NPS-ME-96-001

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Prototype Line of Sight and Target Acquisition Software for High Resolution Databases

by

Morris Driels
Judith Lind

December 1995

Approved for public release; distribution is unlimited.

Prepared for: TRAC-Monterey
Monterey, CA 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, California

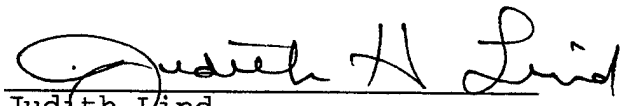
Rear Admiral M.J. Evans
Superintendent

R. Elster
Provost

This report was prepared for TRAC-Monterey, Naval Postgraduate School, Monterey, CA 93943.

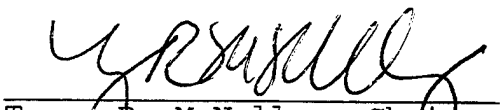
The report was prepared by:

Morris R. Driels
Professor
Department of Mechanical Engineering



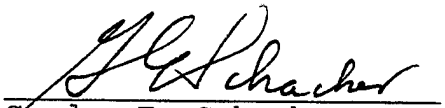
Judith Lind
Adjunct Professor
Department of Operations Research

Reviewed by:



Terry R. McNelley, Chairman
Department of Mechanical Engineering

Released by:



Gordon E. Schacher
Dean of Research

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1995	Technical Report

4. TITLE AND SUBTITLE
Prototype Line of Sight and Target Acqusition Software for High Resolution Databases

5. FUNDING NUMBERS

6. AUTHOR(S)
Morris Driels and Judith Lind

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
NPS-ME-96-001

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
TRAC-Monterey
Monterey, CA 93943

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this report are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report describes the generation of terrain visualizations from a database of 1 meter resolution. It utilizes a ray trace algorithm, but because it is not object based, requires the same time to render a scene irrespective of the number of objects viewed. A menuing system for allowing data input to the program is described. Implementation of the ACQUIRE target detection algorithms are also described, together with some advantages of using a higher resolution database.

14. SUBJECT TERMS
Search, target acquisition, synthetic environments, terrain, visualization.

15. NUMBER OF PAGES
44

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLAS	UNCLAS	UNCLAS	

PROTOTYPE LINE OF SIGHT AND TARGET ACQUISITION SOFTWARE FOR JANUS (A) VERY HIGH RESOLUTION TERRAIN DATABASES

FINAL REPORT FY95

MORRIS DRIELS
PROFESSOR, MECHANICAL ENGINEERING DEPARTMENT

AND

JUDITH LIND
ADJUNCT PROFESSOR, OPERATIONS RESEARCH DEPARTMENT

NAVAL POSTGRADUATE SCHOOL,
MONTEREY, CALIFORNIA 93943.

This report covers the period 10/1/94 to 9/31/95 on Naval Postgraduate School Reimbursable Research Project (Non-Navy) RETDL/RET4L.

TABLE OF CONTENTS

ACKNOWLEDGMENT

The authors wish to thank TRAC Monterey for the financial support of the research reported, and for supplying the Fort Hunter Liggett databases used in the program.

1. OVERVIEW OF PERSPECTIVE VIEW GENERATOR METHODOLOGY

In the creation of synthetic environments (virtual reality scenarios), two approaches are currently taken:

(1) Object based simulations
(2) Database driven simulations

The first of these is the most commonly encountered, and there are numerous examples of such systems. In previous studies by Young, Whitney, commercially available software, World Toolkit, was used to create an empty universe into which objects may be inserted. In this particular example, a flat terrain was generated and tree-like objects and buildings were constructed and inserted. Then a tank was placed in the scene and was moved around. The tank was viewed from a fixed observation point in the scene, and calculations were performed to determine the area of the tank viewable from the observation point, as it passed behind the various obstructions to the line of sight.

Although this was a simple and rapid way of building such scenes, it still has the cartoon-like appearance of object based scenarios, and as the complexity of the scene increased in an attempt to achieve more realism, the execution speed fell dramatically. It became obvious that if true realism is to be achieved, such as a scene with perhaps hundreds of differently shaped trees, another method of simulation will be required if seamless movement around the scene is to be achieved. This means achieving at least 15 image frames per second, irrespective of the complexity of the visual scene.

The database driven simulation provides a possible solution to this problem, and is particularly suited to the representation of terrain. In such a simulation, the terrain is not regarded as a collection of objects, but its physical characteristics are encoded and stored in a data base. Consider a flat terrain divided into, say, elements measuring 1 meter by 1 meter, and suppose that a color (or gray shade) is measured for each square on the terrain. Such data may come from aerial photographs, satellite measurements, or from measurements taken on the ground. The collection of such information, and its correlation with the (x,y) coordinates on the ground constitute the database.

Now consider the problem of observing the terrain. Suppose we wish to generate a view of the terrain and display it in digital form. Furthermore, suppose our generated image is to have a resolution of 256 x 256 screen pixels. Figure 1 shows the possible relationship between the grid of screen pixels and the elements on the database, for an arbitrary placement of the observation point relative to the ground.

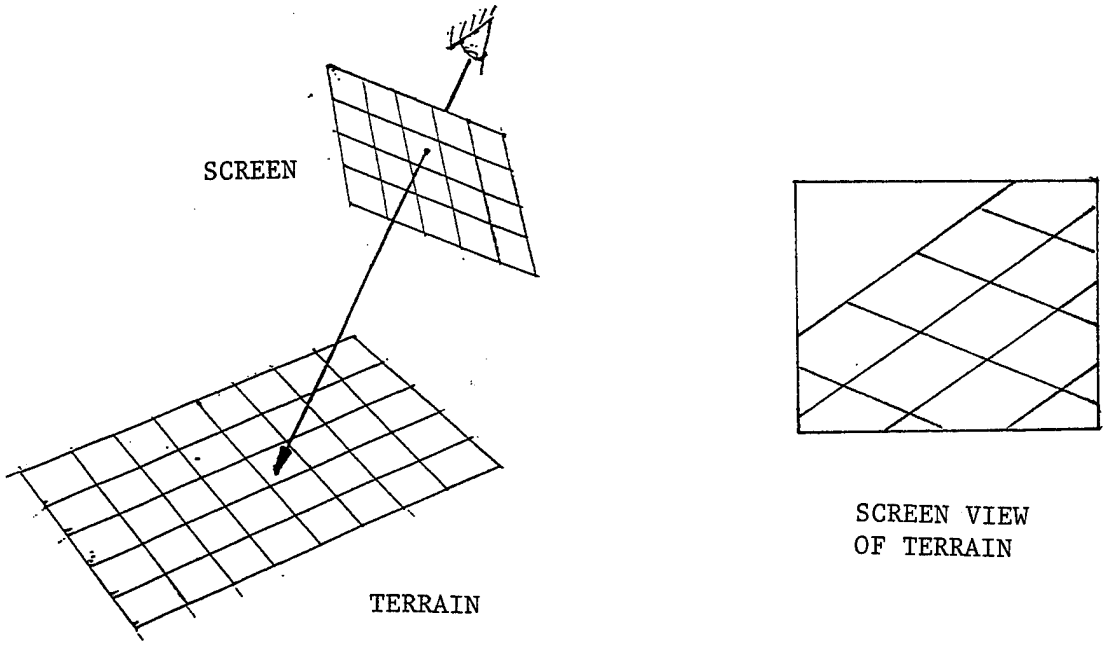


Figure 1 Relationship Between Observation Grid and Terrain

We define the boresight vector as the vector passing through the center of the image, and is specified in terms of the x,y,z coordinates of the screen center, and two direction cosines specifying the orientation of the vector. If we imagine stepping along this vector in one meter steps, we eventually intersect the terrain at some location. If we then determine the gray shade of the element at the intersection, and places that gray shade in the element corresponding to the screen center, we have performed an elementary ray trace operation.

We now introduce the concept of a field of view. If our observation of the terrain is made by a camera, its lens would have a certain

solid cone of observation determined by the lens. It may have a narrow field of view, say 10 degrees, or a wide field of view, perhaps as much as 90 degrees. For convenience, let us assume the field of view is 20 degrees, a common number for common lenses. We can approximate the image plane to a linear distribution of angular increments in both azimuth and elevation, as indicated in figure 2.

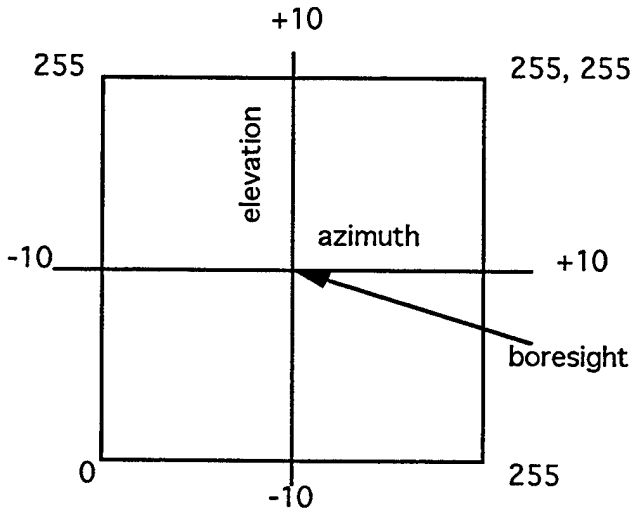


Figure 2 Relationship Between Field of View and Image Plane

In the figure, we assume that the +-10 degree azimuth field of view is distributed over the 256 pixels the image has in the horizontal direction. The same argument applies to the vertical direction. What this means is that given the direction cosines of the boresight vector, the direction cosines of each element in the image may be determined. Hence a ray trace may be performed for each pixel, and the corresponding gray scale at the point of intersection determined. If the collection of gray scale values is displayed, the resulting view will be that of a camera with a 20 degree field of view placed and oriented in accordance with the boresight vector. It is this approach that we define as a perspective view, and the program used to compute it is known as the perspective view generator (PVG).

It is now a simple extension to include the undulating nature of the terrain. In such a case, as we move down a particular vector or ray, it passes over different ground elements. Knowing the start point of the ray, and its direction cosines, for every one meter moved along the ray we are able to determine what are the new x,y,z coordinates of the current location on the vector, and hence which ground element we are over. If the elevation of each ground element is also encoded

into the database, we can test to see if the z coordinate of the current location on the ray is above the elevation of the corresponding ground element. If it is, we have not yet hit the terrain, and a further step along the ray may be taken. The ray trace stops when the ray z coordinate is equal to the ground elevation, and the corresponding gray scale for that element is stored in the image array. The ray trace then continues for the next pixel until the whole screen has been completed and displayed. Note that, if the terrain is perfectly flat, we could use simple geometry to determine ray-ground intersections but, with variable elevations, this is not possible, and a ray trace is the only method available.

Clearly, the problem with this method is the computational time needed to build the image array. For a view image of 256 x 256 pixels where the boresight vector is locates 500 m from the terrain, a total of 256 x 250 x 500 ray trace steps have to be taken. This means that the intersection of a ray with the ground has to be checked about 32 million times in construction 1 image frame. In addition to this is the time needed to extract, store and display the resulting gray shade, although this will only have to be done 256 x 256=65536 times per frame. To achieve frame rates of 20 frames/sec over half a billion ray steps have to be done each second. Although quite daunting, it should be remembered that this computation is independent of the complexity of the scene, and will remain fairly constant whether there is one tree in the view or several thousand. In the case of highly complex or cluttered scenes, the ray trace method proves more realistic than any object based generator, on the same hardware platform.

So far, we have discussed the boresight vector as being stationary, but clearly it may be moved. If it is possible to link the boresight vector with a device capable of generation 5 or 6 degrees of freedom, such as a joystick or spaceball, then current data from such a device may be used to control the boresight vector, hence the image frame. In this way, it is possible to fly through a terrain database and observe the ground characteristics. Apart from the computational time needed to allocate joystick data to the boresight vector, no additional time is needed to generate images from a moving observer compared to a stationary observer. Functionally, a program which generates images from an existing database takes the form shown in figure 3.

Basically the program is an infinite loop in which position and orientation data are obtained from a joystick or other input device, direction cosines for each screen pixel are calculated, and a ray trace is performed for each ray emanating from each pixel. When the ground is hit by the ray, the corresponding gray shade is placed in that pixel's location in a frame buffer. When each frame is completed, a graphics command displays the whole buffer.

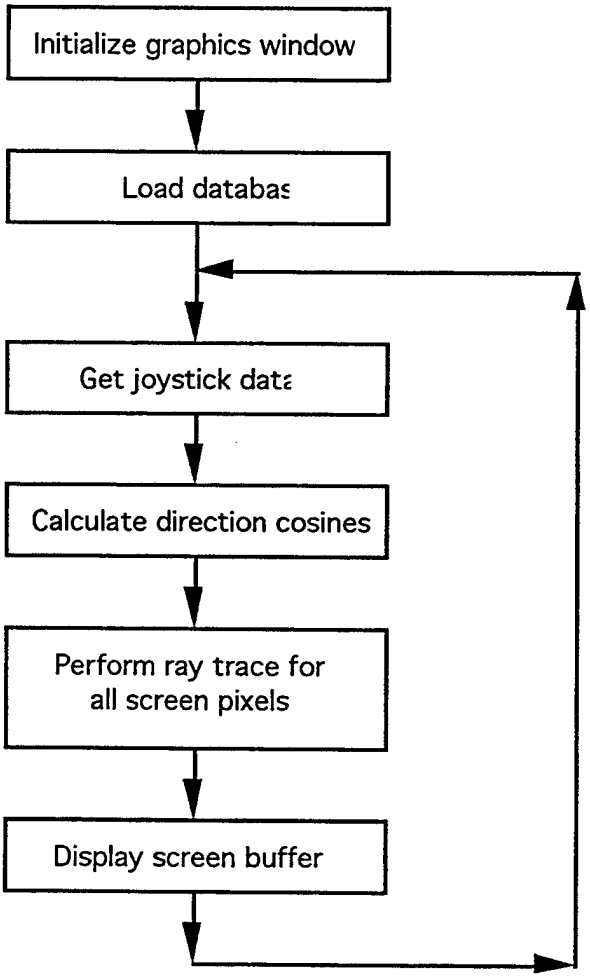


Figure 3 Generalized Flow Chart for PVG Program

Most of the work we describe here is based on an existing database compiled under contract to the US Army by Nascent, Inc. The construction of the database itself is no mean feat, and details of this

process may be found elsewhere (Baer1). The original database was of Fort Hunter Liggett (FHL) in Central California. Pioneering work in this area was originally done by Dr. Wolfgang Baer, and its implementation on specialized architecture computers may be found in the literature (Baer2). Our intent here is to develop algorithms which will execute reasonably fast on commercially available single processor workstations. In addition, features regarding target delectability in such high resolution databases have been incorporated, to demonstrate how existing battlefield simulation tools such as JANUS may be integrated with a high resolution database.

2. IMPLEMENTATION OF PVG ON SINGLE PROCESSOR WORKSTATION

The original perspective view generator (PVG) was implemented on a multi-processor, transputer based computer. In order to make the execution time per frame decrease to a reasonable figure when run on a conventional, single processor workstation, several constraints were placed on the algorithm. These are

(1) Only a limited area of terrain is loaded for the PVG. The original algorithm had dynamic data loading, which means that only part of the database, that being viewed, is maintained in RAM. As the viewpoint changes, part of the database is discarded, while other sections are loaded. This dynamic database management requires some CPU time to execute. In the version reported here, a limited terrain area is loaded into RAM, thereby avoiding the loading/unloading overhead. The area of terrain which may be held in memory depends on the amount of RAM the machine has. The development machine, an SGI R4000-based workstation, has 48 Mbytes of RAM, and may hold an area of about 2km by 2km of data. Clearly, this area may be increased by increasing the amount of RAM. It was felt that this approach was justified, since we are mainly interested in relatively close engagements of ground forces.

(2) The effect of the movement of the sun with time has been ignored. It was felt that this feature, which can require considerable computational load and memory resources, does not significantly add to the detectability of targets.

(3) The variation of database resolution with viewing distance was not included. The FHL database is available at 1 m, 4 m, 16 m and 64 m resolutions, and the transputer based system loaded a database pixel into the frame buffer depending on the viewing range. Since we are dealing with a limited area of terrain, all of it is viewable at the highest resolution, namely 1 m post sizes.

The best way to explain the operation of the program, and its associated algorithms, is to look at each function and describe the task it performs. We will start with the main segment

PVG_MAIN.C

This program is the main program segment, and is responsible for initializing the database in RAM, and cyclically executing the ray trace algorithm. A copy of the include file PVG_DEC.IN is shown, and is used to define most of the global constants used in the program. The first three variables in PVG_DEC.IN allow the size of the database to be set, and the display screen resolution to be set. Note that increasing the size of the terrain database beyond that which may be contained in RAM will cause the overflow to be swapped to disk. As a result program execution will slow down considerably. The layout of PVG_MAIN.C is very similar to that shown in figure 3.

INIT.C

This program initializes some of the global variables used in the program, and loads data from a file cmr.dat which is used in the target acquisition algorithms to be discussed later. The program begins by requesting the user for a value of MAG, the display magnification. A "standard" perspective view window is 256 x 256 pixels on the screen, which physically turns out to be about 1.5 inches square. This may be magnified MAG times. A good value for MAG is 2, since this produces a View window of about 3 inches square, leaving sufficient room on the display for other windows created by the program. Only the perspective view window is affected by MAG.

The vector IFOVNOW contains ten elements. The first six are the initial position and orientation of the boresight vector, and represent a position above the terrain origin (x=y=0), and 700 m above sea level, looking down at the terrain. Element IFOVNOW[6] is the field of view angle in degrees. The first two elements of the vector RAYSEG represent half of the frame buffer in pixels, and they are used in the generation of the direction cosines. TAR_X and TAR_Y give the initial coordinates of the target. Since the terrain measures 1024 m x 1024 m, the target starts in the center of the terrain. The final section of the program deals with the CMR (cycles per milliradian) data table used for the target acquisition function to be described later.

DATAGEN.C

The program uses only one of the 4096 m square tiles which form the FHL database. This is tile number 33. Since the battlefield

measures 1024 x 1024, a window this size may be placed anywhere in the tile. The location of the window is defined by the coordinates of the lower left corner of the battlefield relative to the lower left corner of the tile. The coordinates are expressed in blocks, which represent increments of 256 m, hence each tile is divided up into 16 x 16 blocks. Figure 4 shows how the tile is segmented. Two 1024 m square battlefields are drawn in the lower left and upper right of the tile. To select these two battlefields, the required coordinates would be 0, 0 for the lower left, and 12, 12 for the upper right.

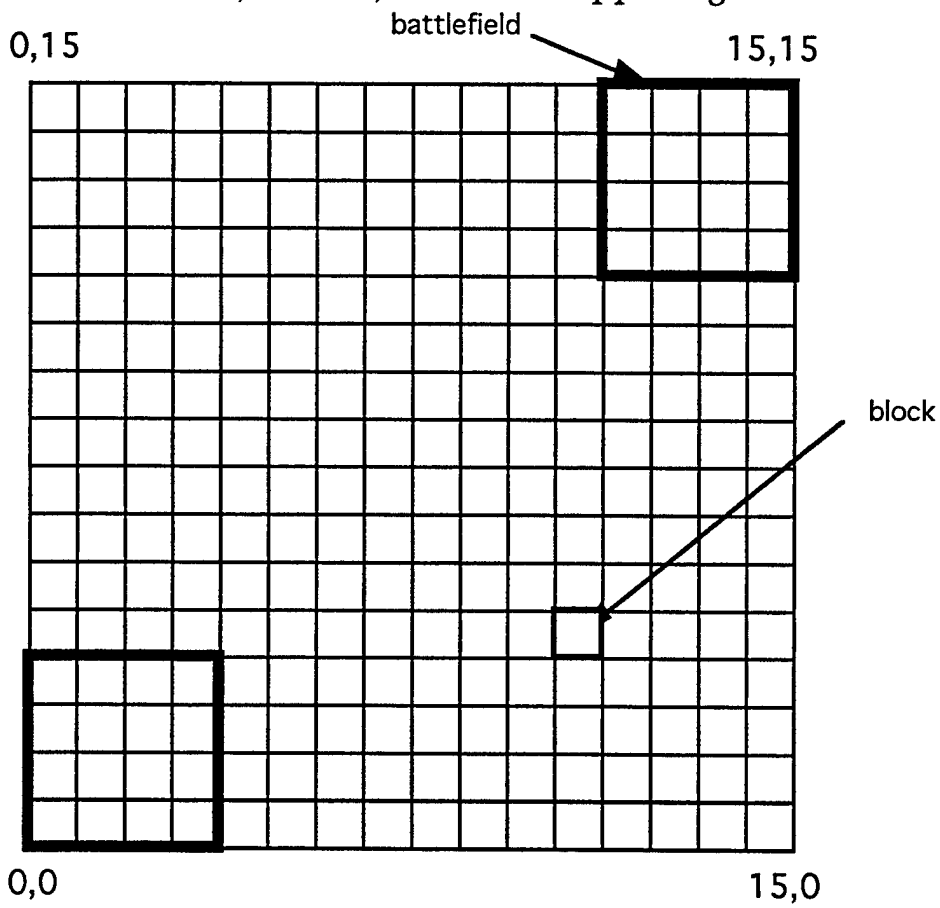


Figure 4 Division of Tiles into Blocks

Program DATAGEN requests the user for the coordinates of the lower left corner of the battlefield; each must be in the range 0 to 12. Another 4 m resolution database is also included with the distributed code, also for tile number 33. The user therefore has the option of viewing the tile at either 1 m or 4 m resolution. DATGEN then calls GET_BLOCK.C. This program loads the data from the appropriate database one block at a time, and fills up the array DAT. The array

DAT holds the 32-bit numbers corresponding to the 1024 x 1024 battlefield. The array, therefore is 4 Mbytes in size. For simplicity, the code for loading from the 1 m database is separate from the code for the 4 m database. Upon exit from DATAGEN, the array DAT has been loaded into memory in preparation for the ray trace.

INIT_WINDOW

This program uses GL library calls to initialize three windows. The first of these is titled "Fort Hunter Liggett" and is intended to show the whole of the FHL database by displaying the 64 m resolution database in a window 512 x 448 pixels in size. The next window is labeled "Battlefield Area" and displays a fixed, downward looking view of the 1024 m x 1024 m area selected in program DATAGEN. This window is fixed in size at 512 x 512 pixels. The third window is titled "Observer" and displays the view obtained from the ray trace program. It is this window which changes as the input device is manipulated, and is of a size determined by the user input variable MAG.

BIGMAP

Program BIGMAP loads the data from the 64 m resolution database, and displays it in the window titled "Fort Hunter Liggett." Figure 5 shows how the FHL database is split into tiles.

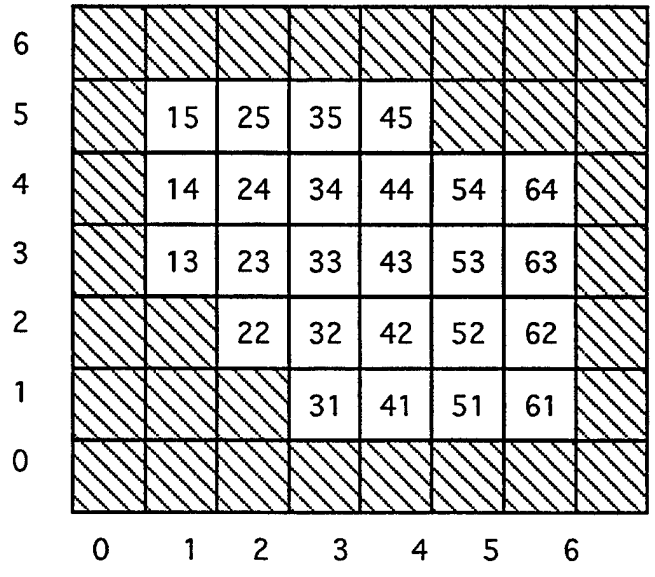


Figure 5 FHL Database Tile Numbering System

```
/*

        Program INIT_WINDOW.C to initialize
        the GL graphics window

*/

#include<stdio.h>
#include"PVG_DEC.IN"
#include<gl/gl.h>

long    MAPWIN_ID, MAINWIN_ID, BIGMAP_ID;

        init_window()
        {

        extern int MAG;

        prefsize(512,448);
        BIGMAP_ID=winopen("Fort Hunter Liggett");
        color(BLACK);
        clear();
        RGBmode();
        gconfig();

        prefsize(512, 512);
        MAPWIN_ID=winopen("Battlefield Area");
        color(WHITE);
        clear();
        RGBmode();
        gconfig();

        prefsize(PVG_HEIGHT*MAG,PVG_WIDTH*MAG);
        MAINWIN_ID=winopen("Observer");
        color(BLACK);
        clear();
        RGBmode();
        gconfig();



        }
```

The complete database comprises 56 tiles arranged in an array 8 tiles wide and 7 tiles high. The shaded tiles have no data available; only the clear tiles in the center have terrain data (Figure 5). BIGMAP loads the 64 m database in a hierarchical manner. It loads by tile, beginning with number 00 and finishing with number 76. To do this it utilizes program GET_TILE_64. Program GET_BLOCK_64 extracts from the database file pvdb.64 a block of 16 x 16 posts. When these two called functions are completed, the 64 m database is held in an array called FHL. Program BIGMAP then puts this two dimensional array into a vector VIEWMAP, which the GL command LRECTWRITE is able to display directly into the window labeled "Fort Hunter Liggett." The result of this is shown in figure 6

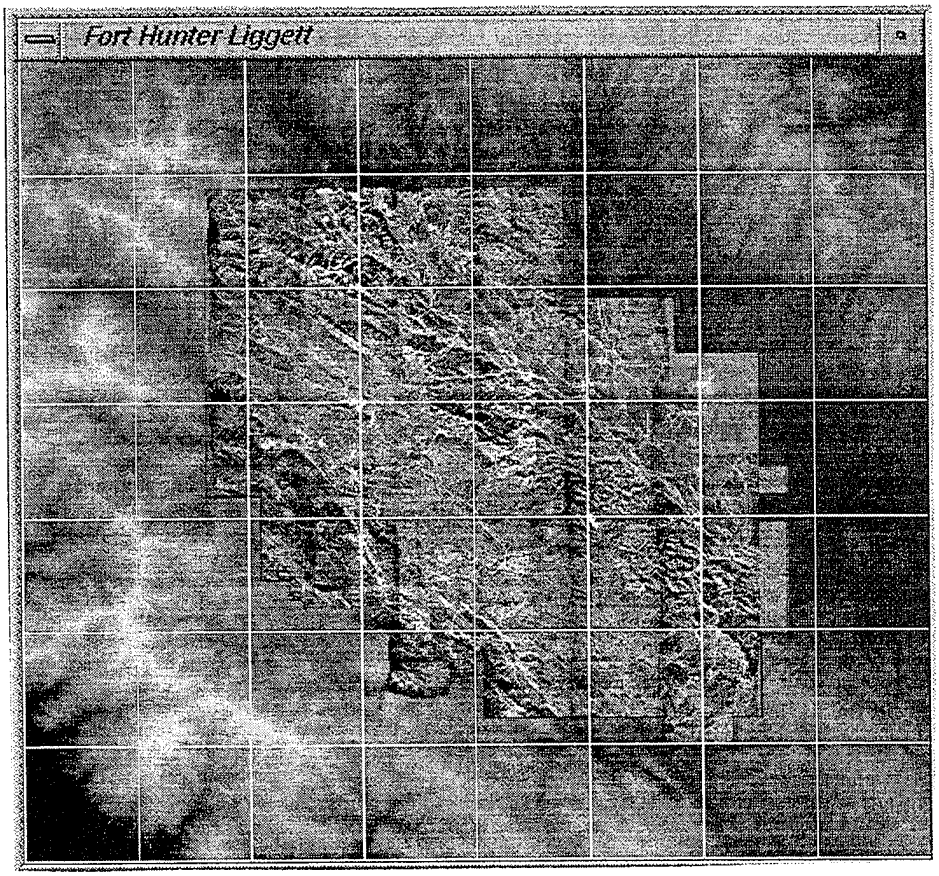


Figure 6 FHL 64m Database

CALL_MAP

This program is similar to BIGMAP, except that it displays the 1024 m x 1024 m battlefield. It accomplishes this by selecting every other pixel from the data array DAT, which was initialized in DATAGEN. It

then writes the resulting array to a linear vector VIEWMAP and again uses the GL command LRECTWRITE to place the contents of VIEWMAP in the "Battlefield Area" window. Figure 7 shows an example of this, as the lower left battlefield in figure 4 selected from tile 33.

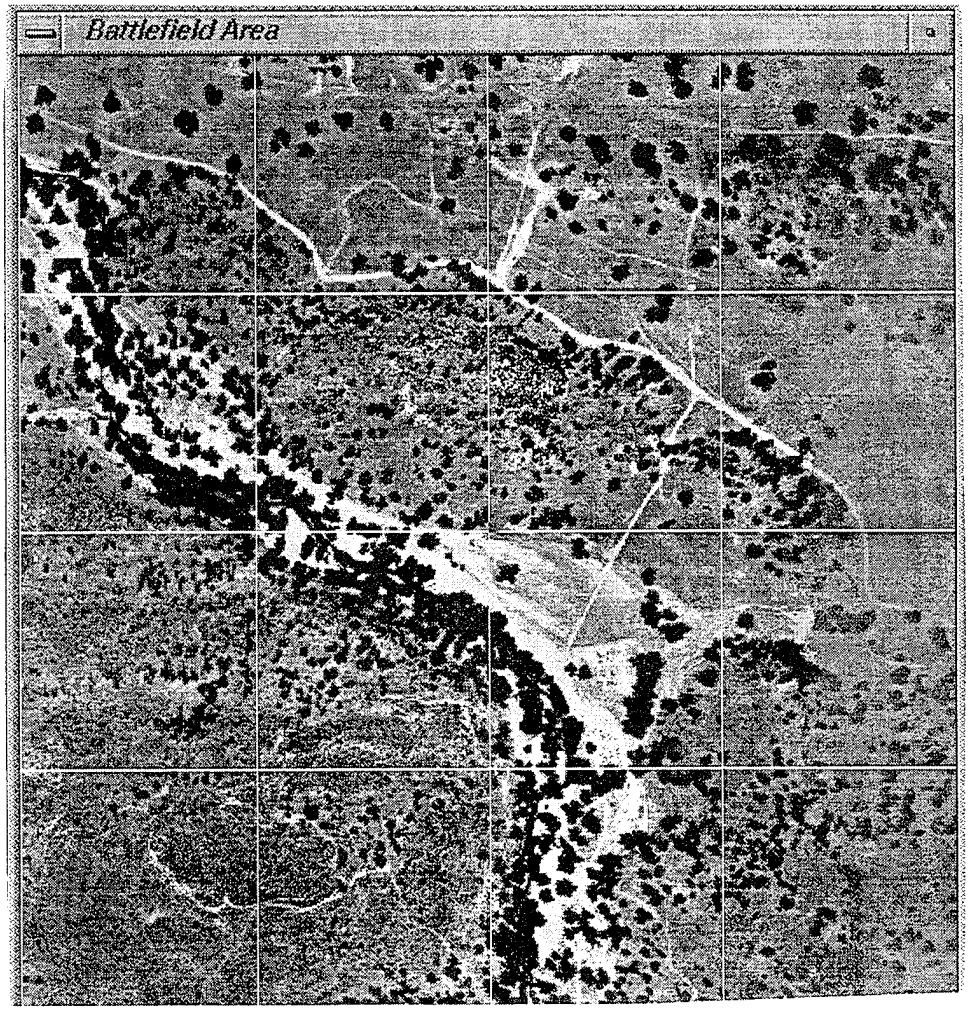


Figure 7 Battlefield Area Window

This completes the initialization part of the program, which now enters an indefinite loop. The following programs are called each time through the loop

TARGET

In order to evaluate the performance of target acquisition algorithms when used with a high resolution database, some type of target has to be inserted into the terrain. This presents a fundamental difficulty in a ray trace type of program, since objects are not defined, but

instead are perceived by the viewer as an expected collection of gray shades. For example, a tree is a collection of darker 1 m x 1 m elements set against a background of lighter shaded grass. As a result, it is difficult to insert artificial objects, such as targets, onto the terrain. The problem is solved by modifying the database in the locality of the target, to include its size. For example, the target is considered to be a regular prism (block) measuring 4 m x 6 m, and standing 4 m high. The variables TAR_X and TAR_Y define the location of the target's left nearside corner on the database. The array DAT is modified to

(1) Increase the terrain elevation within the 6 x 4 footprint by 4 m.

(2) Change the nature bit to indicate a man-made element.

(3) Alter the color of the element to that of the target. In the case of the distributed software, the target is colored red for ease of recognition. Actually, TARGET performs other functions as well as embedding the target into the database. These other functions include

(1) Providing the target with some motion. In the program, the (x,y) coordinates are increased 1 m for each frame. This is an arbitrary motion intended simply to make a moving target viewable by the observer. The target could, for example, be controlled by another operator using a mouse. The programmed motion causes the target to move diagonally across the battlefield from its initial position in the center.

(2) Small squares are written to the Battlefield Area map to indicate the current locations of the target and the observer. This is to assist the operator close in on the target if it is not viewable from the Observer window. The target is colored red, while the observer is green.

SPACEBALL

The spaceball is the input device of preference, although the distribution software has a mouse version. Both devices are supported through GL function calls to control the position and orientation of the viewpoint from which the ray trace algorithm is performed. Essentially, this program reads inputs from the device and loads the first six elements of the IFOVNOW vector. The use of

the spaceball is intuitive, but the MOUSE program source code defines how to control the viewpoint with the mouse and its buttons.

GENDDCOS

This program is identical to that used in the Sun version written by Wolfgang Baer. The purpose of this routine is to calculate the elements of the 2 dimensional array DDCOS[3][3]. These elements are defined as

DDCOS[0][i]= direction cosines of start ray, i.e. the ray through lower left screen pixel.
DDCOS[1][i]= change in direction cosines when a row index changes.
DDCOS[2][i]= change in direction cosines when a column index changes

With this array, it is possible to calculate the direction cosines of a vector from any screen pixel, which is what is needed in the ray trace algorithm. In order to handle the coordinate transformations used in this program, two other functions are used, VEH2UTM_MATRIX.C and MATNMUL.C. These programs are also identical to those used in the original Sun version.

ZIGZAG

Program ZIGZAG is the heart of the PVG system and performs the ray trace operation above the current terrain, for all 256 x 256 screen pixels. Its main inputs are

(1) Array DAT containing the battlefield terrain data.
(2) Vector IFOVNOW defining where the observer is located relative to the terrain.
(3) Array DDCOS containing how the direction cosines vary for incremental row/column screen pixels.

Figure 8 shows how the screen coordinates are defined for each pixel.

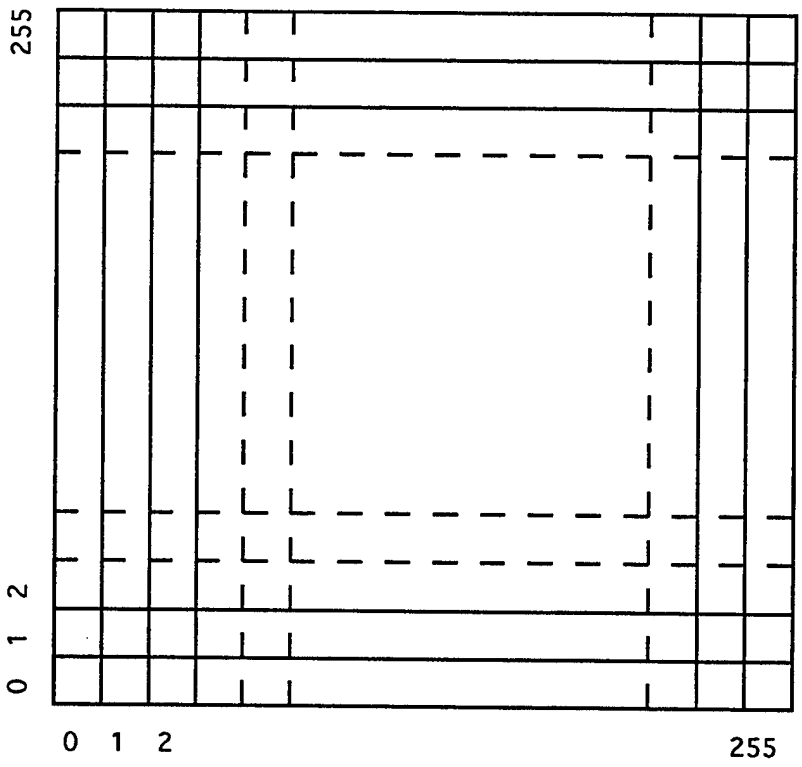


Figure 8 Definition of Pixel Screen Coordinates

The ray trace starts with the lower left pixel (0,0). It is important to recognize that the direction cosines are also the changes in the (x,y,z) coordinates of the start point of the ray trace, for each unit step along the ray. Hence, knowing the direction cosines (DC's) for the first pixel, each meter step along the ray, allows us to determine the (x,y,z) coordinates of that point on the ray. The (x,y) coordinates allow us to determine which element of DAT we are currently above, while the z value may be compared with the elevation of the elements whose (x,y) coordinates we have just found. By comparing these elevations, we can determine whether a ground intersection has occurred. If not, we take another step along the ray and test again. This continues until either we do hit the ground, or the (x,y) coordinates pass outside the battlefield limits.

If we hit the ground, one of several cases is considered. First, we determine if the target has been hit, by examining the nature/man-made bit of the element of DAT. If this is the case, the color red is written into the corresponding screen pixel; if not, function HITGRND is called to determine what color should be given to the pixel. If we pass outside the terrain database before hitting the ground, the pixel is given the color white. This enables us to perceive the "edge" of the

battlefield. The color/gray shade of the pixel is written into the vector VIEW, and it is this vector which will be displayed later.

Once the first screen pixel has been dealt with, pixel (0,1) is treated in the same manner, and so on until the first column has been considered. Then element (1,0) is ray traced, then (1,1) until the second column is completed. This raster proceeds until the right-most column completes the whole screen. Program ZIGZAG, therefore, consists essentially of a double loop in which the column index is incremented, then the row index, and, for each iteration corresponding to a particular pixel, a ray trace is performed.

Investigation of the time that the whole PVG program takes to execute (code profiling) indicates that about 70% to 80% of that time is spent in the ray trace algorithm. Improving the frame rate of the program, therefore, means optimizing the ray trace operation as much as possible. One of Dr. Baer's more significant contributions is the so-called zig-zag argument which is applied to the ray trace when successive pixels in the same column are considered. Figure 9 shows two ray traces for successive column pixels, intersecting the ground.

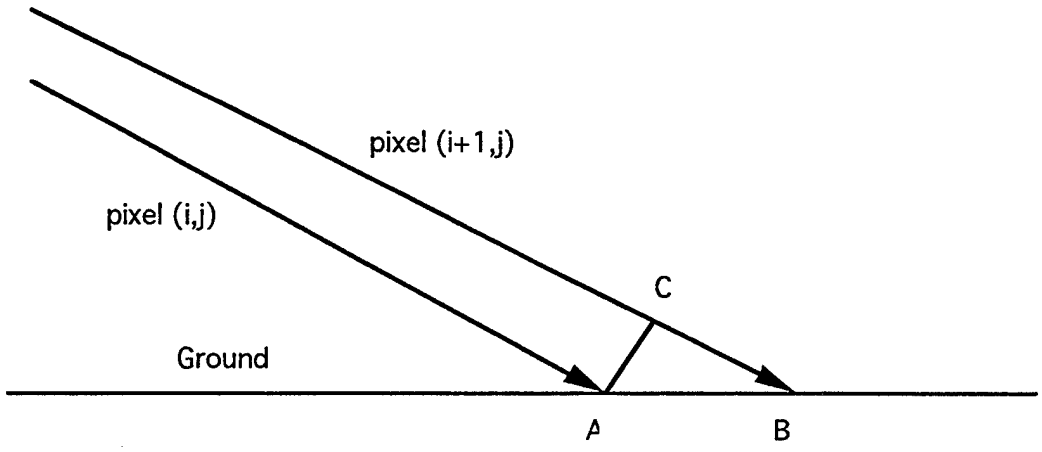


Figure 9 Zigzag Feature of Ray Trace

Suppose the first ray intersects the ground at A, after making 1000 steps along the vector. When the column pixel is incremented for the next ray trace, we do not start the ray trace from the observer, perhaps making 1003 steps along the vector before intersecting the ground at B. Instead, be begin our stepping 1000 steps along the new

vector from C, thereby reducing considerably the time taken to intersect the ground.

A final section of ZIGZAG calculates the total number of pixels correspond to ray hits on the target. This number, which is a dynamic measure of visible target area, is used later in the target detection algorithm.

HITGRND

This program determines what color should be stored in the appropriate screen pixel if the ray trace has not hit the target. Note that a ground hit will be reported if the ray trace intersects the terrain. Considering the way the terrain is defined, such a hit may occur if the ray hits the top of a tree, as shown in figure 10 by line XY

The current program allows three possible options.

(1) Bald Terrain: In this option, the under cover index confirms that an overhang does not exist above the element where the ground hit took place. In this case the gray shade stored in the screen pixel is extracted from the appropriate bit field in the database element. This is indicated by line AB or by line XY in figure 10

(2) Ground hit Below Overhang: Here, the undercover index confirms an overhang does exist at the ground hit element. In figure 11, a ground hit by ray EF would be such an example. In this case, it is assumed that the ground element is in a shadow, and the color black is stored in the pixel.

(3) No Ground hit, but Overhang: This is typified by the ray beginning at G in figure 10. Here the color of the pixel is again extracted directly from the corresponding ground element.

(4) Ray Passes Beneath Overhang: In this case, we are beneath the overhang, but have not yet hit the ground. In program HITGRND, this option is still under development. What we want to do is basically to continue the ray trace beneath the overhang until one of two things happens:

(a) The ground is hit, in which case the color black (shadow) is displayed, or

(b) The ray leaves the overhang without hitting the ground, as does ray CD in figure 10. In this case, if a return to ZIGZAG is made, having suitably updated our position on the ray, a normal ray trace can continue in ZIGZAG. This case will occur only when the observer is close to the ground, but could be significant if the PVG program is used for ground-to-ground encounters.

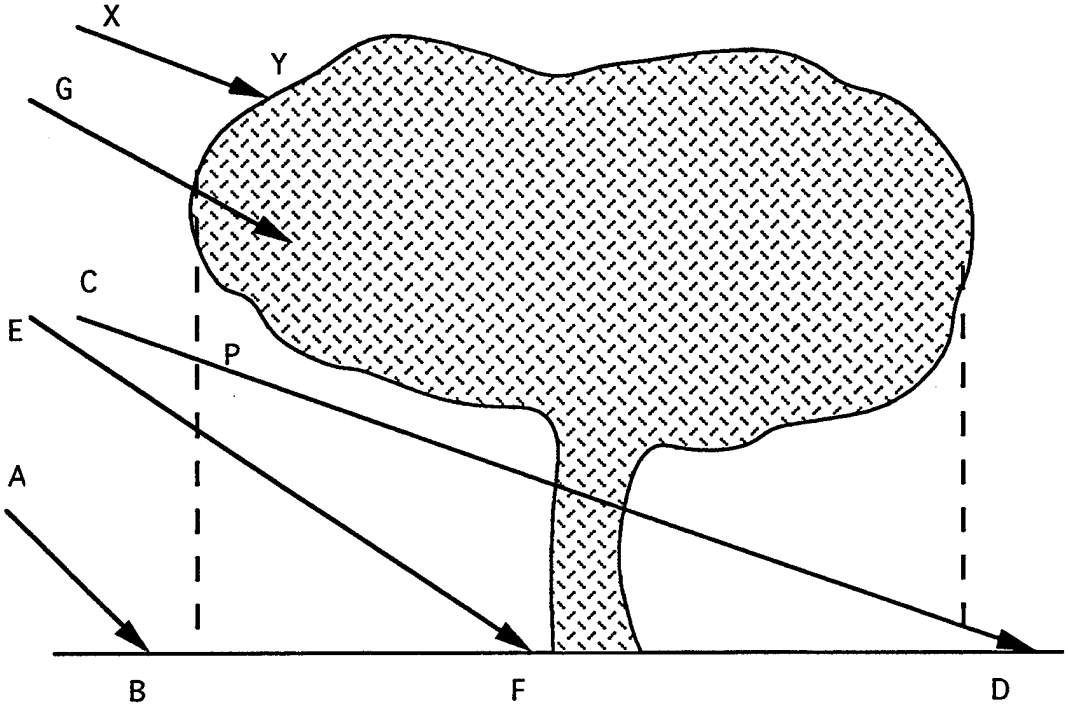


Figure 10 Ray Interactions with a Tree

ACQUIRE

This program implements the target detection and recognition algorithms used in the CASTFOREM approach, and applies them to the target and high resolution database found in the PVG program. The algorithms may be specified by the flow chart in figure 11. In implementing these algorithms, many improvements over their usual implementation in wargaming simulations such as JANUS may be made. This is because many of the parameters used in the algorithms become dynamic, and change from frame to frame. This is in contrast with the JANUS approach where, for example, the percentage of a target partially obscured by an object is fixed in the whole simulation, even though the target moves on the terrain. In the PVG program, targets may sometimes hide behind trees, or may move out into open ground. Their detectability will therefore vary as the target moves.

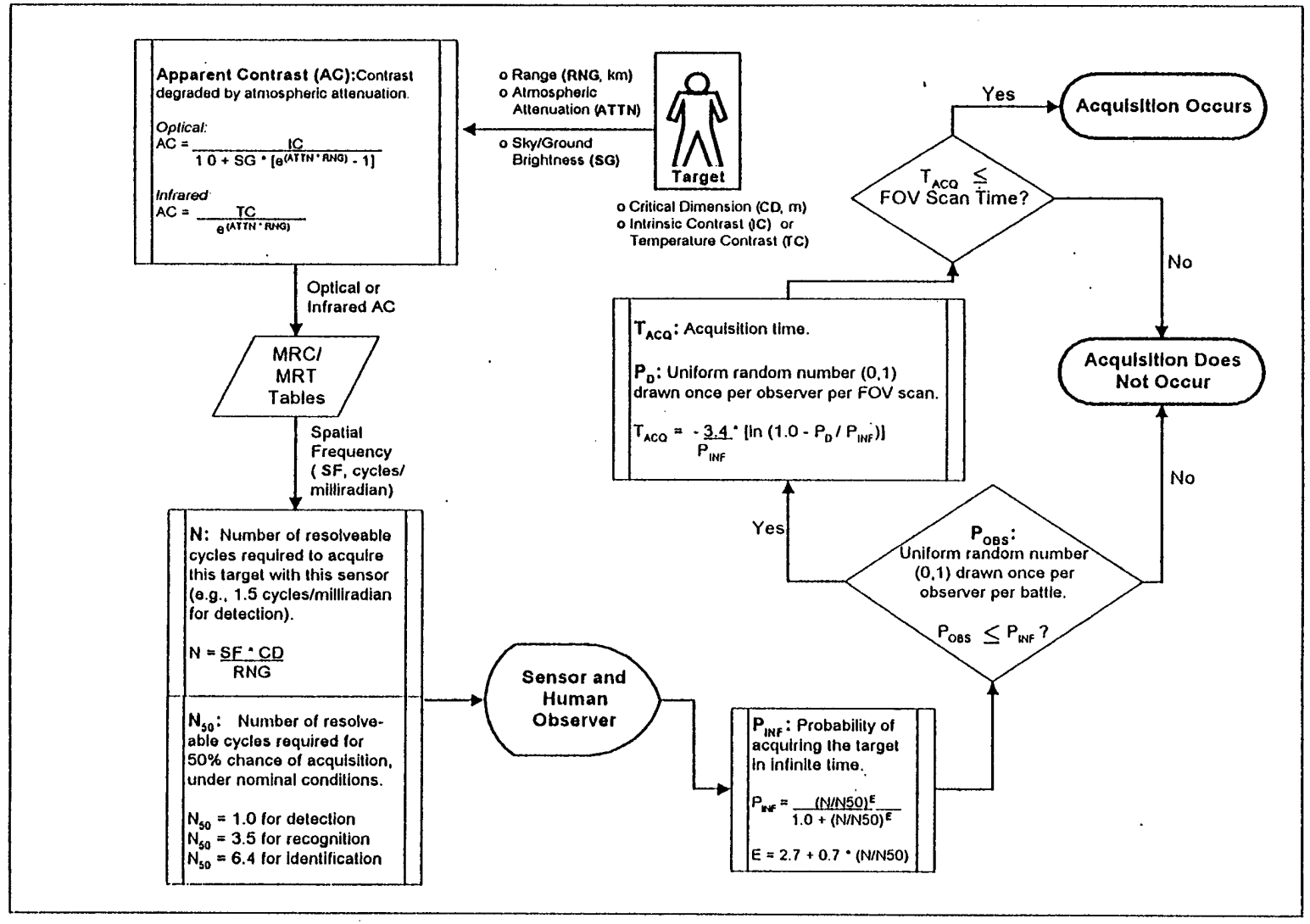


Figure 11 CASTFOREM Target Detection Algorithm

The program begins by calculating the range to the target from the observer. Then the apparent contrast is calculated. Based on the apparent contrast, the number of cycles/milliradian may be determined from the lookup table loaded into array CMR. Given the area of the target in the current frame, the number of cycles resolvable on the target may be determined. Then P-infinity, the probability of detecting, recognizing and identifying the target, may be found.

Finally, the time for acquisition is determined, based on the calculated P-infinity values and on a random draw to represent the probability that the target will be seen in a single field of view scan. Acquisition will occur if this time is less than the field of view scan time, which is defined to be 10 seconds. The completed results, including the three P infinity values and the results of the target acquisition tests (1 or 0) are written to the text screen for each PVG frame.

DISPLAY

Display writes the vector VIEW to the "Observer" window, and consists of two commands. The first is a RECTZOOM command which increases the size of the pixel array by the variable MAG. This will match the pixel array to the size of the screen in INIT_WINDOW. The second instruction is the LRECTWRITE command. This takes a vector, VIEW in this case, and writes it to the currently active window. This is a powerful GL command which handles a graphical array write operation to the screen with a single command.

RESTORE

This program is the last in the loop, and its function is to remove the target from the database. Recall in program TARGET how the terrain database, stored in array DAT, was modified to include a prismatic target, colored red, at the current target location. Program RESTORE restores the database to its original condition so that in the next frame TARGET may re-insert the target at the new target position. The restoration process consists of reducing the elevation in the vicinity of the target, and restoring the nature bit.

This concludes a description of the major functions in the PVG program. Figure 12 shows a sample frame produced by the program.

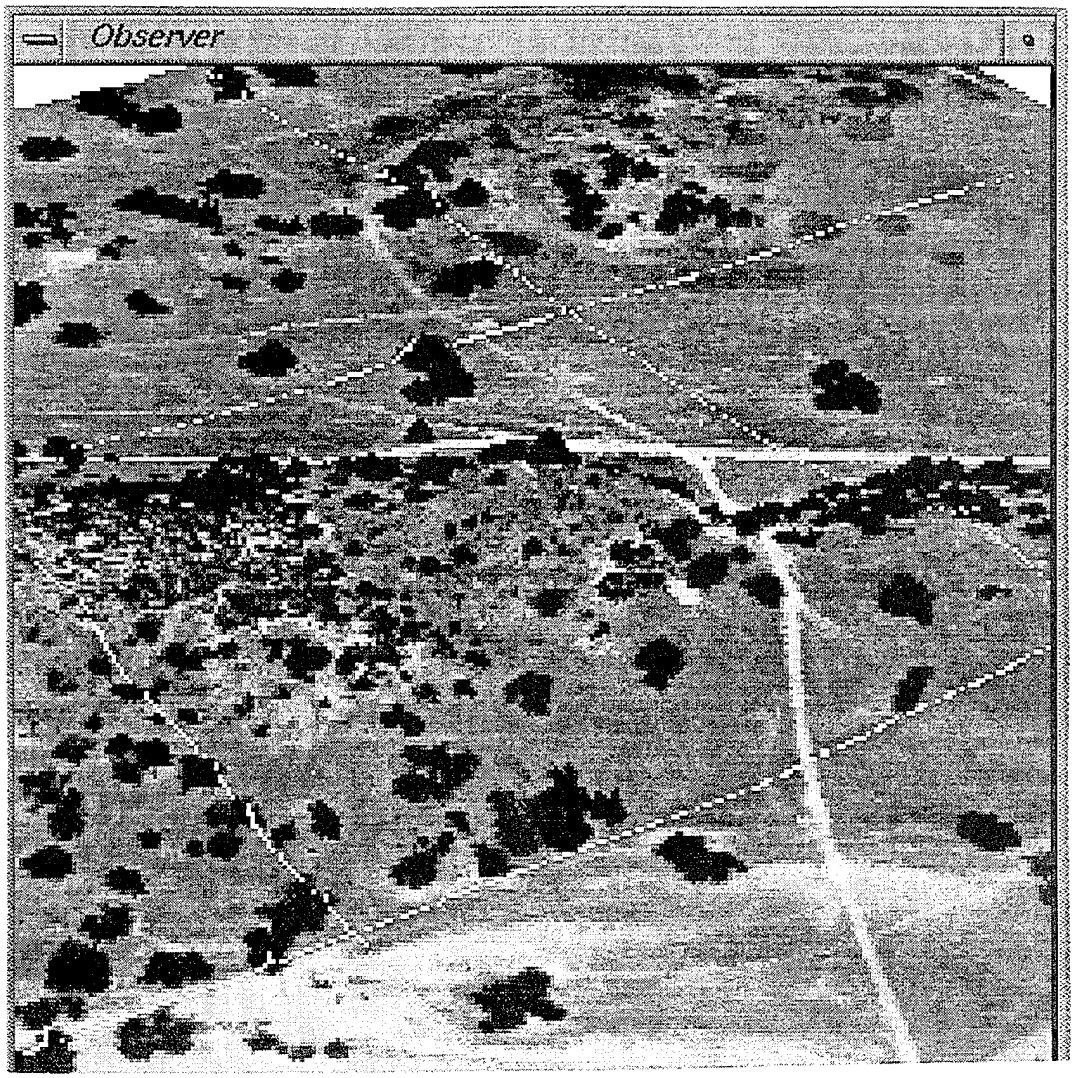


Figure 12 Sample Frame Produced by the PVG Program

Clearly there are many functions in the complete program, and therefore many dependencies between different sections of the program. A copy of the MAKEFILE for compiling the code follows, showing how each run time module is constructed.

When the complete program runs, all three graphics windows, and a text window are shown on the screen. This is illustrated in figure 13.

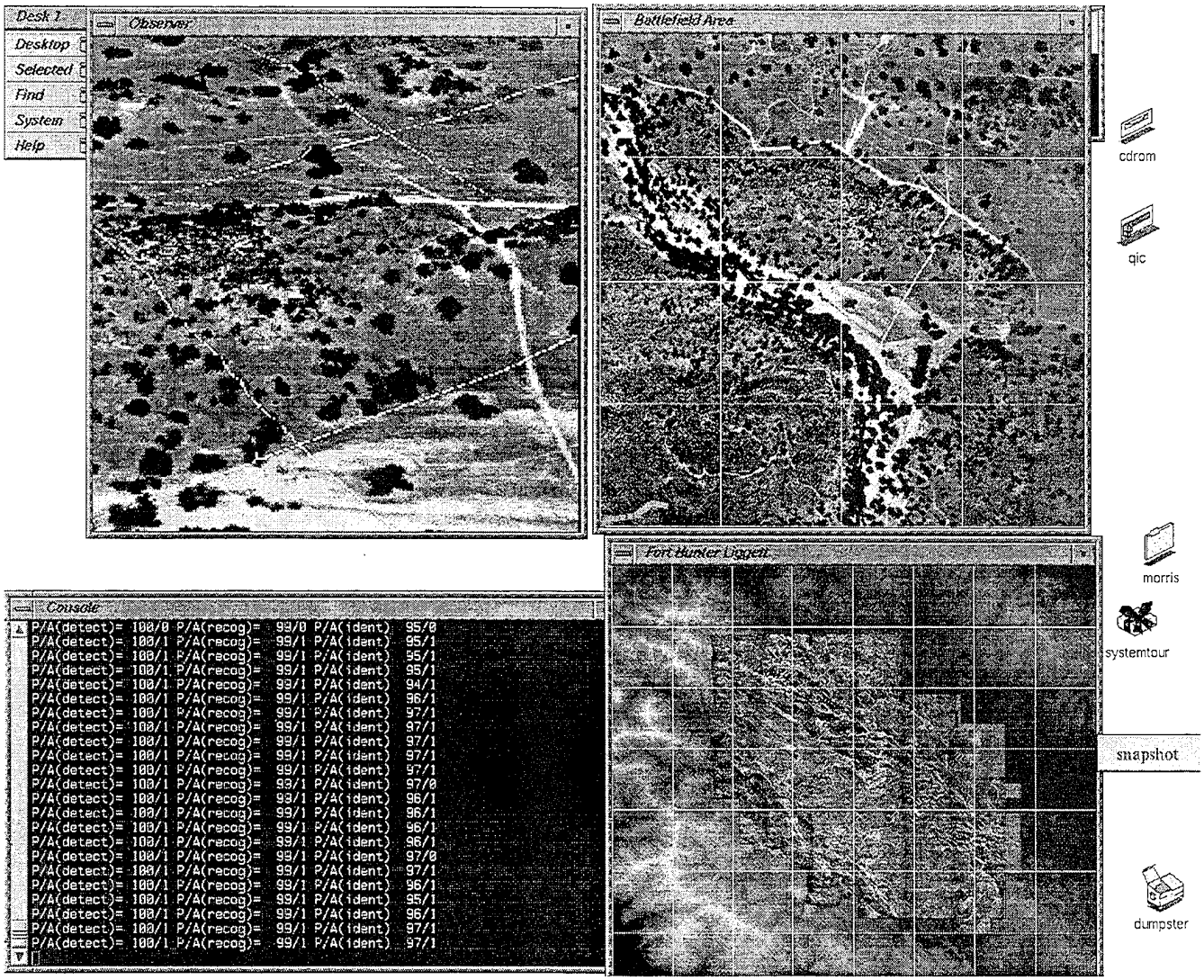


Figure 13 Screen Layout for PVG Program

3. MENUING SYSTEM FOR DATA INPUT

The system described so far demonstrates the feasibility of implementing a perspective view generator on a single processor workstation. Another important aspect of using such a program is the flexibility with which targets, their motion, and the environment in which they operate may be programmed. This would allow the user to program several engagements over different terrain types, with different targets and environmental scenarios. This would be accomplished with a menu system which would initialize the program. Although not implemented in the distribution version of the software, all the elements needed to do so have been explored. Full implementation will have to be deferred until later, due to time constraints for this fiscal year. The menu system establishes three data sets, dealing with

(1) Display setup
(2) Scenario Parameters
(3) Target Characteristics

The following text shows the interaction between the menuing system and the operator to initialize the program in the three areas outlined above.

DISPLAY SETUP

1 - Select one 4 x 4-km tile from the grid by placing the cursor on it and clicking the mouse button.

Is this the area you want?. . . . . Y

Choose one: Y = Yes
N = No

2 - From this tile, select the lower left corner of a 1 x 1-km area by clicking on the mouse button.

Is this the area you want?. . . . . Y

Choose one: Y = Yes
N = No

3 - Select a size for the graphics display. . . . . M

Choose one: S = Small
M = Medium
L = Large

4 - Is this the size you want?. . . . . .Y

Choose one: Y = Yes
N = No

5 - Would you like to edit the input file?. . . . . Y

Choose one: Y = Yes
N = No

If Yes, would you like to generate a new input file from
the default file? . . . . . .Y

Choose one: Y = Yes
N = No

If No, enter name of existing file to edit . . . . . .

SCENARIO PARAMETERS

Review the following input file.
Change ONLY the parameter values.
Make NO other changes to file.
When finished editing select Save As from File menu, enter name of file, and Quit.

TARGET CHARACTERISTICS

1 - Target Type. C

Enter one target type:

A = M106A1 (Trailer)
B = M125A1 (10 Ton Truck)
C = M1A1 (Abrams Main Battle Tank)
D = T-62 (Main Battle Tank)
E = M101 105mm (Howitzer)
F = M109 155mm (Self-Propelled Howitzer)
G = M2 (Bradley Armored Personnel Carrier)

2 - Number of Targets 1

Enter 1 - 12

3 - Target Configuration . . . .G

Choose one: G = Group
C = Convoy

4 - Target Motion . . . . . S

Choose one: S = Stationary
M = Moving

ENVIRONMENTAL CHARACTERISTICS

5 - Value for Atmospheric Transmittance 0.5

Specify transmission of radiation through atmosphere in sensor's spectral bandwidth:

Value between 0.0 - 1.0;
1.0 = Best Transmission

6 - Amount of Cloud Cover 1/8

Value must be in eighths, from 0 - 8/8;
0 = No Clouds; 8/8 = Total Cloud Cover

7 - Amount of Clutter 1

Based on your selected 1 x 1-km area, indicate density of bushes and trees in the area:

0 = Low Level of Clutter
1 = Medium Level of Clutter
2 = High Level of Clutter

SENSOR CHARACTERISTICS

8 - Sensor Type . . . . . .D

Choose one: D = Direct Vision
T = TV

If TV is selected then choose an option from below: . . . . . W

Choose one: W = Wide Field of View
N = Narrow Field of View

OBSERVER CHARACTERISTICS

9 - Type of Observer . . . . . G

Choose one: G = Ground-Based
F = Flyer

10 - Workload L

Complexity of controls and displays necessary to operate equipment and difficulty and complexity of maintaining it:

Choose one: L = Low
M = Moderate
H = High

When finished editing select Save As from File menu, enter name of file, and Quit.

TARGET CHARACTERISTICS

TYPE	FUNCTION	HEIGHT (m)	WIDTH (m)	SPEED (km/hr)
M106A1	Trailer	2.03	2.36	30.00
M125A1	10 Ton Truck	3.28	2.89	30.00
M1A1	Abrams	2.89	3.65	40.00
T-62	Battle Tank	2.39	3.30	29.00
M109A2	Self-Prop Hwtz	3.28	3.15	35.00
M101	105mm. Hwtz	2.16	1.52	00.00
M2	Bradley (APC)	2.97	3.20	40.00

Range of allowable speeds for OH-58D Kiowa Warrior:

204 km/h - economy cruising 219 km/hr - max cruising
241 km/hr - max

Speed for ground-based observer:

0 - min 40 km/hr - avg 60 km/hr - max

4. ENHANCED TARGET ACQUISITION ALGORITHMS

One objective of this year's project has been to continue the development of enhanced target acquisition software, for use in both the Perspective View Generator and JANUS (A). New algorithms will build on outputs from the ACQUIRE program described in Section 2. Data needed for new computations is provided by the user via the menu system described in Section 3.

The 1995 PVG target acquisition implementation first determines that there is clear line of sight between the target and observer. If this criterion is met, the system calculates observer-to-target range, inherent target-to-background contrast, and apparent contrast after degradation due to atmospheric attenuation. These are the primary factors used by the ACQUIRE program to determine threshold values for three target acquisition probabilities, that is, whether it is humanly possible for the target to be detected, recognized, and positively identified.

If ACQUIRE determines that the target *can* be acquired, the three calculated P-infinity values next are modified based on the user-entered menu selections. These selections may cause the probabilities to be either increased or decreased due to factors specific to the combat mission being simulated. The resulting modified P-infinity values then are used to calculate a modified time for acquisition, which is compared with the field of view scan time to determine whether the target will be detected, recognized, and identified in the time available. The result is a yes-no output similar to that from ACQUIRE but now taking into account additional human factors that influence target acquisition.

The 1995 development effort has been based on preliminary work done in 1993 and 1994 under a related project, *Improved Target Acquisition Algorithms for Janus (A)*. This project was conducted to evaluate the various factors that affect human target acquisition performance and that might be considered for inclusion in combat models; details may be found elsewhere (Lind). Some 150 factors were identified, falling into six categories: mission/tactics, target, environment, target area, sensor, and observer. This very large list was reduced to 37 factors that show reasonable potential for

implementation; 11 of the resulting short list were selected for initial modeling in 1995 (see table 1).[1]

Table 1 Factors Selected for Initial Modeling

Acquisition Parameters	Data Type	Units	Range of Values	Quantity of Values	Affects Tasks:
Target type	Alpha	None	A - G	7	Detection Recognition Identification
Number of targets	Integer	None	1 - 12	12	Detection
Target configuration	Alpha	None	G or C	2	Detection Recognition Identification
Target motion	Alpha	None	S or M	2	Detection
Atmospheric transmittance	Real	km^{-1}	0.0 - 1.0	N/A	Detection
Amount of cloud cover	Integer	None	0 - 8	9	Detection Recognition Identification
Amount of clutter	Integer	None	0 - 2	3	Detection Recognition Identification
Sensor type	Alpha	None	D or T	2	Detection Recognition Identification
Sensor field of view	Alpha	None	W or N	2	Detection Recognition Identification
Ground or air observer	Alpha	None	G or F	2	Detection Recognition Identification
Workload	Alpha	None	L, M, H	3	Detection Recognition Identification

[1]Appreciation is expressed to Jennifer Swoboda, Army Research Laboratory, Aberdeen Proving Ground, who carried out most of the 1995 analyses related to the 11 factors and developed the menu system.

TARGET TYPE

Seven types of targets have been defined for this initial effort (see the menus of Section 3). The user selects one of the listed target options for a given run. This automatically provides the program with target height, width, and speed to be used in calculations for this run, so that separate entry of these values is not required.

NUMBER OF TARGETS

Detection performance improves as the number of targets in view increases. Significant differences have been measured for one, three, and nine vehicles. Thus, although the user enters the quantity of targets he wishes to engage for a given run (up to 12), only three categories actually will be used in calculations: 1 target, 2 to 5 targets, and 6 to 12 targets. Figure 14 shows the quantitative relationship between the number of targets and detection performance.

TARGET CONFIGURATION

Detection performance is substantially affected by the spatial formation of multiple targets, especially when the targets number more than three. Figure 14 shows the relationship between target configurations (groups and convoys) and detection performance.

TARGET MOTION

Whether a target is stationary or moving affects detection, since moving objects catch the attention (even if seen peripherally). However, recognition and identification performance deteriorate if objects move at very high rates that will blur their features. If mobile, it is assumed that the target is moving at the fixed rate included in the program's target table for the selected type of target. This rate will improve detection performance but will not degrade the ability to recognize and identify the target.

ATMOSPHERIC TRANSMITTANCE

Atmospheric transmittance is needed for the ACQUIRE program's calculations. Transmission of radiation in the sensor's spectral bandwidth through the atmosphere is used in determining contrast attenuation as a function of the range to the target.

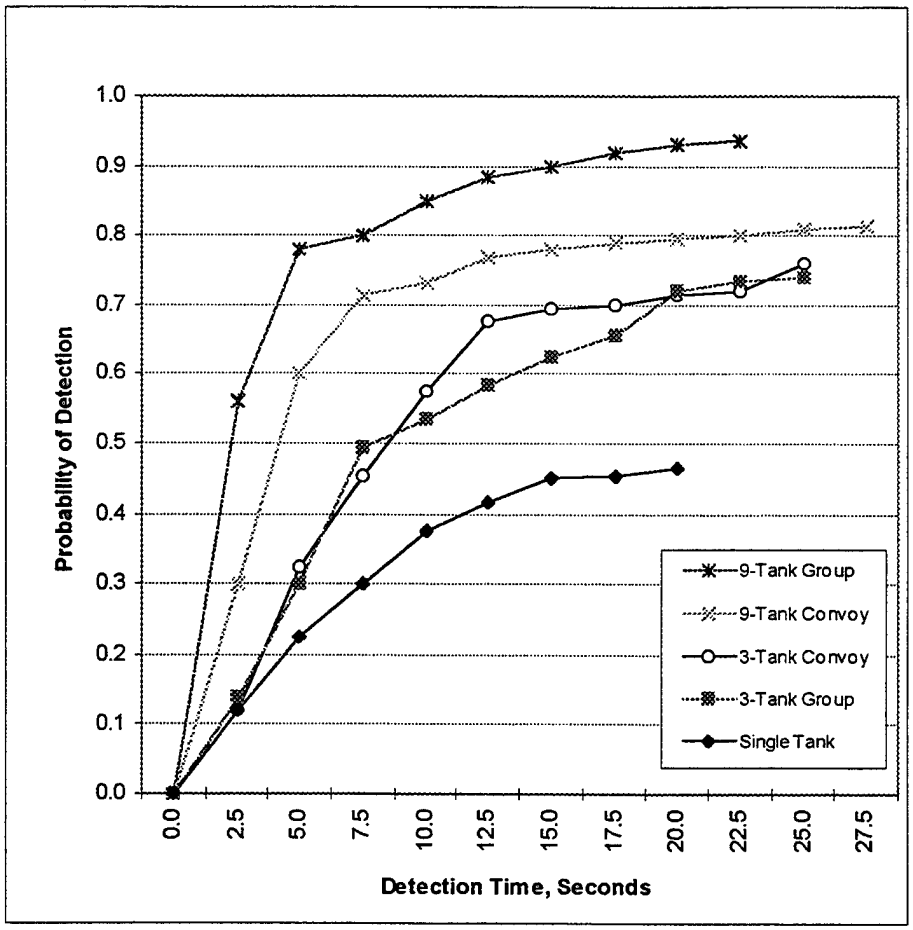


Figure 14 Performance Versus the Number of Targets

AMOUNT OF CLOUD COVER

Cloud shadows on the ground can affect acquisition much as clutter does, changing the apparent shape of a target and possibly causing it to blend into the background. Clouds covering between 3/8 and 7/8 of the sky will have the greatest dappling effect; a cloudless sky or one completely overcast will not substantially influence acquisition. The PVG does not model the sky, so it is necessary for the user to enter a value for this parameter.

AMOUNT OF CLUTTER

The density of target area clutter, proximity of clutter to the target, and how similar in appearance the clutter objects are to the target all strongly influence target acquisition performance. Theoretically the PVG should be able to measure the instantaneous clutter level around the target as the target moves across the terrain, but this is

currently beyond the capability of the system. Instead, the user will select a clutter level for use in probability calculations.

SENSOR TYPE

The type of sensor determines which lookup table of cycles per milliradian versus apparent contrast will be used for ACQUIRE calculations. At present only one table is included in the PVG (the CMR array, for direct vision acquisition), but inclusion of other tables will not be difficult. The menu system presently provides options for additional tables: one for direct vision and at least one for use when a television-type electro-optical sensor is utilized for acquisition.

SENSOR FIELD OF VIEW

Different lookup tables of cycles per milliradian versus contrast (CMR arrays) must be used for a sensor's various fields of view, for those sensors that provide the capability to switch between view fields. As noted above, at present only one CMR array is included in the PVG, but additional arrays can be added easily.

GROUND OR AIR OBSERVER

The PVG user can take the part of a hunter-observer using the spaceball to move across or over the terrain searching for the target. A ground observer will rotate the spaceball to change direction and move it forward and back to change speed. An air observer will have these same capabilities, but also can pull the spaceball upwards to change altitude. Selection of the Ground-based or Flyer option on the menu will determine spaceball use.

WORKLOAD

High observer workloads decrease attention levels and increase mental stress, physical stress, and fatigue æ reducing the probability that the target will be detected and, if detected, will be correctly identified. Workload level must be specified by the user based on the modeled situation, including the difficulty of operating weapons, sensor systems, and other equipment.

5. RESULTS AND FUTURE WORK

1. Simple perspective view generator code has been developed for a single processor SGI workstation. This provides a small program (about 400 code lines) for prototyping additional capabilities, such as NVESD detection algorithms.

2. The PVG works with a limited data base, about 3 km square, so that dynamic data flow management is avoided.

3. The program uses only single resolution data, usually the highest resolution available at 1 m x 1 m., although other resolutions work also.

4. Use of a spaceball is recommended for observers which fly, although a mouse based version is also available. Ground based observers are also supported.

5. The size of the terrain rendered depends only on the local RAM of the machine. A machine with 48 Mbytes displays about 10 sq km.

6. The frame rate on an entry level SGI Indigo (R4000 processor) is about 1.5 frames/second, if two auxiliary graphics screens are displayed.

7. Program may be adapted to any machine supporting OpenGL graphics language.

As far as future work is concerned, several avenues could increase the utility of the work completed so far.

1. The menuing system outlined previously should be implemented to provide a flexible tool for modeling a wide variety of engagements.

2. The ACQUIRE program should be enhanced to include the additional algorithms that have been developed to improve the modeling of target acquisition performance, for better agreement with observed field test results.

3. Parallel architectures for dividing the computing time for the ray trace algorithm should be investigated. Such architectures may be at the PC or workstation level, and preliminary work on the latter indicates that frame rates in the order of 10 are achievable with machines currently at NPS.

4. The representation of target in the database terrain has to be improved. The use of z-buffering on the SGI workstations should allow realistic target superpositioning. If possible, such targets should be rendered using commercial drawing aids, such as Autocad.

5. A small scale JANUS type program could be developed in which two players control small forces independently of each other. Such a demonstration would indicate that a high resolution database may lead to different strategies for the same scenario played on a conventional JANUS terrain.

6. Implementation of the software on a multi-processor workstation, say with 8 CPUs, and 256 Mbyte or more memory would allow video frame rates adequate for a complete tile of terrain. This would allow realistic, multi-unit engagements to be simulated, using multiple players.

5. REFERENCES

Young "Synthetic Environments for C3 Operations" MS Thesis, Mechanical Engineering Department, Naval Postgraduate School, Monterey, CA 93943. September 1994.

Whitney "Visualization of Improved Target Acquisition Algorithm for JANUS (A)". MS Thesis, Mechanical Engineering Department, Naval Postgraduate School, Monterey, CA 93943. December 1994.

Baer1 "An Approach for Real-Time Terrain Database Creation From Imagery", Baer, W., and Akin, J., SPIE/OE Aerospace and Remote Sensing Conference, Paper 1943-17, State of the Art Mapping Symposium, 13-15 April, 1993, Orlando FL.

Baer2 "Implementation of a Perspective View Generator Replicator", Baer, W., Transputing '91, edited by Welch, et. al., Vol 2, pp643-656, Isopress, Amsterdam, 1991.

Lind "Target Acquisition Models for Janus (A)," Lind, J. Naval Air Warfare Center Weapons Division TM 7811, China Lake, CA, February 1995.

INITIAL DISTRIBUTION LIST

1.	Research Office (Code 09) Naval Postgraduate School Monterey, CA 93943-5000	1
2.	Dudley Knox Library (Code 52) Naval Postgraduate School Monterey, CA 93943-5100	2
3.	Defense Technical Information Center 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	2
4.	Department of Mechnical Engineering Attn: Chairman Naval Postgraduate School Monterey, CA 93943	1
5.	Prof. Judith H. Lind (Code OR/Li) Naval Postgraduate School Monterey, CA 93943-5000	1
6.	US Army TRADOC Analysis Command Artillery Division Attn: ATRC-WBD, Bob Bennett White Sands Missile Range, NM 88002-5502	10
7.	TRAC Monterey Attn: LTC Ralph Wood Naval Postgraduate School Monterey, CA 93943-5000	1
8.	TRAC Monterey Attn: MAJ Bill Murphy Naval Postgraduate School Monterey, CA 93943-5000	1
9.	Prof. T. McNelley (Code ME/Mc) Naval Postgraduate School Monterey, CA 93943-5000	1
10.	LTC Mark Youngren (Code OR/Ym) Naval Postgraduate School Monterey, CA 93943-5000	1
11.	Prof. Morris Driels (Code ME/Dr) Naval Postgraduate School Monterey, CA 93943-5000	10
12.	Dale Robison Code 455530D	1

Naval Air Warfare Center Weapons Division
China Lake, CA 93555

13. Army Research Laboratory 1
Attn: AMSRL-HR-SD (J. Swoboda)
Aberdeen Proving Ground, MD 21005

14. Ronald A. Erickson 1
ASI Systems International
2835 Loraine Dr.
Missoula, MT 59803