
      IFINISHED=0
      SUMANG=0

C LOAD CONSTANTS
      IXSO=0
      IYSO=0

C OUTPUT DATA

      WRITE(10,*),'IE(3)=',IE
      WRITE(10,*),'PITCH,BANK,HEADING=',PITCH,BANK,HEADING
      WRITE(10,*),'NADIR=',NADIR
      WRITE(10,*),'ANGLE BETWEENE SCANS=',ANG
      WRITE(10,*),'SCREEN CENTER LOCATION IN EYE COORDS',
     *  IUS,IVS,IWS
      WRITE(10,*),'SCREEN DIMENSIONS IN PIXELS (NX,NY) =',
     *  NX,NY
      WRITE(10,*),'SCREEN DIMENSIONS IN LENGTH (RLX,RLY) =',
```

APPENDIX F4

PZSLINIT.FOR

```
C       PZSLINIT.FOR
C
C PROGRAMED BY PHILIP GATT 896-4741
C THIS ROUTINE INITIALIZES ALL SCANLINE PARAMATERS
C THE SCREEN CORNERS ARE NUMBERED COUNTERCLOCKWISE 1-4 STARTING
C AT THE UPPER RIGHT HAND CORNER

        SUBROUTINE SLINIT(*)
        INCLUDE 'PZCOMDAT.FOR'
        COMMON /REAL/  ANG,SQRA,AA(4,3),CANG(4),CMAG(4),
     *  CORUSE(4,3)
        COMMON/SL/ NADIR,KSL,IFIRST,NEXTCOR
        COMMON/SLSTAR/NCRLIM,XLN2,IHGT,IXY2,JXY2,IDXY,DELTA
        COMMON/SCAN/INC,INC10,INC4,INC410,INCR,INCR10,
     *  INHORIZ(2),IEEE,NCRL
        COMMON/VIS/LOWN(3),IHTEGHT
        COMMON/PROJ1/NPIX,IAXIS
        DIMENSION HORIZ(2),RL(3)

        NCRL=NCRLIM
        IF (IFIRST .EQ.1) THEN
                IND = 4
                IN = 3
                IFIRST = 0
                IFLAG = 1
1               IF (NADIR.NE.1)THEN
C                       NADIR IS OUTSIDE THE FOOTPRINT
                        RL(1)=CORUSE(IND,1)
                        RL(2)=CORUSE(IND,2)
                        RL(3)=CORUSE(IND,3)
                RK1=ANG/(CORUSE(IND,1)*AA(IND,2)-CORUSE(IND,2)*
     *          AA(IND,1))
                ENDIF
                SCAL=HORIZONLIM/CMAG(IND)
                HORIZ(1)=CORUSE(IND,1)*SCAL
                HORIZ(2)=CORUSE(IND,2)*SCAL
        ELSE
                IF (NADIR .NE. 1) THEN
```

```
C                THEN THE NADIR IS OUTSIDE
         TL=RK1*(RL(1)**2+RL(2)**2)*(1+RK1*(AA(IND,1)*RL(1)+
     *   AA(IND,2)*RL(2)))
                 RL(1)=RL(1)+AA(IND,1)*TL
                 RL(2)=RL(2)+AA(IND,2)*TL
                 IF(ABS(AA(IND,1)).GT..25) THEN  !WHY .25 ?
                 RL(3)=CORUSE(IN,3)+(RL(1)-CORUSE(IN,1))*
     *           AA(IND,3)/AA(IND,1)
                 ELSE
                 RL(3)=CORUSE(IN,3)+(RL(2)-CORUSE(IN,2))*
     *           AA(IND,3)/AA(IND,2)
                 ENDIF
                 ENDIF

                 SUMANG=SUMANG+ANG

C        COMPUTE THE HORIZON BY A ROTATION MATRIX APPROXIMATION
                 X=HORIZ(1)*(1-SQRA/2) - HORIZ(2)*ANG
                 Y=HORIZ(1)*ANG +HORIZ(2)*(1-SQRA/2)
                 HORIZ(1)=X
                 HORIZ(2)=Y
         ENDIF
         IF((NADIR .EQ.-1).AND.(IFLAG .EQ.1)) THEN !CORNER CASE
                 IF(CANG(3) .LE. SUMANG) THEN
                         IFLAG = 0
                         TYPE*,'CROSSED CORNER   3 AT LINE=',LINE
                         TYPE*,'RESETING LOWER BOUND LINE'
                         IND = 3
                         IN = 2
                         GO TO 1
                 ENDIF
         ENDIF

C CORNER CROSS TEST
         IF(CANG(NEXTCOR) .LE. SUMANG) THEN
                 TYPE*,'CROSSED CORNER ',NEXTCOR,'AT LINE ',LINE
         TYPE*,'WITH LAST POINT, SCREEN AND WORLD COORDS=',
     *   IXBS,IYBS,IPP
                 IF (NEXTCOR .EQ. (3+NADIR)) THEN
                         TYPE*,'LAST LINE IS COMPLETED'
                         RETURN1
                 ENDIF
                 KSL=JMOD(KSL,4)+1
                 NEXTCOR=NEXTCOR+1
                 GOTO(100,200,300,400),KSL
100              IAXIS = 1
                 NPIX = NX
                 GO TO 500
200              IAXIS=2
                 NPIX=0
                 GOTO 500
```

```
                    AAVTRAEQUIPCEN 80-D-0014-2


300                 IAXIS=1
                    NPIX=0
                    GOTO 500
400                 IAXIS=2
                    NPIX=NY
500                 CONTINUE
            ENDIF

            IHORIZ(1)=HORIZ(1)
            IHORIZ(2)=HORIZ(2)
            IHORIZ(1)=IHORIZ(1)+IE(1)
            IHORIZ(2)=IHORIZ(2)+IE(2)
C           ICASE DEFINES THE AXIS OF FASTEST CHANGE ALONG THE SCAN
C           LINE, IOTHR DEFINES THE AXIS OF SLOWEST CHANGE ALONG
C           THE SCANLINE
            IF(IABS(IHORIZ(2)).GT.IABS(IHORIZ(1)))THEN
                    ICASE=2
                    IOTHR=1
            ELSE
                    ICASE=1
                    IOTHR=2
            ENDIF
            JXY2=ABS(IXY2*HORIZ(IOTHR)/HORIZ(ICASE))
            INC=ISIGN(1,IHORIZ(ICASE))
            INCIO=ISIGN(1,IHORIZ(IOTHR))
C  WE ATTEMPT TO APPROXIMATE THE STARTING POINT ON THE GROUND.
C  IN A MOTION PICTURE SEQUENCE, IHGT WOULD BE AVAILABLE FROM
C THE FIRST CALL TO THE DATA BASE WHICH IS PERFORMED IN PZSCENE
            IF((IHGT.LT.0) .AND.(NADIR.NE.1))THEN
                    IF(RL(3).GE.0)THEN
                            IP(1)=IHORIZ(1)
                            IP(2)=IHORIZ(2)
                    ELSE
                            IP(ICASE)=ABS(IHGT*RL(ICASE)/RL(3))*INC
                            IP(IOTHR)=ABS(IP(ICASE))*JXY2/IXY2*INCIO
                    ENDIF
            ELSE
                    IP(ICASE)=0
                    IP(IOTHR)=0
            ENDIF
            X=1.*(IP(ICASE)**2+IHGT**2)/NCRDI4**2
            IF(X.LT.4)THEN
                    ICL=0
                    NCR=1
            ELSE
                    ICL=ALOG(X)*XLN2/2 -1
                    NCR=2**ICL
            ENDIF

            IPP(1)=IP(1)+IE(1)
            IPP(2)=IP(2)+IE(2)
```

```
C CALL THE DATA BASE TO INITIALIZE LASTIP IN SCAN
      GOTO (1001,1005),IDATBAS
1001  CALL ZRDATA(1010)
1005  CALL COLORDATA(1010)
1010  CONTINUE

      IP(3)=IP(3)-IP(3)
C  SET THE NORMALIZED COORDINATES IN NCR UNITS WITH ORIGIN AT
C  THE EYE. BUT SUCH THAT IP(*) HAS ABOUT 12 BITS.

      DO  600 I=1,3
600           IP(I)=(IP(I) + NCR/2)*4/NCR
      IHIEGHT = IP(3)
              INCR=NCR*INC
              INCRIO=NCR*INCIO
              INC4=4*INC
              INC4IO=4*INCIO
      IEEE=-IOXY+(IP(ICASE)*JXY2*INC-IP(IOTHR)*IXY2*INCIO)/4
              IF(IEEE.GT.IXY2) THEN
                      IPP(IOTHR)=IPP(IOTHR)+INCRIO
                      IP(IOTHR)=IP(IOTHR)+INC4IO
                      IEEE=IEEE-IXY2
              TYPE*,'IEEE.GT. IXY2  AT VIS 7600 DUE TO NCR/2
     *        ROUNDOFF'
              ENDIF

              IF(IEEE.LT.-IXY2) THEN
              TYPE*,'IEEE.LT.-IXY2 AT VIS LINE 8100 DUE TO
     *        NCR/2 ROUNDOFF'
                      IEEE=IEEE+IXY2
                      IPP(IOTHR)=IPP(IOTHR)-INCRIO
                      IP(IOTHR)=IP(IOTHR)-INC4IO
              ENDIF

      IF(NADIR.NE.1) THEN
              LOWN(ICASE)=IXY2*INC
              LOWN(IOTHR)=JXY2*INCIO
              LOWN(3)=RL(3)*LOWN(ICASE)/RL(ICASE)
      ELSE
              LOWN(1)=INCR
              LOWN(2)=INCRIO
              LOWN(3)=IP(3)
      ENDIF

C     ALPHAV IS USED TO ATTENUATE ROOF DATA IN ZRDATA1
      ALPHAV=-4.5/IHORIZ(ICASE)

      RETURN
      END
```

APPENDIX GA

OVERHAUSER-COONS ROUTINES

Appendices GB through GC contain the routines that can be used for data compression using the Overhauser-Coons Bicubic Patch Algorithm.

APPENDIX 3H

LCOVRCNS.FOR

```
C       LCOVRCNS.FOR
C
C       OVERHAUSER-COONS BICUBIC PATCH ROUTINE
C               PROGRAMMED BY:  GERALD BECKER AND LINDA COULTER
C               DATE:           6 FEB 1981
C               THIS IS THE OVERHAUSER-COONS BICUBIC PATCH
C       ROUTINE.  IT USED 12 DATA POINTS FROM A DATA ARRAY
C       (SAMPT) TO CALCULATE THE VALUE OF THE HEIGHT FOR THE
C       'REALSCAN' TEST DATA BASE.  REFERENCES FOR THIS ROUTINE
C       ARE:
C            1. 'VISUAL INTERACTION WITH OVERHAUSER CURVES AND
C               SURFACES', J.A. BREWER AND D.C. ANDERSON,PURDUE
C               UNIVERSITY, REFERENCED FROM COMPUTER GRAPHICS,
C               VOL. 11, 2, SUMMER '77.
C            2. COMMUNICATIONS BETWEEN DR. R.W. PATZ AND G.L.
C               BECKER ON 31 JUL 80.
C       THIS VERSION OF THE O-C ROUTINE ACCEPTS THE VALUE FOR
C       PICWIT.  THE ROUTINE CAN BE TESTED IN ONE OF TWO WAYS BY
C       PLACING A 0 AS THE FIRST CHARACTER OF A LINE FOR THE
C       TEST WHICH IS NOT DESIRED.

C               TEST 1:
C               THE INTERPOLATION OF THE SURFACE IS COMPLETED 40
C               TIMES VARYING THE SAMPLE SPACING AND THE DATA
C               POINT DENSITY EACH TIME.  SAMPSPAC VARIES FROM
C               APPROXIMATELY 50 TO 600.  ISAMPSP/512 VARIES
C               FROM .0957 TO 1.0966.

C               TEST 2:
C               THE INTERPOLATION OF THE SURFACE IS COMPLETED 30
C               TIMES VARYING THE SAMPLE SPACING AND THE DATA
C               POINT DENSITY EACH TIME.  SAMPSPAC VARIES FROM
C               APPROXIMATELY 10 TO 600.  ISAMPSP/512 VARIES
C               FROM .0195 TO 1.0215.

C               OPTION IS DETERMINED SO THAT THE
C               NUMBER OF POINTS INTERPOLATED PER PICTURE WILL
C               BE CONSTANT.
```

```
C       THE FOLLOWING FIGURE SHOWS THE ARRANGEMENT OF THE 'P
C       VALUES' AS THEY ARE USED IN THIS ROUTINE.
C
C                (IX,IY+2)@                  @(IX+1,IY+2)
C                         |                  |
C                         |                  |
C                         |       C(2,2)     |
C                (IX,IY+1)|        C(3)      |(IX+1,IY+1)
C (IX-1,IY+1)@------------@------------------@----------@(IX+2,IY+1)
C                         |Z3              Z4|
C                    C(1,1)|                  |C(1,2)
C                     C(2) |                  | C(4)
C                         |                  |
C                   (IX,IY)|Z1              Z2|(IX+1,IY)
C   (IX-1,IY)@------------@------------------@----------@(IX+2,IY)
C                         |       C(2,1)     |
C                         |        C(1)      |
C                         |                  |
C                         |                  |
C                (IX,IY-1)@                  @(IX+1,IY-1)
C
C       THE FOLLOWING FIGURE IDENTIFIES THE VARIABLES USED
C       TO DESCRIBE THE SURFACE:
C
C   |<-----------------PICWIT------------------>|
C
C   @============@==========@==========@==========@
C   |      :     |     :    |     :    |     :    |
C   |      :     |     :    |     :    |     :    |
C   |------------|----------|----------|----------|
C   |      :     |     :    |     :    |     :    |
C   |      :  (3)|     :    |     :  (5)|     :    |
C   @============@=====*====@=====*====@==========@
C   |      :     |     :    |     :    |     :    |
C   |      :     |     :    |     :    |     :    |
C   |------------*-----*----*-----*----*----------|
C   |      :     |     :    |     :    |     :    |
C   |      :  (2)|     :    |     :    |     :    |
C   @============@=====*====@=====*====@==========@
C   |      :     |     :    |     :    |     :    |
C   |      :     |     :    |     :    |     :    |
C   |------------*-----*----*-----*----*----------|
C   |      :     |     :    |     :    |     :    |
C   |      :  (1)|  (4):    |     :    |     :    |
C   @============@=====*====@=====*====@==========@
C   |      :     |     :    |     :    |     :    |
C   |      :     |     :    |     :    |     :    |  Y
C   |------------|----------|----------|----------|  ^
C   |      :     |     :    |     :    |     :    |  |
C   |      :     |     :    |     :    |     :    |  |
C   @============@==========@==========@==========@  ---->X
```

```
C
C-->|            |<--SAMPSPAC                -->|      |<--DELTA
C
C          IPP                          IUP
C    !-----:-----!-----:-----!-----:-----!-----:-----!--->DELTA
C    1           2           3          ...         IEND
C
C    @DATA ARRAY, SAMPT(IX,IY)
C    *INTERPOLATED VALUES, FEND
C
C        THE PROGRAM FLOW STARTS AT (1) WHERE THE BOUNDARY LINES
C        ARE CALCULATED FOR THE PATCH, THEN FOLLOWS AN INCREMENT
C        ON Y UNTIL REACHING (2) THUS COMPLETING ONE 'SEGMENT.'
C        THE BOUNDARY LINES FOR THE PATCH ARE RECALCULATED AND
C        THE Y-INCREMENT CONTINUES UNTIL (3), COMPLETING ONE
C        'LINE.'  THE X VALUE IS THEN INCREMENTED TO (4) SO
C        BOUNDARY LINES ARE RECALCULATED.  Y- AND X-INCREMENTS
C        CONTINUE UNTIL (5) COMPLETING THE INTERPOLATION REGION
C        OR ONE 'PICTURE.'
C
C
C        DOUBLE PRECISION IS USED ON ALL VALUES USED FOR
C        STATISTICS.
         DOUBLE PRECISION SGSMDF,SGSMDF2,SGSMA,SMLND,SMLND2,
     -   SMLNA,SMDIF,SMDIF2,SMABS,IDIAVG,RMS,DAVG,LNAVG,LNRMS,
     -   LNDABS,SGAVG,SGRMS,SGABS,DIF,DIF2,DABS,DIFMXPS,DIFMXNG
C        EQUIVALENCE IS USED FOR EASE OF COMPUTATION IN SEVERAL
C        DO-LOOPS.
         EQUIVALENCE (B(1),B0Y),(B(2),B0X),(B(3),B1Y),(B(4),B1X),
     -   (T(1),TX),(T(2),TY),(T(3),TXX),(T(4),TYY)
         DIMENSION T(4),B(4),C(4)
         COMMON ANS(4),IX,IY,SAMPT(250,250),V(4,4),Z1,Z2,Z3,Z4,
     -   IPCN
C
C        NOTE:  FOR SAMPLE ARRAY, SAMPT(A,B), THE DIMENSIONS A
C                AND B MUST BE GREATER THAN THE PICTURE WIDTH
C                DIVIDED BY THE SAMPLE SPACING OR
C
C                       A,B > PICWIT/ISAMPSP.
C
C                IF SAMPT(A,B) IS CHANGED, REMEMBER TO CHANGE IT
C                IN THE SUBROUTINES.
C
C*************************************************************
C        FUNCTION USED IN COMPARISON
         FT(A,B)=127*(SIN(6.283185*(A)/1024))*(SIN(6.283185*
     -   (B)/1024))
C*************************************************************
C
         TYPE*,'ENTER THE PICTURE WIDTH'
         ACCEPT*,PICWIT
```

```
C
      TYPE*,'ENTER THE DESIRED NUMBER OF POINTS TO BE
     - INTERPOLATED'
      ACCEPT*,POINTS
C
      WRITE(74,*)'INTERPOLATION BY OVERHAUSER-COONS FUNCTION'
      WRITE(74,*)
      WRITE(74,*)'PICWIT=',PICWIT
      WRITE(74,*)'APPROXIMATE NUMBER OF POINTS INTERPOLATED=',
     - POINTS
      WRITE(74,*)'    NOTE:  MAX. VALUE OF TEST FUNCTION=127'
      WRITE(74,*)'**************************'
      WRITE(74,*)' '
      WRITE(74,200)
200   FORMAT(3X,'ISAMPSP',3X,'DPTDEN',
     - 4X,'DIFMXPS',4X,'DIFMXNG',6X,'RMS/127',
     - 7X,'DAVG')
      WRITE(74,*)' '

      WRITE(75,*)'INTERPOLATION BY OVERHAUSER-COONS FUCNTION'
      WRITE(75,*)
      WRITE(75,*)'PICWIT=',PICWIT
      WRITE(75,*)'APPROXIMATE NUMBER OF POINTS INTERPOLATED=',
     - POINTS
      WRITE(75,*)'**************************'
      WRITE(75,*)' '
      WRITE(75,500)
500   FORMAT(5X,'SAMPSPAC',3X,'ISAMPSP',7X,'DPTDEN',8X,
     - 'COUNT',7X,'SAMPLE')
      WRITE(75,520)
520   FORMAT(57X,'ARRAY')
      WRITE(75,*)' '

      WRITE(75,*)'INTERPOLATION BY OVERHAUSER-COONS FUNCTION'
      WRITE(75,*)
      WRITE(76,*)'PICWIT=',PICWIT
      WRITE(76,*)'APPROXIMATE NUMBER OF POINTS INTERPOLATED=',
     - POINTS
      WRITE(76,*)'    NOTE:  MAX. VALUE OF TEST FUNCTION=127'
      WRITE(76,*)'           ONE HALF THE PERIOD OF TEST
     - FUNCTION=512'
      WRITE(76,*)'**************************'
      WRITE(76,*)' '
      WRITE(76,600)
600   FORMAT(3X,'ISAMPSP/512',6X,'RMS/127')
      WRITE(76,*)' '
C
C     THE DISTANCE BETWEEN THE SAMPLE DATA POINTS IS THE
C     SAMPLE SPACING.
D     SAMPSPAC=50/1.064692     !TEST 1:  1.0640929=12**1/40
      SAMPSPAC=10/1.1462298    !TEST 2:  1.1462298=60**1/30
```

```
C
D       DO 210 K=1,40              !TEST 1
        DO 210 K=1,30              !TEST 2
                TYPE*,'K=',K
C
D       SAMPSPAC=SAMPSPAC*1.0640929      !TEST 1
        SAMPSPAC=SAMPSPAC*1.1462298      !TEST 2
        ISAMPSP=SAMPSPAC
                !MUST BE INTEGER TO MAINTAIN A GRID PATTERN
        DPAC=1./ISAMPSP
C
C       DPTDEN IS THE NUMBER OF DIVISIONS MADE BY THE TEST
C               POINTS PER SAMPLING INTERVAL.
        SQRPTS=SQRT(POINTS)        !CALCULATION OF DPTDEN ENSURES A
        IPICSMP=PICNIT*DPAC        !CONSTANT NUMBER OF INTERPOLATED
        IF(IPICSMP.LE.2) GOTO 400       !REQUIRED DATA BASE NOT
C                                        AVAILABLE
        DENOM=IPICSMP-2            !    POINTS FOR EACH STEP
        DPTDEN=SQRPTS/DENOM
                IF(DPTDEN.LT.1)THEN
                        TYPE*, 'ROUTINE IS INVALID FOR DPTDEN<1'
                        GO TO 210
                ENDIF
C
        XSAMPSP=FLOAT(ISAMPSP)
C
C       ROUTINE TO CREATE THE SAMPLED DATA ARRAY
        IEND=(PICNIT*DPAC)+1
        ICNTSMP=IEND*IEND-4   !REQUIRED SIZE OF SAMPLE DATA BASE

        DO 10 I=1,IEND
                XMIS=(I-1)*ISAMPSP
                DO 10 J=1,IEND
                        SAMPT(I,J)=FT(XMIS,(J-1)*XSAMPSP)
 10     CONTINUE
D       WRITE (74,100) ((SAMPT(I,J),J=1,IEND),I=1,IEND)
 100    FORMAT (' ',15F8.3)
C*****************************************************************
C       OVERHAUSER-COONS FUNCTIONS BEGIN
C
C
        DELTA=ISAMPSP/DPTDEN
C
C       IPP AND IUP ARE IN UNIT  OF DELTA.
C       IPP AND IUP ARE THE LIMITS OF THE X- AND Y-INCREMENTS
C       SUCH THAT (IPP+1)*DELTA AND (IUP+1)*DELTA ARE THE FIRST
C       AND LAST WORLD COORDINATES INTERPOLATED, RESPECTIVELY.
C
        IUP=(IEND-2)*DPTDEN-1
        IPP=DPTDEN-1
C
```

```
C       CHECK FOR CROSSING BOUNDARY DUE TO ROUND-OFF ERROR IN
C       DELTA
        IF(ISAMPSP.GT.((IPP+1)*DELTA))IPP=IPP+1
        IF(PICWIT-ISAMPSP.LE.(IUP+1)*DELTA)IUP=IUP-1
C
C       (X,Y):=WORLD COORDINATES
        XX=IPP*DELTA
        X=XX            !INITIALIZE X VARIABLE FOR X-INCREMENT
C
C       (IRLX,IRLY):=THE LOWER LEFT BOUNDARY IN WORLD
C       COORDINATES COORESPONDING TO THE FIRST POINT INSIDE
C       THE PATCH.
        IRLX=ISAMPSP
C
C       (IRUX,IRUY):=THE UPPER RIGHT BOUNDARY IN WORLD
C       COORDINATES CORRESPONDING TO THE POTENTIAL LAST POINT
C       IN THIS PATCH OR THE FIRST POINT IN A NEIGHBORING PATCH.
        IRUX=IRLX+ISAMPSP
C
C       (IX,IY):=THE INDEX OF THE SAMPLED DATA ARRAY
        IX=2            !INITIALIZE X INDEX
C
C*************************************************************
C       INITIALIZE VARIABLES FOR PICTURE STATISTICS
C
        ICNT=0          !COUNTER:  POINTS PER PICTURE
        SMDIF=0         !SUM OF DIFF FOR PICTURE
        SMDIF2=0        !SUM OF DIFF SQUARED FOR PICTURE
        SMABS=0         !SUM OF ABS VALUE OF DIFF FOR PICTURE
        DIFMXPS=0       !MAXIMUM POSITIVE ERROR
        DIFMXNG=0       !MAXIMUM NEGATIVE ERROR
C*************************************************************
C
        DO 90 I=IPP,IUP         !START OF X-INCREMENT, ONE LINE
C                                PER LOOP
                Y=XX            !INITIALIZE Y VARABLE FOR
C                                Y-INCREMENT
                IY=2            !INITIALIZE Y INDEX
                IRLY=ISAMPSP
                IRUY=IRLY+ISAMPSP
                X=X+DELTA
C
C       CHECK FOR CROSSING BOUNDARY
                IF (X.GT.IRUX) THEN     !NEXT LINE
                        IX=IX+1         !INDEX FOR NEXT LINE
                        IRLX=IRUX
                        IRUX=IRUX+ISAMPSP
                ENDIF
C
C       TX:=THE INCREMENTAL DISTANCE, X, WITHIN THE PATCH.
                TX=(X-IRLX)*DPAC
```

```
                TXX=TX          !TO ALLOW I(3) EQUIVALENCE
                TX2=TX*TX
                B1X=(3.0-2.0*TX)*TX2
                B0X=1.0-B1X
        CALL PTCHNG
C
C       CALCULATION OF X-DEPENDENT BOUNDARY LINES:  C(1), C(3)
C               NOTE:  TX3=TX**3
C            V(4,4) - COEFFICIENTS OF "C" EQUATIONS FOR THE
C                     BOUNDARY LINES
C               C(1)=V(1,1)*TX3+V(1,2)*TX2+V(1,3)*TX+V(1,4)
C               C(3)=V(3,1)*TX3+V(3,2)*TX2+V(3,3)*TX+V(3,4)
C
                DO 150 IOX=1,3,2
                        CCX=V(IOX,1)
                        DO 140 IEX=2,4
                                CCX=CCX*T(IOX)+V(IOX,IEX)
 140                    CONTINUE
                        C(IOX)=CCX
 150            CONTINUE
C
C       SUMMATION OF X-DEPENDENT C VALUES AND PARTIAL Z TERMS
                CZ1=C(1)-Z1*B0X-Z2*B1X
                CZ2=C(3)-Z3*B0X-Z4*B1X
C
C       INITIALIZE VARAIBLES FOR LINE AND SEGMENT STATISTICS
                ICNT1=0         !COUNTER:  POINTS PER SEGMENT
                ICNTA=0         !COUNTER:  POINTS PER LINE
                SMLND=0.0       !SUM OF LINE DIFFERENCES
                SMLND2=0.0      !SUM OF LINE DIFFERENCES SQUARED
                SMLNA=0.0       !SUM OF ABSOLUTE VALUE OF LINE DIFF
                SGSMDF=0.0      !SUM OF SEGMENT DIFFERENCES
                SGSMDF2=0.0     !SUM OF SEGMENT DIFFERENCES SQUARED
                SGSMA=0.0       !SUM OF ABSOLUTE VALUE OF SEG DIFF
C
C
                DO 80 J=IPP,IUP         !Y-INCREMENT, ONE SEGMENT
C                                           PER LOOP
                        Y=Y+DELTA
C
C               CHECK FOR CROSSING BOUNDARY
                        IF (Y.GT.IRUY) THEN             !NEXT SEGMENT
                                IY=IY+1  !INDEX FOR NEXT SEGMENT
                                IRLY=IRUY
                                IRUY=IRUY+ISAMPSP
                                CALL PTCHNG
C
                DO 151 IOX=1,3,2 !C(1) AND C(3) FOR NEXT SEGMENT
                        CCX=V(IOX,1)
                        DO 141 IEX=2,4
                                CCX=CCX*T(IOX)+V(IOX,IEX)
```

```
 141                    CONTINUE
C
                        C(IDX)=CCX
 151            CONTINUE
                CZ1=C(1)-Z1*B0X-Z2*B1X
                CZ2=C(3)-Z3*B0X-Z4*B1X
C
C       STATISTICS ARE DETERMINED PER LINE SEGMENT OF LENGTH
C       SAMPSPAC.
C
        ICNTA=ICNTA+ICNT1
        SMLND=SMLND+SGSMDF
        SMLND2=SMLND2+SGSMDF2
        SMLNA=SMLNA+SGSMA
D       SGAVG=SGSMDF/ICNT1       !AVERAGE DIFF. IN SEGMENT
D       SGRMS=SQRT(SGSMDF2/ICNT1-SGAVG*SGAVG) !SEGMENT RMS
D       SGABS=SGSMA/ICNT1        !AVERAGE ABSOLUTE DIFF OF SEGMENT
C
        ICNT1=0          !INITIALIZE FOR NEXT SEGMENT
        SGSMDF=0.0       !CLEAR THE SUM OF SEGMENT DIFFERENCE
        SGSMDF2=0.0      !CLEAR THE SUM OF SEGMENT DIFF SQUARED
        SGSMA=0.0        !CLEAR THE SUM OF SEGMENT ABS DIFFENCES
C
                         ENDIF
C
C
C       TY:=THE INCREMENTAL DISTANCE, Y, WITHIN THE PATCH
                         TY=(Y-IRLY)*DPAC
                         TYY=TY   !TO ALLOW FOR T(4) EQUIVALENCE
                         TY2=TY*TY
                         B1Y=(3.0-2.0*TY)*TY2
                         B0Y=1.0-B1Y
C
C       SUMMATION OF X-DEPENDENT C VALUES AND COMPLETE Z TERMS
C               CZ=C(1)*B0Y-Z1*B0X*B0Y-Z2*B1X*B0Y
C                 +C(3)*B1Y-Z3*B0X*B1Y-Z4*B1X*B1Y
C
                         CZ=CZ1*B0Y+CZ2*B1Y
                         CSTUF=0
C
C       CALCULATION OF Y-DEPENDENT BOUNDARY LINES:  C(2), C(4)
C               NOTE:  TY3=TY**3
C               C(2)=V(2,1)*TY3+V(2,2)*TY2+V(2,3)*TY+V(2,4)
C               C(4)=V(4,1)*TY3+V(4,2)*TY2+V(4,3)*TY+V(4,4)
C
                         DO 70 IEY=2,4,2
                                 CCY=V(IEY,1)
                                 DO 60 IDY=2,4
                                    CCY=CCY*T(IEY)+V(IEY,IDY)
60                               CONTINUE
                         CSTUF=CSTUF+CCY*B(IEY) !CSTUFF=C(2)*B0X
```

```
C                                                    +C(4)B1X
C
 70                     CONTINUE
C*****************************************************************
        FEND=CSTUF+CZ        !INTERPOLATED OVERHAUSER-COONS VALUE
C
C               FEND=Z(X,Y)=C(1)*B0Y-Z1*B0X*B0Y
C                          +C(2)*B0X-Z2*B1X*B0Y
C                          +C(3)*B1Y-Z3*B0X*B1Y
C                          +C(4)*B1X-Z4*B1X*B1Y
C
C*****************************************************************
C       THE CORRESPONDING EXACT VALUE IS:
                        EXACT=FT(X,Y)
C
C*****************************************************************
C       DATA DETERMINED IN PREPARATION FOR SEGMENT AND LINE
C       STATISTICS.
                        DIF=EXACT-FEND
                        DIF2=DIF*DIF     !SQUARE DIFF.
                        DABS=ABS(DIF)    !ABSOLUTE VALUE OF DIFF.
                        IF(DIF.GE.0) THEN
                                DIFMXPS=DMAX1(DIFMXPS,DIF)
                        ELSE
                                DIFMXNG=DMAX1(DIFMXNG,DABS)
                        ENDIF
C*****************************************************************
C       SUMMATION OF SEGMENT DATA FOR THE CALCULATION OF SEGMENT
C       STATISTICS.
        SGSMDF=SGSMDF+DIF        !COMPUTE THE SUM OF SEGMENT
C                                 DIFFERENCES
        SGSMDF2=SGSMDF2+DIF2     !COMPUTE THE SUM OF SEG DIFF.
C                                 SQUARED
        SGSMA=SGSMA+DABS         !COMPUTE THE SUM OF SEG DIFF
C                                 ABSOLUTE VALUE
        ICNT1=ICNT1+1            !INCREMENT SEGMENT COUNTER
C
        IF ((J.EQ.IUP).AND.(Y.LE.IRUY))THEN
                !SEGMENT STATS FOR LAST SEGMENT
        SMLND=SMLND+SGSMDF
        SMLND2=SMLND2+SGSMDF2
        SMLNA=SMLNA+SGSMA
D       SGAVG=SGSMDF/ICNT1       !AVERAGE DIFF IN SEGMENT
D       SGRMS=SQRT(SGSMDF2/ICNT1-SGAVG*SGAVG)     !SEGMENT RMS
D       SGABS=SGSMA/ICNT1        !AVERAGE ABSOLUTE VALUE OF DIFF
C                                 IN SEGMENT
        ICNTA=ICNTA+ICNT1
        ENDIF
 80     CONTINUE
C
C*****************************************************************
```

```
C       SUMMATION OF LINE DATA FOR THE CALCULATION OF PICTURE
C       STATISTICS.
        SMDIF=SMDIF+SMLND
        SMDIF2=SMDIF2+SMLND2
        SMABS=SMABS+SMLNA
        ICNT=ICNT+ICNTA
C*************************************************************
C       LINE STATISTICS
D       LNAVG=SMLND/ICNTA       !COMPUTE AVERAGE LINE DIFFERENCE
D       LNRMS=SQRT(SMLND2/ICNTA-LNAVG*LNAVG)  !COMPUTE LINE RMS
D       LNDABS=SMLNA/ICNTA   !COMPUTE AVERAGE ABSOLUTE DIFF/LINE
C*************************************************************
 90     CONTINUE                              !END X-INCREMENT LOOP
C*************************************************************
C       PICTURE STATISTIC
C
        TOTAVG=SMDIF/ICNT   !COMPUTE AVERAGE ERROR FOR PICTURE
        RMS=SQRT(SMDIF2/ICNT-TOTAVG*TOTAVG) !COMPUTE RMS PICTURE
        DAVG=SMABS/ICNT          !COMPUTE AVERAGE ABSOLUTE ERROR
C                                 FOR PICTURE

        XSMP512=ISAMPSP/512.
        RMS127=RMS/127

        WRITE(74,220)ISAMPSP,DPTDEN,
     -  DIFMXPS,DIFMXNG,RMS127,DAVG
220     FORMAT(I8,F11.4,2(F11.4),F13.7,F11.4)
        IF((DIFMXPS.LT.1E-4).OR.(DIFMXNG.LT.1E-4))THEN
                WRITE(74,*)'           DIFMXPS=',DIFMXPS,
     -          'DIFMXNG=',DIFMXNG
        ENDIF

        WRITE(75,510)SAMPSPAC,ISAMPSP,DPIDEN,ICNT,ICNTSMP
510     FORMAT(F13.4,I10,F13.4,2(I13))

        WRITE(76,630)XSMP512,RMS127
630     FORMAT(F14.4,F13.7)
 210    CONTINUE
C*************************************************************
 400    STOP
        END
        SUBROUTINE PTCHNG
C
C
C       THIS SUBROUTINE CALLS FOR THE GENERATION OF THE
C       COEFFICIENTS OF THE BOUNDARY LINES (C EQUATIONS).
C
        COMMON ANS(4),IX,IY,SAMPT(250,250),V(4,4),Z1,Z2,Z3,Z4,
     -  IPCN
C
C
```

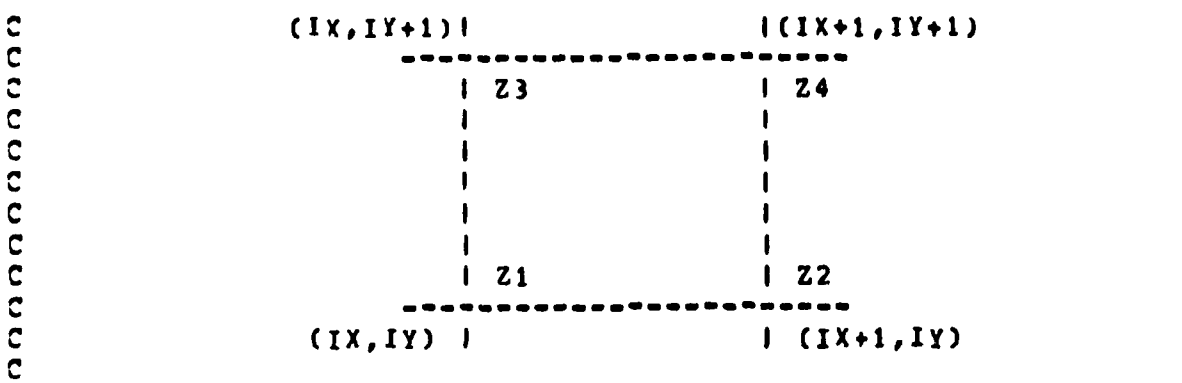
```
      IX1=IX+1
      IY1=IY+1
      Z1=SAMPT(IX,IY)     !LOWER LEFT CORNER OF PATCH
      Z2=SAMPT(IX1,IY)    !LOWER RIGHT CORNER OF PATCH
      Z3=SAMPT(IX,IY1)    !UPPER LEFT CORNER OF PATCH
      Z4=SAMPT(IX1,IY1)   !UPPER RIGHT CORNER OF PATCH
C
      CALL C(2,1)         !LOWER HORIZONTAL LINE, C(1)
      DO 20 J2=1,4
            V(1,J2)=ANS(J2)
 20   CONTINUE
C
      CALL C(1,1)         !LEFT VERTICAL LINE, C(2)
      DO 30 J2=1,4
            V(2,J2)=ANS(J2)
 30   CONTINUE
C
      CALL C(2,2)         !UPPER HORIZONTAL LINE, C(3)
      DO 40 J2=1,4
            V(3,J2)=ANS(J2)
 40   CONTINUE
      CALL C(1,2)         !RIGHT VERTICAL LINE, C(4)
C
      DO 50 J2=1,4
            V(4,J2)=ANS(J2)
 50   CONTINUE
      RETURN
      END
C
      SUBROUTINE C(IC,JC)
C
C     THIS SUBROUNTINE PERFORMS THE MATRIX MULTIPLICATION
C     REQUIRED TO GENERATE THE COEFFICIENTS OF THE C EQUATIONS
C     (BOUNDARY LINES).
C
      COMMON ANS(4),IX,IY,SAMPT(250,250),V(4,4),Z1,Z2,Z3,Z4,
     - IPCN
      REAL MAT(4,4)
C
C     GIVEN A PATCH WITH LOWER LEFT INDICES (IX,IY),
C     CALCULATION OF THE COEFFICIENTS OF THE C EQUATIONS
C     INVOLVES MULTIPLICATION OF THE DATA MATRIX BY THE
C     4-BY-1 MATRIX,
C           -FOR C(1), SAMPT(IX-1,IY), SAMPT(IX,IY),
C                      SAMPT(IX+1,IY), SAMPT(IX+2,IY)
C           -FOR C(2), SAMPT(IX,IY-1), SAMPT(IX,IY),
C                      SAMPT(IX,IY+1), SAMPT(IX,IY+2)
C           -FOR C(3), SAMPT(IX-1,IY+1), SAMPT(IX,IY+1),
C                      SAMPT(IX+1,IY+1), SAMPT(IX+2,IY+1)
C           -FOR C(4), SAMPT(IX+1,IY-1), SAMPT(IX+1,IY),
C                      SAMPT(IX+1,IY+1), SAMPT(IX+1,IY+2)
```

```
C
      DATA MAT(1,1),MAT(2,4),MAT(3,1)/3*-0.5/,MAT(1,2)/1.5/,
     -  MAT(1,3)/-1.5/,MAT(1,4),MAT(3,3)/2*0.5/,MAT(2,1),
     -  MAT(4,2)/2*1.0/,MAT(2,2)/-2.5/,MAT(2,3)/2.0/,MAT(3,2),
     -  MAT(3,4),MAT(4,1),MAT(4,3),MAT(4,4)/5*0.0/
C
      IF(IC.EQ.1)THEN  !Y-DEPENDENT BOUNDARY LINES, C(2), C(4)
            IP=IX+JC-1
            K=2-IY
            DO 10 I=1,4
                  ANS(I)=0
                  DO 10 J=IY-1,IY+2
                     ANS(I)=ANS(I)+MAT(I,J+K)*SAMPT(IP,J)
10    CONTINUE
      ELSE              !X-DEPENDENT BOUNDARY LINES, C(1), C(3)
            IP=IY+JC-1
            K=2-IX
            DO 20 I=1,4
                  ANS(I)=0
                  DO 20 J=IX-1,IX+2
                    ANS(I)=ANS(I)+MAT(I,J+K)*SAMPI(J,IP)
20    CONTINUE
      ENDIF
      RETURN
      END
```

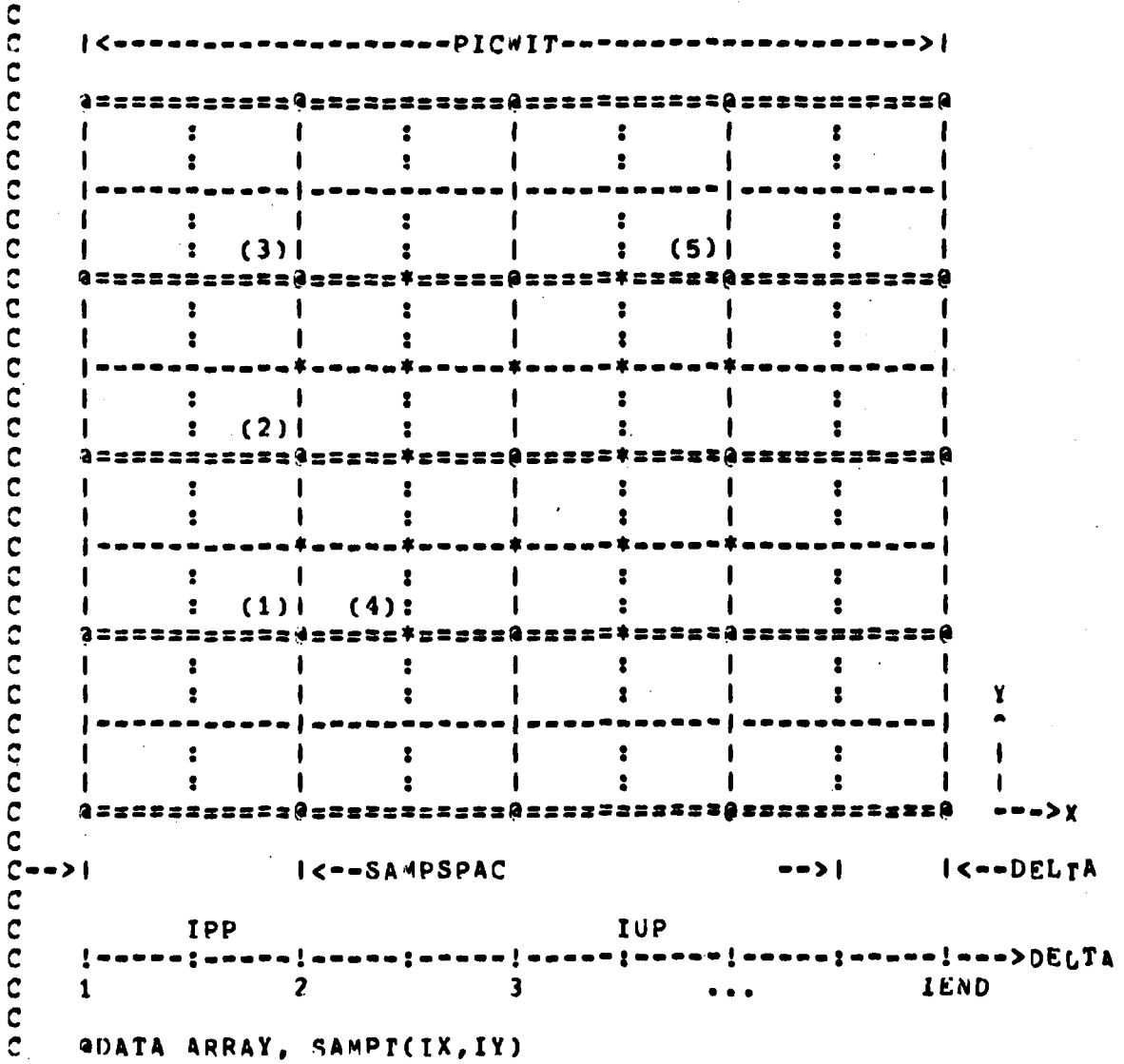
APPENDIX GC

LCSTRAIT.FOR

```
C       LCSTRAIT.FOR
C
C       ROUTINE FOR STRAIGHT-LINE APPROXIMATION OF A PATCH
C               PROGRAMMED BY:  LINDA COULTER
C               DATE:           14 MAY 1981
C
C               THIS ROUTINE FOR STRAIGHT LINE APPROXIMATION OF
C       A PATCH USES 4 DATA POINTS FROM A DATA ARRAY (SAMPT) TO
C       CALCULATE THE VALUE OF THE HEIGHT FOR THE 'REALSCAN'
C       TEST DATA BASE.
C
C       THIS VERSION OF THE STRAIGHT-LINE APPROXIMATION ROUTINE
C       ACCEPTS THE VALUE FOR PICWIT.
C               THE ROUTINE CAN BE TESTED IN ONE OF TWO WAYS BY
C               PLACING A D AS THE FIRST CHARACTER OF A LINE FOR
C               THE TEST WHICH IS NOT DESIRED.
C
C               TEST 1:
C               THE INTERPOLATION OF THE SURFACE IS COMPLETED
C               40 TIMES VARYING THE SAMPLE SPACING AND THE
C               DATA POINT DENSITY EACH TIME.  SAMPSPAC VARIES
C               FROM APPROXIMATELY 50 TO 600.  ISAMPSP/512
C               VARIES FROM .0957 TO 1.0966.
C
C               TEST 2:
C               THE VALUE FOR PICWIT.  THE INTERPOLATION OF THE
C               SURFACE IS COMPLETED 30 TIMES VARYING THE SAMPLE
C               SPACING AND THE DATA POINT DENSITY EACH TIME.
C               SAMPSPAC VARIES FROM APPROXIMATELY 10 TO 600.
C               ISAMPSP/512 VARIES FROM .0195 TO 1.0215.
C
C               DPTDEN IS DETERMINED SO THAT THE NUMBER OF
C               POINTS  INTERPOLATED PER PICTURE WILL BE
C               CONSTANT.
C
C
C       THE FOLLOWING FIGURE ILLUSTRATES THE ARRANGEMENT OF THE
C       VALUES USED IN THIS ROUTINE.
```



THE FOLLOWING FIGURE IDENTIFIES THE VARIABLES USED TO DESCRIBE THE SURFACE:



@DATA ARRAY, SAMPT(IX,IY)

```
C    *INTERPOLATED VALUES, PEND
C
C        THE PROGRAM FLOW STARTS AT (1) WHERE THE BOUNDARY LINES
C        ARE CALCULATED FOR THE PATCH, THEN FOLLOWS AN INCREMENT
C        OF Y UNTIL REACHING (2) THUS COMPLETING ONE 'SEGMENT.'
C        THE BOUNDARY LINES FOR THE PATCH ARE RECALCULATED AND
C        THE Y-INCREMENT CONTINUES UNTIL (3), COMPLETING ONE
C        'LINE.'  THE X VALUE IS THEN INCREMENTED TO (4) SO
C        BOUNDARY LINES ARE RECALCULATED.  Y- AND X-INCREMENTS
C        CONTINUE UNTIL (5) COMPLETING THE INTERPOLATION REGION
C        OR ONE 'PICTURE.'
C
C
C        DOUBLE PRECISION IS USED ON ALL VALUES USED FOR
C        STATISTICS.
         DOUBLE PRECISION SGSMDF,SGSMDF2,SGSMA,SMLND,SMLND2,
     -   SMLNA,SMDIF,SMDIF2,SMABS,TOTAVG,RMS,DAVG,LNAVG,LNRMS,
     -   LNDABS,SGAVG,SGRMS,SGABS,DIF,DIF2,DABS,DIFMXPS,DIFMXNG
         COMMON IX,IY,SAMPT(250,250),Z1,Z2,Z3,Z4,C1,C2,C3
C
C        NOTE:  FOR SAMPLE ARRAY, SAMPT(A,B), THE DIMENSIONS A
C                AND B MUST BE GREATER THAN THE PICTURE WIDTH
C                DIVIDED BY THE SAMPLE SPACING OR
C
C                        A,B > PICWIT/ISAMPSP.
C
C                IF SAMPT(A,B) IS CHANGED, REMEMBER TO CHANGE IT
C                IN THE SUBROUTINES.
C
C*************************************************************
C        FUNCTION USED IN COMPARISON
         FT(A,B)=127*(SIN(6.283185*(A)/1024))*(SIN(6.283185*(B)/
     -   1024))
C*************************************************************
C
         TYPE*,'ENTER THE PICTURE WIDTH'
         ACCEPT*,PICWIT
C
         TYPE*,'ENTER THE DESIRED NUMBER OF POINTS TO BE
     -   INTERPOLATED'
         ACCEPT*,POINTS
C
         WRITE(74,*)'INTERPOLATION BY STRAIGHT LINE APPROX.'
         WRITE(74,*)
         WRITE(74,*)'PICWIT=',PICWIT
         WRITE(74,*)'APPROXIMATE NUMBER OF POINTS INTERPOLATED=',
     -   POINTS
         WRITE(74,*)'     NOTE:  MAX. VALUE OF TEST FUNCTION=127'
         WRITE(74,*)'*************************'
         WRITE(74,*)' '
         WRITE(74,200)
```

```
200     FORMAT(3X,'ISAMPSP',3X,'DPTDEN',
     -  4X,'DIFMXPS',4X,'DIFMXNG',6X,'RMS/127',
     -  7X,'DAVG')
        WRITE(74,*)' '

        WRITE(75,*)'INTERPOLATION BY STRAIGHT LINE APPROX.'
        WRITE(75,*)
        WRITE(75,*)'PICWIT=',PICWIT
        WRITE(75,*)'APPROXIMATE NUMBER OF POINTS INTERPOLATED=',
     -  POINTS
        WRITE(75,*)'**************************'
        WRITE(75,*)' '
        WRITE(75,500)
500     FORMAT(5X,'SAMPSPAC',3X,'ISAMPSP',7X,'DPTDEN',8X,
     -  'COUNT',7X,'SAMPLE')
        WRITE(75,520)
520     FORMAT(57X,'ARRAY')
        WRITE(75,*)' '

        WRITE(76,*)'INTERPOLATION BY STRAIGHT LINE APPROX.'
        WRITE(76,*)
        WRITE(76,*)'PICWIT=',PICWIT
        WRITE(76,*)'APPROXIMATE NUMBER OF POINTS INTERPOLATED=',
     -  POINTS
        WRITE(76,*)'    NOTE:  MAX VALUE OF TEST FUNCTION=127'
        WRITE(76,*)'           ONE HALF THE PERDID OF TEST
     -  FUNCTION=512'
        WRITE(76,*)'**************************'
        WRITE(76,*)' '
        WRITE(76,600)
600     FORMAT(3X,'ISAMPSP/512',6X,'RMS/127')
        WRITE(76,*)' '

C
C       THE DISTANCE BETWEEN THE SAMPLE DATA POINTS IS THE
C       SAMPLE SPACING.
D       SAMPSPAC=50/1.064692     !TEST 1:  1.0640929=12**1/40
        SAMPSPAC=10/1.1462298    !TEST 2:  1.1462298=60**1/30
C
D       DO 210 K=1,40            !TEST 1
        DO 210 K=1,30            !TEST 2
        TYPE*,'K=',K
C
C
D       SAMPSPAC=SAMPSPAC*1.0640929      !TEST 1
        SAMPSPAC=SAMPSPAC*1.1462298      !TEST 2
        ISAMPSP=SAMPSPAC
                !MUST BE AN INTEGER TO MAINTAIN A GRID PATTERN
        ISAMP2=ISAMPSP*ISAMPSP
        DPAC=1./ISAMPSP
```

```
C       DPTDEN IS THE NUMBER OF DIVISIONS MADE BY THE TEST
C       POINTS PER SAMPLING INTERVAL.
C
        SQRPTS=SQRT(POINTS)      !CALCULATION OF DPTDEN ENSURES A
        IPICSMP=PICWIT*DPAC      !CONSTANT NUMBER OF INTERPOLATED
        IF(IPICSMP.LT.1) GOTO 400       !REQUIRED DATA BASE NOT
C                                        AVAILABLE
        DENOM=IPICSMP-2          !   POINTS FOR EACH STEP
        DPTDEN=SQRPTS/DENOM
                IF(DPTDEN.LT.1)THEN
                        TYPE*, 'ROUTINE IS INVALID FOR DPTDEN<1'
                        GOTO 210
                ENDIF
C
        XSAMPSP=FLOAT(ISAMPSP)
C
C       ROUTINE TO CREATE THE SAMPLED DATA ARRAY
        IEND=(PICWIT*DPAC)+1
        ICNTSMP=IEND*IEND-4

        DO 10 J=1,IEND
                XMIS=(I-1)*ISAMPSP
                DO 10 J=1,IEND
                        SAMPT(I,J)=FT(XMIS,(J-1)*XSAMPSP)
 10     CONTINUE
D       WRITE (74,100) ((SAMPT(I,J),J=1,IEND),I=1,IEND)
 100    FORMAT (' ',15F8.3)
C*************************************************************
C       OVERHAUSER-COONS FUNCTIONS BEGIN
C
C
        DELTA=ISAMPSP/DPTDEN
C
C       IPP AND IUP ARE IN UNIT  OF DELTA.
C       IPP AND IUP ARE THE LIMITS OF THE X- AND Y-INCREMENTS
C       SUCH THAT (IPP+1)*DELTA AND (IUP+1)*DELTA ARE THE FIRST
C       AND LAST WORLD COORDINATES INTERPOLATED, RESPECTIVELY.
C
        IUP=(IEND-2)*DPTDEN-1
        IPP=DPTDEN-1
C
C       CHECK FOR CROSSING BOUNDARY DUE TO ROUND-OFF ERROR IN
C       DELTA
        IF(ISAMPSP.GT.((IPP+1)*DELTA))IPP=IPP+1
        IF(PICWIT-ISAMPSP.LE.(IUP+1)*DELTA)IUP=IUP-1
C
C       (X,Y):=WORLD COORDINATES
        XX=IPP*DELTA
        X=XX             !INITIALIZE X VARIABLE FOR X-INCREMENT
C
C       (IRLX,IRLY):=THE LOWER LEFT BOUNDARY IN WORLD
```

```
C       COORDINATES CORRESPONDING TO THE FIRST POINT INSIDE
C       THE SAMPLE SQUARE.
        IRLX=ISAMPSP
C
C       (IRUX,IRUY):=THE UPPER RIGHT BOUNDARY IN WORLD
C       COORDINATES CORRESPONDING TO THE POTENTIAL LAST POINT
C       IN THIS SAMPLE SQUARE OR THE FIRST POINT IN A
C       NEIGHBORING SAMPLE SQUARE.
        IRUX=IRLX+ISAMPSP
C
C       (IX,IY):=THE INDEX OF THE SAMPLED DATA ARRAY
        IX=2            !INITIALIZE X INDEX
C
C*************************************************************
C       INITIALIZE VARIABLES FOR PICTURE STATISTICS
C
        ICNT=0          !COUNTER:  POINTS PER PICTURE
        SMDIF=0     !SUM OF DIFF FOR PICTURE
        SMDIF2=0    !SUM OF DIFF SQUARED FOR PICTURE
        SMABS=0     !SUM OF ABSOLUTE VALUE OF DIFF FOR PICTURE
        DIFMAX=0    !INITIALIZE VARIABLE USED TO COMPUTE
                     MAXIMUM ERROR
C*************************************************************
C
        DO 90 I=IPP,IUP !START OF X-INCREMENT, ONE LINE PER LOOP
                Y=XX    !INITIALIZE Y VARIABLE FOR Y-INCREMENT
                IY=2    !INITIALIZE Y INDEX
                IRLY=ISAMPSP
                IRUY=IRLY+ISAMPSP
                X=X+DELTA
                IF (X.GT.IRUX) THEN     !NEXT LINE
                        IX=IX+1         !INDEX FOR NEXT LINE
                        IRLX=IRUX
                        IRUX=IRUX+ISAMPSP
                ENDIF
C
        CALL PITCH
C
C       TX:=DISTANCE, X, WITHIN THE PATCH
                IX=X-IRLX
C
C       INITIALIZE VARIABLES FOR LINE AND SEGMENT STATISTICS
C
                ICNT1=0     !COUNTER:  POINTS PER SEGMENT
                ICNTA=0     !COUNTER:  POINTS PER LINE
                SMLND=0.0   !SUM OF LINE DIFFERENCES
                SMLND2=0.0 !SUM OF LINE DIFFERENCES SQUARED
                SMLNA=0.0   !SUM OF ABSOLUTE VALUE OF LINE DIFF
                SGSMDF=0.0 !SUM OF SEGMENT DIFFERENCES
                SGSMDF2=0.0!SUM OF SEGMENT DIFFERENCES SQUARED
                SGSMA=0.0   !SUM OF ABSOLUTE VALUE OF SEG DIFF
```

```
C
C
C
                DO 90 J=IPP,IUP      !Y-INCREMENT, ONE SEGMENT
C                                     PER LOOP
                        Y=Y+DELTA
C
                        IF (Y.GT.IRUY) THEN      !NEXT SEGMENT
                                IY=IY+1  !INDEX FOR NEXT SEGMENT
                                IRLY=IRUY
                                IRUY=IRUY+ISAMPSP
C
                        CALL PITCH
C
C
C       STATISTICS ARE DETERMINED PER LINE SEGMENT OF LENGTH
C       SAMPSPAC.
C
        ICNTA=ICNTA+ICNT1
        SMLND=SMLND+SGSMDF
        SMLND2=SMLND2+SGSMDF2
        SMLNA=SMLNA+SGSMA
        SGAVG=SGSMDF/ICNT1       !AVERAGE DIFF. IN SEGMENT
        SGRMS=SQRT(SGSMDF2/ICNT1-SGAVG*SGAVG) !SEGMENT RMS
        SGABS=SGSMA/ICNT1    !AVERAGE ABSOLUTE DIFF OF SEGMENT
C
        ICNT1=0          !INITIALIZE FOR NEXT SEGMENT
        SGSMDF=0.0       !CLEAR THE SUM OF SEGMENT DIFFERENCE
        SGSMDF2=0.0      !CLEAR THE SUM OF SEGMENT DIFF SQUARED
        SGSMA=0.0        !CLEAR THE SUM OF SEGMENT ABS DIFFENCES
C
                        END IF
C
C
C       TY:=DISTANCE, Y, WITHIN THE PATCH
                TY=Y-IRLY
C
C
C***********************************************************************
C
C       STRAIGHT-LINE APPROXIMATION
C
                FEND=Z1+TX*C1/ISAMPSP+TY*C2/ISAMPSP+TX*TY*C3/
     -                  ISAMP2
C
C***********************************************************************
C       THE CORRESPONDING EXACT VALUE IS:
                        EXACT=FT(X,Y)
C
C***********************************************************************
C       DATA DETERMINED IN PREPARATION FOR SEGMENT AND LINE
```

```
C       STATISTICS.
                        DIF=EXACT-FEND
                        DIF2=DIF*DIF     !SQUARE DIFF.
                        DABS=ABS(DIF)    !ABSOLUTE VALUE OF DIFF.
                        IF(DIF.GE.0) THEN
                                DIFMXPS=DMAX1(DIFMXPS,DIF)
                        ELSE
                                DIFMXNG=DMAX1(DIFMXNG,DABS)
                        ENDIF
C*************************************************************
C       SUMMATION OF SEGMENT DATA FOR THE CALCULATION OF
C       SEGMENT STATISTICS.
        SGSMDF=SGSMDF+DIF        !COMPUTE THE SUM OF SEGMENT
C                                 DIFFERENCES
        SGSMDF2=SGSMDF2+DIF2     !COMPUTE THE SUM OF SEG DIFF
C                                 SQUARED
        SGSMA=SGSMA+DABS         !COMPUTE THE SUM OF SEG DIFF
C                                 ABSOLUTE VALUE
        ICNT1=ICNT1+1            !INCREMENT SEGMENT COUNTER
C
        IF ((J.EQ.IUP).AND.(Y.LE.IRUY))THEN !SEGMENT STATS FOR
C                                             LAST SEGMENT
        SMLND=SMLND+SGSMDF
        SMLND2=SMLND2+SGSMDF2
        SMLNA=SMLNA+SGSMA
D       SGAVG=SGSMDF/ICNT1       !AVERAGE DIFF IN SEGMENT
D       SGRMS=SQRT(SGSMDF2/ICNT1-SGAVG*SGAVG)     !SEGMENT RMS
D       SGABS=SGSMA/ICNT1        !AVERAGE ABSOLUTER VALUE OF DIFF
C                                  IN SEGMENT
        ICNTA=ICNTA+ICNT1
        ENDIF
 80     CONTINUE
C
C*************************************************************
C       SUMMATION OF LINE DATA FOR THE CALCULATION OF PICTURE
C       STATISTICS.
        SMDIF=SMDIF+SMLND
        SMDIF2=SMDIF2+SMLND2
        SMABS=SMABS+SMLNA
        ICNT=ICNT+ICNTA
C*************************************************************
C       LINE STATISTICS ARE CALCULATED
D       LNAVG=SMLND/ICNTA        !COMPUTE AVERAGE LINE DIFFERENCE
D       LNRMS=SQRT(SMLND2/ICNTA-LNAVG*LNAVG)  !COMPUTE LINE RMS
D       LNDABS=SMLNA/ICNTA   !COMPUTE AVERAGE ABSOLUTE DIFF/LINE
C*************************************************************
 90     CONTINUE                          !END X-INCREMENT LOOP
C*************************************************************
C       PICTURE STATISTICS ARE CALCULATED
C
        TOTAVG=SMDIF/ICNT    !COMPUTE AVERAGE ERROR FOR PICTURE
```

```
      RMS=SQRT(SMDIF2/ICNT-TOTAVG*TOTAVG) !COMPUTE RMS PICTURE
      DAVG=SMABS/ICNT       !COMPUTE AVERAGE ABSOLUTE ERROR FOR
C                            PICTURE

      XSMP512=ISAMPSP/512.
      RMS127=RMS/127
      WRITE(74,220)ISAMPSP,DPTDEN,
     - DIFMXPS,DIFMXNG,RMS127,DAVG
220   FORMAT(I8,F11.4,2(F11.4),F13.7,F11.4)
      IF((DIFMXPS.LT.1E-4).OR.(DIFMXNG.LT.1E-4))THEN
        WRITE(74,*)'          DIFMXPS=',DIFMXPS,'DIFMXNG=',
     -  DIFMXNG
      ENDIF

      WRITE(75,510)SAMPSPAC,ISAMPSP,DPTDEN,ICNT,ICNTSMP
510   FORMAT(F13.4,I10,F13.4,2(I13))

      WRITE(76,630)XSMP512,RMS127
630   FORMAT(F14.4,F13.7)
210   CONTINUE
C*************************************************************
400   STOP
      END
      SUBROUTINE PITCH
C
C
C     THIS SUBROUTINE SUMS Z VALUES TO BE USED IN ONE PATCH.
C
      COMMON IX,IY,SAMPT(250,250),Z1,Z2,Z3,Z4,C1,C2,C3
C
C
D     WRITE(74,*)'IX,IY',IX,IY
            IX1=IX+1
            IY1=IY+1
            Z1=SAMPT(IX,IY)     !LOWER LEFT CORNER OF PATCH
            Z2=SAMPT(IX1,IY)    !LOWER RIGHT CORNER OF PATCH
            Z3=SAMPT(IX,IY1)    !UPPER LEFT CORNER OF PATCH
            Z4=SAMPT(IX1,IY1)   !UPPER RIGHT CORNER OF PATCH
D     WRITE(74,*)'Z1=',Z1,'Z2=',Z2,'Z3=',Z3,'Z4=',Z4
D     WRITE(74,*)'IX1,IY1=',IX1,IY1
C
      C1=Z2-Z1
      C2=Z3-Z1
      C3=Z4-Z3-Z2+Z1
C
D     WRITE(74,*)'C1,C2,C3=',C1,C2,C3
D     WRITE(74,*)' '
C
      RETURN
      END
C
```

APPENDIX HA

NOISE AND TEXTURE

Appendices HB and HC contain the routines that are used to generate noise and texture for the REALSCAN scenes.

APPENDIX H6

RLPCAL.FOR

```
C       RPPCAL.FOR
C
C       THIS ROUTINE COMPUTES SUN VECTOR REFLECTANCE FOR A BYTE
C       DATA BASE.
C
        INTEGER*2 JDAT(512,256),ITEMBUF(512,512)
        BYTE IDAT(512,512)
        EQUIVALENCE(IDAT,JDAT)
        COMMON  ITEMBUF
C
        TYPE*,'ENTER THE FILE TO READ IDAT(BYTE) FROM'
        ACCEPT*,IFILE
        READ(IFILE,1000) ((JDAT(I,J),I=1,512),J=1,256)
1000    FORMAT(66A2)
C
C       CREATE AN ALTITUDE PICTURE
C
        TYPE*,' DO YOU WISH TO CREATE AN ALTITUDE PICTURE ?
     *  (1=YES)'
        READ*,IAN
        IF(IAN.NE.1)GOTO 123
        DO 120 I=1,512
           DO 120 J=1,512
              ITEMBUF(J,I)=IDAT(J,I)+128
120     CONTINUE
        CALL SDICBW(0,0,0)
C
C       THE INTENT OF THE SLOPE FACTOR IS TO SCALE THE MAXIMUM
C       SLOPE IN THE DATA BASE TO A PREDETERMINED MAXIMUM ANGLE
C
123     TYPE*,'ENTER THE FILTER SIZE OF THE NOISE FILE YOU ARE
     *  USING'
        READ*,FLTSIZE
20      TYPE*,'ENTER THE SLOPE FACTOR'
        TYPE*,'    .0003      TERRAIN'
        TYPE*,'    .002       TREES'
        TYPE*,'    .005       DETAIL'
        READ*,SLPFAC
```

```
      SLOPE=SLPFAC*FLTSIZE
C
C
C----------------INPUT SCENE PARAMETERS--------------------
C
C
      TYPE*,'Type the polar sun angle from the Z axis
     *  (DEGREES)'
      ACCEPT*,TZDEG
      TZRAD=TZDEG/57.3
      SINZ=SIN(TZRAD)
      TYPE*,'Type the cylindrical sun angle from the X axis
     *  toward the Y axis (DEGREES)'
      ACCEPT*, TXDEG
      TXRAD=TXDEG/57.3
      C=COS(TZRAD)
      A=SINZ*COS(TXRAD)
      B=SINZ*SIN(TXRAD)
      DO 40 J=1,512
      DO 40 I=1,512
            K=IDAT(I,J)
            IF(I.NE.512) THEN
                  PDX=IDAT(I+1,J)-K
            ELSE
                  PDX=IDAT(511,J)-K
            ENDIF
            IF(J.NE.512) THEN
                  PDY=IDAT(I,J+1)-K
            ELSE
                  PDY=IDAT(I,511)-K
            ENDIF
            PDX=PDX*SLOPE
            PDY=PDY*SLOPE
            REMAG=SQRT(1+PDX*PDX+PDY*PDY)
            IR=255*(-A*PDX -B*PDY +C)/REMAG
            IF(IR.LT.0) IR=0
            IF(IR.GT.255) IR=255
            ITEMBUF(I,J)=IR
40    CONTINUE
1734  TYPE*, 'SHALL WE MAKE A PICTURE? (YES=1)'
      ACCEPT *, IYES
      IF (IYES.NE.1) GO TO 373
      TYPE*, ' '
      TYPE*, 'DATA WILL BE OUTPUT TO THE DICOMED'
      CALL SDICBW(0,0,0)
 373  TYPE*,'DO YOU WANT TO DO ANOTHER PICTURE?(YES=1)'
      ACCEPT*, IAN
      IF (IAN.EQ.1) GO TO 20
      STOP
      END
```

APPENDIX HC

TRYTEX.FOR

```
C       UNITEX.FOR, APTEX.FOR, RLTEX.FOR
C
C       THIS PROGRAM GENERATES A SQUARE ARRAY OF FILTERED NOISE
C       THAT CAN BE REDISTRIBUTED TO 4 POSSIBLE AMPLITUDE
C       DISTRIBUTIONS.
C
C       PROGRAM BY: G. BECKER, S. RICHIE, P. GATT, R. LEBLANC,
C                   AND DR. PATZ
C       LATEST REVISION DATE:   28  JULY 81     PL
C
C
C       APPROPRIATE ARRAYS ARE DIMENSIONED AND DATA TYPES
C       SPECIFIED
C
C
      BYTE JBUF(560,560)
      DIMENSION FLTARA(560,29),FLTEMP(560,29)
      DIMENSION B(1001),BEQ(1001),BUCK(3001),FRAC(10),W(31,31)
      INTEGER*2 IRO*(29),IBUF(560,560),KBUF(280,560)
      INTEGER QSIZE,IFLTSIZE(10),IDIST(10),FINSIZ,G(3001)
      CHARACTER*10 DSTRB(4),FLTYPE
      EQUIVALENCE (BEQ(2),B(1)),(JBUF,KBUF)
      COMMON/IBUF/IBUF
      COMMON/BUF/BUF(560,560)
C
      DATA (DSTRB(I),I=1,4)/'UNIFORM','TRIANGULAR',
     *  'PARABOLIC','CUSP'/
C
C       I/O TO THE TERMINAL FOR EACH RUN
C
C
      TYPE*,'INPUT COMMAND:'
      TYPE*,'          (1) CREATE AND FILTER NOISE'
      TYPE*,'          (2) READ I*2 NOISE FOR REDISTRIBUTION'
      TYPE*,'          (3) READ BYTE NOISE FOR REDISTRIBUTION'
      READ*,ICOMAND
      GOTO(30),100,100),ICOMAND
```

```
100     TYPE*,'ENTER THE FILE FROM WHICH TO READ THE DATA.'
        READ*,IFILE
        TYPE*,'DOES THIS FILE HAVE HEADINGS ? (1=YES)'
        READ*,ANS
        IF(ANS.EQ.1)THEN
           READ(IFILE,200),FLTYPE
           READ(IFILE,200),DSTRB(1)
           READ(IFILE,*),FRACT,ISIZE,NPAS,FINSIZ
           TYPE*,'        FILTER TYPE = ',FLTYPE
           TYPE*,'       DISTRIBUTION = ',DSTRB(1)
           TYPE*,' DISTRIBUTION PEAK = ',FRACT
           TYPE*,'        FILTER SIZE = ',ISIZE
           TYPE*,'  NUMBER OF PASSES = ',NPAS
           TYPE*,'          FILE SIZE = ',FINSIZ
        ELSE
           TYPE*,'ENTER THE SIDE DIMENSION OF THE FILE'
           READ*,FINSIZ
        ENDIF
200     FORMAT(A19)
        FILTMAX=0
        FILTMIN=9999
        IF(ICOMAND.EQ.2)THEN
           READ(IFILE,230),((IBUF(I,J),I=1,FINSIZ),J=1,FINSIZ)
           DO 260 I=1,FINSIZ
              DO 260 J=1,FINSIZ
                 VAL=IBUF(J,I)
                 BUF(J,I)=VAL
                 IF(VAL.GT.FILTMAX)FILTMAX=VAL
                 IF(VAL.LT.FILTMIN)FILTMIN=VAL
260        CONTINUE
        ELSE
           READ(IFILE,230),((KBUF(I,J),I=1,FINSIZ/2),J=1,FINSIZ)
           DO 250 I=1,FINSIZ
              DO 250 J=1,FINSIZ
                 VAL=JBUF(J,I)+128
                 BUF(J,I)=VAL
                 IF(VAL.GT.FILTMAX)FILTMAX=VAL
                 IF(VAL.LT.FILTMIN)FILTMIN=VAL
250        CONTINUE
        ENDIF
230     FORMAT(66A2)
        TYPE*,'ENTER THE NEW DISTRIBUTION'
        TYPE*,'       (1) UNIFORM'
        TYPE*,'       (2) TRIANGULAR'
        TYPE*,'       (3) PARABOLIC'
        TYPE*,'       (4) CUSP'
        READ*,IDIST(1)
        IF (IDIST(1).NE.1)THEN
           TYPE*,'ENTER A NUMBER BETWEEN 0 AND 1 THAT WILL'
           TYPE*,'IDENTIFY THE PEAK OF THE DISTRIBUTION'
           READ*,FRAC(1)
```

```
      ENDIF
      TYPE*,'ENTER THE UPPER BOUND FOR THE FINAL VALUES'
      TYPE*,'THE LOWER BOUND IS ASSUMED TO BE 0'
      TYPE*,'NORMAL IS 255 FOR DICOMED PICTURES'
      READ*,IRANGE
      IRANGE=IRANGE+1
      NPAS=1
      ICNT=1
      IFFSA=FINSIZ
      GOTO 1450
C
C     I/O FOR NOISE CREATION
C
300   TYPE*,'ENTER THE SIDE DIMENSION OF THE FINAL ARRAY'
      TYPE*,'NORMAL IS 512 FOR DICOMED PICTURES'
      READ*,FINSIZ
      IFFSA=FINSIZ
      TYPE*,'ENTER THE NUMBER OF PASSES TO BE MADE THROUGH THE
     * FILTER'
      TYPE*,'NO MORE THAN 10'
      READ*,NPAS
      DO 500 I=1,NPAS
400      TYPE*,'ENTER THE FILTER SIZE FOR PASS',I
         TYPE*,'THIS MUST BE ODD AND GREATER THAN ONE'
         READ*,IFLTSIZE(I)
         IF (((IFLTSIZE(I)/2)*2).EQ.IFLTSIZE(I))THEN
            TYPE*,'MUST BE ODD !!'
            GOTO 400
         ENDIF
         IFFSA=IFFSA+IFLTSIZE(I)-1
         TYPE*,'ENTER THE FINAL DISTRIBUTION FOR PASS',I
         TYPE*,'       (1) UNIFORM'
         TYPE*,'       (2) TRIANGULAR'
         TYPE*,'       (3) PARABOLIC'
         TYPE*,'       (4) CUSP'
         READ*,IDIST(I)
         IF (IDIST(I).NE.1)THEN
            TYPE*,'ENTER A NUMBER BETWEEN 0 AND 1 THAT WILL'
            TYPE*,'IDENTIFY THE PEAK OF THE DISTRIBUTION'
            READ*,FRAC(I)
         ENDIF
500   CONTINUE
      TYPE*,'ENTER THE UPPER BOUND FOR THE FINAL VALUES'
      TYPE*,'THE LOWER BOUND IS ASSUMED TO BE 0'
      TYPE*,'NORMAL IS 255 FOR DICOMED PICTURES'
      READ*,IRANGE
      IRANGE=IRANGE+1
      TYPE*,'ENTER A SEED FOR THE RANDOM NUMBER GENERATOR'
      READ*,II
      INIT=II
      ISPREAD=1
```

```
C       FILTERING LOOP
C
        DO 3200 ICNT=1,NPAS
        IFSIZE=IFLTSIZE(ICNT)
        FRACT=FRAC(ICNT)
C
C
C       SETUP THE WEIGHT FOR FILTER ARRAY
C
C
600     FAC=3.14159265/(IFSIZE+1)
        ICENT=IFSIZE/2+1
        IDIAM=IFSIZE+1
        W(ICENT,ICENT)=1.
        DO 700 J=ICENT+1,IFSIZE
           K=J-ICENT
           SQR=K*K
           KY=IDIAM-J
           DO 700 I=ICENT,J
              K=I-ICENT
              XK=SQRT(K*K+SQR)
              KX=IDIAM-I
              IF(XK.GE.ICENT)THEN
                 VAL=0.0
              ELSE
                 VAL=COS(XK*FAC)
              ENDIF
              W(I,J)=VAL
              W(J,I)=VAL
              W(I,KY)=VAL
              W(KY,I)=VAL
              W(J,KX)=VAL
              W(KX,J)=VAL
              W(KX,KY)=VAL
              W(KY,KX)=VAL
700     CONTINUE
C
C
C       WITH THE FILTER SIZE CORRECT, WE GENERATE AN ARRAY THAT
C       IS (511+FILTER SIZE*  OF PASSES) X (FILTER SIZE) AND
C       FILLS IT WITH RANDOM NUMBERS
C
        LIM=IFFSA
        IFFSA=IFFSA-IFSIZE+1
        TYPE*,'FILTERING THE ARRAY FOR PASS  ',ICNT
        DO 900 J=1,IFSIZE
           DO 800 I=1,LIM
              IF(ICNT .GE. 2) THEN
                 FLTARA(I,J)=IBUF(I,J)
              ELSE
                 FLTARA(I,J)=RAN(II)
```

```
                ENDIF
800           CONTINUE
              IROW(J)=J
900        CONTINUE
C
C
C          THE AVERAGE IS NOW CREATED FOR A SQUARE ARRAY OF THE
C          DIMENSION OF THE FILTER SIZE. THIS IS A WEIGHTED AVERAGE
C
C
           FILTMIN=9999
           FILTMAX=0
           DO 1400 J=1,IFFSA
              DO 1100 I=1,IFFSA
                 AVG=0.0
                 DO 1000 J2=1,IFSIZE
                    J3=IROW(J2)
                    DO 1000 I2=1,IFSIZE
                       I3=I+I2-1
                       AVG=AVG+W(I2,J3)*FLTARA(I3,J2)
1000                CONTINUE
                    BUF(I,J)=AVG
                    IF(AVG .GT. FILTMAX) FILTMAX=AVG
                    IF(AVG .LT. FILTMIN) FILTMIN=AVG
1100           CONTINUE
C
C          CREATE NEW LINE AND CHANGE LINE POINTERS
C
              DO 1400 J2=1,IFSIZE
                 IF(IROW(J2).LT.IFSIZE)THEN
                    IROW(J2)=IROW(J2)+1
                 ELSE
                    IROW(J2)=1
                    IF(ICNT.GE.2)THEN
                       JINCR=J+1
                       DO 1200 I2=1,LIM
                          FLTARA(I2,J2)=IBUF(I2,JINCR)
1200                   CONTINUE
                    ELSE
                       DO 1300 I2=1,LIM
                          FLTARA(I2,J2)=RAN(II)
1300                   CONTINUE
                    ENDIF
                 ENDIF
1400       CONTINUE
C
C
C          SET UP FOR REDISTRIBUTION
C
C
1450       IF (ICNT.EQ.NPAS)THEN
```

```
            QSIZE=IRANGE
         ELSE
            QSIZE=1000
         ENDIF
         IFRAC=FRACT*QSIZE
1500     IF(QSIZE.GT.256)THEN
            LIMT=3*QSIZE
         ELSE
            LIMT=10*QSIZE
         END IF
         EPSILON=(FILTMAX-FILTMIN)/LIMT
         DEP=1./EPSILON
C
C
C        CREATE THE DISTRIBUTION ARRAY
C
C
         DO 1650 I=1,LIMT+1
            G(I)=0
1650     CONTINUE
         DO 1700 I=1,IFFSA
            DO 1700 J=1,IFFSA
               K=(BUF(J,I)-FILTMIN)*DEP + 1
               G(K)=G(K) + 1
1700     CONTINUE
         G(LIMT)=G(LIMT)+G(LIMT+1)
C
C
         GOTO(1800,2000,2200,2400) IDIST(ICNT)
C
C
C  UNIFORM
C
1800     TEMP1=(1.*IFFSA*IFFSA)/QSIZE
         DO 1900 I=1,QSIZE
            BUCK(I)=TEMP1
1900     CONTINUE
         GOTO 2600
C
C  TRIANGLE
C
2000     TEMP1=(1.*IFFSA*IFFSA)/QSIZE**2
         TEMP2=2*QSIZE+1
         DO 2100 I=1,QSIZE
            IF(I.LE.IFRAC)THEN
               BUCK(I)=(2*I-1)*TEMP1/FRACT
            ELSE
               BUCK(I)=(TEMP2-2*I)*TEMP1/(1-FRACT)
            ENDIF
2100     CONTINUE
         GOTO 2600
```

```
C  PARABOLA
C
2200     TEMP1=(3.*IFFSA*IFFSA)/(2*QSIZE**2)
         DO 2300 I=1,QSIZE
            IF(I.LE.IFRAC)THEN
               BUCK(I)=(TEMP1/FRACI)*(2*I-1-((1+3.*I*(I-1))/
     *                    (3.*FRACT*QSIZE)))
            ELSE
               TEMP2=QSIZE-I+1
               BUCK(I)=(TEMP1/(1-FRACT))*(2*TEMP2-1-((1+3*TEMP2*
     *                    (TEMP2-1))/(3.*(1-FRACT)*QSIZE)))
            ENDIF
2300     CONTINUE
         GOTO 2600
C
C  CUSP
C
2400     TEMP1=(1.*IFFSA*IFFSA)/QSIZE/QSIZE/QSIZE
         DO 2500 I=1,QSIZE
            IF(I.LE.IFRAC)THEN
               BUCK(I)=(1+3.*I*(I-1))*TEMP1/(FRACT*FRACT)
            ELSE
               TEMP2=QSIZE-I+1
               BUCK(I)=(1+3*TEMP2*(TEMP2-1))*TEMP1/((1-FRACT)*
     *                    (1-FRACT))
            ENDIF
2500     CONTINUE
C
C
C        TEST BUCKET SIZES
C
2600     TOT=0
         SUM=0
         DO 123 I=1,QSIZE
            SUM=SUM+BUCK(I)
123      CONTINUE
         DO 124 I=1,LIMI
            TOT=TOT+G(I)
124      CONTINUE
         TYPE*,'SUM OF ARRAY BUCKETS = ',TOT
         TYPE*,'SUM OF DISTRIBUTION BUCKETS = ',SUM
         TYPE*,'N**2 = ',IFFSA*IFFSA
C
C
C        COMPUTE THE BREAKPOINTS
C
C
         K=0
         A=0
         DO 2800 I=1,QSIZE
2750        IF(A.LT.BUCK(I))THEN
```

```
                K=K+1
                IF(K.GT.LIMI)GOTO 2800
                A=A+G(K)
                GOTO 2750
             ENDIF
             A=A-BUCK(I)
             B(I)=(K-A/G(K))*EPSILON+FILTMIN
2800      CONTINUE
          B(QSIZE)=FILTMAX
C
C
C CONVERT BUF, THE FILTERED NOISE ARRAY, FROM A GAUSIAN
C DISTRIBUTION TO ANOTHER DISTRIBUTION STILL POSSESSING THE
C SPACIAL FILTERED PROPERTIES. IBUF HAS DATA POSSESSING SOME
C DISTRIBUTION AND A SPACIAL PERIOD UP TO ABOUT (2/PI)*IFFSA.
C 1/2 INTERVAL SEARCH
C
          MAX=0
          MIN=99999
          NTIM=LOG(FLOAT(QSIZE))/LOG(2.0)
          IF(QSIZE.GT.2**NTIM) NTIM=NTIM+1
          BEQ(1)=FILTMIN
C
          DO 3100 I=1,IFFSA
             DO 3100 J=1,IFFSA
                LOW=0
                IUP=QSIZE
                KV=IUP/2
                VAL=BUF(J,I)
                DO 3000 K=1,NTIM
                   IF(VAL.LT.B(KV))THEN
                      IUP=KV
                   ELSE
                      LOW=KV
                   ENDIF
                   KV=(IUP+LOW)/2
3000            CONTINUE
                IBUF(J,I)=LOW
                IF(MAX.LT.LOW) MAX=LOW
                IF(MIN.GT.LOW) MIN=LOW
3100      CONTINUE
C
3200      CONTINUE
C
C
C         WRITE OR SCALE FINAL FILE
C
3300      TYPE*,'INPUT COMMAND     (1) = WRITE ARRAY IN BYTE FORM'
          TYPE*,'                  (2) = WRITE ARRAY IN INTEGER*2
     *    FORM'
          TYPE*,'                  (3) = SCALE ARRAY'
```

```
      TYPE*,'                    (4) = STOP'
      READ*, ICOMAND
      GOTO(3600,3600,3400,4200),ICOMAND
C
C
C     SCALE ARRAY
C
3400  TYPE*,'INPUT A MINIMUM AND A MAXIMUM FOR A SCALE'
      TYPE*,'THESE MUST BE INTEGER'
      READ*,JMIN,JMAX
      SCALE=(JMAX-JMIN)/(MAX-MIN)
      DO 3500 I=1,FINSIZ
         DO 3500 J=1,FINSIZ
            IBUF(J,I)=SCALE*(IBUF(J,I)-MIN) +JMIN
3500  CONTINUE
      GO TO 3300
C
C
C     WRITE DATA ABOUT THE ARRAY
C
3600  TYPE*,'        FILTER TYPE = POLAR'
      TYPE*,'       DISTRIBUTION = ',DSTRB(IDIST(NPAS))
      TYPE*,' DISTRIBUTION PEAK = ',FRAC(NPAS)
      TYPE*,'        FILTER SIZE = ',IFLTSIZE(NPAS)
      TYPE*,'  NUMBER OF PASSES = ',NPAS
      TYPE*,'          FILE SIZE = ',FINSIZ
      TYPE*,' '
      TYPE*,'INTO WHICH FILE DO YOU WISH TO WRITE THE ARRAY ?'
      READ*,IFILE
      TYPE*,'DO YOU WISH TO RECORD THE ABOVE DATA ABOUT THE
     * FILE'
      TYPE*,'AT THE BEGINNING OF IT ? (1=YES)'
      READ*,ANS
      IF(ANS.NE.1)GOTO 3700
      WRITE(IFILE,*),'POLAR'
      WRITE(IFILE,*),DSTRB(IDIST(NPAS))
      WRITE(IFILE,*),FRAC(NPAS)
      WRITE(IFILE,*),IFLTSIZE(NPAS)
      WRITE(IFILE,*),NPAS
      WRITE(IFILE,*),FINSIZ
3700  GOTO(4000,3800),ICOMAND
C
C
C     WRITE ARRAY AS INTEGER*2
C
C
3800  CONTINUE
      TYPE*,' WRITING IBUF TO FILE  ',IFILE
      WRITE(IFILE,230) ((IBUF(J,I),J=1,FINSIZ),I=1,FINSIZ)
      GOTO 3300
C
```

```
C
C
C
4000    TYPE*,'CONVERTING TO BYTE'
        DO 4100 I=1,FINSIZ
           DO 4100 J=1,FINSIZ
              JBUF(J,I)=IBUF(J,I)-128
4100    CONTINUE
        TYPE*,' WRITING KBUF INTO FILE ',IFILE
        WRITE(IFILE,230) ((KBUF(I,J),I=1,FINSIZ/2),J=1,FINSIZ)
        GOTO 3300
C
C
C       STOP !!!!
C
4200    STOP
        END
```

APPENDIX IA

TST8W.FOR

```
C       TST8W.FOR
C
C       THIS PROGRAM IS DICOMED ROUTINE FOR PAINTING THE
C       BLACK AND WHITE DATA BASE FOR INITIAL TESTING.
C
        SUBROUTINE FRAME1
        INCLUDE 'DRA0:[UTILITY.DICOMED]COMMON.FOR'
        REAL RX,RY,PI,TEST1,TEST2
        INTEGER IDATBAS,IRTH0,IWRT,IDICO,K
        INTEGER*2 ITEM1(1024)
C
        TYPE*,'INPUT DATABASE CHOICE: 1=RADIAL LINES'
        TYPE*,'                       2=CONCENTRIC CIRCLES'
        ACCEPT*,IDATBAS
        TYPE*,'INPUT IR DEVIATION FROM GRAY AT THETA=0
     *  (RANGE IS'
        TYPE*,' 0 TO 127 WHERE 127 IS FULL SCALE DEVIATION)'
        ACCEPT*,IRTH0
        TYPE*,'WRITE ARRAY TO DISK FILE   ? (0=NO WRITE)'
        ACCEPT*,IWRT
C       TYPE*,'ENTER RESOL: H=1,M=2,L=3 (NORMAL=3)'
C       ACCEPT*,MODE
        MODE=3
C       TYPE*,'ENTER MAGZ CONTROL: 0 OR 1 (NORMAL=0)'
C       ACCEPT*, MAGZ
        MAGZ=0
        TYPE *,'ENTER NELE (512 OR 1024)'
        ACCEPT *,NELE
        TYPE *,'ENTER NLIN (512 OR 1024)'
        ACCEPT *,NLIN
C
        TYPE*,'DO YOU REALLY WANT A 256 RESOLUTION?'
        TYPE*,' YES=1,NO=0'
        ACCEPT*,IRW256
        CALL VIDSET
        NCOL(1)=0
        NCOL(2)=0
        CALL FILTER
```

```
      PI=3.14159
C
      IDO=NLIN
      IF(IR.256.EQ.1) IDO=256
C
C
      DO 444 I=1,IDO
      CALL VDINIT
      DO 555 J=1,IDO
            GOTO(10,20) IDATBAS
            TYPE*,'ERROR:IDATBAS=',IDATBAS
            STOP
C
10    TEST1=PI/256
      TEST2=ATAN2(I-.99999,J-.99999)
      K=TEST2/TEST1
300     IF((K/2)*2.NE.K) GOTO 100
      GOTO 200
C
20    TEST2=J
      TEST1=I
      TEST1=(TEST1**2+TEST2**2)**.5
      K=TEST1/(((3*TEST1)/(1.414*NLIN))+1)
      TEST2=ATAN2(I-.99999,J-.99999)
      GOTO 300
C
200   BUF1(J)=((-2.*IRTH0)/PI)*TEST2-128+IRTH0 ! BLACK
      GOTO 554
100   BUF1(J)=((2.*IRTH0/PI))*TEST2+127-IRTH0  ! WHITE
554   GOTO(555) IWRT+1
      ITEM1(J)=BUF1(J)
      IF(BUF1(J).LT.0) ITEM1(J)=BUF1(J)+256
555   CONTINUE
      IF(IR.256.EQ.1) THEN
            DO 333 KK=256,1,-1
            BUF1(2*KK)=BUF1(KK)
            BUF1(2*KK-1)=BUF1(KK)
333         CONTINUE
            CALL VIDOUT
            CALL VDINIT
      ENDIF
      CALL VIDOUT
      IF(IWRT.NE.0) WRITE(IWRT,9) (ITEM1(K),K=1,NLIN)
      TYPE*,'LINE',I
444   CONTINUE
      CALL OPCHK
      TYPE*,'THE END'
      CODE=-1
9     FORMAT(65A2)
      END
```

APPENDIX IB

SRCAMERA.FOR

```
C       SRCAMERA.FOR
C
        PROGRAM SRCAMERA
        INCLUDE   'DRA0:[UTILITY.DICOMED]COMMON.FOR'
        INTEGER*4 SYSSASSIGN
        REAL TIME1,TIME2,SECNDS,FDELTA,IDELTA,FDELTAM
        REAL FDELTAS,IDELTAH,TMP,IDELTAM,IDELTAS
        INCLUDE 'DRA0:[UTILITY.DICOMED]ID.FOR'
        INCLUDE 'DRA0:[UTILITY.DICOMED]DEF.FOR'
C ASSIGN THE DR11B TO A CHANNEL
        ISTAT=SYSSASSIGN('UZA0:',ICHAN,,)
C IF WE CANNOT DO THE ASSIGN WE CANNOT CONTINUE
        IF(.NOT. ISTAT)GO TO 800
C
C               SET UP AST FOR RECEIPT OF UNSOLICITED INPUT
        CODE=0
        MODE=3
        BUFREF=    '77577' O
        BUFREQ=0
        CALL VIDSET
        MODE=0
        CODE=1                      ! SET UP CALL FOR FRAME1
C
C


        TIME2=SECNDS(0.0)
        EXTFLG = 0
C
C
10      GOTO (100,200,300,400,500,600)CODE+2
C
100     EXTFLG=1
        GOTO 1000
C
200     WRITE (6,*)'CODE EQUALED 0   -   THIS IS AN ERROR
     *  CONDITION'
        EXTFLG=1
        GOTO 1000
```

```
 300     CALL FRAME1
         GOTO 1000
C
400      CALL FRAME2
         GOTO 1000
C
500      CALL PRIPRO
         GOTO 1000
C
600      CALL FRAME3
C
C                GET THE TIME
         FDELTA=SECNDS(TIME1)
         TDELTA=SECNDS(TIME2)
C                CALCULATE THE ELAPSED TIME
C
         FDELTAM=INT(FDELTA/60.)
         FDELTAS=FDELTA-(FDELTAM*60)
C
C           CALCULATE THE ELAPSED TIME FOR THE TOTAL SEQUENCE
C
         TDELTAH=INT(TDELTA/3600.)
         TMP=TDELTA-(TDELTAH*3600)
         TDELTAM=INT(TMP/60.)
         TDELTAS=TMP-(TDELTAM*60)
C
         IF (ISW12 .EQ. 1) THEN
            WRITE (6,700) FDELTAM,FDELTAS,TDELTAH,TDELTAM,TDELTAS
         END IF
C
C                CHECK IF PRINTOUT IS DESIRED ON PRINTER
C
         IF (ISW11 .EQ. 1) THEN
            WRITE (5,700)FDELTAM,FDELTAS,TDELTAH,TDELTAM,TDELTAS
         END IF
C
700      FORMAT (///T15,'ELAPSED TIME FOR FRAME'/T50,F10.2,T65,
     1   'MINUTES',T75,F15.10,T100,'SECONDS'//T15,'ELAPSED TOTAL
     2   TIME',/T35,F6.2,T45,'HOURS',T50,F10.2,T65,'MINUTES',T75,
     3   F15.10,T100,'SECONDS')
C
C
         GOTO 1000
C
1000     CLOSE (UNIT=1)
         CLOSE (UNIT=2)
         CLOSE (UNIT=3)
         CLOSE (UNIT=4)
C
         IF (EXTFLG .EQ. 1)      THEN
         WRITE (6,*) 'ERROR CONDITION'
```

```
            GOTO 9999
            END IF
C
            GOTO 10
C
800         PRINT 801
801         FORMAT('0*** C A N N O T   A S S I G N   D R 1 1 B ***')
            EXTFLG=1
            GOTO 1000
C
C9999       CALL EXIT
C
C
C
9999        STOP
            END
```

APPENDIX JA

ANALYSIS OF APPENDIX J SOLUTIONS

This Appendix compares the three solutions of Appendix J (the lower bound problem), and a two other solutions which are presented in this Appendix. The five solutions are labeled A through D and D'. The equations for the five solutions are given in equations JA-1 through JA-5.

Solution D is similiar to solution C. Solution C uses the the last computed point along the lower bound line (X,Y) to compute the new point (X',Y'). Solution D uses some intermediate point (X+dX,Y+dY) to compute (X',Y'). This intermediate point is choosen, such that solution D is more accurate than solution C. The intermediate point is between the old point (X,Y) and the new point (X',Y').

The intermediate point is computed by adding a portion of the last distance between data points to the latest data point. This portion is computed by multiplying the last distance by a scale factor SCALE. Thus SCALE is less than 1 and greater than 0.

Solution D can be modified, such that SCALE is a constant and a power of 2. The case where SCALE is a constant and equal to .5 is solution D'.

Solution A: JA-1

$$T = (X^2 + Y^2)(1 + K)(K1*X + K2*Y)$$
$$X' = X + K1*T$$
$$Y' = Y + K2*T$$

SOLUTION B: JA-2

$$T = 1 + K*(K1*X + K2*Y)$$
$$X' = (X - ANG*Y)*T$$
$$Y' = (ANG*X + Y)*T$$

SOLUTION C: JA-3

$$T = X^2 + Y^2$$
$$X' = X + K1*T$$
$$Y' = Y + K2*T$$

SOLUTION D: JA-4

$$T = (X + dX*SCALE)^2 + (Y + dY*SCALE)^2$$
$$dX = K1*T$$
$$dY = K2*T$$
$$X' = X + dX$$
$$Y' = Y + dY$$

SOLUTION D': JA-5

$$T = (X + dX/2)^2 + (Y + dY/2)^2$$
$$dX = K1*T$$
$$dY = K2*T$$
$$X' = X + dX$$
$$Y' = Y + dY$$

Appendix JA lists the FORTRAN code which implement these five solutions. The mathematical complexity of these five solutions is given in Table JA-1.

TABLE JA-1. MATH OPERATIONS FOR EACH SOLUTION

SOLUTION	MULTIPLIES	ADDS	SHIFTS
A	8	4	0
B	7	4	0
C	4	3	0
D	6	5	0
D'	4	5	2

Table JA-2 tabulates the root mean square error (RMS) and the maximum error (MAX) for solutions A,B, and C versus 11 different lines. The RMS error is calculated by the square root of average of the sum of the errors squared along both axes. MAX is the distance from the exact point to the computed point.

The 11 lines are parallel to the X axis, and they have a distance from the origin (DIST) which ranges from 0 to 1024. Each line has a length of 1024 units. The case where DIST equals 1024 is the case when the REAL SCAN eye orientation is along the horizontal. The case when DIST equals 1, is the worst case, and is when the REAL SCAN nadir falls as close as possible (with integers) to the lower bound line.

TABLE JA-2. RMS ERROR FOR SOLUTIONS A,B, AND C VERSUS DIST
ANGLINE = 0 DEGREES

DIST	RMSA	AMAX	RMSB	BMAX	RMSC	CMAX
1	0.863E-01	0.207E+03	0.202E+00	0.435E+03	0.104E+00	0.183E+03
2	0.297E-01	0.655E+02	0.198E+00	0.366E+03	0.439E-01	0.387E+02
4	0.965E-02	0.175E+02	0.179E+00	0.270E+03	0.269E-01	0.152E+02
8	0.317E-02	0.438E+01	0.140E+00	0.168E+03	0.179E-01	0.683E+01
16	0.104E-02	0.106E+01	0.941E-01	0.860E+02	0.123E-01	0.322E+01
32	0.329E-03	0.240E+00	0.553E-01	0.373E+02	0.859E-02	0.155E+01
64	0.107E-03	0.545E-01	0.286E-01	0.138E+02	0.611E-02	0.745E+00
128	0.228E-04	0.817E-02	0.127E-01	0.428E+01	0.436E-02	0.356E+00
256	0.101E-04	0.204E-02	0.455E-02	0.103E+01	0.283E-02	0.174E+00
512	0.524E-05	0.667E-03	0.112E-02	0.171E+00	0.141E-02	0.930E-01
1024	0.511E-05	0.583E-03	0.331E-03	0.259E-01	0.491E-03	0.353E-01

Table JA-2 indicates that solution A is the most accurate and solution B is the least accurate, of these three solutions..

Table JA-3 tabulates the RMS error and the MAX error for solutions A, B, and C versus 10 lines, which have different angles from the X axis (ANG). Each line has a length of 1024 units and a distance from the origin (DIST) equal to 1. The angle from the X axis ranges from 0 to 45 degrees, which is sufficient to cover a line at any arbitrary angle.

TABLE JA-3. RMS ERROR FOR SOLUTIONS A,B, AND C
VERSUS ANG
DIST = 1

ANG	RMSA	AMAX	RMSB	BMAX	RMSC	CMAX
0	0.863E-01	0.207E+03	0.202E+00	0.435E+03	0.104E+00	0.183E+03
5	0.857E-01	0.205E+03	0.201E+00	0.433E+03	0.105E+00	0.185E+03
10	0.844E-01	0.202E+03	0.201E+00	0.433E+03	0.105E+00	0.186E+03
15	0.872E-01	0.209E+03	0.201E+00	0.433E+03	0.102E+00	0.176E+03
20	0.858E-01	0.206E+03	0.201E+00	0.433E+03	0.105E+00	0.184E+03
25	0.834E-01	0.200E+03	0.201E+00	0.433E+03	0.111E+00	0.204E+03
30	0.856E-01	0.205E+03	0.201E+00	0.433E+03	0.103E+00	0.178E+03
35	0.889E-01	0.213E+03	0.201E+00	0.433E+03	0.103E+00	0.179E+03
40	0.866E-01	0.208E+03	0.201E+00	0.433E+03	0.113E+00	0.208E+03
45	0.865E-01	0.207E+03	0.201E+00	0.433E+03	0.107E+00	0.191E+03

Table JA-3 indicates that the RMS and MAX error are not strongly dependent upon the lower bound angle (ANG). The worst case for solution A is approximatly 15 degrees, and for solution C is approximatly 40 degrees.

Another important feature of these solutions is the error perpendicular to the lower bound line. The lower bound line defines visibility for REAL SCAN, thus the perpendicular error will effect the visible scene. Table JA-4 tabulates the average perpendicular error (PER) and the maximum perpendicular error (PMAX) for solutions A, B, and C versus DIST.

TABLE JA-4. PERPENDICULAR ERROR FOR SOLUTIONS A,B, AND C VERSUS DIST
ANGLINE = 45 DEGREES

DIST	APER	APMAX	BPER	BPMAX	CPER	CPMAX
1	0.344E-04	-.458E-04	0.227E-01	-.848E+00	0.228E-04	-.305E-04
2	0.107E-04	0.305E-04	0.470E-01	-.143E+01	0.340E-05	0.610E-04
4	0.345E-04	0.534E-04	0.602E-01	-.211E+01	0.285E-05	0.610E-04
8	0.676E-05	-.458E-04	0.935E-01	-.262E+01	0.340E-05	0.381E-04
16	0.547E-04	-.944E-04	0.136E+00	-.269E+01	0.400E-04	-.107E-03
32	0.979E-04	0.126E-03	0.180E+00	-.233E+01	0.124E-03	0.198E-03
64	0.122E-03	-.183E-03	0.210E+00	-.172E+01	0.715E-04	-.206E-03
128	0.181E-03	0.290E-03	0.201E+00	-.104E+01	0.391E-04	0.198E-03
256	0.169E-03	0.366E-03	0.135E+00	-.462E+00	0.324E-03	0.641E-03
512	0.203E-03	0.427E-03	0.524E-01	-.132E+00	0.148E-03	-.305E-03
1024	0.307E-03	0.488E-03	0.112E-01	-.256E-01	0.145E-03	0.854E-03

From Table JA-4 solutions A and C have similiar perpendicular errors, and solution B has the worst error. Note that the perpendicular error increases as the distance from the origin increases. This indicates that the angular error is close to a constant. Thus from table JA-4 the visibility error is nearly constant.

Table JA-5 lists the perpendicular error for solutions A, B, and C for versus ANG, with DIST equal to 1.

TABLE JA-5. PERPENDICLUAR ERROR FOR SOLUTIONS A,B, AND C VERSUS ANG
DIST = 1

ANG	APER	APMAX	BPER	BPMAX	CPER	CPMAX
0	0.000E+00	0.000E+00	0.227E-01	-.849E+00	0.000E+00	0.000E+00
5	0.306E-05	-.381E-05	0.227E-01	-.848E+00	0.119E-05	0.381E-05
10	0.141E-04	0.154E-04	0.227E-01	-.849E+00	0.292E-05	-.763E-05
15	0.355E-05	-.954E-05	0.226E-01	-.849E+00	0.644E-06	-.763E-05
20	0.179E-05	0.763E-05	0.227E-01	-.848E+00	0.100E-05	0.153E-04
25	0.814E-05	0.114E-04	0.227E-01	-.848E+00	0.580E-05	0.305E-04
30	0.382E-05	0.153E-04	0.227E-01	-.848E+00	0.507E-05	-.305E-04
35	0.259E-04	-.305E-04	0.227E-01	-.848E+00	0.497E-05	-.153E-04
40	0.104E-04	0.229E-04	0.226E-01	-.848E+00	0.403E-04	0.610E-04
45	0.344E-04	-.458E-04	0.227E-01	-.848E+00	0.228E-04	-.305E-04

Table JA-5 indicates that the perpendicular error is a function of the lower bound angle. The general trend is that the perpendicular error increases with increasing angles up to 45 degrees.

The final conclusion of these three solutions is that solution A is the most complex, and accurate. Solution C is the least complex and more accurate than solution B. Thus solution C is the optimum solution of the three solutions compared above.

Solution D and D' are potentialy more accurate than solution C. The rest of this appendix will compare the accuracy of solutions D, C, and D'

Table JA-6 tabulates the results of a search for SCALE (within 1/1000), which yields a minimum RMS error for solution D versus DIST.

TABLE JA-6. OPTIMUM VALUE OF SCALE VERSUS DIST
ANGLINE = 0 DEGREES

DIST	SCALE	RMSD	DMAX
1	0.174	0.307E-01	0.543E+02
2	0.234	0.170E-01	0.329E+02
4	0.328	0.852E-02	0.161E+02
8	0.403	0.361E-02	0.562E+01
16	0.456	0.134E-02	0.161E+01
32	0.485	0.455E-03	0.397E+00
64	0.489	0.143E-03	0.952E-01
128	0.497	0.304E-04	0.136E-01
256	0.499	0.778E-05	0.222E-02
512	0.501	0.305E-05	-.423E-03
1024	0.504	0.368E-05	-.549E-03

From table JA-6 SCALE ranges from .174 to .504 (approximatly .5). Tables JA-6 and JA-2 indicate that solution D is the most accurate solution!

Table JA-7 tabulates the RMS error for solution D as a function of SCALE approximations. Scale is approximated by the closet binary equivalent, with a given number of bits. In table JA-7 the number of bits vary from 1 to 6.

TABLE JA-7. RMS ERROR FOR SOLUTION D
VERSUS SCALE APPROXIMATIONS
ANGLINE = 0 DEGREES

	NUMBER OF BITS FOR SCALE					
DIST	1	2	3	4	5	6
1	0.104E+00	0.398E-01	0.355E-01	0.312E-01	0.312E-01	0.308E-01
2	0.439E-01	0.172E-01	0.172E-01	0.172E-01	0.172E-01	0.171E-01
4	0.154E-01	0.102E-01	0.928E-02	0.859E-02	0.859E-02	0.854E-02
8	0.531E-02	0.531E-02	0.386E-02	0.386E-02	0.363E-02	0.363E-02
16	0.180E-02	0.180E-02	0.180E-02	0.144E-02	0.139E-02	0.135E-02
32	0.579E-03	0.579E-03	0.579E-03	0.579E-03	0.579E-03	0.462E-03
64	0.173E-03	0.173E-03	0.173E-03	0.173E-03	0.173E-03	0.167E-03
128	0.372E-04	0.372E-04	0.372E-04	0.372E-04	0.372E-04	0.372E-04
256	0.117E-04	0.117E-04	0.117E-04	0.117E-04	0.117E-04	0.117E-04
512	0.551E-05	0.551E-05	0.551E-05	0.551E-05	0.551E-05	0.551E-05
1024	0.552E-05	0.552E-05	0.552E-05	0.552E-05	0.552E-05	0.552E-05

Thus from Table JA-7 a safe number of bits to approximate scale is two. Table JA-8 compares the RMS error and the MAX error of solutions D, C, and D'.

TABLE JA-8. RMS ERROR FOR SOLUTIONS D,C, AND D'
VERSUS DIST
ANGLINE = 0 DEGREES

DIST	RMSD	DMAX	RMSC	CMAX	RMSD.5	D.5MAX
1	0.307E-01	0.543E+02	0.104E+00	0.183E+03	0.871E-01	0.217E+03
2	0.170E-01	0.329E+02	0.439E-01	0.387E+02	0.406E-01	0.900E+02
4	0.852E-02	0.161E+02	0.268E-01	0.152E+02	0.154E-01	0.279E+02
8	0.361E-02	0.562E+01	0.179E-01	0.683E+01	0.531E-02	0.738E+01
16	0.134E-02	0.161E+01	0.123E-01	0.322E+01	0.180E-02	0.183E+01
32	0.455E-03	0.397E+00	0.859E-02	0.155E+01	0.579E-03	0.422E+00
64	0.143E-03	0.852E-01	0.611E-02	0.745E+00	0.173E-03	0.887E-01
128	0.304E-04	0.136E-01	0.436E-02	0.356E+00	0.372E-04	0.133E-01
256	0.778E-05	0.222E-02	0.283E-02	0.174E+00	0.117E-04	0.250E-02
512	0.305E-05	0.423E-03	0.141E-02	0.930E-01	0.551E-05	0.698E-03
1024	0.368E-05	0.549E-03	0.491E-03	0.353E-01	0.552E-05	0.613E-03

Table JA-8 indicates that solution D yields the least RMS error, and solution C yields the most RMS error of these three solutions. Table JA-9 tabulates the perpendicular error for solutions D, C, and D' versus DIST, whit ANG equal to 45 degrees.

TABLE JA-9. PERPENDICULAR ERROR FOR SOLUTIONS D,C, AND D' VERSUS DIST
ANGLINE = 45 DEGREES

DIST	DPER	DPMAX	CPER	CPMAX	D.5PER	D.5PMAX
1	0.197E-04	-.305E-04	0.228E-04	-.305E-04	0.163E-04	-.229E-04
2	0.269E-04	-.610E-04	0.340E-05	0.610E-04	0.132E-05	-.153E-04
4	0.238E-05	-.381E-04	0.285E-05	0.610E-04	0.137E-04	-.381E-04
8	0.605E-04	0.916E-04	0.340E-05	0.381E-04	0.672E-05	0.107E-03
16	0.315E-04	-.496E-04	0.400E-04	-.107E-03	0.279E-04	-.687E-04
32	0.856E-04	-.168E-03	0.124E-03	0.198E-03	0.143E-03	0.183E-03
64	0.299E-04	-.916E-04	0.715E-04	-.206E-03	0.384E-04	-.122E-03
128	0.213E-03	0.427E-03	0.391E-04	0.198E-03	0.286E-03	0.412E-03
256	0.646E-04	0.214E-03	0.324E-03	0.641E-03	0.190E-03	0.397E-03
512	0.142E-03	0.305E-03	0.148E-03	-.305E-03	0.191E-03	0.427E-03
1024	0.235E-03	0.366E-03	0.145E-03	0.854E-03	0.327E-03	0.488E-03

Table JA-9 indicates that solutions D, C, and D' yield similiar perpendicular errors. One would expect solution D to be more accurate than solution C or D'. The reason why solution D is not more accurate in Table JA-9 is that SCALE was selected for to minimize the RMS error not the perpendicular error.

Table JA-10 tabulates the perpendicular error for solutions D, C, and D' versus ANG with DIST equal to 1024.

TABLE JA-10. PERPENDICULAR ERROR FOR SOLUTIONS D,C, AND D' VERSUS ANG
DIST = 1024 SCALE = .504

ANG	DPER	DPMAX	CPER	CPMAX	D.5PER	D.5PMAX
0	0.000E+00	0.000E+00	0.147E-01	-.209E-01	0.000E+00	0.000E+00
5	0.192E-03	-.153E-02	0.100E-01	-.240E-01	0.351E-03	-.110E-02
10	0.132E-02	-.281E-02	0.117E-01	-.249E-01	0.147E-02	-.299E-02
15	0.114E-02	0.208E-02	0.104E-01	-.230E-01	0.478E-03	0.110E-02
20	0.521E-03	-.110E-02	0.103E-01	-.231E-01	0.389E-03	-.732E-03
25	0.436E-03	-.110E-02	0.113E-01	-.237E-01	0.564E-03	-.128E-02
30	0.376E-03	0.732E-03	0.108E-01	-.240E-01	0.213E-03	-.732E-03
35	0.568E-03	-.134E-02	0.948E-02	-.222E-01	0.586E-03	0.110E-02
40	0.674E-03	0.146E-02	0.111E-01	-.243E-01	0.613E-03	0.122E-02
45	0.307E-03	0.488E-03	0.112E-01	-.256E-01	0.145E-03	0.854E-03

As expected Table JA-10 indicates that solution D is the most accurate and solution C is the least accurate.

Thus the final conclusion of this appendix is that solution D is the most accurate given the appropriate scale. And that solution D' is more accurate than solution C. Table JA-1 indicates that solution D requires 2 more multiplies and adds than solution C, and solution D' only requires 2 more shifts and adds than solution C. Thus the optimum solutions are solutions C and D'. C because it is simplest and D' because it is generally more accurate. However, the projection accuracy, from SECTION III of this report, is three fractional bits. Thus, if the maximum perpendicular error is less than .125 for a solution, then that solution meets the accuracy requirements. Thus the final recomedation of this appendix is that solution C is accurate enough and therefore C is the optimum solution.

The value of scale used for solution D in Tables JA-8, JA-9, and JA-10 is the appropriate value found in Table JA-2. The FORTRAN code which created the data for Tables JA-2 through JA-10 are listed in appendices JC through JH.

APPENDIX JB

LOWER.FOR

```
C THIS ROUTINE COMPARES THE SOLUTIONS OF APPENDIX J
C FOR ACCURACY.  SEE APPENDIX J OF THE 1982 REAL SCAN
C REPORT FOR A MATH DESCRIPTION OF THE SOLUTIONS.

      SUBROUTINE LOWER
      COMMON/LOWER/ANGLINE,DIST,SCALE
      COMMON/RMS/RMSA,RMSB,RMSC,RMSD
      COMMON/PER/APER,BPER,CPER,DPER
      COMMON/RMSMAX/AMAX,BMAX,CMAX,DMAX
      COMMON/PERMAX/APMAX,BPMAX,CPMAX,DPMAX
      REAL*8 EX,EY,TERME,XR,YR,THETA,ANGLE

C COMPUTE THE START AND END POINT OF THE LINE
      ANGLE = 3.14159*ANGLINE/180.
      C = DCOS(ANGLE)
      S = DSIN(ANGLE)
      XO = -S*DIST
      YO = C*DIST
      X  = C*512
      Y  = S*512
      XS = XO + X
      XE = XO - X
      YS = YO + Y
      YE = YO - Y

C COMPUTE ANGLES TO START AND END POINTS ANGS AND ANGE
      ANGS = ATAN2(YS,XS)
      IF(ANGS.LT.0) ANGS=ANGS+6.28318
      ANGE = ATAN2(YE,XE)
      IF(ANGE.LT.0) ANGE=ANGE+6.28318
```

```
C COMPUTE ANG FOR RNUMBER ITTERATIONS
      RNUMBER = 3000
      ANG = (ANGE - ANGS)
      IF(ABS(ANG).GT.3.14159) ANG=ANG-6.28318
      ANG=ANG/RNUMBER

C COMPUTE CONSTANTS FOR THE SOLUTIONS
      RK1 = XE - XS
      RK2 = YE - YS
      RK  = RK2*XS - RK1*YS
      ANGK = ANG/RK
      RKX= RK1*ANGK
      RKY = RK2*ANGK

C INITIALIZE ALL POINTS TO XS,YS
      THETA = 0
      AXOLD = XS
      AYOLD = YS
      BXOLD = XS
      BYOLD = YS
      CXOLD = XS
      CYOLD = YS
      TERMD = (XS**2 + YS**2)
      DOX = RKX*TERMD
      DOY = RKY*TERMD
      DX = XS
      DY = YS
      EXOLD = XS
      EYOLD = YS
      AVGA = 0
      AVGB = 0
      AVGC = 0
      AVGD = 0
      C = DCOS(-ANGLE)
      S = DSIN(-ANGLE)
      APER = 0
      BPER = 0
      CPER = 0
      DPER = 0
      AMAX = 0
      BMAX = 0
      CMAX = 0
      DMAX = 0
      APMAX = 0
```

```
      BPMAX = 0
      CPMAX = 0
      DPMAX = 0
C COMPUTE SOLUTIONS (AX,AY), (BX,BY), (CX,CY), (DX,DY),
C AND EXACT (EX,EY)

      NUMBER1 = RNUMBER
      DO 100 ICOUNT = 1,NUMBER1

C SOLUTION A     (AX,AY)
      TERMA =(AXOLD**2 + AYOLD**2)*
     *  (1 + ANGK*(RK1*AXOLD+RK2*AYOLD))
      AX = AXOLD + RKX*TERMA
      AY = AYOLD + RKY*TERMA

C SOLUTION B     (BX,BY)
      TERMB = 1 + ANGK*(RK1*BXOLD + RK2*BYOLD)
      BX = (BXOLD - ANG*BYOLD)*TERMB
      BY = (ANG*BXOLD + BYOLD)*TERMB

C APPROXIMATE SOLUTION (SOLUTION C)
C ie.(DROP ALL ANG**2 TERMS)
      TERMC = (CXOLD**2 + CYOLD**2)
      CX = CXOLD + RKX*TERMC
      CY = CYOLD + RKY*TERMC

C SOLUTION D (DX,DY)
      TERMD=(DX + DDX*SCALE)**2 + (DY + DDY*SCALE)**2
      DDX = RKX*TERMD
      DDY = RKY*TERMD
      DX = DX + DDX
      DY = DY + DDY

C COMPUTE THE EXACT SOLUTION (EX,EY)
      THETA = THETA + ANG
      XR = DCOS(THETA)*EXOLD - DSIN(THETA)*EYOLD
      YR = DSIN(THETA)*EXOLD + DCOS(THETA)*EYOLD
      TERME = RK/(RK2*XR - RK1*YR)
      EX = XR*TERME
      EY = YR*TERME

C UPDATE OLD POINTS TO NEW POINTS
      AXOLD = AX
      AYOLD = AY
```

```
      BXOLD = BX
      BYOLD = BY
      CXOLD = CX
      CYOLD = CY

C FIND SUM OF (ERROR)**2

      AERR = (AX - EX)**2 + (AY - EY)**2
      AVGA = AVGA + AERR
      BERR = (BX - EX)**2 + (BY - EY)**2
      AVGB = AVGB + BERR
      CERR = (CX - EX)**2 + (CY - EY)**2
      AVGC = AVGC + CERR
      DERR = (DX - EX)**2 + (DY - EY)**2
      AVGD = AVGD + DERR

C FIND MAX ERROR DISTANCE FROM THE EXACT POINT
      IF(AERR. GT. AMAX) AMAX = AERR
      IF(BERR .GT. BMAX) BMAX = BERR
      IF(CERR .GT. CMAX) CMAX = CERR
      IF(DERR .GT. DMAX) DMAX = DERR

C FIND THE ERROR IN THE DIRECTION WHICH IS PERPENDICULAR
C TO THE LOWER BOUND LINE.  THIS IS ACCOMPLISHED BY
C ROTATING THE THE SOLUTIONS ERROR BY -ANGS.  THE
C PERPENDICULAR ERROR WILL BE THE Y COMPONENT OF
C THE NEW POINT ROTATED BY -ANG MINUS THE Y INTERCEPT.

      AA = S*AX + C*AY - DIST
      BB = S*BX + C*BY - DIST
      CC = S*CX + C*CY - DIST
      DD = S*DX + C*DY - DIST

C ACCUMULATE THE PERPENDICULAR ERROR
      APER = APER + ABS(AA)
      BPER = BPER + ABS(BB)
      CPER = CPER + ABS(CC)
      DPER = DPER + ABS(DD)

C FIND MAXIMUM PERPENDICULAR ERROR
      IF(ABS(AA).GT.ABS(APMAX)) APMAX = AA
      IF(ABS(BB).GT.ABS(BPMAX)) BPMAX = BB
      IF(ABS(CC).GT.ABS(CPMAX)) CPMAX = CC
```

```
      IF(ABS(DD).GT.ABS(DPMAX)) DPMAX = DD
100   CONTINUE

C FIND RMS OF ERROR
                RMSA = SQRT(AVGA)/RNUMBER
                RMSB = SQRT(AVGB)/RNUMBER
                RMSC = SQRT(AVGC)/RNUMBER
                RMSD = SQRT(AVGD)/RNUMBER

C AVERAGE THE PERPENDICULAR ERRORS
                APER = APER/RNUMBER
                BPER = BPER/RNUMBER
                CPER = CPER/RNUMBER
                DPER = DPER/RNUMBER

C TAKE THE SQUARE ROOT OF THE ERROR

      AMAX = SQRT(AMAX)
      BMAX = SQRT(BMAX)
      CMAX = SQRT(CMAX)
      DMAX = SQRT(DMAX)

      RETURN
      END
```

APPENDIX JC

LOWDIST.FOR

```
C LOWDIST.FOR   2/20/82
C THIS ROUTINE TABULATES THE DATA FOR SOLUTIONS A,B, AND C.
C THIS DATA IS THE AVERAGE RMS ERROR, THE MAXIMUM RMS
C ERROR, THE AVERAGE PERPENDICULAR ERROR, AND THE MAXIMUM
C PERPENDICULAR ERROR VERSUS THE LINE INTERCEPT DIST.

      COMMON/LOWER/ANGLINE,DIST,SCALE
      COMMON/RMS/RMSA,RMSB,RMSC,RMSD
      COMMON/PER/APER,BPER,CPER,DPER
      COMMON/RMSMAX/AMAX,BMAX,CMAX,DMAX
      COMMON/PERMAX/APMAX,BPMAX,CPMAX,DPMAX

      TYPE*,'INPUT ANGLINE'
      ACCEPT*,ANGLINE
      WRITE(15,*) 'ANGLINE =', ANGLINE
      TYPE*,'INPUT TYPE: RMS (0) PER (1)'
      ACCEPT*,ITYPE
      IF(ITYPE .EQ.0) THEN
      WRITE(15,*),'DIST  RMSA AMAX  RMSB BMAX  RMSC CMAX '
      ELSE
      WRITE(15,*),'DIST  APER APMAX BPER BPMAX CPER CPMAX'
      ENDIF


      DO 500 J = 1,11
      DIST = 2**(J-1)
      IY = DIST
      CALL LOWER

      IF(ITYPE .EQ.0) THEN
      WRITE(15,1000) IY,RMSA,AMAX,RMSB,BMAX,RMSC,CMAX
```

```
      ELSE
      WRITE(15,1000) IY,APER,APMAX,BPER,BPMAX,CPER,CPMAX
      ENDIF
500   CONTINUE


1000  FORMAT(' ',I4,1X,E9.3,1X,E9.3,1X,E9.3,1X,E9.3,
     *  1X,E9.3,1X,E9.3)
      END
```

APPENDIX JD

LOWANG.FOR

```
C LOWANG.FOR    2/20/82
C THIS ROUTINE FINDS THE RMS/PERPENDICULAR ERROR FOR
C SOLUTIONSC OF APPENDIX J. VS. ANY LINE ROTATED 10
C TIMES AT AN ANGLE OF 5 C 5 DEGREES PER ROTATION.
C THUS THE LINES RANGE FROM FROM 0 TOC 45 DEGREES.

      COMMON/LOWER/ANGLINE,DIST,SCALE
      COMMON/RMS/RMSA,RMSB,RMSC,RMSD
      COMMON/PER/APER,BPER,CPER,DPER
      COMMON/RMSMAX/AMAX,BMAX,CMAX,DMAX
      COMMON/PERMAX/APMAX,BPMAX,CPMAX,DPMAX

      TYPE*,'WHICH TYPE OF ERROR DO YOU WANT RMS=0, PER=1'
      ACCEPT*,ITYPE
      TYPE*,' INPUT THE DISTANCE TO THE LINE '
      ACCEPT*,DIST
      WRITE(15,*) 'DIST = ',DIST
      IF(ITYPE .EQ.0) THEN
      WRITE(15,*),'ANGLE  RMSA AMAX  RMSB BMAX  RMSC CMAX '
      ELSE
      WRITE(15,*),'ANGLE  APER APMAX BPER BPMAX CPER CPMAX'
      ENDIF
      TYPE*,'INPUT RANGE OF ANGLES FOR IANG'
      ACCEPT*,IRANGE
      INC = IRANGE/9

      DO 100 IANG = 0,IRANGE,INC
      ANGLINE = IANG
      CALL LOWER

      IF(ITYPE .EQ.0) THEN
```

```
      WRITE(15,1000) IANG,RMSA,AMAX,RMSB,BMAX,RMSC,CMAX
      ELSE
      WRITE(15,1000) IANG,APER,APMAX,BPER,BPMAX,CPER,CPMAX
      ENDIF
100   CONTINUE
1000  FORMAT(' ',I3,1X,E9.3,1X,E9.3,1X,E9.3,1X,E9.3,1X,E9.3,1X,
     *E9.3,1X,E9.3)

      END
```

APPENDIX JE

LOWDDISI.FOR

```
C LOWDIST.FOR   2/20/82
C THIS ROUTINE TABULATES THE DATA FOR SOLUTIONS A,B, AND C.
C THIS DATA IS THE AVERAGE RMS ERROR, THE MAXIMUM RMS
C ERROR, THE AVERAGE PERPENDICULAR ERROR, AND THE MAXIMUM
C PERPENDICULAR ERROR VERSUS THE LINE INTERCEPT DIST.

      COMMON/LOWER/ANGLINE,DIST,SCALE
      COMMON/RMS/RMSA,RMSB,RMSC,RMSD
      COMMON/PER/APER,BPER,CPER,DPER
      COMMON/RMSMAX/AMAX,BMAX,CMAX,DMAX
      COMMON/PERMAX/APMAX,BPMAX,CPMAX,DPMAX
      DIMENSION VSCALE(11)
      DATA VSCALE/.174,.234,.328,.403,.456,.485,.489,.497,
     *  .499,.501,.504/

      TYPE*,'INPUT ANGLINE'
      ACCEPT*,ANGLINE
      WRITE(15,*) 'ANGLINE =', ANGLINE
      TYPE*,'INPUT TYPE: RMS (0) PER (1)'
      ACCEPT*,ITYPE
      IF(ITYPE .EQ.0) THEN
      WRITE(15,*),'DIST  RMSD DMAX  RMSC CMAX  RMSD.5 D.5MAX '
      ELSE
      WRITE(15,*),'DIST  DPER DPMAX CPER CPMAX D.5PER D.5PMAX'
      ENDIF


      DO 500 J = 1,11
      DIST = 2**(J-1)
      IY = DIST
      SCALE = .5
```

```
      CALL LOWER
      DDRMS = RMSD
      DDMAX = DMAX
      DDPER = DPER
      DDPMAX = DPMAX
      SCALE = VSCALE(J)

      CALL LOWER

      IF(ITYPE .EQ.0) THEN
      WRITE(15,1000) IY,RMSD,DMAX,RMSC,CMAX,DDRMS,DDMAX
      ELSE
      WRITE(15,1000) IY,DPER,DPMAX,CPER,CPMAX,DDPER,DDPMAX
      ENDIF
500   CONTINUE


1000  FORMAT(' ',I4,1X,E9.3,1X,E9.3,1X,E9.3,1X,E9.3,
     *  1X,E9.3,1X,E9.3)
      END
```

APPENDIX JF

LOWDANG.FOR

```
C LOWDANG.FOR   2/20/82
C THIS ROUTINE FINDS THE RMS/PERPENDICULAR ERROR FOR
C SOLUTIONS C AND SOLUTION D AND D', OF APPENDIX J.
C VS. ANY LINE ROTATED 10 TIMES AT AN ANGLE OF 5 A
C CONSTANT NUMBER OF DEGREES PER ROTATION.

      COMMON/LOWER/ANGLINE,DIST,SCALE
      COMMON/RMS/RMSA,RMSB,RMSC,RMSD
      COMMON/PER/APER,BPER,CPER,DPER
      COMMON/RMSMAX/AMAX,BMAX,CMAX,DMAX
      COMMON/PERMAX/APMAX,BPMAX,CPMAX,DPMAX
      DIMENSION VSCALE(11)
      DATA VSCALE/.174,.234,.328,.403,.456,.485,.489,.497,
     *  .499,.501,.504/

      TYPE*,'WHICH TYPE OF ERROR DO YOU WANT RMS=0, PER=1'
      ACCEPT*,ITYPE
      TYPE*,' INPUT THE Y INTERCEPT (WORST CASE = 1)'
      ACCEPT*,DIST
      J = LOG(DIST)/LOG(2.) + 1
      SCALE = VSCALE(J)
      WRITE(15,*) 'DIST = ',DIST,'SCALE =',SCALE
      IF(ITYPE .EQ.0) THEN
      WRITE(15,*),'ANGLE  RMSD DMAX  RMSC CMAX  RMSD.5 D.5MAX '
      ELSE
      WRITE(15,*),'ANGLE  DPER DPMAX CPER CPMAX D.5PER D.5PMAX'
      ENDIF
      TYPE*,'INPUT RANGE OF ANGLES FOR IANG'
      ACCEPT*,IRANGE
      INC = IRANGE/9
```

```
      DO 100 IANG = 0,IRANGE,INC
      ANGLINE = IANG

      SCALE = .5
      CALL LOWER

      RMSOD = RMSD
      DDMAX = DMAX
      DDPER = DPER
      DDPMAX = DPMAX

      SCALE = VSCALE(J)
      CALL LOWER

      IF(ITYPE .EQ.0) THEN
      WRITE(15,1000) IANG,RMSD,DMAX,RMSC,CMAX,RMSOD,DDMAX
      ELSE
      WRITE(15,1000) IANG,DPER,DPMAX,CPER,CPMAX,DDPER,DDPMAX
      ENDIF
100   CONTINUE
1000  FORMAT(' ',I3,1X,E9.3,1X,E9.3,1X,E9.3,1X,E9.3,1X,E9.3,1X,
     *E9.3,1X,E9.3)

      END
```

APPENDIX JG

LOWSCALE.FOR

```
C LOWSCALE.FOR  2/20/82
C THIS ROUTINE COMPUTES THE VALUE OF SCALE WHICH
C MINIMIZES THE RMS ERROR FOR SOLUTION D.  SOLUTION
C D IS THE SAME AS SOLUTION C EXCEPT THAT SOLUTION
C D USES AN APPROXIMATION FOR THE NEW X,Y TO FIND
C THE NEW X,Y.  THIS GUESS IS A VALUE OF SCALE
C WHICH YIELDS THE LEAST RMS ERROR.
C (NOTE  0 < SCALE < 1.)

      COMMON/LOWER/ANGLINE,DIST,SCALE
      COMMON/RMS/RMSA,RMSB,RMSC,RMSD
      COMMON/PER/APER,BPER,CPER,DPER
      COMMON/RMSMAX/AMAX,BMAX,CMAX,DMAX
      COMMON/PERMAX/APMAX,BPMAX,CPMAX,DPMAX
      DIMENSION D(1000)

      WRITE(15,*) '   DIST     SCALE     RMSD     DMAX '

      TYPE*,'INPUT ANGLINE'
      ACCEPT*,ANGLINE
      WRITE(15,*) 'ANGLINE=',ANGLINE
      DO 500 J = 1,11
      DIST = 2**(J-1)

      DMIN = 1000000.
      SAVE = 0
      DELSCALE = 1./1000.
      SCALE = 0

C LOAD D WITH RMS ERRORS FOR 1000 DIFFERENT SCALES
C SINCE SCALE RANGES FROM  0 TO 1EB
```

```
      DO 100 I=1,1000
             SCALE = SCALE + DELSCALE
             CALL LOWER
             D(I) = RMSD
100   CONTINUE

      DO 200 I = 1,1000
      IF(D(I) .LT.DMIN) THEN
             DMIN = D(I)
             SAVE = I
      ENDIF
200   CONTINUE

      SCALE = SAVE/1000.
      CALL LOWER
      WRITE(15,1000) DIST,SCALE,RMSD,DMAX
500   CONTINUE

1000  FORMAT(' ',F5.0,5X,F5.3,2X,E9.3,2X,E9.3,2X,E9.3,2X,E9.3)
      END
```

APPENDIX JH

LOWBITS.FOR

```
C LOWBITS.FOR   2/20/82
C THIS ROUTINE TABULATES DIFFERENT VALUES
C OF SCALE VS. THE RMS ERROR OF SOLUTION D TO
C THE LOWER BOUND PROBLEM.  EACH VALUE OF SCALE
C USED HERE IS AS CLOSE TO THE BEST VALUE OF
C SCALE (FOUND FROM LOWSCALE.FOR) WITH THE
C GIVEN NUMBER OF BITS.
      COMMON/POWER/ANGLINE,DIST,SCALE
      COMMON/RMS/RMSA,RMSB,RMSC,RMSD
      COMMON/PER/APER,BPER,CPER,DPER
      COMMON/RMSMAX/AMAX,BMAX,CMAX,DMAX
      COMMON/PERMAX/APMAX,BPMAX,CPMAX,DPMAX
      DIMENSION VSCALE(11),RMS(5)
      DATA VSCALE/.174,.234,.328,.403,.456,.485,.489,.497,
     *  .499,.501,.504/

      WRITE(15,1000)
      WRITE(15,2000)
      WRITE(15,3000)
      WRITE(15,3500)

      ANGLINE = 0
      DO 500 J = 1,11
      DIST = 2**(J-1)
      IY=DIST

      DO 100 JBITS = 1,6
            RPOWER = 2**(JBITS)
            BITS = RPOWER*VSCALE(J) + .5
            IBITS = BITS
            SCALE = (1.*IBITS)/RPOWER
```

```
                CALL LOWER
                RMS(JBITS) = RMSD
100         CONTINUE
            WRITE(15,4000)IY,RMS
500         CONTINUE
1000        FORMAT(' ',11X,'TABLE JA-4. RMS ERRORS FOR SOLUTION
     * D AS A FUNCTION OF SCALE')
2000        FORMAT(///,15X,'NUMBER OF BITS FOR SCALE')
3000        FORMAT(' ',' DIST    1       2       3       4',
     *      '          5       6')
3500        FORMAT(/,'-------------------------------------------'
     *      ,'----------------------')
4000        FORMAT(' 'I4,1X,E9.3,1X,E9.3,1X,E9.3,1X,E9.3,
     *      1X,E9.3,1X,E9.3)
            END
```

END

FILMED

1-83

DTIC