AFIT/GCS/ENG/93D-13

RADAR CROSS SECTION VISUALIZATION USING SAMPLE BUFFER PROGRESSIVE REFINEMENT VOLUME RENDERING

THESIS

Alain L. M. Jones, Captain, USAF

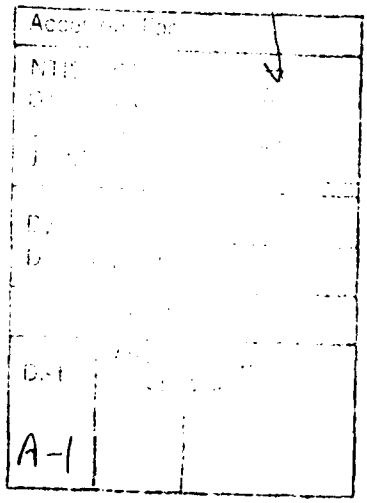
AFIT/GCS/ENG/93D-13

"Original contains color plates: All DTIC reproductions will be in black and white."

Approved for public release; distribution unlimited

Disclaimer

The views expressed in this thesis are those of the author and do not reflect the official policy of the Department of Defense or the United States Government.



DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF COLOR PAGES WHICH DO NOT REPRODUCE LEGIBLY ON BLACK AND WHITE MICROFICHE.

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 1993	Master's Thesis

4. TITLE AND SUBTITLE
RADAR CROSS SECTION VISUALIZATION USING SAMPLE BUFFER PROGRESSIVE REFINEMENT VOLUME RENDERING

5. FUNDING NUMBERS

6. AUTHOR(S)
Alain L. M. Jones, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/93D-13

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
ESC/YVM
Hanscom AFB, MA 01731

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

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

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This study developed a prototype for an interactive radar cross section visualization software system. The system, hosted on a Silicon Graphics workstation, is intended to support aircrews, mission planners, aircraft designers, and others who require an understanding of aircraft radar cross section characteristics. The input to the system is a set of radar cross section samples taken at various aspect angles. A pre-processor developed as part of this study transforms the input radar cross section data into a three-dimensional cuberille data volume. This data volume is then visualized using an interactive volume renderer. The interactive volume renderer implements progressive image refinement using the Ke & Chang Sample Buffer algorithm. A graphic user interface allows users to modify rendering parameters and see the results of their changes in near-real-time.

14. SUBJECT TERMS
Volume Rendering, Scientific Visualization, Radar Cross Sections, Stealth Technology, Computer Graphics

15. NUMBER OF PAGES
66

16. PRICE CODE

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

AFIT/GCS/93D-13

RADAR CROSS SECTION VISUALIZATION USING SAMPLE BUFFER PROGRESSIVE REFINEMENT VOLUME RENDERING

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Engineering

Alain L. M. Jones, B.S.C.S

Captain, USAF

December 1993

Preface

This thesis explores how to interactively examine the radar cross section characteristics of an object. As such, this research encompasses two rapidly burgeoning fields — scientific visualization and low-observables technology. I consider myself fortunate to have been able to spend a year working in these two dynamic fields.

I would like to thank my thesis advisor, Lt Col Martin Stytz, for providing the right level of guidance — enough to keep me on track, but with enough latitude to make the research truly mine. My thanks also goes to the members of my thesis committee, Lt Col Philip Amburn and Dr. Andrew Terzuoli, for their support during this thesis cycle.

Table of Contents

List of Figures

List of Tables

Abstract

This study developed a prototype for an interactive radar cross section visualization software system. The system, hosted on a Silicon Graphics workstation, is intended to support aircrews, mission planners, aircraft designers, and others who require an understanding of aircraft radar cross section characteristics.

The input to the system is a set of radar cross section samples taken at various aspect angles. A pre-processor developed as part of this study transforms the input radar cross section data into a three-dimensional cuberille data volume. This data volume is then visualized using an interactive volume renderer.

The interactive volume renderer implements progressive image refinement using the Ke & Chang Sample Buffer algorithm. A graphic user interface allows users to modify rendering parameters and see the results of their changes in near-real-time.

RADAR CROSS SECTION VISUALIZATION USING SAMPLE BUFFER PROGRESSIVE REFINEMENT VOLUME RENDERING

I. Introduction

Background

'First sight, first kill' has been a fighter pilot's maxim since the first days of aerial combat. Low Observable (LO) technology is playing an ever-larger role in insuring that our pilots are the ones who get that 'first sight.' LO technology must be a consideration in all future air weapons systems [Foul92]. Reducing an aircraft's *radar cross section* is one aspect of LO technology.

Radar cross section is a term used to characterize how much of the radar energy incident on a target is reflected back towards the emitter. Formally, the radar cross section (σ) of a target is defined by the following equation:

$$\sigma = \lim_{R \to \infty} 4\pi R^2 \frac{|E_s|^2}{|E_i|^2}$$

where R is the distance from the scatterer, E_S is the scattered field strength and E_i is the incident field strength [Knot87]. The radar cross section of a target varies greatly from one aspect angle to another. Because of these large variations, it is often convenient to

express radar cross section in logarithmic units. Radar cross section is usually expressed in terms of dBsm, which means "decibels over square meter" and is defined as

$$dBsm = 10 \log_{10}(\sigma) \quad [Knot85]$$

Radar cross section is also a function of, among other things, the radar frequency, as well as the incident and received polarizations [Knot85]. For each combination of horizontal or vertical incident polarization and horizontal or vertical receiver polarization, we have a different radar cross section [Knot85].

Problem

Aircrews, mission planners, and air vehicle designers require a clear understanding of the radar cross section characteristics of their aircraft. This study investigates the use of computer graphics to generate visualizations of this radar cross section data.

In particular, this study will apply the scientific visualization technique of *volume rendering* [Kauf91]. The visualization of radar cross section data using volume rendering was previously investigated by Tisdale [Tisd92]. Tisdale concluded that volume rendering could generate useful visualizations of radar cross section data, but that the time to generate these visualizations was long relative to other available techniques. This thesis will follow-on to Tisdale's work by investigating speed-up techniques for volume rendering and applying those fast volume rendering techniques to the problem of visualizing the radar cross section of an object.

Research Objective

The research objective is to develop a prototype interactive volume rendering system which generates visualizations of radar cross section data. The volume renderer will implement the Sample Buffer progressive refinement volume rendering algorithm described by Ke & Chang [Ke93].

Scope and Limitations

This thesis effort will limit itself to the design, implementation, testing and analysis of the objective volume rendering system. Only volume rendering approaches to radar cross section visualization will be considered.

The application of surface rendering and particle systems to this problem has been investigated by Tisdale & Wojszynski [Tisd92], [Wojs92]. Those approaches will not be addressed in this thesis.

Approach

This research began with a review of the current literature in the field of volume visualization, with particular emphasis on volume rendering fundamentals and speed-up techniques. Radar fundamentals references were also reviewed to gain some familiarity with the domain of the visualization.

After completion of the literature review, design and implementation of the prototype Interactive Volume Renderer began. The VOXELIZE pre-processor for

converting RCS data to volumetric data was also developed. This effort took place in the AFIT Graphics Laboratory, using Silicon Graphics workstations.

Once the software tools were developed, they were evaluated using a representative RCS data set . This is the same RCS data set used by Tisdale and Wojszynski in their radar cross section visualization efforts [Tisd92], [Wojs92].

Thesis Overview

The remainder of this thesis is organized as follows. Chapter 2 is a brief literature review in the area of volume rendering. Chapter 3 describes the design and implementation of the Interactive Volume Renderer software system. Chapter 4 presents the results of this research. Finally, chapter 5 presents this study's conclusions and suggests areas for follow-on research.

II. Current Knowledge

The first step in developing the RCS visualization system was a review of the technical literature. This chapter presents the results of that literature review. The first section provides background on volume rendering fundamentals. The second section discusses the volume rendering speed-up techniques considered. The final section provides a more detailed description of the particular speed-up technique which was used in our visualization system, the Ke & Chang Sample Buffer algorithm [Ke93]

Volume Rendering Background

Volume visualization is a collection of computer graphics techniques for visualizing three-dimensional functions [Laur91], where the term *function* is used in a general sense to indicate a set of values over three-space. Kaufman sub-divides the field of volume visualization into *volume rendering* and *surface rendering* [Kauf91]. In volume rendering, the 3-D function to be visualized is represented by a discrete voxel space — a 3-D grid of unit volume cells called *voxels*. This is distinguished from surface rendering, which extracts isosurfaces from the 3-D function and represents them as geometric primitives (*e.g.* polygons).

Kaufman generalizes the processing steps of any volume visualization system into a *volume visualization pipeline* [Kauf91]. A volume rendering system can *enhance*, *manipulate*, and *classify* its input voxel space. It then *maps* the voxels to display primitives, *projects* them into 2-D space, and *shades* them, yielding the completed 2-D

image. This generalized pipeline is represented in Figure 1. The following section explains the steps of this pipeline, as well as speed-up techniques for each step.

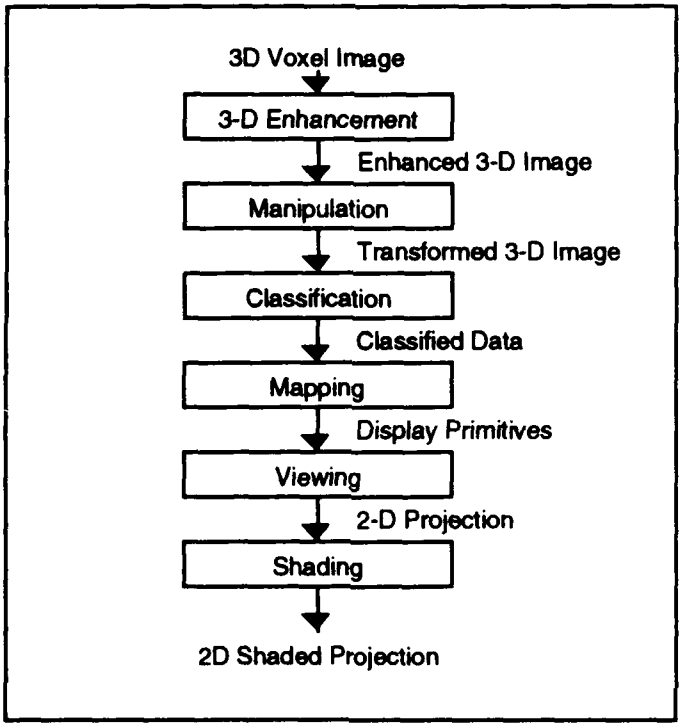


Figure 1. Volume Rendering Pipeline [Kauf93]

A popular volume rendering architecture, exemplified by the VIPER system used in previous thesis efforts [Tisd92, Brid88], implements this pipeline as follows. Each voxel is assigned a color and opacity , and a 2-D projection of the resulting colored semi-transparent cube is computed [Levo90]. A simple, brute-force volume rendering algorithm simply casts rays into the volume to compute the 2-D image, as illustrated in Figure 2. At even intervals along the ray, the color and opacity of the volume is sampled by tri-linearly interpolating the colors and opacities of the eight nearest voxels. These samples are composited with each other to yield the color of the ray.

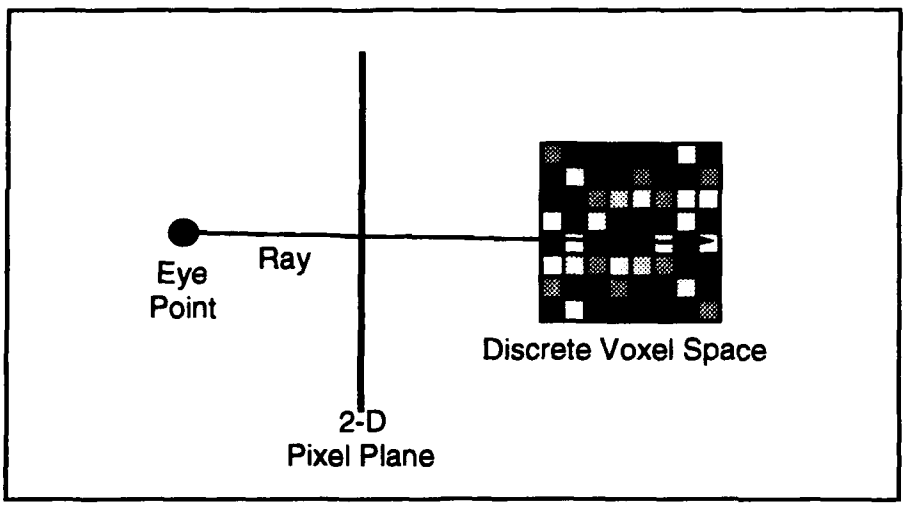


Figure 2. Ray Casting into Discrete Voxel Space

Volume Rendering Speedup Techniques

A number of volume rendering techniques have been developed in an effort to achieve faster visualizations. These various speed-up techniques can be organized by the portion of the volume visualization pipeline they address.

3-D Enhancement. This step deals with improving the understandability of the voxelized data by, for example, applying image processing techniques to improve contrast [Kauf91]. Image enhancement was not an objective of this thesis — speed-up techniques in this area were not researched.

Manipulation. This stage deals with geometric or domain transformation on the voxel data, as well as voxblt operations [Kauf91]. In this step, speed-up could be accomplished by clipping the voxel space down to a *volume of interest* [Styt91], thereby reducing the amount of data to be processed. This can be accomplished by applying a *matting* transform to remove a portion of the data [Dreb88].

Classification. For volume rendering, this step may assign optical properties to individual voxels (e.g. color, opacity) [Kauf91]. This can be done by applying a *color transfer function* to the data value of a voxel. As this transfer function can be expensive, speed-up in this area can be accomplished by minimizing the number of times it is called [Ke93].

Mapping. In this stage, 3-D voxel data is mapped into display primitives [Kauf91]. This stage can be sped-up by using an octree representation of the voxel space [Same91]. An octree represents the 3-D volume at varying levels of resolution, with the root of the octree representing the volume at minimum resolution, and the leaves representing the volume at maximum resolution. By selecting the octree level from which primitives will be mapped, a trade-off can be made between number of primitives and resolution. Speed-up is accomplished by reducing resolution and hence the number of primitives.

Hierarchical splatting uses an octree to represent the voxel space at varying levels of resolution, then maps the octree nodes to 2-D *splats* of varying sizes [Laur91]. A splat draws each voxel as a cloud of points, spread across multiple pixels [Yage93]. Progressive image refinement is implemented by progressively increasing the resolution of the octree. Williams has developed a splat-based approach to render an image from non-rectilinear grids [Will92] Under this approach, splats can be generated without first interpolating the data onto a rectilinear grid.

Adaptive Isotriangular Subdivision reduces the computational expense of traditional ray casting by sampling on a grid of isosceles triangles instead of quadrants [Shu91]. The geometry of the isosceles triangle grid allows sampling of the volume with fewer rays than a rectangular grid, without sacrificing image resolution.

Viewing. In this stage, the display primitives are projected to form a 2-D screen image [Kauf91].

Avila and Sobierajski's *Polygon Assisted Ray-Casting (PARC)* algorithm uses a Z-buffered polygon-oriented graphics engine to speed up volume rendering. It does this by forming a polygonal approximation of the front and back edges of the object being rendered. Cast rays need only traverse those parts of the volume between the front and back edges of the approximation. Best results are achieved with sparse volumes [Avil92].

Volume rendering lends itself to a parallel implementation. Elvins reports implementing a distributed splatting system on both a specialized multi-processor architecture [Elvi91] and a network of general-purpose workstations [Elvi92]. Each processor computes the image contribution of a slice of voxels. However, message passing becomes a bottleneck for large images.

Fuchs *et al.* propose opacity thresholds on ray casting, wherein a sampling along a cast ray would cease after the ray has reached some user-specified opacity level [Fuch90]. Beyond that point, the contributions of the remaining untraversed voxels could be assumed negligible.

Shading. The final step of the pipeline is to apply shading. The ultimate goal of this step is to obtain a photorealistic 2-D image [Kauf91]. However, speed-up can be obtained by trading-off realism for speed. Hibbard *et al.* propose that rapid rendering and interactivity add more to understanding than esthetics and photorealism [Hibb90].

Ke & Chang Sample Buffer Algorithm

The Ke & Chang Sample Buffer algorithm provides a progressive refinement approach to ray-casting volume rendering[Ke93]. In essence, the algorithm functions as follows. The volume data is mapped into an octree. The octree is then traversed according to a user-specified resolution criterion. Increasingly refined images are

generated by increasing the resolution criterion. Ray casting is used to sample the octree. A *sample buffer* data structure is used to speed up ray casting by saving the results of previous rendering passes.

Two data structures are central to this algorithm: the *octree* and the *sample buffer*. This section will describe these two data structures, how the algorithm functions uses these data structures, and describe a run-length encoding enhancement.

Octree Data Structure. The input voxel space is encoded into an octree. The octree is a tree structure with a branching factor of eight [Styt91]. The root node represents the entire voxel space. The children of the root each represent one octant of the voxel space. Each subsequent level of the tree represents a further eight-way subdivision of the voxel space. The leaf nodes of the octree represent individual voxels. Figure 3 depicts an octree with a depth of two, and its correspondence to voxel space.

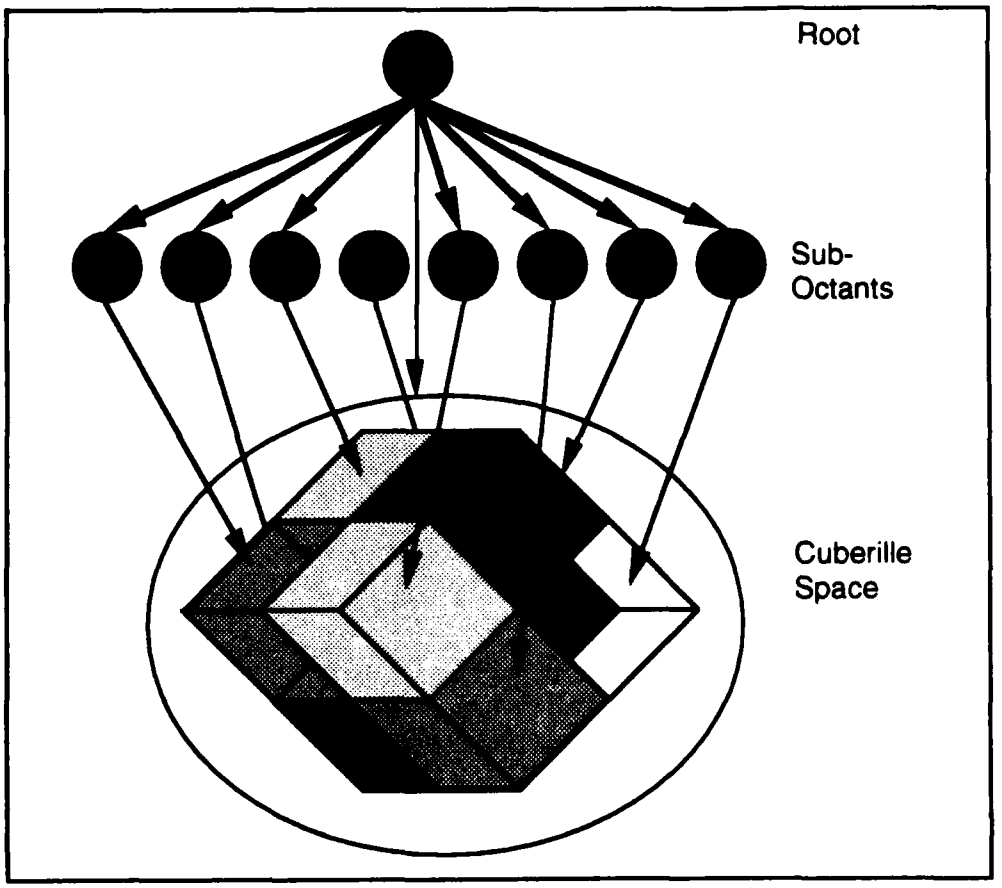


Figure 3. Relationship between Octree and Voxel Space

For each node of the octree, three important attributes of the voxel space it encloses are also included: the *color* and *opacity* of the space, and the *error* of the space. The color and opacity of the space is the average of the voxels within it. The error of the space reflects the amount of variation within the space. The implementor may select any reasonable error measure — Ke & Chang use the difference between the highest and lowest data value within the space.

Sample Buffer Data Structure. For each pixel, there is a corresponding sample buffer. The octree consists of a list of sample records. The data contained in the sample record is shown in figure 4.

```
struct {
  TYPE       // TYPE == approximate if the sample's color is approx
             // TYPE == exact if the sample's color is true
  union {
    OCTANT   // used if TYPE == approximate
    RGBA     // used if TYPE == exact
  }
}
```

Figure 4. Sample Record Definition [Ke93]

For each data sample taken along a ray, a sample record is added to the corresponding pixel's sample buffer, as shown in figure 5.

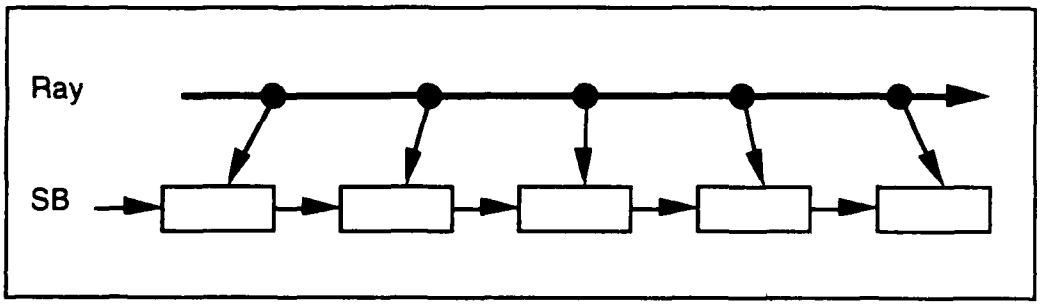


Figure 5. Sample Buffer List Associated with a Ray [Ke93]

Sample Buffer Algorithm. The progressive refinement algorithm has N phases. For each phase, the *acceptable error* criterion is progressively reduced. During each phase, a ray is cast through each pixel.

During the first phase, for each sample along the ray, the octree is searched from the root down until a node that meets the acceptable error criterion is found. There are two possible outcomes — either a node which meets the acceptable error criterion is found, or not.

In the former case, an octree node which meets the acceptable error criterion is found. A sample record of type 'approximate' is then inserted into the sample buffer. This type 'approximate' sample record points to the octree node found.

In the alternate case, a leaf node of the octree is reached and the acceptable error criterion is still not met. In this case, the sample color is determined by interpolating the data value and applying the color transfer function to the interpolated value. A sample record of type 'exact' is inserted in the sample buffer. A type 'exact' sample record, rather than pointing to an octree node, instead contains the actual sample color just computed.

During subsequent phases, for each sample along the ray, there is already at least one sample record in the sample buffer. There is no need to begin searching the octree from the root node. Instead, if the corresponding sample record is of type 'approximate', the octree is searched starting from the octant pointed to by the sample record. If the sample record is of type 'exact', there is no need to search the octree at all, as the previously computed color value stored in the sample record may be used.

The color of each pixel is determined by simply front-to-back compositing the color values represented in its sample records. The equations used for compositing are:

$$C_{out}(i) = C_{out}(i-1) + C(i) * \alpha(i) * (1-\alpha_{in}(i))$$

$$\alpha_{out}(i) = \alpha_{out}(i-1) + \alpha(i) * (1-\alpha_{in}(i))$$

For $i = 1 .. N$

where

$$C_{out}(0) = 0$$

$$\alpha_{out}(0) = 0$$

The final color and opacity of the ray are given by $C(N)$ and $\alpha(N)$.

Run-Length Encoding Enhancement. Memory usage of the algorithm can be reduced by applying run-length encoding to the sample buffers. There are two cases when run-length encoding can be applied: consecutive 'exact' records, and samples in the same leaf or octant. These two cases are explored below.

When two or more consecutive sample records in a sample record are of type 'exact', they can be replaced by a single sample record. The color and opacity values of this combined record is obtained by front-to-back compositing the colors and opacities of the consecutive 'exact' records. The resulting sample record has the same optical characteristics as the series of sample records it replaces.

When two or more samples are in the same leaf or in the same octant, their color attributes can be composited and stored as a single sample record. Applying this compression option would reduce storage requirements. However, using this compression scheme could either speed up or slow down subsequent processing. Speed-up would occur if all of the composited nodes meet the error criterion for the next refinement phase, since the pre-computed composite color value would be re-used. Slow-down would occur if any of the composited nodes do not meet the error criterion of the next refinement phase. In that case, the composited value computed would only be used once, and the computational overhead of the compression scheme would not be offset by any execution time savings.

III. System Design and Implementation

Overview

The software prototype developed for this research consists of two parts. An *Interactive Volume Renderer* implements the Ke & Chang Sample Buffer progressive refinement volume rendering algorithm [Ke93]. A *Voxelizer* converts the radar cross section simulator output file into a cuberille data set readable by the Interactive Volume Renderer. This chapter will describe the design and implementation of each part of the software prototype.

Interactive Volume Renderer

The Interactive Volume Renderer (IVR) is implemented in ANSI C. It runs on the Silicon Graphics platform. IVR has four major components. The *octree generator* builds an octree from the input voxel file. The *rendering engine* renders the octree using the Sample Buffer algorithm. The *color transfer function* maps voxel data values to optical characteristics (*i.e.* color and opacity). The *user interface* allows interactive modification of the rendering parameters. Each will be discussed in turn.

Octree Generator. The octree generator reads in a cuberille data file and converts it to an octree that can be used by the Sample Buffer algorithm.

Each voxel in the input cuberille data is taken to be a unit cube. For simplicity of the prototype, the decision was made to limit the input to cuberille data, rather than a more general rectangular or irregular grid. A method for volume rendering rectangular

grids is described and implemented by Bridges [Brid88]. Methods for rendering irregular grids are described by Foley [Fole90].

The data structure used for each node of the octree is shown using C-like pseudo-code in Figure 6.

```
struct octree_record
{
  color_type color;    // The average RGBA of the volume
  float error;         // Variation in data values w/in the volume
  int span;            // Length of a side of the volume
  vector min_corner;   // Lowest x, y, z coords in volume
  boolean is_leaf;     // Flag indicates a leaf node
  union
  {
    // Pointers to the sub-octants
    // Used when the node is not a leaf
    struct octree_record *node_ptr [8];

    // Pointers to eight voxels which make up the volume
    // Used when the node is a leaf
    int voxel_offset [8];
  } children;
};
```

Figure 6. Octree Node Record

The octree generator operates recursively. It creates a root node which encompasses the entire cuberille data space. It then invokes itself for each of its sub-octants. When the sub-octants of a node consist of only one voxel, the color of each voxel is computed, and the color of the node is set to their average.

The color of the parent node is the average of its contained voxels. The opacity of a node is the average opacity of its contained voxels, corrected for thickness by multiplying it by the cube root of the number of contained voxels. This opacity correction is used so that our Sample Buffer implementation need not concern itself with the thickness represented by any given node.

The *error* parameter stored in each octree node is a measure of the data value variations within the sub-volume it represents. In this prototype, the error parameter is computed by simply subtracting the minimum data value found in the sub-volume from the maximum data value in the sub-volume.

Some redundant information is stored in the octree in order to improve the speed of the Sample Buffer algorithm. The minimum x, y, and z coordinates of the contained volume is stored in each node, even though that information can be derived from the position of the node relative to the root. However, since the Sample Buffer data structure points directly to octree nodes, it cannot readily determine the relative position of the node without searching the octree. For the same reason, the width of the contained volume is also stored in the node record.

The octree is stored using a conventional dynamic memory implementation, where each node contains pointers to each of its eight children. This method was chosen for its simplicity of implementation. Alternative implementations include using a linear octree structure [Garg86] or a hash table [Glas84].

The child nodes of the octree are numbered 0 through 7. In order to simplify traversal calculations, the child numbers are assigned according to their position relative to the center of the parent octant. The numbering scheme is explained in figure 7.

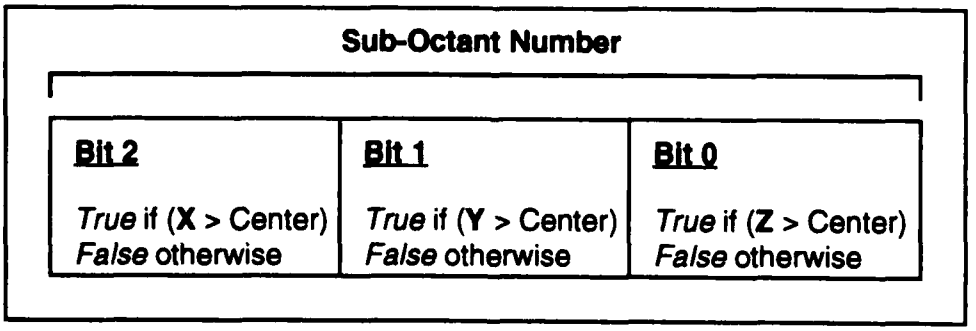


Figure 7. Octree Child Numbering Convention

Rendering Engine. This is the heart of the Interactive Volume Renderer. The rendering engine implements the Ke & Chang sample buffer algorithm [Ke93].

Progressive refinement is implemented by multiple rendering passes. Each pass displays octree nodes that have an error parameter less than or equal to that pass' acceptable error. The acceptable error ranges from a user-specified initial value for the first pass, down to zero for the last pass. At the end of each pass, the acceptable error is divided by a user-specified divisor.

Data Structures. The sample record data structure implemented in the rendering engine differs slightly from the one described by Ke & Chang [Ke93]. In addition to the data fields specified by Ke & Chang, this implementation adds an *entry point* field. This field specifies the exact coordinates of the point at which the ray enters the octree node. By adding this field to the data structure, the entry point of the ray into the sampled sub-volume need not be re-computed each phase.

Initialization. The sample buffer for each pixel is initialized to one of two values. If a ray cast from the eye point through the pixel intersects the data volume, then the sample buffer is initialized to point to the root node of the octree, and the point of entry into the volume is saved in the sample buffer. If the ray does not intersect the data volume, the sample buffer is initialized to a null value. In this way, the ray-volume intersection calculation need only be done once for each pixel.

Rendering. Rendering is accomplished by traversing and sub-dividing the sample buffer for each pixel. The pseudo-code for this traversal is shown in figure 8. If a sample record points to an octree node that does not meet the acceptable error criterion, the algorithm computes the path of the ray through the node, and replaces the original sample record with one or more sample records representing the path of the ray through the higher-resolution portions of the octree.

```
IF sample_buffer is null THEN
  render pixel in background color
ELSE
  initialize pixel color to zero;
  REPEAT
    WHILE error(sample record) > acceptable error;
      subdivide (sample record);
    composite current sample record onto pixel color;
    move to next sample record;
  UNTIL at end of sample buffer;
```

Figure 8. Sample Buffer Traversal and Sub-Division Pseudo-Code

Sample Record Subdivion. The sub-division of a sample record is central to this implementation of the Sample Buffer algorithm. To subdivide a sample record, the algorithm at Figure 9 is used. The sub-division algorithm depends on the fact that the sample record includes the exact location of the entry point into the octree node's sub-volume. Without this information, the algorithm would have to perform an expensive ray-volume intersection computation for each sub-division operation.

```
original_node := octree node pointed to by sample record;
delete (sample record);
IF original_node is a leaf node THEN
  tri-linearly interpolate the color of the voxel;
  insert exact-type sample record;
ELSE
  set ray to the entry point of the original_node;
  REPEAT
    compute which sub-octant the ray is in;
    insert new sample record which points to sub-octant;
    advance ray to the exit point from sub-octant;
 UNTIL ray exits original_node;
```

Figure 9. Sample Record Sub-Division Pseudo-Code

The initial entry point into the volume and the exit point from each sampled sub-volume are analytically computed. Implementing the volume traversal using the 3D-Digital Difference Analyzer [Fuji86] was initially considered. The 3D-DDA algorithm

was appealing because it can rapidly compute the traversal path through a volume grid. However, the 3D-DDA was not used in this implementation because it did not readily provide the precise voxel entry points required for interpolation.

Run-Length Encoding. The run-length encoding enhancement described by Ke & Chang was partially implemented [Ke93]. Successive type 'exact' sample records are composited together. However, compositing sample records from the same octant was omitted for simplicity. The run-length encoding feature can be user-disabled for performance impact evaluations.

In order to avoid significant errors due to round-off during compositing, this implementation of run-length encoding uses a floating-point representation of color values. In this way, any round-off errors are so small as to disappear when the floating-point color values are converted to 24-bit form. However, this accuracy comes at the cost of increased computation time and larger sample record size.

Color Transfer Function. The color transfer function maps a voxel's data value to a color and opacity. For the color, the prototype uses a rainbow scale, ranging from red to violet. To implement the color scale, a Hue Saturation Value (HSV) color scale is used. The data value is mapped to a Hue angle from 0 to 350, while saturation and value are kept at unity. Finally, the HSV color specification is converted to RGB space. Opacity also varies linearly with the data values between user-specified extrema.

This implementation can be readily extended to the display of multi-variate data. The now-unused portions of the color space (*i.e.* those colors where saturation and value are less than unity) could be used to map additional variables.

User Interface. The user interface for the Interactive Volume Renderer was implemented using the Forms Library, a public domain graphical user interface library for Silicon Graphics workstations. The goal of the user interface was to give the user significant real-time control of the rendering process so as to facilitate data exploration.

The user interface consists of three parts: a main interface window, a legend window, and a color specification window.

Main Interface Window. The main user interface window is shown in Figure 10.

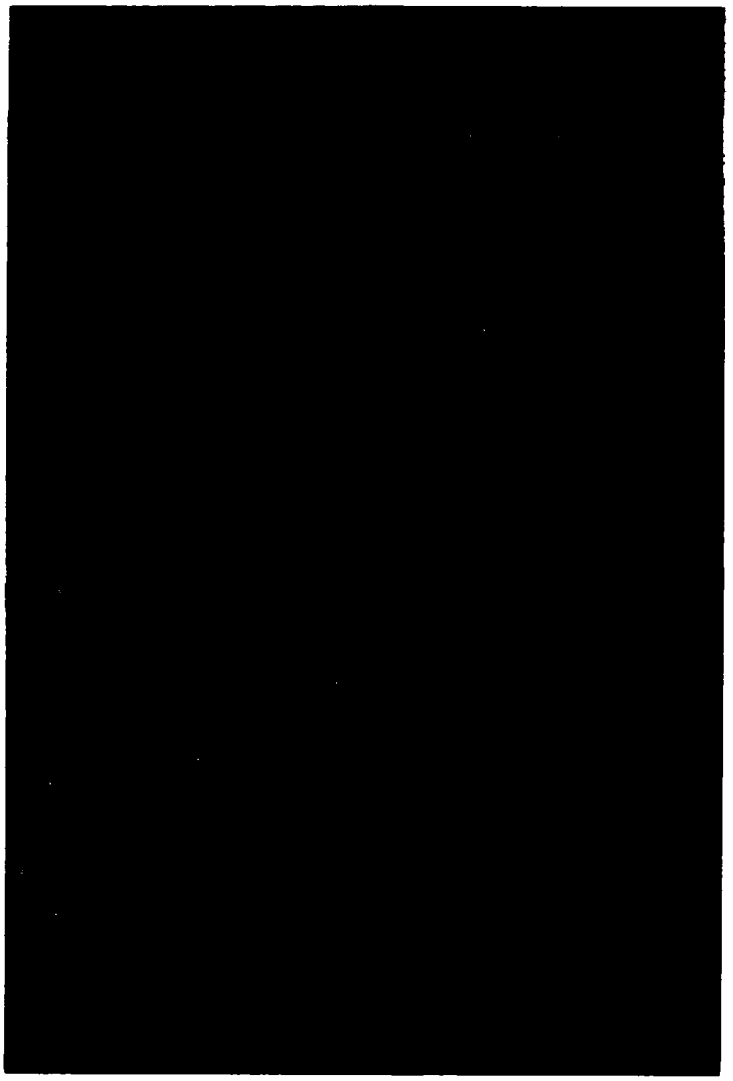


Figure 10. IVR Main User Interface Window

The main window has four functional areas. Similar controls are grouped together in their own functional areas. The functional areas are clearly delineated by

bounding boxes of different colors. The four functional areas are the Viewing Geometry area, the Acceptable Error area, the Physical Screen area, and the Command area.

The Viewing Geometry area includes sliders for controlling the rotation, translation, and scaling of the data volume. The area also includes sliders to control the eye-point and the projection screen location — there are sliders for the distance of the eye point from the center of the volume, the distance of the projection screen from the eye point, and the width of the projection screen. This functional area is the user's primary tool for navigating through the data volume, and so is placed at the top of the window.

The Acceptable Error area allows the user to specify the acceptable error to be used on the first refinement phase, the divisor to be applied to the acceptable error after each phase, and the value below which the acceptable error is to be forced to zero. Numeric input fields are used rather than sliders so as to allow the user full experimental latitude with respect to the numbers used.

In the Physical Screen area, the rendering window height and width can be specified, in pixels. The background color can be specified as an RGB triple.

The Command area at the bottom of the window consists exclusively of buttons which either toggle features or perform one-time actions. Run-length encoding compression is enabled or disabled using a toggle button — the button is neutral gray if compression is on, and bright yellow if it is off. Another toggle button controls whether or not the renderer pauses after each progressive refinement. The one-time command buttons allow the user to load a file, save the screen image, summon the color specification window, reset all parameters to default values, pause/continue rendering or exit the renderer. Ample room was left for additional command buttons or functional areas.

Legend Window. The color scale being used for rendering is presented to the user in a window adjacent to the rendering window (Figure 11). This allows the user

to see how the colors map to the data values. Note that the color ribbon consists of two halves. The left half shows the color uncorrected for opacity. This is what an infinitely thick volume of that data value would look like. The right side shows the color multiplied by its opacity. This shows what a single voxel with that data value would look like. These two views help the user understand the effect of opacity on color.

The legend window can also accommodate additional user cues, such as a data volume orientation icon, when these cues are implemented.

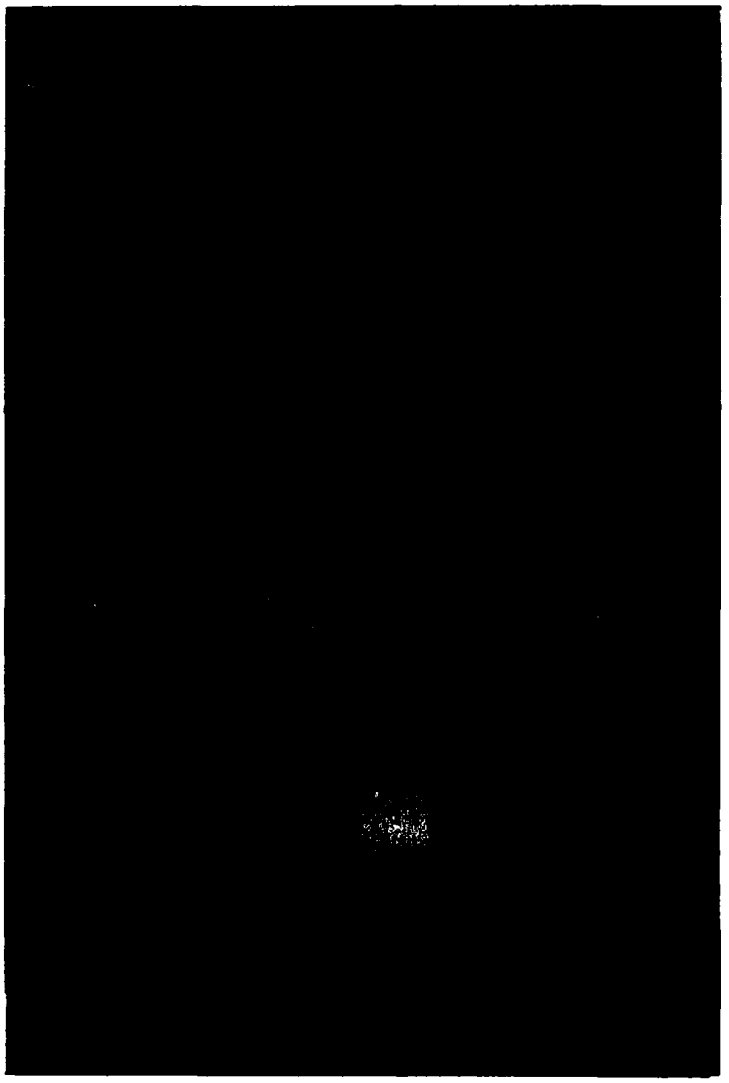


Figure 11. Legend Window.

Color Specification Window. The color specification window (Figure 12) allows the user to specify the operating parameters of the color transfer function. The inclusion of this feature was prompted by Rheingan's experimental evidence that user confidence in the interpretation of color gamuts is highest when the user can interactively control them [Rhei92].

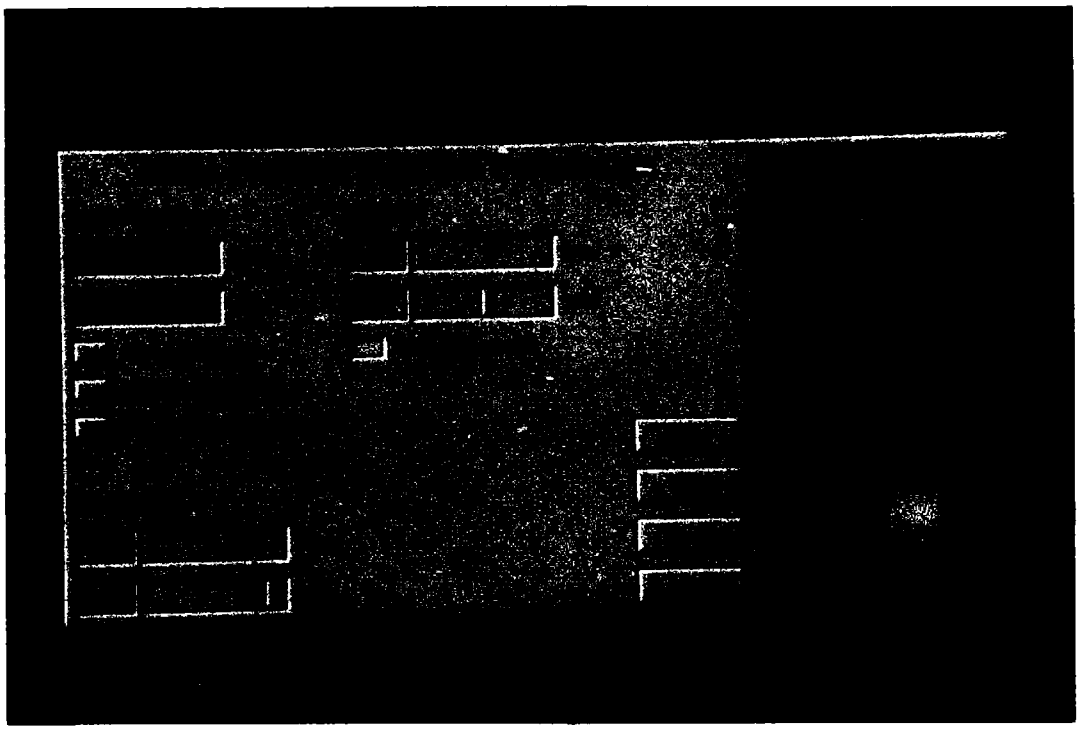


Figure 12. Color Specification Window.

The color specification window allows the user to specify the minimum and maximum data value range to be mapped by the color transfer function. Immediately below these input fields, the 'Descending Scale' toggle button controls whether the high end of the data values is mapped to the low or the high end of the hue scale. The 'Render voxels < min' and 'Render voxels > max' toggle buttons control whether or not data values outside the specified data value range are rendered. Two sliders control the

minimum and maximum opacity values, while a toggle button allows the user to map the high opacity to either the low or the high data value.

As the user changes the color transfer function parameters, the color scale indicator in the right-hand portion of the window changes in real time. This gives the user instant feedback as to how the specified color transfer function appears. The color transfer function is not applied to the octree until the user presses the 'Apply' button. When the 'Apply' button is pressed, the Legend window is updated with the new color parameters, the octree is re-built, and the progressive refinement rendering process is re-started. Alternatively, the user can choose to return to the previously used color parameters by pressing the 'Revert' button, or reset the color parameters to their default values by pressing the 'Reset' button.

The 'Hide' button dismisses the color specification window. This feature allows the user to reduce on-screen crowding.

VOXELIZE

VOXELIZE is a pre-processor for the Interactive Volume Renderer. It is implemented in ANSI C and uses no machine- or architecture-specific features. VOXELIZE reads RCS data files created by the Radar Cross Section - Basic Scattering Code (RCS-BSC) and generates IVR-format cuberille data files.

Input File. The RCS-BSC file is organized by azimuth and elevation angle. It has a record for every combination of elevation angle between 0 to 180 degrees step 0.5 degrees, and azimuth angle between 0 and 180 degrees, step 0.5sin(elevation). Each record consists of the RCS (in centibels over square meter) and phase angle of the return for each of four transmitter/receiver antenna polarization combinations: Horizontal/Vertical, Vertical/Horizontal, Vertical/Vertical, and Horizontal/Horizontal.

Wojszynski discusses the parameters used to generate the RCS data set, as well as a detailed discussion of the RCS-BSC file format [Wojs92].

General Approach. VOXELIZE takes the following approach to represent RCS data in a cuberille format. An emitter with the characteristics described in the RSC-BSC file is assumed to be at the center of the volume. This emitter can be visualized as emitting one *data ray* for each entry in the RCS-BSC file. This data ray has associated with it one attribute: the RCS in the ray's direction for the user-specified antenna polarization combination.

For each voxel in the volume, the data rays which intersect the voxel are averaged. The average is weighted by how long the ray's path inside the voxel is (i.e. a glancing ray weighs less than a ray through the center). The voxel is assigned a data value which is a function of that average. Unless otherwise specified by the user, that function is the identity function.

VOXELIZE's operating parameters can be specified through an extensive set of command-line options.

Filtering. VOXELIZE can be set to disregard data rays which are either below a specified RCS or above a specified RCS, or both. This feature was adopted from the pre-processor used by Tisdale in his RCS volume rendering work [Tisd92].

Spherical Masking. The user can request the blanking of all voxels which are exterior to the largest sphere which can be inscribed in the data volume. Blanked voxels are set to a null value and ignored by the volume renderer. This spherical masking option produces a data volume which appears spherical rather than cubical. However, this option guarantees that at least 47.6% of the voxels in the data volume will be blanked, leading to significant storage inefficiencies. This 47.6% minimum blanking factor is computed by subtracting the volume of the inscribed sphere from the volume of a cube.

Proportional Masking. The user can request the blanking of all voxels whose value divided by their distance from the volume's center are below a specified ratio. Using this feature, regions of large RCS will form longer 'rays' from the center of the volume than do the regions of smaller RCS.

Contour Coloring. The user can specify that the value of a non-blank voxel be strictly proportional to its distance from the center of the volume. Used in conjunction with the proportional shaping feature, this can produce an appearance much like a color-coded contour map.

Data Volume Size. The user can specify the dimensions of the data volume, in voxels. For simplicity of implementation, VOXELIZE only generates cubic data volumes — the dimensions of every side will be identical. The voxels will be considered to be unit cubes.

IV. Results

Introduction

The goals of this research was to implement an interactive volume renderer based on the Sample Buffer algorithm [Ke93], and apply it to the visualization of RCS data. These goal were accomplished. This section describes the observed behavior of the progressive refinement algorithm, the visualization techniques which were found to be most effective, and the performance of the system.

Progressive Refinement Algorithm Behavior

Figures 13 through 18 show the progressive refinement of an image. The subject is a head-on view of vertical/vertical RCS data, voxelized in the VOXELIZE default mode. In this default mode, detailed in Chapter 3, the value of a voxel is simply the weighted average of all RCS data rays which intersect it.

The sequence of figures shows how the resolution of the scenes rendered progresses from an extremely low first phase to a fully-detailed final phase as the error criterion is gradually tightened. Figure 13 shows the result of the first refinement phase. At this point, the acceptable error is 100. The root node of the octree has an error of 88.15, so no sub-division takes place. Therefore, all of the sample buffers remain as they were initialized, either null or pointing to the root node of the octant. The result is a rendering of the root node of the octree — a large cube whose color is the average color of the data volume.

Figure 14 shows the second refinement. The acceptable error is now 50, and some sample record sub-division has begun to take place. The areas of large data value deviation are already becoming apparent because of the higher degree of sub-division in those areas. The areas of the image which will be sparse are also becoming apparent.

Figure 15 is the third refinement. Acceptable error is 25. There are very few large cubes left, and a good working approximation of the final image is already displayed.

Figure 16 is the fourth refinement. Acceptable error is 12.5. The difference between this image and the previous is not as dramatic as the previous changes. The algorithm is now just filling in small details. This trend continues through Figure 17, the fifth refinement, with an acceptable error of 6.25.

Figure 18 is the final image, rendered at an acceptable error of zero. The aliasing artifacts of the VOXELIZE program stand out clearly in this figure.

Effective Visualization Techniques

The Interactive Volume Renderer, in conjunction with its VOXELIZE pre-processor, presents a wide variety of options for visualizing RCS data. However, some experimentation with these options was required to achieve informative visualizations. The ability to interactively modify the color transfer function parameters proved very useful during this trial and error process.

Figure 19 shows vertical/vertical RCS data, viewed head-on, voxelized using the spherical masking, proportional shaping, and contour-coloring VOXELIZE features. While some regions of low RCS can be made out, regions of particularly high RCS do not stand out as well as they do in the default voxelization.

Figure 20, 21 and 22 show three views of the vertical/vertical RCS data, voxelized using the proportional shaping and spherical masking options. This is the most effective visualization developed for this system. Color cues help make regions of relative highs and lows stand out. The overlapping opacities provide some limited depth cueing. However, this visualization still suffers from inadequate depth-cueing.

Figures 23 through 26 are visualizations of different antenna polarity RCS data using the same voxelizing and color scale as the previous figures. Figures 23 and 24 present head-on and side views of the horizontal/horizontal RCS data. Figures 25 and 26 show head-on and side-views of the horizontal/vertical RCS data.

Performance of the Interactive Volume Renderer

Table 1 below summarizes some performance metrics collected during the generation of Figures 13 through 18. The image in question had a size of 256 by 256 pixels. Only 29.5% of the pixels were part of the data volume projection. The remaining 70.5% of the pixels were part of the background and demanded relatively little computation. The data volume was a 65x65x65 grid.

Table 1. Progressive Refinement Performance Metrics

Phase	Rendering Time (seconds)	Average sample records per pixel	Percent of exact sample records	Memory in use (Megabytes)
1	1.65	1	0.00%	19.3
2	5.56	6.29	0.00%	30.0
3	12.9	20.1	0.72%	46.6
4	31.8	36.7	10.17%	66.6
5	40.9	29.5	25.42%	52.1
6	26.4	1	100.00%	17.0

Note the effect of the run-length encoding compression on these metrics. Memory use peaks in phase 4, then decreases as an increasing portion of the sample records becomes type-exact. By the end of the final phase, all of the sample records are type-exact, and every sample buffer consists of only one run-length encoded entry.

However, the peak amount of memory required to render a 256-by-256 pixel image in this case is 66 megabytes. This exceeds the main memory available in all but one of the workstations in the AFIT Graphics Laboratory. Performance plummets if the Interactive Volume Renderer must resort to swapping-in virtual memory.

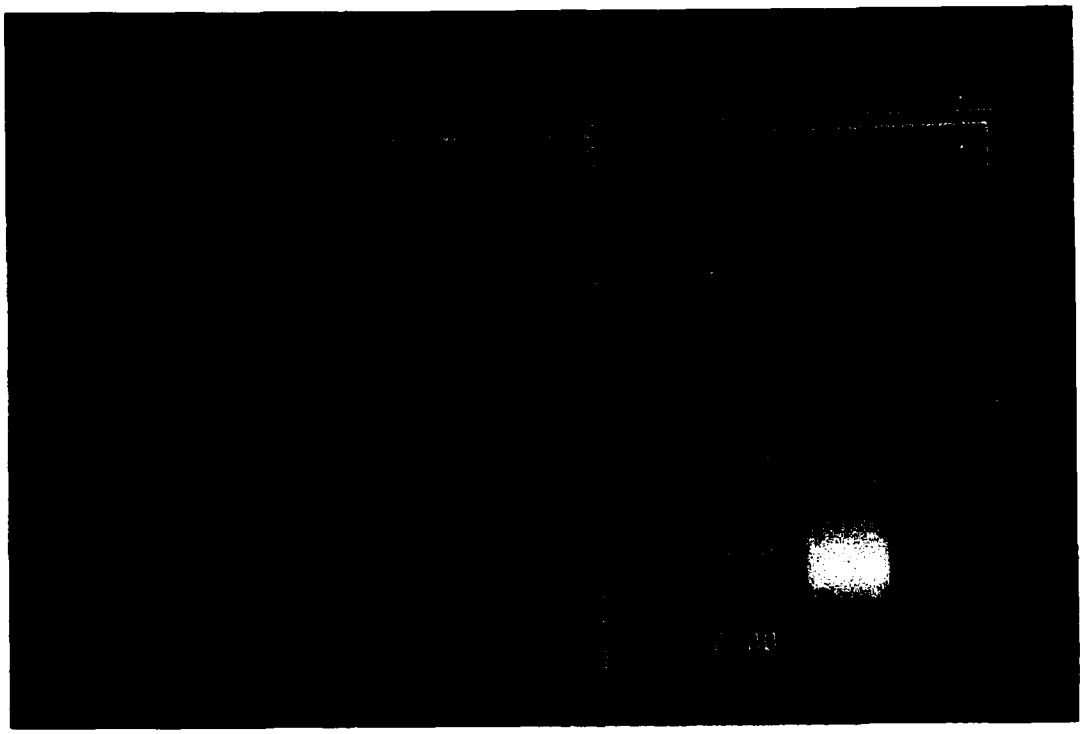


Figure 13. Vertical/Vertical RCS, Default Voxelization, Head-On View, Phase 1 of 6

Figure 14. Vertical/Vertical RCS, Default Voxelization, Head-On View, Phase 2 of 6

Figure 15. Vertical/Vertical RCS, Default Voxelization, Head-On View, Phase 3 of 6

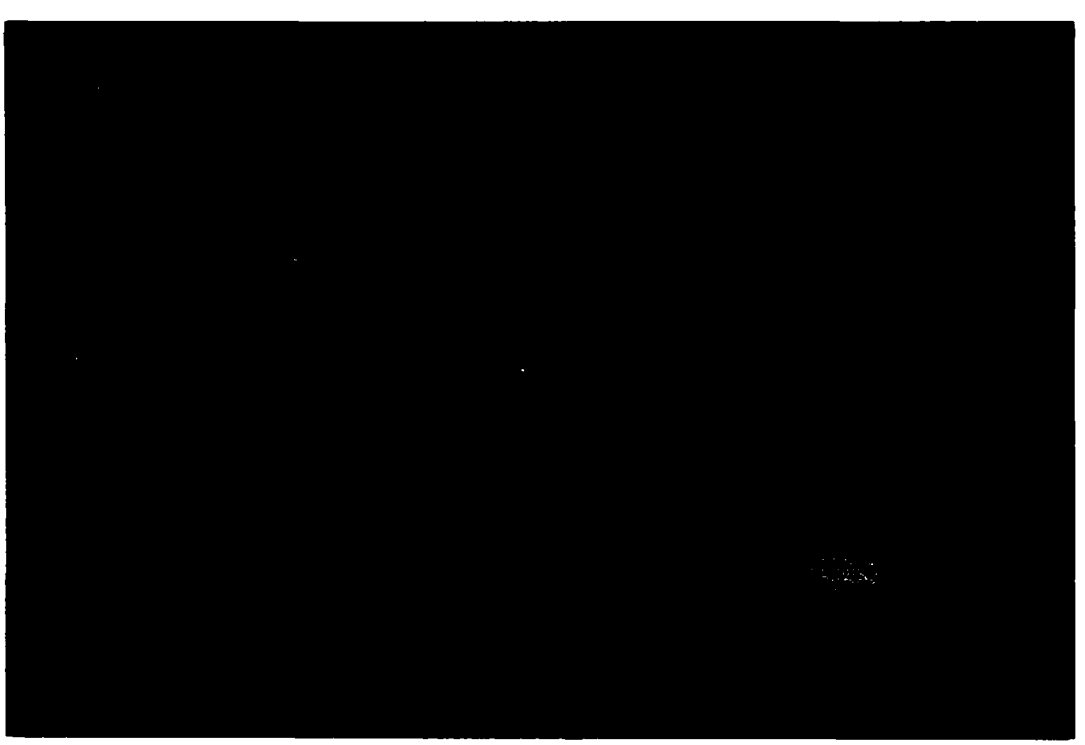


Figure 16. Vertical/Vertical RCS, Default Voxelization, Head-On View, Phase 4 of 6

Figure 17. Vertical/Vertical RCS, Default Voxelization, Head-On View, Phase 5 of 6

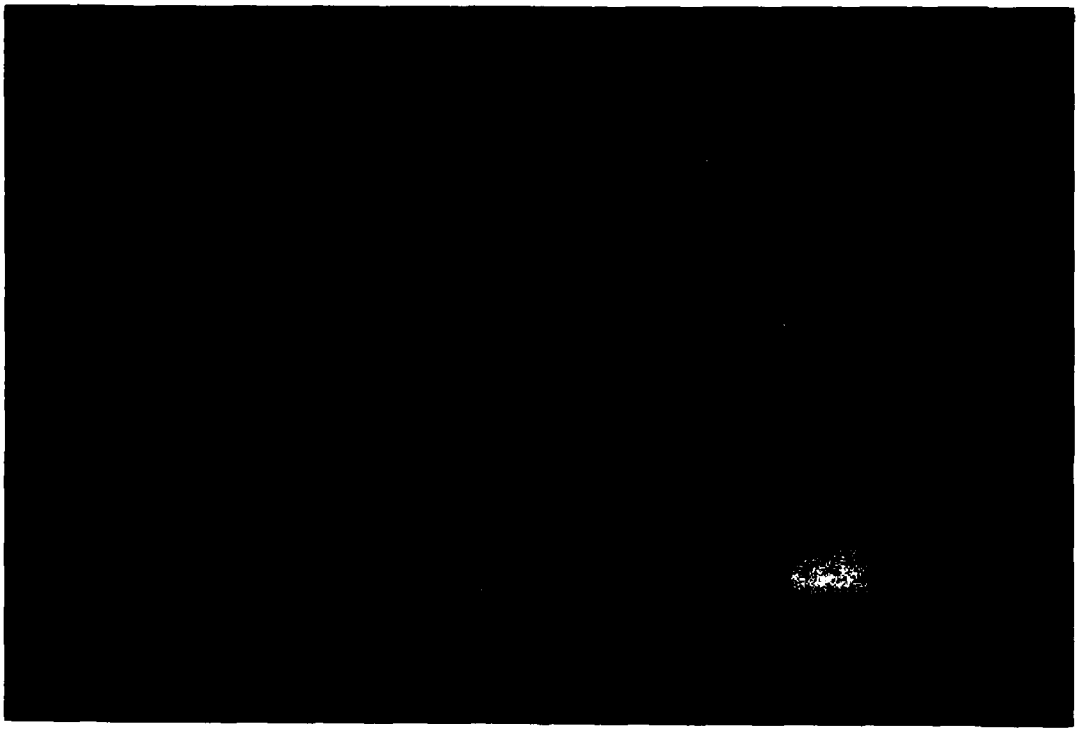


Figure 18. Vertical/Vertical RCS, Default Voxelization, Head-On View, Phase 6 of 6

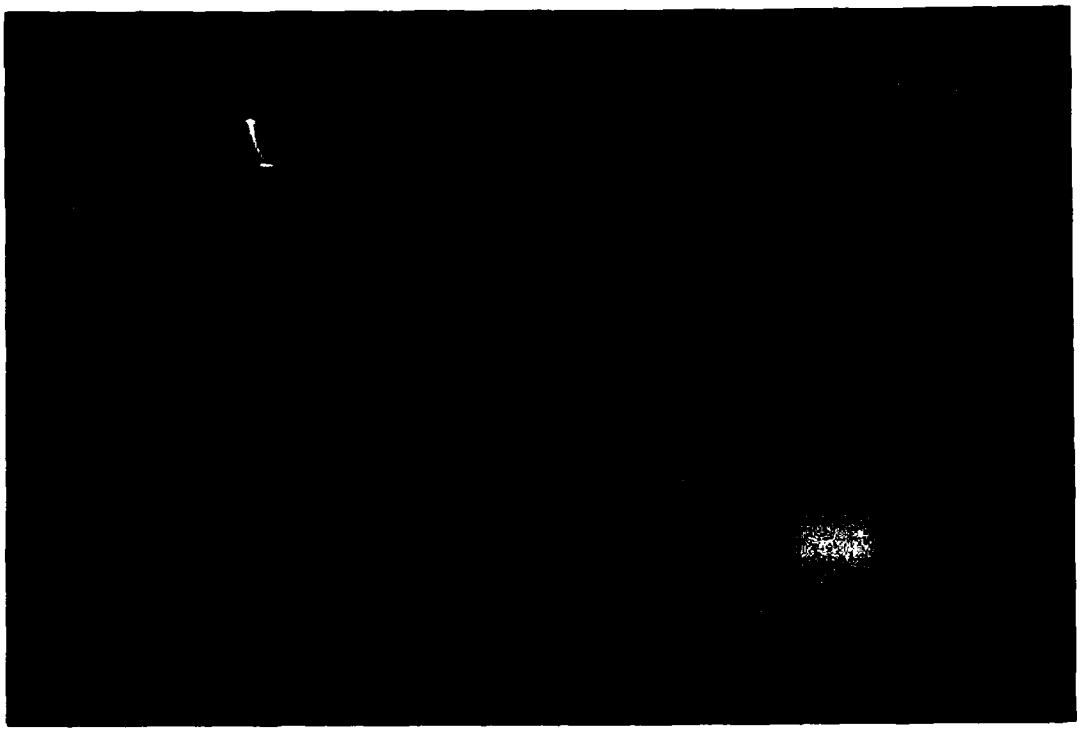


Figure 19. Vertical/Vertical RCS, Contour-Colored, Head-On View

Figure 20. Vertical/Vertical RCS, Proportional, Head-On View

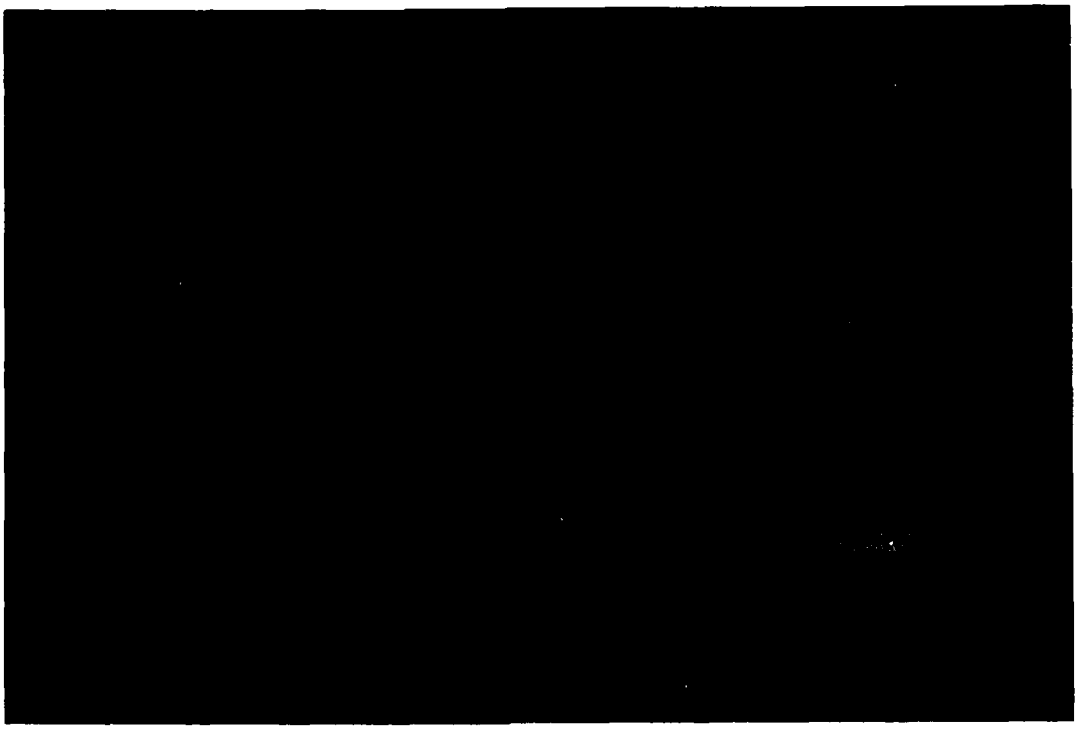


Figure 21. Vertical/Vertical RCS, Proportional, Side View

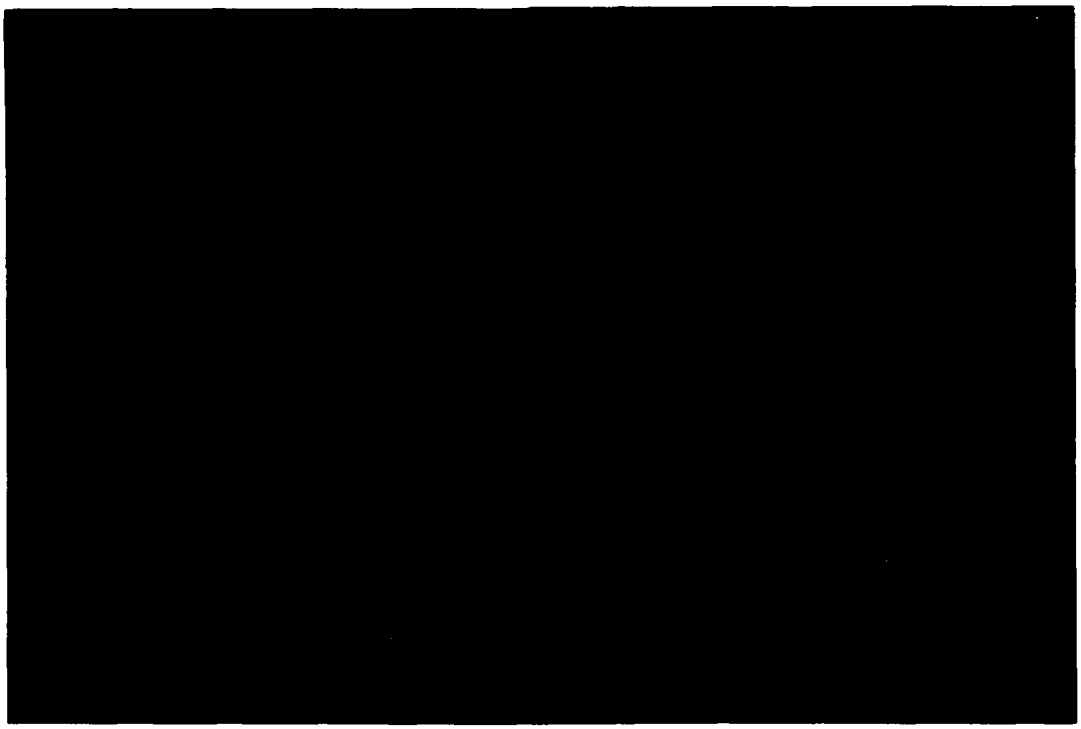


Figure 22. Vertical/Vertical RCS, Proportional, God's-Eye View

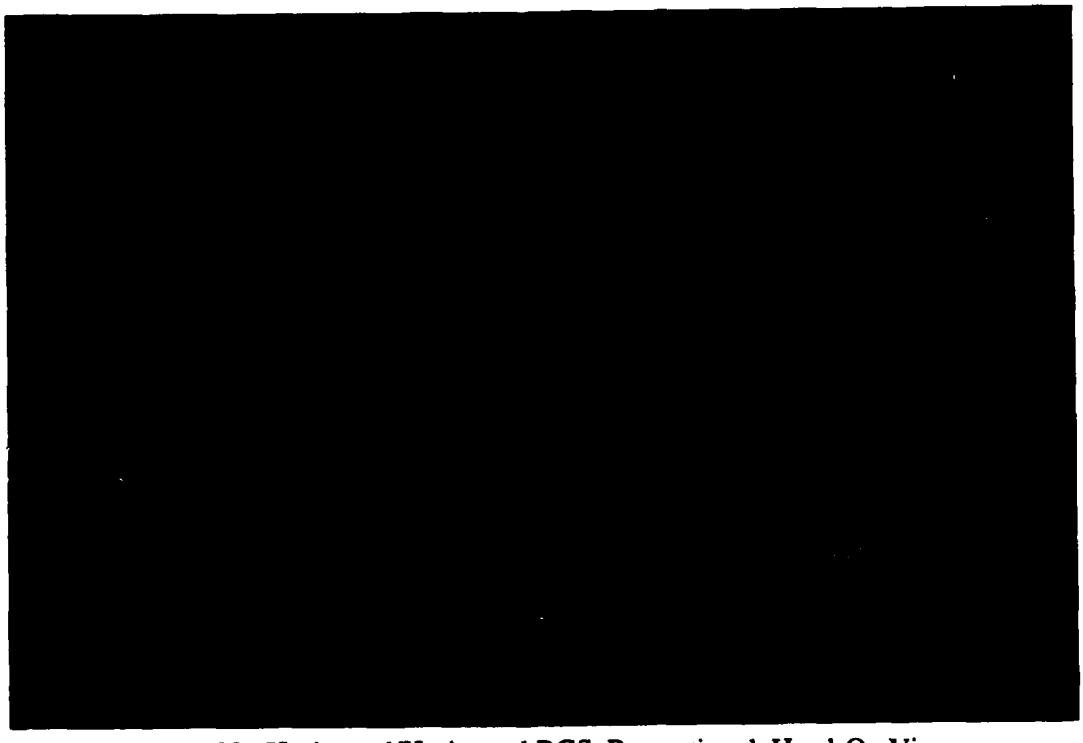


Figure 23. Horizontal/Horizontal RCS, Proportional, Head-On View

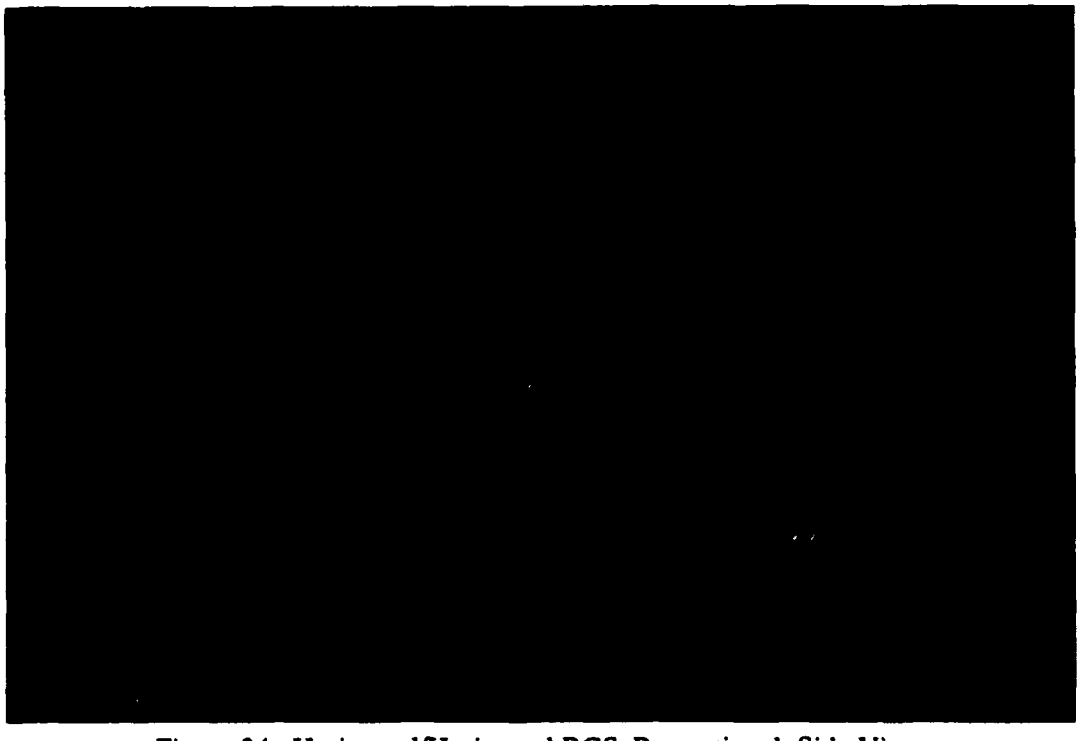


Figure 24. Horizontal/Horizontal RCS, Proportional, Side View

Figure 25. Horizontal/Vertical RCS, Proportional, Head-On View

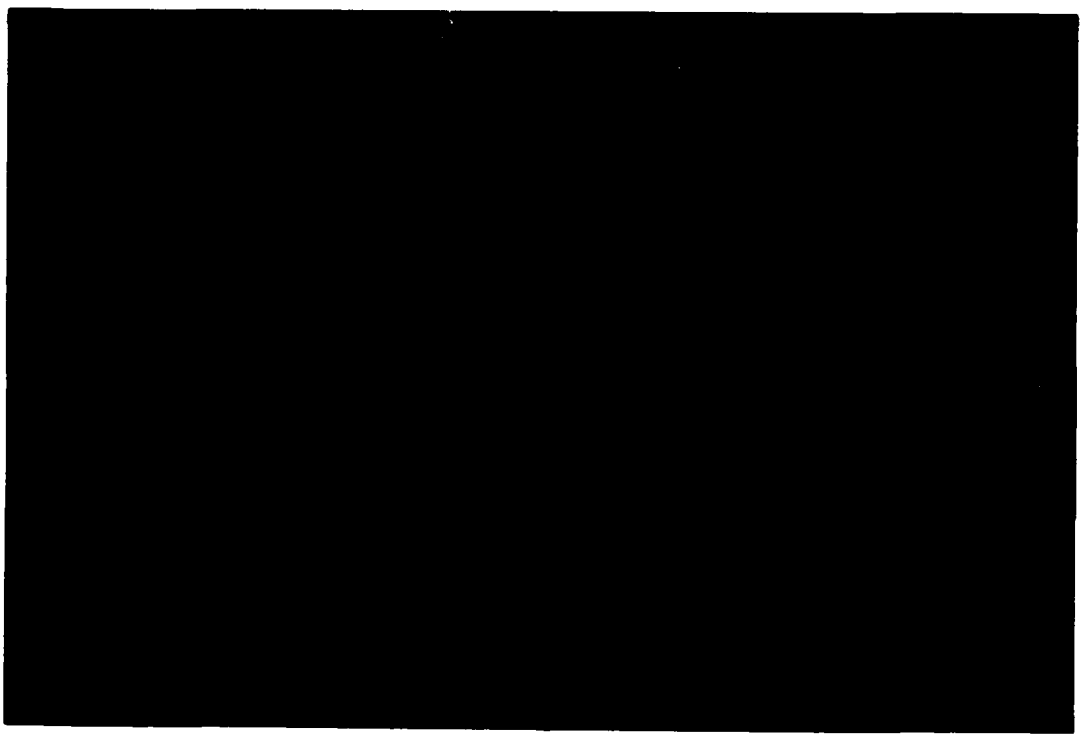


Figure 26. Horizontal/Vertical RCS, Proportional, Side View

V. Conclusions

Assessment of Results

This research has successfully developed a fast volume renderer prototype for RCS data. It uses the recent Sample Buffer algorithm [Ke93] to render initial coarse images at near-interactive speeds. The renderer then progressively refines the image to fine resolution in a time on the order of minutes.

However, the system demands huge amounts of memory resources in order to draw relatively small images. It is effectively restricted to running on machines equipped with 128 megabytes or more of main memory. However, optimization of the rendering algorithm's data structures (*e.g.* by reducing the size of the sample record) could alleviate this problem.

The system also suffers from an inadequate RCS data pre-processor. The VOXELIZE pre-processor as now implemented cannot support the production of visualizations on a par with previous AFIT RCS visualization research in particle systems and surface rendering [Tisd92, Wojs92].

Future Work

This thesis suggests a large amount of promising additional research. The current effort has done little more than implement a prototype. Some potential areas of follow-on research are:

Sample Buffer Algorithm Optimization. The implementation of the algorithm in this prototype is very inefficient in parts and would benefit greatly from optimization.

To name just one example, the run-length encoding compression routines repeat expensive compositing operations that have just been completed by the rendering routines.

Multi-Polar Visualizations. Adding the capability to visualize multi-polar RCS data using a multi-variate color scheme. The HSV color scale used in the current color transfer function has two unused dimensions that could be used to encode phase and polarity data.

Improved RCS Voxelization Pre-Processor. As noted above, the current voxelizer is deficient. Better paradigms for voxelizing RCS data must be developed. More statistically sophisticated methods of sampling the RCS data may be appropriate. The rendering-speed-versus-resolution trade-off also needs to be analyzed.

Distributed Implementation. Distributed or parallel rendering of the data has the potential for yielding significant speed-up. Hardware already in-place at the AFIT Graphics Laboratory could support this research.

Virtual Environment. A long-term research goal may be the development of an immersive, interactive RCS visualization system, similar to the Virtual Wind Tunnel developed by Bryson [Brys91]. In such a system, the visualization system could be linked directly to the RCS data simulator The user could modify a simulated radar target and view the changes to the RCS in real time.

General Purpose Rendering. The Interactive Volume Renderer is not restricted to RCS data — it can serve as a general purpose volume renderer for any type of cuberille data. As such, it is a candidate for further research in a wide variety of fields.

Appendix 1: IVR User's Manual

Introduction

IVR stands for Interactive Volume Renderer. It uses a progressive refinement rendering algorithm, coupled with a graphic user interface, to provide fast and interactive image generation.

The remainder of this document gives you detailed instructions on how to operate IVR, but here's how it works in a nutshell: You can start IVR by simply invoking it from the command line — no parameters are mandatory. Once IVR is started, three windows will open. A blank rendering window, a *legend* window which will show you the current color scale, and a *control* window, which is where you can change IVR's parameter. Once you load a file by clicking the 'Load File' button in the control window, IVR starts volume rendering it. You'll see the image being drawn into the rendering window, progressively getting more and more detailed, until it's finally fully accurate. At any time, you can change any parameters in the control window, and IVR will automatically update the rendering window accordingly. When you're done, click the 'Exit' button. That's all you need to know to get started in IVR.

Input File Format

IVR accepts cuberille data files. Its file syntax is as follows:

- Line 1 is a comment. The comment is echoed when IVR reads the file in, but otherwise, it is completely ignored. The comment may be up to 254 characters long and must be terminated with a newline character.
- After the comment line, IVR expects nothing but numbers. The numbers must be separated by whitespace — tabs, blanks or newline characters. The numbers it expects are:
- The size of the grid (in voxels) along the X dimension
- The size of the grid (in voxels) along the Y dimension
- The size of the grid (in voxels) along the Z dimension
- A blanking flag. If you wish to flag a particular data value as meaning 'blank voxel', then set this blanking flag to some non-zero integer, otherwise set it to zero.
- A blank value. If you have a non-zero blanking flag, then this entry specifies the data value which will be considered 'blank.' If you set the blanking flag to zero, some dummy number must be present, but its value is ignored.
- The data values for the voxels. These must be specified in z-then-y-major order. That is, they should be in the same order as they would be if output by a code fragment such as :

```
for z
  for y
    for x
      print value[x,y,z]
```

Command Line Options

IVR has two command lines. To see them on-line, just type `ivr -?`

The two command line options are:

`-i` *<number>* This option specifies how verbose IVR will be with its status messages.

0 - No status messages at all

1 - Minimal status messages

2 - Full status messages

3 - Instrumentation and debugging messages.

`-v` *<voxel_file>* Specifies a voxel file to be opened immediately upon program start.

Control Window

The control window is IVR's cockpit. From this window, you can control a wide variety of simulation parameters, and immediately see what effect your changes had.

The first thing you'll notice is that the control window is split up into four color-coded areas. The four areas are the Viewing Geometry area, the Acceptable Error area, the Physical Screen area, and the Command area. We'll examine what each of these control areas does.

Viewing Geometry Area . This is where you'll find the controls that affect your viewpoint and perspective into the scene. The *rotation* sliders let you turn the scene about the three axes. The *translation* sliders move the data volume back and forth along the three axes. The *scale* sliders shrink or grow the data volume along the three axes.

You control your perspective with the *eye-point distance* and the *projection screen* sliders. If you increase your eye-point distance, the data volume will appear further away. Similarly, by making the projection screen wider, you can see more of the

scene, but things will look more smaller. If you increase the screen distance, it's like zooming in with binoculars — things will start looking bigger.

Acceptable Error Area. The Acceptable Error area allows you to control how IVR progressively renders the scene. For technical information on how progressive refinement works, please refer to chapter 3 of this thesis. The *initial* acceptable error is used on the first refinement pass. After each pass, the acceptable error is divided by the *divisor*. When the acceptable error drops *below* a certain value, it is forced to zero, and the scene is drawn at full resolution. The numeric input fields in the Acceptable Error area allow you to control all of these parameters.

Physical Screen Area. In the Physical Screen area you can change the parameters that control the appearance of the rendering window. The rendering window's height and width in pixels can be specified by two numeric input fields. Below that, you can specify the color of the rendering window's background. You do this by specifying a number between 0 and 255 for the background color's red, blue, and green components.

Command Area. The Command area at the bottom of the window allows you to enter IVR commands or toggle in and out of modes.

The *Compression* button controls whether or not run-length encoding is enabled (reference chapter 3). For best results, leave compression on all the time — it saves both memory and CPU time. Note that the Compression button has a bright yellow spot on it if compression is on, and that the spot turns neutral gray if compression is off. All toggle buttons in this area behave the same way.

The *Pause after each refinement* button controls whether or not IVR pauses after each progressive refinement. This is a toggle button too.

The *Save Picture* button lets you save the rendering window to a Utah RLE format graphics file. Just push the button when you want to save the rendering window's contents, and IVR will prompt you for a destination file name.

The *Load File* button lets you load in a new cuberille data file. If a file is already loaded, the new one will supersede it.

The *Set Colors* button brings up the Color Specification window, discussed in the next section. The Color Specification window lets you change the way the scene is colored.

The *Reset* button sets all the parameters back to the values the had when IVR started up. This is handy if you have changed the parameters so much you can't find the data volume anymore. Note that the Reset button does NOT reset the color parameters. There's a different button in the color specification window to do that.

The *Pause* button works just like you'd expect. Hit it once, and the renderer stops drawing. Hit it again, and the renderer continues. The renderer is paused when the pause button shows a yellow light.

The *Exit* button terminates IVR, no questions asked.

Color Specification Window

The color specification window allows you to specify how the data values in the voxel file are mapped to colors and opacities. This is a powerful tool for data exploration. In this section, we'll show you how the various controls in the color specification window work.

Apply Button. The first thing you should keep in my mind is that all none of the changes you make in the color specification window go beyond the window until and unless you hit the *Apply* button. The changes you make will be reflected instantly in the

color scale *inside* the color specification window, but nothing outside the window will change until you hit the Apply button. The window was designed this way because changes to the color scale can take up to a few minutes to implement.

Data Value Range. The data value range controls allows you to specify the minimum and maximum data values to be mapped by the color transfer function.

The *Descending Scale* toggle button, when active, maps the lowest data value onto the highest color value, and vice-versa.

The *Render Voxels > max* toggle button, when active, tells the renderer to draw voxels even though they are over the maximum data value. Those voxels will be drawn in the same color as the maximum data value.

The *Render voxels < min* button, when active, tells the renderer to draw voxels even though they are below the minimum data value. Those voxels will be drawn in the same color as the minimum data value.

Opacity. Two sliders control the minimum and maximum opacity values. Note that the sliders enforce mutual consistency. The minimum slider can never be more than the maximum slider, not even for a moment.

The *Descending scale*toggle button , when enabled, causes the high data values to be mapped to the high opacity values.

HSV Value. You can control the range of colors used in the color scale using the *HSV Value* sliders.

Command Buttons. You choose to return to the previously used color parameters by pressing the *Revert* button. Alternatively, you can return to the start-up default color settings by pressing the *Reset* button.

The 'Hide' button closes the color specification window. Use this feature if you want a less crowded screen. You can always bring the color specification window back, color settings unchanged, by hitting the *Set Color* button in the Control window.

Appendix 2: VOXELIZE User's Manual

Introduction

VOXELIZE is a pre-processor for the Interactive Volume Renderer. It reads RCS data files created by the Radar Cross Section - Basic Scattering Code (RCS-BSC) and generates IVR-format cuberille data files. For a discussion of VOXELIZE's theory of operation, please refer to chapter 3 of this thesis.

Command Line Options

VOXELIZE's operating parameters can be specified through an extensive set of command-line options. The command line options are explained in Table 2 below.

Table 2. VOXELIZE Command Line Options

Option	**Explanation**
-i <*level*>	**Information level.** Controls the level of detail of status messages. 0 - No status messages 1 - Minimal status messages 2 - Full status messages 3 - Debugging messages
-l <*low-bound*> -h <*high-bound*>	**Filtering.** Disregards data rays which are either below a specified RCS or above a specified RCS, or both.

-s	**Spherical Masking**. Blanks all voxels which are exterior to the largest sphere which can be inscribed in the data volume. Blanked voxels are set to a null value and ignored by the volume renderer. This spherical masking option produces a data volume which appears spherical rather than cubical.
-p <*low-bound*> <*high-bound*>	**Proportional Masking**. Blanks all voxels whose value divided by its distance from the volume's center is below a specified ratio. Using this feature, regions of large RCS will form longer 'rays' from the center of the volume than regions of smaller RCS. Voxels at the *low-bound* can only be found at the very center of the volume. Voxels at the *high-bound* may exist anywhere inside the spherically-masked volume. Values in between the low- and high-bounds have maximum altitudes computed linearly between 0 and 1/2 of the data volume's width.
-c	**Contour Coloring**. Specifies that the value of a non-blank voxel is strictly proportional to its distance from the center of the volume. Used in conjunction with the proportional shaping feature, this can produce an appearance much like a color-coded contour map.
-r <*filename*>	**RCS-BSC Input File Name**. Default is stdin
-b <*blank-value*>	**Blank Voxel Value.** The special data value to be used as a flag for a blank voxel. Default is 0.0. Value may not have more than 3 significant figures.
-v <*filename*>	**Output File**. Specifies the output voxel file. Default is stdout.
-s <*Size*>	**Size of Data Volume**. Specifies the number of voxels on a side of the output data volume. VOXELIZE only produces cubic volumes. Minimum is 2, maximum is 1024, default is 65.
-t <*Type*>	**Type of Data**. Specifies the type of RCS or phase data to be used to build the output file. Options are: 0 - Vert/vert RCS [default] 1 - Vert/vert phase angle 2 - Horiz/horiz RCS 3 - Horiz/horiz phase angle 4 - Horiz/vert RCS 5 - Horiz/vert phase angle 6 - Vert/horiz RCS 7 - Vert/horiz phase angle

Bibliography

[Brid88] Bridges, David J. *Volume Rendering Techniques for the Display of Three-Dimensional Aerodynamic Flow Field Datas*. MS thesis, AFIT/GCS/ENG/88D-2. School of Engineering, Air Force Institute of Technology (AU), Wright Patterson Air Force Base, OH, December 1988.

[Brys91] Bryson, Steve, and Creon Levitt. "The Virtual Wind Tunnel: An Environment for the Exploration of Three-Dimensional Unsteady Flows," *Proceedings of the 1991 IEEE Visualization Conference.* 17-24. Los Alamitos, California: IEEE Press, 1991.

[Dreb88] Drebin, Robert A., Loren Carpenter, and Pat Hanrahan. "Volume Rendering," *Computer Graphics, 22:4* 65-74 (August 1988).

[Elvi91] Elvins, T. Todd, and David R. Nadeau. "NetV: An experimental Network-based Volume Visualization System," *Proceedings of the 1991 IEEE Visualization Conference.* 239-245. Los Alamitos, California: IEEE Press, 1991.

[Elvi92] Elvins, T. Todd. "Volume Rendering on a Distributed Memory Parallel Computer," *Proceedings of the 1992 IEEE Visualization Conference.* 93-98. Los Alamitos, California: IEEE Press, 1992.

[Fole90] Foley, Thomas A., and David A. Lane. "Visualization of Irregular Multivariate Data," *Proceedings of the 1990 IEEE Visualization Conference.* 247-254. Los Alamitos, California: IEEE Press, 1990.

[Foul92] Foulke, Kenneth, and Thomas E. Reinkuser. "Aircraft Susceptibility: Controlling Radar Signature," *Aerospace America, 30:* 44-46 (August 1992).

[Fuch92] Fuchs, Henry, Marc Levoy, and Stephen M. Pizer. "Interactive Visualization of 3D Medical Data," in Visualization *is Scientific Computing*. Ed. Gregory M. Nielson and Bruce Shriver. 140-146. Los Alamitos, California: IEEE Press, 1990.

[Fuji86] Fujimoto, Akira, Takayuki Tanaka, and Kansei Iwata. "ARTS: Accelerated Ray-Tracing System," *IEEE Computer Graphics and Applications*, 16-26 (April 1986).

[Garg86] Gargantini, I., T.R.S. Walsh and O.L. Wu. "Displaying a voxel-based object via linear octtrees," *SPIE:626:* 460-466 (1986).

[Glas84] Glassner, Andrew S. "Space Subdivision for Fast Ray Tracing," *IEEE Computer Graphics & Applications:* 15-22 (October 1984).

[Hibb90] Hibbard, Bill, and Dave Santek. "The VIS-5D System for Easy Interactive Visualization," *Proceedings of the 1990 IEEE Visualization Conference.* 28-35. Los Alamitos, California: IEEE Press, 1990.

[Kauf91] Kaufman, Arie. "Introduction to Volume Visualization," in *Volume Visualization.* Ed. Arie Kaufman. 1-18. Los Alamitos, California: IEEE Computer Society Press, 1991.

[Kauf93] Kaufman, Arie. Volume Visualization Course Notes, presented at SIGGRAPH '93, Anaheim, California. August 1993.

[Ke93] Ke, Hao-Ren, and Ruei-Chuan Chang. Sample Buffer: A Progressive Refinement Ray-Casting Algorithm for Volume Rendering. *Computers & Graphics, 17:* pp 227-283 (1993).

[Knot85] Knott, Eugene F., John F. Shaeffer, Michael T. Tuley. *Radar Cross Section: Its Prediction, Measurement and Reduction.* Dedham, Massachusetts: Artech House, 1985.

[Knot87] Knott, Eugene F. "EM Waves and the Reflectivity Process," in *Principles of Modern Radar.* Jerry L Eaves and Edward K. Reedy, editors. New York: Van Nostrand Reinhold Company, 1987.

[Laur91] Laur, David, and Pat Hanrahan. "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering," *Computer Graphics: 25,* 285-289 (July 1991).

[Levo90] Levoy, Marc. "Efficient Ray Tracing of Volume Data," *ACM Transactions on Graphics:9,* 245-261 (July 1990).

[Rhei92] Rheingans, Penny. "Color, Change, and Control for Quantitative Data Display," *Proceedings of the 1992 IEEE Visualization Conference.* 252-259. Los Alamitos, California: IEEE Press, 1992.

[Same91] Samet, Hanan, and Robert E. Webber. "Hierarchical Data Structures and Algorithms for Computer Graphics," *IEEE Computer Graphics & Applications.* 48-68 (May 1988).

[Shu92] Shu, Reuben, Alan Liu. "A Fast Ray Casting Algorithm Using Adaptive Isotriangular Subdivision," *Proceedings of the 1991 IEEE Visualization Conference.* 232-238. Los Alamitos, California: IEEE Press, 1991.

[Styt91] Stytz, Martin R., Gideon Frieder and Ophir Frieder. "Three-Dimensional Medical Imaging: Algorithms and Computer Systems," *ACM Computing Surveys, 23(4):* 422-499 (December 1991).

[Tisd92] Tisdale, David Jesse. *Methods for Viewing Radar Cross Section Data in Three Dimensions.* MS thesis, AFIT/GCS/ENG/92D-18. School of Engineering, Air Force Institute of Technology (AU), Wright Patterson Air Force Base, OH, December 1992.

[Will92] Williams, Peter L. "Interactive Splatting of Nonrectilinear Volumes," *Proceedings of the 1992 IEEE Visualization Conference*. 37-44. Los Alamitos, California: IEEE Press, 1992.

[Wojs92] Wojszynski, Thomas George. *Scientific Visualization of Volumetric Radar Cross Section Data*. MS thesis, AFIT/GCS/ENG/92D-21. School of Engineering, Air Force Institute of Technology (AU), Wright Patterson Air Force Base, OH, December 1992.

[Yage93] Yagel, Roni. Volume Visualization Course Notes, presented at SIGGRAPH '93, Anaheim, California. August 1993.

Vita

Captain Alain L. M. Jones was born on January 13, 1966 in La-Chapelle-Saint-Mesmin, France. He graduated from Hendrick Hudson High School in Montrose, New York in 1983 and accepted an appointment to the United States Air Force Academy. He graduated in 1987 and received a Bachelor of Science in Computer Science. His first assignment was to the 82nd Student Squadron at Williams Air Force Base, Arizona. In 1988, Captain Jones was assigned to the 1002d Space Systems Squadron at Falcon Air Force Base, Colorado, where he served first as a Database Administrator and then as Squadron Executive Officer. In 1991, he transferred to the 2nd Space Wing at Falcon Air Force Base, where he worked as a Logistics Plans and Programs Officer for the Global Positioning System Control Segment. In May of 1992, Captain Jones entered the School of Engineering, Air Force Institute of Technology, to pursue a Master of Science Degree in Computer Engineering.

Permanent Address: 5301 Witham Court
Tampa, Florida 33647

December 1993 Master's Thesis

RADAR CROSS SECTION VISUALIZATION USING SAMPLE BUFFER PROGRESSIVE REFINEMENT VOLUME RENDERING

Alain L. M. Jones, Capt, USAF

Air Force Institute of Technology, WPAFB OH 45433-6583

AFIT/GCS/ENG/93D-13

ESC/YVM
Hanscom AFB, MA 01731

This study developed a prototype for an interactive radar cross section visualization software system. The system, hosted on a Silicon Graphics workstation, is intended to support aircrews, mission planners, aircraft designers, and others who require an understanding of aircraft radar cross section characteristics. The input to the system is a set of radar cross section samples taken at various aspect angles. A pre-processor developed as part of this study transforms the input radar cross section data into a three-dimensional cuberille data volume. This data volume is then visualized using an interactive volume renderer. The interactive volume renderer implements progressive image refinement using the Ke & Chang Sample Buffer algorithm. A graphic user interface allows users to modify rendering parameters and see the results of their changes in near-real-time.

14 SUBJECT TERMS
Volume Rendering, Scientific Visualization, Radar Cross Sections, Stealth Technology, Computer Graphics

66

17 SECURITY CLASSIFICATION OF REPORT	18 SECURITY CLASSIFICATION OF THIS PAGE	19 SECURITY CLASSIFICATION OF ABSTRACT	20 LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL