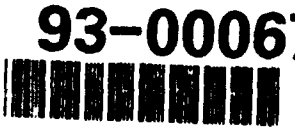
AFIT/GCS/ENG/92D-18

METHODS FOR VIEWING RADAR CROSS SECTION DATA IN THREE DIMENSIONS

THESIS

David Jesse Tisdale
Captain, USAF

AFIT/GCS/ENG/92D-18



AFIT/GCS/ENG/92D-18

METHODS FOR VIEWING RADAR CROSS SECTION DATA IN THREE DIMENSIONS

THESIS

Presented to the Faculty of the 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

David Jesse Tisdale, B.S.

Captain, USAF

December, 1992

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank all of the people who made it possible for me to finish my work here at AFIT and provided support along the way. The order of their acknowledgment in no way indicates their contribution or significance, they each helped in their own ways.

First, I'd like to thank my thesis advisor, Lt Col Marty Stytz, for his help throughout this thesis effort. Without his guidance and encouragement, this study would have been substantially diminished in its scope.

I'd also like to thank Lt Col Phil Amburn, who provided many fun and challenging hours in graphics research. His enjoyment of graphics always provided an inspiration, and he was invariably there with fresh insight when I needed it.

Special thanks go to Dr. Andy Terzuoli who helped me understand and explain electromagnetics in terms appropriate to this effort.

Finally, I would like to thank the friends I have made here at school as well as those I knew prior to AFIT, but were there to provide support and encouragement the last eighteen months. I know you've put up with a lot, and I appreciate it. This thesis is dedicated to Max and Saetee. Thank you for all you have given me.

ABSTRACT

Radar Cross Section data is an important factor in the design of modern fighter and bomber aircraft. Minimization of the reflected radar energy is one of the key issues when choosing shapes and materials in new aircraft. Visualization of the energy scattered back to a radar is neither intuitive nor easy. The mission planner and pilot need to gain an understanding of the vulnerabilities inherent in the design of the systems they use. The advent of relatively low cost graphics workstations has made their use affordable in applications inconceivable only a few short years ago.

This thesis examines three graphics rendering techniques and their applicability to the display of three-dimensional radar cross section data. This study looks at three tools, PSR, a particle system renderer; the Satellite Modeler, a three-dimensional surface renderer; and VIPER, a volumetric renderer for their applicability, utility, and ease of use in the display of radar cross section data.

Each of the systems was tested using the same data set, and the results compared later in this treatise. Although the purpose of this study is to determine the applicability of different techniques, the results can be used to further develop fast, efficient rendering systems for the visualization of radar cross section data.

METHODS FOR VIEWING RADAR CROSS SECTION DATA IN THREE DIMENSIONS

I. INTRODUCTION

1.1 Background.

One of the key factors in the design of modern aircraft is the application of stealth or low observable (LO) technology. The purpose of this technology is to reduce the chance of detection and increase survivability of an aircraft while in hostile airspace. Three recent airplanes incorporating stealth technology are the B-2 bomber, the F-22 fighter and the F-117 stealth fighter. The skins of all three aircraft are composed of a myriad of surface shapes and materials designed to reduce their Radar Cross Section (RCS).

The study and analysis of radar cross section data are important in both the design and deployment phases of military aircraft. While still in the conceptual phase, aircraft designs must minimize reflected radar energy, while at the same time, maximize performance characteristics. During the operational phase, the mission planners must take into account the vulnerability generated by the aircraft's radar cross section in relation to threat systems when routing sorties.

Two factors affecting the amount of energy reflected back to a radar and hence the RCS are the frequency of the pulse and the polarity of the wave with respect to the antenna. The frequency of the pulse varies greatly among the different radar systems, with the centers of the five major radar bands at 38 GHz for the Ka band, 15 GHz for the Ku band, 10 GHz for X band, 6 GHz for C band and 3 GHz for S band radar [Sti

83]. Polarity can take one of four cases, Horizontal-Horizontal (HH), Vertical-Vertical (VV), Horizontal-Vertical (HV), or Vertical-Horizontal(VH). Each of these pairs refers to the orientation of the transmitting antenna and the orientation of the receiving antenna. The term crosspole describes those cases where the transmitting antenna and receiving antenna are in different orientations (HV and VH). Traditionally, the radar cross section for an object is expressed either as a value in decibel meters squared (dBsm) or displayed in a two-dimensional polar plot. When examining RCS values, the lower the value, the lesser chance of detection for that orientation of the target and frequency of the radar. The polar plot shows the relative strengths of the RCS return for a specific orientation and configuration of a platform. Figure 1 shows a typical two-dimensional polar plot. A single polar plot or value is insufficient to represent the RCS of a target, since an RCS value or polar plot is only valid for the frequency for which it was generated and its polarization. The typical non-engineer has little use for either of

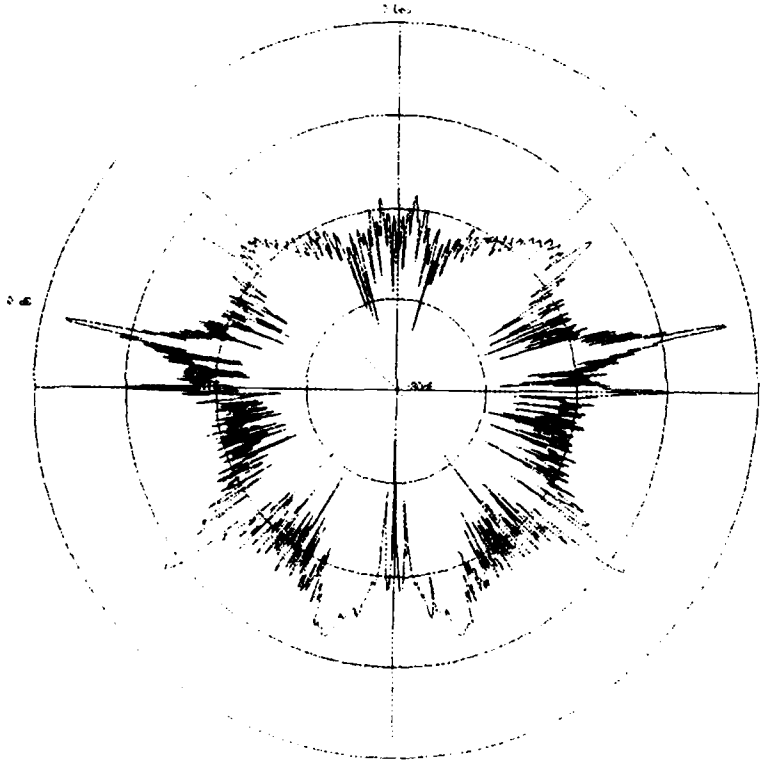


Figure 1. Typical Polar Plot.

these formats, and generally is unable to incorporate the RCS information during mission planning.

RCS is the ratio of the power of the radio waves that a target reflects (scatters) back in the direction of the radar to the power density of the radar's transmitted power. RCS accounts for cross section of the target, as well as reflectivity, and directivity, and may be used to predict the signal energy returned back to a radar. Reflectivity is the term for the fraction of the intercepted power reradiated (scattered) by the target [Sti 83]. Directivity is ratio of the power scattered back in the direction of the radar to the power that would have been scattered back if the reflection had been uniform in all directions (isotropic) [Sti 83]. RCS does not account for environmental factors such as ionization, humidity, or static. Each of these factors may alter the amount of energy reflected back to the radar receiver, affected the perceived target. Stimson [Sti 83] defines Radar Cross Section (σ) in equation 1.

$$\sigma = 4\pi \lim_{R \to \infty} R^2 \frac{\left|E^s\right|^2}{\left|E^i\right|^2} \tag{1}$$

where:

E^s = the energy scattered back to the radar

E^i = the power density function of the energy incident on the target

RCS directly affects the amount of energy returned to the radar receiver. The more energy received by the antenna, the greater chance of detecting and identifying a target. In equation 2, Balanis [Bal 1989] defines the energy returned to the radar in terms of the average transmitted power, the time the energy is actually on the target, the range to the target, the effective area of the antenna, and the gain of the antenna.

$$\text{Energy} = \left(\frac{1}{4\pi}\right)^2 \frac{P_{avg} G \sigma A_e T_{ot}}{R^4} \qquad (2)$$

where:

P_{avg} = the average transmitted power

T_{ot} = the time on target

R = the range to the target

A_e = the effective area of the antenna

G = antenna gain

Sweetman states that RCS is determined by first measuring, or calculating the amount of radar energy reflected toward an observer from the target. The designer then calculates the size of a sphere that would reflect the same amount of energy. The area of the calculated sphere is the RCS [Swe 86]. The key to LO technology is the minimization of the energy reflected from the target back to the radar receiver. Since the RCS (σ) of the target is in the numerator of the equation, a value of zero for RCS would reduce the returned energy to zero. Although this would be the ideal situation, design of components such as engines, wings, and fuselage for minimum RCS must be compromised with design for performance. In order to view the RCS of an entire airborne platform with 0.5 degree resolution, 360 separate polar plots would be required for each frequency of interest. It is unreasonable to assume anyone could deal with this volume of information for multiple frequencies or multiple configurations.

Viewing large and complex data sets such as RCS data is the basis of scientific visualization. Scientific visualization incorporates not only the application of scientific and engineering techniques to graphics, but also includes database theory, animation, simulation techniques, and signal processing [Fol 90]. Some or all of these fields may be necessary to display scientific and engineering data. Some of the ongoing research

areas in scientific visualization include medical data imaging, flow field visualization, electron density visualization, chemistry, astrophysics, and the geosciences.

Volume visualization describes the series of techniques for displaying three dimensional data in two dimensions. Volume visualization techniques have traditionally been divided into volume rendering techniques and surface rendering techniques. Surface rendering is the group of techniques used to display the three dimensional data as a shell, or surface of data. Volume visualization describes the group of techniques that display the volumetric data as multiple shells or surfaces. For further readings on scientific visualization techniques, see [Udu 91], [Kau 91], and [Sty 91].

1.2 Problem.

The purpose of this study was to determine if state of the art rendering techniques could be used to display Radar Cross Section data. This study was not intended to find the best method of rendering RCS data, nor was it intended to finesse any one method to increase efficiency or speed up rendering time. I examine three techniques for rendering three dimensional data, and determine their applicability to rendering single frequency (monolithic) RCS data.

1.3 Assumptions.

Conceptually, there are three major techniques for rendering three dimensional (3D) data: particle systems, volume rendering, and surface rendering. Each of these techniques has found its way to the low-end of computer graphics hardware. Since all three techniques are currently in use on typical microcomputer systems, I examined each method for applicability for rendering RCS data. Researchers at Ohio State University generated the data used in this study with its Radar Cross Section - Basic Scattering Code (RCS-BSC) [Mar 90] and a model of a fighter aircraft. Although the data set is unclassified, it is an accurate representation of RCS data currently in use. It

is important to note that this study was not limited to real-time manipulation and rendering of the data due to the large size of the data sets.

1.4 Scope and Limitations.

This focus of this study is the applicability of common 3D rendering techniques to radar cross section data. Because of the size of the data set, and the different data format requirements of each of the rendering systems, format converters and filters were constructed to feed existing radar cross section data into each of three different rendering systems, and reduce the number of raw data points. The results are presented to the reader later in this document for evaluation. The author makes no judgment as to the best technique for a user's particular application.

1.5 Approach.

The first step in completing this research was the literature review. Because this is a cross-discipline research project, the literature review included both methods for the generation of radar cross section data and techniques for the rendering of three dimensional, volumetric data. The review was performed to determine the current state in both electromagnetic studies and computer graphics.

The second step was the construction of a general purpose particle system renderer. Since application of radar cross section data to a particle system is one of the major research areas of my thesis, a flexible particle system renderer was needed. The design for the particle system renderer was based on the description given by Reeves [Ree 83]. The code was written using AT&T C++ version 2.1, and was designed with portability and flexibility as primary considerations. Because the AT&T C++ translator is the most restrictive of the C++ compilers or translators available, code that compiles with it should compile with only modest changes on other systems or with

other C++ compilers. The third step was the construction of a general purpose filter for the radar cross section data. A typical data file contains over one million data points, describing only one half of the object being tested, requiring the filter to be both flexible and accurate. After construction of the initial filter, the filtered data was formatted for each of the rendering systems. The formatting software was written in C++ and hosted on the Silicon Graphics 4D440/VGXT workstations in the AFIT Graphics Lab.

After formatting the data, it was rendered by each of the three rendering systems. The results for the different systems are compared and evaluated later in this document.

1.6 Thesis Overview.

The remainder of this document is organized as follows: Chapter 2 is the design and implementation of the three different rendering techniques, Chapter 3 discusses the results of the study, and Chapter 4 contains my conclusions and recommendations.

II. SYSTEM DESIGN

2.1 Introduction

Evaluation of the different techniques for rendering RCS data was an iterative process of design, development, implementation and testing. The particle system renderer was the first technique examined, followed by the volume renderers. Concurrent with this research, Wojszynski [Woj 92] examined surface renderers for use in scientific visualization of RCS data.

This chapter discusses the three techniques evaluated for rendering RCS data. For particle systems and volume rendering, the background of the technique is discussed, development issues are described, and implementation details are noted. The third section summarizes the work of Wojszynski, and the fourth section examines the data preprocessing.

2.2 Particle Systems.

2.2.1 Background.

Particle system renderers are typically used to model environments whose appearance change over time. The characteristics tracked in each particle include color, velocity, particle lifetime, and tail length. One or more equations describe the velocity of the particle, and generally take into account gravity, starting velocity, starting position and direction of travel to generate realistic effects. The lifetime property of the particle describes the length of time a particle is visible, and additional information such as the spawning time, spawn source and system time track the particle's ability to

spawn children. Since a particle is normally represented by a single point, a tail length helps simulate the blurring of a moving object by adding additional visual queues [Kor 83]. The types of objects modeled by particle systems include both dynamic and static systems. Probably the most viewed example of a particle system was shown in the **Genesis Demo Sequence** [Smi 82] from the movie ***Star Trek II: The Wrath of Khan*** [Par 82]. William Reeves describes the generation of the wall of fire used in the sequence in his 1983 SIGGRAPH article [Ree 83]. Other applications of particle systems discussed by Reeves in his article are generation of fireworks and explosions in cinematography [Ree 83]. Other environments modeled by particle systems include fog, mist, and other diffuse media [Fol 90]. A unique application of particle systems to static models is Reeves and Blau's rendering of trees and grass in a forest [Ree 85]. For the trees, the random characteristics of the particle system determined placement, length and direction of growth for the branches and twigs. The random characteristics of the particle system also generate the placement, length and curve of the blades of grass. Nature is not normally symmetric, and the randomness of the particle generator removed the synthetic look common to traditionally generated images. The drawback of applying the particle system to new fields is each application generally requires the development of a new model.

Because particle systems do not attempt to map data points onto a surface, but display each input data point as a member of the overall system, I felt this technique might have potential in the display of robust, vectorized data sets. No additional overhead is required for surface fitting computations, and no bias is introduced to the data set from interpolation of the data set.

2.2.2 System Design.

The main purpose of the particle system developed for this research is a portable platform for the generation of a frame by frame depiction of an environment for use in

video recording and playback. The animation provided by the video playback provides the user with additional three-dimensional queues for analysis of the data. The system was ***not*** designed for interactive display and manipulation of the frames. The system uses the Utah Raster Toolkit [Tho 86] for generation and storage of the frames, and was designed to run on a graphics workstation with at least 16 megabyte of memory. The Utah Raster Toolkit stores color images using the Utah run length encoded (RLE) format for the data. The RLE format was designed for efficiency and device independence [Tho 86]. The modularity of object oriented design in C++ combined with the portability of the Utah Raster Toolkit insures a portable, flexible rendering system. The basic algorithm for implementing the particle system is contained in figure 2.

```
For each particle:
      Check for spawning
      If spawning
        Generate new system
      Check for death of particle
      If particle is dead
        Mark and ignore
    Move Particle
 Render Particles
 Increment time step
```

Figure 2. Algorithm for Particle Generation

To keep the particle rendering system portable, I chose not to implement an interactive user interface. Instead, the system was designed to read and parse standardized data and configuration files at run time, and Appendix B contains the format for these.

All particles are generated from a central or base object known as a centroid. The user selects the shape of the centroid prior to rendering as an option in the data file. For the RCS data, a spherical centroid was chosen since it best represents the center of a spherical coordinate system. The user also specifies the placement (origin) of the

centroid in world coordinates and the coordinate system type, either spherical or cartesian. Particles and their respective centroid are stored in a dynamic linked list. At each time step, the list is first checked for new particles being spawned. The new particles are generated, and added to the linked list structure. The list is then checked for particles that have died. Dead particles are marked as such, and no longer displayed. The control file also allows the user to set, then change the view of the overall system. This feature allows the view to be changed during the rendering of the particle system. The user can select whether the particle system continues to grow, or freezes during the view change.

2.2.3 System Implementation.

The view volume requires specification of a viewer's position (PRP), where the camera is looking (CW), a view plane normal (VPN), and a view up vector (VUP). See figure 2 for the coordinate system. This view is based on the three-dimensional view coordinate system described by Foley [Fol 90]. The user must also specify viewport dimensions for rendering, and world limits (including front and back) for clipping.

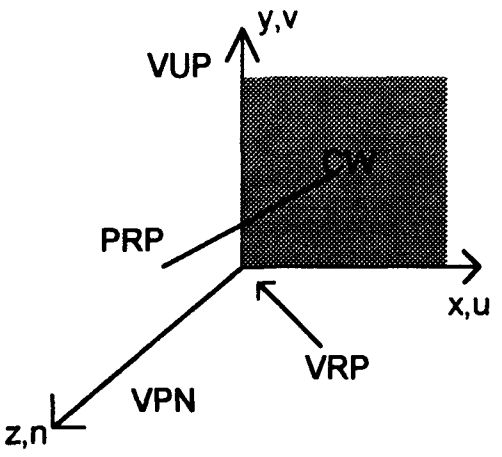


Figure 3. View Coordinate System

The particle system renderer generates images using a Z-Buffer and Wu antialiased lines [Wu 87]. The Wu antialiased lines are drawn by incrementing one pixel at a time along the major (longer) axis. For each increment in the major axis, two pixels are drawn bracketing the axis. The intensity of the pixels are determined by their distance from the actual line. The Wu method for generating lines was chosen for its simplicity, accuracy, and efficiency. The particle system renderer generates either static or random particle systems, as determined by the user. Because the RCS data was mapped to specific locations, the random particle placement was not used . After viewing several images generated from the RCS data set, I determined the resulting images lacked sufficient cues for the volumetric nature of the data. This issue was addressed by animating the development of the particle system over time. A feature added to the particle rendering system was the ability to change the user's view, including rotation around any of the three axis while the particle system is growing. The user also has the option of stopping the particle system growth while the view changes, and can specify the number of frames to spread the view change over. During the taping of the frames to generate the final animated images, this feature provided a smooth panning of the view, rather than an abrupt change in the image.

2.3 Volume Renderer

2.3.1 Background.

Stytz defines volume rendering as displaying multiple surfaces or an entire volume and presenting the user with the visualization of the entire space [Sty 91]. Three of the more common volume rendering techniques are ray casting, V-buffer and splatting. Ray casting is a technique where a vector (ray) traces through the volume of data. A ray is fired from each pixel through the volume of data. Ray casting algorithms conduct

an image-order traversal of the image plane. Similar to ray tracing, the ray accumulates color and opacity until the end of the data volume is reached. It differs from ray tracing in that the ray is not deflected or refracted as it passes through the volume. Descriptions of ray casting algorithms can be found in [Ups 88] , [Lev 88] and [Lev 90]. Upson's [Ups 88] V-buffer technique is a front to back, object-order rendering technique where the ray accumulates color and opacity as it passes through the volume. The ray stops when it exits the volume, or an opacity of one is reached. Splatting [Wes 90] is an algorithm which also performs a front to back, object-order traversal of the volume. Lookup tables are used to calculate and composite the contribution of each voxel to the view. The disadvantage to most volumetric data sets is that the data is not continuous. Care must be taken when mapping the data set to a regular grid to avoid aliasing the data. The key feature of volume rendering is that all of the data is used, not just the surface information.

2.3.2 System Design.

I examined the Volumetric Imagery Program for Engineering Research (VIPER) system developed by Bridges [Bri 88] as the Volume Renderer for displaying the RCS data. VIPER was designed to be a flexible and portable volume rendering system for use in Computational Fluid Dynamics (CFD). This system was written using C and the Utah Raster Tool kit. Another key feature of VIPER is its extensive user interface for environmental setup and image construction. In order to run VIPER using a non-regular data set, a front-end processor was needed. Fortunately, Lentz [Len 89] developed a VIPER pre-processor for use with irregularly spaced CFD data. Lentz's front end processor, MAP, was designed to compute regularly spaced values for mach and pressure numbers. The input data coordinates are read into separate lists in memory, then searched for the minimum and maximum values. The data set

coordinates are then normalized using the number of planes specified by the user. The system defaults to 100 planes for each axis. The grid coordinates are interpolated to generate points in the new planes generated by MAP. After the new points are generated, the data set is filtered for manageability. Using MAP developed by Lentz as a starting point, I was able to map the RCS data into a regular grid for use in VIPER. The data output by MAP was plotted to determine the validity of the interpolation. In order to use the RCS data in the volume renderer, each polarity of the data was assigned a distinct scalar value. The initial results were unacceptable due to a poor choice of values. Closely spaced values, and values close to zero were found to significantly bias the final results. A spacing of 50 units was found to be sufficient to produce an accurate representation of the initial data set.

2.3.3 System Implementation.

MAP uses trilinear interpolation to map non-regular data onto a rectangular grid coordinate. The first modification I made to MAP was the alteration of the input and processing sections to accept the X, Y, Z, Value format of the RCS data set. Additionally, I modified MAP to allow processing of data sets other than mach values and pressure numbers. I then modified the filter for the raw RCS data set to convert the spherical coordinate system data to cartesian coordinate system data. The filter also translated all of the coordinates to the positive axis and passed the data through a high pass and low pass filter. The filtered data was stored as two files in binary format, one containing the coordinate values and the second containing the data values. The coordinates were generated using phi and theta as the basis for X and Y and the strength of the RCS return was used to generate the radius value for Z. Figure 4 shows the spherical coordinate system used in this design. The data values were generated by assigning a different scalar value to each of the polarities of the RCS

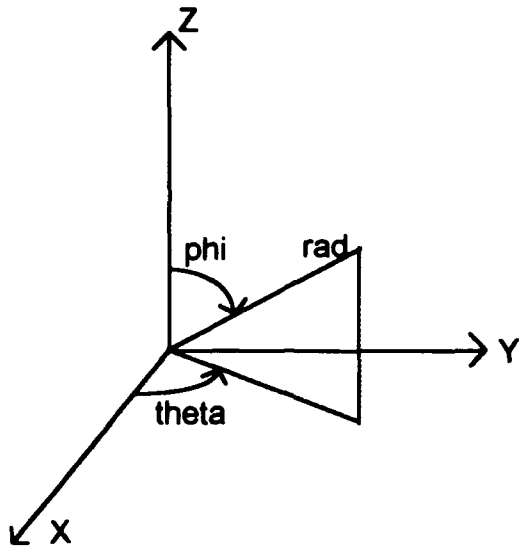


Figure 4. Spherical Coordinate System

data. Vertical polarity (VV) and Horizontal polarities (HH) were given unique values, while both crosspole polarities (HV and VH) were given the same scalar weight, as they are a minor contribution to the overall data set. As with the particle system, this gives the cross-pole data a reasonable contribution to the final image.

2.4 Surface Rendering

2.4.1 Background

Surface rendering is concerned with generating a smooth image of a single surface from a discrete data set [Cli 88]. Although they are both methods of volume visualization, surface rendering differs from volume rendering in that it is only concerned with rendering a single shell of data. The shell is rendered as a continuous surface, with different techniques employed to create a closed data set. Of all the surface rendering techniques available, bicubic fit, polygonal-mesh, and convex hull techniques appear to be most useful for visualizing RCS data. The bicubic fit is applied to polynomials that are cubic in both parameters. Bicubic surfaces are joined through

iterative evaluation of the surfaces to be joined. Recursive subdivision of the surfaces is necessary to eliminate cracking [Lan 79]. In the polygonal-mesh technique, a polygon mesh or net is applied to the discrete data points to approximate a smooth surface. A polygon mesh is a collection of edges, vertices, and polygons connected such that each edge is shared by at most two polygons [Fol 90]. Then the mesh is displayed as a combination of filled and outlined polygons, with shading applied to smooth the edges. Convex hull techniques describe the family of techniques that treat the surface to be rendered as a collection of convex points. Again, adaptive subdivision and interpolation are used to generate, smooth, realistic surfaces. For more information on surface rendering techniques using the Satellite Modeling System [Pon 92] , see Wojszynski [Woj 92].

2.5 Data Preprocessing.

This section describes the data filters built to support rendering of the RCS data for the volume and particle rendering systems. The RCS data filter was used to convert the raw data files provided by Ohio State University into a format usable by the different rendering systems. The file header contains the spacing of the measurements taken, the frequency at which they were taken, and format specifications. On the basis of this information, the filter extracts the data into spherical coordinates. The primary data of concern is the phi (ϕ) and theta (θ) of the measurement, and the strength of the return for the four polarities, and the phase of the returned signal is available to the filter. The raw data set is stored in two files, one containing the top of the model and the second containing the bottom of the model. Both files must be read by the filter to process the entire data set. The data contains measurements for only the first 180 degrees of the model with the front of the model at zero degrees. Symmetry is assumed along the longitudinal axis. Based on this symmetry, the filter generates data

for the full 360 degrees and processes both the top and bottom data files, creating a data set for the complete model in a single data file. Each rendering technique required a different version of the filter, as the input files for each renderer were incompatible with the others. Keywords were added to the output for the particle system, and the volume renderer required conversion of the spherical coordinates to the cartesian coordinate system, and the data in binary format. All versions of the digital filter contained the ability to shift the data into the positive coordinate system, and pass the data through a high pass filter and a low pass filter. The thresholds of the filters are selected by the user when the raw data file is processed.

2.6 Summary

This chapter presented in detail the software development for the particle system, the volume renderer with its associated front-end processor, and the data preprocessor. Concentration was on the particle system renderer since it was original work developed for this application. With an understanding of the design process in place, the next chapter discusses the results from the particle system renderer and the volume renderer.

III. RESULTS

3.1 Introduction

The goal of this research was to determine if graphics rendering techniques could be used to display RCS data. It was not the purpose of this research to determine which filter values best represented the overall data set, nor was the purpose of this research to identify areas of potential design flaws in the data sets examined. Research in these areas is left to those specializing in electromagnetics.

The primary purpose of this study was generating accurate, 3D images of the data sets provided. An image was considered acceptable if it accurately displayed the relative magnitude and polarity of the data with respect to the origin of the object without aliasing. This chapter presents the results of each of the three rendering systems, as well as an analysis of the applicability of each method.

3.2 Particle System Renderer.

The particle system renderer allowed me to display the RCS data with the least amount of filtering and data manipulation. To establish the coordinates of the particle, the phi and theta of the value in the RCS data were used as the phi and theta of the particle. The value for rad was set to zero for all particles. This placed the starting point of all the particles in a sphere around the centroid. All particles started at the same time, and the strength of the RCS return was used for the velocity of the particle. The color of the particle was determined by the polarity of the data. The first data set rendered by the system was the VV polarity data. During the prototyping phase of the development, the data was processed through the band pass filters with the low pass

filter set with 20 decibels per square meter (dBsm) as the lower threshold and the high pass filter set at 65 dBsm as the upper threshold. With this data set, these filter values display both the low return areas and the high strength return areas with a reduced number of data points. These values may change with each new data set, and appropriate values should be determined by inspection of the raw data set. Figures 5, 6 and 7 show the results of the original data set along each of the three axis. The figures show only the VV polarity data, with the particle color set to red. The data was passed through the preprocessor with both the high pass and low pass filter limits set to 0. Setting both filters to the same value passes the entire data set to the output files. Figures 8, 9, and 10 show the results of the HH polarity data along each of the three axis. The data was passed through the preprocessor with both the high pass and low pass filter limits set to 0, and the particle color was set to blue. Figures 11, 12, and 13 show the results of the HV and VH polarity data along each of the three axis. The data was passed through the preprocessor with both the high pass and low pass filter limits set to 0. The particle color for both the HV and VH data sets were set to green. After the successful rendering of single polarity data, the raw data was run through the preprocessor a second time to generate a data set containing all four polarities, and figures 14, 15, 16, and 17 show the results of this attempt. Figure 16 shows the view from the nose of the target, while figure 17 shows the view from the tail of the target. For this data file, there was a noticeable contribution from the crosspole data to the nose-on view of the aircraft. For each of the multi-polarity images, the low pass filter was set to 0, and the high pass filter was set to 70. These values were chosen to reduce the final size of the data set for rendering without compromising the resulting image. After rendering the data as a series of still images, a video tape was made of the particle system growing from the centroid to a maximum value, rotating around each of the three axis, one at a time, to give the viewer a better sense of the volumetric

nature of the data. The final tape lasted about three minutes, took 1900 separate images, and required six hours to record. Rendering the final data set required 10 minutes to load the data files, and new frames were produced approximately every 95 seconds. One problem noted in the recording was the significant loss of resolution when transferring the image to tape, and then displaying it on NTSC monitors. As the resolution decreases, the data tends to blur together, potentially masking areas of interest. A second problem noted was the lack of standard aspect ratios in the NTSC color monitors. The "look" of the RCS data set is significantly altered by minor changes in the aspect ratio of the monitors. When the aspect ratio changes, the resulting image may falsely indicate areas as having large returns, while hiding areas which do. Figure 18 shows an image on the Sony PVM-2030 NTSC graphics monitor, and figure 19 shows the same image on a square NTSC monitor. The final tape provided the user with a good, intuitive feel for the data and its areas of interest.

3.3 Volume Renderer.

The images generated with the volume renderer VIPER did not allow real-time manipulation by the user. Several images could be generated interactively by VIPER through command line entries, however, the resulting images were written into individual RLE files. This removed the overhead of reading the data set for each image if the user knew what parameters to select prior to executing VIPER. Figures 20, 21, and 22 show an image generated by VIPER, viewed along each of the three axis. The data for these images was filtered at 20 dBsm for the lower threshold and 100 dBsm for the upper threshold. One of the problems encountered by VIPER was the time required by the preprocessor MAP. An image containing 430,00 data points required over 27 days of CPU time on a Silicon Graphics 4D/440 VGXT workstation to process. The algorithms employed by MAP are order N^5, resulting in significant increases in

processor time for minor increases in data set size. Due to the nature of the trilinear interpolation designed into MAP, reduction of the order of complexity of the program was not a viable option during this thesis cycle. Additional problems were noted in the granularity of the rendered data set. Again, as the resolution of the data set was increased, the time required for MAP to process the data increased. VIPER requires approximately 20 minutes to generate the final images with 250 planes in the data grid. The highly irregular nature of the data also proved troublesome to the volume render as it tried to generate a smooth shell from the data. In some cases, data in adjacent cells varied as much as 180 dBsm.

3.4 Summary.

In this chapter I presented the results of the particle system renderer and the volume renderer. The images for both systems were derived from the same initial data set. The final set provided to the volume renderer was somewhat smaller due to time constraints. The particle system renderer was able to handle the larger data set, eliminating the requirement for filtering altogether. The volume renderer required careful selection of the filter limits through trial and error to produce an acceptable image that compromises resolution and time.

The results of this study indicate that although some work is still required to improve the performance of the 3D rendering systems, the provide a useful tool in the display of RCS data. The next chapter discusses the overall conclusions for this effort, and provides some direction for future work in this area.

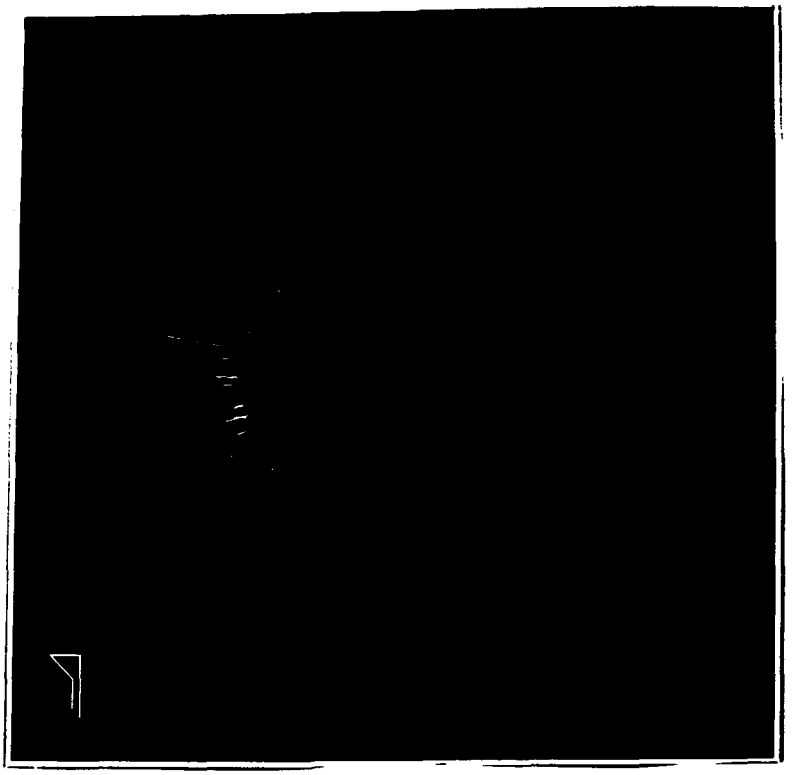


Figure 5. VV Polarity Unfiltered Data Viewed From the X axis

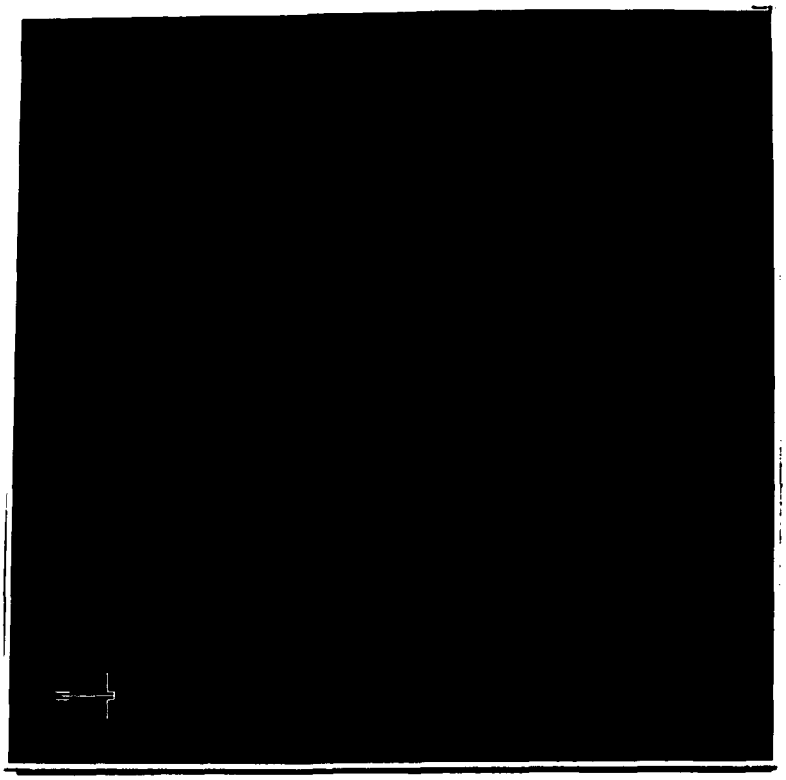


Figure 6. VV Polarity Unfiltered Data Viewed From the Y axis

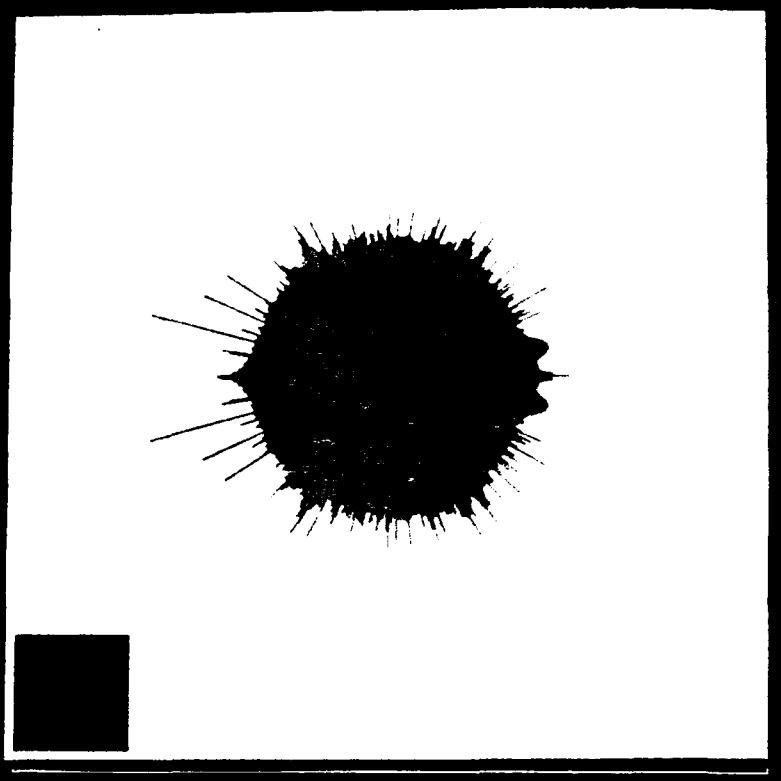


Figure 7. VV Polarity Unfiltered Data Viewed From the Z axis

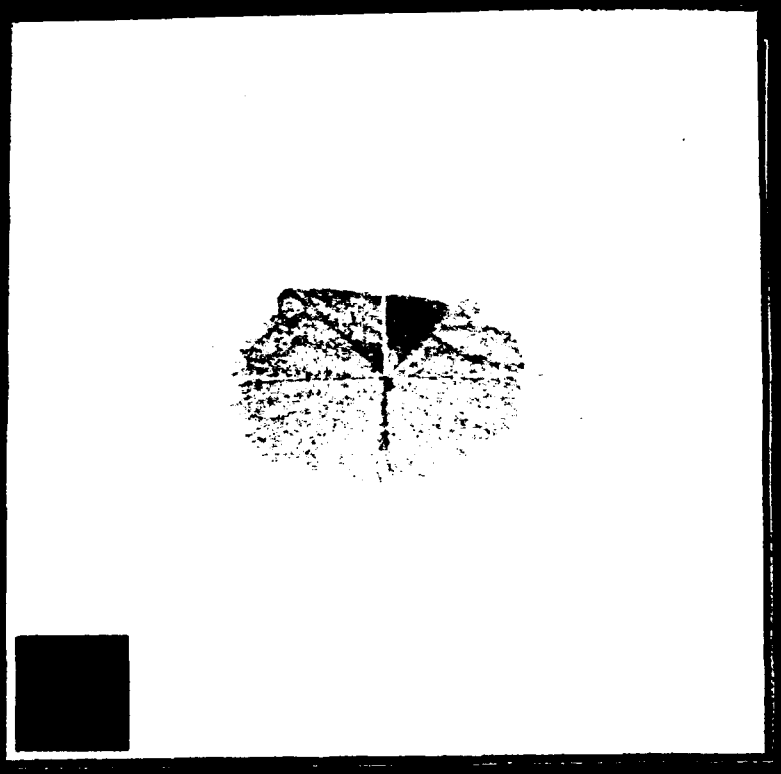


Figure 8. HH Polarity Unfiltered Data Viewed From the X axis

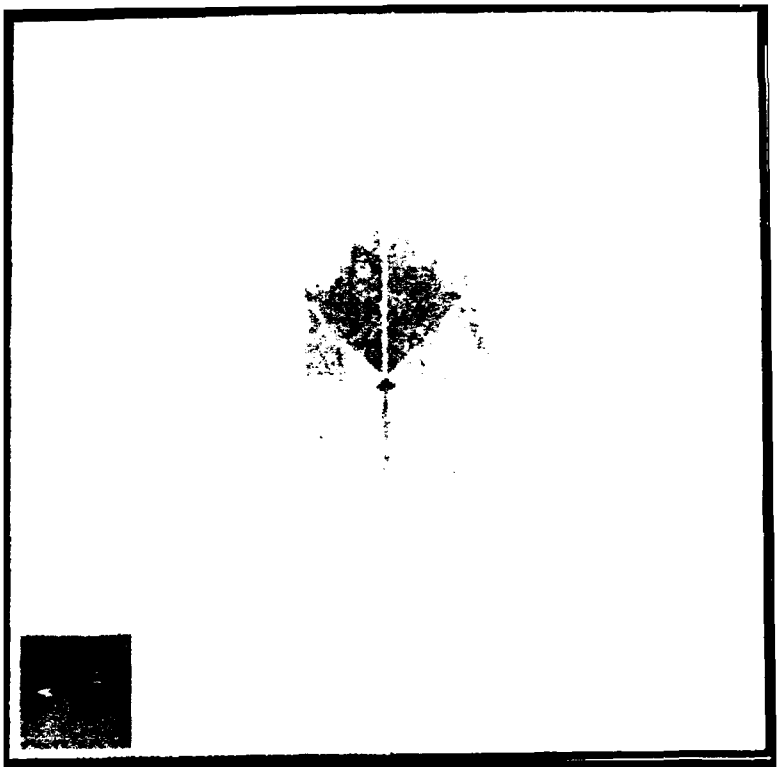
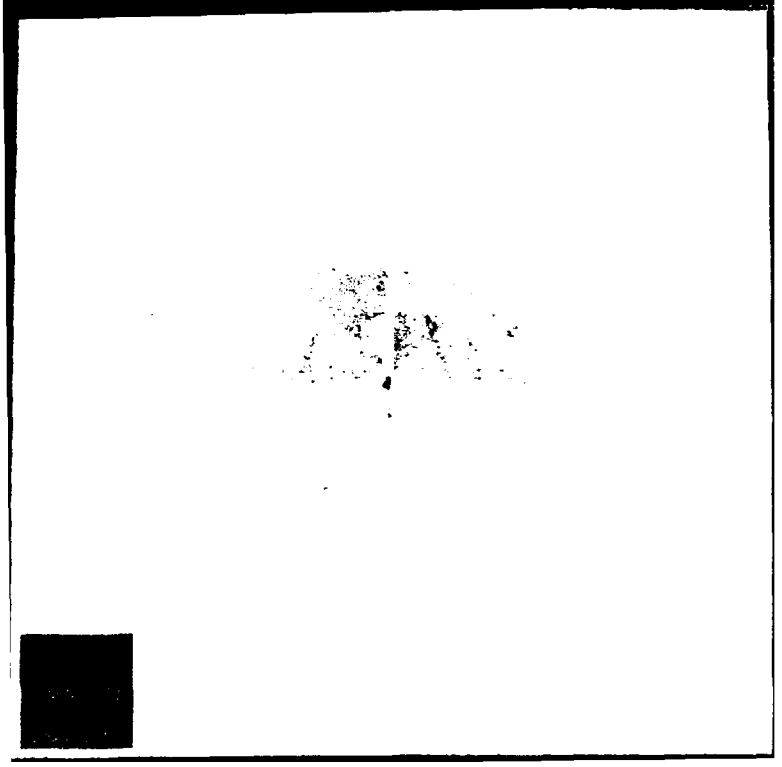


Figure 9. HH Polarity Unfiltered Data Viewed From the Y axis



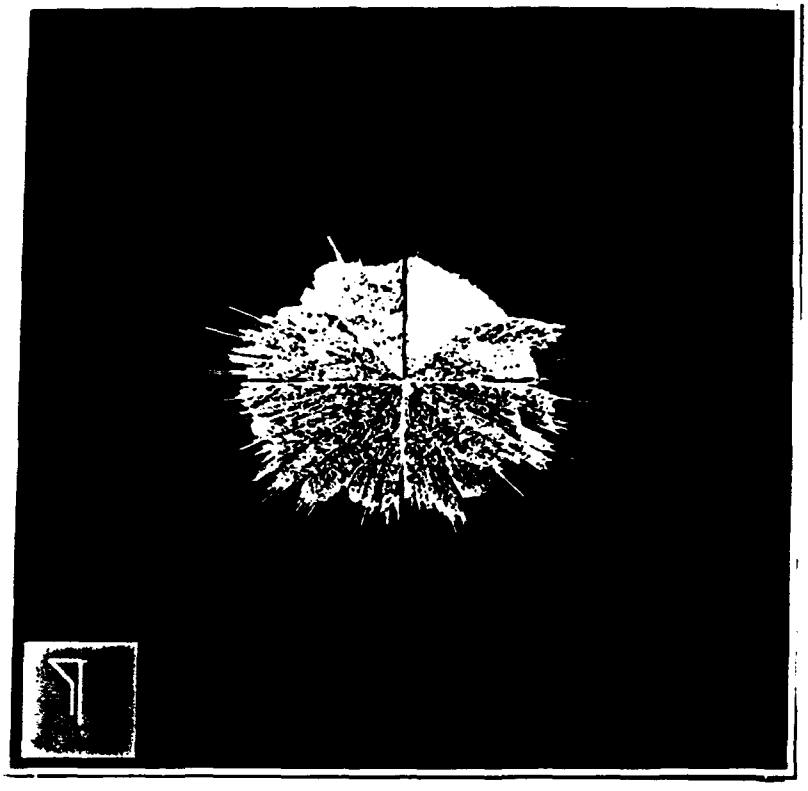


Figure 11. Crosspole Unfiltered Data Viewed From The X Axis

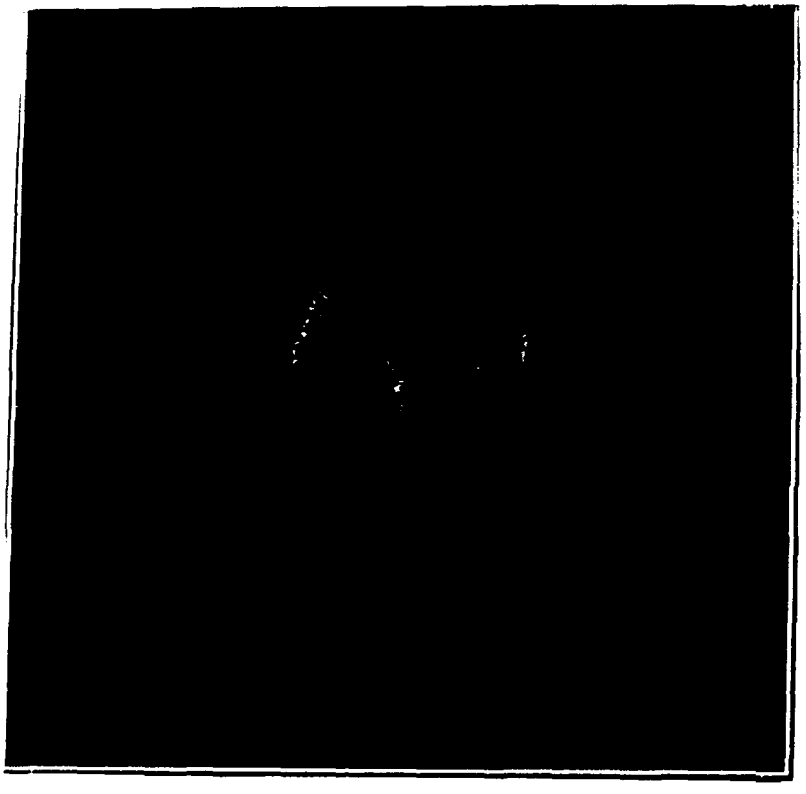


Figure 12. Crosspole Unfiltered Data Viewed From The Y Axis

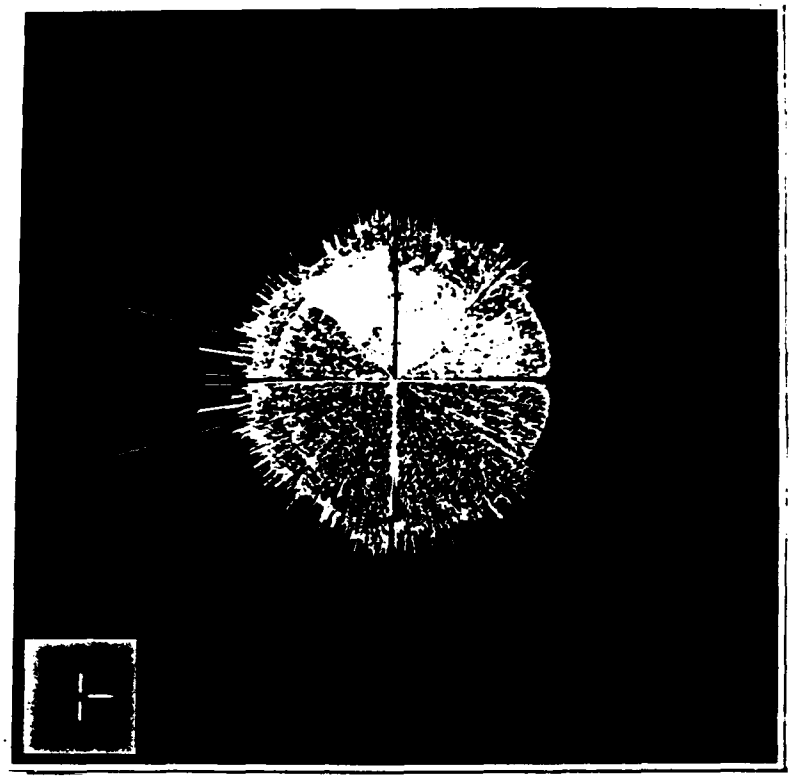


Figure 13. Crosspole Unfiltered Data Viewed From The Z Axis

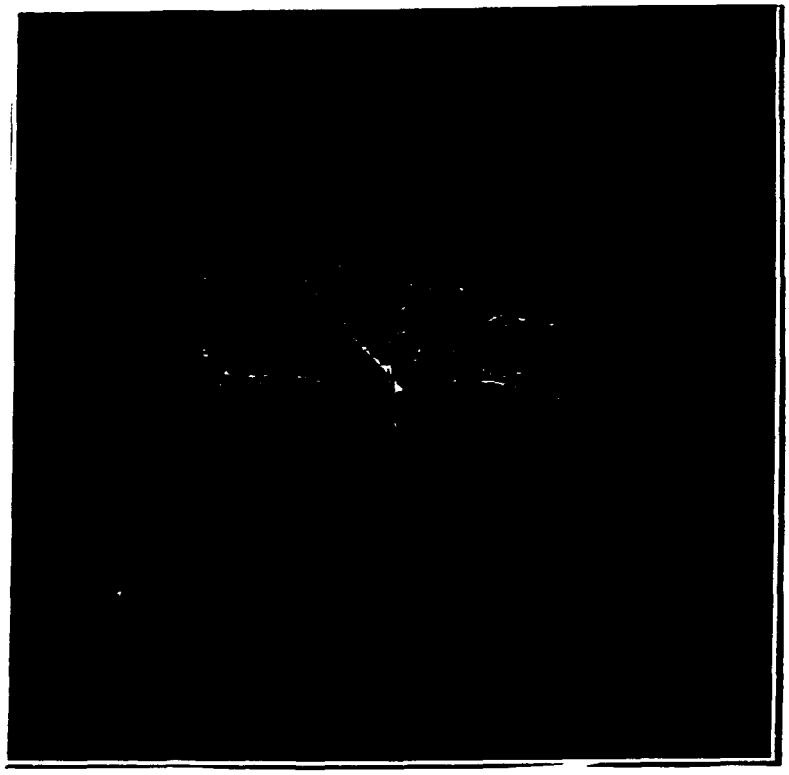


Figure 14. Multi-polarity Data Viewed From The X Axis

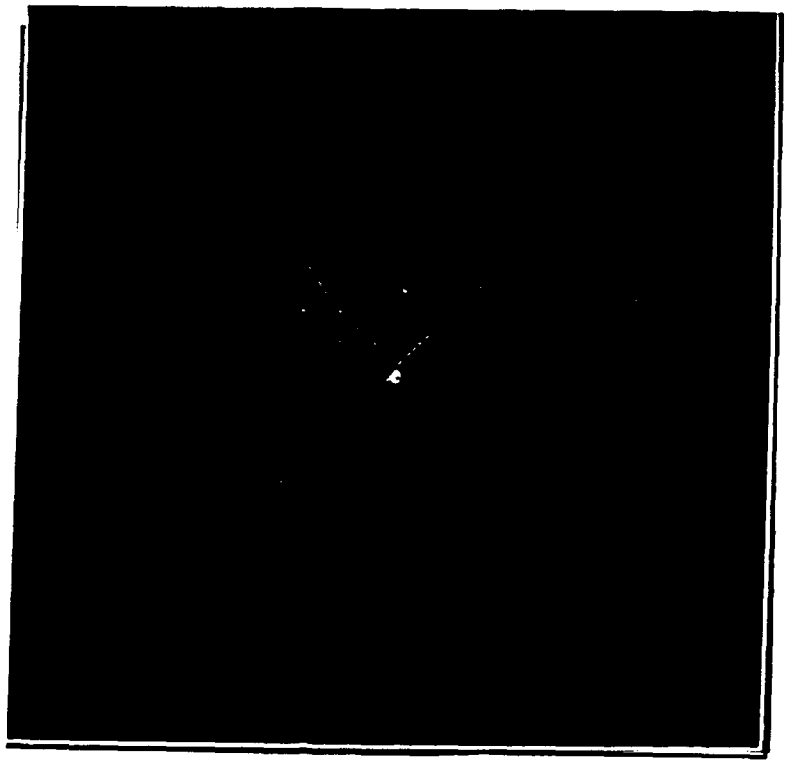


Figure 15. Multi-polarity Data Viewed From the Y axis

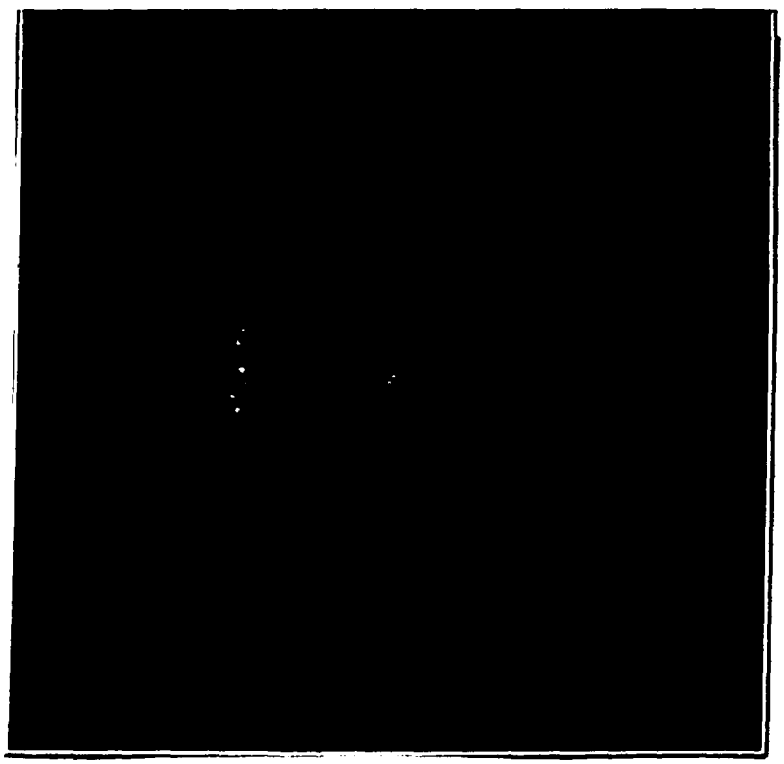


Figure 16. Multi-Polarity Data Viewed From the Nose

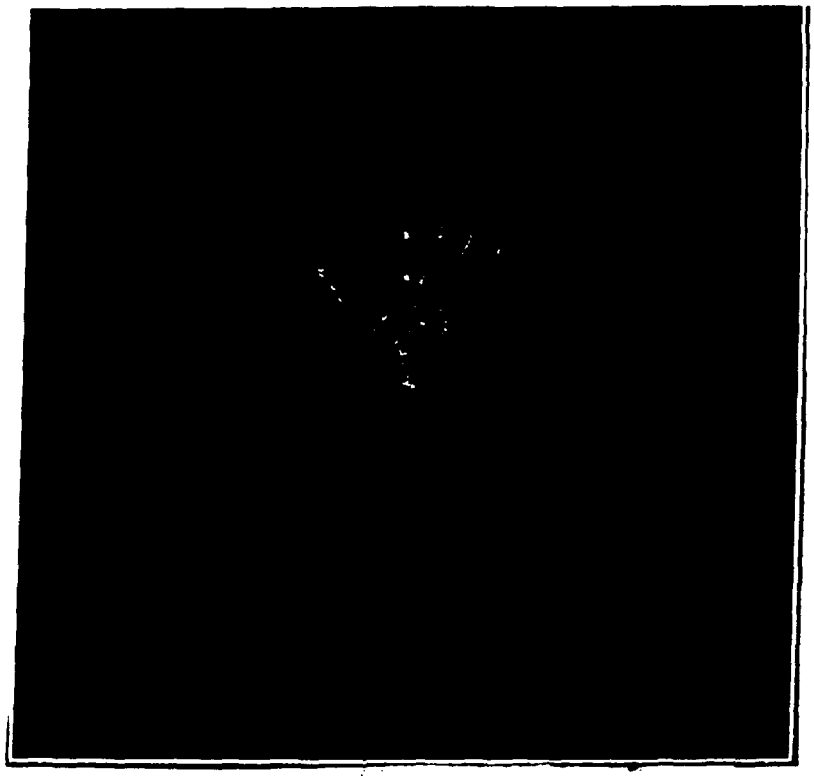


Figure 17. Multi-Polarity Data Viewed From the Tail

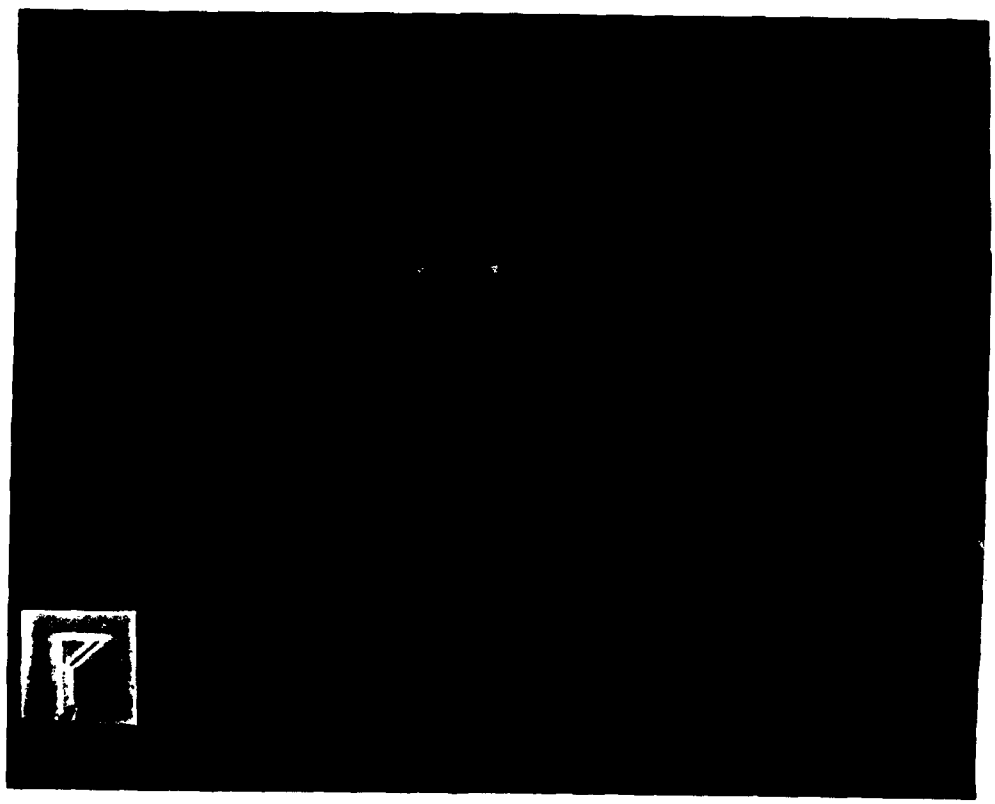


Figure 18. Multi-Polarity Data Viewed On Sony PVM-2030 Monitor

Figure 19. Multi-Polarity Data Viewed On Square Monitor

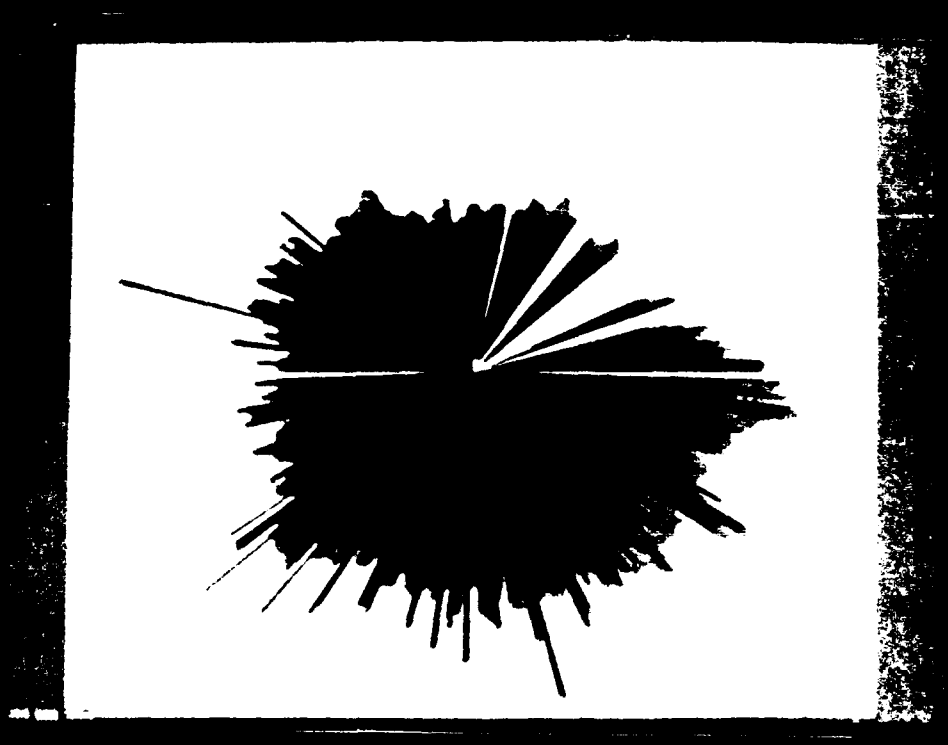


Figure 20 Multi-Polarity VIPER Image Viewed From the X axis

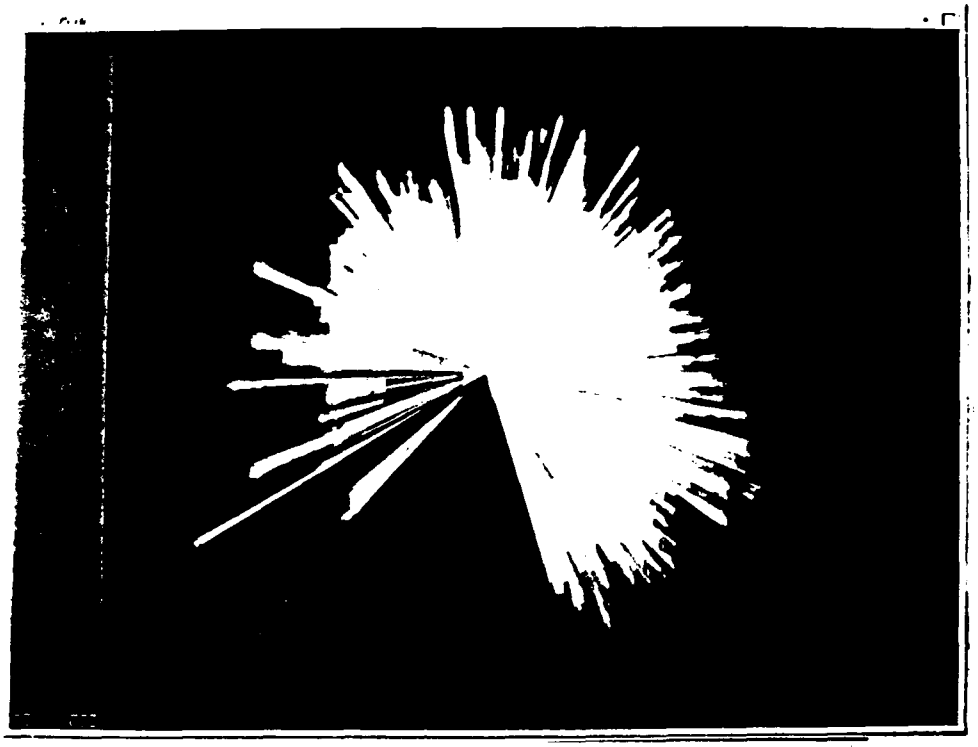


Figure 21. Multi-Polarity VIPER Image Viewed From the Y axis

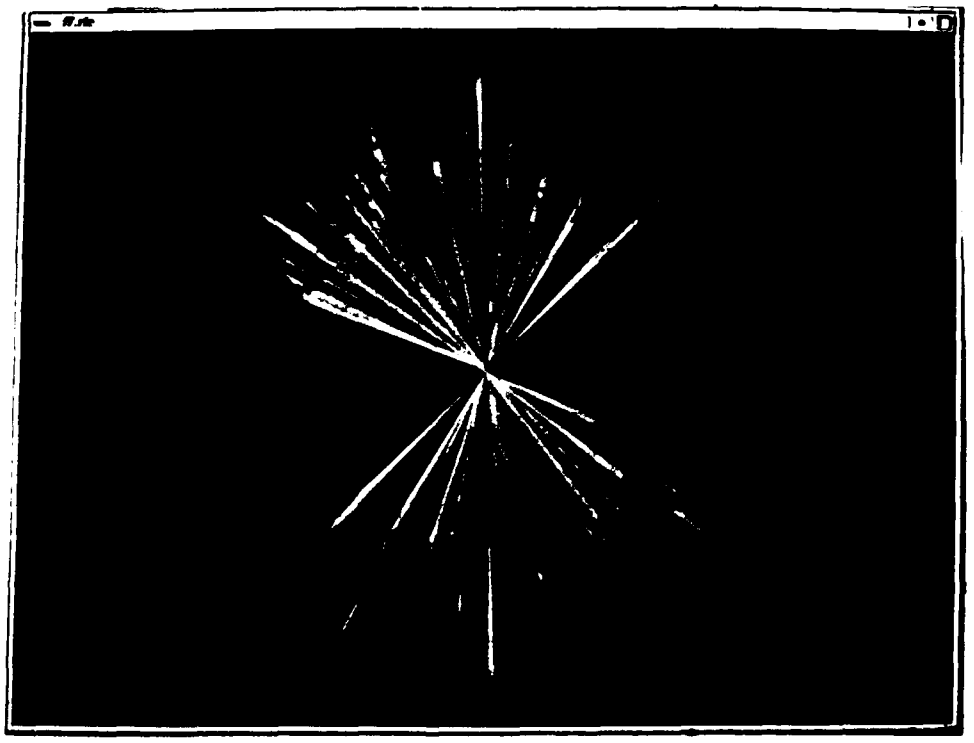


Figure 22. Multi-Polarity VIPER Image Viewed From the Z axis

IV. CONCLUSIONS.

The first three chapters of this thesis present the rationale and details involved in applying state-of-the-art graphics rendering techniques to RCS data. In this chapter I assess the results of each of the methods examined. I will also address areas of future work.

4.1 Particle Systems.

The particle system renderer is able to handle large data sets with a minimum of modification. Even with the large data sets, after the initial data files are read into memory, frames are generated at a rate of one frame every forty seconds. The particle system also displays the data as a series of discrete points without any interpolation of the data. This precludes any aliasing that might be generated by interpolating the values into a surface. Performance of the particle rendering system was inversely related to the size of the data file. As the size of the data file grew, the amount of time required to process the data grew in a linear fashion. The image quality of the rendered data set was exceptional on the workstation monitors, and good on the NTSC monitors. Although some data was obscured when viewing the static images, the animated sequences eventually displayed all of the original data through rotation around the different axis. The use of primary colors as the basic color set was well received by the users, even on the recorded images [RTF 92]. The users also felt the infinite tail length gave them a better feel for the vectorized property of the data, while displaying the data as a cloud of discrete particles was not considered intuitive to the nature of the data [RTF 92]. The addition of the slaved icon to the image significantly aided the viewer in conceptualizing the mapping of the data back to the original target shape.

Some of the disadvantages seen in the particle system renderer were the lack of real time manipulation, and the reliance on additional equipment for support of the

animation. Although the animated sequences were well received, the still images were considered informative, but not as useful. Another disadvantage of the particle system renderer was the large data files generated by the preprocessor. Because of the general nature of the particle rendering system, a large amount of overhead was generated in the form of tokens or place holders to set the characteristics for each of the particles. Additionally, rendering very large systems (those in excess of 10 million data points) was limited by the available swap space and memory of the workstation. Although this problem was resolved by moving to another platform, it has the potential to limit the user's applications.

4.2 Volume Rendering Systems.

The volume renderer provided the user a different view of the same data set. The preprocessing performed by MAP allowed the generation of a uniformly spaced data set displaying surfaces instead of discrete points. Some users were more comfortable viewing the data as a series of transparent shells rather than as discrete points, and this rendering method gave them a better insight into the data. Data in the foreground of the image was clear, but data located toward the middle and back of the volume was obscured or lost. The overall quality of the image was lower than that provided by the particle system, and automated production of multiple frames was not well supported. Use of primary colors for the surfaces provided good distinction between different surfaces to the user, but the transparency was sometimes confusing.

The major disadvantages to the use of volume rendering for the display of RCS data are the significant time required for the preprocessing of the data, the coarse granularity of the final image, and like the particle system, the lack of a means for the real-time manipulation of the data and the view. It was not uncommon for a data set containing as few as 1000 data points to require in excess of 800 CPU minutes to run.

Even as a low-end prototype system, this was not considered acceptable. Additionally, as we tried to increase the resolution of the final image, the preprocessing time also increased exponentially. With only 100 planes of resolution for the final image, the granularity of the image made it difficult to use and interpret. With the current volume renderers in place in the AFIT Graphics Lab, we require a significant amount of CPU time to generate low quality, low resolution images without the benefit of real-time manipulation.

4.3 Future Work.

Several improvements can be made to the systems to increase their ease of use, and resulting images. The volume renderers can best be improved by altering the front-end processor MAP. Alternative algorithms such as kriging should be examined as a possible alternatives to the trilinear interpolation. Additionally, alternate volume rendering techniques that do not require regularly spaced input data should be examined for use with multi-polar vectorized data. The user should also have the ability to examine all four polarities of the data while manipulating the final image in real-time. This could be accomplished by distributing the rendering process over multiple processors. Finally, the volume renderer chosen should have the capability of rendering an icon or core to represent the orientation of the data being rendered. Future work for the particle system renderer includes optimization of the renderer for multi-polar vectorized data as a method to increase performance. Reduction of the number of ASCII tokens in the data file would be a significant speedup. Distributing the frame calculations over several processors, using a separate processor for each centroid rendered should also provide a linear improvement in performance. Phase information is currently ignored by both renderers, and should be incorporated into the final image. Methods for incorporating the information include alteration of color intensity or the tint (base values) of the colors to show differences in phase. Finally,

additional filtering techniques, such as the sector averaging technique [Swe 92] should be examined for reducing the size of the data set without compromising the integrity of the original data. Both particle system renderers and Volume renderers have shown promise for the visualization of multi-polar vectorized data, and future research into these techniques is warranted.

Appendix A. Data Preprocessor User's Guide.

A.1 Introduction

The filter designed to read the data produced by the RCS-BSC code is a general purpose high pass filter, low pass filter pair. There are currently two versions of the filter available to the user, PSRFilter and MAPFilter. PSRFilter was designed for use with the particle system renderer PSR, while MAPFilter was designed for use with the Volumetric Imaging Program for Engineering Research (VIPER) and its preprocessor, MAP.

This User's guide outlines the installation of the filters on a typical computer system, and provides pointers for modifying the Makefile for installation on systems other than the Silicon Graphics Workstations running the Unix operating system. The filters were written to be system independent C++ routines, with changes to the compilation flags the only requirement for system conversion.

A.2 Installing The Filter

The filter requires compilation of a single file to generate an executable image. The file Makefile is located in ~dtisdale/thesis/psr/cnv to generate the executable file. The Unix command line is simply make PSRFilter or make MAPFilter. This assumes you are building the filter on a Unix machine. If your system does not use the AT&T C++ compiler, then you must modify two lines in Makefile. The first modification is to the line containing the symbol definition

CC = CC.

You must modify this line to read

CC = <your compiler name>

The second modification is to the flags line. The line is

CFLAGS = g

The CFLAGS must be set to any compiler flags your system requires. Additionally, the file requires linking to the C++ math library. If this library is not in the default path for the compiler, it must be added. The -L make option is required to add a library search path.

A.3 Using the Filter.

Using the filter is a straight forward procedure requiring the proper RCS-BSC data files and a few user supplied parameter values. The two data files required by the filter are the RCS-BSC data files for the top and the bottom of the object scanned.

A.3.1 Command Line Entries

The command line entry for PSRFilter is

PSRFilter -option

The **-option** allows the user to specify the data files, the output file, and the filter thresholds. The -t option allows the user to specify the path and name of the data file containing the description of the top of the object scanned. The -b option allows the user to specify the path and name of the data file containing the bottom of the object scanned. The -o option allows the user to specify the name of the output file. The -f option allows the user to specify the upper threshold of the notch filter, and the -n option allows the user to specify the lower threshold of the notch filter. The command line is parsed by the filter, and error messages will be generated if problems occur.

A.3.2 Differences for MAPFilter.

MAPFilter differs from PSRFilter in that more output files are generated. MAPFilter produces x.dat, y.dat, z.dat, i.dat, and d.dat. The separate files provide the user with the flexibility in setting up the input files for MAP. The version of MAP modified for this thesis is expecting two binary data files. The first file contains the index information,

and the second file contains the data values for each index location. To create the files required by MAP, use the Unix cat command as follows:

```
cat i.dat x.dat y.dat z.dat > index.dat
```

The order of the first four files is critical for the proper operation of MAP. The name of the output file is not critical, and is specified by the user. The two files index.dat (or the data file specified by the user with cat) and d.dat are required for input to MAP.

A.4 Conclusion.

In developing the filter, I have tried to produce a flexible tool which can be quickly adapted to changing requirements. The user specified options allows fast modification of the filtered data without the need for editing and recompilation. Modifications to the output format should be documented in the code, and new versions of the software created.

APPENDIX B Using PSR.

B.1 Introduction.

The Particle System Renderer PSR is a general purpose system for the generation of particle systems. Although features were added for support of RCS data rendering, it remains a flexible utility.

B.2 Installing.

PSR requires the Utah Raster Toolkit version 3.0 to install and run properly. Contact the system manager for the location of these libraries if they are not available. The routines are public domain, and are required to generate the RLE files produced by PSR.

B.3 Using PSR.

Some of the features of this system include randomly generated particles about a centroid, user defined particles, and particle spawning. All particle systems are generated from a centroid. The shape of the centroid can be selected by the user. The placement (origin) of the centroid is specified by the user in world (Cartesian) coordinates. The particle system uses spherical coordinates to define particle placement. This feature allows for easier placement of particles around a spherical base. The origin of the particle is defined by its relationship to the centroid, and is translated by the origin of the centroid before rendering. The spherical coordinate system can be over-ridden by an option in the data file.

Particles and centroids are stored in a dynamic linked list. For each time step, the list is first checked for new particles being spawned. The new particles are generated, and added to the list structure. The list is then checked for particles which have died. Dead particles are marked as such, and no longer displayed. The view volume requires specification of a viewer's position, where the camera is looking, a view plane

normal, and a view up vector. The user must also specify viewport dimensions for rendering, and world limits (including front and back) for clipping. This section will describe the features of the particle renderer, and explain how they affect the overall picture.

B.3.1 Centroids.

This section will describe the centroid class, and the features which affect the final product.

B.3.1.1 Axis.

The axis system of the centroid can be specified as having either cartesian or spherical coordinates. The axis defines the base coordinate system of the centroid. The keywords **cartesian** or **spherical** will follow the token **centroid.**

B.3.1.2 Shape.

The centroid shape will contain one of an enumerated type. Currently, only spherical shapes are supported, but the parser will accept **sphere, cube,** or **plane** following the token **shape**.

B.3.1.3 Color.

The color of the centroid is held here. Color is specified by and rgb triple with values between zero and one. The rgb triple is specified immediately following the token **color.**

B.3.1.4 Diameter.

The centroid diameter is the size of the centroid. This value specifies the diameter for spheres, the side length for cubes, and the side length for planes. A single float value is specified after the token **diameter.**

B.3.1.5 Origin.

This variable defines the center of the centroid at the beginning of its life. Currently, this value does not change during the life of the system. The triple specifying the origin

of the centroid follows the token **origin.** The coordinate system of the origin was specified after the token **centroid**.

B.3.2 Particles.

This section will describe the attributes of the particle class which are important to the end user. The data entries for a particle will start with the token **particle** or the token **generate**.

B.3.2.1 Color.

As with the centroid, this variable describes the color of the particle as an RGB triple. The token **color** is followed by the rgb triple.

B.3.2.2 Diameter.

Diameter is a hook to allow particles of different sizes to be rendered. It will be used to describe the width of the drawn line. This feature is not currently implemented.

B.3.2.3 Origin.

The particle's origin is in relation to the center of the centroid, it is defined in the centroid coordinate system. In other words, for spheres, the particle's origin will be define using phi, theta, and radius, while Cartesian (XYZ) coordinates will be used for all other centroid shapes. The triple defining the origin immediately follows the token **origin**.

B.3.2.4 Fade Time.

This hook will be used to fade out the tail of a particle after its death. The purpose of this feature is to keep particles from just winking out and destroying the realism of the scene. This feature is not yet implemented.

B.3.2.5 Tail Length.

Tail length is the maximum length of the particle's tail. This feature prevents particles from leaving a permanent trace if used. Default tail length is -1. A negative tail length

signals the system to ignore this feature. The length of the tail is specified as a float immediately following the token **tail**.

B.3.2.6 Direction.

Original used in conjunction with the velocity, this attribute helped specify the particle's movement. This feature is obsolete.

B.3.2.7 Time of Death.

This feature allows the user to specify the time at which a particle dies. The time is relative to the particle's creation, not system time. The time of death is specified in seconds following the token **death**.

B.3.2.8 Time of Spawn.

This feature allows the user to specify when to spawn a new particle system from the particle. The new system has it's origin at the current position of the particle spawning it. Time is relative to the creation of the parent particle, not system time, and is specified immediately after the token **spawns**.

B.3.2.9 Spawn From.

This feature will allow the user to specify either a file to be used for the information required to spawn a new system, or tell the system to generate the particles based on some minimal information. The token **from** follows the time value to specify the source of the data to generate the spawned system. The acceptable tokens are **generate** for a randomly generated system, or **read** for input from a new file. Currently, this feature only supports randomly generated spawned systems.

B.3.2.10 Velocity.

Velocity is specified as either a Cartesian(XYZ) or spherical value. Currently only linear movement is supported by the system. The velocity triple follows the token **velocity.**

B.3.2.11 Intensity.

A modifier for the particle's intensity. This token will take a float value in the range zero to one. I see no real use for this feature, but left the ``hook'' in case somebody else might find it useful.

B.3.3 The Environment.

Although some of these features have already been mentioned, this section will discuss some of the user controllable features of the environment.

B.3.3.1 Ambient Color.

The ambient color is another name for background color in the particle system. It is also defined by an RGB triple. The default color for the background is black.

B.3.3.2 Viewport.

The viewport for display is defined as two min/max pairs of integers. Remember, the viewport is inclusive, that is a min of zero and a max of ten is eleven pixels wide.

B.3.3.3 World.

The world is the limits of the particle system for clipping. Again, a min/max integer pair is specified for the world's limits. Additionally, a front and back clipping plane are required.

B.3.3.4 View.

Again, the view is defined by the camera position, the center of interest, a view plane normal, and a view up vector. Intimate knowledge of the right-hand rule is recommended before changing these values.

B.3.3.5 Time Increment.

The system allows the user to specify both the time increment and the length the system will run. It is recommended that the user use a large step value and a small duration when initially rendering images. Time is defined in seconds or fractions of seconds. The step variable allows the use to set the step size for the time increment.

The seconds can be used to specify the number of seconds the system will run, while the step variable specifies a specific number of steps to execute for.

B.3.3.6 View Changes.

The user can change the view after the system has grown for the specified number of steps. The change in view is started with the keyword **change.** This keyword is followed by the change in view. The change in view is specified by the starting viewplane normal and view up vectors (svpn and svup). If these values are not specified, the current values for VPN and VUP are used. The user also specifies the finishing VPN and VUP using the tokens fvup and fvpn. The token **freeze** on the line indicates the system will not grow during the view change. The tokens **step, seconds**, and **for** are once again used to specify the time or number of frames to use for the change in view.

B.3.4 Implementation.

This section describes how to use the system, and what is required for the data and control files.

B.3.4.1 File Formats.

This system does not require file suffixes, nor does it append them. It is recommended that the user use the .con suffix for control files and .dat for data files. The control file format is shown in table 1 and the data file format is shown in table 2. The output file will be given an extension indicating the frame number. For example, if the user chooses the output file name rcs1, the first file created will be rcs1.1, the second file will be rcs1.2, etc. until the last frame is generated.

B.4 Sample Control File.

The following control file sets up the initial view for a series of frames to be transferred to video tape. The viewport size is correct for conversion to an NTSC monitor, and the view parameters set up the initial view, then rotate for a total of 360

degrees on two of the three axis. The rotation on each axis is split into four pieces to insure a smooth rotation in the proper direction.

```
viewport 0 643 0 485 world -320 320 -240 240
position 0.0 0.0 350.0 coi 0.0 0.0 0.0 vup 0.0 10.0 0.0 vpn 10.0 0.0 0.0 clip 300.0 -400.0
step 0.06 for 25
change freeze frames 50 rotate fvup 0.0 0.0 10.0 fvpn 10.0 0.0 0.0
change freeze frames 50 rotate fvup 0.0 -10.0 0.0 fvpn 10.0 0.0 0.0
change freeze frames 50 rotate fvup 0.0 0.0 -10.0 fvpn 10.0 0.0 0.0
change freeze frames 50 rotate fvup 0.0 10.0 0.0 fvpn 10.0 0.0 0.0
change freeze frames 50 rotate fvup 0.0 10.0 0.0 fvpn 0.0 0.0 10.0
change freeze frames 50 rotate fvup 0.0 10.0 0.0 fvpn -10.0 0.0 0.0
change freeze frames 50 rotate fvup 0.0 10.0 0.0 fvpn 0.0 0.0 -10.0
change freeze frames 50 rotate fvup 0.0 10.0 0.0 fvpn 10.0 0.0 0.0
change freeze frames 50
```

Line	Keywords	Description
1	viewport	<xmin> <xmax> <ymin> <ymax>
	world	<xmin> <xmax> <ymin> <ymax>
2	position	<x> <y> <z>
	coi	<x> <y> <z>
	vup	<x> <y> <z>
	vpn	<x> <y> <z>
	clip	<front> <back>
3	step	<step size>
	seconds	<duration in seconds>
	for	<duration in number of steps>
4+	change	"Indicates change in view"
	freeze	"Stop system growth during view change"
	rotate	"Rotate view using vup, vpn"
	svup	<start x> <start y> <start z>
	svpn	<start x> <start y> <start z>
	fvup	<finish x> <finish y> <finish z>
	fvpn	<finish x> <finish y> <finish z>
	step	<set step size>
	seconds	<duration of change in seconds>
	for	<number of steps>

Table 1. Control File Syntax

Line	Keywords	Description
1	centroid	{spherical, cartesian}
	shape	{sphere, cube, plane}
	diameter	<size of centroid>
	color	<r> <g> <b>
	origin	<location of centroid>
2+	centroid	{spherical, cartesian}
	shape	{sphere, cube, plane}
	diameter	<size of centroid>
	color	<r> <g> <b>
	origin	<location of centroid>
2+	generate	"Generate Particle List"
	number	<base> <variation>
	system	{spherical, cartesian }
	color	<r> <g> <b> <variation>
	type	{ spherical, cartesian }
	velocity	<x> <y> <z> <variation>
	tail	<base> <variation>
	death	<base> <variation>
2+	particle	"Generate Particle List"
	system	{spherical, cartesian}
	origin	<location of centroid>
	color	<r> <g> <b>
	velocity	<x> <y> <z>
	length	<tail length>
	dies	<time of death>
	spawns	<time of spawn>
	from	{generate, read}
	type	{spherical, cartesian }

Table 2. Data File Syntax

B.5 Sample Data File.

```
centroid spherical shape sphere diameter 0 color 0.2 0.2 0.7 origin 2.0 55.2 1.0
particle origin 0.0 0.0 0.0 dies 2.2 spawns 2.2 from generate velocity 0.2 49.4 0.3 type cartesian color 0.9 0.05 0.01 length 22.5
generate number 15000 45 velocity 20.85 15.7 150.82 70.35 color 0.0 0.6 0.05 0.13 tail 25.5 3.25 death 0.7
centroid spherical shape sphere diameter 0 color 0.2 0.2 0.7 origin 2.0 25.2 41.0
particle origin 0.0 0.0 0.0 dies 2.0 spawns 2.0 from generate velocity -2. 32.4 -0.3 type cartesian color 0.9 0.05 0.01 length 22.5
generate number 15000 45 velocity 20.85 15.7 150.82 70.35 color 0.80 0.06 0.05 0.13 tail 45.5 3.25 death 1.2
centroid spherical shape sphere diameter 0 color 0.2 0.2 0.7 origin 92.0 155.2 10.0
particle origin 0.0 0.0 0.0 dies 1.5 spawns 1.5 from generate velocity 8. 22.4 5. type cartesian color 0.9 0.05 0.01 length 32.5
generate number 15000 45 velocity 20.85 15.7 150.82 70.35 color 0.02 0.06 0.805 0.13 tail 25.5 3.25 death 0.5
```

B.6 Command Line Options.

PSR has a few command line options the user should be familiar with. The first command option is the **-d** option. This option is used to specify the path and name of the data file. The **-c** option allows the user to specify the path and name of the command file. The **-o** option allows the user to specify the name of the output file. The output file name should not have an extension, PSR appends the frame number to the end of the file name. The final option is the **-i** option. This allows the user to set the information level. Information levels range from **0** to **4**, with **0** being minimal information, and **4** being debug level information. If these parameters are not supplied, default values will be used. The default information level is **1**, the default data file is **test1.dat**, the default output file is **temp**, and the default control parameters are internally set. A sample command line entry might be:

psr -d ../cnv/rcs1.dat -c ../rc.con -o /leo2/def/rcs1 -i 2

B.7 Summary.

This particle system renderer (PSR) generates images using a Z-Buffer and Wu anti-aliased lines. It generates either static or random particle systems, as determined by the users. Although it still has room for improvement, I believe it represents a pretty good first cut. It was a valuable learning experience developing a system from scratch without any hard guidelines or restrictions. With a little more effort, I believe PSR will be a valuable commodity for the graphics department at AFIT.

Appendix C. Using VIPER and MAP

C.1 Introduction

Both VIPER and MAP are the results of previous thesis work at AFIT. VIPER was modified by Lentz in support of his thesis effort, and I modified MAP to work with volumetric RCS data. The version of the software installed in the subdirectories in my thesis area. This appendix explains the installation of the software and adjusting the run time parameters.

C.2 Software Installation

Before using either VIPER or MAP, they will need to be installed on the system you are using. If the source is no longer available on the system, then contact the system administrator for a copy of the backup containing this thesis effort. Makefiles are located in each of the required directories for rebuilding the software. MAP is located in the directory MAP. The makefile is setup for the AT&T C++ compiler. The Unix command make will build the software for you. The source for VIPER is located in the directory VIPER. Again, a makefile in the directory is setup to build VIPER on a Unix system using the AT&T C++ compiler. VIPER requires the Utah Raster Toolkit to install successfully. If the toolkit is not installed on the system, contact the system administrator.

C.3 Running MAP.

Running MAP first requires establishing a configuration file with the required runtime parameters. The map program creates a rectangular grid, and samples input data from the files specified in the configuration file. The program will build a grid of variable, or constant plane spacing, and create an output file conforming to VIPER's input requirements.

Two sampling methods are offered; A-Buffer, and Z-Buffer. If A-Buffer is chosen, either a density weighting, or a Gaussian filter can be specified.

The program expects the grid input file structure in the form used by the Computational Fluid Dynamics Group at WPAFB. These grids come in two files: one for the grid indices, and one for the grid data. The first line of the index file contains the number of data points, followed by the x indices, the y indices, and then the z indices. The data file contains the data points in the index order. Both files are in binary format.

The -f option is the only command line option available with the map program. It forces the program to maintain the input data coordinate ranges when using the variable spaced method for plane construction.

The user configurable parameters for the program should appear in the configuration file. The parameters are listed below in the order expected by the map program. Each parameter is briefly explained, and a sample configuration file is given later.

C.4 USER CONFIGURABLE PARAMETERS

1 The input grid index file (paths acceptable)

2 The input grid data file (paths acceptable)

3 The output rectangular grid file name (paths acceptable)

4 A `1' or `0' for carving the input grid. A `1' means carve the grid, a `0' means do not carve the grid. If carving the grid, the next six lines of the configuration file should have the i, j, and k lower and upper bounds, (one bound per line, in the given order).

5 A `1' or `0' for the viewing parameter. A `1' means view pressure numbers, a `0' means view mach numbers.

6 A `1' or `0' for the number of planes to use in constructing the grid. A `1' means use the default (100 per axis), a `0' means other values are specified. If other values are used, all three should appear on the next line (x, y, and z).

7 A `1' or `0' for plane spacing method. A `1' means use constant spacing, a `0' means use variable spacing.

8 A `1' or `0' for the sampling method. A `1' means use Z-buffer, a `0' means use A-Buffer.

9 A floating point value for the filter radius multiplier.

10 If a-buffer sampling was selected, a `1' or `0' to specify the filtering function. A `1' means use the Gaussian filter, a `0' means use the density weighting filter.

11 If the Gaussian filter is used, a floating point number for the number of standard deviations spanning the filter radius.

C.5 MAP EXAMPLE

Running MAP is as simple as typing:

```
map -f map1.cfg
```

SAMPLE CONFIGURATION FILE MAP1.CFG

```
/usr/eng/dtisdale/thesis/psr/cnv/rcsi.dat     * The input grid index file
/usr/eng/dtisdale/thesis/psr/cnv/d.dat        * The input grid data file
rcs2.bin                                      * The grid output filename
0                                             * Indicates do not carve the grid
0                                             * Produce a mach shell grid
1                                             * Use the default number of planes
0                                             * Use variable plane spacing method
0                                             * Use the A-buffer sampling technique
1.1                                           * The filter radius multiplier
0                                             * Use the density weighting function
```

C.6 Using VIPER

Using VIPER is straightforward, requiring a properly formatted data file. MAP generates the proper data file format for the user, alleviating this potential problem. VIPER does require that you use a rectangular grid of data points. Again, MAP generates a rectangular grid of data for the user. Failure to use rectangular grid will generate unpredictable results. The command line for VIPER allows one of two use options. The -k option will place the image in the lower left corner of the screen, while the -c option places the image in the center of the screen. These options are affected by the version of the RLE display routine being used, and the version of the operating system. Although VIPER still requires the command line parameter, the 3.0 version of the Utah Raster Toolkit contains a get4d command that allows the user to specify the location of the image. The VIPER command line is:

viper -option input output

The input option is the name of the properly formatted input file. The output option is the name of the output file. The output file name should have a .rle extension. When VIPER starts, it will prompt the user for the configuration parameters. The next section describes the configuration entries.

C.6.1 Feedback.

This value determines if VIPER will provide feedback during the image generation. Feedback is in the form of an asterisk (*) printed for each data value read. I strongly recommend not using this option. For large data files, this option will significantly increase the execution time of VIPER.

C.6.2 Scaling.

This option allows you to set the scaling factors for the x, y, and z axis. For most cases, use a scale factor of 1.0. This renders the image in the original volume. If the scale factor entered is not the same for all three axis, the image will be distorted.

C.6.3 Volume Trimming.

This option allows you to trim off portions of the data volume. To decline this option enter a **0**. If you enter a **1**, you will be prompted for information regarding trimming each of the six faces. This option was not used with the RCS data.

C.6.4 Object Distances.

These options allow you set the distance from your eye to the center of the view volume, and your eye to the viewing plane. I have found that using the volume size plus 1 is a good start for the distance to the center of the view volume, and **1** is a good distance to the viewing plane.

C.6.5 Data Orientation.

The next series of prompts allows the user to change the orientation of the data. For the initial run, set all three rotations to **0**. Then, on subsequent runs, I suggest rotating on one axis at a time to get a better feel for the data set.

C.6.6 Changing the View Plane Dimensions.

The next question asks if you wish to change the screen size. If you enter yes, you must enter the new horizontal and vertical measurements for the screen. The units are in pixels for this option.

C.6.7 Ambient Light Level.

This option asks the user for the ambient light in the volume. This light is in addition to any of the light sources specified later. A value of **0.8** seems to work well for RCS data.

C.6.7 Number of Light Sources.

The next prompt is for the number of light sources. The user must enter a number between **1** and **6**. Initially, the user should use a single light source. You will then be asked for the location of each light source requested. Remember, your eye is at the origin for the light specification, and the location of the light source is relative to the viewer, not the volume.

C.6.8 Target Values.

The system will then ask you for the number of target values to find. For RCS data, enter 3 to find the contribution of all four polarities. For each of the target values specified, you will be prompted for the target value, the variance, the RGB triple associated with the value, and its opacity. For the RCS model, use the value 50, 150, and 200 for the target values. A variance of 8 to 12 will yield good results. For the RGB triples, I stayed with the primary colors, using 0.8 for the dominant color, and 0.2 for the minor colors

C.6.9 Border Matte.

The final question prior to rendering will be the color of the border matte. This is mainly an esthetic issue, and the value is up to the user. A good neutral blue could be specified by **0.2 0.2 0.4**.

C.6.10 Repeating the Session.

After the image is rendered, the system will ask you "One more time?". This option allows you to render another image without reloading the data set. This can be time saving for large data sets.

C.7 Sample Command Line.

Although there is extensive interaction once VIPER starts, the command line for invoking it is very simple. A typical command line might be:

viper -k rcs2.bin

C.8 Summary.

MAP and VIPER are both useful tools for the display of volumetric data. VIPER is restrictive in that it requires regularly spaced data for input. MAP will convert irregularly spaced data to a rectangular grid using trilinear interpolation. Both tools are fairly well documented, and straightforward in their use.

BIBLIOGRAPHY

[Bal 89] Balanis, Constantine A. *Advanced Electromagnetic Engineering.* New York:John Wiley & Sons, 1989.

[Bri 88] Bridges, Capt D. *Volumetric Rendering Techniques for the Display of Three-Dimensional Aerodynamic Flow Field Data.* MS Thesis, Air Force Institute of Technology, December 1988.

[Cli 88] Cline, H.E., Lorensen W.E., Ludke S., Crawford,C.R., and Teeter,B.C., "Two Algorithms for Three-dimensional Reconstruction of Tomograms," Medical Physics, 15(3):320-327 (May/June 1988).

[Eko 91] Ekoule, A., et al. "A Triangulation Algorithm from Arbitrary Shaped Multiple Planer Contours," *ACM Transactions on Graphics*, pages 182-199 (April 1991).

[Fol 90] Foley, James D., et al. *Computer Graphics, Principles and Practice.* Reading Massachusetts: Addison-Wesley, 1990

[Kau 91] Kaufman, Arie, *Volume Visualization*, IEEE Computer Society Press, Los Alamitos, California, 1991.

[Kor 83] Korein, Jonathan and N. Badler, Temporal Anti-aliasing in Computer Generated Animation, SIGGRAPH, 1983, Volume 17:377-388 (1983)

[Lan 79] Lane, J. and L. Carpenter, A Generalized Scan Line Algorithm for the Computer Generation of Piecewise Polynomial Surfaces, Computer Graphics and Image Processing, 1979, Volume 11, pp. 290-297

[Len 89] Lentz, Capt P. *Sampling Data Onto Rectangular Grids For Volume Visualization.* MS Thesis, Air Force Institute of Technology, December 1989.

[Mar 90] Marhefka, R.J. *Radar Cross Section - Basic Scattering Code, RCS-BSC (Version 2.0) User's Manual.* ElectroScience Laboratory, The Ohio State University, Department of Electrical Engineering, Columbus, Ohio 43212, 1990.

[Mey 91] Meyers, D. et al. "Surfaces From Contours: The Correspondence and Branching problem," *Proceedings of Graphics Interface '91*, pages 246-254 (June 1991)

[Par 82] PARAMOUNT," Star Trek II: The Wrath of Khan, June 1982

[Pon 92] Pond, Capt D. *A Synthetic Environment for Satellite Modeling and Satellite Orbital Motion.* MS Thesis, Air Force Institute of Technology, December 1992.

[Ree 83] Reeves, William T., Particle Systems - A Technique for Modeling a Class of Fuzzy Objects, SIGGRAPH, 1983, Volume 17:359-376

[Ree 85] Reeves, William T. and R. Blau, Approximate and Probabilistic Algorithms for Shading and Rendering Particle Systems, *Proceedings of SIGGRAPH '85*, 19(3):313-322 (July 1985)

[RTF 92] Conversations with the Radar Engineers at the Radar Test Facility, Tyndall AFB FL, May 1992.

[Smi 82] Smith, Alvey Ray and others, GENESIS DEMO DOCUMENTARY, June 1982,

[Sti 83] Stimson, George W., *Introduction to Airborne Radar,* Hughes Aircraft Company, El Segundo, California, 1983

[Sty 91] Stytz, Martin R. et al. Three-Dimensional Medical Imaging: Algorithms and Computer Systems, ACM Computing Surveys, 23(4):421-499 (December 1991)

[Swe 86] Sweetman, Bill StealthAircraft, Motorbooks International Publishers & Wholesalers Osceola Wisconsin, 1986

[Swe 92] Swerling,P. "Sectorized Statistical Target RCS Characterizations," Course Handout for WENE 445, Low Observables, (original Dated July 13, 1992) 1992.

[Tho 86] Thomas, Spencer. *Design of the Utah RLE Format.* University of Utah, Department of Computer Science, 1986.

[Udu 91] Udupa, J. and G. Herman, *3D Imaging in Medicine*, CRC Press, Boca Raton Florida, 1991.

[Ups 88] Upson, C and M. Keeler, V-BUFFER: Visible Volume Rendering, SIGGRAPH, 1988, pp. 59-64

[Woj 92] Wojszynski, Capt T. *Scientific Visualization of 3D Radar Cross Section Data.* MS Thesis, Air Force Institute of Technology, December 1992.

[Wu 91] Wu, Xiaolin, An Efficient Anti-aliasing Technique, Computer Graphics, July 1991

Vita

Captain David J. Tisdale was born on November 4, 1961 in Vallejo California. He graduated from Beaver Local High School in Lisbon, Ohio in 1979, then entered the United States Air Force Academy. He graduated in 1983 and received a Bachelor of Science in Computer Technology. His first assignment was to the 31st Student Squadron in Columbus, Mississippi. In 1984 he was assigned to the 83rd Fighter Weapons Squadron at Tyndall Air Force Base, Florida were he worked as the Database administrator for the Weapons System Evaluation Program. In June, 1985 Capt Tisdale transferred to the 4484th Test Squadron at Tyndall Air Force Base in Florida. He worked there as a computer systems engineer until May, 1991. In May 1991 he entered the School of Engineering, Air Force Institute of Technology to pursue a Master of Science Degree in Computer Engineering.

Permanent address:

14637 Avalon Avenue
Baton Rouge, Louisiana
70816

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 1992	Master's Thesis

4. TITLE AND SUBTITLE
METHODS FOR VIEWING RADAR CROSS SECTION DATA IN THREE DIMENSIONS

5. FUNDING NUMBERS

6. AUTHOR(S)
David J. Tisdale, 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/92D-18

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

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)*

Radar Cross Section data is an important factor in the design of modern fighter and bomber aircraft. Minimization of the reflected radar energy is one of the key issues when choosing shapes and materials in new aircraft. Visualization of the energy scattered back to a radar is neither intuitive nor easy. The mission planner and pilot need to gain an understanding of the vulnerabilities inherent in the design of the systems they use. The advent of relatively low cost graphics workstations has made their use affordable in applications inconceivable only a few short years ago. This thesis examines three graphics rendering techniques and their applicability to the display of three-dimensional radar cross section data. This study looks at three tools, PSR, a particle system renderer; the Satellite Modeler, a three-dimensional surface renderer; and VIPER, a volumetric renderer for their applicability, utility, and ease of use in the display of radar cross section data. Each of the systems was tested using the same data set, and the results compared later in this treatise. Although the purpose of this study is to determine the applicability of different techniques, the results can be used to further develop fast, efficient rendering systems for the visualization of radar cross section data.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Graphics, Multi-polarity Vectorized Data, Scientific Visualization	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