Rowan University

Rowan Digital Works

Theses and Dissertations

7-31-2009

Integration of multiple data types in 3-D immersive virtual reality (VR) environments

George Demetrius Lecakes Jr.
Rowan University

Follow this and additional works at: https://rdw.rowan.edu/etd

Part of the Electrical and Computer Engineering Commons

Let us know how access to this document benefits you - share your thoughts on our feedback form.

Recommended Citation

Lecakes, George Demetrius Jr., "Integration of multiple data types in 3-D immersive virtual reality (VR) environments" (2009). *Theses and Dissertations*. 641.
https://rdw.rowan.edu/etd/641

This Thesis is brought to you for free and open access by Rowan Digital Works. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Rowan Digital Works. For more information, please contact LibraryTheses@rowan.edu.

INTEGRATION OF MULTIPLE DATA TYPES IN 3-D IMMERSIVE VIRTUAL REALITY (VR) ENVIRONMENTS

by
George Demetrius Lecakes, Jr.

A Thesis

Submitted in partial fulfillment of the requirements of the
Master of Science Degree
of
The Graduate School
at
Rowan University
July 31, 2009

Thesis Chair: Dr. Shreekanth Mandayam

©2009 George D. Lecakes, Jr.

ABSTRACT

George D. Lecakes, Jr.
INTEGRATION OF MULTIPLE DATA TYPES IN 3-D IMMERSIVE VIRTUAL REALITY (VR) ENVIRONMENTS
2008/09
Dr. Shreekanth Mandayam
Master of Science in Engineering (Specialization in Electrical Engineering)

Intelligent sensors have begun to play a key part in the monitoring and maintenance of complex infrastructures. Sensors have the capability not only to provide raw data, but also provide "information" by indicating the reliability of the measurements. The effect of this added information is a voluminous increase in the total data that is gathered. If an operator is required to perceive the state of a complex system, novel methods must be developed for sifting through enormous data sets. Virtual reality (VR) platforms are proposed as ideal candidates for performing this task – a virtual world will allow the user to experience a complex system that is gathering a multitude of sensor data and are referred as "Integrated Awareness" models.

This thesis presents techniques for visualizing such multiple data sets, specifically – graphical, measurement and health data inside a 3-D VR environment. The focus of this thesis is to develop pathways to generate the required 3-D models without sacrificing visual fidelity. The tasks include creating the visual representation, integrating multi-sensor measurements, creating user-specific visualizations and a performance evaluation of the completed virtual environment.

ACKNOWLEDGEMENTS

I would like to thank Dr. Shreekanth Mandayam for the opportunity to work in this field. These projects have introduced me to the area of virtual reality and visualization. I would like to thank Dr. John Schmalzel, Dr. Fernando Figueroa, Jon Morris and Mark Turowski and everyone else at NASA Stennis Space Center in Mississippi for the chance to work on the ISHM visualization project. I would like to thank my committee members, Dr. Beena Sukumaran and Dr. Gina Tang for taking the time to review this thesis.

I would like to thank both my father and mother, George and Mary Caroline Lecakes for the freedom they provided me to find my own direction in life. I would also like to thank those who have helped me both personally and professionally here at Rowan University. KJ Brown, Pat Giordano, Kevin Garrison, Hector Suarez, Phil Mease, Michael Russell, Steve Latman and Michael Bloom; thanks to all of you for making my stay at Rowan memorable. I would also like to thank Jennifer Marie Curtis, without her enduring support and love I would never have come so far over the past two years.

Finally, I would like to thank Dr. Nathan Sovik, Dr. Ramona Travis, and Ms. Nancy Bordelon the opportunity to participate in the NASA Graduate Student Research Position (GSRP) which has funded me through my graduate degree (NASA Order No. NNS06AC08P, Grant Number NNX07AO92H).

Table of Contents

List of Figures

List of Tables

CHAPTER 1 : INTRODUCTION

The task of creating data visualizations is not a simple endeavor. Whenever the human eye views a scene a myriad of mental tasks deconstruct the image into simple components. Size, shape, color, texture orientation and other properties are analyzed and divided into understandable concepts. Associations are made, important areas are noted and many items are filtered out. The human mind has developed the ability to analyze and sift through enormous visual information through hundreds of thousands of years of evolution. Scientific visualization aims to utilize these finely tuned assets of the human mind in identifying patterns and interpreting results. With so many parameters that the mind can process the task of creating intuitive visualizations that enhance understanding can be difficult.

Virtual reality is a tool utilized for viewing spatial information. Through the use of immersive displays such as large projectors or head-mounted screens the user is transported into a synthetic world. Virtual reality environments provide the user with the ability to transform two-dimensional screens that lack depth information into three-dimensional windows into the environment. Through stereoscopic rendering techniques the human eye is fooled into believing that elements exist outside of the two-dimensional screen. Couple this with head-tracking software, which adjusts the view based on the angle the user is to the screen and the user is made to believe they are seeing reality.

In the realm of virtual reality visualization, three core data types have been identified: graphical, measurement and health [1]. Each of these data types must be presented in a visual manner to the user. Based on the source of the data, there are many different ways in which the visualization can be achieved. Various means can be used to convey temperature, pressure and other measurement information. Scalar and vector data require visualization fields to display the changes in magnitude and direction for flow information. For images, the data can be captured and the properties of an object extracted to create 3-D models.

With the various types of data and many methods of visualizing information, the need to develop a series of algorithms and procedures for creating virtual worlds quickly and consistently is a necessity.

1.1 Applications

Scientific visualization is a powerful tool that can be utilized in many fields. Typically, it is utilized to visualize information in a spatial domain. Several disciplines have used scientific visualization to create insightful imagery to help describe underlying principles that may be lost in compact mathematical forms.

The visualization of fields and flow has important application in many areas. Visualization of flowing fluids has evolved from simple sketches to complex computer simulations [2].

The medical profession has also utilized visualization to create three-dimensional images that describe many biological functions. 3-D models of the human body have been created from various forms of scanning equipment to create pictures of the human body in action. There are numerous other areas in the medical industry which have utilized scientific visualization to create educational and explorative 3-D models.

In the realm of engineering, analysis plays a crucial role in developing models to simulate interactions between objects. Coupling finite-element analysis with visualization provides the ability to watch changes in the geometry of an object over time whether it be stress, temperature, pressure or other variable. The automotive industry utilizes this technique to create simulations of car crashes and the extent of damages [3]. New developments have been made in the area of fluid flow thanks to the analysis and visualization of whales and porpoises, with the visualization providing the 'why' to how certain shapes outperform others in nature [4].

Through visualizations, abstract information can be conveyed through properties of color, size or shape. Even complex interactions of variables resulting in the health or state of a system can be visualized to provide a simple measure of how a system is performing.

Finally, visualization is very important in understanding the structure of an object. The geometric shape of an object can play a crucial role in the properties it possesses. Even simple and small object such as sand can have vastly different mechanical properties based on the shape and size of the particle. Through visualization, it is possible to compare specimens and gain insight into the reasons why various sands require different considerations when building upon them. Virtual reality also provides an excellent tool for inspection of the geometric elements of an object, such as a building to gain a personal understanding of its shape, size and characteristics. Complex objects such as rocket test stands can be viewed from any angle in a virtual environment to gain an introspective understanding of the structural layout and be used as a tool to educate an individual on the layout of a structure.

1.2 Motivation

The goal of visualization is to garner insight from a pool of data through images. Therefore it is important to have algorithms in place that can create these visualizations regardless of the dataset presented.

Data visualization can be broken down into three categories, graphical, measurement and health information. It is important to provide graphical information (3-D models representing objects) so that a frame of reference can be established of the environment. Viewing data without knowing where or what it pertains to can inhibit understanding. Measurement data pertains to the data gathered from sensors. Functional data is created through manipulation of measurement data or through simulations. These data types require several 3-D visualization algorithms to visualize properly.

The importance behind visualizing 3-D particles of sand builds on the work of P. Giordano, J. Corriveau and D. Barrot [5] [6] [7]. Describing the 3-D shape of particle aggregate mixtures is important when analyzing particle size and shape. These and many other properties can affect the inter-particle interactions of soil mechanics which can contribute to differences in the shear strength characteristics, particle interaction, granular friction and impact to the environment.

Currently a system is utilized to create a highly detailed and low-error 3-D representation of a particle with an X-ray tomography system. However, this system is not only expensive to purchase but requires extensive computational power and can take up to five hours to scan and several additional hours to reconstruct the model.

There is a definite need for an inexpensive system that could create a model in shorter period of time. Past work has created an algorithm utilizing optical tomography to create a particle of rough approximation to the X-ray tomography system. However, the algorithm was limited in the resolution of the reconstruction and took nearly the same amount of time as the X-ray system. A fast visualization system is needed to increase resolution and decrease overall algorithm processing time.

This updated algorithm can provide 3-D models of sand particles that can be brought into a virtual world. Through user interaction, it can be possible to visually understand the differences in size, angularity and shape from different mixtures. These models could later be utilized inside of a discrete element modeling simulation to determine micro mechanics of granular media.

When creating virtual environments, it is important to be able to quickly create a world. It is dually important that the techniques discovered are relayed to future generations so that the nuances of creating a virtual world are not re-invented or lost. Clear and concise visualization algorithms are required so that the observer can spend their time looking for important trends and not waiting for a demonstration to be developed.

1.3 Objectives

The primary goal of this thesis is to investigate and improve upon existing techniques for developing virtual reality environment visualizations and integrating them with various datasets. The specific objectives are outlined as:

1. A survey of current techniques for:
    a) The design, development and implementation of immersive, navigable and interactive VR environments for advanced scientific visualization.

   b) The design, development and implementation of 3-D modeling algorithms in a VR environment.

2. Specific contributions in the area of designing, developing and implementing 3-D models inside of a VR environment, specifically for:
   a) Integrated awareness models of graphical, measurement and health data
   b) 3-D reconstruction models of tomography data
   c) Applications and validations of methods in a sufficient number of example VR environments
3. Recommendations for future research directions

This thesis will contribute pathways and algorithms for creating data visualizations in a virtual world. Data types will be broken down and coupled with appropriate visualizations. Several virtual reality environments will be created to display these visualizations. In addition, an improved ART algorithm will be presented which utilizes simple image manipulation techniques to create higher resolution models quickly. These contributions will provide the framework for creating most virtual reality visualization environment.

1.4 Scope

The survey of current technologies focused on how visualizations and VR have been utilized in creating immersive worlds is located in the background section of this thesis. The 3-D modeling algorithms researched are covered in the background section. Various algorithms for one to three dimensions are covered in addition to scalar and vector data types.

Development of these algorithms in the Vizard environment is covered through several visualization of temperature data through one and two-dimensional surfaces. Vector visualizations were created that utilize synthetic data to demonstrate the algorithm functioning.

The integrated awareness models are covered in the creation of a virtual environment for NASA Stennis Space Center of a Methane Thruster Test-bed Platform (MTTP) rocket engine test stand. Reconstruction models of 3-D tomographic data are addressed in the creation of an improved method for implementing the Algebraic Reconstruction Technique (ART) and through the use of several algorithms for creating data visualizations.

The recommendations for future work are covered in the conclusion section of this thesis, detailing areas of improvement and further advancement.

1.5 Organization

This thesis is organized as follows. Chapter 1 provides introductory information concerning the difficulties of creating virtual worlds and visually representing data, as well as the possible benefits. Possible applications, the motivations, objectives, scope and organization are presented. Chapter 2 provides adequate background information starting with the basics of visualization to 3-D modeling and finally a detailed account of data visualization. Chapter 3 outlines the approach taken to address integrating data and models into a virtual world. Chapter 4 is an account of the results of creating the virtual world, following the premises outlined in Chapter 3. Chapter 5 is a summary of accomplishments, future recommendations along with the conclusions of this thesis.

1.6 Expected Contributions

This thesis will provide a detailed summary of current technologies for virtual reality, visualization, and data visualization algorithms. It will also provide an introduction to computer graphics and necessary introductory knowledge for creating a virtual environment. Several virtual reality simulations will be covered that visualize one, two and three-dimensional data of both scalar and vector quantities. Additional algorithms are shown for visualizing small data sets such as those from sensors in the NASA Stennis Space Center Methane Thruster Test Bed Platform (MTTP) trailer system. A quicker and higher resolution algorithm will be created to reconstruct objects from optical images. This algorithm will provide new particle models for comparison in a virtual world against the X-ray models. Finally, the flow charts in the approach section will provide the necessary protocols required for creating various virtual environments.

CHAPTER 2 : BACKGROUND

The following section contains pertinent information for understanding many topics concerning visualization, virtual reality and how data is managed in a virtual world. The information here will provide a detailed account of what visualization is, and the many types available, how virtual reality works; the kinds available; and commonly used software platforms. Three basic types of data will be defined and methods for visualizing the information discussed.

2.1 Visualization

Visualization is defined as forming a mental model or mental image of something [8]. Information visualization utilizes the mind's cognitive facilities to entice discovery in datasets. The data utilized in visualization need not be the same kind. Quantitative data such as temperature or pressure readings, ordinal ranking data, categorical data and relationship data can all be viewed through various means in the same visualization. Information visualization is concerned with creating a representation of data to make a solution transparent [9]. Information visualization is typically concerned with the visualization of abstract concepts such as relations and scores. Scientific visualization is concerned with the visualization of physical objects, such as 3-D scans for medical purposes or the display of volumetric temperature gradients. Scientific visualizations create displays which aid in the understanding of complex scientific principles and concepts created from gathered data or computer simulations [10]. Many times, scientific visualization is concerned with data over the spatial domain in two or three dimensions. Visualization provides many positive features for the viewer by condensing enormous amounts of data into comprehensible images. These images can reveal details about the data, how it was collected and possible errors or data artifacts that do not blend in with the rest of the picture [11].

2.2 Virtual Reality

A virtualization is something not real, but generated usually from a computer. Virtual reality is the attempt to replace the operator's surroundings with a computer generated version. VR provides the psychological experience of being surrounded by a false environment through hardware, software, tracking and visual displays [12]. A broader definition is also used, which defines VR as any experience in which the user is effectively immersed in a responsive virtual world [13]. In recent years, the notion of what makes for an immersive environment has changed and does not directly require sensory inputs to fool the operator. It is possible for the user to be so mentally and emotionally involved that they begin to forget it is a synthetic environment as in the case of recent online virtual worlds [14]. In this situation, the hardware does not matter and any input into the virtual world will suffice. This type of immersion is very rare and is not utilized to engage an operator since it deals completely with the user's frame of mind, something not easily manipulated.

Virtual reality, in the case of a projection based system, must allow the operator to be immersed, possess the ability for navigation and be able to partially or fully interact with the environment [15]. By adhering to these three tenets, a feeling of presence is created for the operator in the virtual world. Presence is when the operator believes they are in an accurate and believable depiction of reality [16].

One method of increasing immersion is by capitalizing on the fact that human beings perceive a portion of depth information through stereoscopic vision. By utilizing stereoscopic vision in virtual worlds, the user is able to sense the depth of objects and become further immersed.

2.3 Stereoscopic Depth

Human beings possess two eyes that are roughly 6.4 cm apart from one another [8]. This difference in location means that the human brain receives two slightly different images. The difference between these two images allows the brain to determine relative distances between pairs of objects. Figure 1 illustrates the differences between two images perceived from slightly different angles.

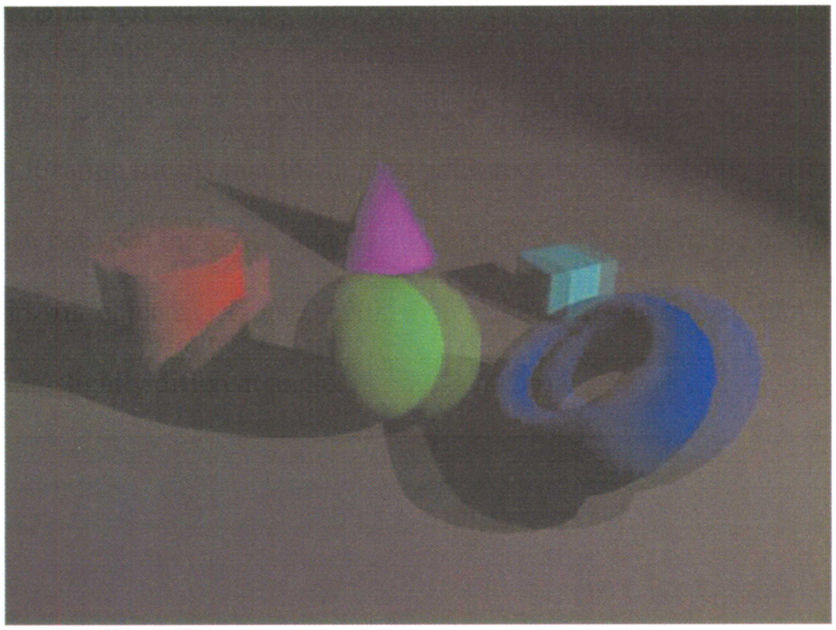


Figure 1: A Stereoscopic view of a scene with two images from two different perspectives overlapped.

The human brain reconciles the differences between the two images by perceiving depth. That is, the human brain merges the images by perceiving that some objects are closer or further away. However, the human brain is limited to images of a particular separation disparity. When the disparity of the images is too great, a condition known as diplopia or double vision occurs. The area in which two images can be successfully fused by the mind is known as Panum's fusional area. Panum's fusional area can be seen in Figure 2 where two eyes perceive a scene from slightly different angles. The area of successful fusion is shaded green with everywhere else a region where double vision occurs.

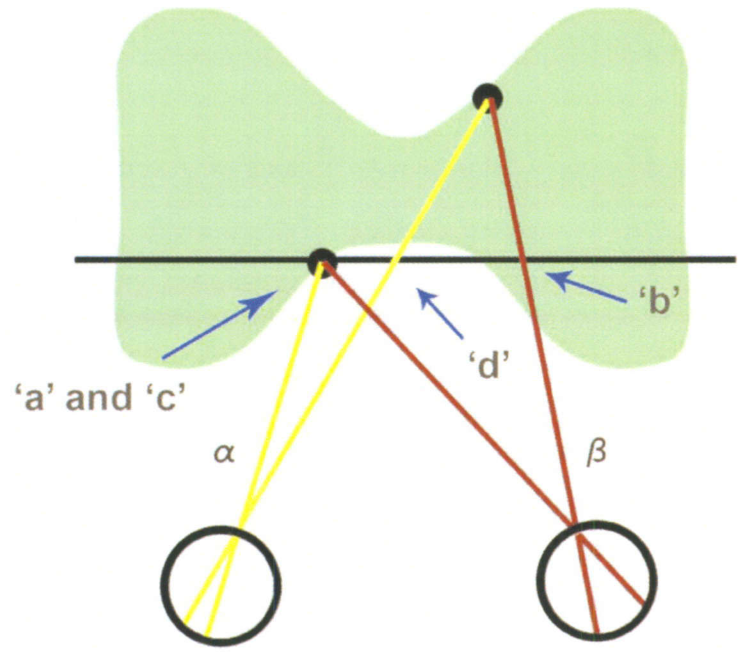


Figure 2: Panum's fusional area.

From Figure 2, two forms of disparity can be defined. Angular disparity, which is the difference between the angular separations of a pair of points from the two eyes, can be determined from the expression:

$$\textit{angular disparity} = \alpha - \beta \quad (2.1)$$

Screen disparity results from the distance between parts of an image on the screen, or:

$$screen\ disparity = (c - d) - (a - b) \quad (2.2)$$

Even when double vision does occur, the human mind can still judge depth, however it is far less accurate. To increase immersion in a virtual environment, it is important to try and utilize Panum's fusional area to create the effect of depth. In a VR environment it is simple to achieve diplopia by moving too close to the screen. At a certain distance the user will be unable to unite the disparate images.

2.4 Types of Virtual Reality

Most immersive virtual reality systems utilize four pieces of technology to create a synthetic environment. The first is the visual and haptic display device which blocks the real world and replaces it with a computer generated version. A graphics rendering system is required to create the 3-D scenes with a refresh rate of at least thirty frames per second (fps), although 60 fps is preferable. The third item is a tracking system that indicates the location of the head or arms of the operator. The tracking system can relay positional information into the virtual world to aid navigation and interaction. Finally, a database of realistic models is needed to create the geometry of the virtual world [13].

Most VR displays utilize stereoscopic views to create the feeling of depth. There are two main types of VR stereoscopic displays: head mounted units and shuttering units.

Head mounted displays have until recently been large devices placed over the head which block the view of the operator and replace it with images on two embedded view screens. In the past decade, recent technological advantages have reduced the size of head mounted displays but they are still seen as awkward.

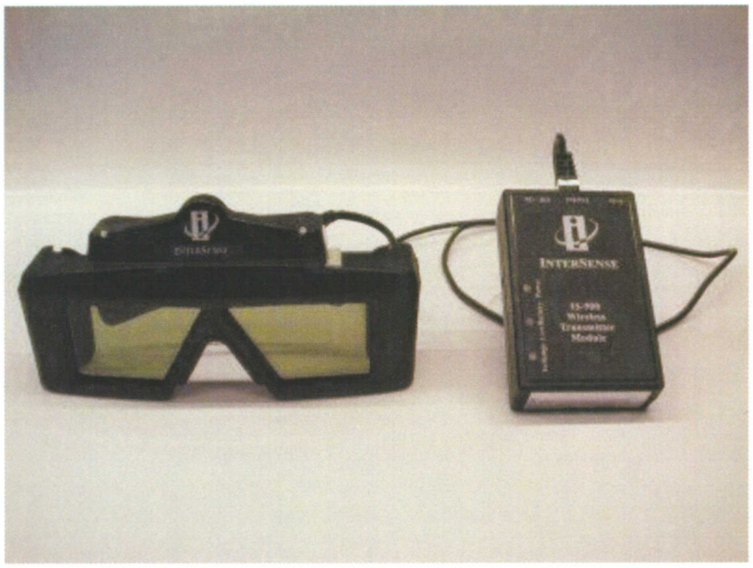


Figure 3: Shutter glasses for projector based VR units. [17]

Another display option is utilizing projector based systems with shutter glasses as shown in Figure 3. Shutter systems utilize a pair of goggles that quickly alternate between opaque and clear. The projector switches views from two different points about eye distance apart inside the virtual world. The shutter glasses are synchronized with the view change and only expose one eye to one particular view. This provides two separate images for each eye on a single projector screen.

There are several different types of projector setups available. Semi-immersive displays utilize a single projector as in Figure 4.

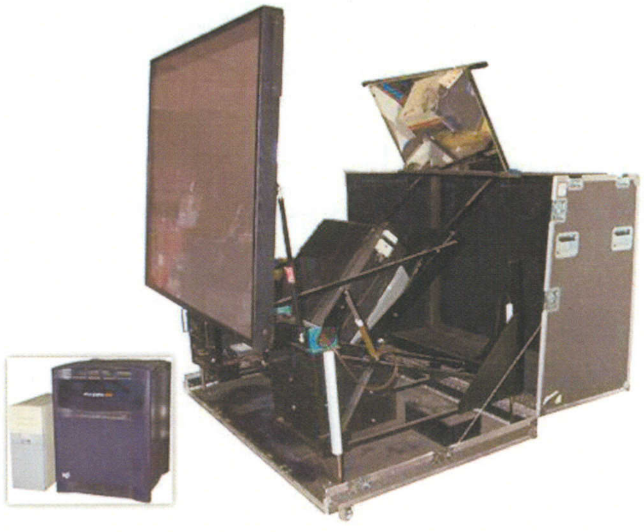


Figure 4: Mechdyne single projector system. [17]

Multiple projectors can be utilized to create a completely immersive experience. These are referred to as CAVE systems and are usually composed of at least three projectors. The operator stands inside the space, surrounded by projector screens and is immersed in the visualization as shown in Figure 5.

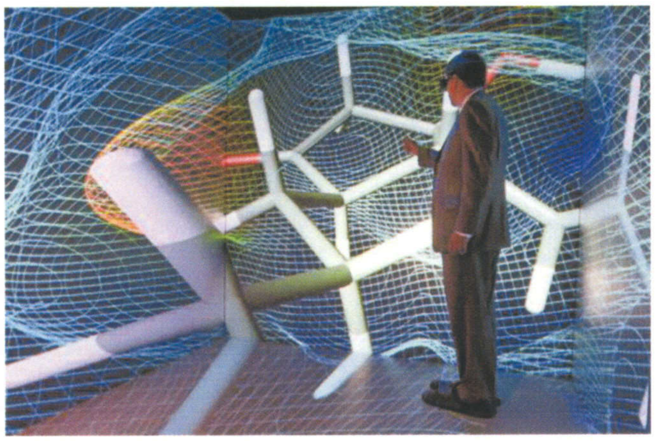


Figure 5: A fully immersive multiple projection system. [17]

2.5 Disadvantages of Virtual Reality

Some users of virtual reality turn off the stereoscopic display due to double-image problems which arise from the perfectly crisp renderings that a machine produces [11]. In the real world objects not in focus are blurred. This blurring allows the mind to assemble two disparate images together easily. In virtual reality systems the ability to locate the focal object is not present so the entire image is sharp and completely in focus and no depth of field is simulated. This makes the double image in VR display very noticeable and distracting. Figure 6 shows a crisp computer generated scene with two overlaid images as well as an image of a real life object seen from two angles. The depth of field in the real image causes blurring and helps to separate foreground and background as well as merge background information together.

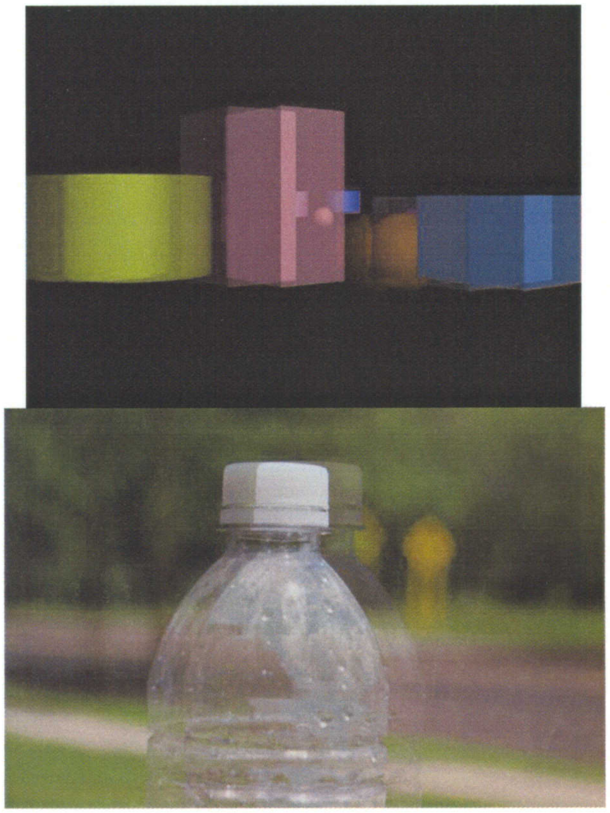


Figure 6: A crisp computer generated scene and a real life image with focal blurring.

Another problem occurs when a semi-immersive display is utilized. Frame cancellation is a term coined to represent the 3-D disparity apparent when an object closer to the user is cut off by the edge of the frame [18]. This causes the illusion of simulated depth information to collapse. The only way to solve this issue is by never seeing the edge of the display, such as in a fully immersive display like the CAVE.

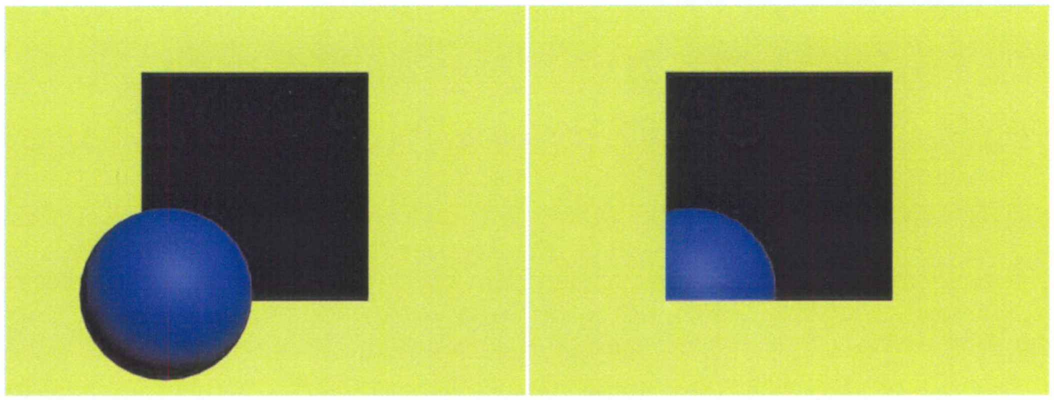


Figure 7: Frame cancellation results when an image that should be in front of the video screen is cropped, making it appear behind the frame collapsing the depth effect VR simulates.

A third issue with VR has to do with the resolution of stereoscopic vision, which utilizes disparities between two images to simulate depth. However, this disparity does not extend to infinity, making for a noticeable depth of only 30 meters. The best range for depth is between 1 and 10 meters from any subject. Anything further back in the virtual world loses depth and becomes flat, making scenes of enormous size benefit less from a VR display.

Another issue concerns the navigation methods of a VR scene when utilizing visualizations to understand complex data. Typical virtual reality systems require the operator to utilize walking or flying navigation techniques. Minimizing cognitive effort and shorting navigation time enhances the pattern recognition process of a human being [11]. An easily usable, fast and effective user interface is required when attempting to discern simple patterns, which is not something VR software contains. VR programs such as vGeo and Vizard utilize flying navigation to move around the world, which can take long periods of time, making for overall pattern recognition difficult. Custom pattern finding interfaces would have to be created for use in these types of virtual environments.

Additional VR disadvantages include a learning curve for beginners. Many initial operators suffer from user disorientation and difficulties in navigation and movement control [19]. These problems can be overcome with time and experience inside a VR environment. However, some individuals seem more apt for navigating a VR environment than others.

2.6 Applications of Virtual Reality

Even with the previously mentioned disadvantages of VR, it has benefited several areas. VR has been used in the world of finite element analysis to create immersive simulations of the deformation of a car body during collision [3]. VR has been utilized to simulate a wind tunnel and the resultant fluid dynamics through visualizing streamlines, iso-surfaces, cutting planes, arrows and numerical values [10]. VR helped to create links between data separated by orders of magnitude from one another in geological research studies. By including the geological site and inserting data pertaining to particular locations all the way from the macroscopic to microscopic, VR linked the information in a convenient visualization [20]. Medical procedures have also benefited from the use of VR visualizations of tissues and organs. Surgeons typically utilize flat, gray scale slice planes of a patient and have to mentally construct a 3-D model of the internal structure. VR has been utilized to aid in the planning stage of surgeries to provide a fully immersive and 3-D representation of the internal structure of a patient [21]. Finally, VR has been utilized as a psychological tool to help individuals deal with their personal fears. In one instance users with public speaking issues were put in front of a virtual audience to slowly build up confidence for speaking in the real world as well as learn more about the phobia [22].

There have been few broad and encompassing studies on the effectiveness of VR and those that have been performed are inconclusive [23]. However, a point of agreement arises when virtual reality is being utilized as a searching tool, and has been found to benefit the ability of the operator in building a reference frame of the environment quickly. This means that a user can quickly assess their surroundings and minimize redundant searching [24]. VR can also provide a positive learning example when moving from Immersive 3-D displays back to 2D monitors.

Virtual reality has become very important in diverse professions and disciplines. Even with disadvantages, the ability to immerse a user into a simulation, whether it is for training or visualization purposes has marketable benefits. Creating the sense of depth is the job of the stereoscopic devices utilized by the operator. To create the actual scene geometry a 3-D rendering language must be available.

2.7 The 3-D Rendering Software

Virtual reality utilizing a stereoscopic view is simply a three dimensional scene viewed from two distinct points that are spaced apart by about the breadth of a human's eyes. The actual rendering of the scene does not require any special virtual reality specific software allowing the use of standard three dimensional graphical interface languages and techniques to create the rendered images. By creating a camera which can alternate between two positions and render out two distinct images the requirements of the hardware are met. There are two dominant methods for communicating with graphics hardware to produce three dimensional scenes. These methods are the open standard of OpenGL and the platform dependent Microsoft Direct 3-D.

2.8 OpenGL and Direct 3-D

OpenGL is an API or Application Programmers Interface which contains a library of over one-hundred function calls that communicate to the graphics hardware to create graphics primitives, attributes, geometric transformations and other 3-D scene functions [25]. OpenGL supports all major platforms and can be utilized across a wide range of hardware. It was designed with the intent of being viable for all uses from computer aided drafting (CAD) to modeling to games [26]. This means that basic input and output functions such as keyboard and mouse responses as well as setting up a graphical window are not included in the standard OpenGL library. There are auxiliary libraries such as GLU and GLUT which provide additional functionality for performing these tasks.

OpenGL is a state machine, meaning that it operates in much the same way as a series of switches. Should a particular setting be turned on or off or set to a particular value, all future functions will utilize that setting unless it is changed. An example would be when creating points; all points would be created as the same color unless otherwise specified before each new vertex. After a state is set, everything executed afterwards will follow that set state unless it is changed.

OpenGL utilizes a series of callback functions that are set up by the user to initialize particular scene parameters. Examples would be the functions glDisplayFunc() and glReshapeFunc(). These functions control what is displayed in the scene and what happens during a window resizing, respectively. These functions are passed user-specified functions to control what occurs whenever the scene must be rendered or when the window requires resizing.

A second commonly used API is Direct3-D, which is a part of the Direct X API. It will only run on a Windows operating system, however widespread use of this platform has caused it to thrive. Direct3-D follows OpenGL in many ways, but there is one core difference between the two systems. Direct3-D was designed to be a direct interface to hardware, meaning whatever the hardware provides Direct3-D would provide as well. OpenGL is a rendering platform which may have hardware acceleration but is not dependent upon it, which follows the open standard that OpenGL utilizes. Over the past two decades, these two platforms have become the standard for the creation of 3-D applications.

Regardless of the API utilized to create the 3-D scene, there are several requirements when making a 3-D rendering platform. Geometry is required through the creation of vertices, edges and faces. A viewing frustum is required to act as a camera into the world and dictate the area to be rendered. Lighting is needed in order for faces in the scene to be visible. While there are several other requirements for a 3-D rendering package, these first few are of primary important for virtual reality visualization.

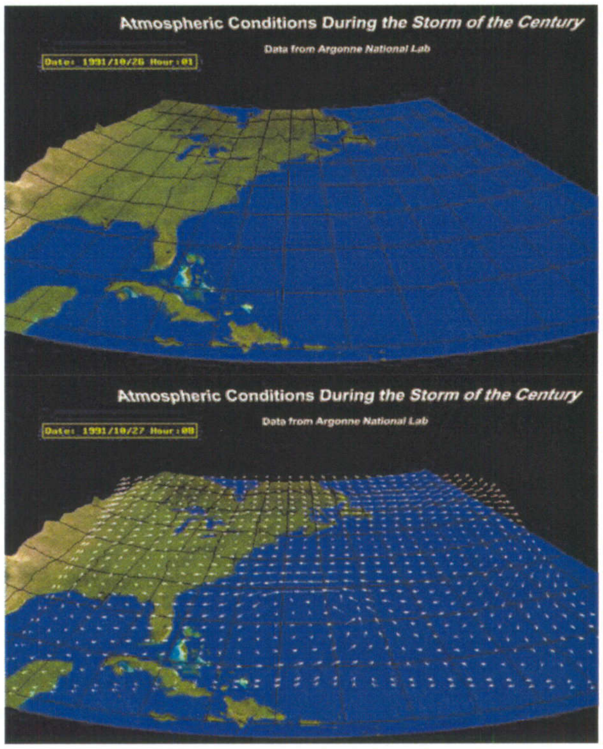
2.9 Virtual Reality Platforms

There are several available programs that assist in the creation and management of a virtual environment. Two are utilized extensively at Rowan University and are the prominent Virtual Reality platforms, VRCO vGeo and WorldViz Vizard. Each program has its advantages and disadvantages in accessing and displaying data.

2.9.1 VGeo™

As an older VR platform created by VRCO, the Virtual Global Explorer and Observatory (vGeo™) was designed to fill a role for oceanographers who were displeased with current tools available to explore, merge, and display time-dependent, three-dimensional, multivariate data sets [27]. Its primary strength is its ability to render raw data into graphical objects that can be static, moving, slices, or surfaces.

VGeo™ utilizes the Virtual Reality Modeling Language (VRML) when importing graphical models into the world that are not expressed by raw data. Figure 8 is a standard vGeo™ demonstration entitled Storm World. In these images, a VRML model of a shoreline is superimposed with wind vector data in the form of arrow plots, the humidity of the shoreline with a color map and finally a 3-D iso-surface generated from the humidity data.


Atmospheric Conditions During the Storm of the Century
Data from Argonne National Lab
Date: 1991/10/26 Hour: 01
Atmospheric Conditions During the Storm of the Century
Data from Argonne National Lab
Date: 1991/10/27 Hour: 00

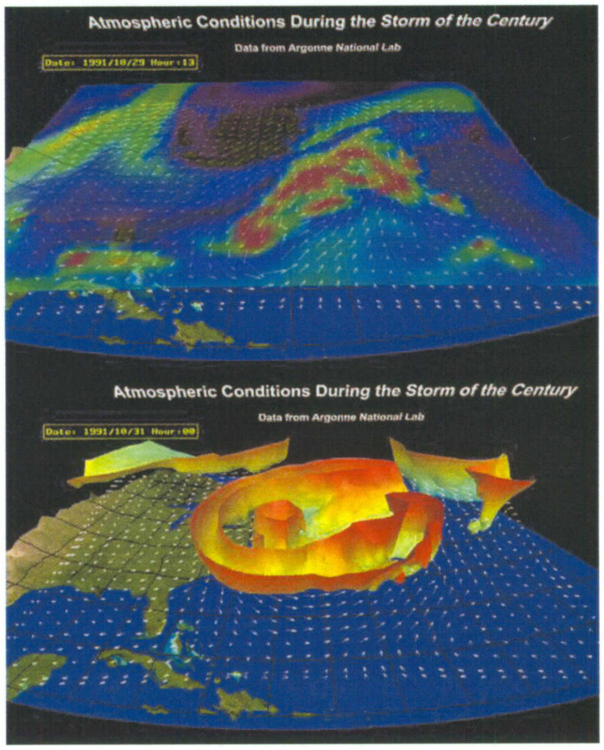
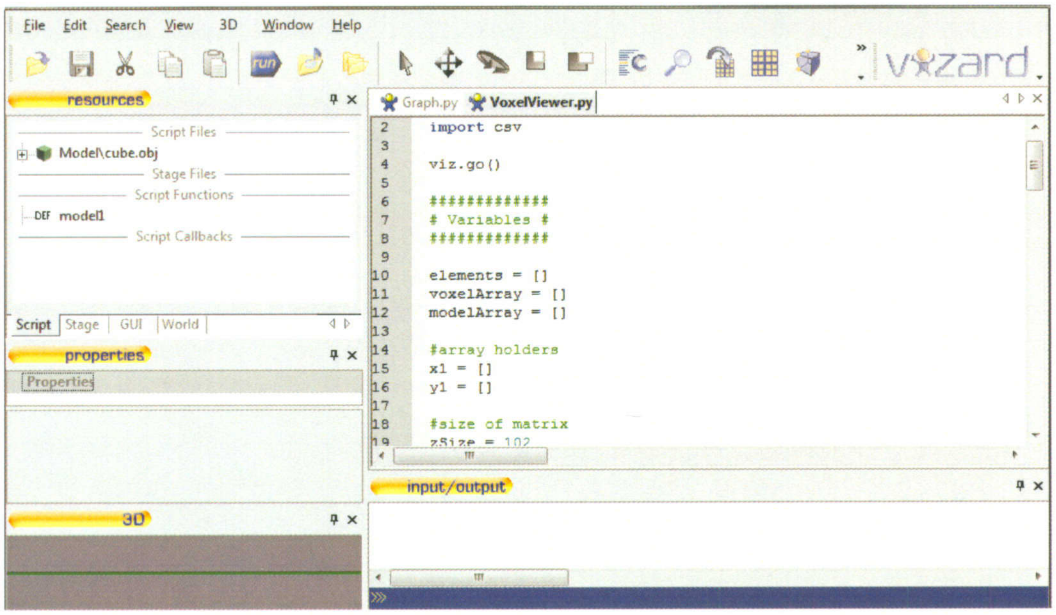

Atmospheric Conditions During the Storm of the Century
Data from Argonne National Lab
Date: 1991/10/29 Hour:13
Atmospheric Conditions During the Storm of the Century
Data from Argonne National Lab
Date: 1991/10/31 Hour:00

Figure 8: A vGeo™ simulation utilizing a 3-D model of a shore line and overlapped data of wind vectors and the humidity.

VGeo™ was designed for data visualization, and incorporates all the tools necessary to create different visuals, whether the data is scalar, volume or vector information. What vGeo™ does not excel at is in the ability to create the virtual world and add custom functionality. Many vGeo™ models have to converted and require several steps to correctly format the data outside of the creation of the model. In addition, many vGeo™ scenes are limited in the depth of functionality available to the coder.

2.9.2 Vizard™

The second virtual reality toolkit discussed here is the WorldViz Vizard™ platform. Vizard™ does not have any data visualization algorithms built in like vGeo™, however it offers the user direct interface with VR hardware, the types of accessible 3-D content and the ability to create non-native functionality. By utilizing the Python scripting language, the programmer can access low level OpenGL functions without having to directly apply OpenGL programming through a low level language such as C++. The Vizard™ library offers built in functionality for importing many kinds of models, arrays, networking protocols and more. Figure 9 is the Vizard™ API with a python script being created in the center and scene assets such as models and textures displayed on the left. Vizard™ comes complete with a detailed development environment that utilizes context sensitive cues to allow the program to anticipate the function calls the programmer intents to access.



```
import csv

viz.go()

#############
# Variables #
#############

elements = []
voxelArray = []
modelArray = []

#array holders
x1 = []
y1 = []

#size of matrix
zSize = 102
```

Figure 9: The WorldViz Vizard™ Interface.

The largest disadvantage to Vizard™ is that there are no preprogrammed data visualizations. To make up for the lack of visualizations, Vizard™ provides you with all the programming tools to be able to create vGeo™ visualizations or completely novel visualizations.

2.10 Virtual Reality Data Types

Regardless of the platform used to create the virtual reality environment, all the data describing the world can be categorized in three ways: graphical, measurement and functional data [1].

Graphical objects are usually thought of as representations of real-world objects such as buildings, furniture or landmasses. Figure 10 is an example of a graphical data set made from thousands of data points which describe the buildings geometry.

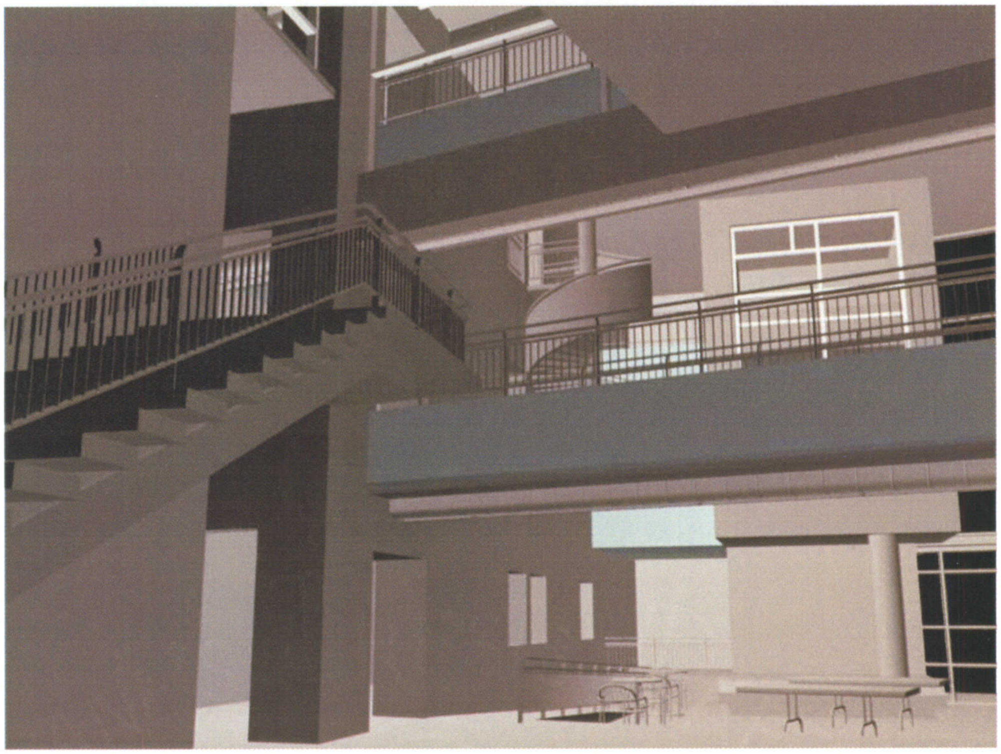
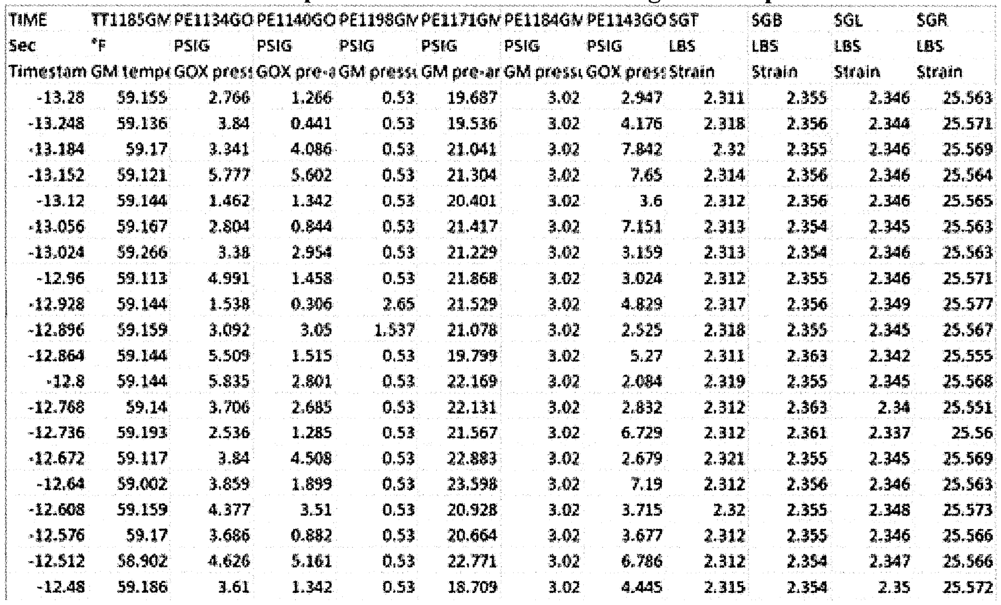


Figure 10: A graphical dataset to create a virtual Rowan Hall.

Graphical models are usually obtained through the task of modeling. Through the use of reference images and dimensioned drawings, models can be created in 3-D modeling software packages such as Autodesk Maya or 3-D Studio Max. In the past, Virtual Reality Modeling Language (VRML) was a popular format to contain the geometry of graphical objects. In recent years the addition of several other formats such as the Wavefront .obj, open scene graph .OSG and 3-D studio max .3-Ds file formats have become far more prevalent when storing model information.

Measurement data, the second data type is composed of sensor readings. Examples would include temperature, pressure, strain and the state of a valve. Measurement data is the raw information taken directly from a sensor with minimal processing. Table 1 is an example of measurement data inside of a spreadsheet. Each row progresses through time while each column describes a different sensor type, being either temperature, pressure of strain in this example.

Table 1: An example of measurement data catalogued in a spreadsheet.



TIME	TT1185GM	PE1134GO	PE1140GO	PE1198GM	PE1171GM	PE1184GM	PE1143GO	SGT	SGB	SGL	SGR
Sec	°F	PSIG	PSIG	PSIG	PSIG	PSIG	PSIG	LBS	LBS	LBS	LBS
Timestam	GM tempe	GOX press	GOX pre-a	GM pressu	GM pre-ar	GM pressu	GOX press	Strain	Strain	Strain	Strain
-13.28	59.155	2.766	1.266	0.53	19.687	3.02	2.947	2.311	2.355	2.346	25.563
-13.248	59.136	3.84	0.441	0.53	19.536	3.02	4.176	2.318	2.356	2.344	25.571
-13.184	59.17	3.341	4.086	0.53	21.041	3.02	7.842	2.32	2.355	2.346	25.569
-13.152	59.121	5.777	5.602	0.53	21.304	3.02	7.65	2.314	2.356	2.346	25.564
-13.12	59.144	1.462	1.342	0.53	20.401	3.02	3.6	2.312	2.356	2.346	25.565
-13.056	59.167	2.804	0.844	0.53	21.417	3.02	7.151	2.313	2.354	2.345	25.563
-13.024	59.266	3.38	2.954	0.53	21.229	3.02	3.159	2.313	2.354	2.346	25.563
-12.96	59.113	4.991	1.458	0.53	21.868	3.02	3.024	2.312	2.355	2.346	25.571
-12.928	59.144	1.538	0.306	2.65	21.529	3.02	4.829	2.317	2.356	2.349	25.577
-12.896	59.159	3.092	3.05	1.537	21.078	3.02	2.525	2.318	2.355	2.345	25.567
-12.864	59.144	5.509	1.515	0.53	19.799	3.02	5.27	2.311	2.363	2.342	25.555
-12.8	59.144	5.835	2.801	0.53	22.169	3.02	2.084	2.319	2.355	2.345	25.568
-12.768	59.14	3.706	2.685	0.53	22.131	3.02	2.832	2.312	2.363	2.34	25.551
-12.736	59.193	2.536	1.285	0.53	21.567	3.02	6.729	2.312	2.361	2.337	25.56
-12.672	59.117	3.84	4.508	0.53	22.883	3.02	2.679	2.321	2.355	2.345	25.569
-12.64	59.002	3.859	1.899	0.53	23.598	3.02	7.19	2.312	2.356	2.346	25.563
-12.608	59.159	4.377	3.51	0.53	20.928	3.02	3.715	2.32	2.355	2.348	25.573
-12.576	59.17	3.686	0.882	0.53	20.664	3.02	3.677	2.312	2.355	2.346	25.566
-12.512	58.902	4.626	5.161	0.53	22.771	3.02	6.786	2.312	2.354	2.347	25.566
-12.48	59.186	3.61	1.342	0.53	18.709	3.02	4.445	2.315	2.354	2.35	25.572

Functional data is any information obtained from any analysis [1]. Functional information is typically derived from mathematical models and computer generated simulations. Functional data can also include the results of analysis on data. An example would be with gas pipeline inspection where measurement data was recorded from sensor readings of gas line pipes. This information was transformed through the use of an artificial neural network into a functional data set. This functional data defined the predicted geometry of the pipe as either welds, cracks, or the general geometry of the pipeline. In the case of simulated data, the underlying information is not always required, only an advisement on what to do. As in the case of the health of a system, simulated faults or user messages can be simply displayed to indicate warnings or areas of interest.

These three data types are not entirely distinct from one another and crossovers between the formats are routine. One case would be in obtaining 3-D density information from a CT-scanner. This initial information would be regard as measurement data. Through several transformations and by setting a threshold for the values in the data, the density information can be made into a 3-D graphical model. This model would be composed of the threshold levels of density values from the gridded array of measurement information. Measurement data has been transformed to make a functional data which is then represented as a 3-D model of graphical data.

2.11 Creating Graphical Assets

Regardless of the type of scene being constructed and the data involved, there will always be the need to create graphical assets. Modeling is the process for creating a 3-D representation of an object in a computer. To model an object, the modeler must specify the characteristics of the object through the shape, spatial layout, and connectivity of the basic components [28].

2.11.1 Polygonal Modeling

A polygon is mathematically defined as a plane specified by a set of three or more coordinate positions, called vertices. These vertices are connected in sequence by straight-line segments, called edges and make up the sides of a polygon [25]. Polygons are normally defined in computer graphics as either triangles or quadrangles. Polygons with more than four sides can be used but are usually converted to the triangles. This is due to a situation where some of the vertices that make up the polygon are not co-planar, meaning they do not all lie on a plane. Figure 11 shows the difference between a planar and non-planar polygon.

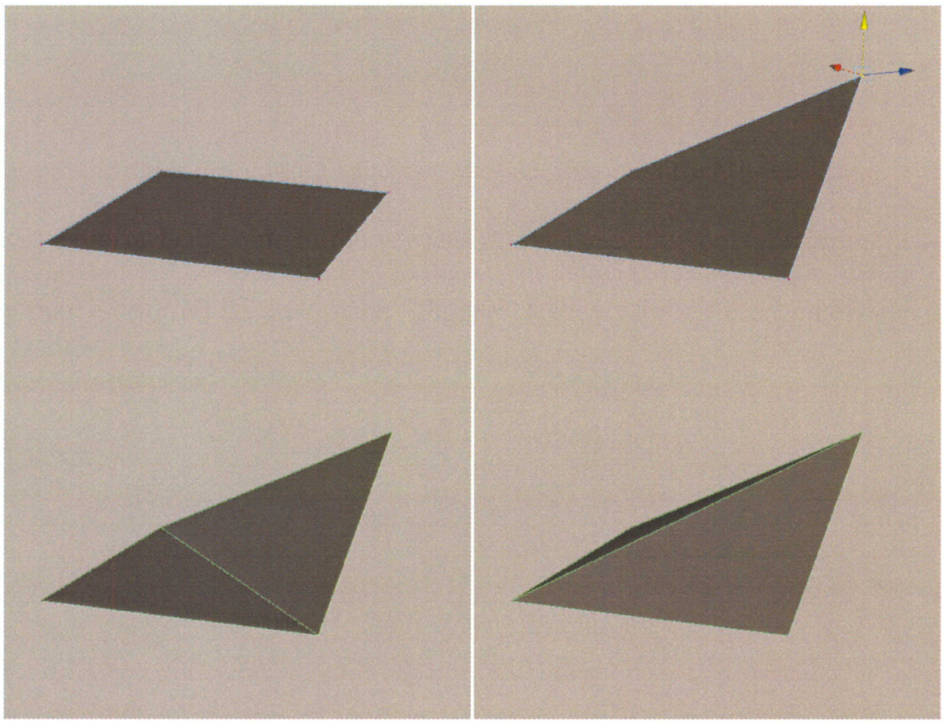


Figure 11: An example of a quadrangle that is planar, non-planar, and the required triangulation to resolve crease and lighting issues.

When a non-planar polygon exists, the rendering software responsible for calculating the surface will not be able to determine how to properly crease and light the surface. This results in a condition known as surface ambiguity. For the simple case in Figure 11, the problem can simply be resolved by breaking the object into two planar portions. However, there is an option of how to divide the object, across the top or down the center. The problem is magnifies as the number of vertices increase as in Figure 12. This is why most surfaces are modeled with triangles or quadrangles. With triangles there is no possibility of the object being non-planar, since three points will always define a plane. Quadrangles are sometimes utilized in polygonal modeling for special cases, and are especially important in subdivisional surface modeling, which will be elaborated on later.

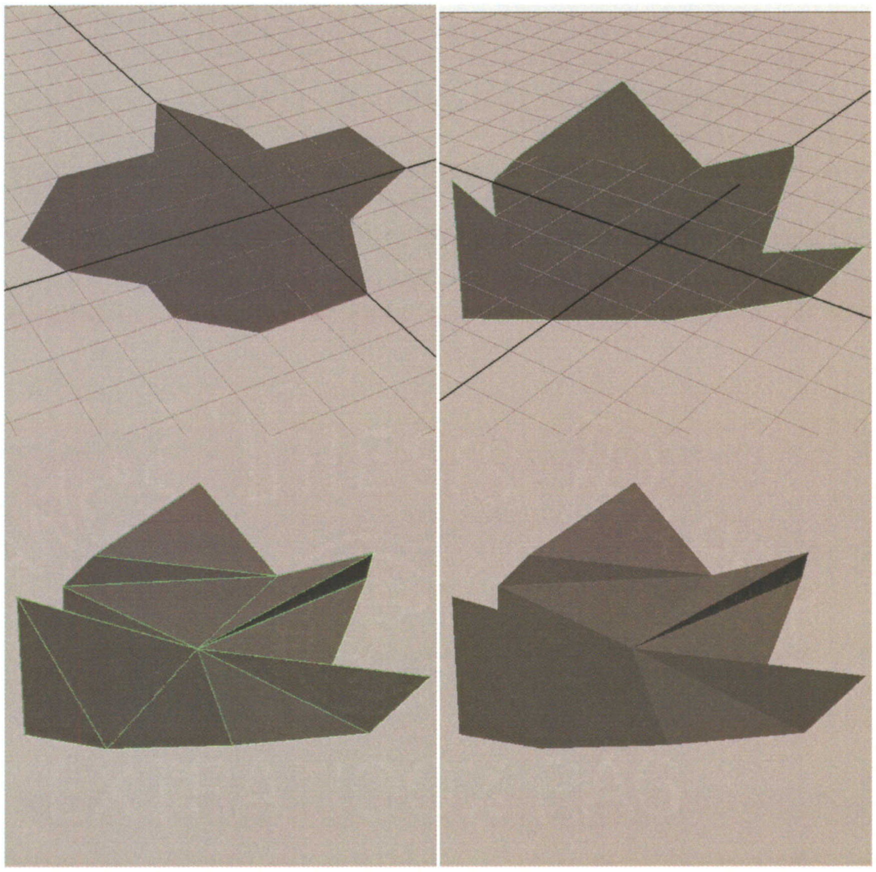


Figure 12: A 14 sided polygon with planar vertices, non-planar vertices and the resultant polygon after subdividing into triangular components.

Polygonal modeling works by stringing triangles together to create more complex objects. Basic primitives such as spheres, cylinders and boxes can easily be created with triangles. From these shapes more advanced objects can be created through the use of polygonal tools which interact with the base primitive object to add, delete and extend the number of vertices, edges and polygons. Figure 13 shows two triangles that are then utilized to create a cube. This cube is then modified to create the more complex geometry of a building.

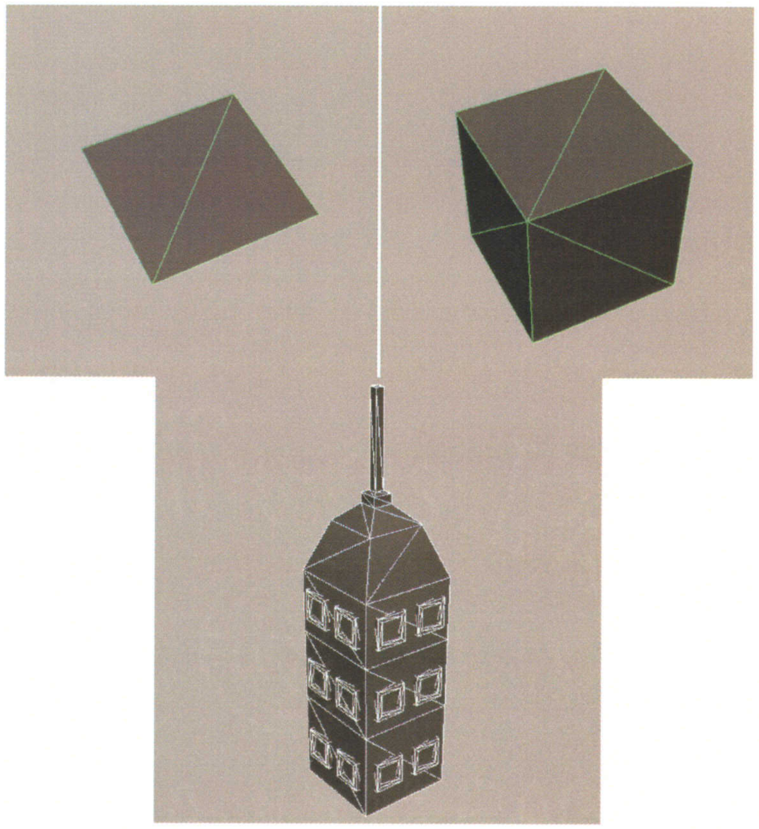


Figure 13: An example of how triangles can be used to create complex structures.

2.11.2NURBS

NURBS stands for Nonuniform rational B-splines. NURBS are mathematically expressed surfaces made from control points that manipulate the shape of the resultant surface. These control points are sometimes referred to as knots. Nonuniform means that these knots can be situated across an unequal spacing, allowing for further freedom when creating a surface. In total, a NURBS surface is composed of a series of control points, a set degree polynomial, weight factors, and a knot vector. With these components different B-splines can be blended together to create a three-dimensional surface. Figure 14 shows two B-spines and the resultant interpolated surface of the NURBS object. While splines are supported in many 3-D packages, many rendering tools such as Autodesk Maya and 3-D Studio Max are unable to directly render the NURBS surface. This limitation means that NURBS objects must be converted into a polygonal approximation before rendering.

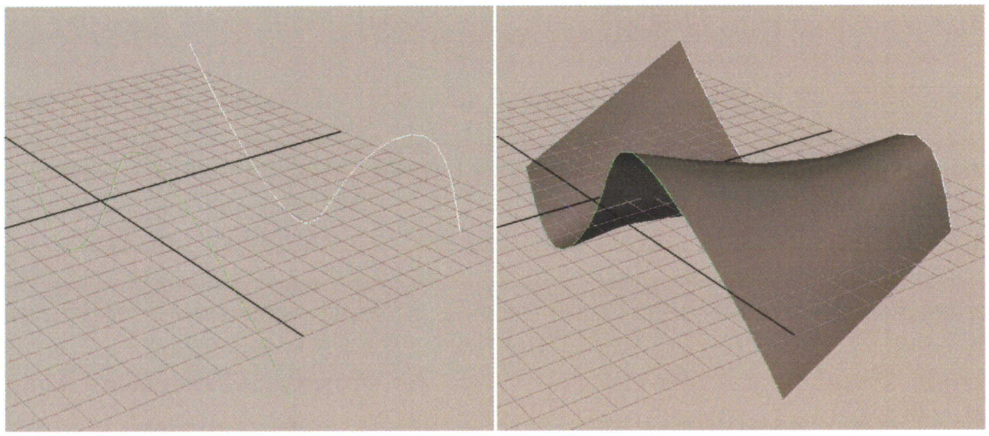


Figure 14: B-splines and the resultant NURBS surface.

NURBS surfaces provide the user with a mathematically expressed surface that can be subdivided indefinitely to make a curve of infinite resolution. Polygons are discreet units and are only able to approximate a curve through an increased number of polygons. Figure 15 is a NURBS cylinder compared with a polygonal model. While the NURBS surface will retain the curve at any scale, the polygon quickly appears nickled upon close inspection of the surface. This ability of NURBS surfaces to create smooth and infinitely detailed surfaces has made it popular when creating organic surfaces.

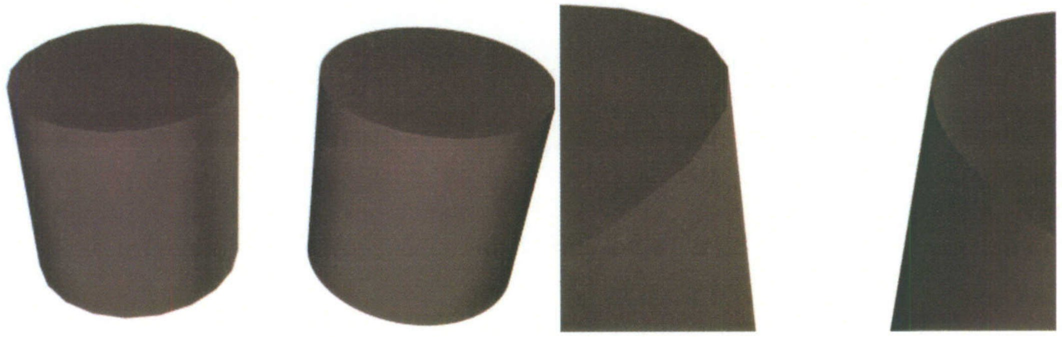


Figure 15: A comparison of a polygonal cylinder on the left and a NURBS cylinder on the right.

There is a primary drawback to using NURBS surfaces [28]. NURBS require a rectangular grid of control points and therefore can only represent a limited pool of surface topologies. To overcome these issues, subdivisional surfaces were created that combine the best of Polygonal and NURBS modeling.

2.11.3 Subdivisional Surface

Subdivisional surfaces provide an advantage over both polygons and NURBS in that they allow for random surface slices like Polygons, while maintaining the smooth curvature of NURBS [29]. Subdivisional surfaces are created by utilizing a base mesh of polygons. This mesh is then divided into regions of finer and finer detail. By modifying the base mesh, the successively smoothed divisions refine the geometry to provide a smooth and complex object. Figure 16 is an example of a base polygonal mesh using subdivisional surfaces to create a far more refined and smooth object than polygons alone could create [30].

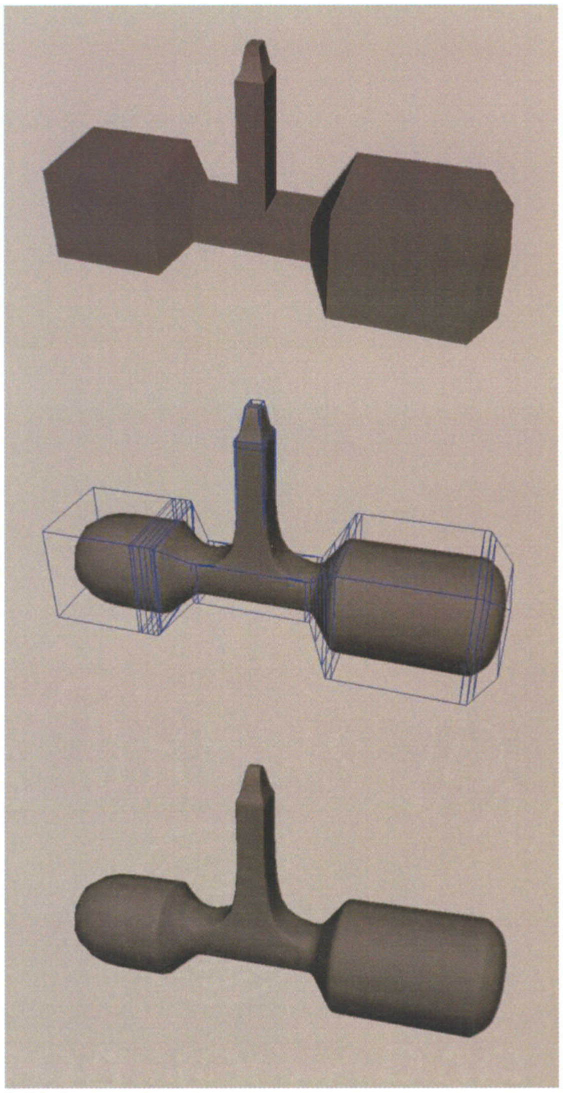


Figure 16: A polygonal object utilizing subdivisional surfaces to create a smoother and more organic shape.

Subdivisional surfaces remove some problems associated with NURBS modeling. During NURBS modeling problems can occur when trying to merge particular curves together at seams. NURBS curves require four sides at all times, while subdivisional surfaces are free to have any number of sides that a polygon can have, although they prefer four-sided geometry. In addition, since subdivisional surfaces are based on a simple polygonal mesh, the modeler is able to retain a simple polygonal workflow and add smoothing later when detail is needed.

These three methods reflect the most popular and commonly utilized methods of modeling graphical data for use inside of a 3-D environment. The model itself is just a 3-D approximation of an object. Additional parameters must be added to each model in order to create an accurate representation through materials and textures.

2.11.4Materials

When modeling an object such as a two-by-four composed of wood, the general shape can be roughly described as a rectangle. However, this does not describe the texture of wood. To include this attribute a material must be applied that controls how the face reacts to light. Depending on the desired effect, various types of materials can be applied to describe the appearance when light strikes the surface. Plastics, woods and metals all react differently with light, which requires various types of materials. Figure 17 shows the same sphere with different applied materials. Note the difference in the specular highlight between the three which makes each material distinct.

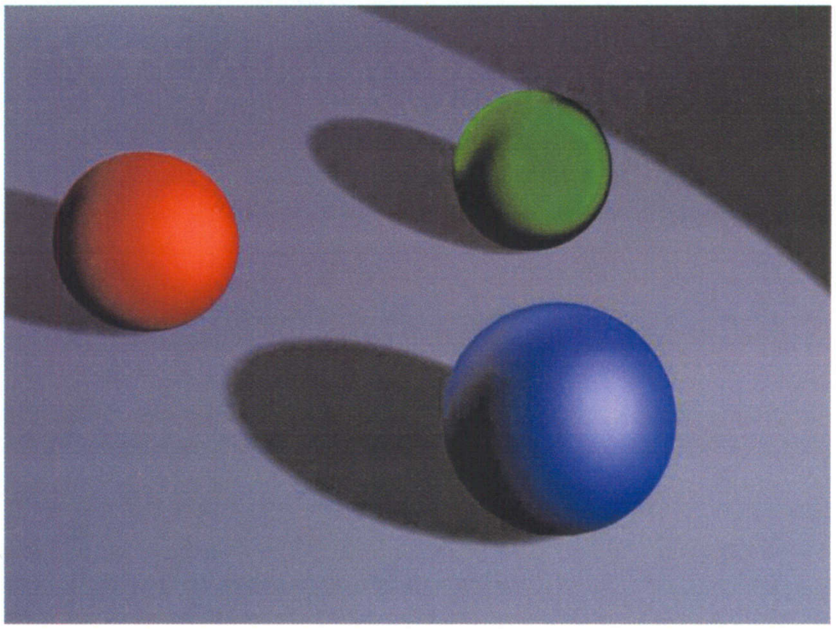


Figure 17: When interacting with light, materials provide a method for conveying the makeup of an object.

Wood that has not been sanded and polished is not shiny, so the material on the red sphere would be a good candidate to apply to the model. A material will adequately describe how light interacts with the surface but does not capture the patterns evident in wood. To visualize that information an image must be applied to the surface.

2.11.5 Textures

When image data is applied to a geometric model, the result is called a texture and the process referred to as mapping [31]. Textures help to describe small details that would be very difficult to convey through geometry. Wood grain would be difficult to model and require billions of polygons to capture the microscopic features that describe the surface. An image of wood placed over the faces of an object would allow for a vastly less complex model and still realistically represent a piece of wood. Figure 18 shows a simple wood image and the result when applied to a 3-D model.

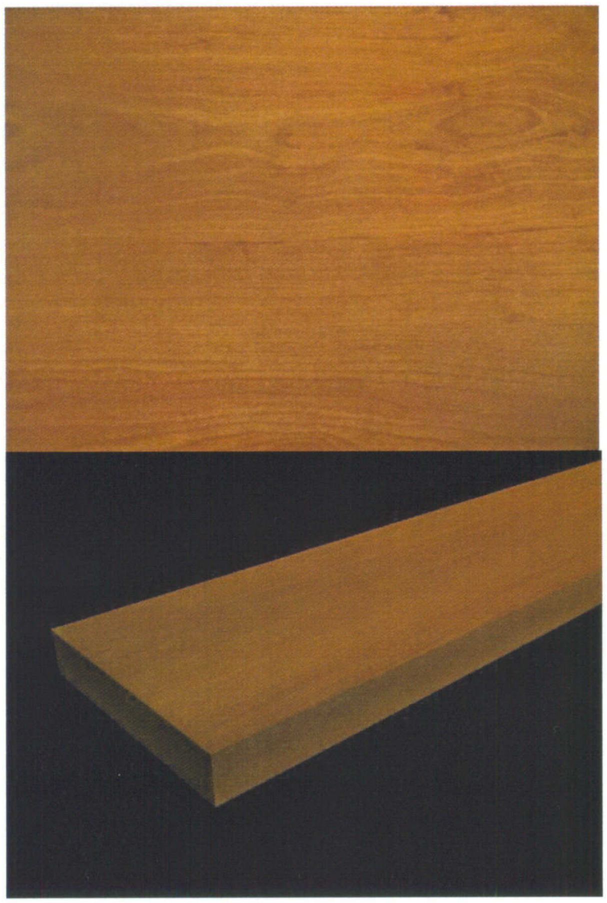


Figure 18: Texturing is the process of sampling an image over the faces of an object to provide visual information without additional geometry.

Texturing works by assigning the vertices of a face to a two-dimensional coordinate space. While the geometry exists in a three-dimensional space, each face is still composed of a series of planes that are two-dimensional. Mapping the two-dimensional texture to these two-dimensional faces allows for a texture to be wrapped over the geometry of a model.

The texture coordinate space is called the UVW coordinate space. U, V and W are not acronyms and are simply placeholder variables like X and Y. U and V are not the same as X and Y and do not have to have a vertical /horizontal correspondence with the model. Most modeling and texturing applications provide a normalized 0 to 1 texture square where the faces of an object can be positioned. By moving the UV vertices of the object around this space, different portions of the texture can be assigned to various faces.

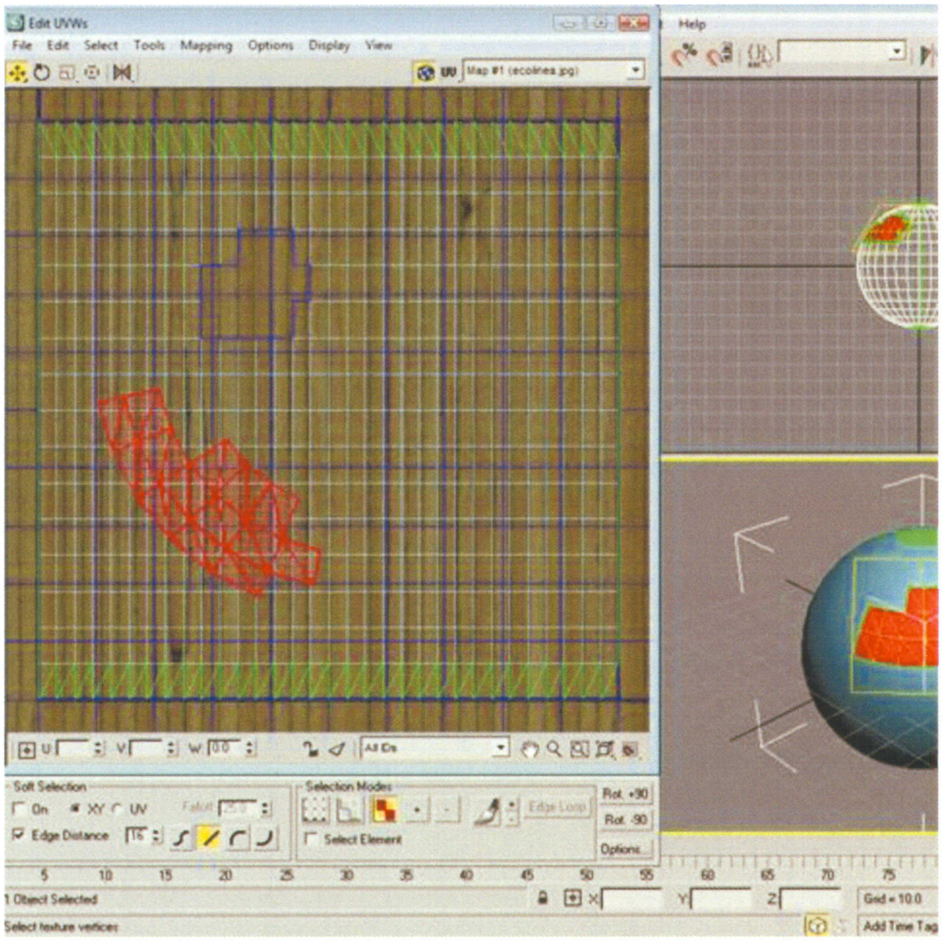


Figure 19: A texture layout utility that allows for UV coordinate manipulation over a normalized 0 to 1 texture space.

Figure 19 is the UV layout window inside of Autodesk 3-D Studio Max. Different faces of a 3-D object they can be translated, rotated or scaled in the UV layout editor to display a portion of a particular texture. In this case, several faces from a sphere are being separated and moved to a specific area of a wood texture. This is the typical flow to generate custom graphical assets from modeling to materials and finally textures.

2.12 Creating Graphical Assets from Data

When data needs to be represented as a graphical model usually an automated process is utilized. The datasets that need to be transformed are usually composed of spatial information. The simplest 3-D data set would be composed of scalar information for each element. The following methods can be utilized when dealing with a scalar volume data.

2.12.1 Scalar Volume Data - Contour/Color map/Height-field Plots

A contour plot is a useful way of visualizing data when looking at cross-sectional planes of 3-D data along a particular axis. Contour plots are the result of identifying a threshold value and creating a boundary between two areas on a two dimensional plane [32]. This is done with a 2D plane composed of polygons with each element corresponding to a vertex. Figure 20 is composed of a series of polygons laid out on a grid where each of the vertices corresponds to an element in the array.

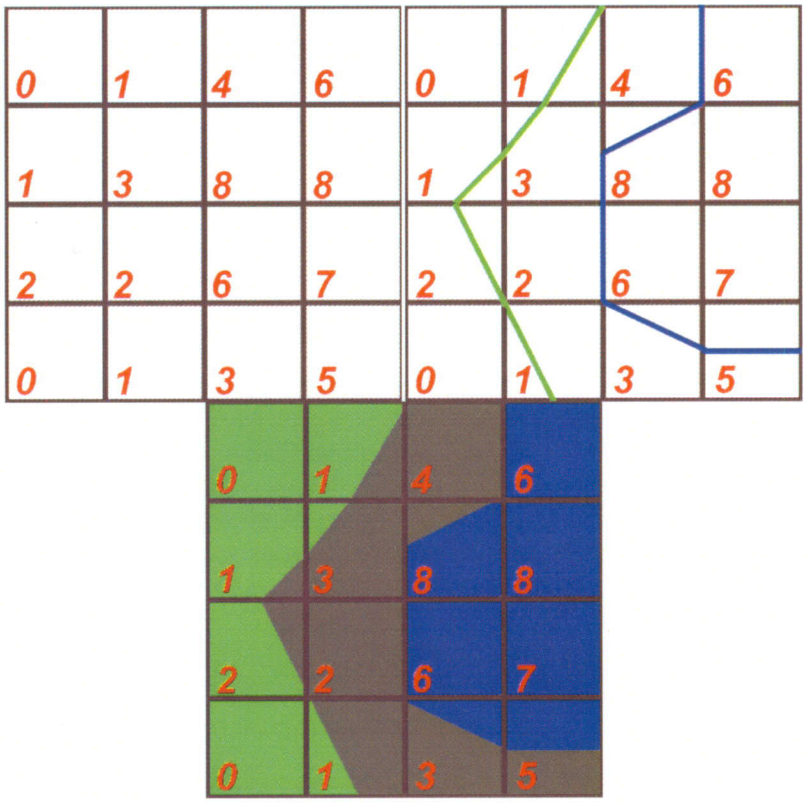


Figure 20: A dataset with a threshold of x<=2 and x>=6 with the resultant contour and color map.

An algorithm would check each point against its neighbors and determine if a division is necessary. It is possible to linearly interpolate between two points and create a division in the polygon at a midpoint. These contours would be assigned different colors to indicate the divisions. The right image in Figure 20 is a contour plot with color indicating the areas with different values based upon the chosen threshold. With the inclusion of color to the surface it is regarded as a color map plot.

Rather than use color, it is possible to assign the actual value to a translation of the plane's vertices in the 3rd dimension. The data will have to be scaled in order to minimize large differences between values. Also, if this is a slice plane with additional planes above and below, a maximum height displacement will have to be assigned to keep the different slices from intersecting. Figure 21 is an example of a height-field plot. The color can be included on the height-field plot but could result in visualization difficulties when lighting and shadows are added to the surface. These operations will result in modifications to the colors and as a result the viewers perception of what is plotted could be changed.

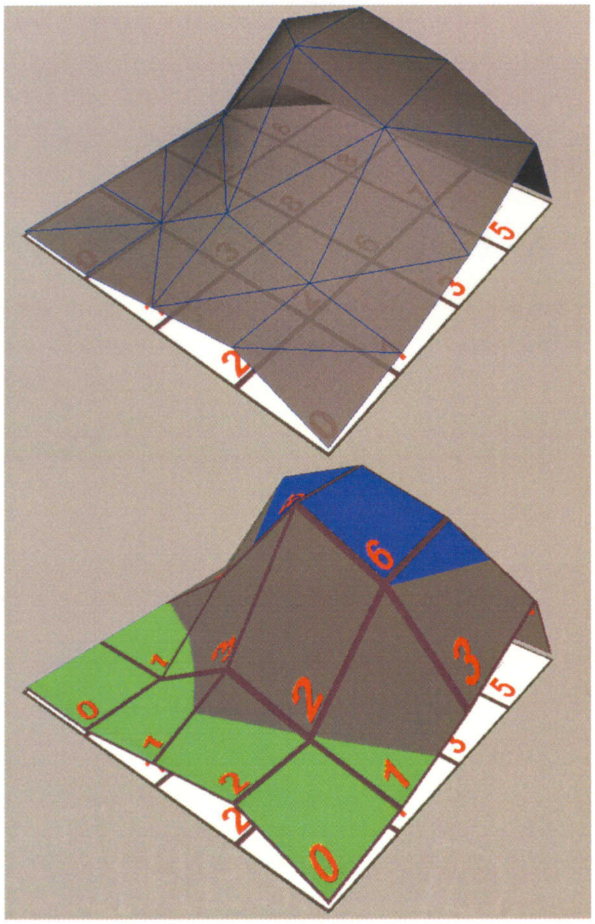


Figure 21: A height-field visualization with and without color.

Slice planes can be effective visualizations but are limited since you can only look at a single plane at a time. A user would have to observe a continuous 3-D volumetric array as a series of discontinuous planes. There are other visualizations that allow the user to view all the information as a large 3-D object.

2.12.2 Scalar Volume Data - Voxels

Voxels are the three-dimensional equivalents of two-dimensional pixels. They can be thought of as volume pixels, hence the name: voxels. Voxels are utilized to represent a three-dimensional space as a three-dimensional object construct of individual blocks. Figure 22 is a 3-D dataset represented as a series of voxels. Voxels are utilized in many 3-D applications, such as medical and industrial tomography to view volume information. Voxels themselves are not visualized due to the large number of elements and the enormous amount of processing power required. While pixel resolutions composed of thousands or even millions of rows and columns are commonplace, the increase in dimensionality with voxels causes storing and rendering difficulties. An image of size 1024x1024 would be composed of 1024^2 or 1,048,576 elements. However, a voxel of the same resolution would require 1024^3or 1,073,741,824 individual elements. Allowing each voxel to take the form of a cube, with six sides and a minimum of two polygons per face would result in a scene composed of 12,884,901,888 polygons. Besides the large number of polygons on the scene, there are inherent occlusion problems with voxels. If a voxel is made completely opaque, it would not be possible to see through the surface to the inner density as seen in the upper left corner of Figure 23. One option would be to increase the transparency (or lower the opacity) based on a user's preference. By also including color after the entire object has been rendered translucent, areas of varying value would be easily noticed as in Figure 23. It is possible that this could lead to misconceptions due to overlapping colors which would distort or hide the interior values. Slice planes are another option when viewing voxels to cull away surface data and see into the interior as shown in lower portion of Figure 23. When viewing voxel information, an iso-surface is usually created, which is a 3-D shell formed by setting a threshold value that excludes particular elements.

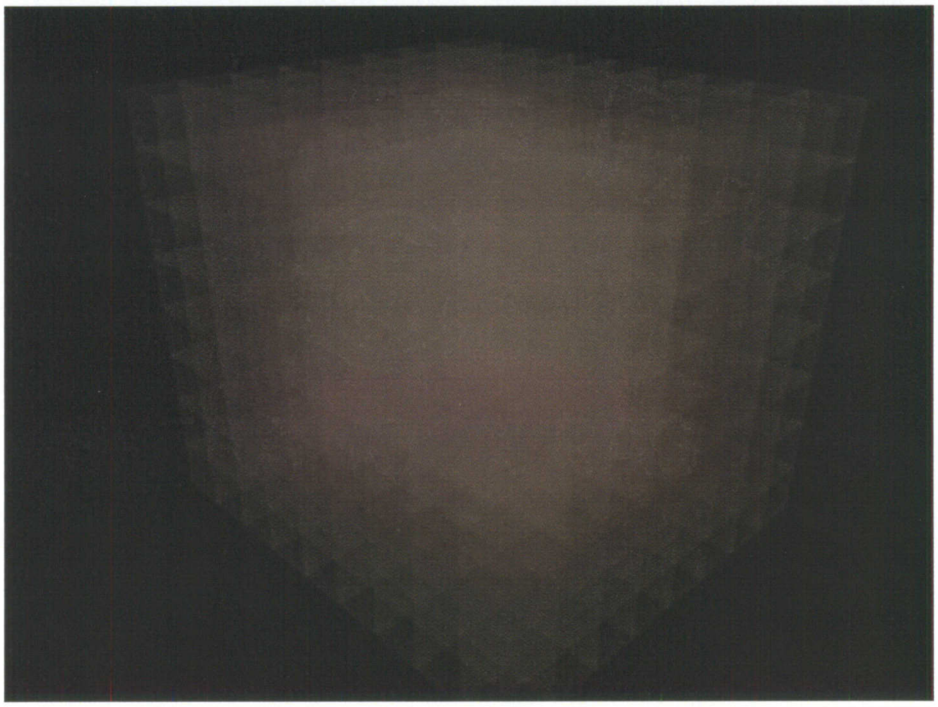


Figure 22: A field of voxels with translucency to view inner details.

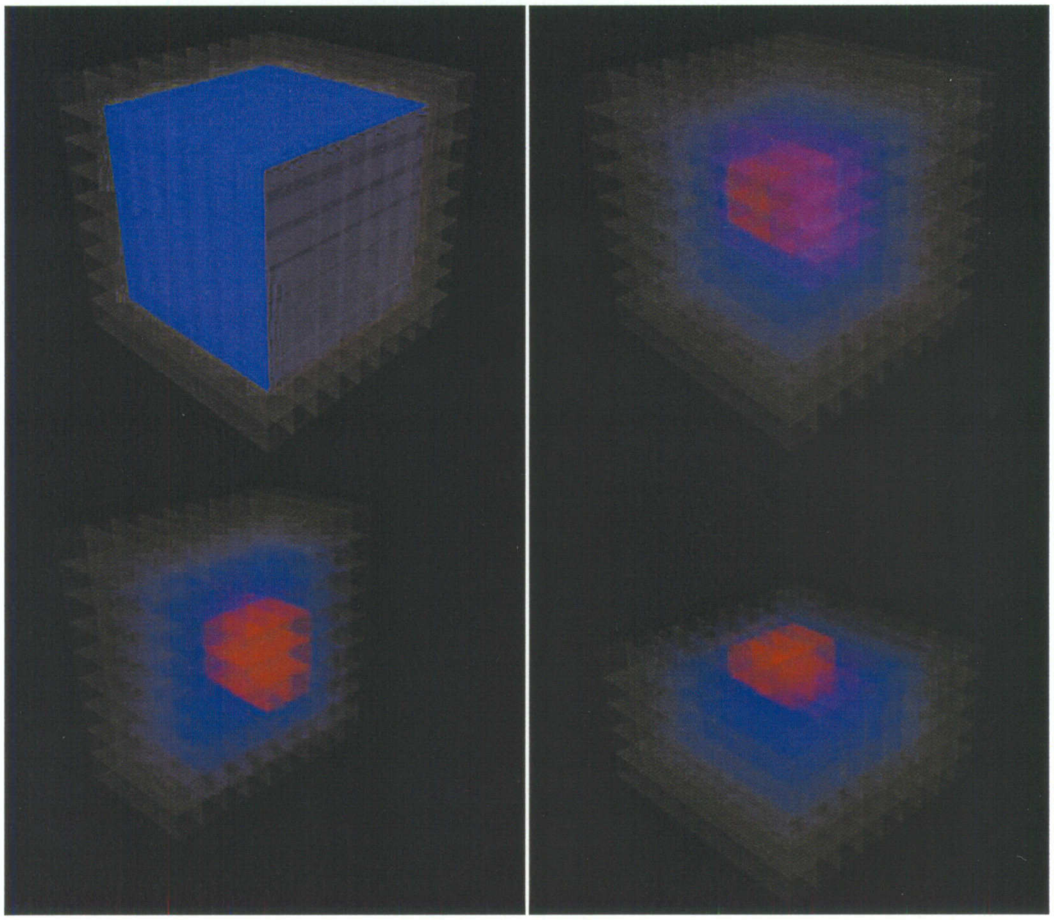


Figure 23: Voxels colored but opaque, colored and transparent, and utilizing slice planes.

2.12.3 Scalar Volume Data – Iso-surfaces

An iso-surface is developed from data when a user sets a threshold value, which acts as a mechanism to split the elements and create a polygonal shell similar to what was performed in creating contour maps. Continuing this division between data points will create a line boundary, which in three dimensions can be extended to a planar division. Figure 24 is an example of an array of values with a boundary being determined based on a threshold value.

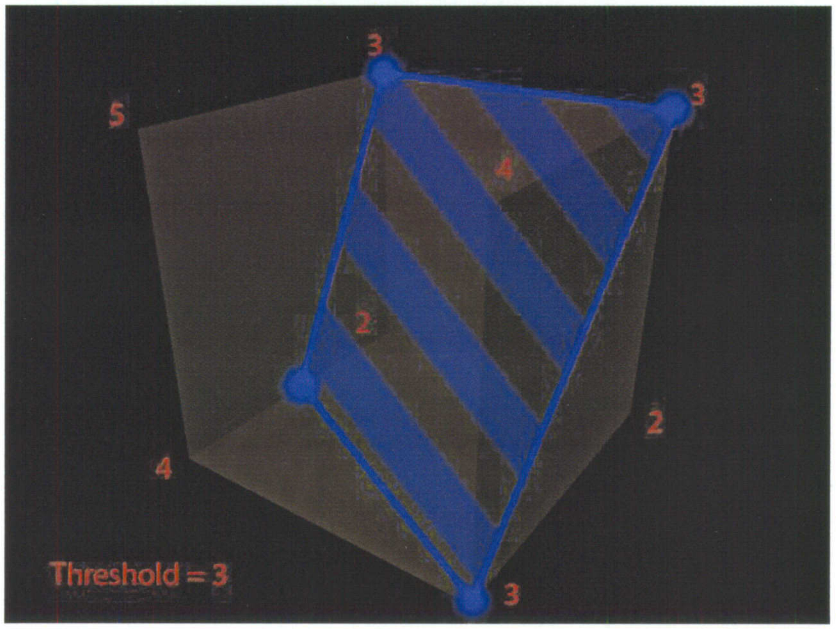


Figure 24: Using a threshold on an array to create a geometric boundary.

The plane itself does not have to be planar and can develop into a closed or open surface depending on the boundary conditions of the object. An open surface would result when the data does not fall into a higher and lower threshold when the algorithm is run. Figure 25 shows an open iso-surface and a closed surface. There are different algorithms for the creation of an iso-surface, with the most popular being the marching cubes algorithm.

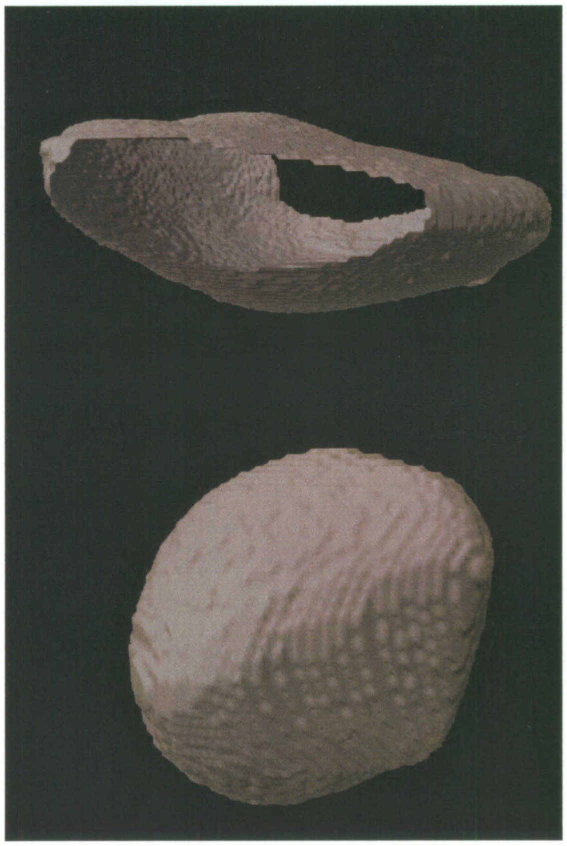


Figure 25: An open iso-surface and a closed iso-surface.

The marching cubes algorithm works by organizing the data array into a series of cubes. The values of each array in the matrix would be the intersection of a particular set of four lines (or one vertex), with each intersection being one of the eight points that creates the cube geometry. Just like before, a threshold value will be set and checked at each point in the cube to see if it is above or below the value. Since there are eight points in a cube, and two possible states (in or out) there would be 2^8 or 256 possible combinations for the boundary of a single cube. However, many of these combinations are symmetrical about an axis or rotation, which leaves 15 possible cases as shown in Figure 26.

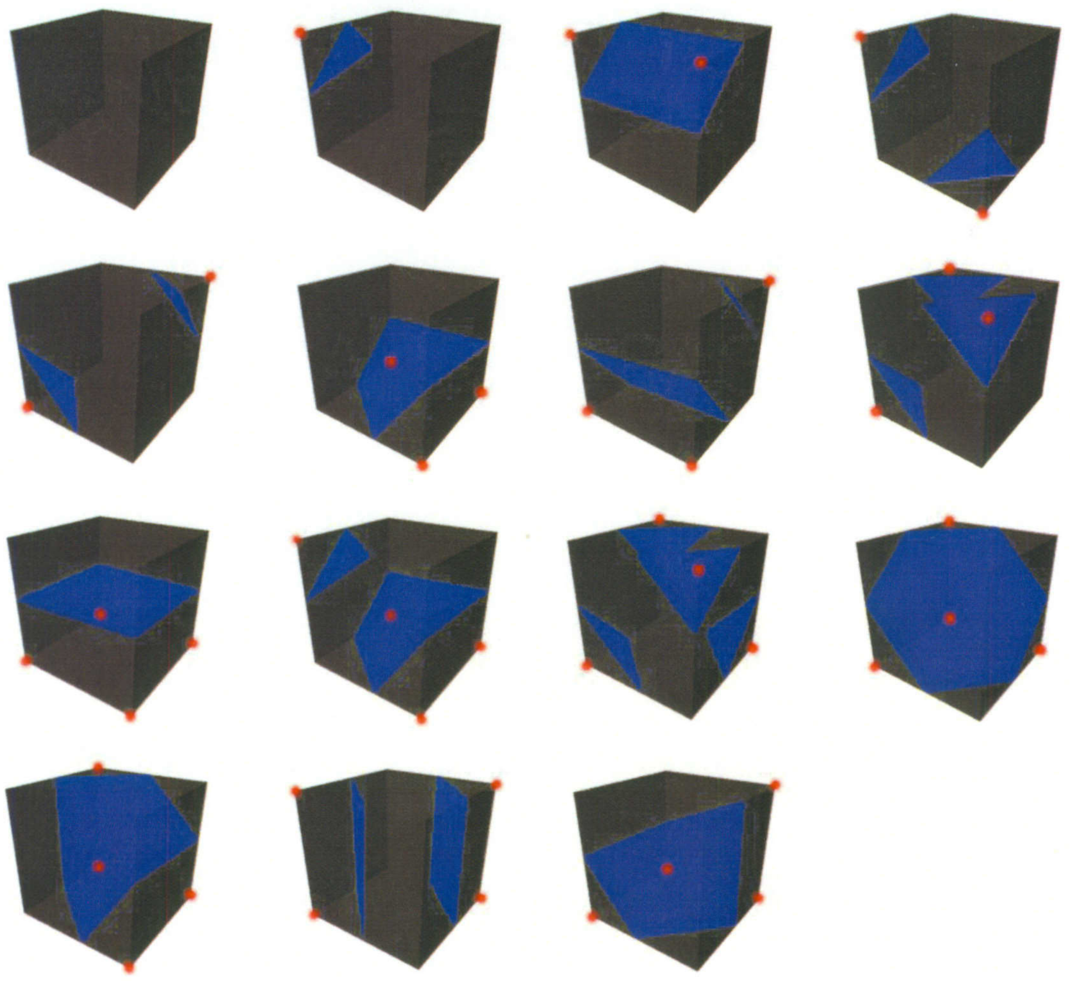


Figure 26: The 15 possible combinations in the marching cube algorithm.

A possible complication arises in this algorithm when the adjacent cubes and their configuration are not regarded. Open models can result in missing faces unless alternate boundaries are assigned to the cube. Figure 27 is a situation where the standard cube configurations would result in open areas in the final model. By using an alternate method of creating an intersection, which still fulfills the role of dividing the boundary it is possible to avoid this problem.

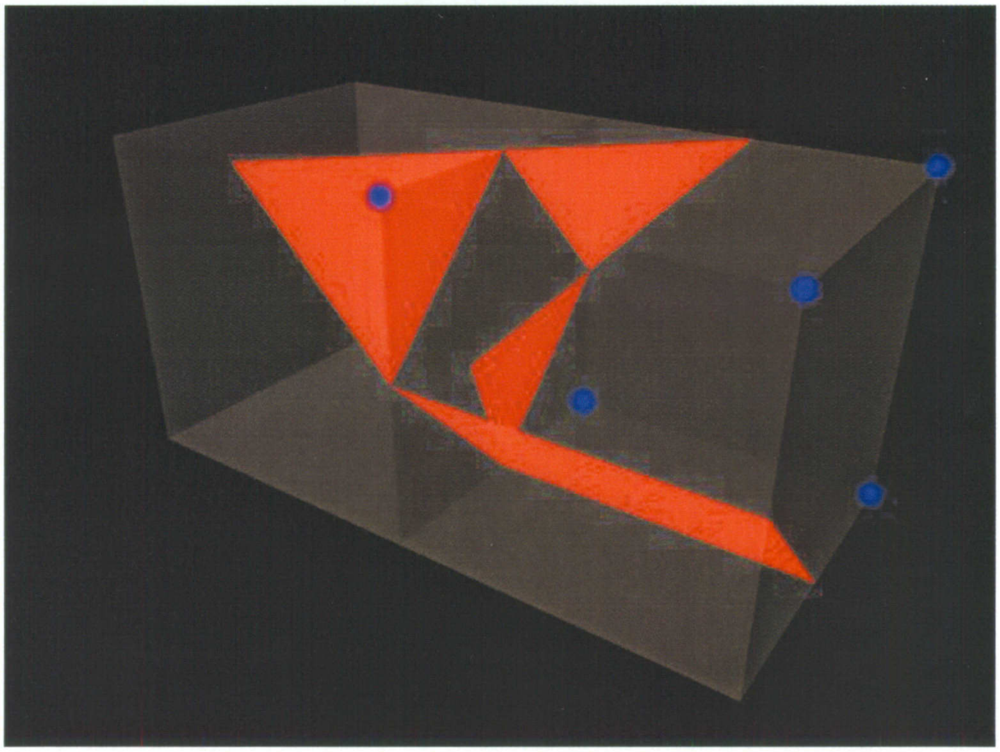


Figure 27: An example of when the cubes algorithm fails to close geometry, leading to an open surface.

While there are other methods for creating surfaces, this is by far the most popular method available.

All of the methods discussed so far only deal with scalar volume information. Different visualizations must be implemented when dealing with vector data.

2.12.4 Vector Volume Data

A vector is a quantity specified by a magnitude and a direction. Magnitude is nothing more than a value representing the length or scale of some quantity. The second component, direction, dictates the angular position of the quantity. An example of a vector dataset would be wind data, providing the direction it is traveling and the strength of the gust. Vector data is often utilized when analyzing the flow of fluids such as air, water or oil. The following are several methods used to visualize vector data.

2.12.5 Vector Volume Data - Arrow Plots

An arrow plot places an arrow at each element in a gridded array. The magnitude of the vector is represented by the length of the arrow while the head of the arrow represents direction. A similar plot called the 'hedgehog' removes the arrow head to simplify the visualization and uses a transparency gradient over the arrow to depict the direction of the line.

The magnitude quantities of the plot may require scaling due to arrow intersection problems. With a 3-D gridded dataset, occlusion problems and difficulty discerning individual arrows may occur with dealing with simple lines. 3-D models could be utilized as arrows, allowing for shading and lighting to break the confusion brought on by pixel arrows. Figure 28 is an example of models utilized to display direction and magnitude rather than 2d pixel representations.

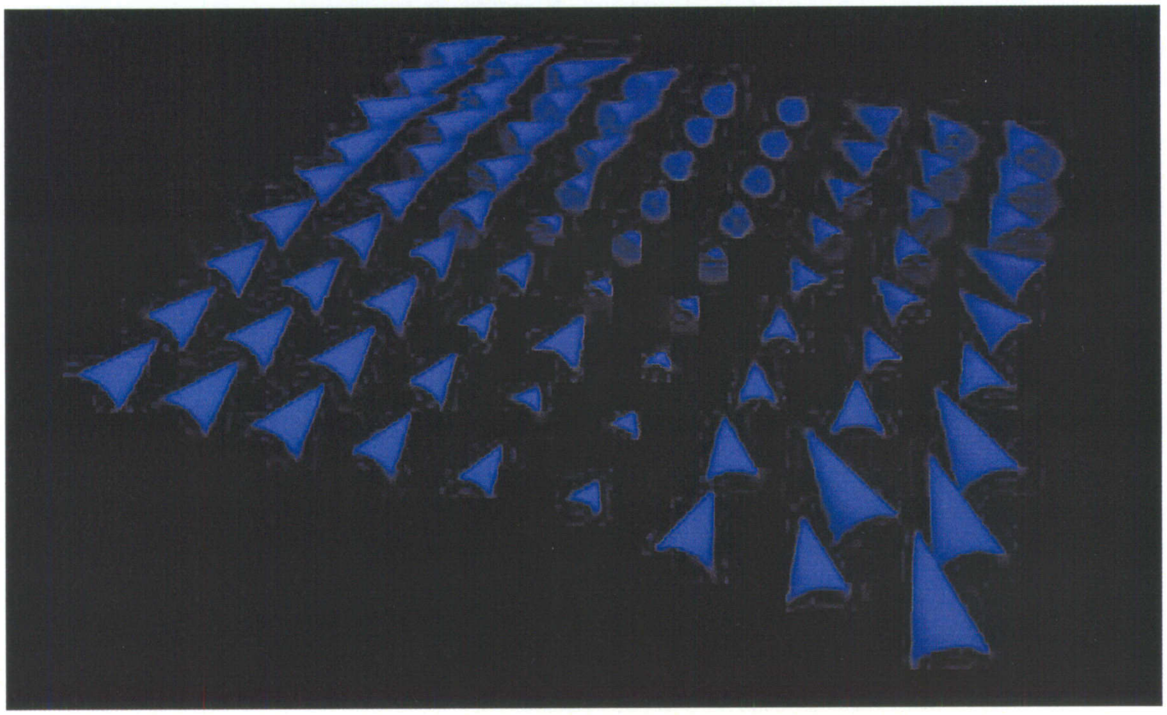


Figure 28: Arrow plot utilizing polygonal models to denote direction and magnitude.

2.13 Summarizing Graphical, Measurement and Functional Data

By combining graphical, measurement and functional data with the visualizations discussed here it is possible to create a powerful suite of tools. Figure 29 is an example of graphical, measurement and functional data all incorporated into a virtual environment. Here, a gas transmission pipe line is shown with all three datasets overlaid. The underlying 3-D model of the pipeline is a graphical data component created with modeling software. The measurement data was transformed into a graphical model and wrapped around the pipeline making the spiky iso-surface surrounding the pipeline. Finally, the functional information is shown in the center of the pipeline with the small colored cylinder which indicated how an artificial neural network predicted the geometry of the pipeline.

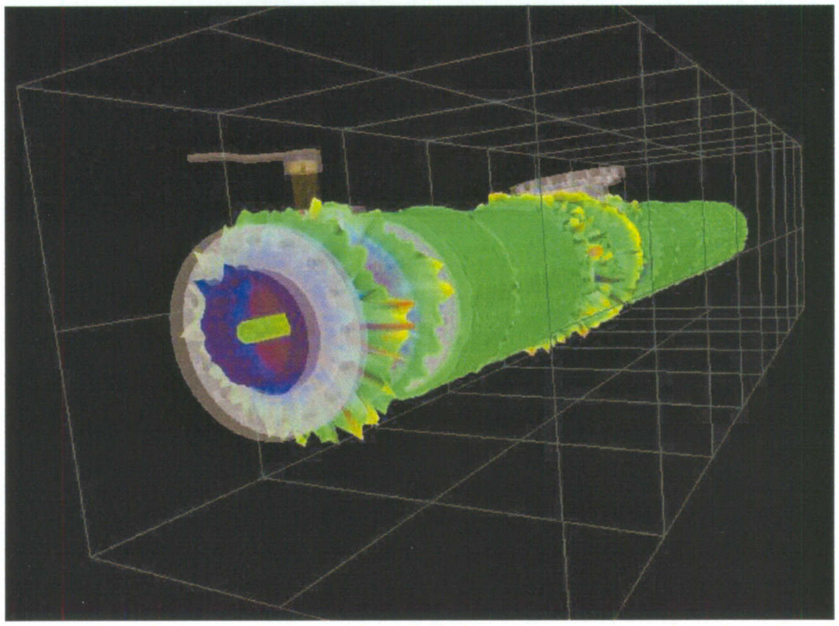


Figure 29: Graphical, measurement, and functional data overlaid in a 3-D world. [1]

2.14 Novel Visualization Applications

The visualizations already discussed are the 'textbook' cases and have been implemented in many situations. There are many needed visualizations that fall outside of the previously mentioned formats. Time varying datasets composed of millions of voxels are difficult to visualize in real time. Overlaying different datasets is another challenge. It is also not always important to create visualizations but utilize these techniques in novel ways. The medical community has implemented visualizations for viewing internal structures, planning surgeries, and training doctors how to suture a wound. Visualization systems are being used in virtual environments to track the navigation of avatars (the user's digital persona in a virtual environment) to optimize paths and note areas of high activity. There are many new avenues of visualization that are constantly being explored and here are a few of those examples.

Visualizing physical and chemical processes and the various changes undergone as time passes is integral in exploring and gaining an intuitive idea of how they work [33]. With recent developments it is now possible to capture simulations that result in hundreds of millions if not billions of voxels over thousands of time increments with hundreds of different variables. The first challenge when dealing with data of such a large magnitude is in viewing it. Encoding the data is one possibility through either separating the information into spatial and temporal domains or by utilizing complex tree structures to store the information in memory. Other performance boosts would include utilizing hardware-accelerated systems to keep up with the enormous volume of information. This type of rendering is limited by the amount of memory on the hardware; requiring very expensive graphics cards with gigabytes of memory space. Even with the best hardware, many simulations are just too enormous and require distributed rendering processes. This is performed by storing the simulation data on a central computer and having the different rendering jobs sent to client computers. The rendering is then returned and stitched together to create the final image. However, such setups are not adept at handling simulations with billions of components in real time.

Even when it is technically possible to view the data there is still difficulty in making a visualization that describes the information. In the area of fluid mechanics, visualization difficulties occur when attempting to look at 3-D data as it changes over time. As with any 3-D display, objects occlude one another making it difficult to see things at a distance in a cluttered scene. The previously mentioned geometric visualizations of streamlines, flow tubes, path lines, and such can effectively visualize local flow features, but lack a continuous global view. A new method utilizes a sparse representation by injecting seed particles and removing them in areas of clutter [34]. Seeding refers to the act of selecting a point where a streamline or other visualization will originate from. Adding additional seeds will allow for continual population of areas that lose seeds due to the natural flow of a fluid. The removal of excess seeds helps to rid cluttering in regions where all the flow paths meet.

Overlaying visualizations can run into many problems beyond simple occlusion. If two visualizations utilize similar colors and lines, they can be difficult to visually separate. Also, visual artifacts can appear when using certain kinds of line visualizations, creating optical illusions. There are several techniques available to contend with this problem. When utilizing the same visualization for two vector fields, by changing the color, line width and opacity or embossing one field it becomes easier to distinguish between the datasets [35]. Figure 30 shows a series of flow lines with various changes to the color, transparency and size to help distinguish between the datasets. Another method is to utilize different visualization techniques for each vector field; however the problem of the best visualization for the data comes into play. It is also interesting to note that at the intersection points of the two streamlines, visual artifacts can be noticed as the mind tries to create lines that link different sections. These artifacts can be trouble when a person is trying to discern patterns. They can be mistaken for actual properties of the data and not simply illusions created by the visualization method.

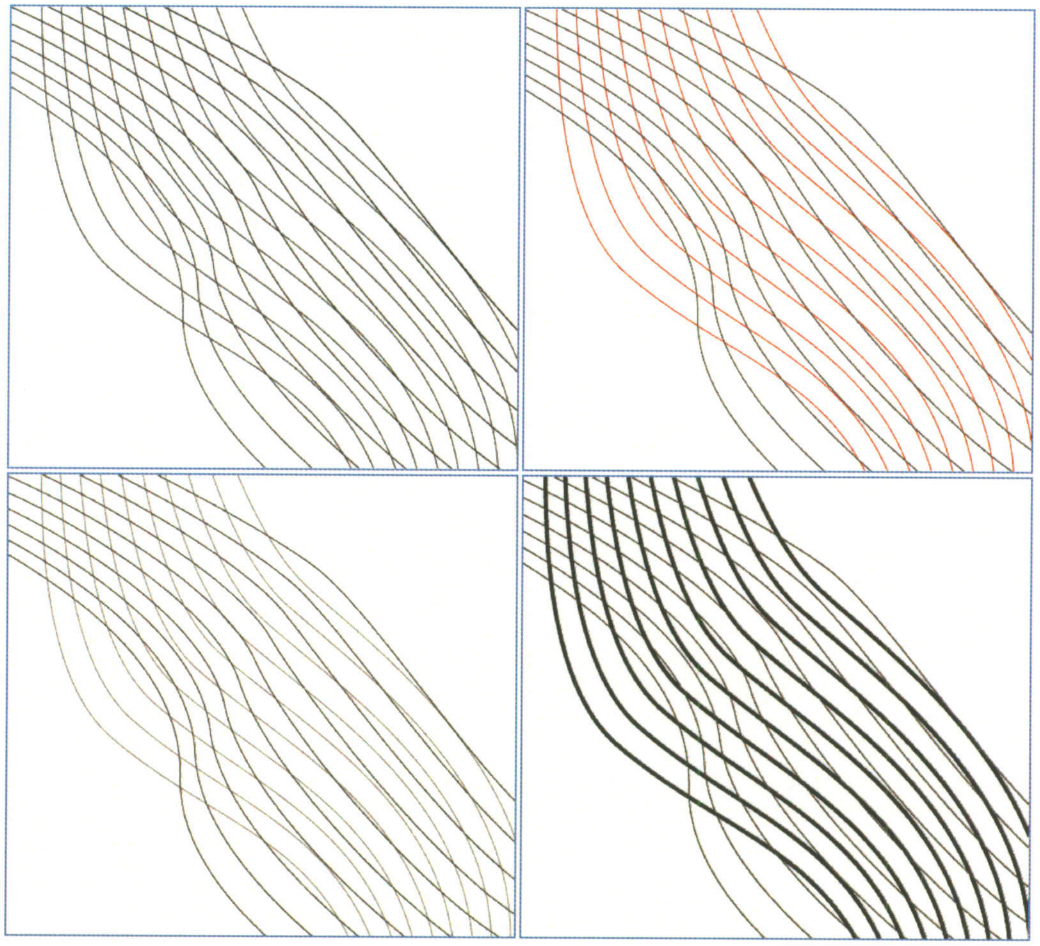


Figure 30: Overlaid streamlines of the same size and color can confuse the viewer. By changing the color, transparency and size of the streamlines a distinction can be made.

As mentioned, the medical community has greatly benefited from visualizations. Recent developments have helped budding doctors learn basic invasive procedures without practicing on humans. Real-time finite element methods have been utilized to create surgery simulations for suturing wounds on patients [36]. Coupled with a virtual reality environment, the operator performs suturing on a virtual representation of a hand that reacts to the user's gestures through deformation, surface contact, adjustable boundaries and multiple contact points. Another area of medical scientific visualization is creating a simulation of near-infrared light (NIR) passing through tissue. NIR is highly permeable in human tissue, making it a popular tool for noninvasive imaging. Utilizing 3-D volume visualizations of tissue and running simulations of the effect of NIR light passing through the different areas that would be difficult on real-world subjects is a new application of the technology [37]. This method allows for the capturing and studying of NIR effects that otherwise are lost in real-world testing due to diffraction. By simulating the expected results, we can create a perfect expectation for NIR imaging without the noise associated with collecting a real world dataset.

The use of visualization in transportation has moved away from typical applications of traffic studies for roadways. Virtual environments now are just as important and have their traffic data evaluated to notice areas of interest or disinterest. Design choices for a virtual environment can be made based on this information. Areas of low interest can be removed or remodeled to entice user traffic. Bottlenecks can be restructured to allow for easier navigation in a virtual world and cut down on performance issues when dealing with a large number of virtual avatars in a small location. The traffic data can be overlaid onto floor plans, utilizing flow lines or color to indicate travel paths and areas of interest [19]. Figure 31 is an example of navigation visualization. The top image shows the actual paths that different users may take through the room. The image below it uses color to indicate the most popular or congested areas of the virtual world.

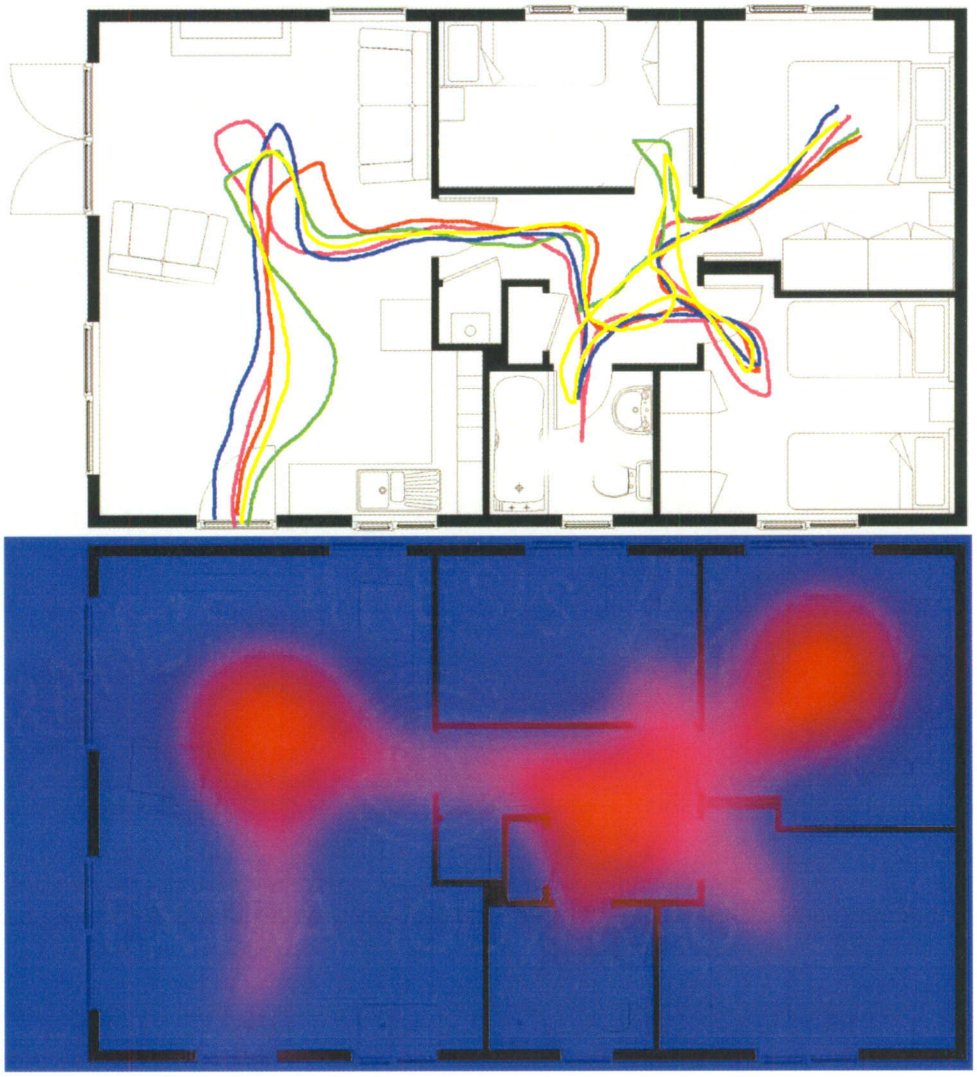


Figure 31: An example of an avatar path visualization and areas of congestion.

Similar visualization programs have been utilized to look at the advantages of monitor based navigation or real-space virtual reality walking areas. Through these visualizations it was shown that real walking situations were far move immersive than stationary and monitor navigation systems [38].

Visualizations are important tools to gain insight as well as discover patterns in information sets. Through the use of immersive virtual reality displays, a user can interactively navigate a data set as if it existed in reality. There are many options for creating and configuring a virtual reality setup, including head mounted devices and projector screens. And while VR does have a few disadvantages in certain applications, it can be extremely beneficial for many disciplines from medical to simulation.

In summation, a VR world utilizes a 3-D application to create the images and there are many available software packages that can quickly create an immersive world. The different data sets inside of these worlds can be broken down into graphical, measurement and function. Graphical data is normally created by a modeler in a 3-D modeling application such as 3-D Studio Max or Maya. Materials and textures can be applied to the model to create a more realistic object. Measurement data can utilize a series of visualization algorithms to create contours plots, voxels, iso-surfaces and more. Functional information is composed of analysis data and can indicate important events as well as system predictions. With all this data coupled together, it is possible to create a world which facilitates user interaction and discovery.

CHAPTER 3 : APPROACH

This thesis attempts to cover visualizations for virtual reality applications and approach this topic in three phases. The first section will describe typical visualizations for data including scalar and vector across one, two or three dimensions. The second portion will describe the three kinds of data typically encountered when creating a virtual world: graphical, measurement and functional data. Methods of displaying the three types of data inside of a virtual world will be developed. Finally, the creation of three-dimensional visualizations from 2D image data will be covered. Flow charts covering each project as well as complex areas will be provided in each section.

3.1 Data visualizations for Vizard™

In order to visualize data, algorithms must be developed to take in and transform information into a graphical form that the user can understand. Many programs offer built in functionality for creating visualizations such as Mathworks Matlab™ and Mechdyne's vGeo™. However, many other programs do not natively support the ability to quickly create visualizations. Attempting to network or bridge two programs together to create real time dynamic visualizations can lead to poor performance and bottlenecks. For that reason, it is important to develop algorithms that create simple data visualizations for the WorldViz Vizard™ platform, which is the currently preferred virtual reality rendering platform at Rowan University.

Data can come in a variety of forms and nearly endless accompaniments of visualizations are possible. For this thesis, the fundamental visualizations will be created for several data types including scalar and vector data in one, two and three dimensions.

3.1.1 Scalar Data Visualizations

Scalar information only contains a magnitude and retains no directionality. Examples of scalar data can be temperature, the force of wind, or the health/state of a system. When utilizing spatial data over an area, many data points are pooled together to show a trend or variation.

To visualize any surface regardless of dimensionality in a 3-D graphics package, there are rules for creating geometry that must be adhered to. Worldviz Vizard™, as well as other applications, utilizes OpenGL functions to create geometry, which have built in methods for handling object creation.

For scalar one-dimensional data, the information must be assembled into an array in memory. This array will be iterated over and the information mapped to a corresponding spatial location in the 3-D world. In OpenGL, a line can be created by specifying a series of points and utilizing the type GL_LINE_STRIP. Inside of Vizard™, these low-level hardware functions are wrapped within Python to allow for easier programming.

The algorithm requires iterating over each point in the array and creating a vertex. The distance between each array element can be specified, and mapped to the x location. Since this is a two dimensional line, the z dimension can be ignored or set to a predetermined value. The scalar value will be mapped to the height or y value of the line for each element. Figure 32 shows the corresponding mapping of array elements to line points.

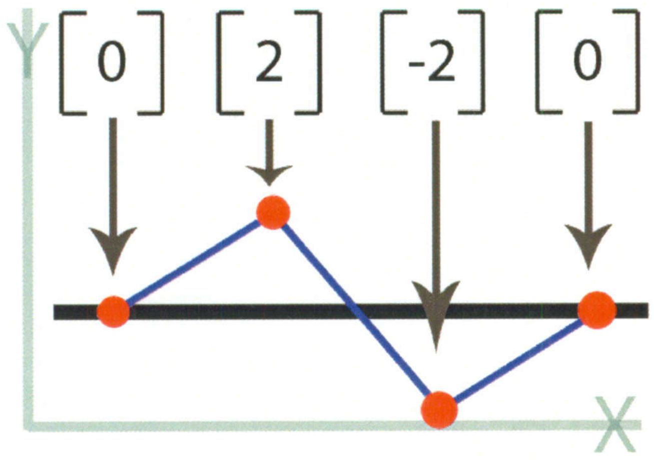


Figure 32: Mapping 1-D array information to a spatial location.

This information can be conveyed as a static plot or a dynamic plot. In a dynamic plot, the spatial elements are continually updated to coincide with a changing array. Typically, an array is given an additional dimension where the changing data is stored. Once the data has been mapped spatially, additional iterations are performed over the second dimension in the matrix. Figure 33 shows how the second dimension of a matrix can be utilized to store the next time increment of data.

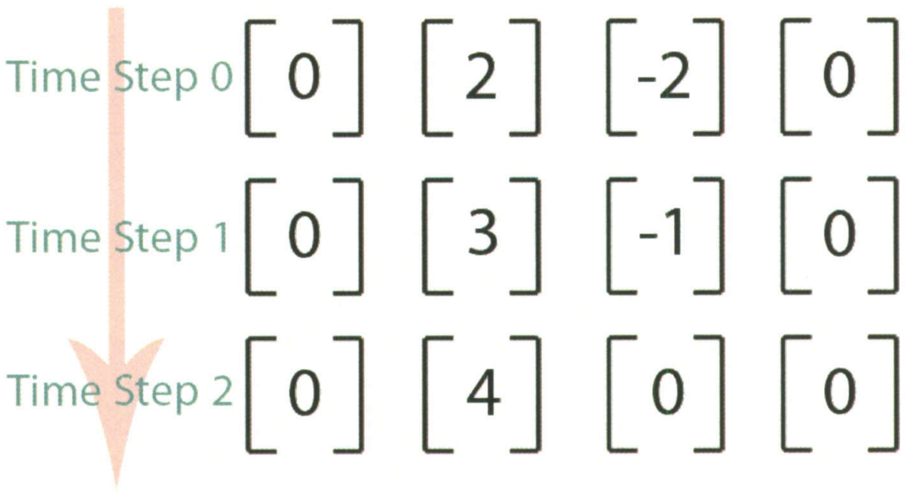

Time Step 0 [0] [2] [-2] [0]
Time Step 1 [0] [3] [-1] [0]
Time Step 2 [0] [4] [0] [0]

Figure 33: A one dimensional matrix holding future data (time) in the second Y dimension.

Besides mapping the scalar values to height, they can also be mapped through color. This requires setting up a color gradient and normalizing any magnitudes to a value between 0 and 1. These values can be directly plugged into the RGB color of a point in the graph. However, maximum and minimum values must be determined to create the normalized gradient values. In the case of blue and red, the intermediate values will be varying shades of purple as seen in Figure 34.

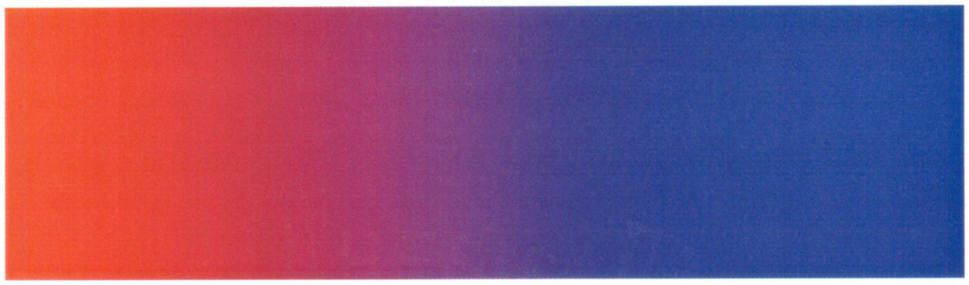


Figure 34: A possible color gradient between red and blue to show changes in magnitude.

Two dimensional data utilizes a similar approach when creating visualizations but the creation of a surface is more complex. As discussed in the background, a polygonal object is typically constructed out of triangles. These triangles have a 'winding' which dictates the direction it is facing. Triangles can be clock-wise or counter-clockwise. Triangle must be created in the same fashion in order for advanced features such as back-face culling to improve performance to work correctly. Back-face culling refers to the act of not rendering both sides of an object, reducing the number of objects that must be rendered and enhancing performance.

To construct the surface, a series of triangles must be produced utilizing the OpenGL TRIANGLE_STRIP type. This function specifies the vertices in a top to bottom order and removes redundant vertices by linking each triangle to the last in a chain.

Figure 35 shows how the vertices in a triangle strip are assembled as well as how many triangle strips must be utilized to create a plane. Each of these strips will have redundant/overlapping vertices on the bottom and top after the first as seen in Figure 36.

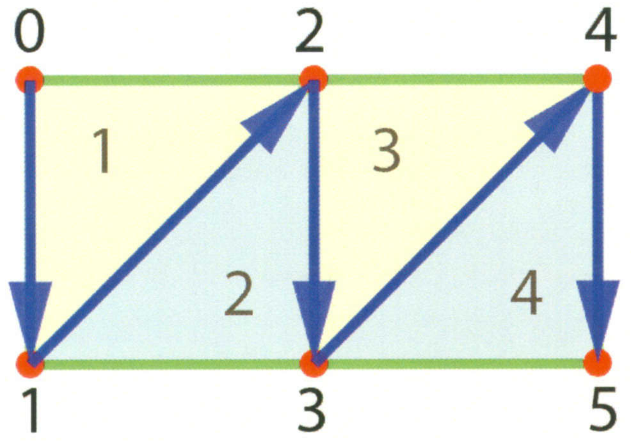


Figure 35: The creation of a triangle strip and the resultant triangles.

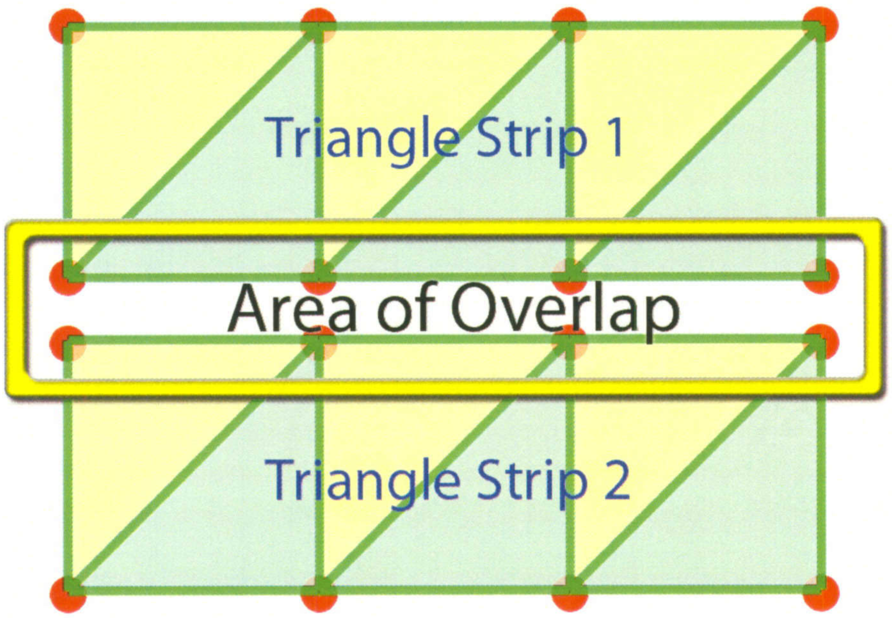


Figure 36: Overlapping vertices when utilizing triangle strips to create a planar surface.

With this redundant area, any algorithm that iterates over the data must make sure to properly duplicate the information of the bottom row of the first strip to the top row of the second. Another difficulty in developing this algorithm is that the vertex numbering is not in the same fashion as array numbering. With the triangle strips, each number progresses vertically and then horizontally, while an array format continues horizontally and then vertically. When creating an iteration sequence, this has to be addressed. Figure 37 pictorially shows the differences between the sequences.

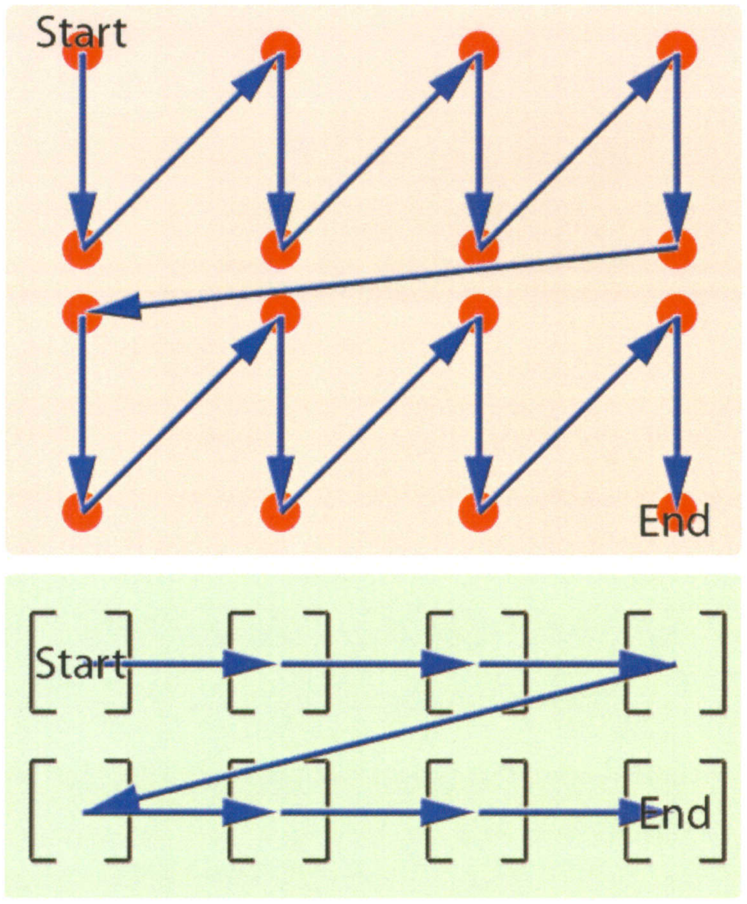


Figure 37: Iteration differences between triangle strips and arrays.

To overcome this problem the triangle strips have to be thought of as vertical pairs of data. Every even instance of a triangle vertex will be on top while every odd on the bottom when starting at a zero index system. If Vizard utilized a one index system, the top values would all be odd and the bottom even. Figure 38 outlines what is done to each of the pairs of vertices and how to assign the array data.

```
For every even vertex
  For the first triangle strip
    If we are not finished with the row
      -Set the current vertex's height to the array
      magnitude at the current row and column.
      -Set the next vertex (the one beneath it) to the array
      magnitude at the next row and current column.
      -Increment column count
    If we are finished with the row
      -Reset the column count
      -Increment the row count
      -Set the current vertex's height to the array
      magnitude at the current row and column.
      -Set the next vertex (the one beneath it) to the array
      magnitude at the next row and current column.
  For all other triangle strips
    If we are not finished with the row
      -Set the current vertex's height to the array
      magnitude at the previious row and current column.
      -Set the next vertex (the one beneath it) to the array
      magnitude at the current row and current column.
      -Increment column count
    If we are finished with the row
      -Reset the column count
      -Increment the row count
      -Set the current vertex's height to the array
      magnitude at the previous row and current column.
      -Set the next vertex (the one beneath it) to the array
      magnitude at the current row and current column.
```

Figure 38: Algorithm for assigning data from an array to a triangle strip.

Two common methods are available to tackle three dimensional data. The first utilizes voxels to create a 3-D visualization. Voxels, as covered in the background are volume pixels which tie a value to volume region. The algorithm for creating these visualizations requires iterating over every element in a three dimensional matrix and creating a cube for that location. The cubes can utilize a threshold value to remove instances outside of a range, so that inner details can be seen. Colors and transparencies can be applied to the cubes to overcome occlusions problems when viewing three dimensional data.

One problem with the above algorithm is that it will produce the three dimensional model, however it will not be properly shaded. Each vertex that is created must include a vertex normal. This object makes it possible for the 3-D rendering software to properly light the surface. Without this information, the entire object will be one flat color, and any surface detail will be lost.

Vector normals can be calculated through the use of the cross product. Each polygon is constructed from three points. These three points can be utilized to construct two vectors. A face normal can then be calculated through the cross product of the two vectors. This creates a vector that is perpendicular to the surface of the polygon, indicating the direction it faces when calculating surface lighting. Figure 39 shows how two vectors can be constructed from the three polygon points and the resultant face normals. This however, is not enough information for Vizard to light the surface.

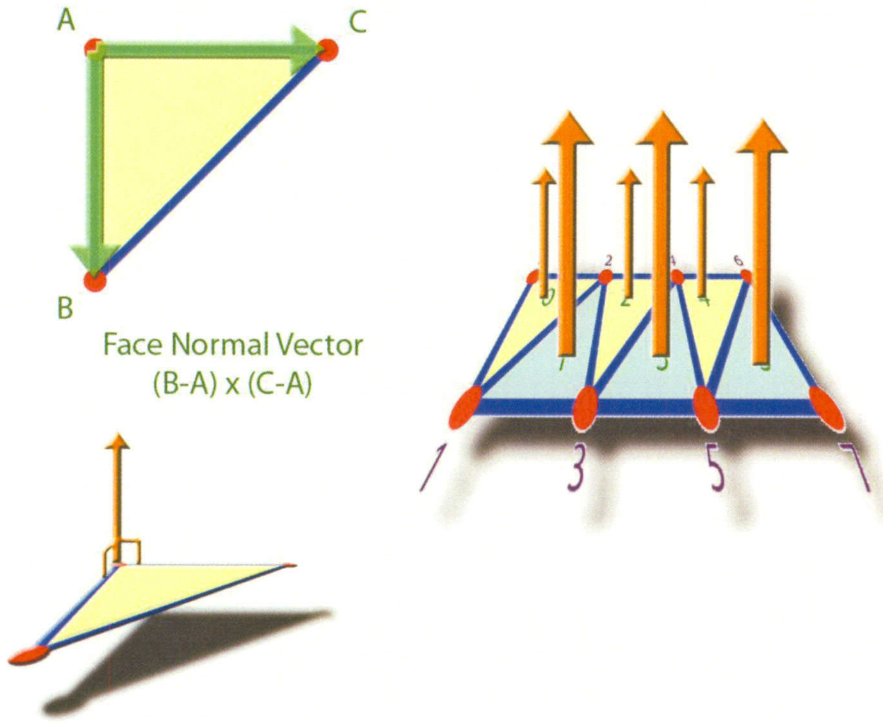


Figure 39: Face normal calculation requires utilizing the cross product of the vertices.

Each vertex can also contain a normal, which is the unit vector resulting from the summation of all nearby face vectors. Calculation of the vertex normal can be difficult because depending on whether a vertex is a corner, edge, or center element a different set of face normals must be summed. Figure 40 summarizes the different face normals that each of the vertices require in creating the final vertex normal.

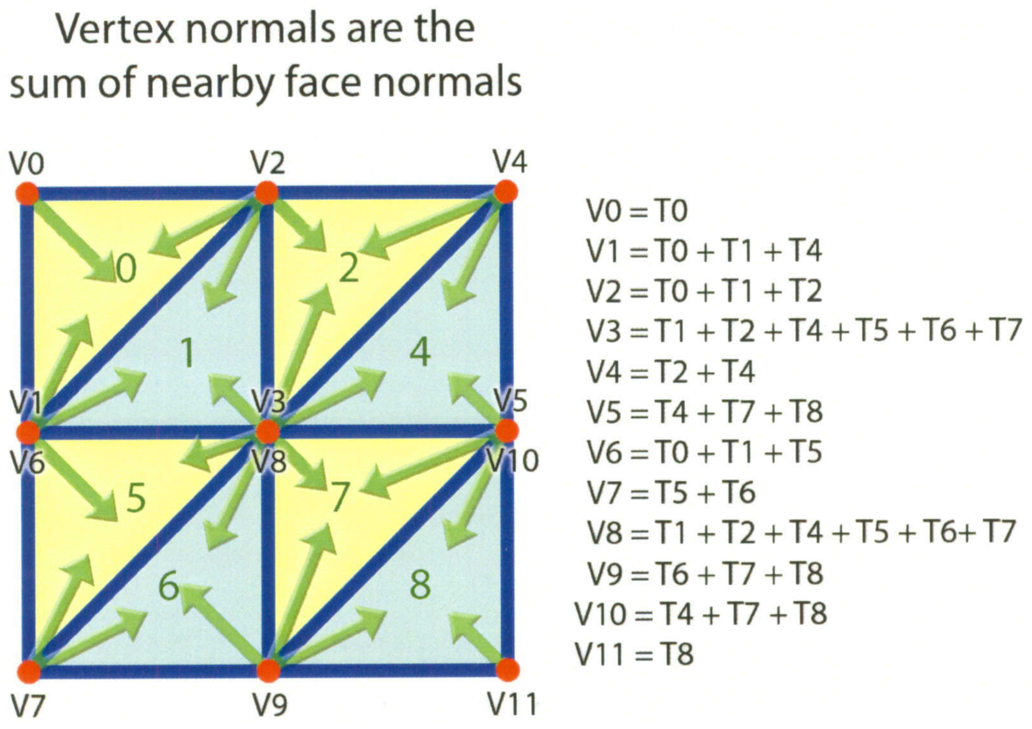


Figure 40: Vertex normals are calculated by summing nearby face normals.

3.1.2 Isosurface and Voxel Visualizations

Isosurfaces are three-dimensional objects that are created to represent a change or gradient in a data set. The most popular algorithm for creating iso-surfaces is the marching cubes algorithm, which was covered in the background section.

WorldViz Vizard™ currently lacks the ability to create iso-surfaces. To create an iso-surface, a data set needs to be loaded and assigned to an array structure. Each array element pertains to a vertex in the yet-to-be-constructed 3-D surface.

To create the iso-surface, the array must be iterated over every square set of vertices for 2-D data and every cube set of vertices for 3-D data as shown in Figure 41.

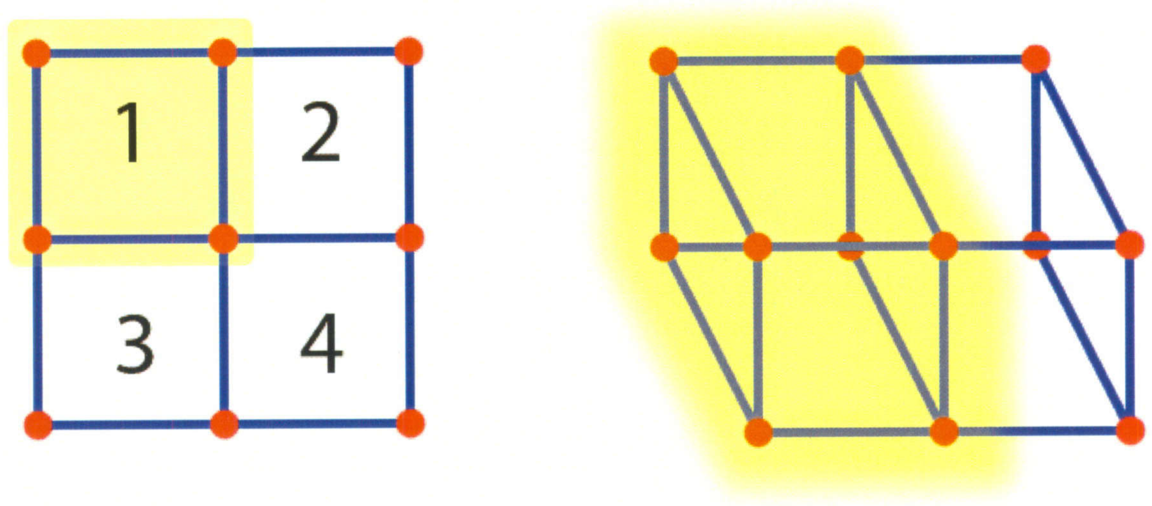


Figure 41: Each set of four vertices must be iterated over in a 2-D marching cubes algorithm while every cube of eight vertices must be iterated over in the 3-D version.

A cutoff value must be set while iterating over these array elements to determine how to surface should be constructed. For the 2-D case, a threshold is set and the elements in the square analyzed. If all of the values are above or below the threshold the algorithm does not produce a 3-D surface at that location. This is because if that area is completely above or below the threshold, then it is inside the iso-surface or outside of it. An iso-surface only marks the boundary of a set threshold and does not visualize elements elsewhere.

For a square with only two states for each vertex (in or out) there are 2^4 or 16 possible combinations. These combinations can be seen in Figure 42.

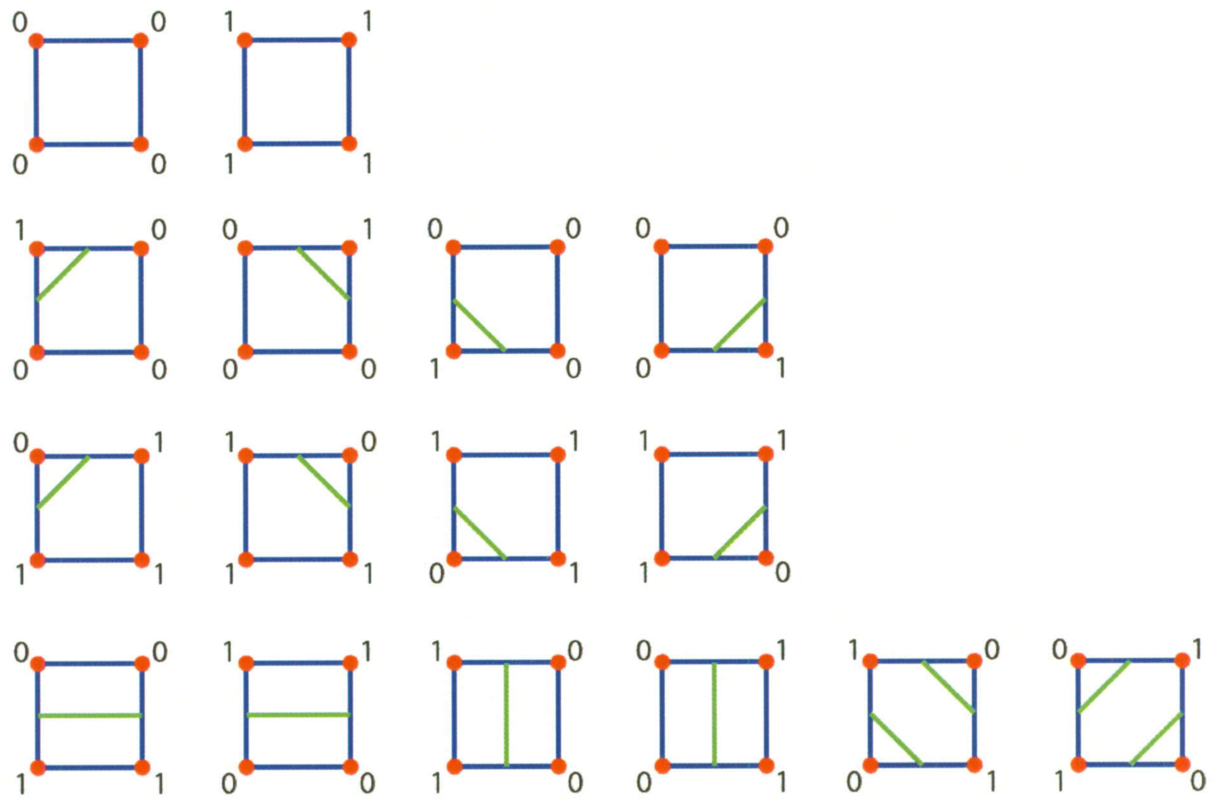


Figure 42: The 16 cases in the 2-D version of the marching cubes algorithm.

For the 3-D version there are eight vertices to consider for a total of 2^8 possible combinations. As covered in the background, these cases can be simplified to 15 base cases. These 15 base cases must use rotations and inversions to create the total 256 possible combinations.

As the squares or cubes are being iterated over the algorithm determines the particular type of geometry needed based on a lookup table of elements. The correct model is then placed at that location in 2-D or 3-D space.

The difficulty in this algorithm is creating a fast and efficient system and creating an effective lookup table that encompasses all of the possible cases.

Voxels are volumetric pixels which take up a discrete segment of 3-D space. Voxels are composed of scalar values indicating the magnitude of some attribute at a particular location in space.

Voxels are not typically visualized due to slow performance. Usually iso-surfaces or other visualizations are utilized to view the voxel information. However, for small sets of voxels it can be an effective visualization tools and provide a different method for viewing data.

Since voxels are three-dimensional objects and have a discrete value for each quantized location in 3-D space, the amount of data to process and visualize is very large. A small voxel volume of 100x100x100 elements constitutes a million values that need to be processed. In addition to that, cubes are normally used to view a voxel region and each cube is created by 8 vertices which translate into 8 million total vertices to calculate per frame. This technique is currently very limited in terms of maximum resolution. For this reason, voxel visualizations can only utilize a small array with no more than 75^3 elements on currently available hardware with the current implementation of OpenGL.

Creating the voxel visualization is similar to the marching cubes algorithm, except each cube is located around each array element as see in Figure 43. This is unlike the marching cubes algorithm which places each array element at each vertex of the cube.

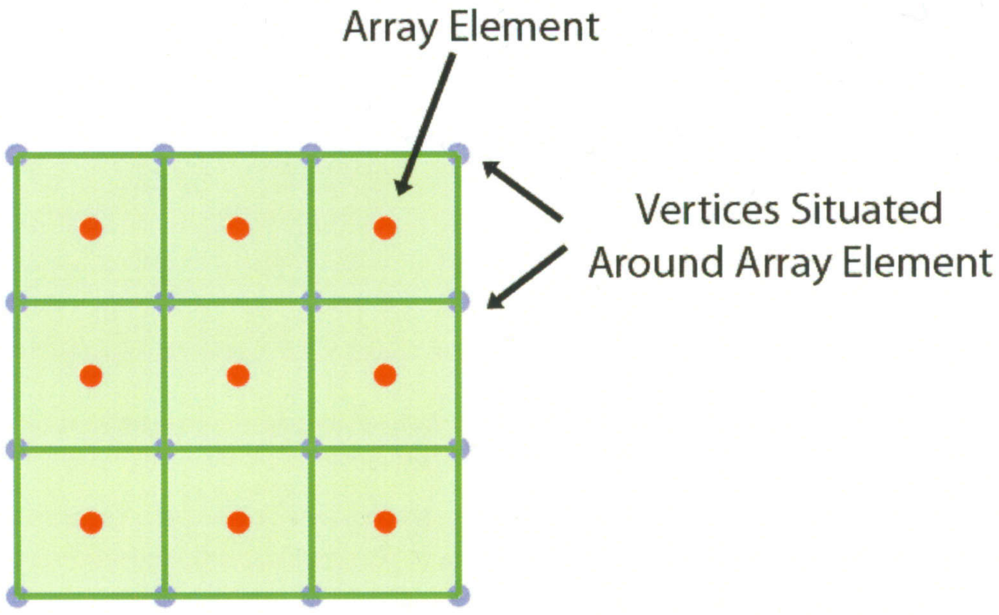


Figure 43: Voxel visualizations have the geometry constructed around the array elements, rather than have the array elements coincide with the vertices.

Each array element is iterated over and a cube is placed in the location. To distinguish between different values the cube is transformed through an attribute. Color and transparency are two parameters that can be set based on the values inside of the array. Low values can correspond to a particular color and transparency range while high values can correspond to another. Figure 44 is an example of how voxel visualizations would utilize color and transparency for different values to convey the magnitude at each location. Higher values receive a color of red and are more opaque. Intermediate values are a purple hue with some transparency and low values have a blue color and high transparency.

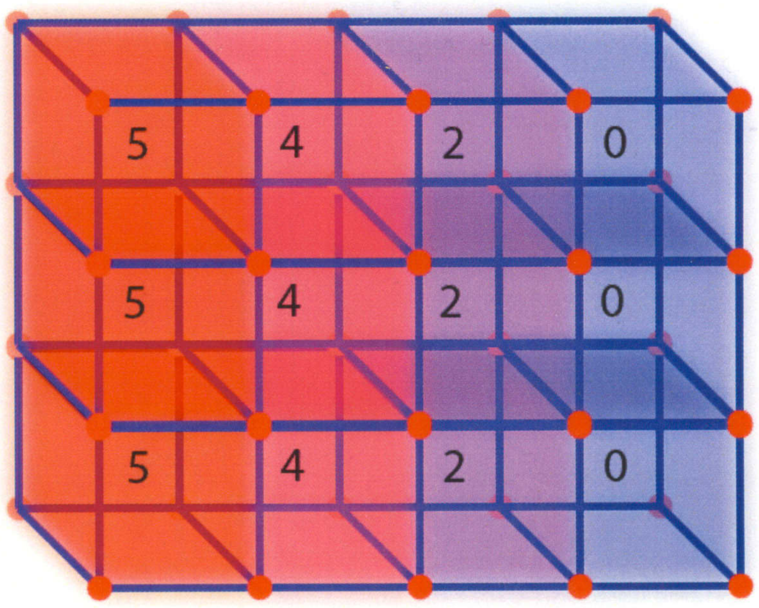


Figure 44: A voxel representation utilizing both color and transparency to show the value at an array element. Larger values are opaque and red while smaller values are transparent and blue.

3.1.3 Hedgehog plot

As covered in the background, hedgehog plots are useful when visualizing vector quantities such as the flow of a fluid. To create the hedgehog plot, the OpenGL LINES type is utilized when constructing the individual vertices. Every two points entered will define a distinct line. Similar to the iteration method for the triangle strip, the array information is best mapped to the vertex when iterated over every second point. The first point is located at the spatial origin of the data point while the end is mapped to the vector quantity describing the flow. To create the required points, iteration needs to be performed over every vertex. The first point for each line will be equal to the iterator's current location in the array. The second point will have the x, y and z components equal to the vector quantities at that array index for the x, y and z data set. Figure 45 shows how the array index and the array value are utilized when creating hedgehog visualizations.

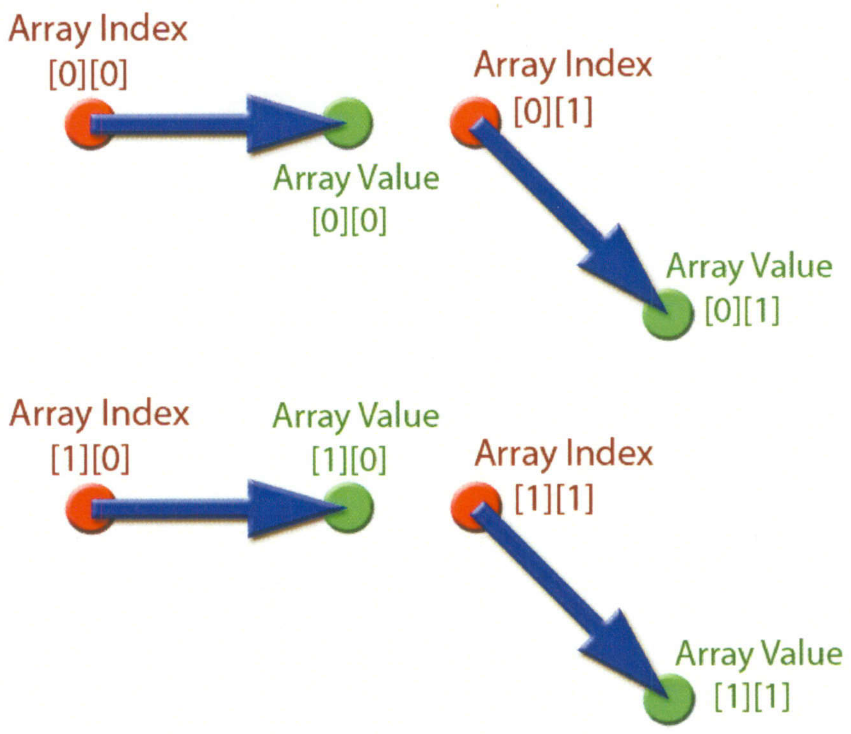


Figure 45: Hedgehog plot algorithm, utilizing the array index as the first point and the array value as the second.

When creating a hedgehog visualization that changes over time the algorithm can be modified for efficiency. Rather than iterate over every vertex, only the odd vertices need to be changed. The first vertex in each line will be an even numbered vertex and does not need to be moved from its spatial position. Only the end of the line needs to move as the vectors change.

Two additional options are available when visualizing hedgehog plots. Rather than normalize the data, the original values can be used to show the magnitude through the lengths of the lines. Color can be utilized to show differences in magnitude between lines.

For the data visualization approaches covered, Figure 46 shows how to decide which visualization to utilize depending on the received data. Most forms of data used for scientific visualization can be broken into vector and scalar information. Depending on the dimensionality of the data, several different visualizations can be utilized. Due to the large amount of data in a 3 dimensional scene no vector visualization has been created. While the 1-d and 2-D hedgehog and cone plot visualizations can be utilized in three dimensions, occlusion problems as well as difficulty in sorting the data make this impractical.

For scalar data, line graphs and surface plots are possible. With three dimensional data of small datasets, voxel representations can be utilized. However, due to the high number of elements in a three dimensional scalar array, very large dataset can cause poor system performance. Isosurfaces can also be utilized; however these require creation outside the Vizard™ program in order to view them in real time.

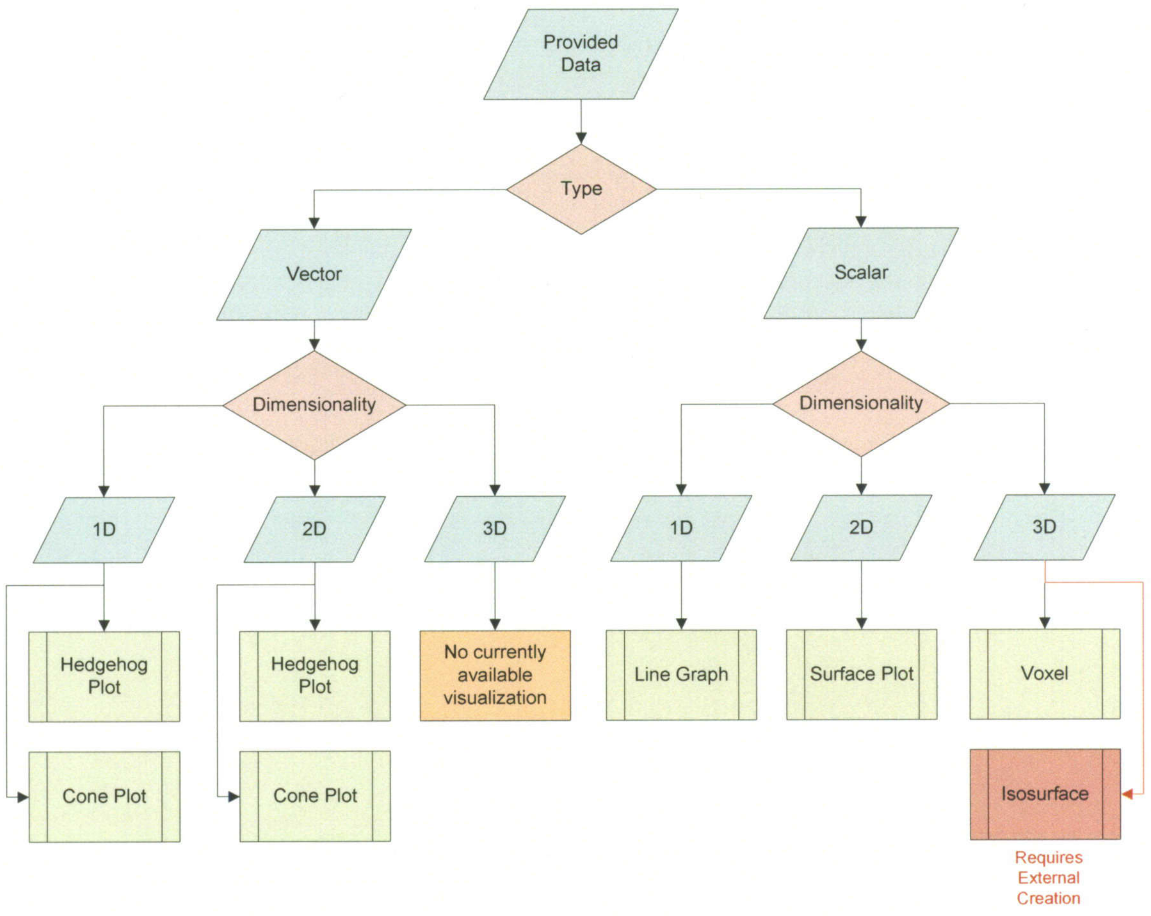


Figure 46: Visualization implementation flow chart.

3.2 Integrated Awareness Graphical, Measurement and Health Data

Previous work has categorized three forms of data when creating a virtual reality environment as well as the general pathway structure to create a virtual world [39]. Here, the development of the visualizations for a virtual environment will be covered. Each of the three data types will be elaborated upon through several virtual reality demonstrations.

3.2.1 NASA Rocket Engine Test Stands

The first demonstration contains a series of several virtual worlds developed to visualize rocket engine test stands for Stennis Space Center in Mississippi. These worlds were created with the direct intent of viewing Integrated Systems Health Management (ISHM) systems. ISHM consists of processes managing erroneous conditions that systems may encounter during their operational life by either designing out failures early on, or defending and mitigating any possible failures [40]. Each of the three forms of data must be integrated into a virtual environment to visualize the ISHM rocket engine test stand.

For these virtual world; graphical, measurement and functional data had to be visualized and incorporated. This project had several stages of development in visualizing these data types. The first stage was developing a virtual representation of the E-1 rocket engine test stand. The E-1 test stand is one of several test stands constructed to develop propulsion systems using high-pressure gases and cryogenic fluids. A Pro Engineer™ formatted file was provided with 3-D model information about the structure.

For the second stage of the project, a different test stand was targeted for visualization. The Methane Thruster Test bed Platform (MTTP) trailer's drafted structural information and excel spreadsheets of test firing data were provided. This was the first time a complete package of data was provided for visualization.

In creating these virtual worlds, each kind of data had to have visualizations created to emphasize their meaning. In the MTTP trailer, several kinds of data needed visualizing. Each type of data is listed below in Table 2 along with a description.

Table 2: The data types provided for the NASA MTTP trailer.

Data Type	Data Description
Valve State	Binary number indicating the physical state of the valve being open or closed.
Valve Command	Binary number indicating the desired state of the valve being open or closed.
Feedback	Percentage value of feedback in the signal.
Pressure	A measurement in PSIG of the pressure.
Current	A measurement in amps of the current flowing through the igniter.
Temperature	A measurement in degrees Fahrenheit of the temperature.
Strain	A reading in lbs of the strain of the rocket nozzle.

For this virtual world, the data was not part of a larger array of elements, meaning that hedgehog, voxel, and other visualizations previously covered are not relevant. Each type of data corresponds to one sensor reading at a set location, with a low overall sensor density across the entire structure. This means novel visualization that can reflect information at a singular point have to be created. Figure 47 shows the generalized layout for creating a VR world that utilizes graphical, measurement and functional data.

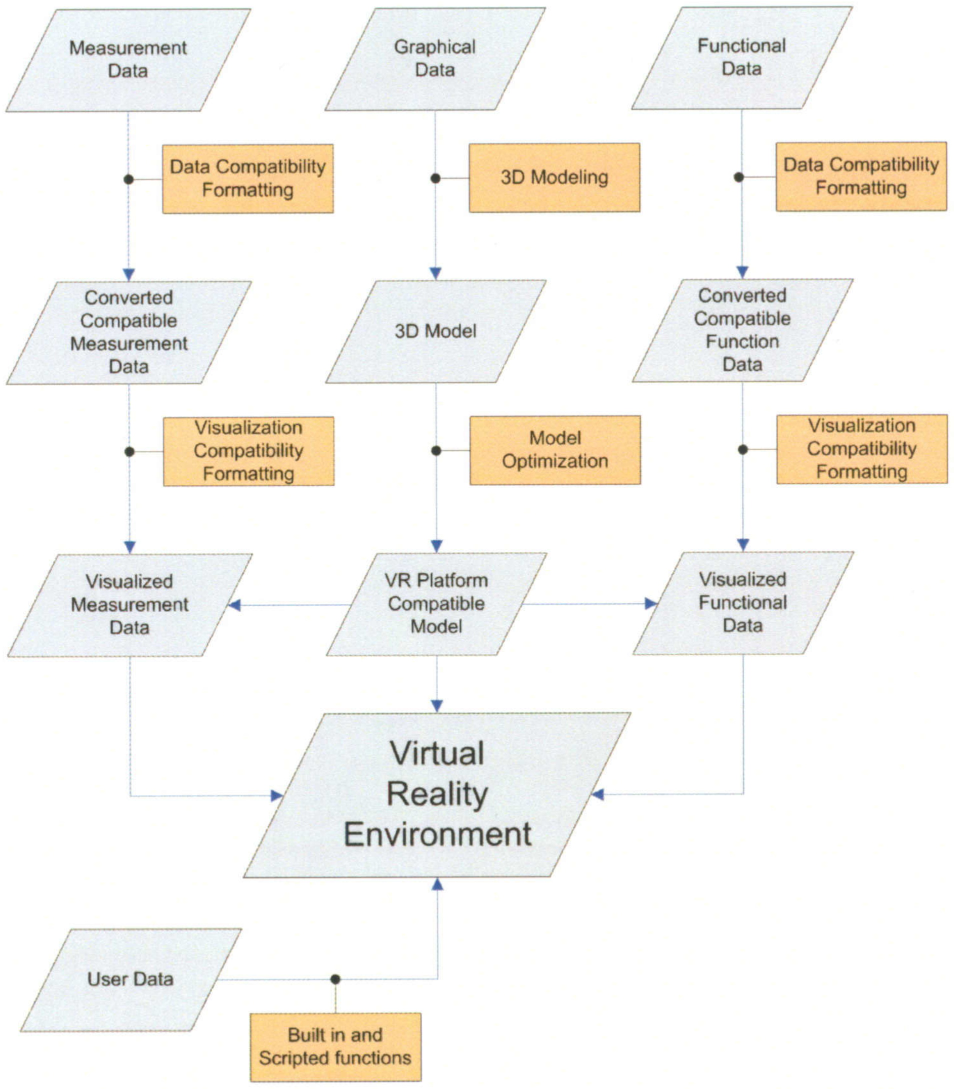

Measurement Data
Graphical Data
Functional Data
Data Compatibility Formatting
3D Modeling
Data Compatibility Formatting
Converted Compatible Measurement Data
3D Model
Converted Compatible Function Data
Visualization Compatibility Formatting
Model Optimization
Visualization Compatibility Formatting
Visualized Measurement Data
VR Platform Compatible Model
Visualized Functional Data
Virtual Reality Environment
User Data
Built in and Scripted functions

Figure 47: Integration of graphical, measurement and functional data in a virtual world. [39]

Figure 48 is a flowchart depicting the pathway needed to create the MTTP 2 virtual environment. This pathway could be utilized in any situation when graphical, measurement and functional data need to be visualized. The flowchart is broken into three main branches, culminating with the virtual environment. Graphical data is depicted in green, functional in orange and measurement in blue. Data is represented by the parallelograms and processes by the rectangles. The final branch incorporates user input into the virtual world to provide navigation.

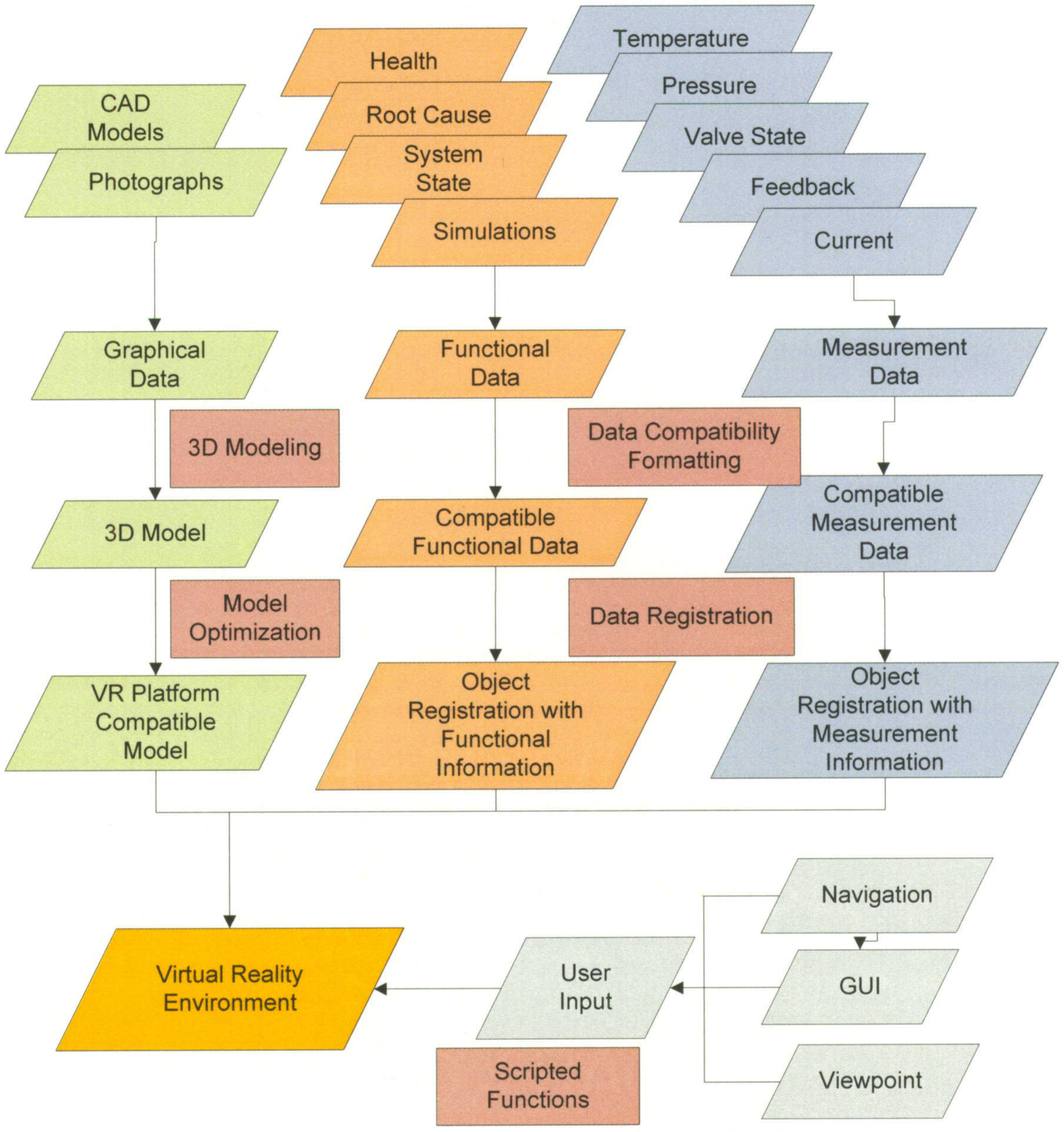


Figure 48: MTTP flowchart for data visualization.

The MTTP 3 demonstration utilizes the same flow for creating and assigning data as in Figure 47. However, due to several additional features and expanded class structures, the demonstration has become more complex. Figure 49 shows an overall breakdown of the demonstration's structure. Classes are shown with the wavy-box, arrays are the parallelograms, callback events are the diamonds and cylinders are external data files.

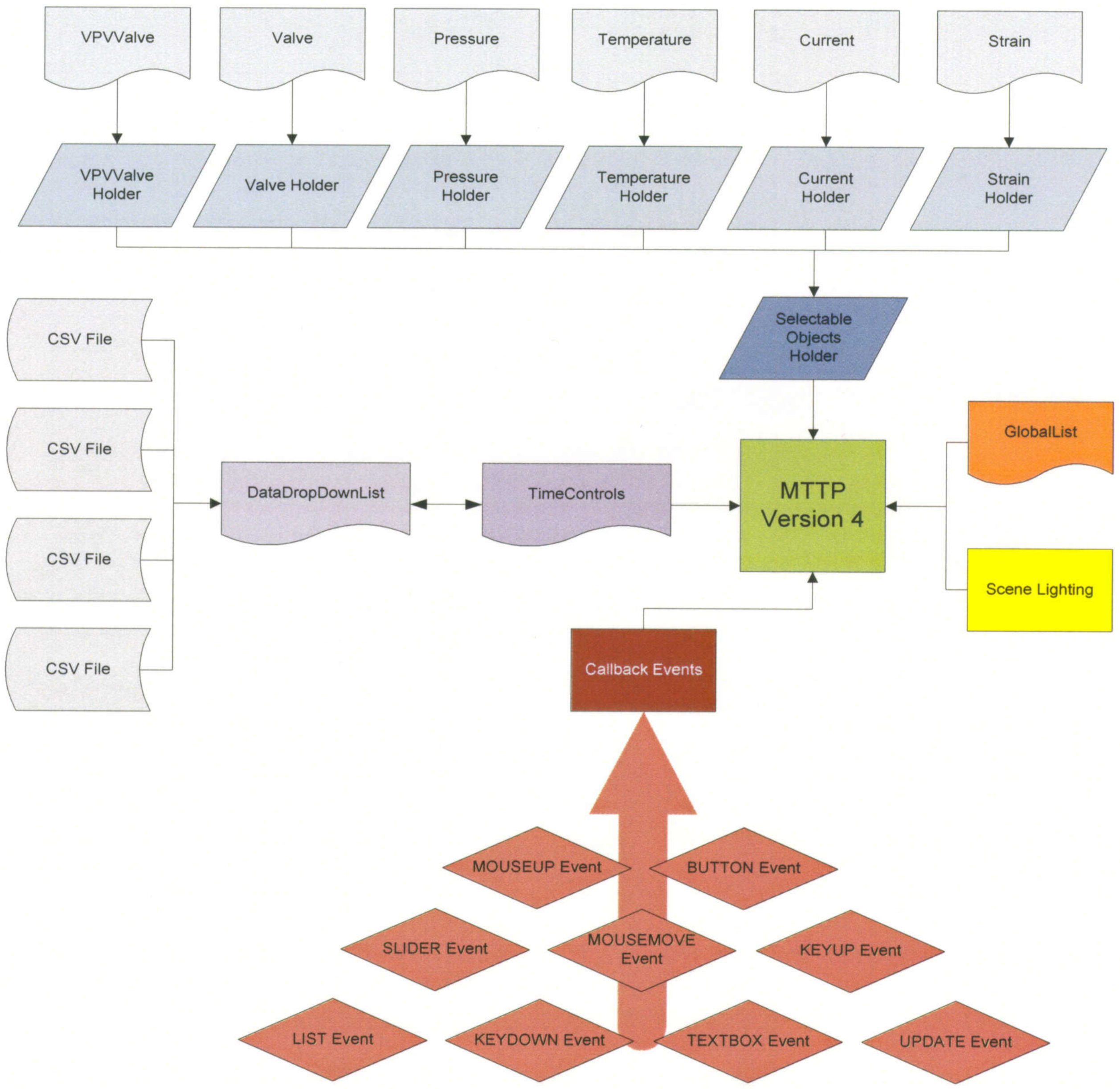


Figure 49: MTTP 4 flow chart of class and functional dependence.

The MTTP 3 demonstration utilizes object selection to allow the user to select individual components. When an object is chosen, a list displaying available information such as the name and visualization available will be shown. The user can make further selections pertaining to that object.

In order to facilitate object selection, each of the instances for pressure, temperature, etc will require an array to house them. These arrays will then make up a larger array which will contain all selectable objects in the scene.

Through the use of callback events, which are specialized functions that are called when a particular event occurs, the previously mentioned arrays will be cycled through to find the object the user has interacted with. Then, based upon their selection the appropriate functions will be called for that particular instance and its visualizations will be changed.

In order to allow for global changes to the scene so that a user does not have to interact with two dozen objects a global class was added. This class is a list of buttons which will expand to show the possible choices the user can choose for each class type. Through the use of the callbacks, when a user makes a selection the appropriate array will be cycled through and all visualizations enabled.

The final major addition to this version of the MTTP trailer is the ability to manipulate time. To access the CSV file information for the many test runs a drop down list will be implemented created. The user would be able to select a list item and the data would load across all objects. To control time across the entire demonstration, a time controls class will be required which contains both the data as well as the current time step and time controls. All the sensor classes will listen to this class to get the current time step as well as data element so they are in synch.

Of the many processes which take place during the running of the MTTP 3 demonstration, two are complex enough to warrant additional flowcharts. Figure 50 breaks down the task of identifying objects that the user can interact with. This flowchart begins with the mouse-move event. This event is called every time that a movement is registered on the mouse.

There are two modes that can be enabled based upon the user's desire to navigate or select objects. If the user is currently in navigation mode by pressing and holding the left 'ALT' key on the keyboard this functions exits with nothing happening when the mouse is moved. If the user is in selection mode, first the function checks to see if this is the first time this loop has been performed. If this is the first time, some initialization is performed. Since this function is called with each movement of the mouse, it will be called many times before the user settles on the object to select, rendering this initialization phase unnoticeable.

If this is not the first time through the event loop, the function iterates through the entire array holding all selectable objects in the scene. If the function comes across a selectable object that equals the object the mouse is currently over then a series of procedures are performed. First, the object that was selected has its emissivity reduced to a normal level. Emissivity is a function which allows the amount of light transmitted by an object to be increased or set to a particular color. In this case, the emissivity is doubled and set to a red color to show the user they are interacting with a selectable object.

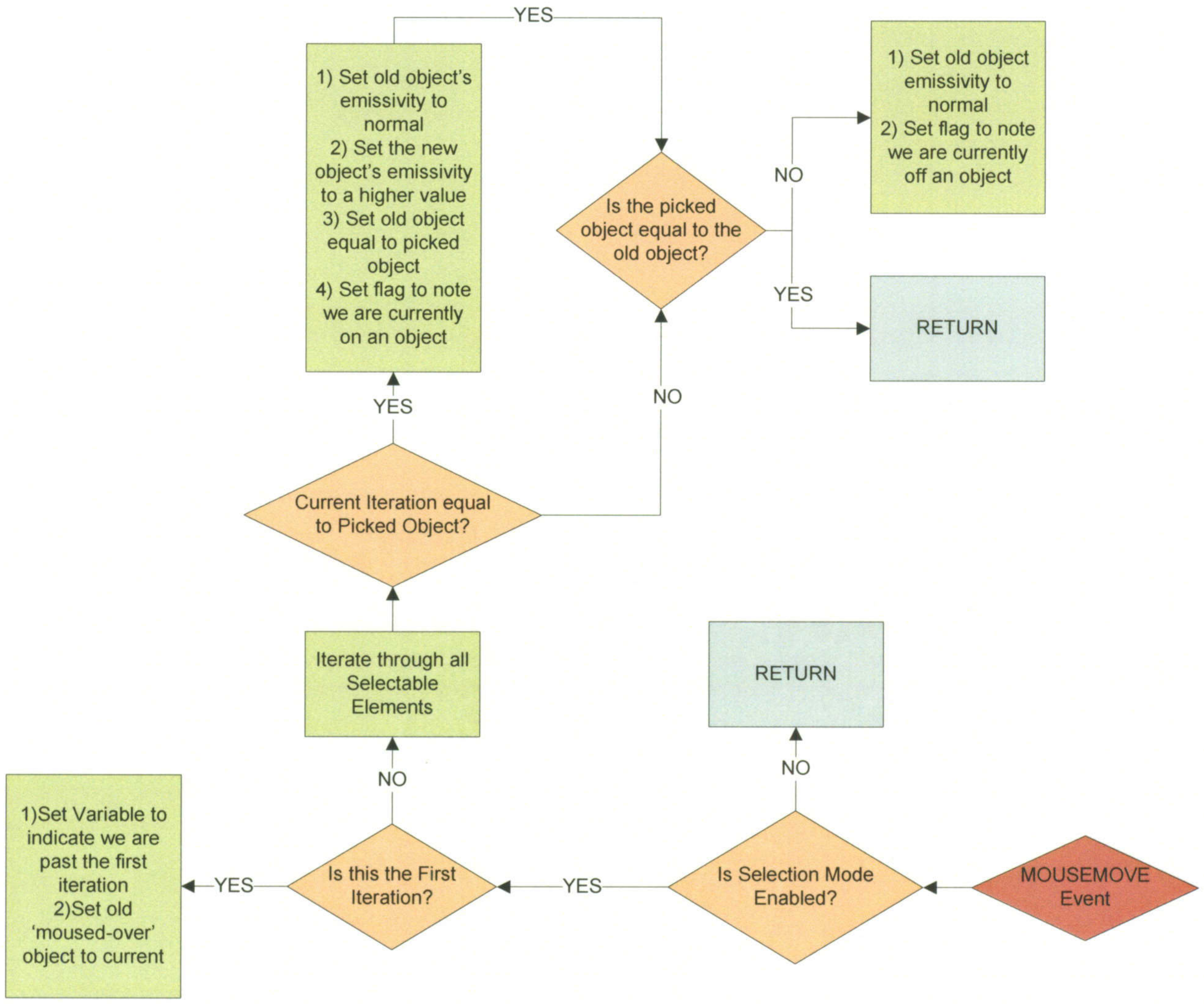

MOUSEMOVE Event
Is Selection Mode Enabled?
NO
RETURN
YES
Is this the First Iteration?
YES
1)Set Variable to indicate we are past the first iteration 2)Set old 'moused-over' object to current
NO
Iterate through all Selectable Elements
Current Iteration equal to Picked Object?
YES
1) Set old object's emissivity to normal 2) Set the new object's emissivity to a higher value 3) Set old object equal to picked object 4) Set flag to note we are currently on an object
YES
NO
Is the picked object equal to the old object?
NO
1) Set old object emissivity to normal 2) Set flag to note we are currently off an object
YES
RETURN

Figure 50: MTTP 3 for highlighting selectable objects.

The variable holding the old object will be updated with the newly moused-over object. Finally, a flag will be set to indicate that the user is over an object. After this step there is a check to see if the picked object is equal to the old object and if it is we simply return and do nothing. However, if it is not the same the old objects emissivity is set back to its normal value and a flag is set to indicate that the user is no longer over an object. Each time a mouse movement is registered this entire sequence is performed.

The second complicated process is when an object that has been highlighted needs to be selected by the user and is shown in Figure 51. To achieve this, the user must click with the mouse on a selectable element. When the mouse is clicked and released, a MOUSEUP event is called. This event will happen every time the user releases a mouse button.

As before, the first check is to see if we are in navigation or selection mode. If the user is in selection mode the variable oldObjectDown will be set to objectDown so that the program can make note of what use to be selected. The variable objectDown is then set to whatever the user has just clicked over to trigger the mouseup event. A check is then performed to see if the objectDown variable is equal to the oldObjectDown variable. If this is the case, the user is clicking on the same object twice. For this demonstration, when this occurs the function will turn off the selection. If the user is not selecting the same object a check is performed to see if there is even an object selected. If there is a selection nothing is done until the user deselects the currently selected element. If there is no selection, the function iterates over all possible selectable objects. If the user has not clicked on a selectable object nothing is done and the function ends. If the user did select a selectable object the target model is moved to the objects location, made visible and scaled to the particular model. The oldObjectDown variable is set to this selectable object; the haveSelection flag is set to true and the objects GUI is activated.

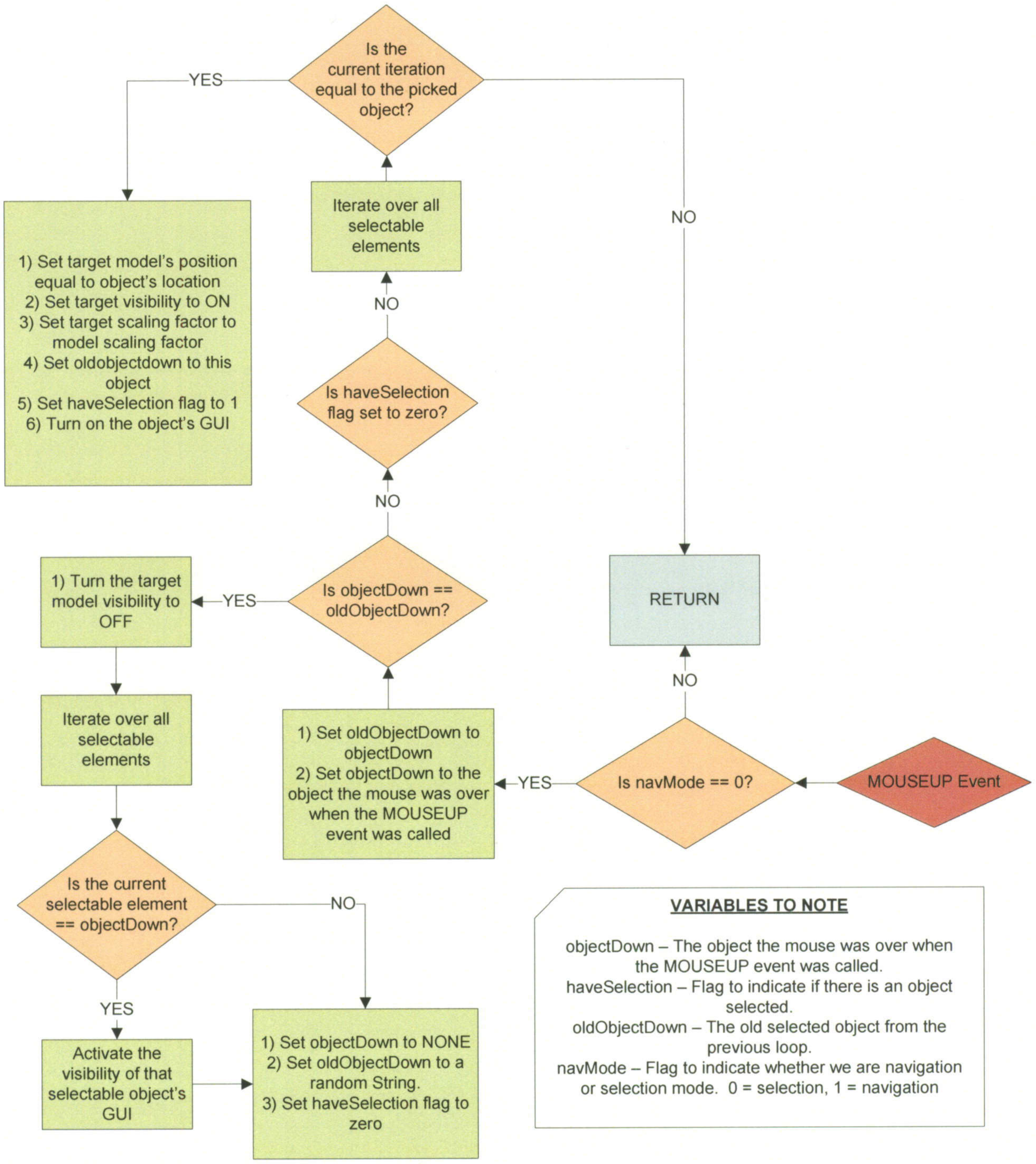

Is the current iteration equal to the picked object?
YES
NO
1) Set target model's position equal to object's location
2) Set target visibility to ON
3) Set target scaling factor to model scaling factor
4) Set oldobjectdown to this object
5) Set haveSelection flag to 1
6) Turn on the object's GUI
Iterate over all selectable elements
NO
Is haveSelection flag set to zero?
NO
1) Turn the target model visibility to OFF
YES
Is objectDown == oldObjectDown?
RETURN
NO
Iterate over all selectable elements
1) Set oldObjectDown to objectDown
2) Set objectDown to the object the mouse was over when the MOUSEUP event was called
YES
Is navMode == 0?
MOUSEUP Event
Is the current selectable element == objectDown?
NO
YES
Activate the visibility of that selectable object's GUI
1) Set objectDown to NONE
2) Set oldObjectDown to a random String.
3) Set haveSelection flag to zero
VARIABLES TO NOTE
objectDown – The object the mouse was over when the MOUSEUP event was called.
haveSelection – Flag to indicate if there is an object selected.
oldObjectDown – The old selected object from the previous loop.
navMode – Flag to indicate whether we are navigation or selection mode. 0 = selection, 1 = navigation

Figure 51: MTTP 3 Object selection flowchart.

3.2.2 3-D Tomographic Data Model Visualizations

Tomographic 3-D models are derived from reconstruction algorithms. In the forthcoming visualizations, only X-ray and optical reconstructions will be viewed, however any 3-D model from reconstruction data could be visualized.

These visualization techniques will be employed through two Vizard virtual reality environments. Of primary concern when visualizing these objects is having the ability to directly compare the same particle from different reconstruction techniques. Mathematical analysis can provide a hard number for the similarities between two separate reconstruction techniques, but being able to visually see how separate algorithms perform is a powerful tool.

The first demonstration will utilize sensors to allow the user to move the particles in 3-D space. The user can overlap the objects and rotate them independent of one another to try and match their shapes. One difficult task in reconstructing a particle with two different methods is re-aligning the object so that they can be viewed from the same vantage point. This hands-on visual realignment of the particles could provide an answer to that problem.

The second demonstration will utilize database information to present details about the particles to the user. This demonstration will feature a graphic user interface (GUI) and dispense with the 3-D sensors so that it can be utilized on any computer station for quick particle comparison.

Figure 52 outlines how to construct the first sand demonstration. The first half of the process concerns getting data and the reconstructions from the X-ray and optical setup. The second half accesses the particle database and utilizes the information to create the final virtual reality world. Finally, through sensors and foot pads, the can interact with the virtual world.

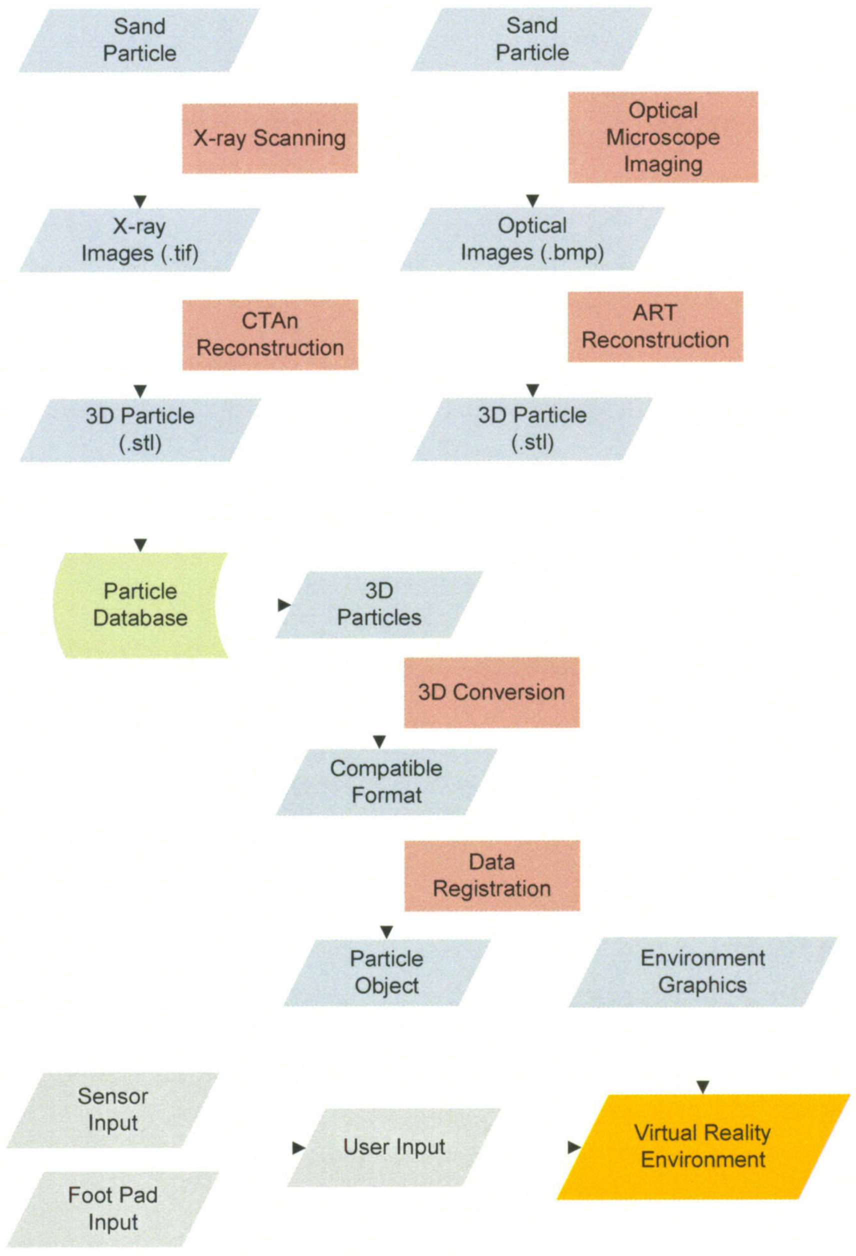


Figure 52: Sand demonstration 1 flow chart.

Figure 53 is the flowchart for the second sand visualization demonstration. The first half is the same as the previous flowchart. The second half, however, now utilizes an XML convention for storing details about the particle including name and reconstruction type. In addition, user input relies solely on a mouse interface.

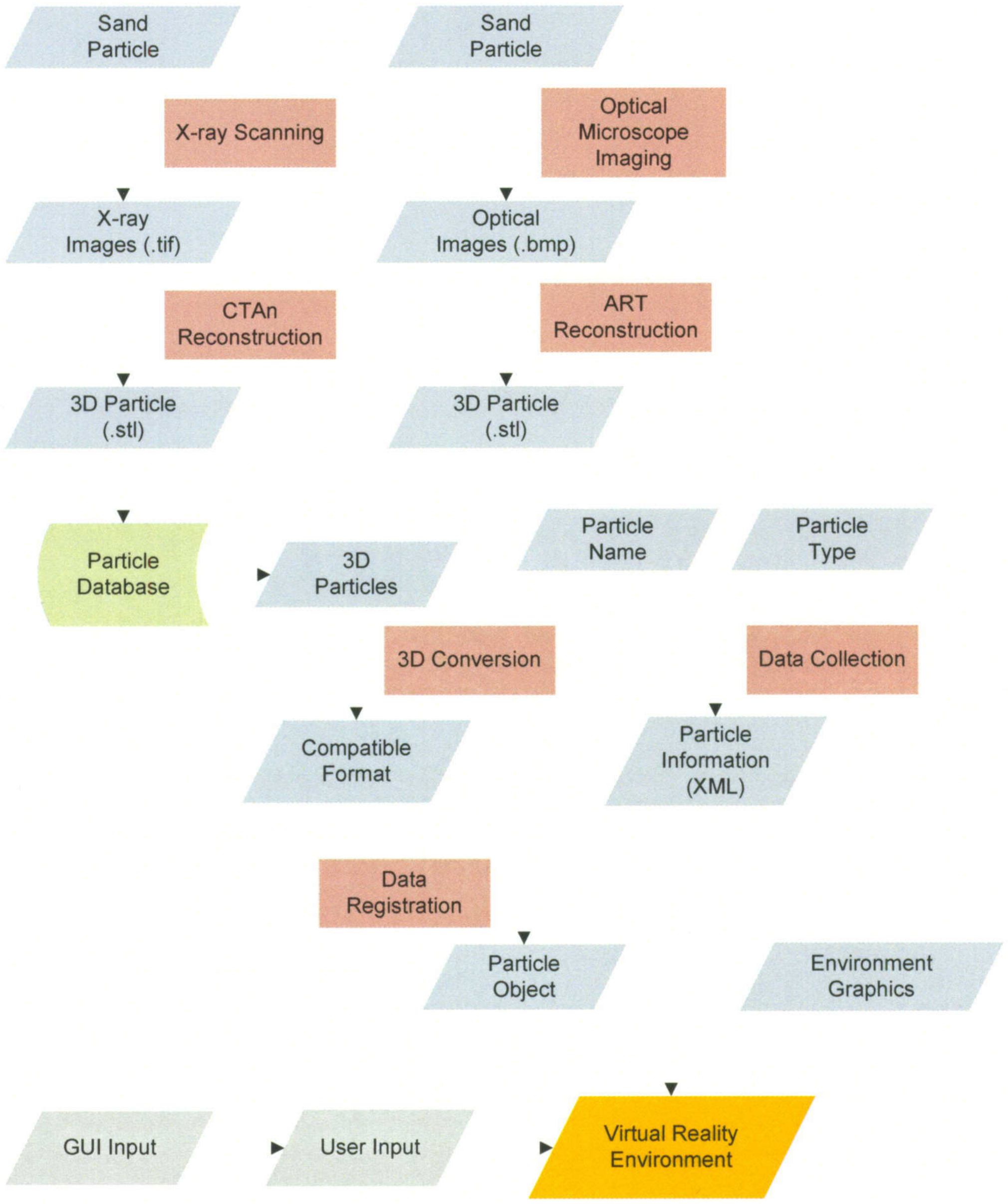


Figure 53: Sand demonstration 2 flow chart.

3.3 3-D reconstruction models of tomographic data

The current reconstruction algorithm in place for creating 3-D models of optically scanned sand particles is limited to processing images at resolutions no higher than 40x40 pixels. Previous algorithms have had run times exceeding four hours for these low resolution models. For these reasons, a new algorithm needs to be developed to speed up reconstruction time and provide higher resolution models. Figure 54 is a relative size comparison of different resolution images. The previous algorithm utilized images about ten pixels smaller than the 50x50 image. However, this is not the true resolution of the particle since in many instances the actual object only took up a 20x20 pixel region, which is equal to the blue square in the bottom left of Figure 54. An enlargement of the pixels for a sphere of size 20x20 can be seen in the upper right hand corner of Figure 54.

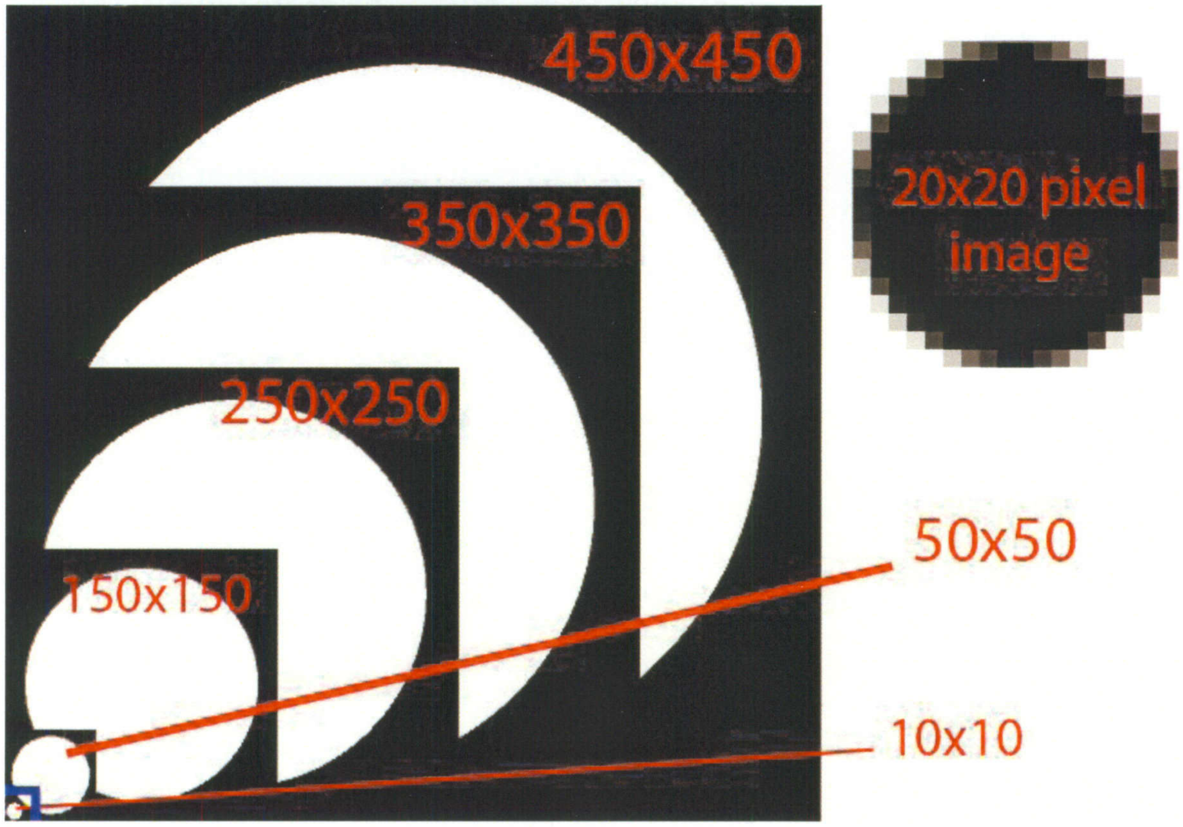


Figure 54: Relative size comparison between various sized images. Not the upper right corner which is a blow-up of a 20x20 image.

Capturing these images required an optical setup, which had been previously developed [5]. The setup for taking images places the object on a rotating stand. A strong light source would be centered on the object to increase overall contrast with the black background. A camera was focused on the object and an image taken at one degree increments. Figure 55 is a visual breakdown of the process for capturing sand particle images.

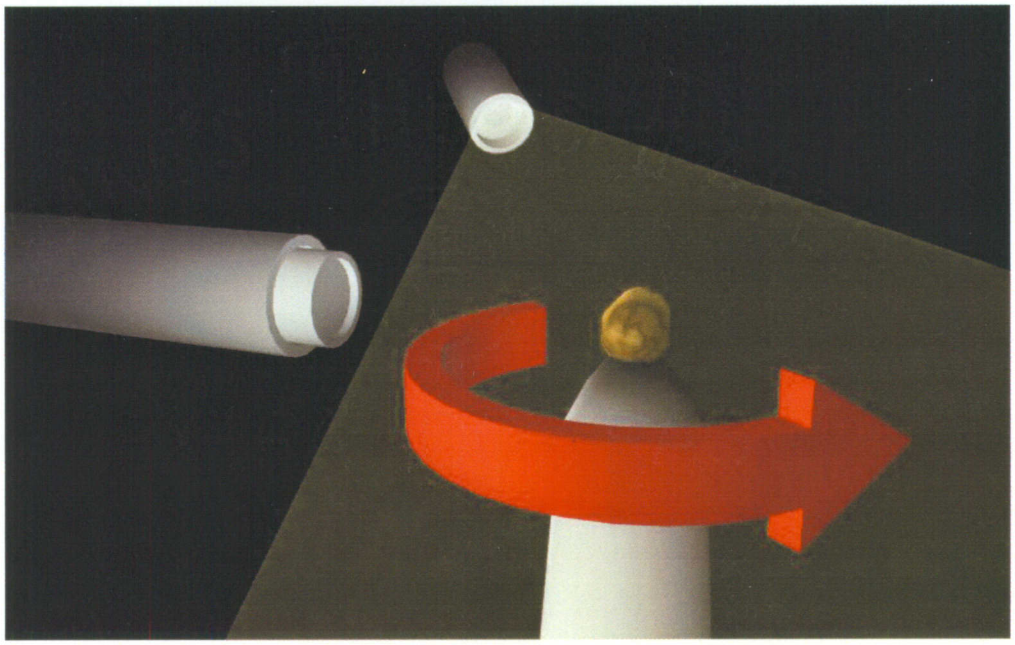


Figure 55: Capturing the images of the particle utilizing a rotating stand, light source and camera.

Once the images had been gathered, they were stored in an online database. The database images have a resolution of 348x260 pixels with the particles taking up no more than a quarter of the available pixels.

The captured images can have several faults. Figure 56 shows a progression of preprocessing commands meant to enhance the original image. The left most image is the original capture by the optical setup. Next to it is an image of the sand particle with adjusted levels to increase the contrast of the shape against the background. Finally the image is cropped to remove areas that feature no particle information to improve reconstruction times.

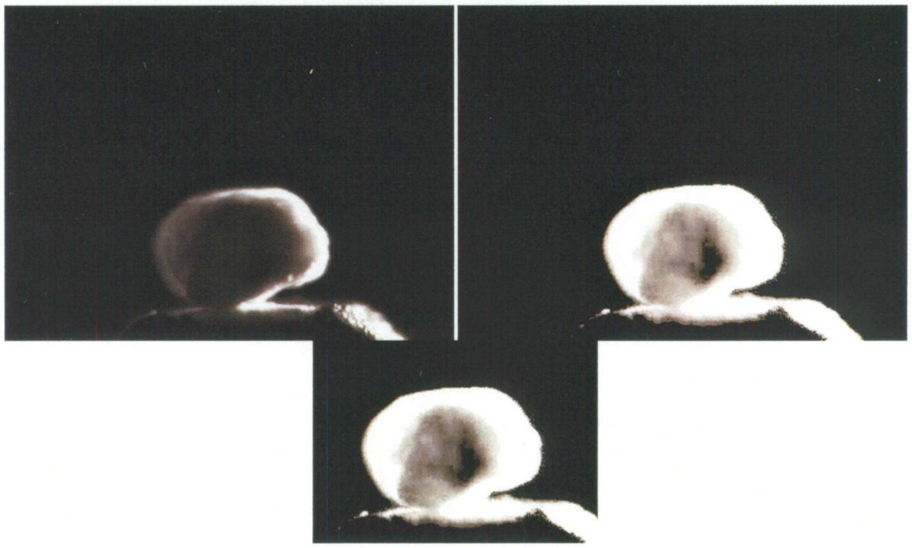


Figure 56: Initial images are not always the best to process. Preprocessing techniques such as level adjustment and image cropping help improve performance.

To create the 3-D model of the sand particle, the images need to be deconstructed. The actual algorithm can be likened to a Boolean operation. The images are each taken from a distinct angle, with only the boundary of the object important. If these images are aligned around a central axis then the intersection will reveal an approximation of the original model, which can be seen in Figure 57.

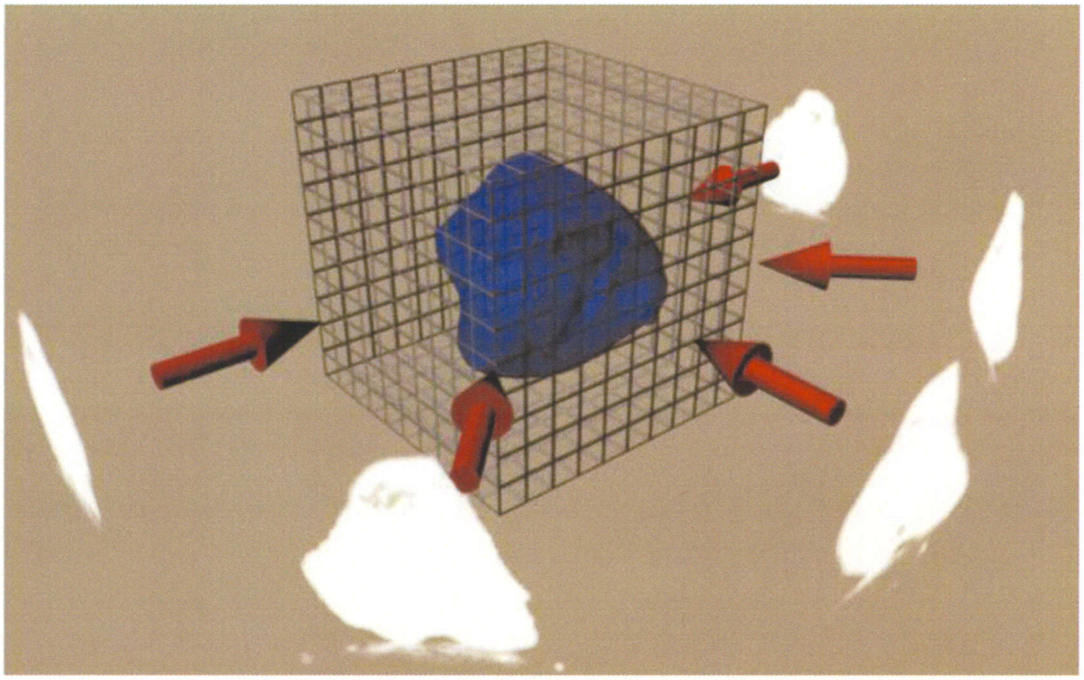


Figure 57: A visual guide to the particle reconstruction algorithm.

The square region around the particle in Figure 57 corresponds to the voxel space. Due to noise a simple intersection of all the images will not provide a perfect solution.

The algorithm for creating these models transforms the images to black and white based on a set threshold. Values above the threshold are set to white while values below it are set to black. This threshold provides leeway when reconstructing the algorithm to help filter out noise.

Passing an image through a voxel space is a simple task when at ninety degree steps; however at angles other than ninety degrees there is no simple solution. As shown in Figure 58, ninety degree projections can slide right into the voxel space. Off angle projections do not line up precisely with a given region and require a transformation.

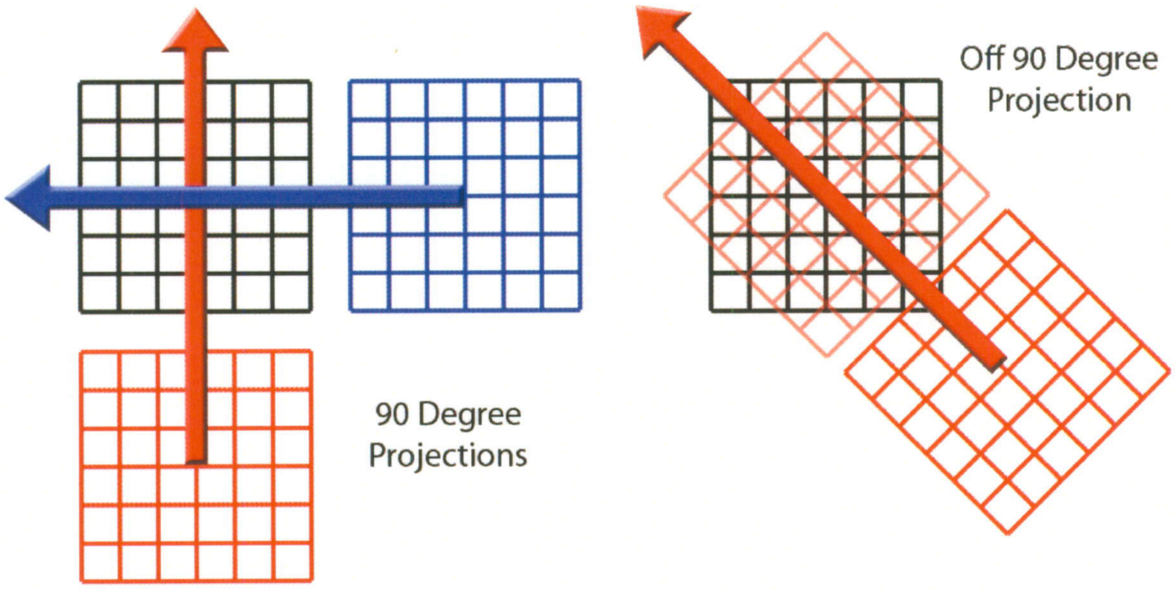


Figure 58: When passing images through the voxel space, 90 degree increments are simple. Off angle projections are more difficult because they do not directly line up with the rectangular grid.

To pass these projections through the voxel space the images have to be deconstructed into rows. This process can be seen in Figure 59 with a magnified image of a sand particle being broken down into several images only composed of the row information.

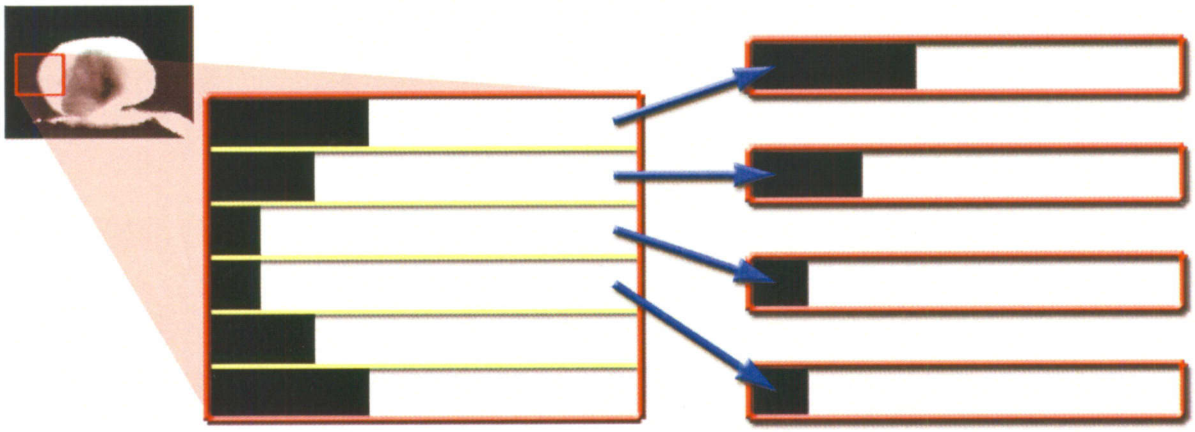


Figure 59: The first step of the algorithm divides each row of the image and saves them to a separate file.

Each new row image must then be stretched outwards twice the size of the original image height, as shown in Figure 60.

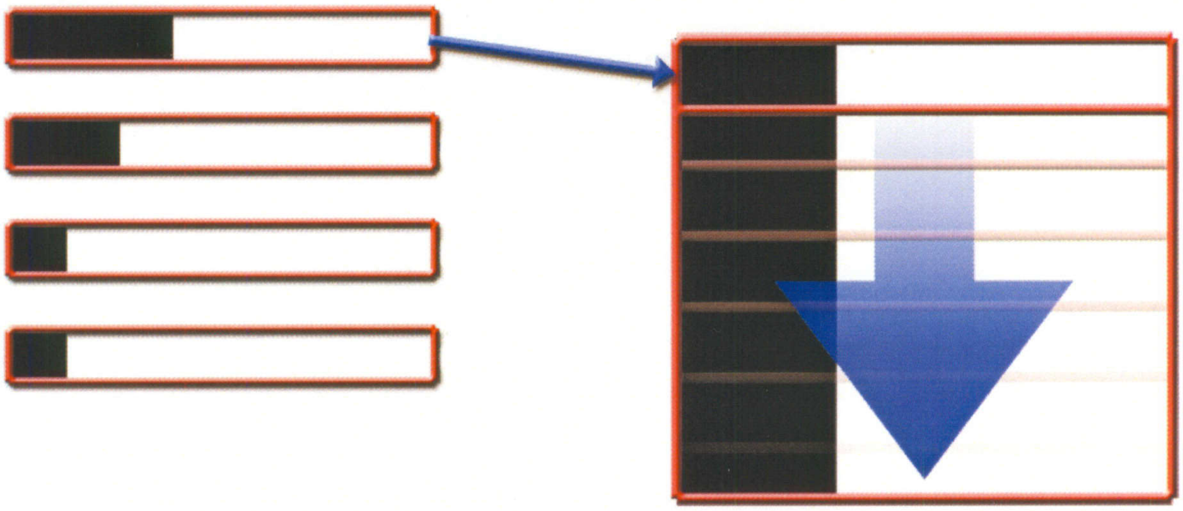


Figure 60: The second step in the reconstruction process is stretching the weights to the size of the original matrix.

The row images must then be rotated based upon the angle the original image was taken from the particle. The size of the row images must be longer than the original image so that when the row image is rotated it extends the full length. This problem and solution is illustrated in Figure 61.

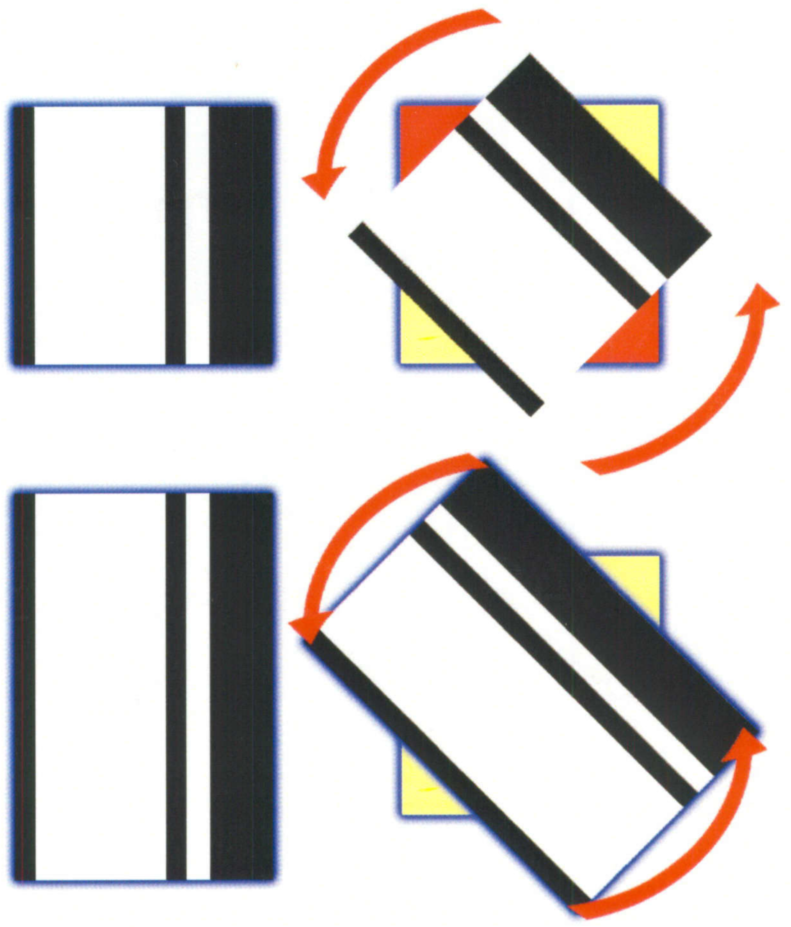


Figure 61: If the row height is not long enough, when it is rotated it will not fill the entire space as seen on the top. As seen below, when the image is properly resized it will cover the missing area.

While in many cases the missing region of the row image will not affect the outcome of the particle there is one case where it can have an effect. If the particle is not centered on the rotating platform will drift across the image. Because of this, the image needs to be stretched completely back across the voxel space to ensure proper reconstruction regardless of improper setup.

After all the rows have been resized and rotated they must be merged back into respective voxel spaces. This means that for each rotation, all the rows that have now been converted to stretched out images will be vertically united as shown in Figure 62.

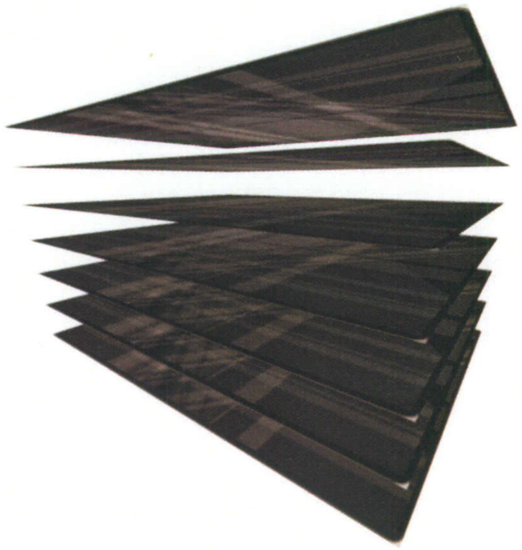


Figure 62: Every row image is united into a 3-D voxel matrix.

After the 3-D voxel matrices are created, they can be added together to create the final resultant matrix. This matrix will have a maximum value equal to the number of projections. This is because the image was initially converted to a binary type of black and white (0 and 1). At the maximum value, all projections agree that this is an area that corresponds to the object scanned. At the lowest value of 0, no projections include that voxel area as a part of the original object.

A threshold value can then be set to remove all values below a certain range. In a perfect situation, only the values where all the projections agreed would be utilized. However, due to noise a value lower than the maximum can be selected to try and incorporate lost portions of the model. This thresholding process can be seen in Figure 63.

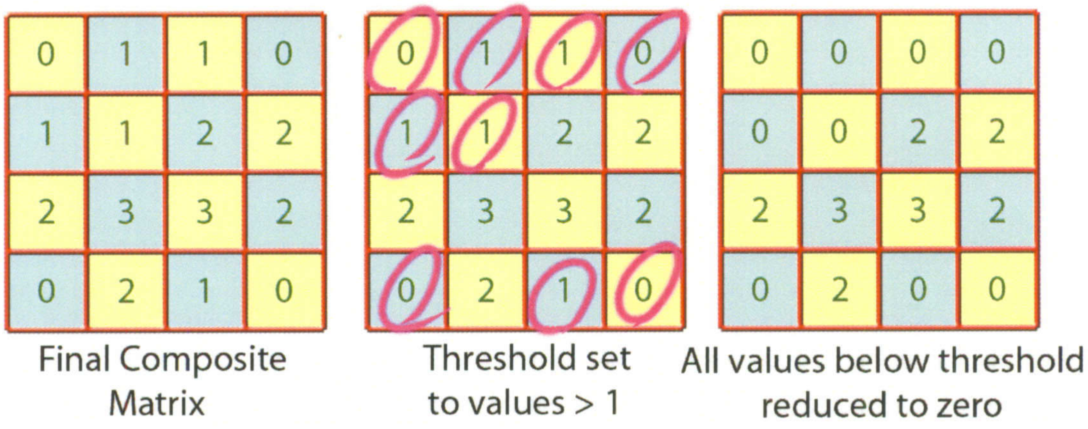


Figure 63: After the final matrix is computed, a threshold is set and anything below that value is set to zero.

The final voxel matrix can be converted into a 3-D model through an iso-surface algorithm.

For testing the algorithms, synthetic data will be produced utilizing three distinct models seen in Figure 64. The sphere will provide a simple example and elaborate on the level of smoothness that the algorithm can reproduce. As shown in Figure 64, at low resolutions curves degrade into distinct blocks creating a stair-step appearance.

The next model, the arrow provides two challenges. The arrow itself is composed of both angular and curved shapes. It also has areas of self-occlusion behind the bend tail and around the arrow head which can cause issues during reconstruction.

The final model to be reconstructed is the humanoid. This is a complex shape with several appendages that can be lost in low resolution images. It also features both concave and convex segments which can pose problems for a reconstruction algorithm that utilizes only the object's silhouette.

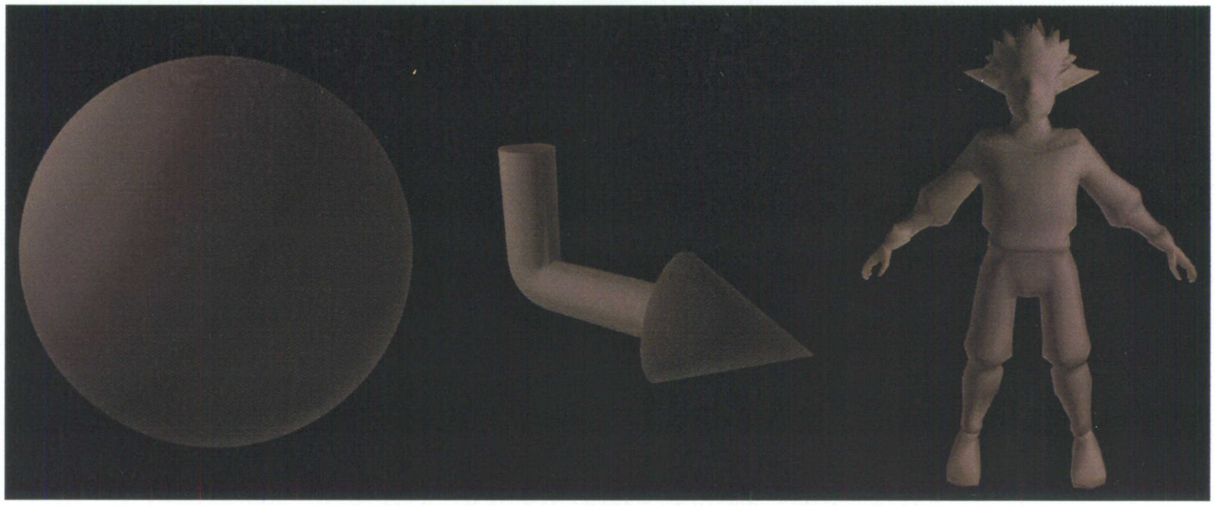


Figure 64: The three models used to create synthesized data.

To validate the reconstructed model, an ellipsoid plot of the first three Fourier descriptors will be created. This method of validation uses an ellipse for each reconstructed model with the center being the mean of the three descriptors with the radius for each dimension radius the variance. These plots of have utilized by Giordano, Barrot and Corriveau as a means of separating different sand particles from one another [5]. Figure 65 shows how the ellipsoid is constructed using the variance and median. Here, we will be using the descriptors to show the refinement of models as both resolution and projections are increased. Since these plots utilize the mean and variance of the descriptors, refinement of the model will be proven by smaller ellipsoids. These plots will also indicate the best combination of resolution and projections. If the ellipsoids share the same median and variance then there would be no benefit to the addition of resolution or projections. A lower resolution or projection reconstruction should be performed to save time.

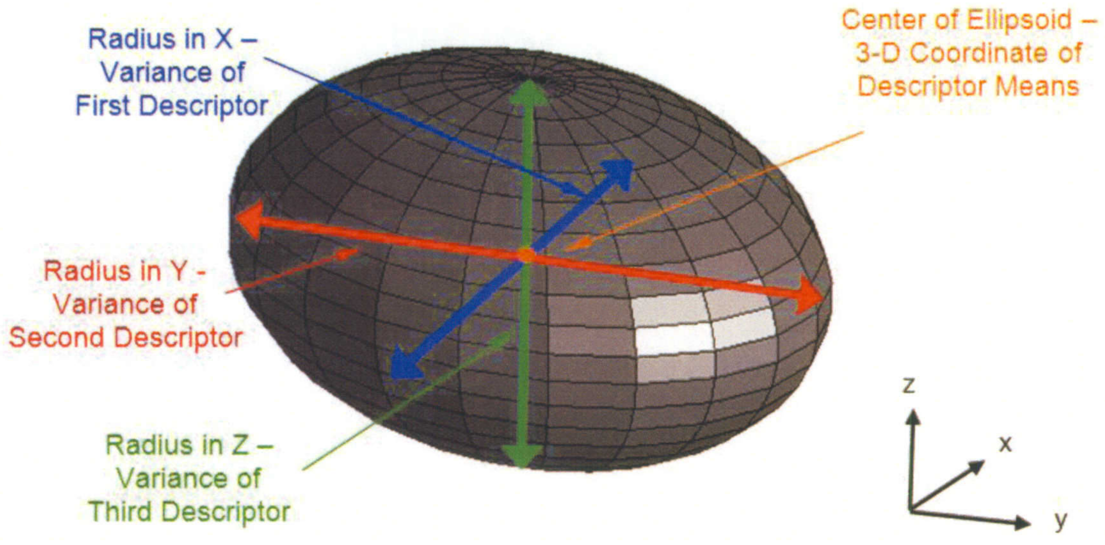


Figure 65: How an ellipsoid plot is constructed. [5]

In addition to the synthetic models, several reconstructions will be performed on optical tomography data and compared to both the X-ray and old reconstruction algorithm. Images for the particle reconstructions will be taken from the Rowan University 3-D Imaging and Modeling Geomaterials Online Database.

CHAPTER 4 : RESULTS

In designing, developing and implementing immersive, navigable and interactive VR environments there is a necessity for a set of procedures to create these worlds. In this thesis, many algorithms were created to deal with various types of data for enhanced scientific visualizations. This chapter will be outlined as follows: data visualizations, optical reconstruction technique visualizations, NASA rocket engine test stand environments and particle synthesis visualizations. Each of these sections will expand upon visualization techniques for varied environments.

4.1 Visualization Algorithms

Depending upon the characteristics of the data provided, a virtual world can display visualizations in numerous ways. However, convenient plotting methods such as graphs, iso-surfaces and the like are not included in the WorldViz Vizard package utilized by Rowan students. Algorithms had to be created to quickly create these visuals that were robust enough to accommodate various types of information. The following list is of the successfully implemented algorithms for data visualization of scalar and vector data.

4.1.1 One Dimensional Scalar Data

One-dimensional data was created in excel and was modeled after a simulation of heat flow through a one-dimensional rod. After several iterations of the algorithm with pre-set boundary conditions the data was exported as a CSV document. This data was then loaded into Vizard inside a list, which is a data structure similar to an array.

In each of the following visualizations, classes were construct that would allow data to be passed to them and the visualization created immediately. This structure will allow for future students to utilize these classes without requiring them to understand the subtly in creating some of the visualizations.

Figure 66 shows the static data visualization complete with x and y axes. Each of these axes can be customized to any desired incrimination.

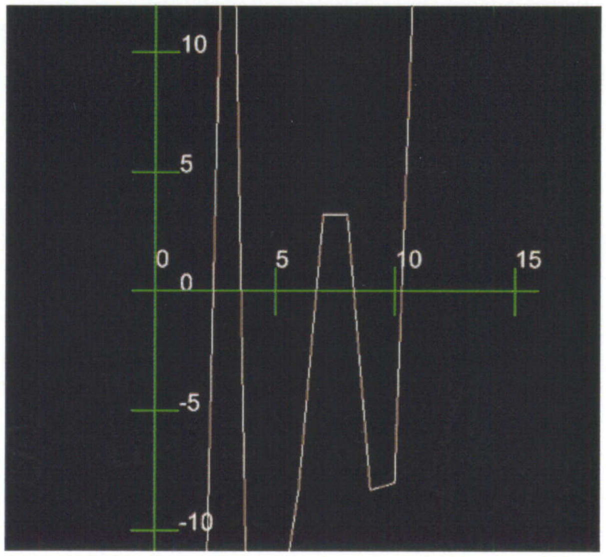


Figure 66: A static graph depicting one dimensional scalar data of heat dissipation.

Figure 67 shows the progression of the graph when a two dimensional array with the rows corresponding to increments in time. The left boundary was held constant at zero while the right boundary fluctuated between positive 5 and -10 degrees.

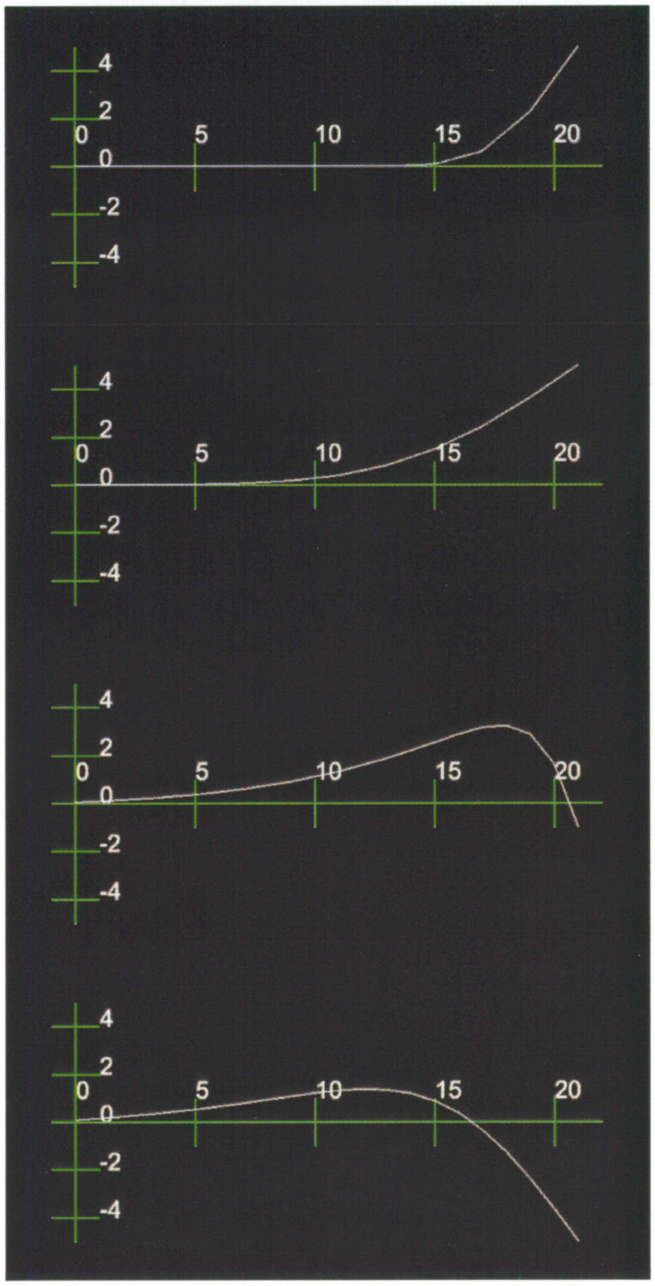


Figure 67: A dynamic graph of temperature data fluctuating at the right boundary.

Figure 68 is of the same data as in Figure 66 but visualized with color instead of height. A gradient from blue to red was created so that the low point of the data would appear blue and the high point red with various shades of purple indicating regions between. Axes are also present along the x direction to indicate the length of the array.

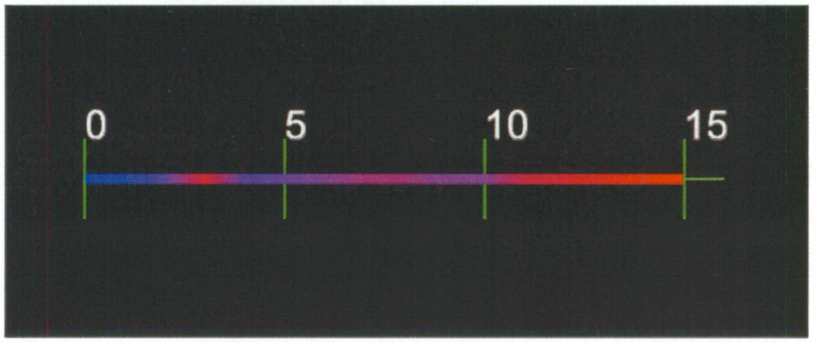
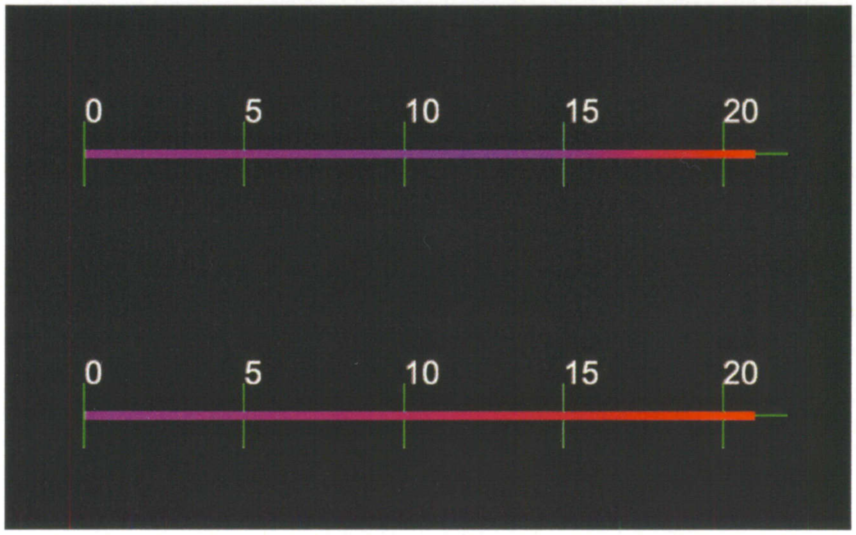


Figure 68: A static color line using the transition between blue and red to show temperature changes.

Figure 69 utilizes the color visualization of a line graph for a series of datasets with the rows corresponding to increments in time. In this figure we can see the right side being heated at +5 degrees for a period of time and then cooled to -10 degrees and finally returned to +5 degrees.



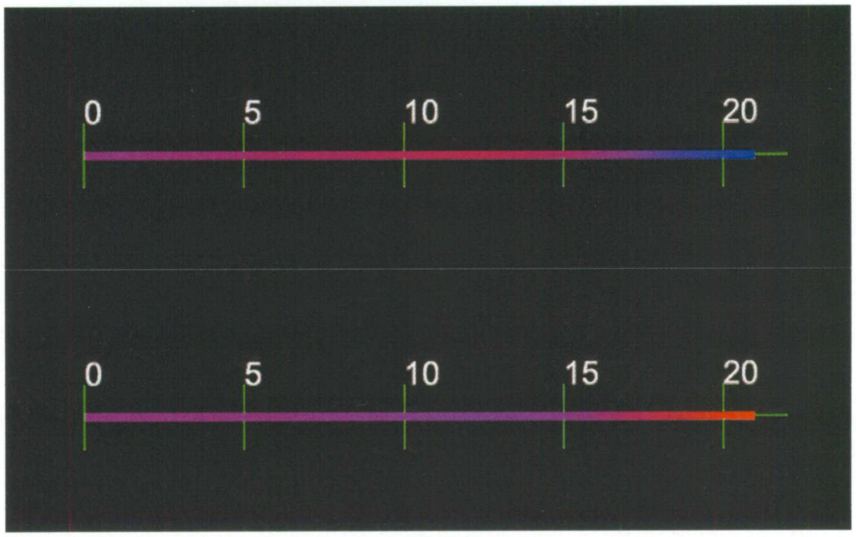
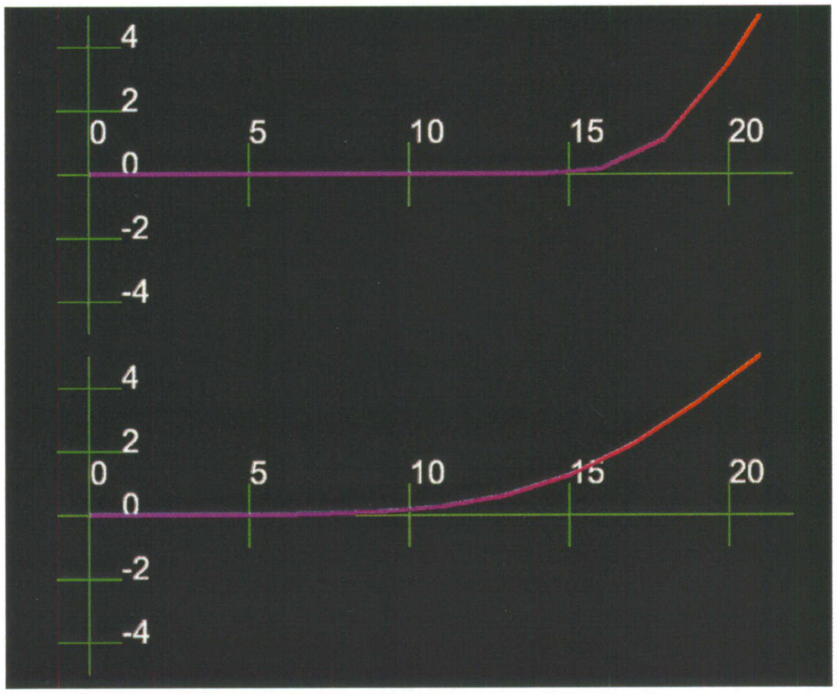


Figure 69: A dynamic color line depicting changing boundary conditions through changes between red and blue.

Figure 70 shows a visualization that incorporates both color and height to indicate temperature on time-varying data. This is the most convincing of the line graph visualizations as the color immediately indicates the high and low points while the height and axes impart the exact temperatures of the data.



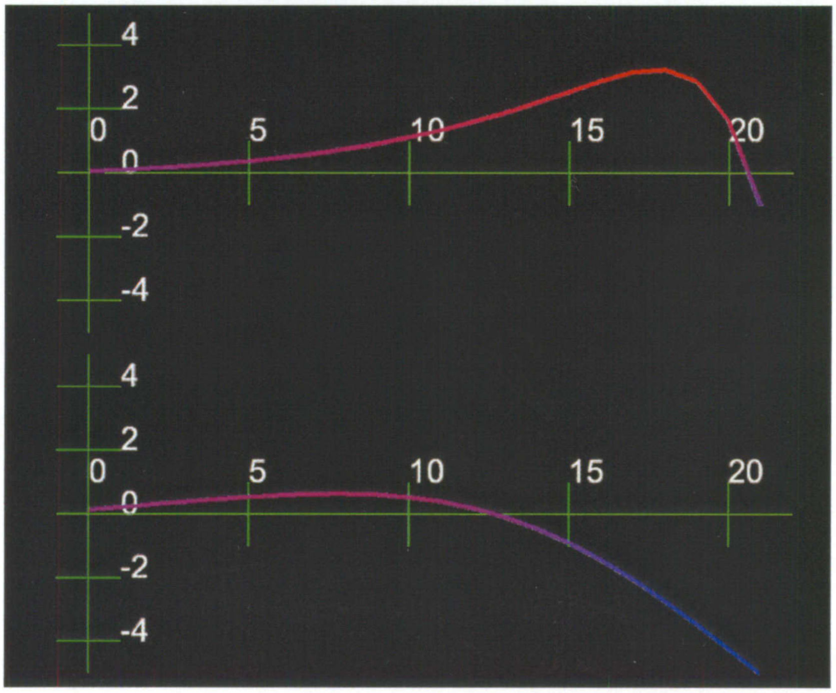


Figure 70: A combination of both height and color to depict the change in temperature of one-dimensional bar.

While it could be reasoned that utilizing two visualizations for the same dataset would be slightly redundant, in this situation the colors help to punctuate the highs and lows. This punctuation makes it an easy task to immediately identify if an area is a local minima/maxima since it would still be colored an intermediate shade of purple.

4.1.2 Two Dimensional Scalar Data

Two dimensional scalar data was created inside of excel and utilized two-dimensional heat flow algorithms with boundaries held at constant temperatures. The left, right, top and bottom temperatures were held at a constant +5, -5, -2, and 0 values, respectively. Figure 71 shows the heat flow data after several iterations on a planar surface with the y positions of the vertices mapped to the scalar heat value.

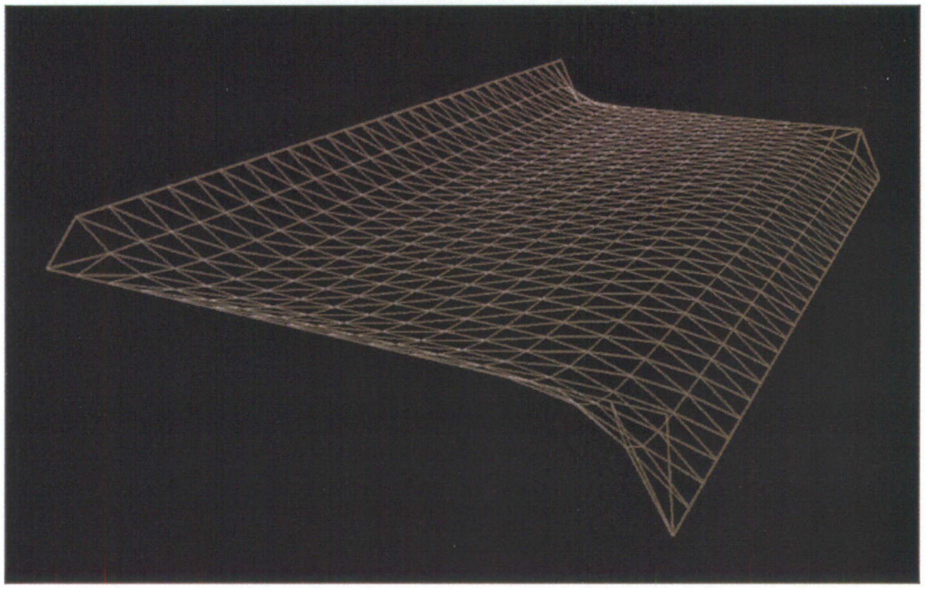


Figure 71: A static plane depicting temperature and heat dissipation across a 2-D surface through height.

Figure 72 depicts a time series of data as the heat flow algorithm progresses through several iterations. In the top image the heat boundaries are clearly seen on the edges of the plane. As time progresses the heat dissipates out to the plane and the surface begins to smoothen.

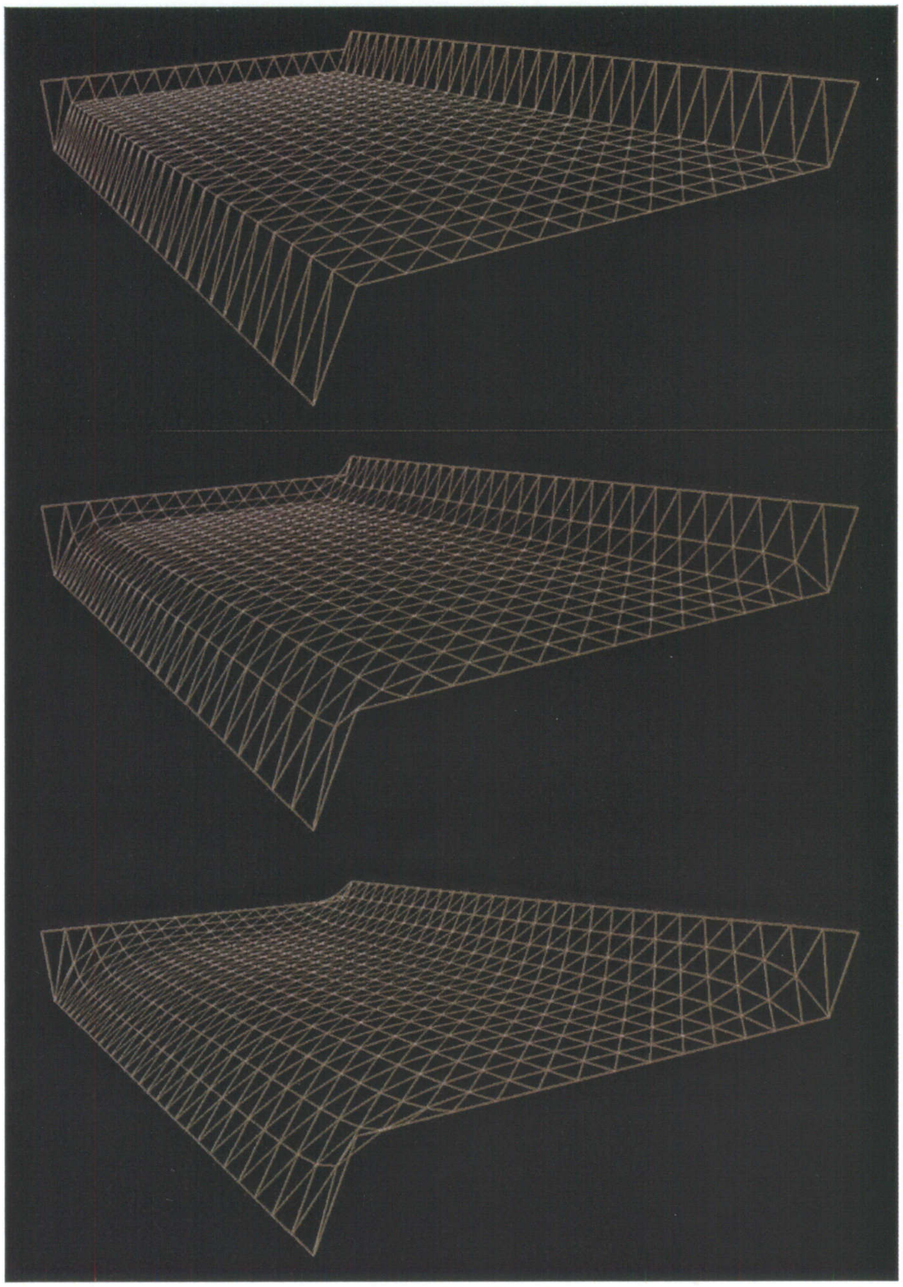


Figure 72: A dynamic heat plot of a 2-D surface depicting the propagation of heat from the boundaries.

Figure 73 utilizes color rather than height to show the propagation of heat across the 2-D surface. Red and blue were chosen as the gradient and values of purple show the mid tones.

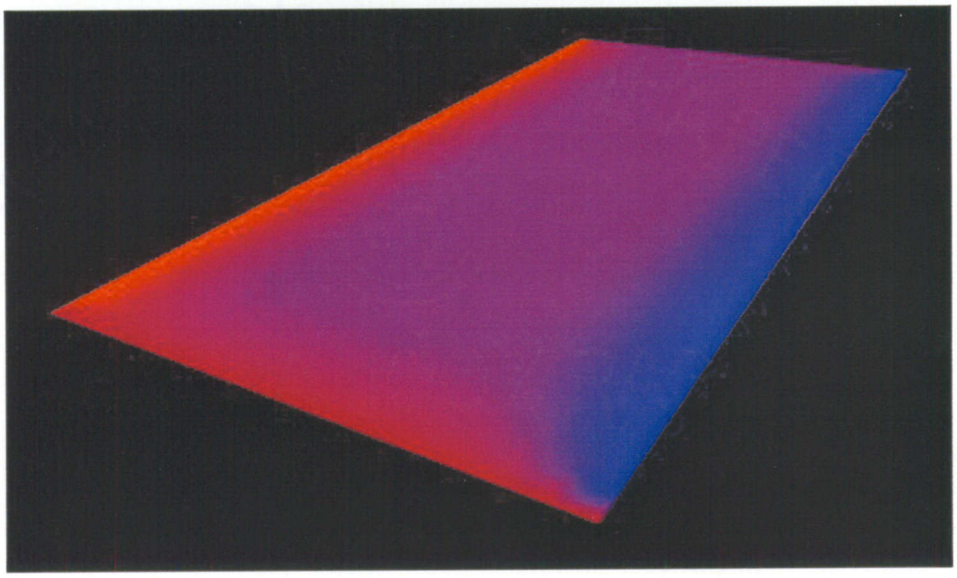


Figure 73: A static color plot of heat across a 2-D plane.

Figure 74 shows increments in time of the color plane visualization. The boundaries appear sharp and distinct from the purple center region. As time progresses the boundary colors begin to diffuse across the surface.

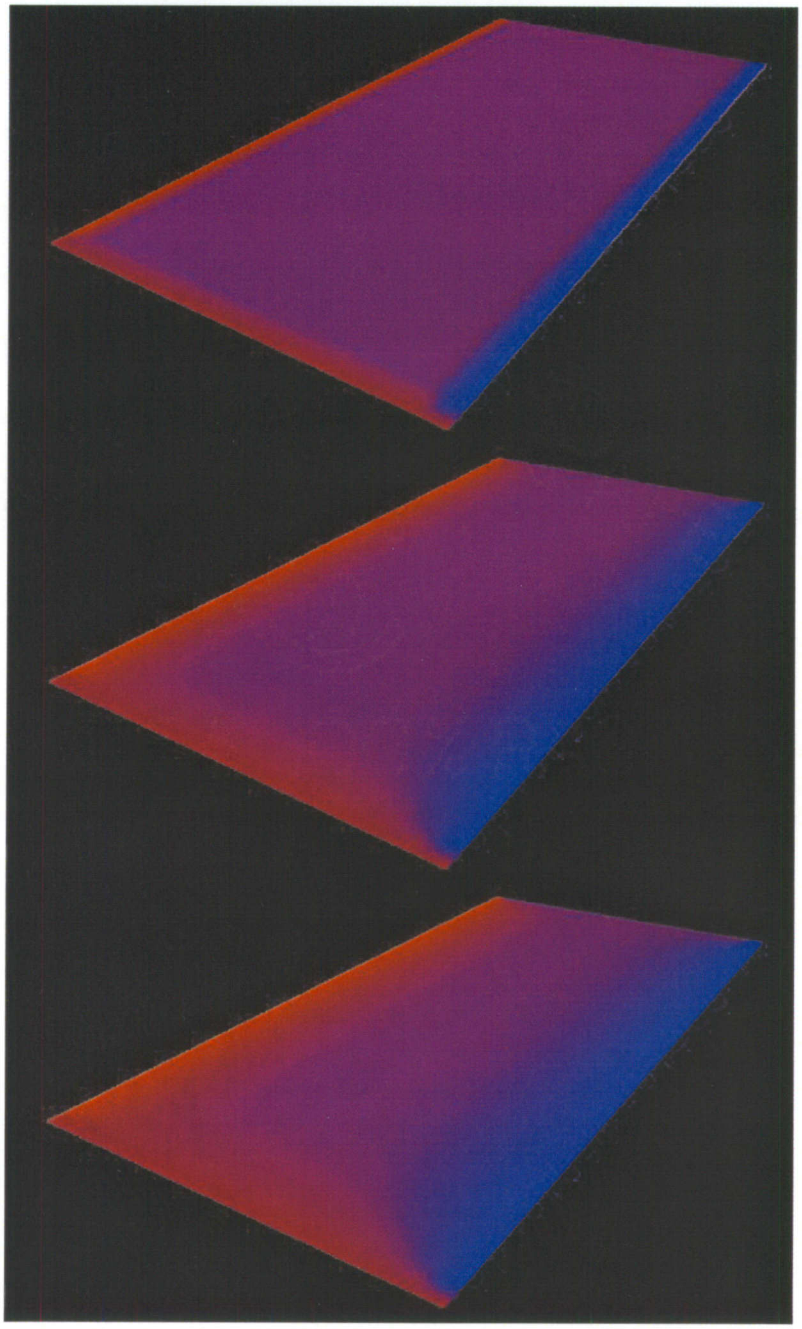


Figure 74: A dynamic plot using color for a 2-D surface and the propagation of heat across it from set boundaries.

Figure 75 combines both the height and color visualization to depict the surface. The top image is of the surface with no light calculations because vertex normals have not been assigned. The bottom image is of a wireframe version which clearly delineates the height of the surface.

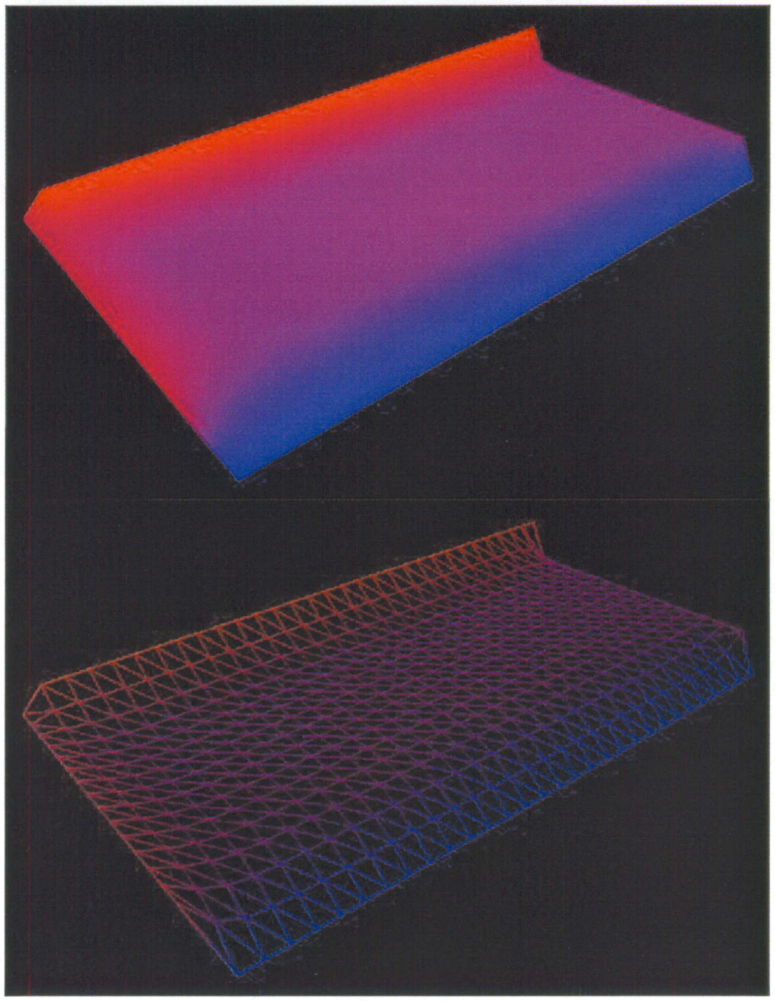


Figure 75: A static plot using color and height to show the propagation of heat across the surface from set boundaries.

Figure 76 is the time stepped version of the previously shown static planar plot. Here we can see the dissipation of color from the boundaries to the inner surface as well as the smoothing of the plane.

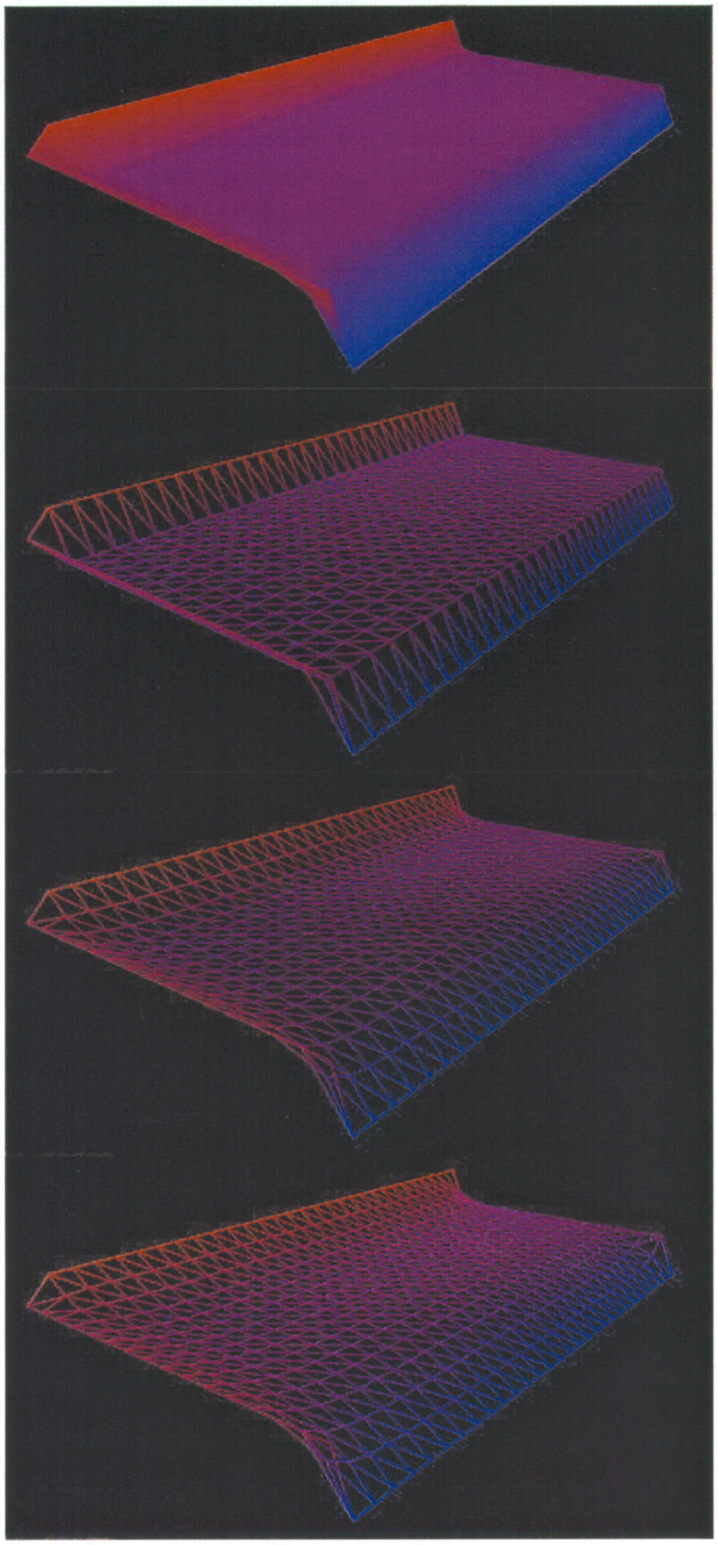


Figure 76: Dynamic 2-D visualization using height and color to indicate the movement of temperature through the plane.

4.1.3 Voxel Plots

Voxels are volumetric pixels which represent a scalar value over a location. Voxels are typically not visualized due to the fact that attempting to view all elements of a 3-D matrix can be memory intensive. However, many times the ability to view information stored as voxels is important especially when conversion to iso-surfaces is not feasible.

Figure 77 is a voxel plot where each element of the array is depicted with a single point. This visualization is difficult to understand, and only when the view is rotated is it possible for the user to register depth. It is a very fast and simple method of displaying voxel information.

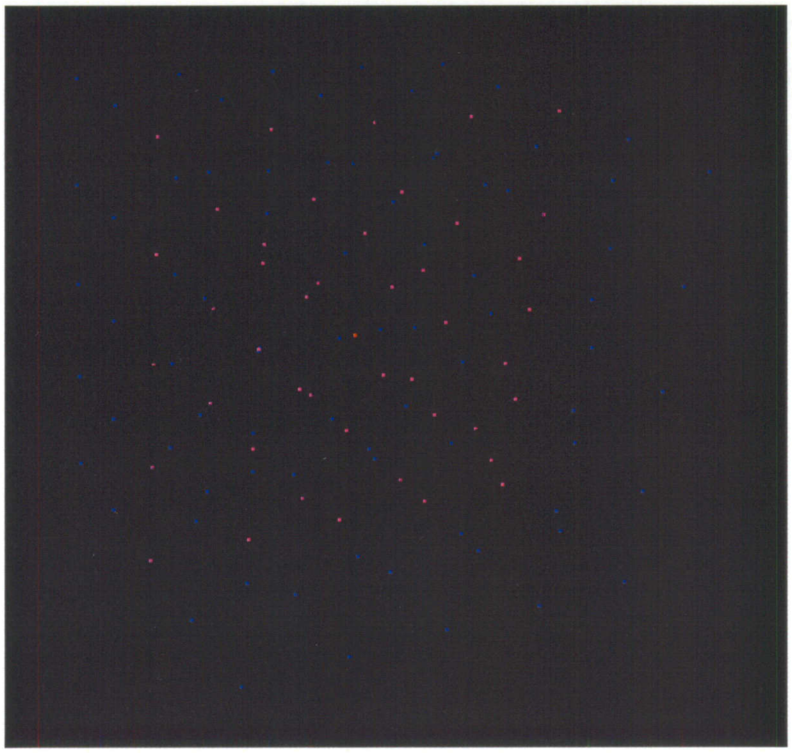


Figure 77: Voxel visualization with a point size set to 1.

OpenGL provides the ability to set the size of the points. Figure 78 is the point visualization with a size set to 20. As the user zooms out from the voxel array the points being to occlude one another and provides a good surface view of the object. When the user moves toward the voxel array, the particles separate apart and an inner view of the array is possible. However, since these are still points there is no way to understand depth information other than through overlapping elements and rotating the view.

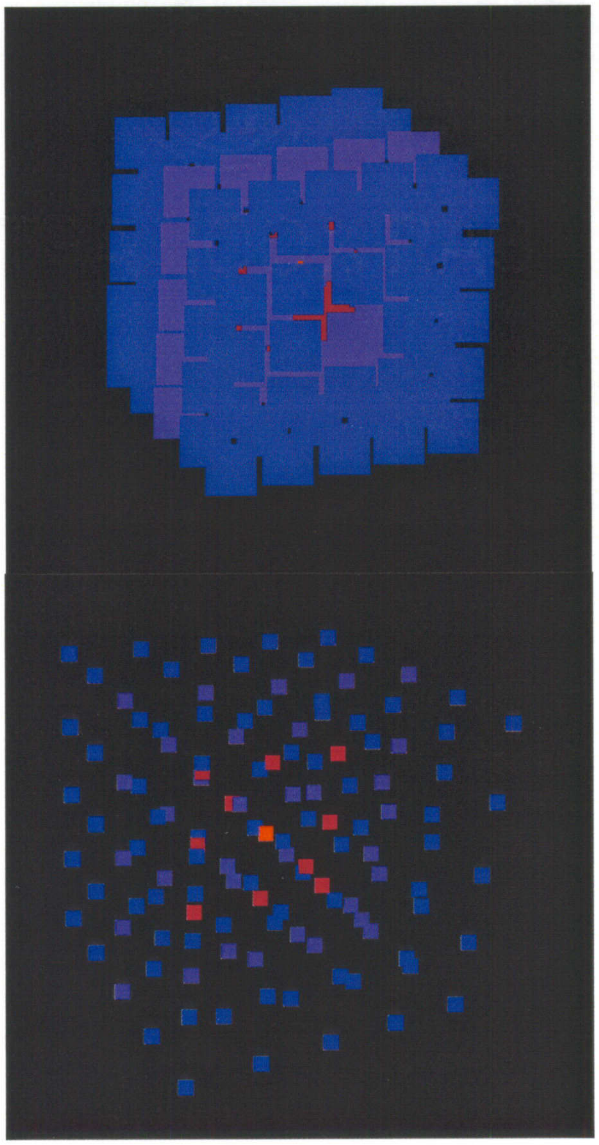


Figure 78: Voxel visualization with a point size set to 20, while far from voxels and close to them.

Figure 79 is another voxel visualization technique where the values are mapped to planes. This allows the user to peer inside the surface and analyze it in a cross-sectional fashion.

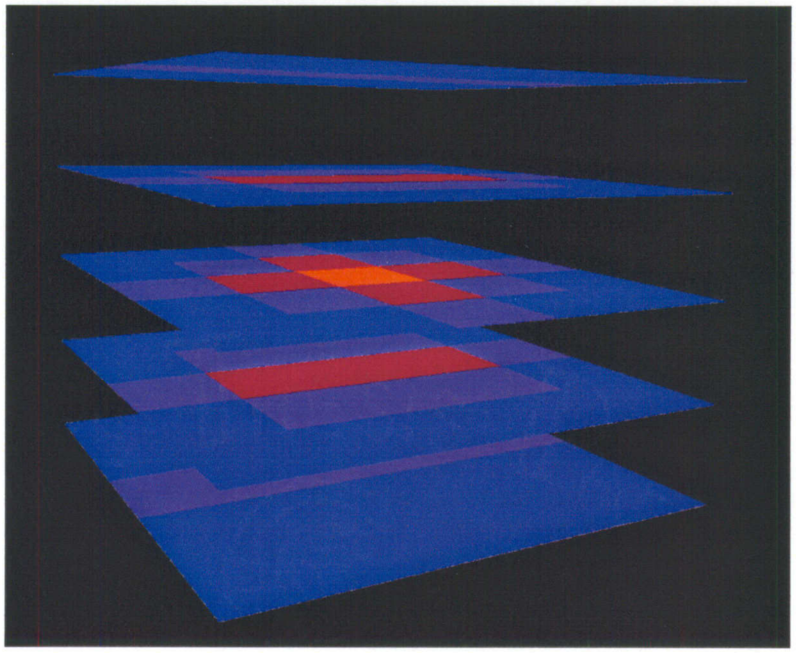


Figure 79: Plane voxel visualization.

Figure 80 is a voxel visualization utilizing transparent cubes. The level of transparency has a direct effect over how easily the user can peer into the surface. However, due to some conflicts with the algorithm that creates the surfaces, some transparencies fail to work. This is noticeable in the left side of the cube in Figure 80.

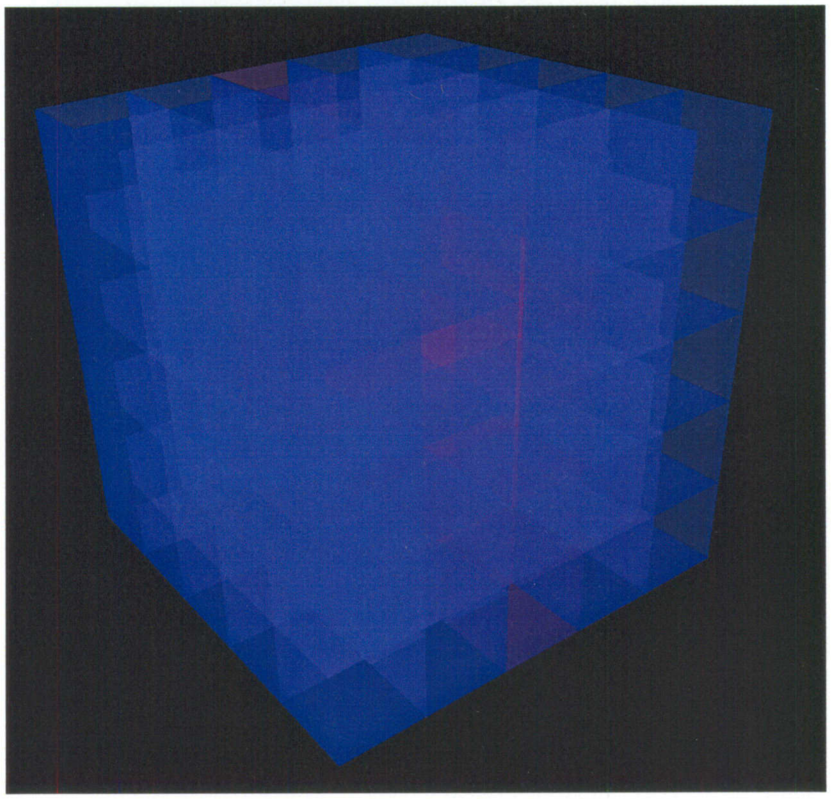


Figure 80: Cube visualization utilizing transparency.

Figure 81 is the last voxel visualization, where a combination of the cubes and points are meant to provide the ability to look internally while keeping depth perspective.

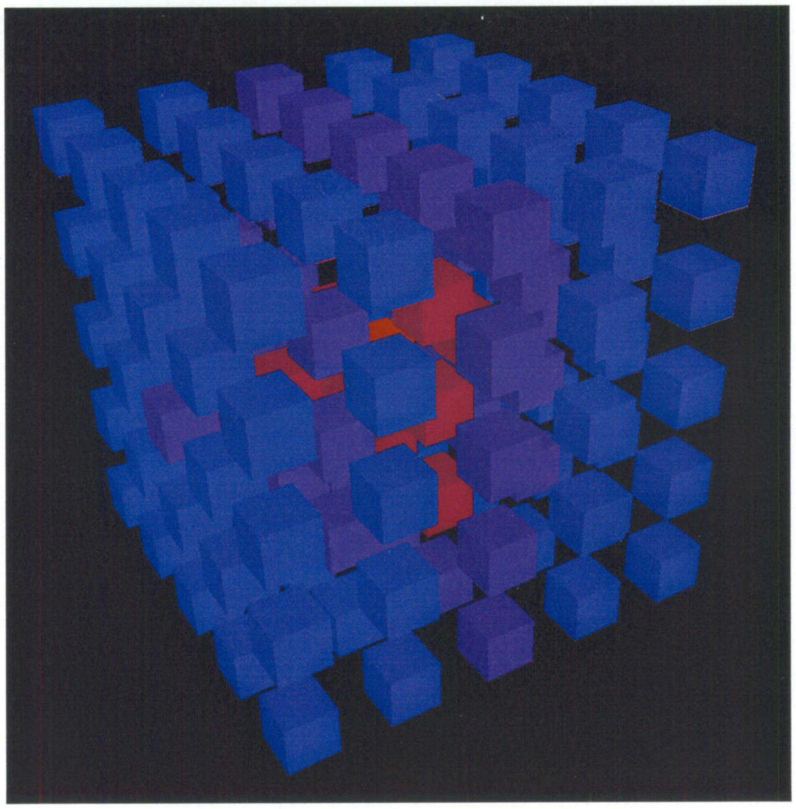


Figure 81: Miniature cube voxel visualization.

4.2 Vector Data

Vector data contains two elements that can be visualized: direction and magnitude. The visualization employed here to view vector data is the hedgehog plot, which is simple to incorporate and is not a memory intensive process.

4.2.1 Hedgehog Plots

Figure 82 is a hedgehog plot that only conveys direction. The magnitudes are normalized and the final outcome is a series of lines bending toward the vector direction. Sinusoidal data was utilized to create this plot as well as to create the rolling motion. In order to impart the start and end of the lines, the first is colored black and the point that corresponds to the vector is colored white. OpenGL fills in a gradient between the two points giving the feeling of motion blur or a comet's tail, which gives the user a sense of direction.

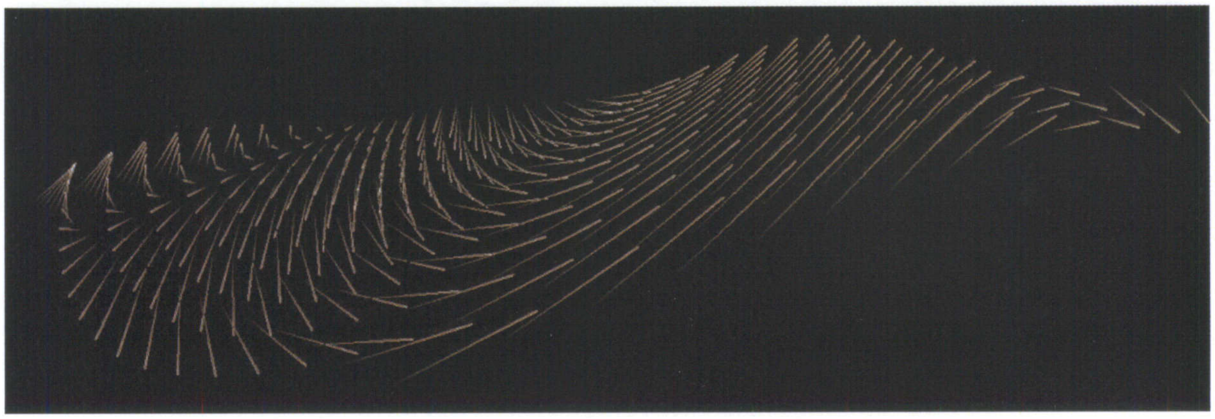


Figure 82: Hedgehog plot only indicating directional flow with all magnitudes normalized.

Figure 83 utilizes color to depict magnitude. As previously shown, a standard gradation between blue and red was utilized. All the data was normalized so that the each line has a length of one.

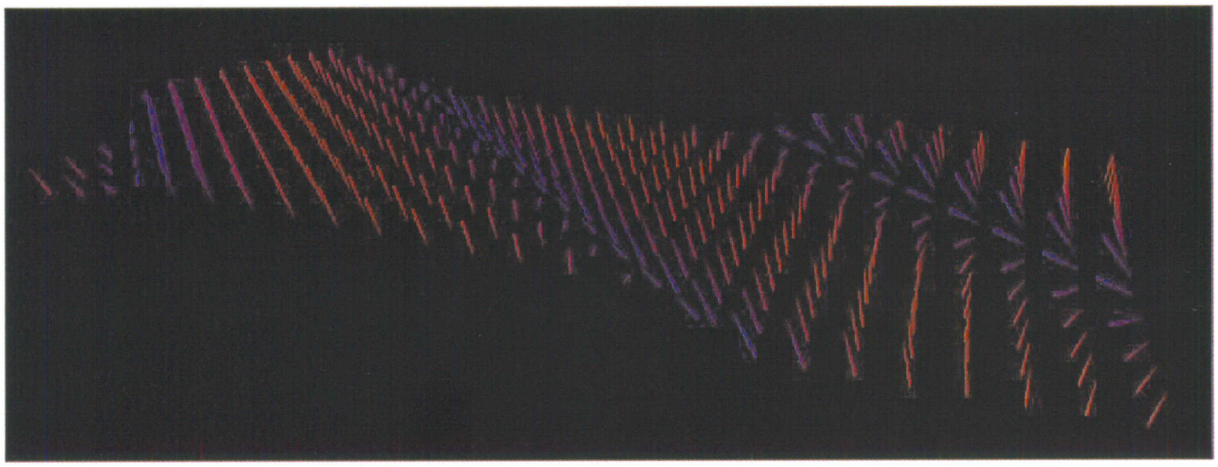


Figure 83: Hedgehog plot of static data utilizing color as an indication of magnitude.

Figure 84 departs from normalization and uses the actual vector to plot the second point. This can lead to problems when the data has very large vectors. These plots can overlap and since there is little depth information it can be tricky to separate elements.

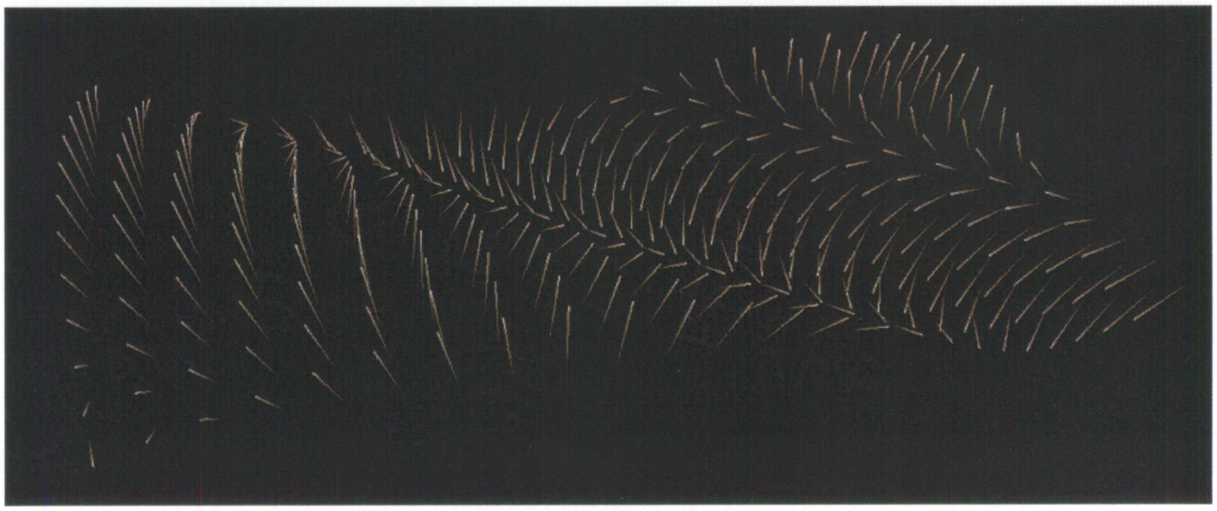


Figure 84: Hedgehog plot of static data utilizing line length as an indication of magnitude.

Figure 85 utilized both line length and color to show the magnitude of each vector.

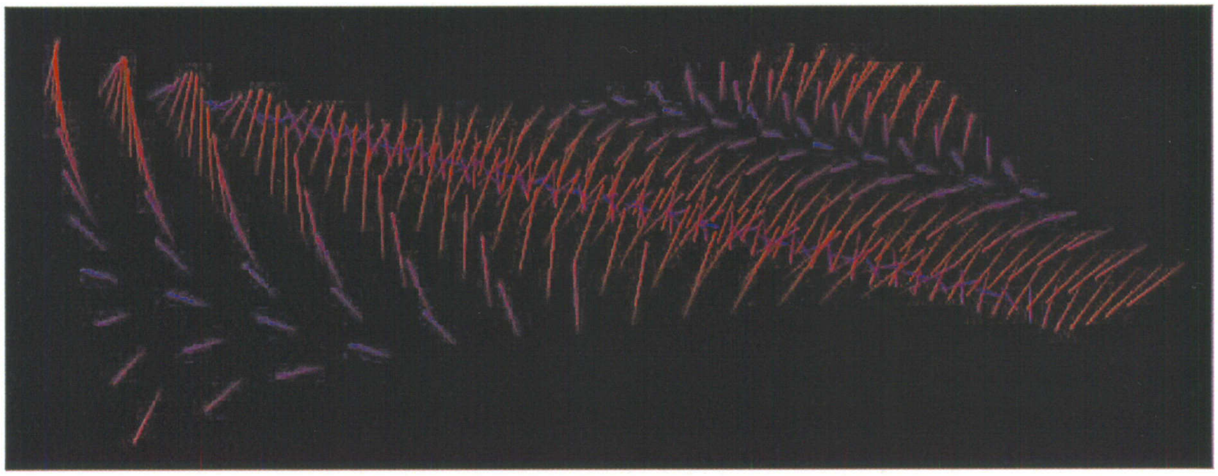


Figure 85: Hedgehog plot of static data with color and length depicting magnitude information.

The previous sets of figures were of static data. The following figures utilize sets of data varying over time to create a dynamic hedgehog plot. Figure 86 is a normalized vector plot that just depicts direction. Here a simple dataset of 5x5 was utilized due to the magnitude of data that has to be generated when dealing with two spatial dimensions and one time dimension for each of the x, y and z directions.

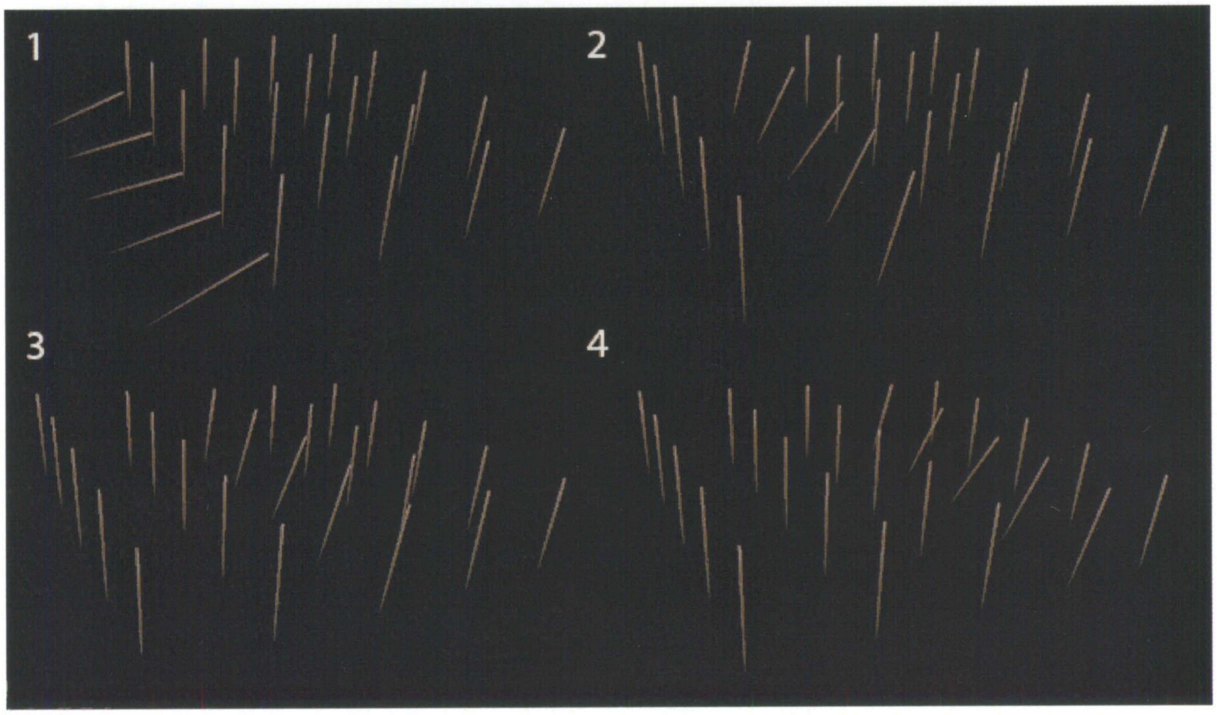


Figure 86: A hedgehog plot of data moving through a field, notice the lack of magnitude information. Only the vector direction is retained.

Figure 87 utilizes color to depict the magnitude.

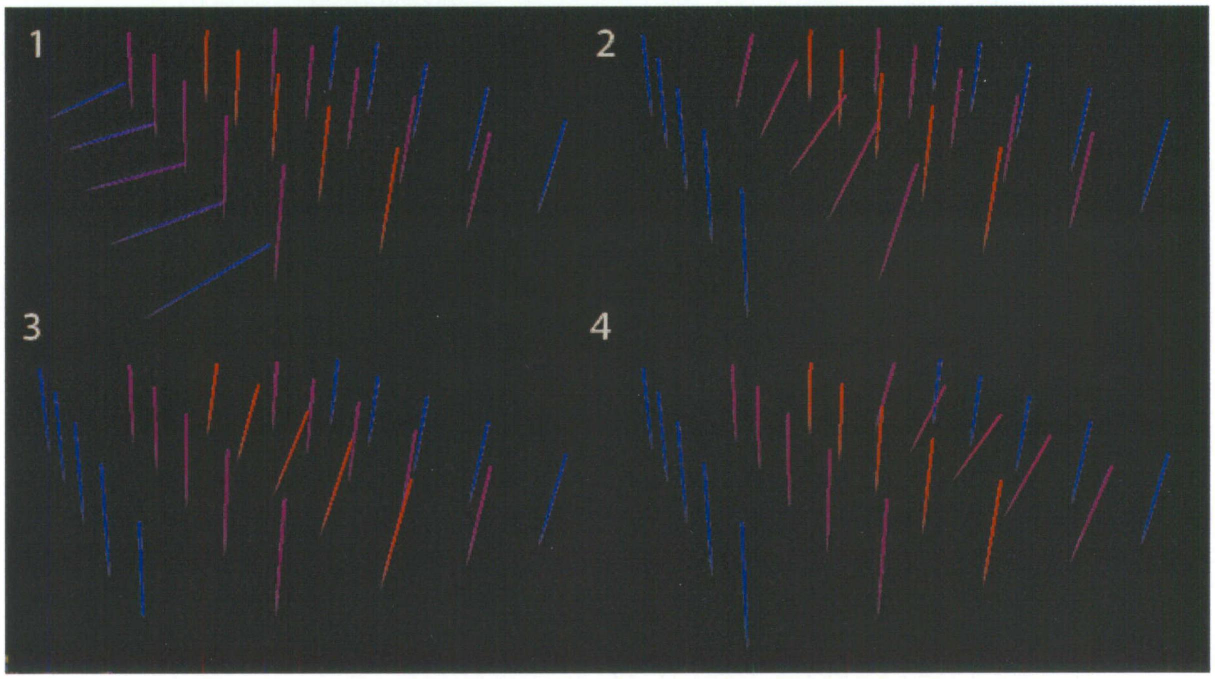


Figure 87: Hedgehog plot utilizing color to indicate the magnitude of a vector.

Figure 88 no longer normalizes the vector and uses the actual lengths to show magnitude. Here it is easy to see that the y directions have a set magnitude throughout the hedgehog plot. As time progresses an x component vector moves from left to right.

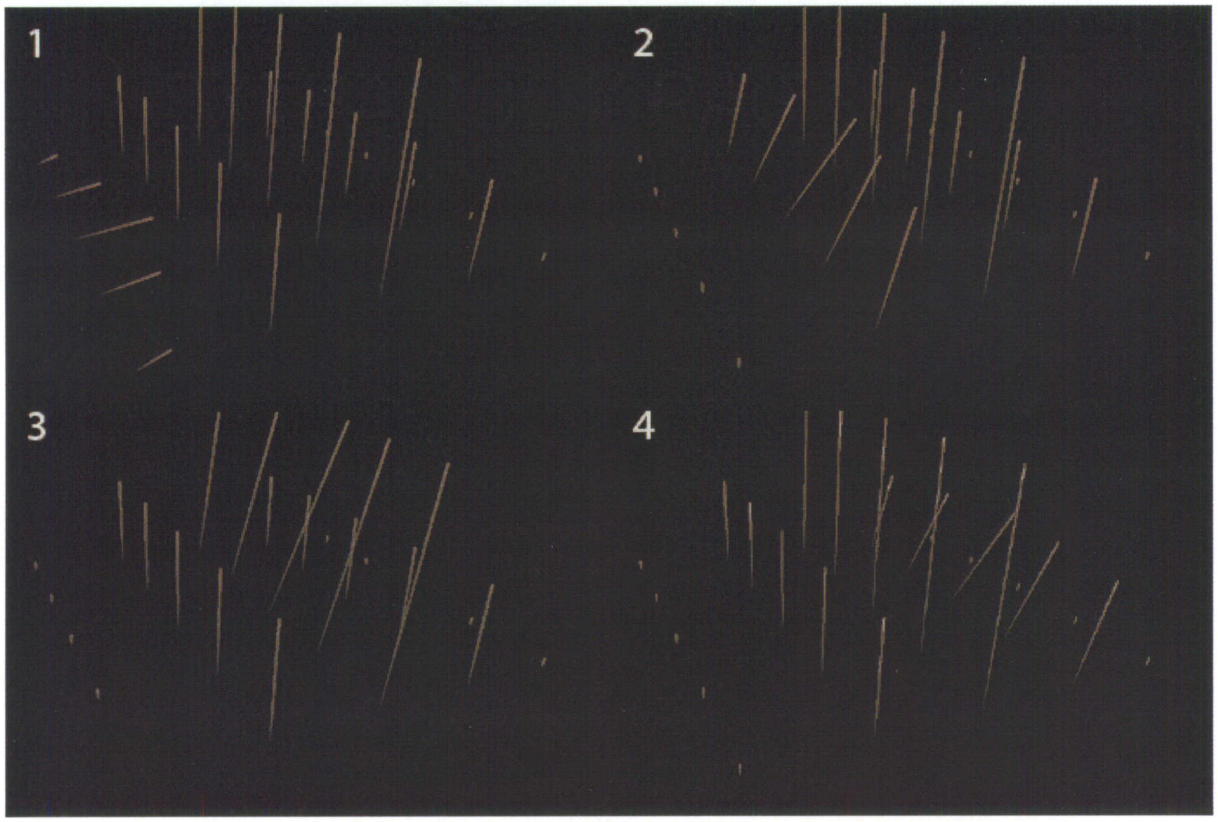


Figure 88: Hedgehog plot of data that has not been normalized, using the vector length to indicate magnitude.

The final hedgehog plot in Figure 89 utilizes both color and line length to show movement thorough the array. The overlapping of color and line length helps to convey the most amount of information to the user. At times when line length is not conveying enough information, the color can provide that information to the user. This is very useful in situations where the user navigates to a point where the lines are seen straight on and they lose all depth.

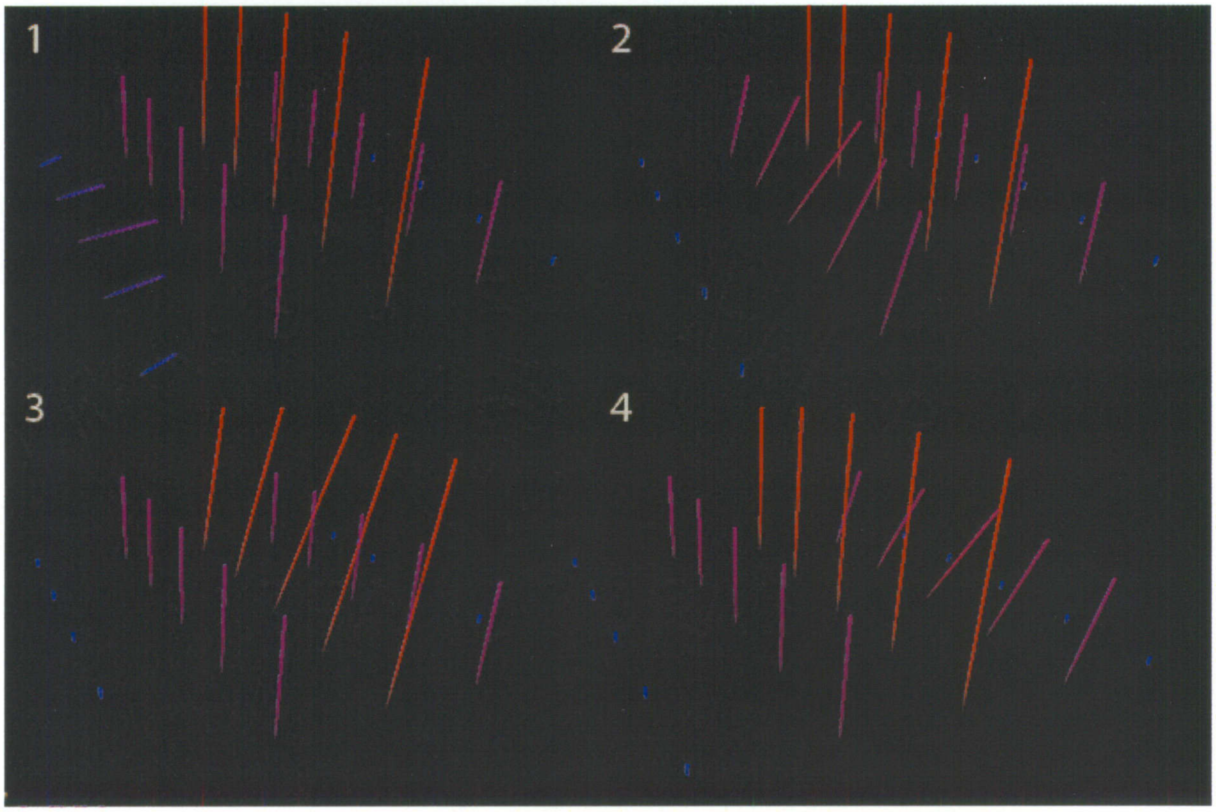


Figure 89: Hedgehog plot of data that has not been normalized, using the vector length and color to indicate magnitude.

This section described several implemented techniques to solve generic data visualizations problems. In the next section, the data will change to disparate points throughout a structure, which possess a number of visualization challenges and require novel techniques.

4.3 NASA Rocket Engine Test Stand Visualizations

The following demonstrations are provided to show different methods of data visualization for types of point data, which can be referred to as 'icons'. Several versions of each project are shown along with the evolution of some visualizations and the removal of those that proved inadequate. The first set of VR environments are for the visualization of rocket engine test stands at Stennis Space Center, Mississippi. The second set of VR environments are for comparing 3-D reconstructions of x-ray and tomographic models of sand particles.

4.3.1 NASA E1

In this first VR environment, a representation of the E-1 test stand was created. The E-1 test stand is designed for testing rocket components that require high pressure capabilities in the range of 15kpsi. Figure 90 shows the completed world along with the GUI to help control and navigate the scene.

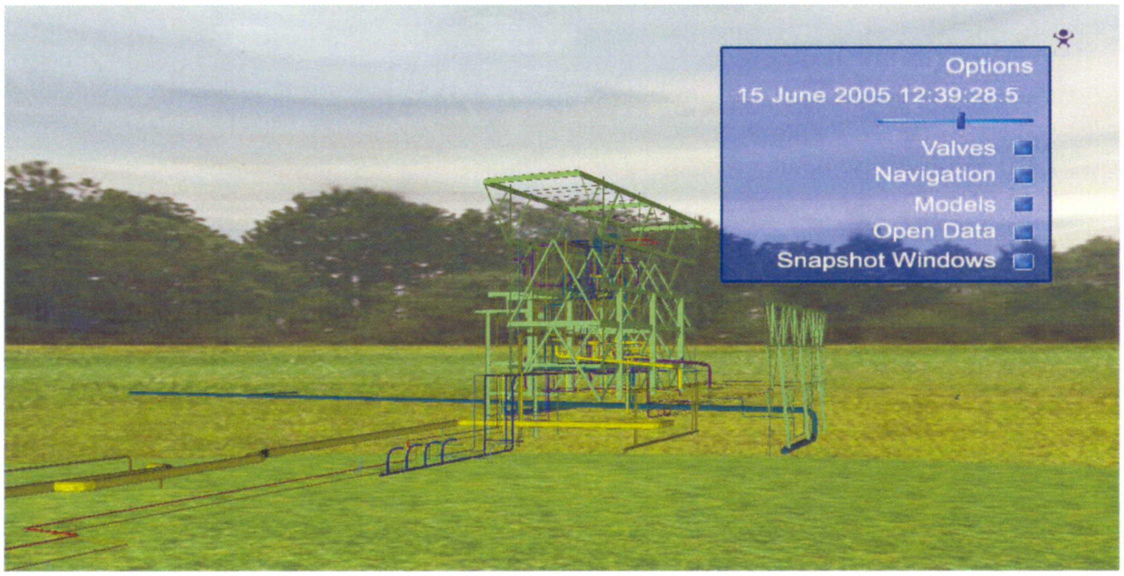


Figure 90: The E-1 test stand and the virtual environment constructed to immerse the user.

The data used to create the graphical objects was from a drafted Pro Engineer file provided by Stennis Space Center. This file format utilizes mathematical representations of geometric objects to create a 3-D model. This form of rendering is not directly supported by WorldViz Vizard and had to be converted to a vertex based format. These files were opened in Solidworks and exported out as Wavefront .obj files. The process of exporting these files ran into several problems. The final output provided a single model but with many missing elements. In addition, three sets of each model were overlain on top of one another of low, medium and high resolution. Figure 91 shows floating pipes and missing segments of the model.

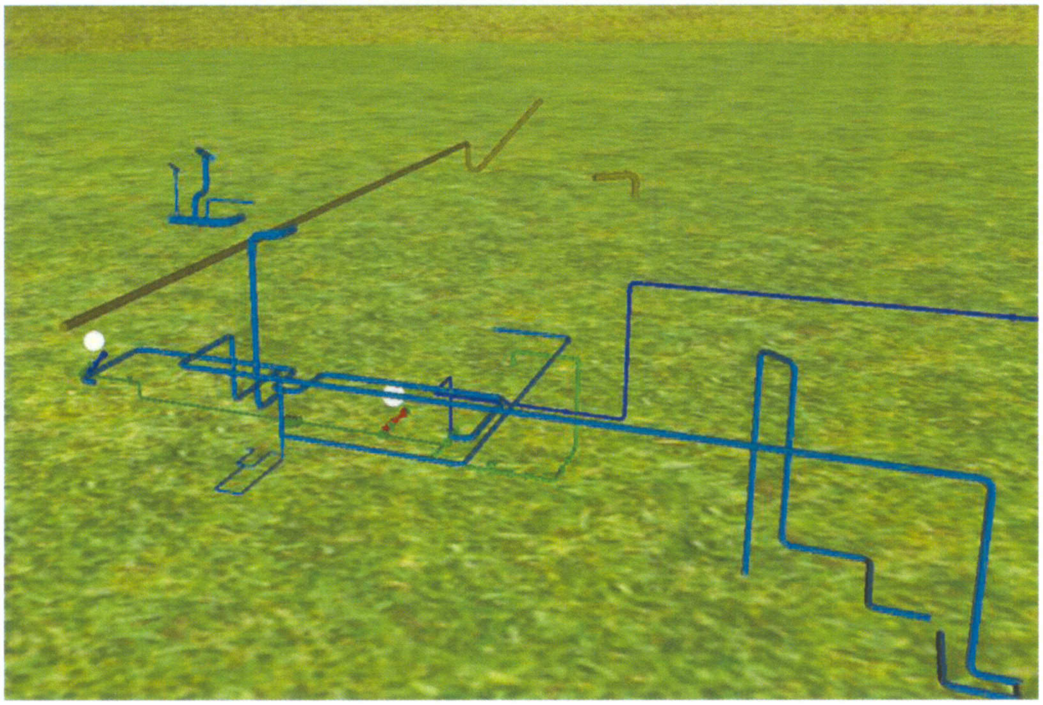


Figure 91: Missing models and broken pipelines were a result of data loss when changing model formats.

A second with this procedure for creating the graphical models was that the objects took on random colors. These colors had no purpose other than distinguishing objects from one another and severely reduced the realism of the scene. Figure 92 shows the inside of the installation and the many different models of various colors.

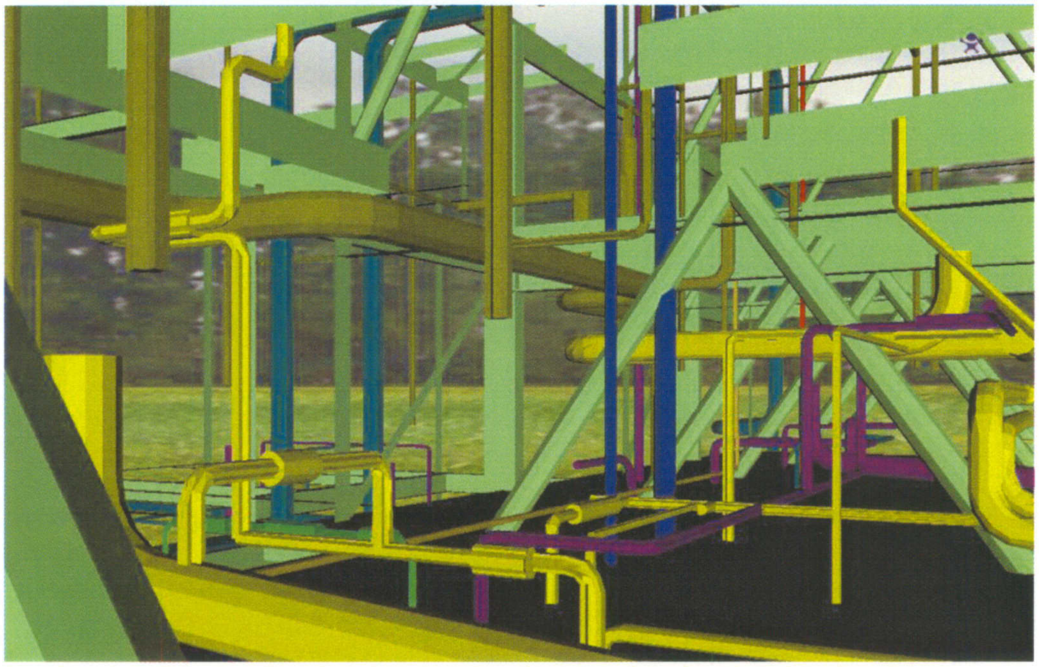


Figure 92: Random model colors were prevalent when importing models from software packages.

The final issue with importing the data was that in some instances the model would become degraded. At complex interfaces between curved objects such as bends and intersection the geometry would become jagged and hardly resemble the smooth pipeline. Figure 93 shows two examples of poor geometry chosen to represent an intersection of two pipelines and a ninety-degree elbow bend. In both instances the Solidworks exporter failed to properly construct a geometric surface that described the intersections.

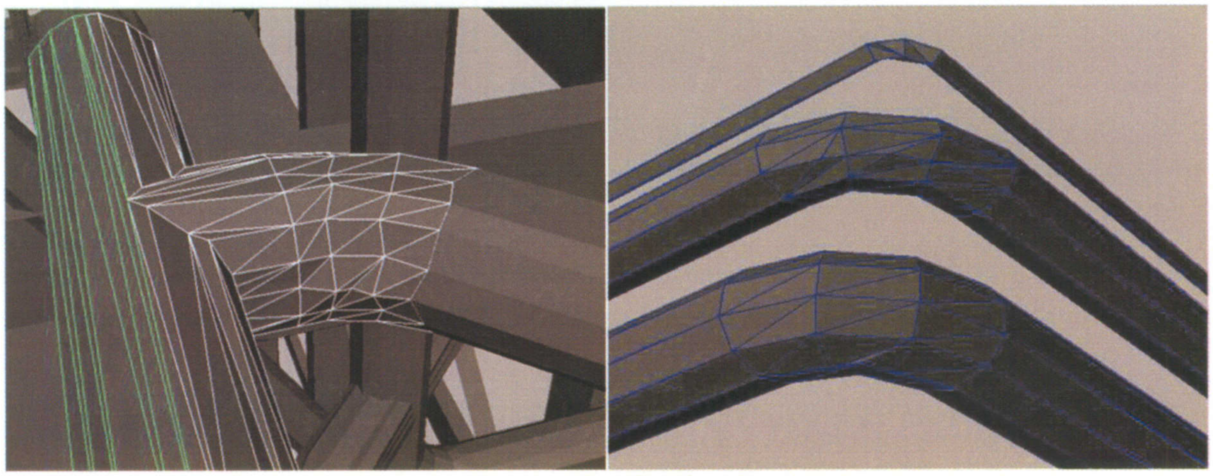


Figure 93: Due to data loss with exporting and converting files, many areas of the model had unacceptable geometric representations of the surfaces.

In order to create data visualizations when no data was provided, information was synthesized to show how it could appear. Three kinds of data were simulated: temperature, pressure and valve state. The data was generated inside of excel and exported as a series of CSV files.

Temperature and pressure can be seen in Figure 94 at a valve. The valve's color would change from blue to red depending on the preset temperature maximum and minimum values. Pressure was indicated by rocking the valve back and forth. As pressure would increase the valve would shake more violently and draw the user's attention. This visualization was ultimately dropped due to poor user response and subjective viewing as to what was 'violent' shaking.

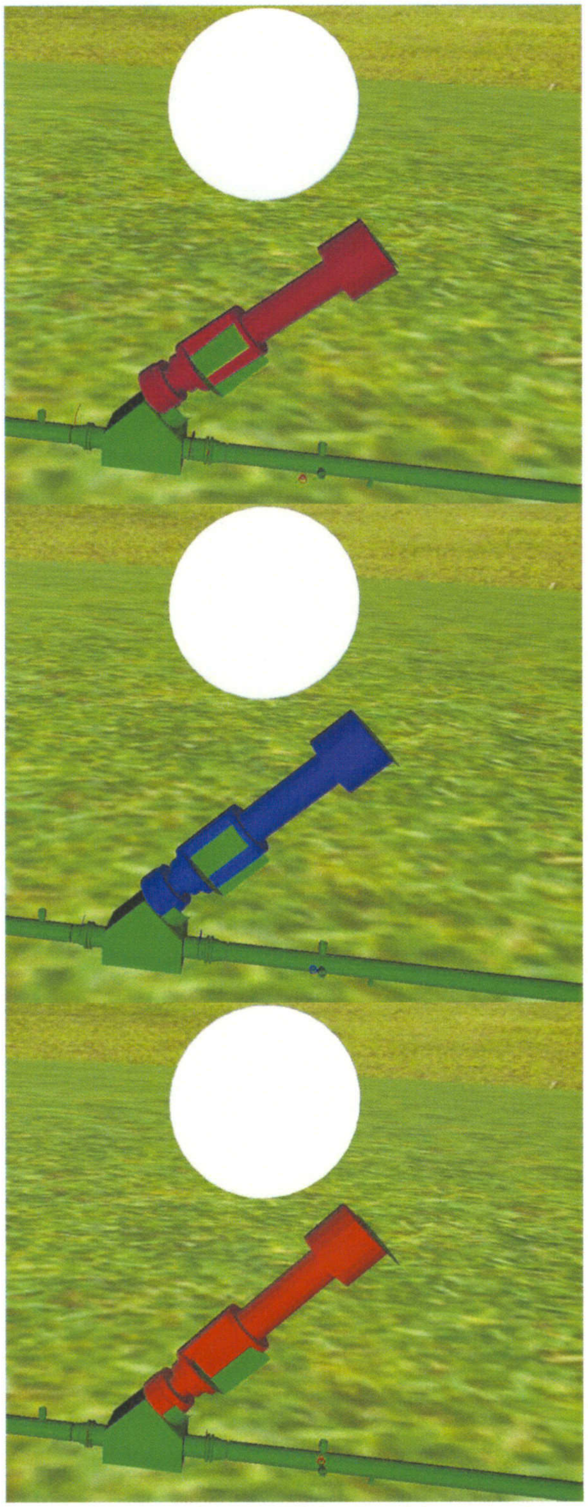
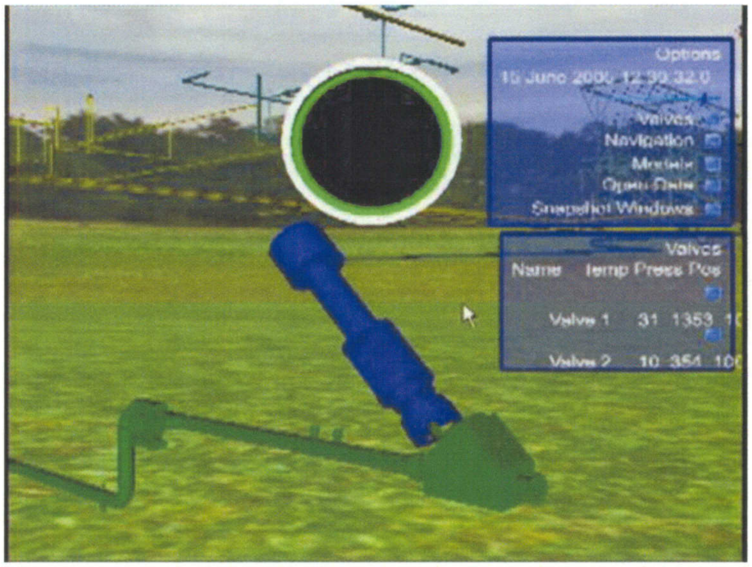


Figure 94: Color changes from blue to red indicate temperature while pressure was indicated by shaking the valve.

Valve state was visualized with a sphere-in-a-sphere visualization. This visualization contained three components, the desired value of the valve in green, the actual state of the valve in black and the maximum opened state in white. The black and green spheres would shrink and expand as the valve was opened and closed. This sphere visualization was possible by inverting the normals of the spheres so that the user would see through the front of the visualization and into the back of each internal model as shown in Figure 95.



Name	Temp	Press	Pos
Valve 1	31	1353	1
Valve 2	10	354	10

Figure 95: The pressure was also displayed by the three spheres above the valve.

A graphical user interface (GUI) was created to provide the user with several options for customizing the VR environment. Each of the fly-out menus could be minimized and maximized by clicking on the purple Vizard icon in the left corners of the screen as shown in Figure 96.

The user was presented with the time of the testing as well as a scroll bar which depicted the current time of the test data. The ability to access submenus for the valves and opening data were provided but never made functional. Navigation links were provided to quickly move between valves; however this resulted in poor performance when moving the viewpoint and would sometimes make user navigation impossible. An option for turning off different sections of the model was provided to speed up performance or highlight areas of importance. Finally, the ability to create snap-shot videos that captured the current location of the user and placed them at the top of the screen was introduced. This allowed the operator to set cameras to watch important areas and then continue to explore. Performance issues restricted the number of snap-shot windows to only three at a time before the frame rate became too low (under 10 frames per second).

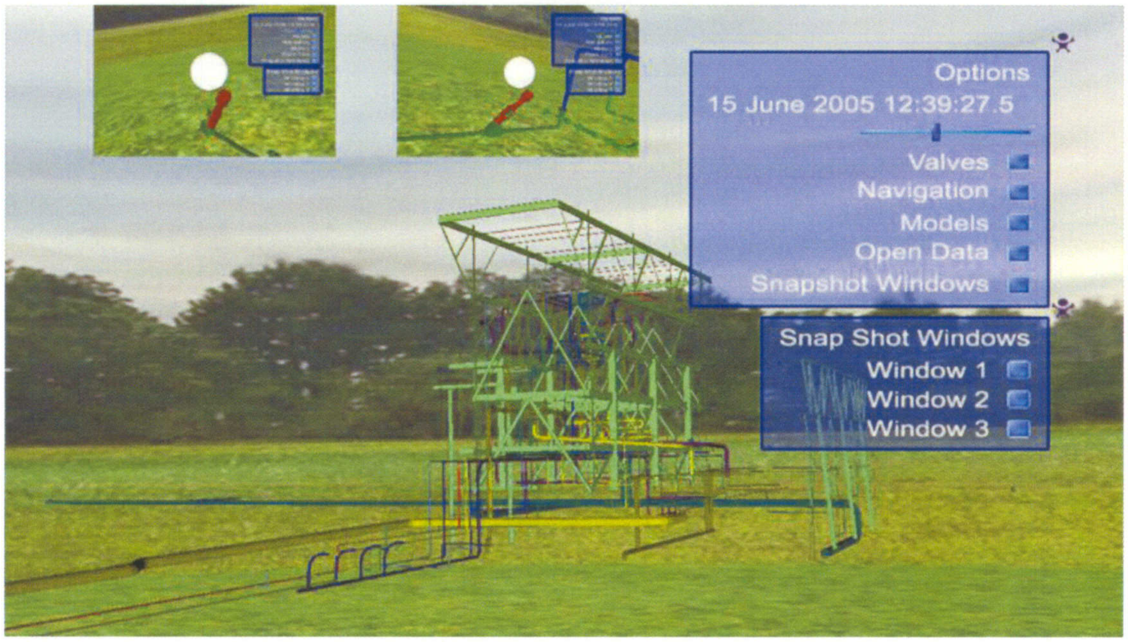


Figure 96: The option to save views and keep them at the top allows for the user to navigate away from objects and still oversee them.

4.3.2 NASA MTTP 1

The results of the first MTTP trailer virtual world can be seen in Figure 97. The user could navigate the world with the mouse while the GUI provided an interface to the visibility of models, the current camera selected to view with, the current timestamp and access to any of the valves. The camera and valve displays were not yet functional; however upon clicking the GUI elements, a fly-menu would appear with additional options that were to be implemented later.

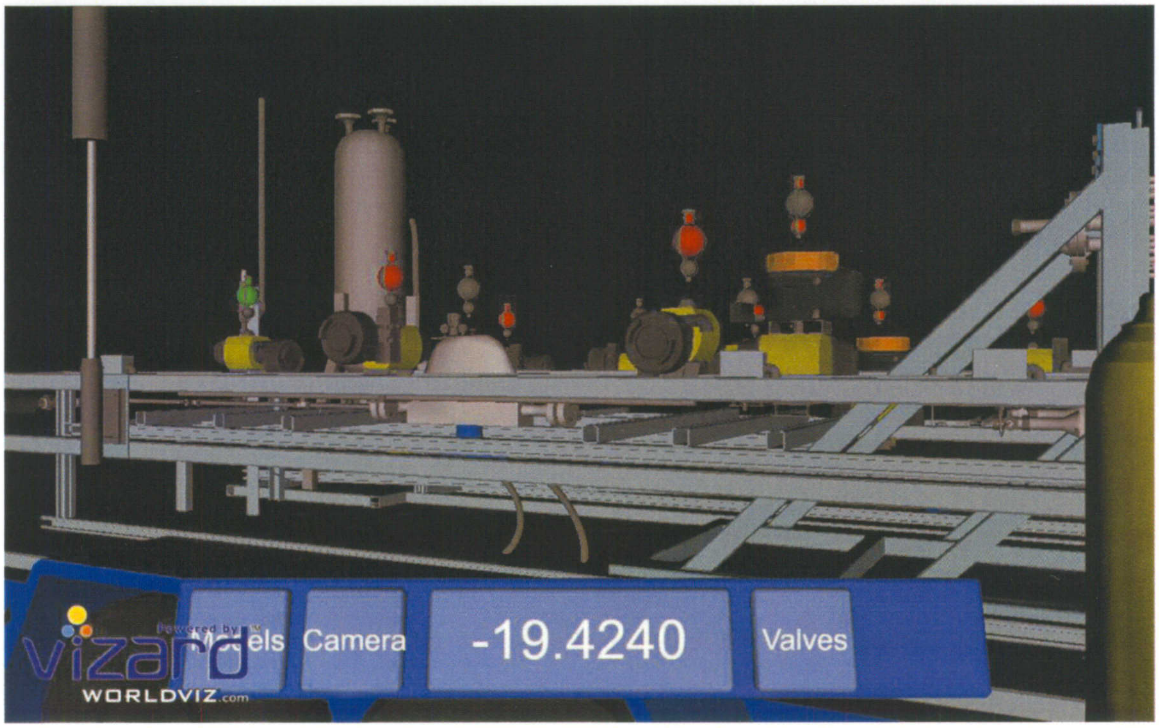


Figure 97: The NASA E3 MTTP Trailer version 1.0 virtual world.

The graphical elements in this virtual world are represented by the MTTP test trailer model. This model was created using the 3-D package Alias Maya 7.0. Initially, several diagrams and schematics were provided but after close scrutiny it was discovered that the actual design of the trailer did not match these plans. Reference images of the model were then requested and provided by Stennis Space Center in the form of 40 images of the trailer from various angles. Figure 98 shows some of the reference images of the MTTP trailer.

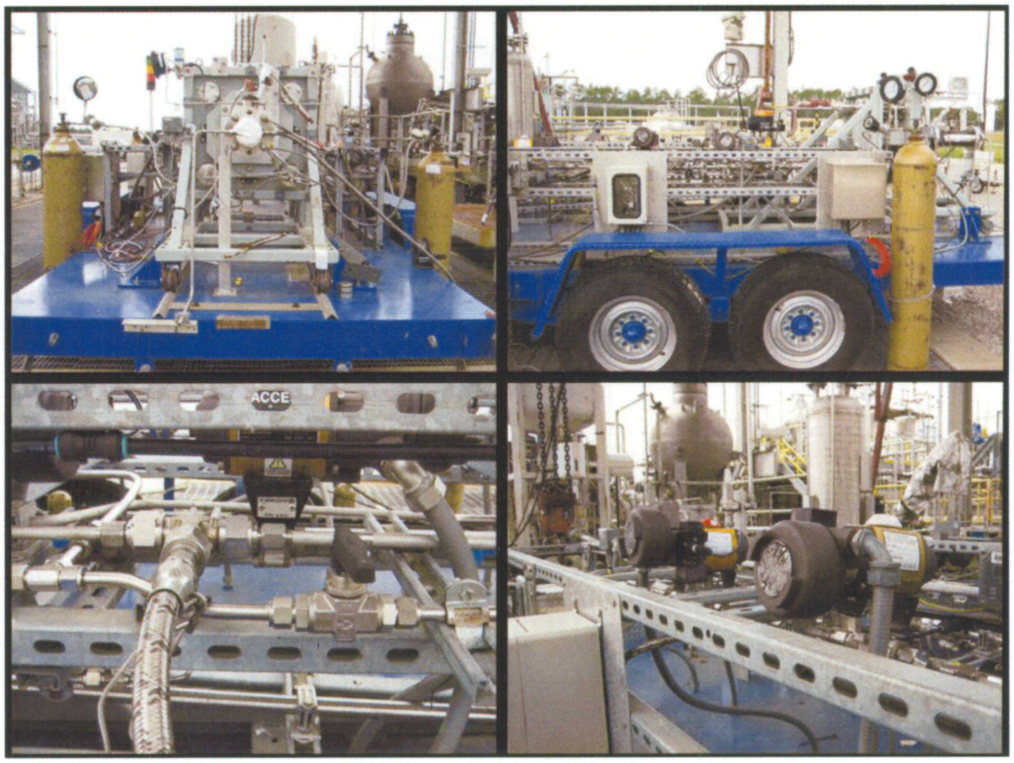


Figure 98: Four of the reference images for the MTTP trailer model.

These reference images provided enough detail to create a visually accurate model of the trailer. This method of creating the graphical elements also had the added advantage of familiarizing the modeler with the structure. The previously mentioned E-1 test stand model suffered due to limited knowledge of the structure, making missing elements difficult to identify. Complete modeling of the entire structure provided intimate knowledge of the various valves, sensors and geometry of the MTTP trailer, which made it easier to position visualizations throughout the structure.

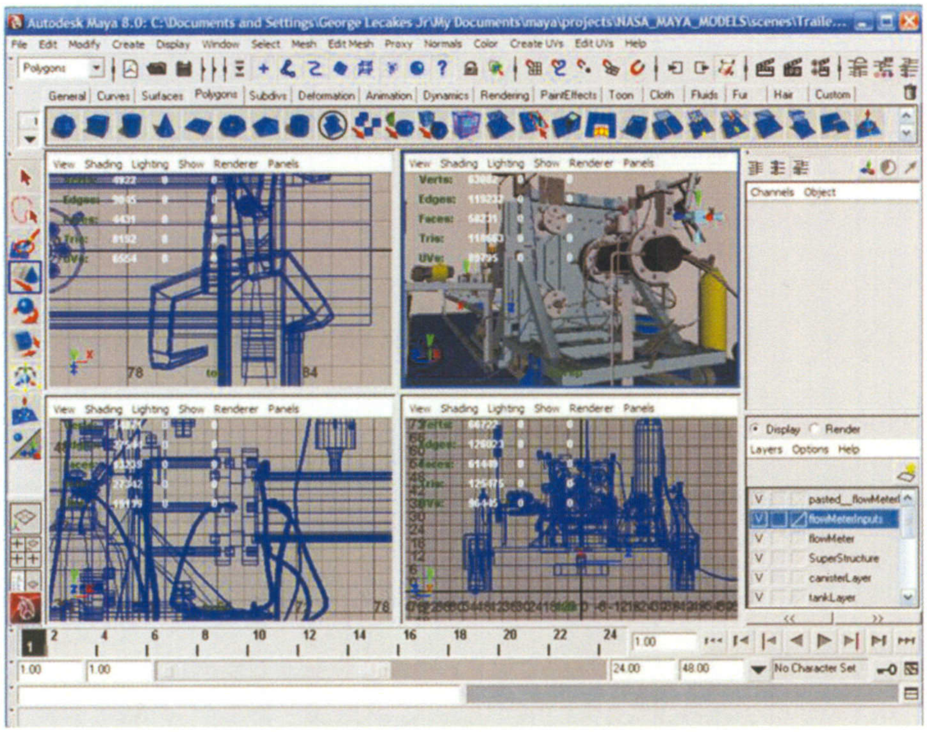

Autodesk Maya 8.0: C:\Documents and Settings\George Lecakes Jr\My Documents\maya\projects\NASA_MAYA_MODELS\scenes\Traile...
File Edit Modify Create Display Window Select Mesh Edit Mesh Proxy Normals Color Create UVs Edit UVs Help
Polygons
General Curves Surfaces Polygons Subdivs Deformation Animation Dynamics Rendering PaintEffects Toon Cloth Fluids Fur Hair Custom
View Shading Lighting Show Renderer Panels
Verts:
Edges:
Faces:
Tris:
UVs:
Channels Object
Display Render
Layers Options Help
pasted_flowMeter
flowMeterInputs
flowMeter
SuperStructure
canisterLayer
tankLayer
No Character Set

Figure 99: The MTTP trailer inside of Alias Maya 7.0.

One important aspect in creating a model from references model rather than attempting to format and import it from outside sources is that great control over the detail and resolution was possible. Figure 100 shows the tires for the trailer at two different resolutions and while the higher resolution might make for a smoother surface it quadruples the number of polygons. The lower resolution version is more than adequate for this section of the trailer, which will attract little attention. Many of the models could be grouped together and exported as a single unit, which made registering the models to the data in the VR world simple. Excessive details such as the nuts and bolts in the structure could be assigned to a separate block and removed from the final visualization if it resulted in poor system performance.

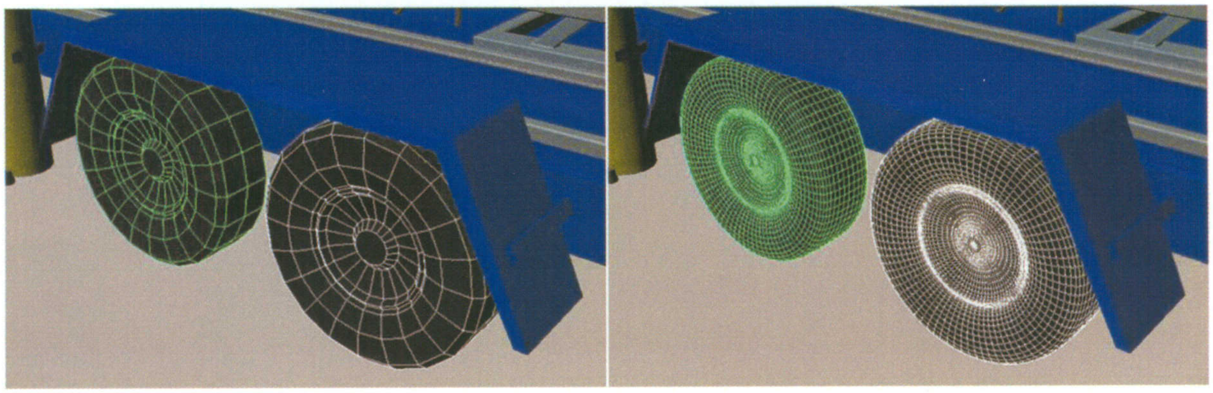


Figure 100: Creating the model provided the ability to scale the resolution of objects to optimize performance.

This version of the MTTP trailer only features three types of data out of the seven types outlined in the approach. Since this was the first attempt at visualizing the data, only the valve states were shown.

The valve states are composed of three pieces of information: valve command, valve state and feedback. The valve command and state are binary numbers, while the feedback is a percentage. Figure 102 is the 3-spheres visualization that the user would see. Each sphere corresponds to a different piece of data, with the top being the command, the middle the state and the bottom the feedback. When the valve is closed a red color is displayed on the sphere and when it is open a green color is shown. During times when it is switching from closed to open, the middle sphere would turn yellow to indicate it is in the process of moving between states. The feedback on the bottom would flash red in instances of high feedback and green when there is no feedback. Figure 102 shows the valve in the neutral gray state and the colored state when data is applied. If a valve cannot receive data, it stays the neutral gray color.

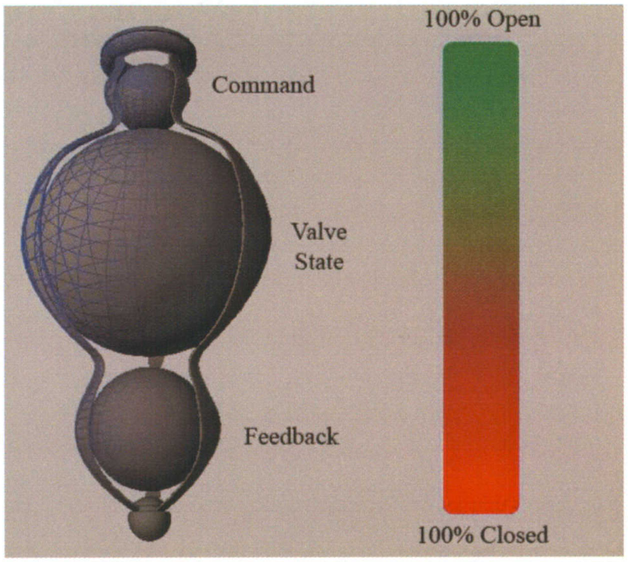


Figure 101: The state of the valve is indicated by color, with green being open and red closed.

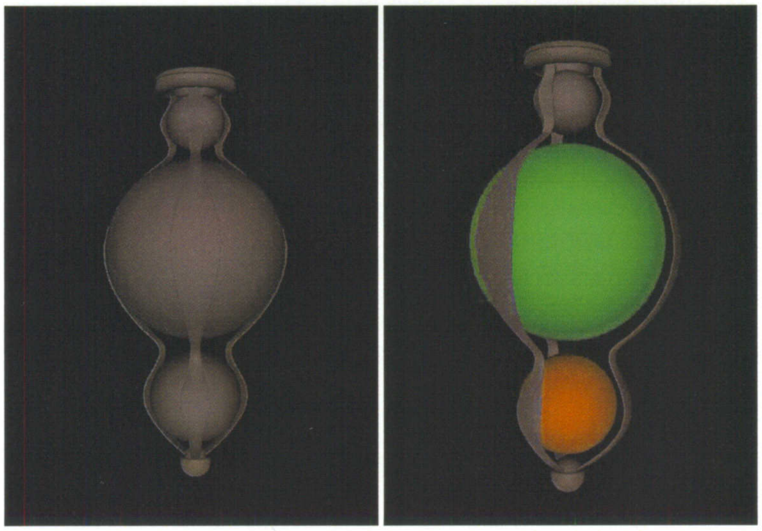


Figure 102: The visualization for valve state, with the top sphere corresponding to command, middle the state and the bottom the feedback.

This preliminary visualization suffers from several flaws. Only a small portion of the data provided was utilized in the visualization. There are also very few options provided to the user to change the visualization method. Finally, no direct way of viewing the data was offered, only the color visualizations.

4.3.3 NASA MTTP 2

The second version of the NASA MTTP trailer virtual environment worked to correct several limitations. Additional features were added to the model including pipe pathways and several valve adjustments. Figure 103 shows MTTP 2 and many of the new features.

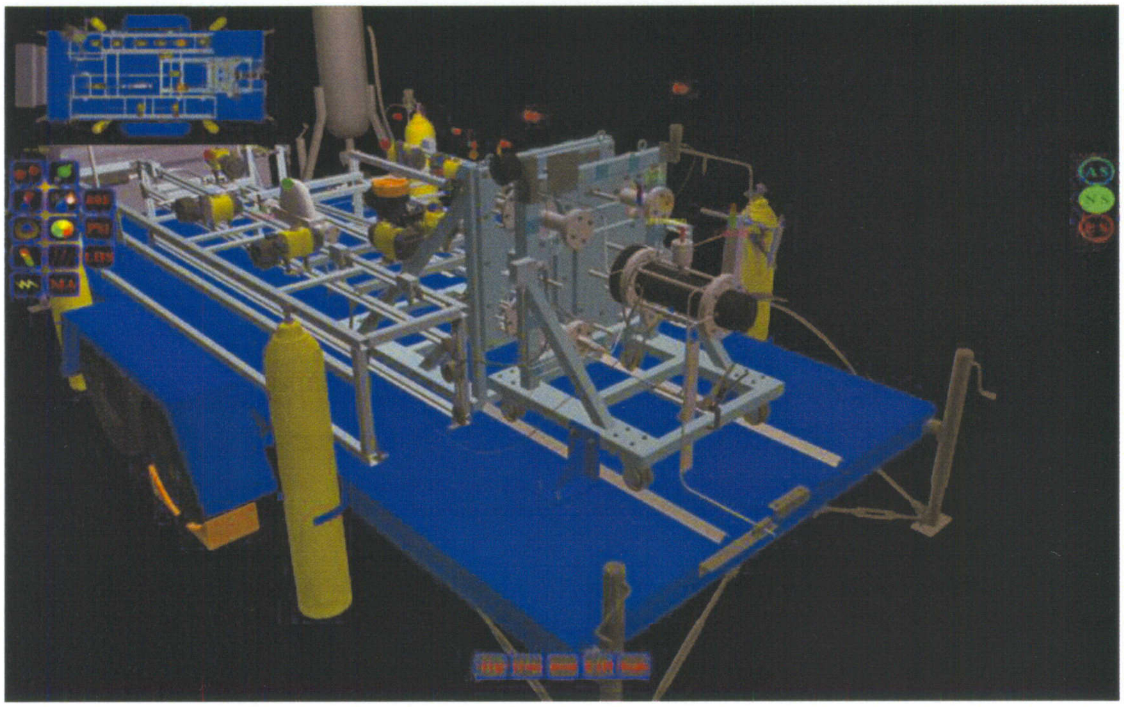


Figure 103: The NASA E-3 MTTP trailer version 2.0 virtual world.

The first addition is a miniature map located on the top left of the screen. This map tracks the user's location in the world with a red dot. While this is not necessary for a small testing site like the trailer, it has merit in the larger E and A series test stands which span hundreds of feet. Figure 104 is the mini-map available to the user. This is a second viewpoint/camera looking into a separate scene with a bitmap image of the trailer. The reason for the bitmap is because the scene would have to be rendered twice if the existing model was viewed, which could compromise performance.

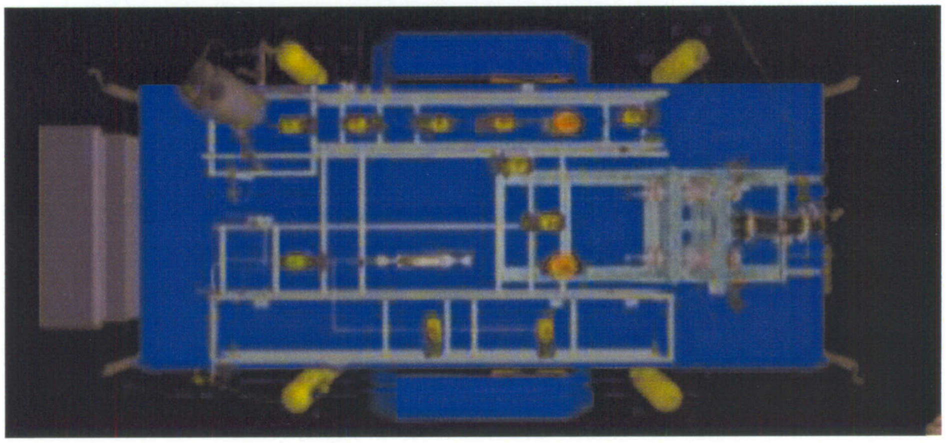


Figure 104: The mini-map indicating user position in the world.

The next new feature added was a shutdown indicator to the right side of the screen. This indicated when the simulation was preparing for shutdown and the type of shutdown commencing. When a particular type of shutdown was reached, the circle would light up as shown in Figure 105 where a simulation was going into normal shutdown.

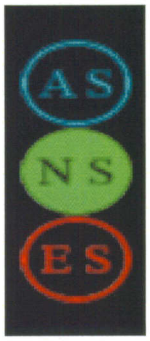


Figure 105: The shutdown states visualized through 3 circles.

Navigation is a key concern when a user is trying to move around a virtual environment. For this reason, several quick buttons were added to assist in moving to predefined views. Figure 106 shows the five buttons positioned on the bottom of the screen. These correspond to the standard isometric style views that most modeling and CAD programs have.

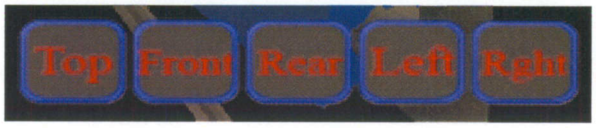


Figure 106: Predefined user views to quickly navigate around the world.

The final piece of the GUI created for the user is the visualization interface shown in Figure 107. This interface provides access to the different visualization methods. Each data type corresponds to a row on the GUI, with an order of valve state, temperature, pressure, strain and current. The ability to turn on or off a numeric display was added to the visualizations. This makes it possible for the user to identify the exact numeric value of the visualization and not just a color.

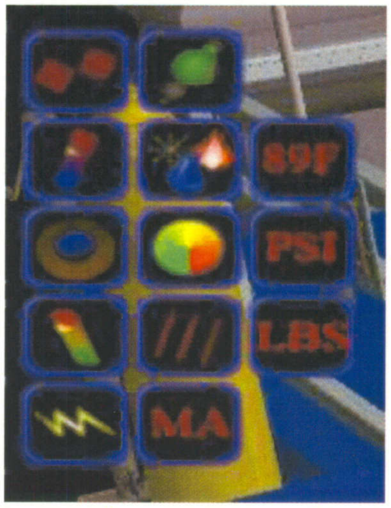


Figure 107: The user interface for changing data visualizations.

Each of the GUI elements can be hidden if they are blocking a portion of the screen. The hotkeys programmed are listed in Table 3.

Table 3: Hotkeys for the NASA MTTP 2 demo.

Hot Key	Descriptions
'G'	Toggle visualization menu GUI
'M'	Toggle mini-map
'V'	Toggle viewpoint GUI
'S'	Toggle shutdown GUI

Several visualizations were introduced with this demonstration because of the incorporation of additional data types. It was also desired to provide the user with different visualizations so they could customize the scene to their liking.

The first group of visualizations corresponds to temperature data. The temperature data was provided as a series of values in Fahrenheit. Two methods for viewing the temperature at a single point were created. The first visualization in Figure 108 is a thermometer visualization approach to viewing the current temperature. The highest and lowest values for temperature at that point are assigned as the minimum and maximum and an arrow moves vertically between those points based on the current temperature. At the front of the test stand, where the rocket fire the temperature has a range from 100 degrees F to over a 1000 degrees. Since the scale is relative to the point, a cold 'blue' reading is for 100 degrees F even though by human standards this is quite hot. Since each point has a relative reading it become important to view the numeric display to get a sense of what hot and cold meant for each temperature point.

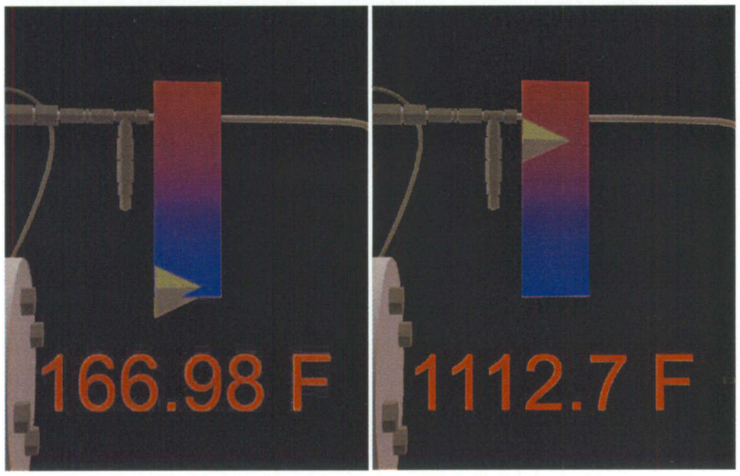


Figure 108: Temperature visualization through a color gradient bar and arrow.

The second visualization is more general and is meant to be viewed over larger distances with numerous other visualizations competing for attention. Figure 109 shows the three states of this visualization with a snow flake meaning cold, a water droplet meaning warm and finally a flame indicating hot temperatures. This visualization works best when several different sensors are being watched together and the user cannot focus on single arrow to read a temperature.

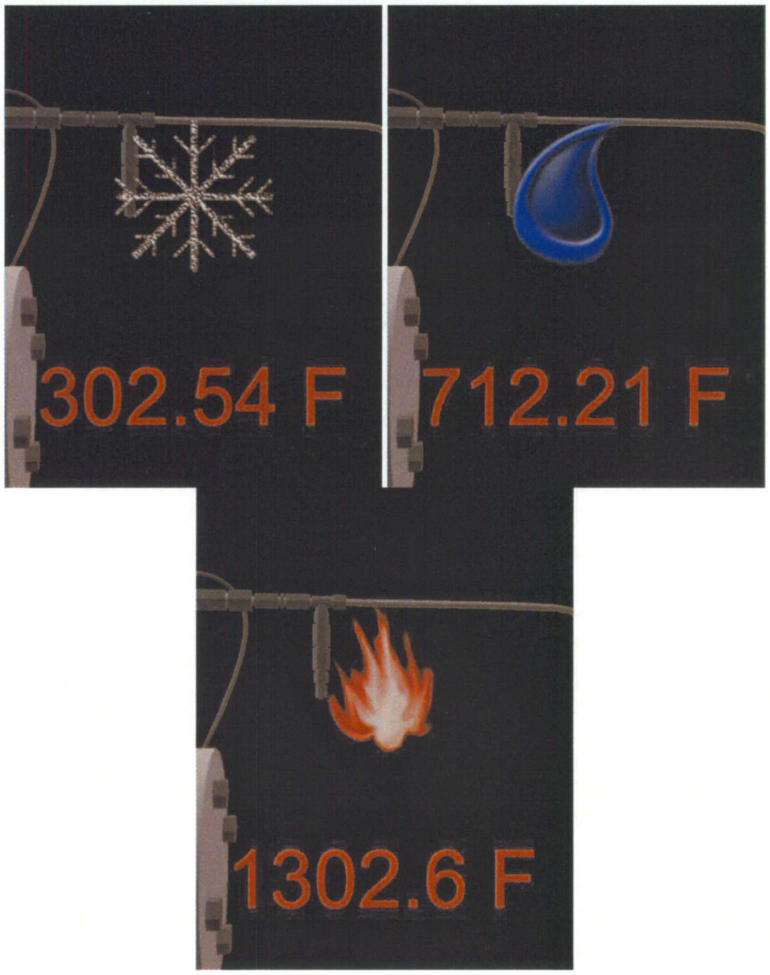


Figure 109: A temperature visualization through images of ice, liquid and fire.

Two pressure visualizations were also created for this MTTP trailer. A variation of the original sphere-in-sphere visualization from the E-1 test stand was created but applied to pressure as shown in Figure 111. The orange portion of the sphere is the maximum possible pressure as determined from the data. The blue sphere shrinks and grows to fill the orange space depending on the current pressure. This visualization is possible by reversing the face normals on the orange sphere, which allows you to look inside the object and see the inner sphere. This visualization is best when attempting to view the pressures of several locations and only a general understanding of the current reading in relation to others is needed.

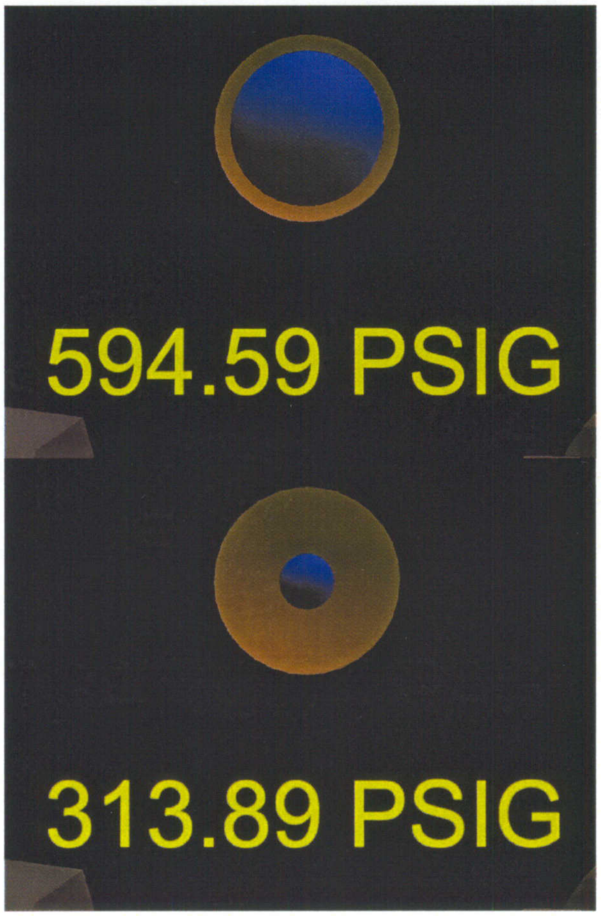


Figure 110: Pressure visualization utilizing a sphere-in-a-sphere.

The second visualization for pressure provides a better understanding of the exact value. A gauge is utilized with a needle that moves between the low and high points as shown in Figure 111.

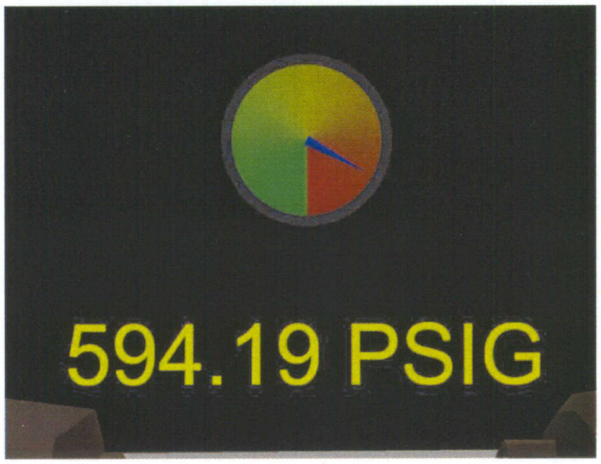


Figure 111: Pressure visualizations utilizing gauge display.

For valve states the visualizations were changed for this version to provide something the user could more easily understand. Rather than use the existing 3-spheres approach, a new method of two diamonds was incorporated. Each diamond was labeled for easy identification. The colors were kept the same, with green meaning open, red meaning closed, yellow in a state of flux and gray meaning no incoming data as shown in Figure 112.

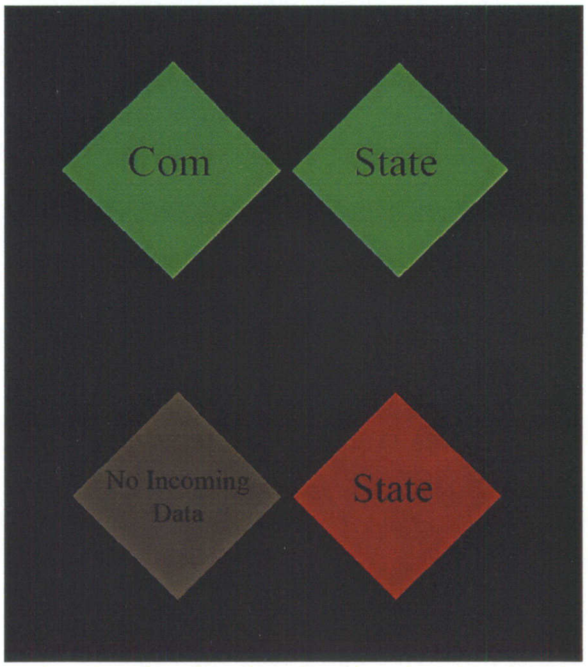


Figure 112: Valve state visualization showing command and valve state with the colors green (open) and red (closed) and gray (no data).

Feedback was separated from the valve state and given a new visualization based on scale. The feedback can be seen in Figure 113 as three yellow pulsing bars. The purple transparent bars represent the highest possible feedback of one-hundred percent. Since the feedback is driven by a function affecting the model's scale, a feedback of zero will result in the disappearance of the model as it shrinks to a single point.

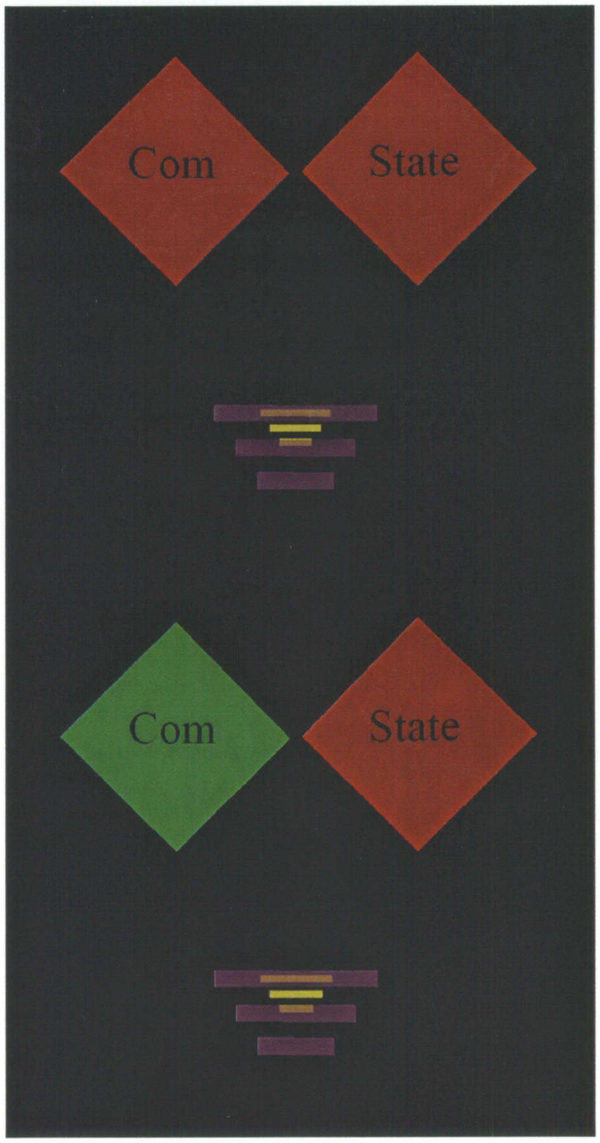


Figure 113: Feedback visualization which displays feedback in the sensor relaying information on the state of the valve.

Strain was handled in two different ways which can be seen in Figure 114. The first visualization shows strain oscillating between three purple bars, similar to the feedback visualization. The purple bars are the maximum strain for the data at that point. The red bars indicate the current level of strain. The second visualization is the same as used for temperature, with a vertical bar indicating the degree of severity. Green is the lowest level with red being the highest possible strain at that point.

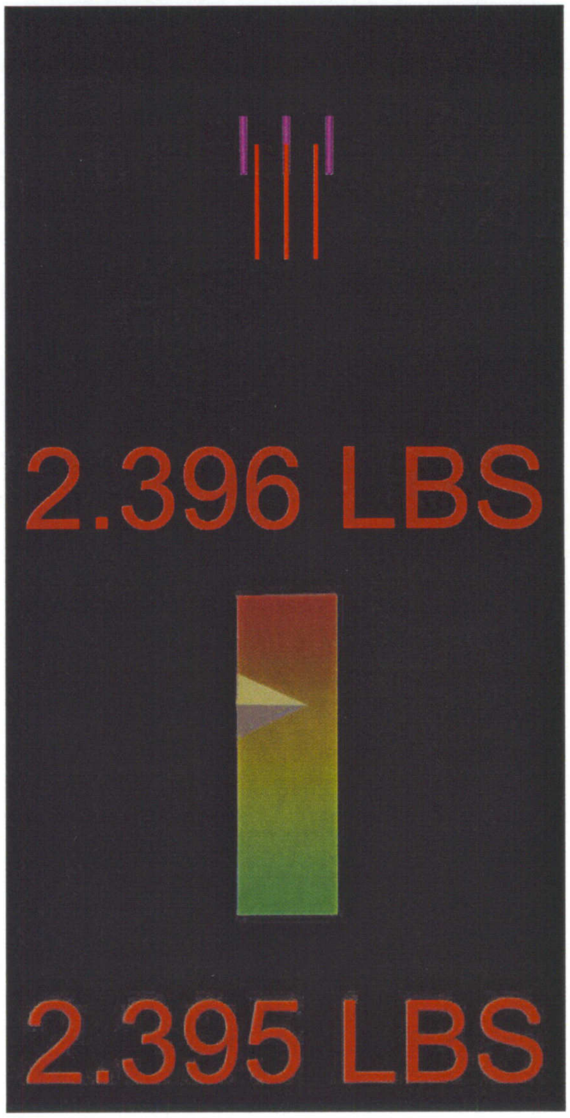


Figure 114: Strain visualizations in MTTP 2.

The visualization for current needed to only indicate that the igniter was fired. Only a single visualization was created to show this feature. A pulsing lightning bolt seen in Figure 115 clearly points out that the igniter is firing when the rocket engine starts.

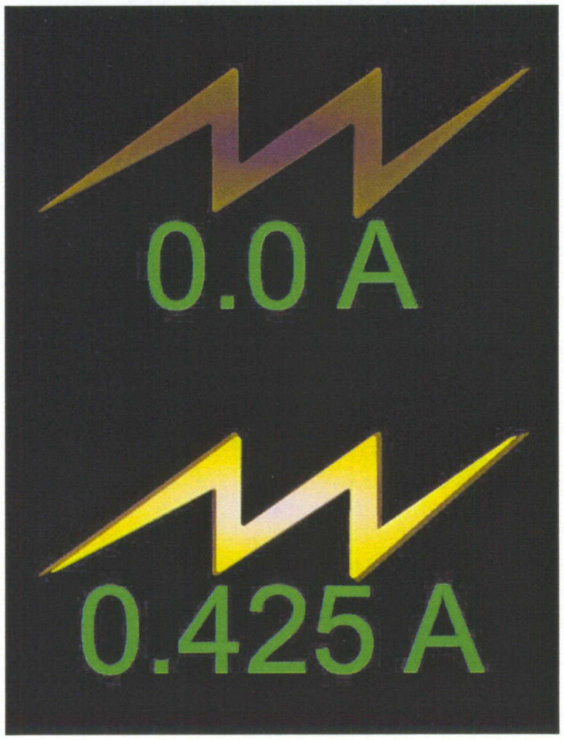


Figure 115: Current visualization shown through a pulsing lightning bolt.

While there were advancements in this version of the MTTP trailer there were several failings. When the demonstration runs, it is impossible to select individual elements or turn off elements that the user is not concerned with. In addition, the user has no control over time and must wait for the program to loop over to revisit a particular point in time. These two features will be directly confronted in the next rendition of the MTTP trailer.

4.3.4 NASA MTTP 3

The third and final version of the MTTP trailer makes several improvements upon the previous two incarnations. The model was refined and completed; with textures representing details too small to create through 3-D modeling. A data set algorithm was created to load any CSV files within an assigned folder which lets the user select them from a drop down list. Time controls were also created to give the user the ability move through time. Finally, interactive elements of the model were produced to allow for the selection of visualizations on a per-object basis. A global menu was included in the event that the user would prefer changes to the entire test stand with a single click of a button.

The previous versions of the MTTP trailer did not contain all the piping and models from the actual MTTP trailer at the Stennis Space Center. Many of the areas inside the MTTP were densely packed making it difficult to identify separate elements. After procuring new photographic reference material, complex piping pathways on the right side of the test stand were completed. Figure 116 shows the right side of the trailer and some of the new piping.

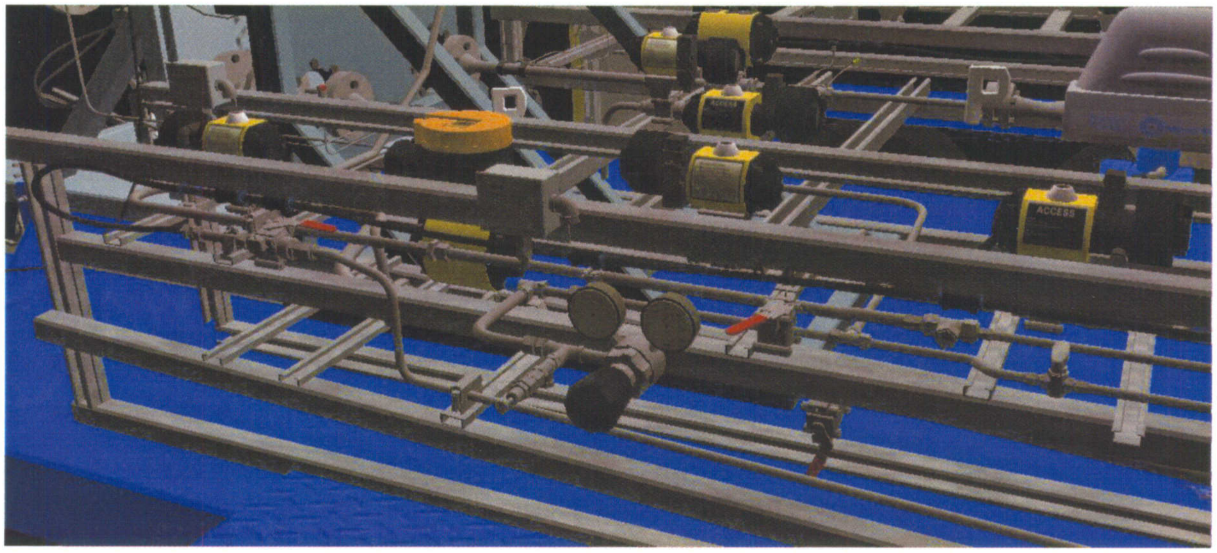


Figure 116: Finalized piping on the MTTP trailer.

There were many details excluded from the previous versions of the trailer due to computational constraints. Many of the fine points were nearly impossible to show through modeling and required textures mapped over the objects. In this trailer, all of the objects were textured to provide additional realism and to include important information about each component. Figure 117 shows several examples of textures applied to the models.

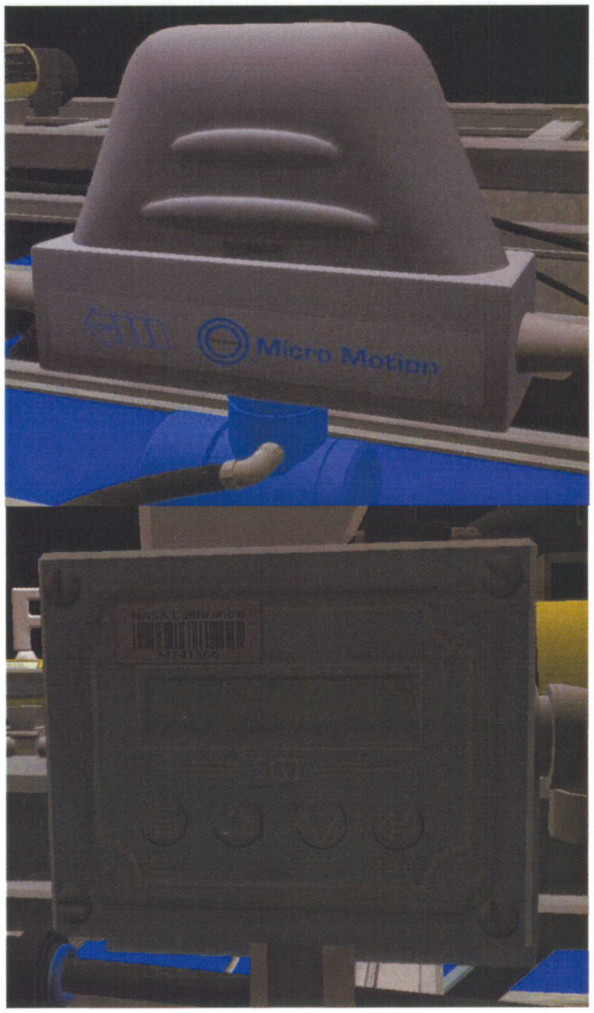


Figure 117: An example of several textured elements inside the visualization.

Previously, the user was locked into a single test firing when utilizing the old demonstration. For the new MTTP, a folder was created to house the many test firings. A drop down list was then added so that the user could easily switch between the different data sets. Figure 118 shows the interface for selecting datasets inside the demo. This interface can also be hidden with the ‘d’ hotkey on the keyboard.

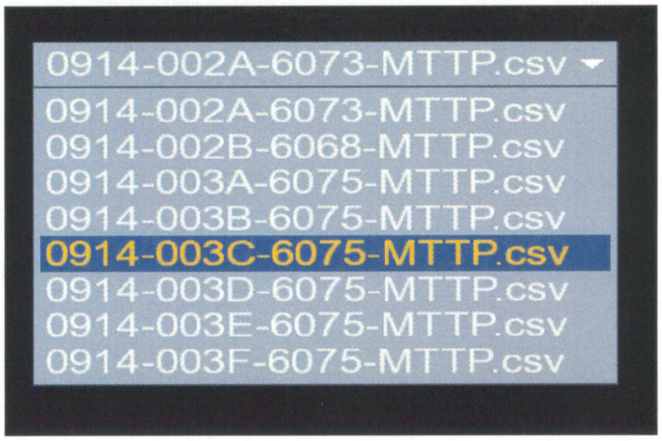


Figure 118: Data menu, which loads all data files from a folder for the user to select.

Time controls were also created for this demonstration to allow the user to sift through the information. Before, the user took a passive role and watched the demonstration as it occurred, waiting for a loop in the playback to see the same information twice. In this demonstration the user has been given several options to move through time as shown interface in Figure 119. The blue icons allow the user to step forward and backward through time. The blue play button causes the simulation to run until paused with the red stop button. The arrow and bar bellow provide a means of scrolling throughout the entire time series when it is paused. The skip field allows the simulation to skip over data and speed up the simulation. The slow field reduces the play rate of the demonstration by an integer input value. Entering a value of two will cause the demonstration to play back at half speed. Finally, on the bottom is the current time stamp the user is at. Here it reads no data since none has been chosen in the drop down menu.

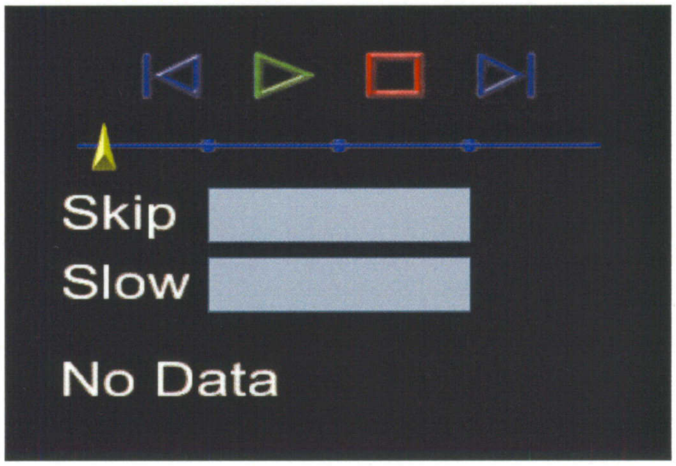


Figure 119: The time controls that can be utilized by a user to step forward, stop, slow down, speed up or scroll through time.

The first and second MTTP demo allowed global changes to be made to visualizations. In this third version the user was given the ability to interact with individual objects. When the mouse icon moves over an interactive object the emissivity is increased and changed to a red color as shown in Figure 120. When the user clicks on the object a red target appears to indicate the currently selected object. A small interface also appears on the right to show you the valve name and available visualizations. The user can deselect the object by clicking it an additional time, which will subsequently close the GUI interface to the right and the red target.

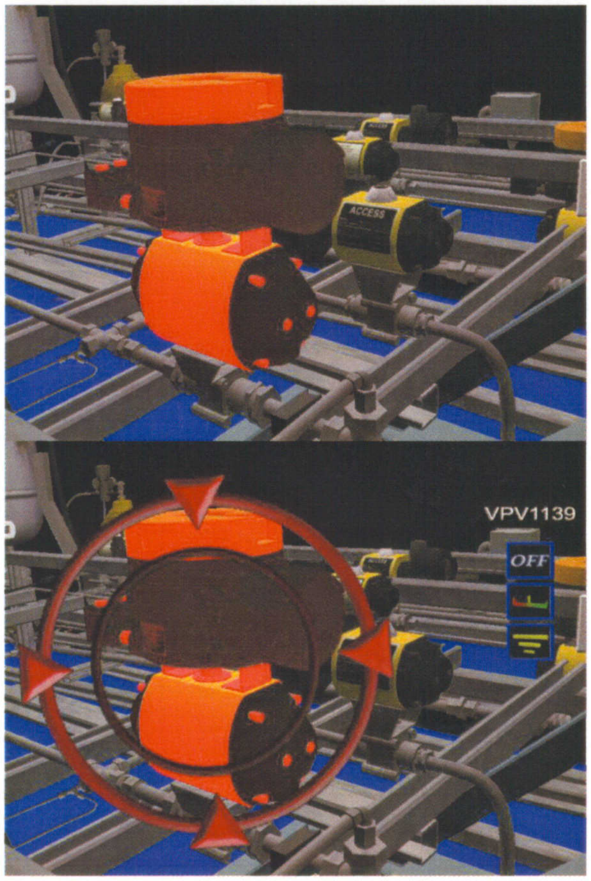


Figure 120: Interactive elements will highlight when the mouse is over them and can be clicked to interact with their visualizations.

Should the user wish to turn off or change all the visualizations at once, a global menu has been implemented. Pressing the ‘g’ key on a keyboard will bring up the Global menu where the user can open each sensor type as shown in Figure 121. One key feature added here is the ability to turn off visualizations. Multiple visualizations on the screen can cause confusion and cluttering issues. Being able to turn off areas that might be irrelevant can help focus a user to a particular task at hand.

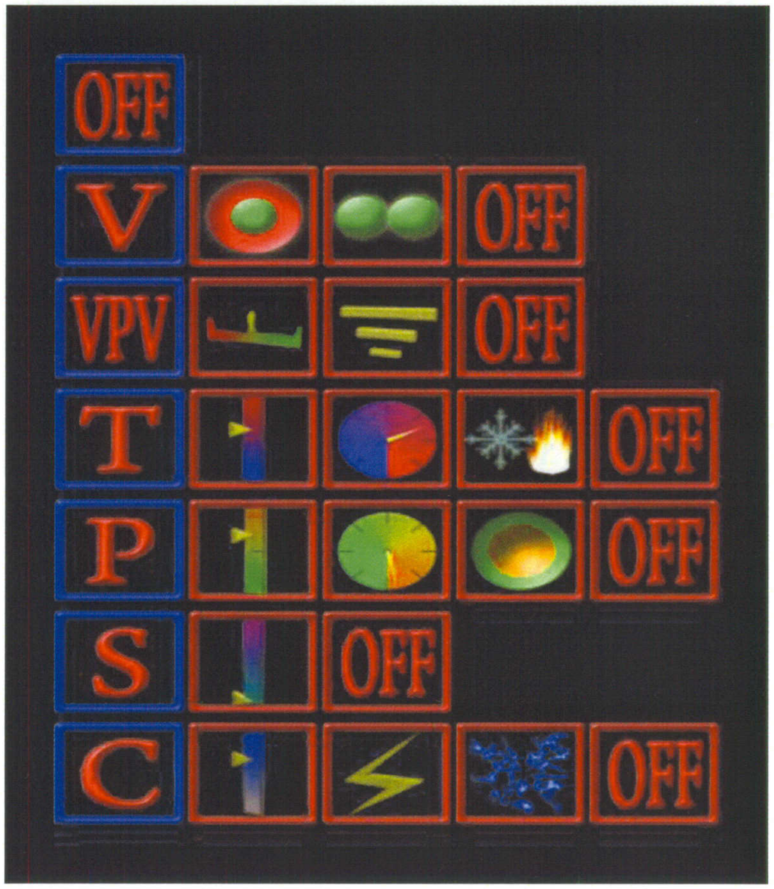


Figure 121: The Global GUI, capable of changing the states of all objects in the scene.

The MTTP demonstrations have provided a development environment for the visualization of singular points which correspond to information that most humans are familiar with. Utilizing simple concepts such as color and imagery, the end goal was to provide an intuitive interface for reviewing and one day monitoring the health of a rocket engine test stand.

4.3.5 Performance Evaluation

Inasmuch as an extensive body of previous research work exists attesting to the effectiveness in VR to provide improvement in the ability of the users to interpret data from complex systems, a human-factors study deploying the visualization platform in the field should form a part of the overall research project. Although, this has not been a major focus of this research work, a preliminary study for obtaining a human-based performance evaluation was conducted.

A questionnaire was presented to test-engineers at NASA-Stennis Space Center, requiring a comparison of the procedures that they currently employ for performing ISHM of rocket engine tests, with the VR platform that we have designed. The survey requested responses to two sets of questions – one for the users (Categories 1-3) and another for the developers (Categories 4-5) of the VR environment. The survey respondents were asked to rate specific categories on a scale of 1 (poor) to 5 (excellent). The user questionnaire focused on the user's navigation, interaction and immersion experiences. The developer section of the questionnaire concentrated on the versatility and memory requirements of the software. The respondents were asked to rate not only the features corresponding to the performance of the visualization environment, but also the importance of the specific feature. In addition, the respondents were asked to identify the methods they currently employ for performing ISHM.

Table 1 summarizes the average responses of the three users and developers, comparing the VR environment presented in this thesis, with the procedures that are currently used at NASA-Stennis Space Center. The responses indicate that features provided by a VR environment, namely, navigation, interaction and immersion are rated to have significant importance; and the most important features being those corresponding to interaction. The VR environment clearly outperforms the current procedure in the navigation and immersion categories, whereas in the interaction category, the performance of the VR environment is equal to or greater than the current process. From the developers' point of view, the VR environment clearly exceeds the performance of the current procedure in the case of development time and flexibility, and is comparable in the case of compatibility, memory requirements and execution time.

These human based performance evaluation results provide us with confidence that the VR platform provides value in integrating graphical, measurement and health data in an immersive, navigable and interactive manner.

Table 4: Performance evaluation results

Category	Question	Average Score		
		Importance	Current Procedure	ISHM Visualizatio n
NAVIGATION	Ease of Navigation to a specific time/location/sensor reading	4.5	2.5	4.0
	Multiple methods of navigation	3.0	3.0	3.5
INTERACTION	Ease of selecting the source of sensor readings, components and health	5.0	3.5	4.0
	Ease of querying data from sensor readings, components and health	5.0	4.0	4.0
	Ease of identifying the source of sensor readings, components and health	5.0	4.5	4.5
IMMERSION	Ability to view sensor readings, components, health	4.0	3.5	4.0
	Model representation faithful to reality	3.5	2.5	4.0
	Identifying the location/time of an anomalous/ benign condition	5.0	4.0	4.5
VERSATILITY	Development time	2.5	1.5	3.5
	Compatible file formats	3.5	3.0	3.0
	Flexibility to develop models	4.0	2.5	4.5
MEMORY	System memory requirements	3.5	3.0	3.5
	Execution time (software package)	4.0	3.0	3.5

4.4 Sand Visualizations

The sand visualizations were an attempt to provide an environment where a user could compare the same two particles of sand that utilized different non-destructive techniques to create 3-D models. Two demonstrations were created, with the first utilizing sensors and a floor pad to interact with the particles. The second demonstration was implemented as a side-by-side viewing instrument with database capabilities.

4.4.1 Sand Demonstration 1

The initial sand demonstration utilized the flock-of-birds sensor setup for interacting with the particles. The two particles were mapped to sensors so that the user could move each hand and the particles would move with them. By manually repositioning the particles through the user's hand movements the aim was to allow for visual inspect to identify differences in the reconstruction techniques. Figure 122 depicts the final demonstration.

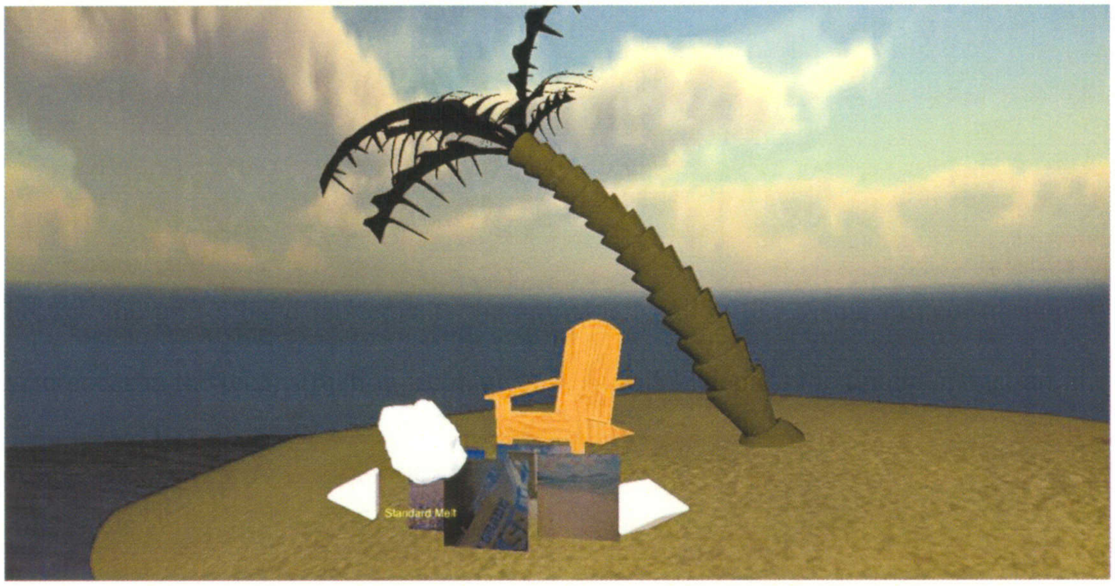


Figure 122: The sand virtual world comparing sand particles of various types.

A menu was provided that allowed for the user to cycle through particles from different locations in the world and compare the reconstruction techniques. This menu was linked to a floor pad so that the user could select a particle with their feet while positioning them in 3-D space with their hands. Figure 123 is a close up of the GUI and the several options available to the user in this demo. In addition to controlling the selected particle the floor pad provided the user with the ability to scale the particles with forward and back buttons for closer inspection.

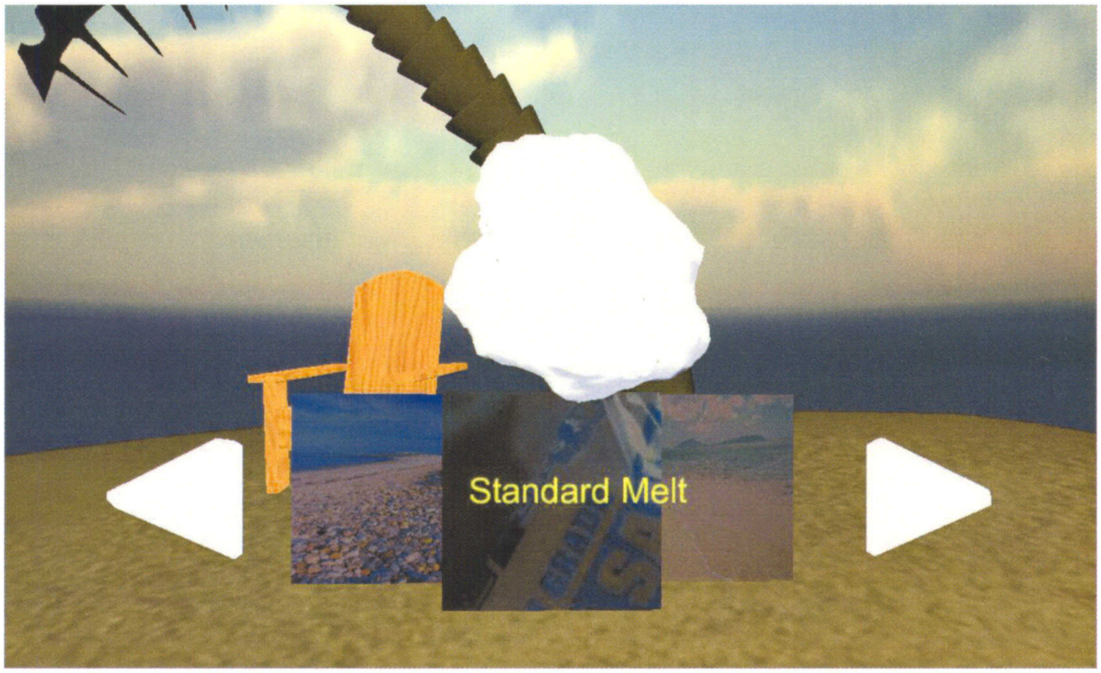


Figure 123: The sand particles can be changed by cycling through a menu.

In order to make the environment more immersive and appealing to the user, several graphical and auditory elements were incorporated. Sounds of the tide and seagulls were added and randomly played as the demo progressed. A beach with a rising tide as well as a palm tree that swayed in the wind was also included. These were provided to add background to the particles so that the user did not have to view an empty black void.

4.4.2 Sand Demonstration 2

The second sand demo was also meant for viewing synthesized x-ray and tomographic particles. The core difference in this demo was that the user interface was switched from sensors to mouse control. The model information was stored in an XML format so that details concerning the name and type of scanning technique could be imported with the models. Figure 124 shows the final demonstration.

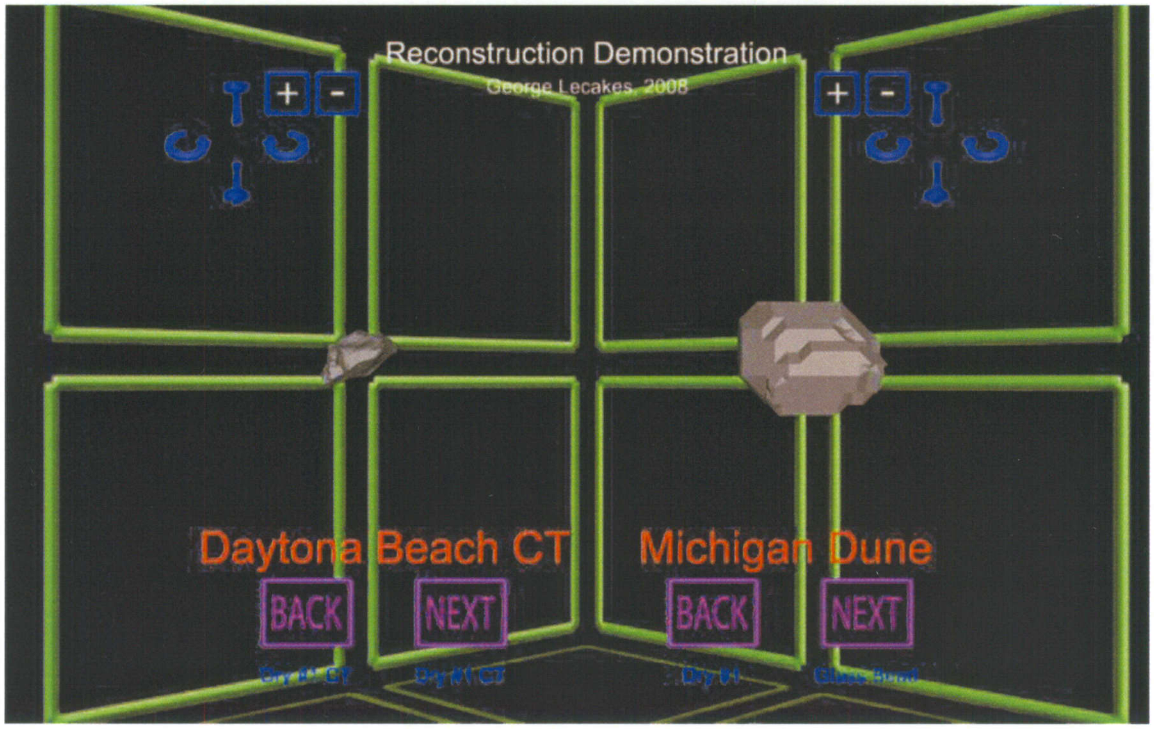


Figure 124: The second sand virtual world, comparing x-ray and optical reconstructions.

Since the demonstration no longer utilizes the floor pad or the sensors, all controls were moved to mouse interactions. Figure 125 shows the new particle interaction menu. The plus and minus keys allow for the user to scale the particle. The arrows rotate the particle along the y and x axis. Clicking on the particle once will set it to auto rotate mode around the y axis. Clicking the particle twice will cause it to rotate in the opposite direction. Finally, clicking the particle a third time will reset all transformations on the particle and bring it to the original position.

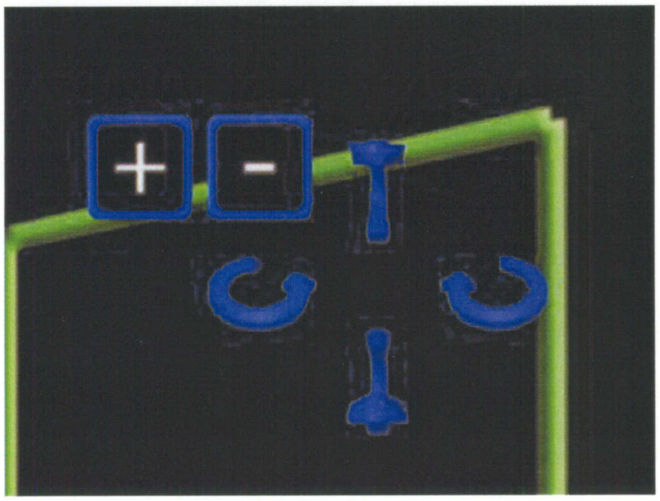


Figure 125: The user can rotate and scale the particles through this interface.

4.5 Tomographic Reconstruction Algorithm

This portion refers to the creation of a tomographic reconstruction technique to improve upon the previously employed algorithm. For the reconstructions two sets of models were created. The first set is composed of synthetic data, created from 3-D models in Maya. The second batch of data was directly taken from the Rowan University geomaterials database of optical tomography.

4.5.1 Synthetic Models

For the synthetic models, several trials were performed at various resolutions and projections. Table 5 is the summation of data for the first synthetic model, the sphere. The sphere was rendered out to images of the square dimensions 10, 50, 150 and 250 pixels. These images were then broken into groups of various projections of 2, 4, 10, 30, 60, 90 and 180.

Table 5: Reconstruction time trials for a sphere

MODEL	PROJECTIONS	RESOLUTION	Time (sec)	Time (sec)	Total Time (sec)
SPHERE	2	10x10	0.5997	0.5901	0.5949
SPHERE	2	50x50	2.0712	2.035	2.0531
SPHERE	2	150x150	22.0128	22.1663	22.08955
SPHERE	2	250x250	118.3328	118.513	118.4229
SPHERE	4	10x10	1.1255	1.0901	1.1078
SPHERE	4	50x50	4.2604	4.2003	4.23035
SPHERE	4	150x150	49.8509	44.1373	46.9941
SPHERE	4	250x250	235.5993	234.0518	234.82555
SPHERE	10	10x10	2.0785	2.0412	2.05985
SPHERE	10	50x50	10.0631	10.0219	10.0425
SPHERE	10	150x150	108.581	108.3267	108.45385
SPHERE	10	250x250	583.8433	579.8994	581.87135
SPHERE	30	10x10	5.307	5.2126	5.2598
SPHERE	30	50x50	30.238	29.6184	29.9282
SPHERE	30	150x150	322.9081	319.3392	321.12365
SPHERE	30	250x250	1673.4	1755.7	1714.55
SPHERE	60	10x10	10.7758	9.9704	10.3731
SPHERE	60	50x50	62.8017	59.2639	61.0328
SPHERE	60	150x150	655.4501	642.9773	649.2137
SPHERE	60	250x250	3499.6	3431.3	3465.45
SPHERE	90	10x10	14.7566	14.72	14.7383
SPHERE	90	50x50	89.6395	88.5015	89.0705
SPHERE	90	150x150	958.2297	986.1672	972.19845
SPHERE	90	250x250	5281.4	5229.9	5255.65
SPHERE	180	10x10	31.0532	29.0604	30.0568
SPHERE	180	50x50	184.5311	176.9476	180.73935
SPHERE	180	150x150	1968.8	1930.3	1949.55
SPHERE	180	250x250	10540	10469	10504.5

Reconstructions were performed on a Hewlett Packard 7800 series machine. This series has an Intel Core 2 Quad CPU clocked at 2.40GHz with 2.98 GB of RAM. The operating system utilized was Microsoft Windows XP 32-bit edition with Service Pack 2 installed.

Each reconstruction was performed twice and the time averaged. This was due to Matlab having a poor initial time trial and with subsequent trials having improved performance. As evident from the data, the higher the resolution and the more projections utilized led to a longer reconstruction time. The shortest time for a possible reconstruction, but at the lowest resolution and with the least number of projections was nearly half a second. The longest reconstruction, which utilized the highest resolution currently possible, took nearly three hours to complete.

There is a technological limit to the maximum resolution the algorithm can handle. A 250x250 resolution image is the maximum that can be handled with a 32-bit system using this algorithm in Matlab. This is due to a constraint on memory, where Matlab is unable to concatenate two matrices larger than 250x250 elements. Increasing the amount of onboard memory through the use of a 64-bit operating system could resolve this problem.

In addition, by utilizing later version of Matlab that are more compatible with multiple core processors it may be possible to drastically increase reconstruction times. Since the algorithm utilizes several loops through the arrays, and each loop is not depended on the previous, the commands could be implemented across several cores.

4.5.1.1 Sphere Reconstruction

Figure 126 compares the original modeled object to the results of the algorithm when only two projections were utilized. The lowest resolution object is second to the left and shows an important problem with reconstructions.

When creating the projections, if the object intersects the border pixels the Matlab iso-surface function is unable to close the surface and yields open portions of the model. This can be seen in all of the reconstructions of spheres at a resolution of 10x10 pixels.

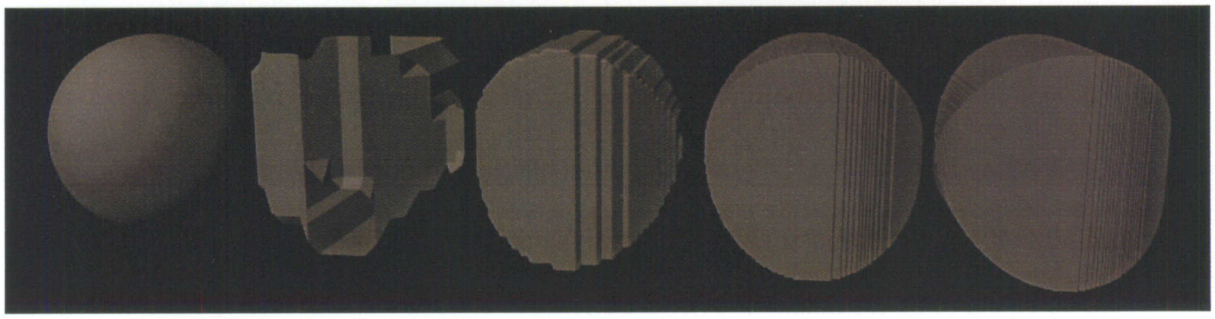


Figure 126: Original sphere versus 2 projections at 10x10, 50x50, 150x150, 250x250.

Figure 127 shows the actual pixels used in the 10x10 image during reconstruction. Because the original model had to be sampled and fit into a 10x10 array, much of the information was lost. This can be readily seen across all three models, as the sphere no longer appears round, the arrow loses much of the information defining the overall shape of the head and the humanoid loses most of the information describing the limbs. Many of the objects have gray levels which may be considered noise and removed during the reconstruction.

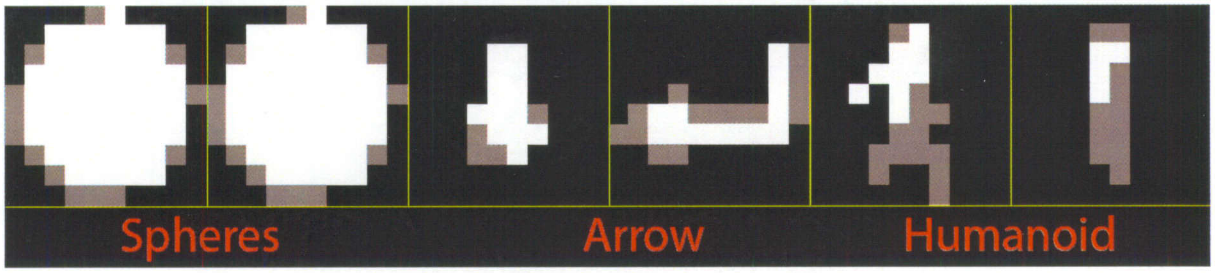


Figure 127: Enlargements of the 10x10 pixel images.

The number of projections can vastly decrease the reconstruction's fidelity to the originally modeled object. Since there are only 2 projections at ninety degree angles the final object looks more like a cut out. The spheres in Figure 126 are closer to a square shape with hard edges on the sides due to an insufficient number of projections to cut away the hard edges.

Figure 128 shows improvement in reconstructing a curved surface by increasing the number of projections. The cuts from each projection can still be clearly seen in all of the objects except for the second, which is due to poor resolution.

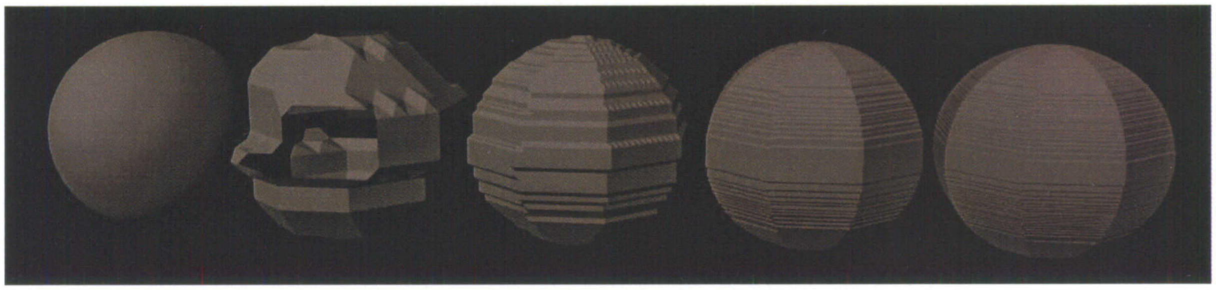


Figure 128: Original sphere versus 4 projections at 10x10, 50x50, 150x150, 250x250.

Increasing the number of projections to ten continues to smooth the surface as shown in Figure 129. One important element to notice is that at ten projections or higher, no new information refines the 10x10 resolution shape. The changes in the following figures will be subtle and barely distinguishable using visual inspection.

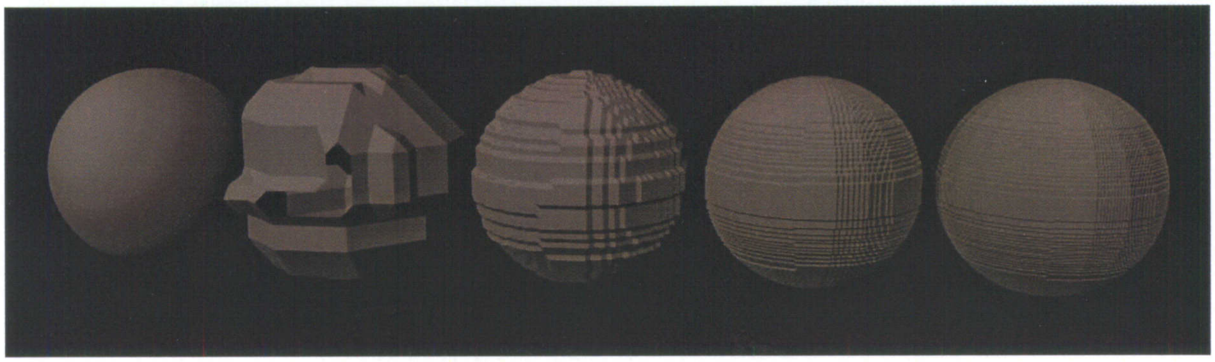


Figure 129: Original sphere versus 10 projections at 10x10, 50x50, 150x150, 250x250.

It is only at the higher resolution of 250x250 where the additional information provided by 30 projections can be noticed as seen when comparing Figure 129 to Figure 130.

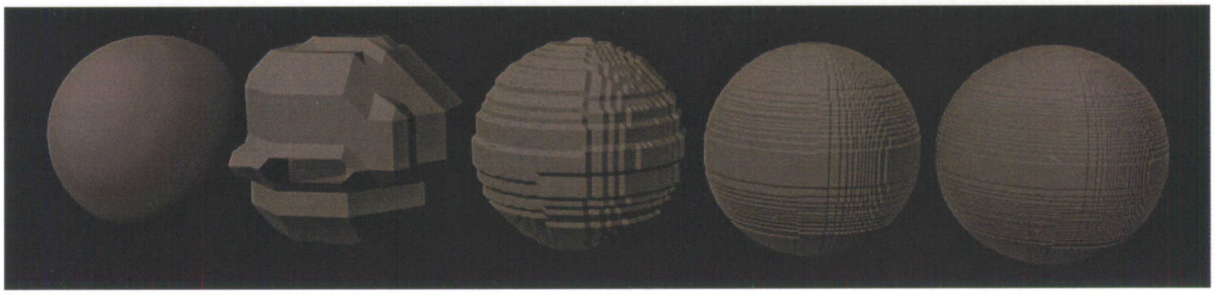


Figure 130: Original sphere versus 30 projections at 10x10, 50x50, 150x150, 250x250.

After thirty projections at a resolution of 250x250 pixels, nearly no new information is presented to the reconstruction. This can be seen in Figure 131, Figure 132 and Figure 133 where no change in the model can be visually seen.

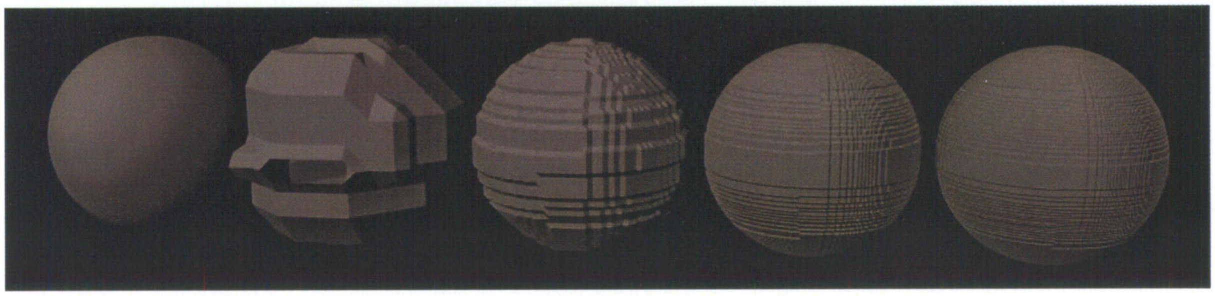


Figure 131: Original sphere versus 60 projections at 10x10, 50x50, 150x150 and 250x250.

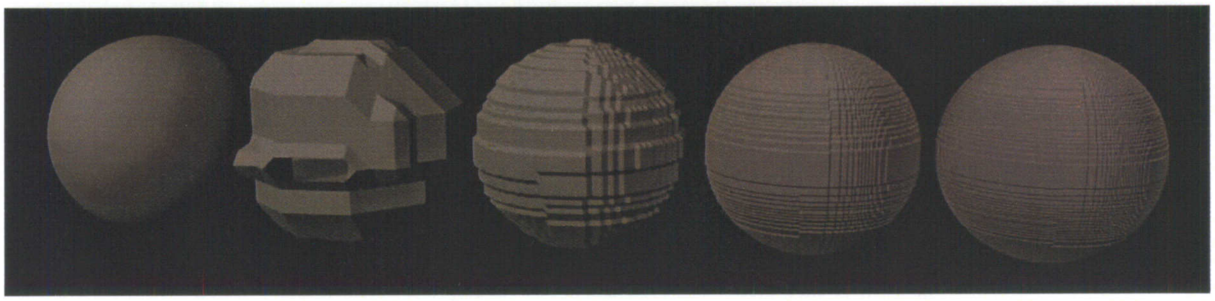


Figure 132: Original sphere versus 90 projections at 10x10, 50x50, 150x150 and 250x250.

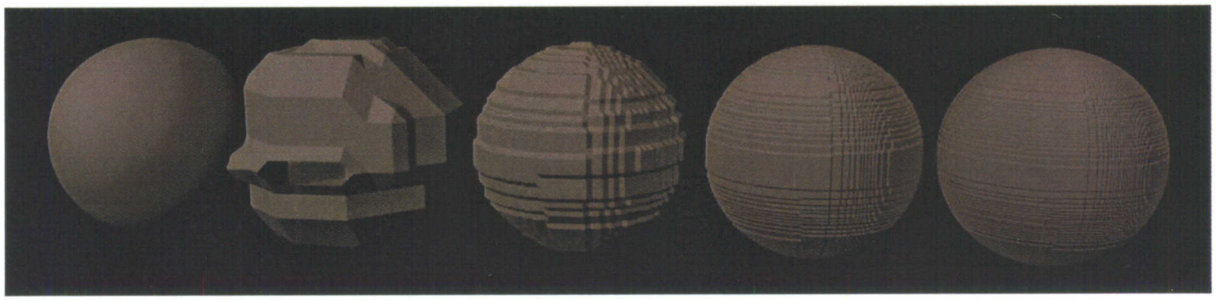


Figure 133: Original sphere versus 180 projections at 10x10, 50x50, 150x150 and 250x250.

For spherical objects there appears to be a limit to the benefit of additional projections when dealing with a fixed resolution of 250x250 pixels. Additional projections over 30 provide little added benefit when under visual inspection.

Figure 134 is a compilation of each resolution and projection to give a side by side comparison.

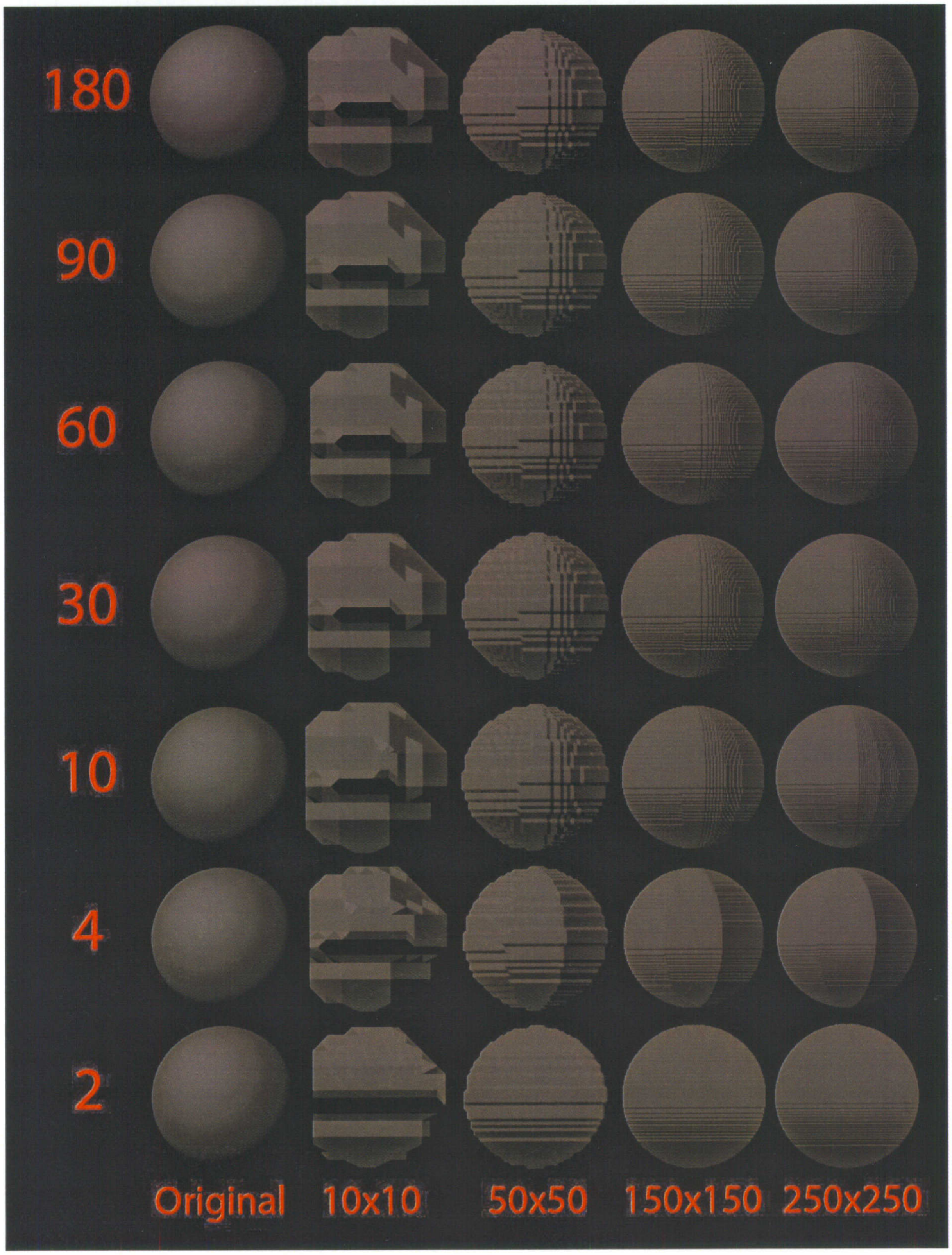


Figure 134: Comparison of reconstructed sphere models from an isometric front view.

At resolutions of 10x10 and 50x50, no additional information appears to be added by increasing the number of projections past 10. The 150x150 and 250x250 resolution reconstruction continue to benefit from additional projections until 30 projections are utilized.

4.5.1.2 Arrow Reconstruction

The arrow reconstruction showed similar results to the sphere. As seen in Figure 135 through Figure 140, the 10x10 resolution arrows do not gain any additional detail when using more than 4 projections. The 50x50 resolution arrow continues to be refined up until 10 projections. The 150x150 resolution image loses refinement at 30 projections and the 250x250 at 60 projections.

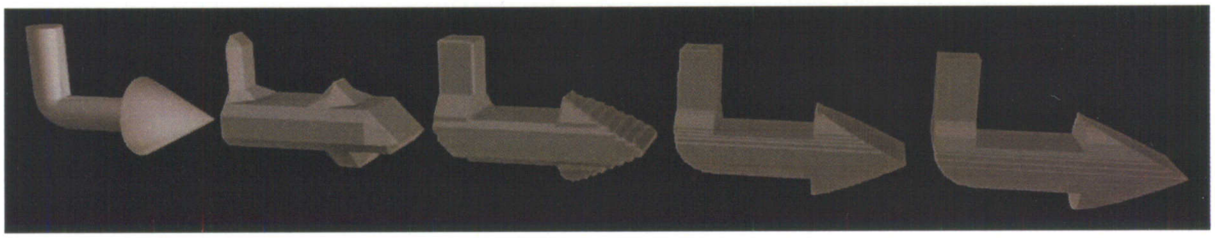


Figure 135: Original arrow versus 2 projections at 10x10, 50x50, 150x150 and 250x250.

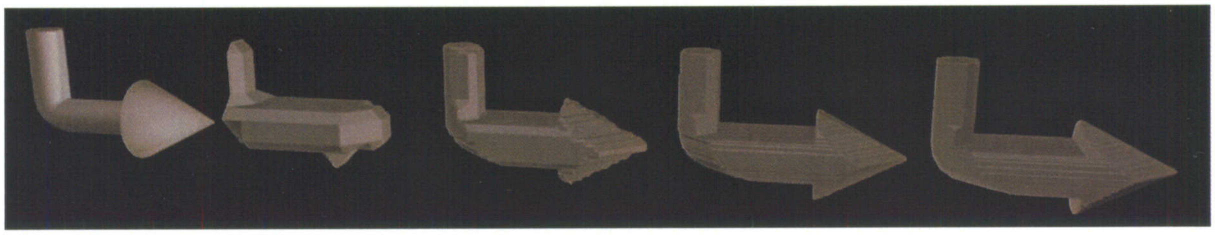


Figure 136: Original arrow versus 4 projections at 10x10, 50x50, 150x150 and 250x250.

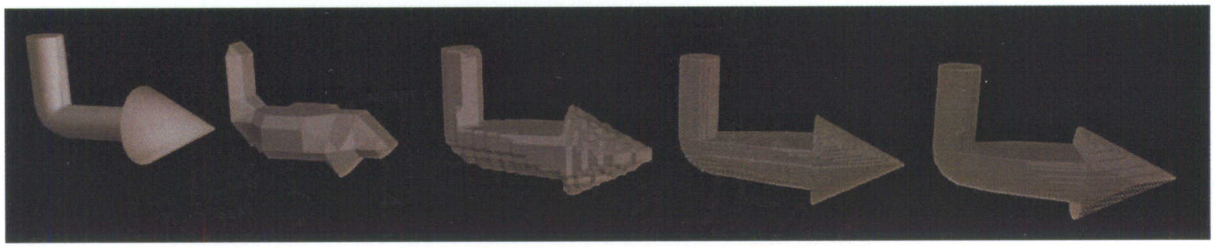


Figure 137: Original arrow versus 10 projections at 10x10, 50x50, 150x150 and 250x250.

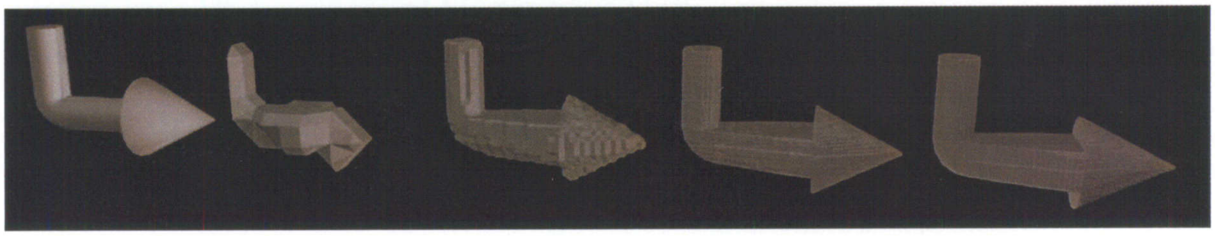


Figure 138: Original arrow versus 30 projections at 10x10, 50x50, 150x150 and 250x250.

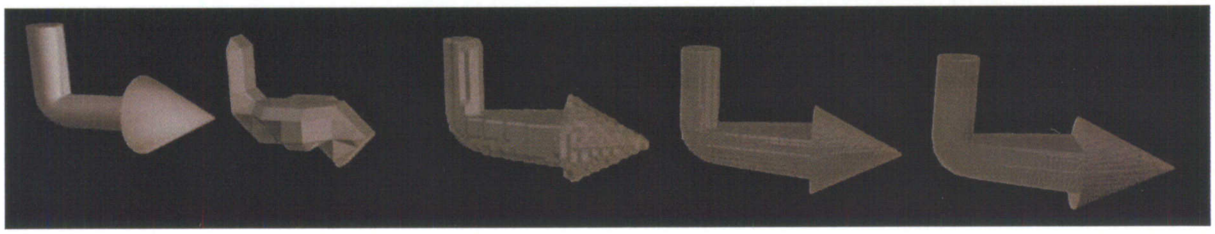


Figure 139: Original arrow versus 60 projections at 10x10, 50x50, 150x150 and 250x250.

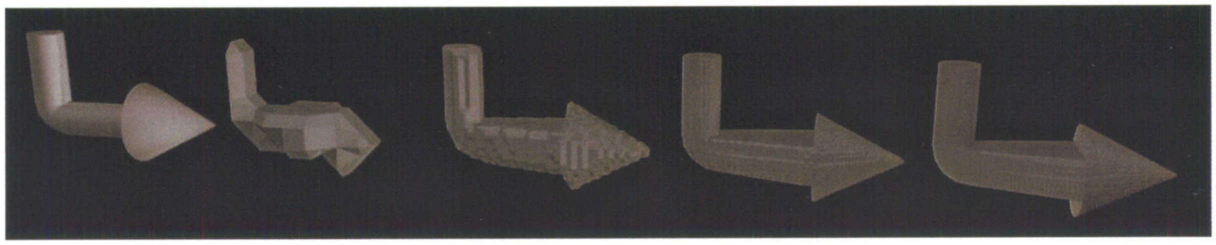


Figure 140: Original arrow versus 90 projections at 10x10, 50x50, 150x150 and 250x250.

Unlike the spheres, the arrows were continually refined past 30 projections. This is because the arrow's conical head continues to gain detail from additional projections.

This can be seen clearly in Figure 141 where all the arrows are shown for easy comparison.

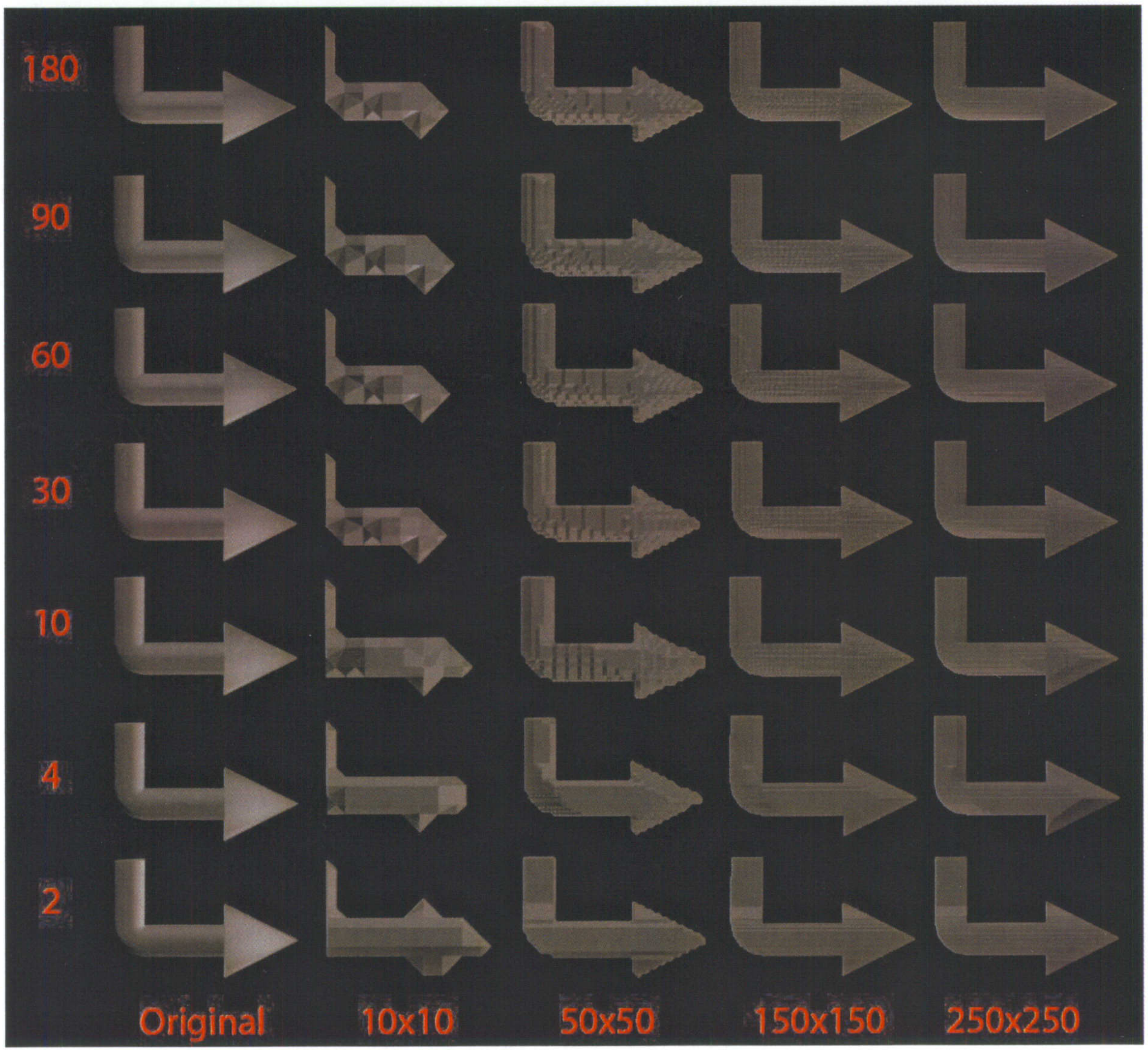


Figure 141: Comparison chart of the arrows at various resolutions and projections from a front isometric view.

However, there is one portion of the arrow that is unable to be resolved due to self-occlusion. As the arrow rotates, the head and tail block the cylinder, causing a loss of information. Whenever an object self-occludes, the covered portion will be lost when reconstructed. This is evident in the fanning out of the cylinder as it approaches the arrow head as seen in Figure 142. The red portion is the reconstruction while the blue portion is the original model.

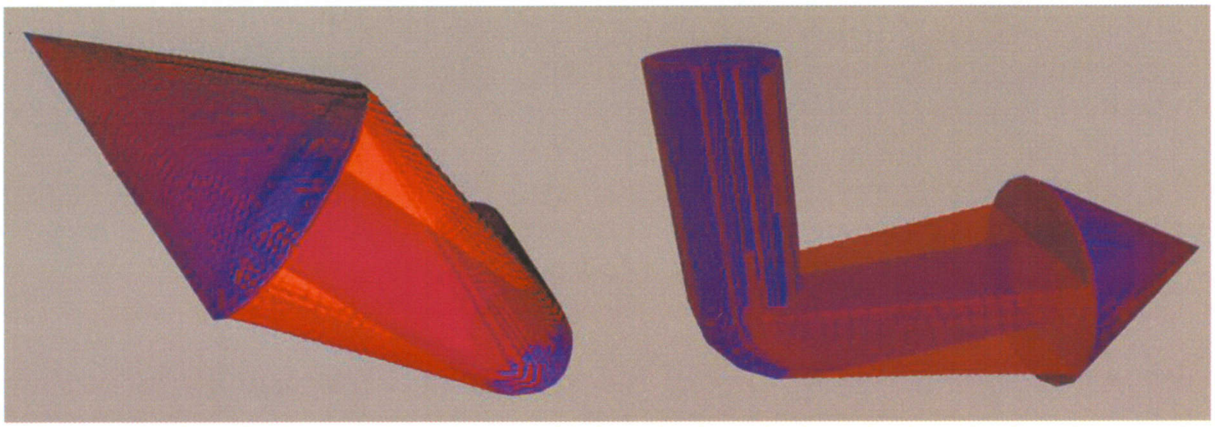


Figure 142: On overlaid view of the 250x250 180 projection model over the original.

Figure 143 shows three projections of the object as it rotates about the y-axis. Here it is clearly evident that the arrow head and tail occlude the cylindrical tail from any viewable position as it rotates. Only by viewing the image from another axis such as an overhead shot of the x-z plane would that information accessible. This is one limitation of rotating a particle about one axis. If the particle was rotated about the other two axes more information could be obtained to refine the object. For each additional axis the entire reconstruction algorithm would have to be run, drastically increasing the time.

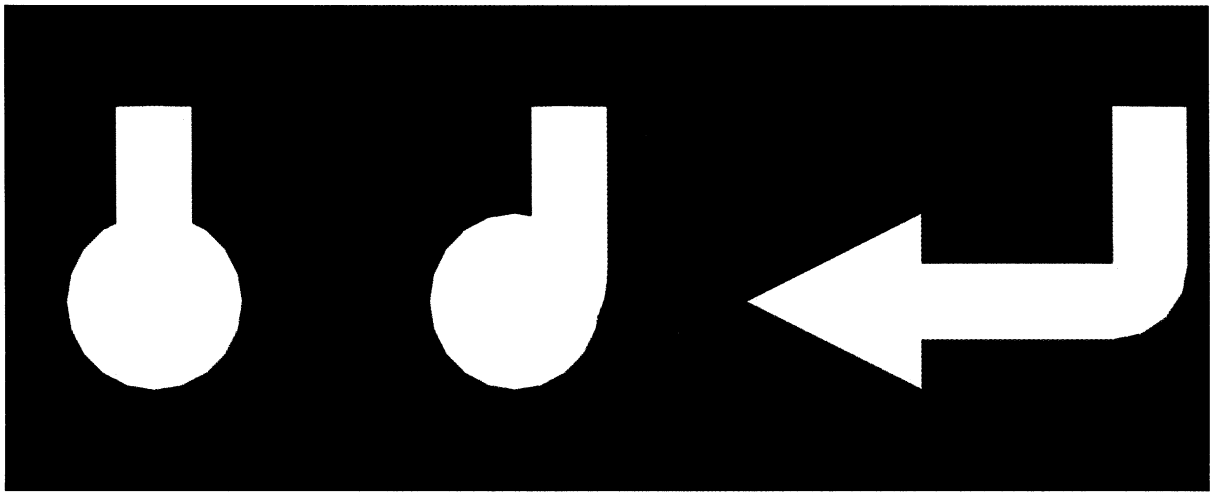


Figure 143: Three projections of an arrow as it rotates. The arrow head occludes information about the cylindrical tail.

4.5.1.3 Humanoid Reconstruction

The humanoid model posed several challenges to the reconstruction algorithm. The sphere was a symmetric object and only required a sufficient number of projections to recreate the smooth surface. The arrow introduced self-occlusion as well as hard edges. The humanoid possesses both hard edges in the hair and ears, large smooth surfaces defining the general body geometry and tiny details in the fingers and face.

Figure 144 shows the results compared to the original model in resolutions of 10x10, 50x50, 150x150 and 250x250 with only two projections of information. The higher resolution images are able to capture small details such as the fingers and hair. At the lowest resolution, hardly a humanoid shape can be discerned. Almost all of the limbs have been lost and what remains is a cylinder attempting to represent a torso.

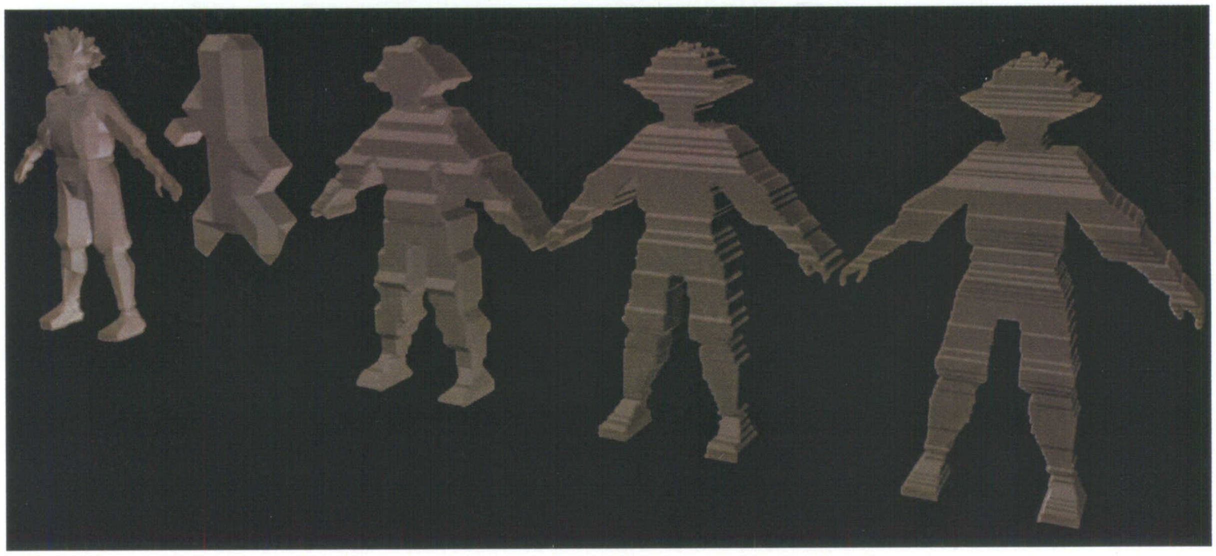


Figure 144: Original humanoid and 10x10, 50x50, 150x150 and 250x250 resolution models at 2 projections.

Figure 145 shows reconstructions using four projections. In this image it is possible to make out the cutting plane of the 45 degree projections on the front of the models at the intersection of the arms and chest. There are several artifacts that remain due to self-occlusion of the arms and legs. The lowest resolution model loses even more information as the additional projections cannot agree on the location of the limbs and are removed.

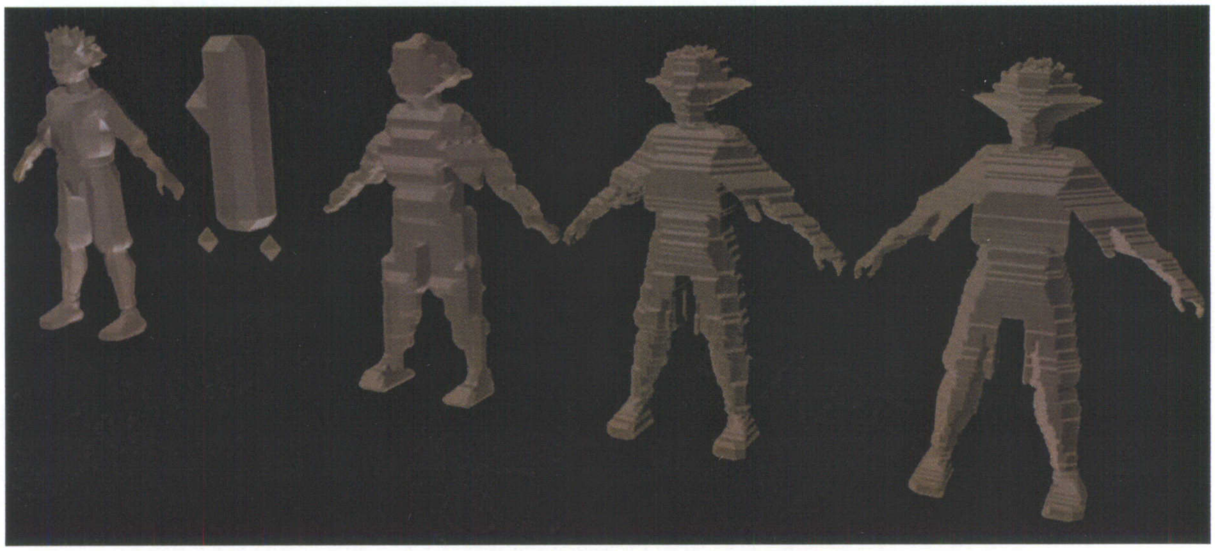


Figure 145: Original humanoid and 10x10, 50x50, 150x150 and 250x250 resolution models at 4 projections.

At ten projections, the overall shape of the humanoid can be easily discerned in Figure 146. Many of the artifacts caused by self-occlusion are missing from these models due to the additional projections cutting away excess material. The more curves an object has which are perpendicular to the camera's viewing plane, the larger the number of projections required to accurately capture the surface. This was clear with the sphere and also along the round areas of the body.

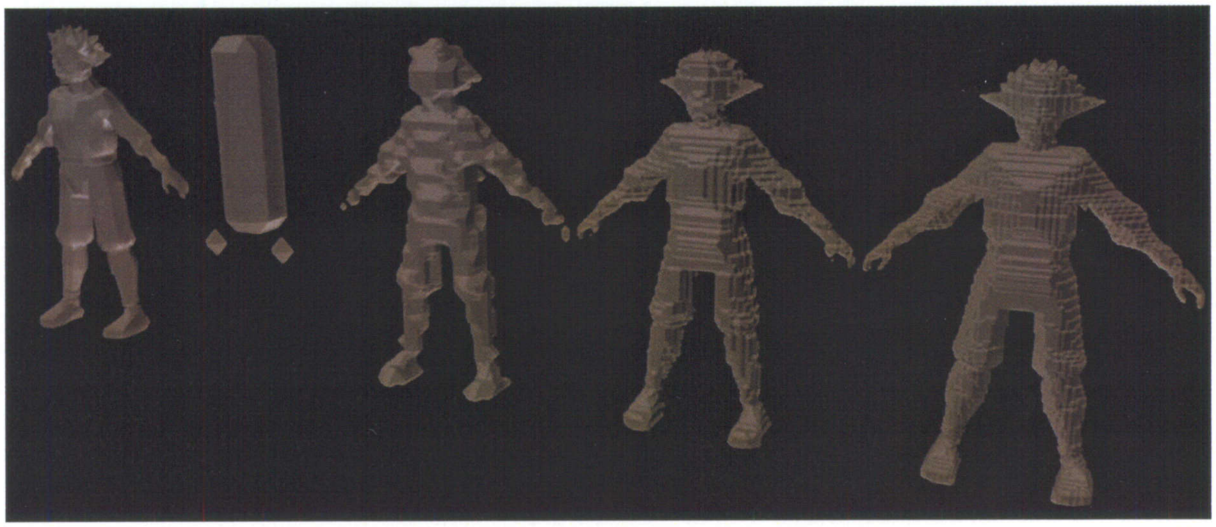


Figure 146: Original humanoid and 10x10, 50x50, 150x150 and 250x250 resolution models at 10 projections.

The 30 projection reconstructions continue to refine the curves of the object in Figure 147. For the 10x10 reconstruction, there is so little detail to utilize that the projections are only able to agree on the general shape of the object, making the entire humanoid into a simple beveled cylinder.

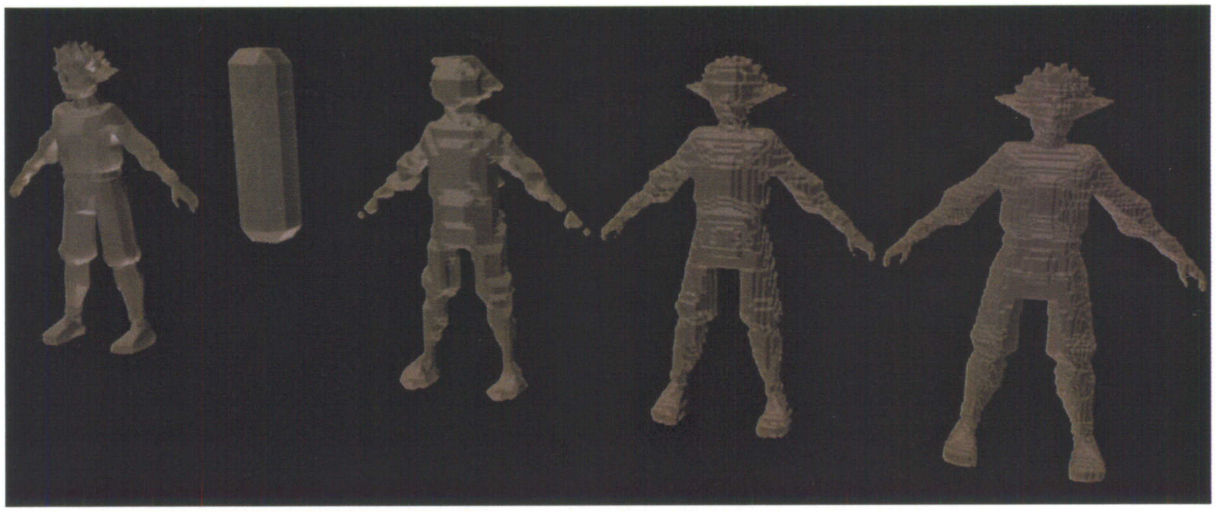


Figure 147: Original humanoid and 10x10, 50x50, 150x150 and 250x250 resolution models at 30 projections.

The next series of figures (Figure 148, Figure 149 and Figure 150) show the reconstructions at 60, 90 and 180 projections.

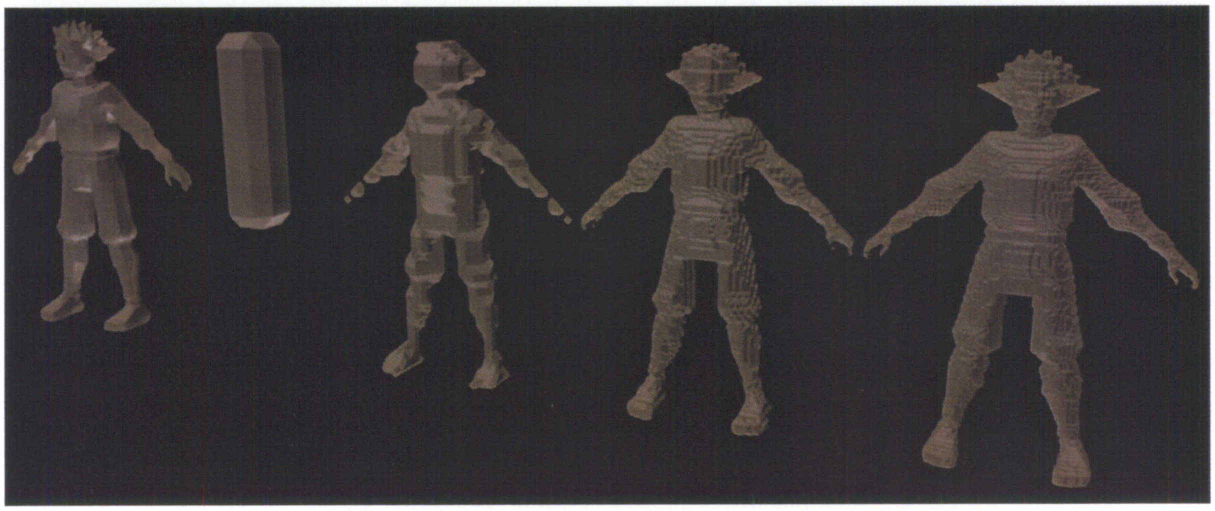


Figure 148: Original humanoid and 10x10, 50x50, 150x150, 250x250 resolution models at 60 projections.

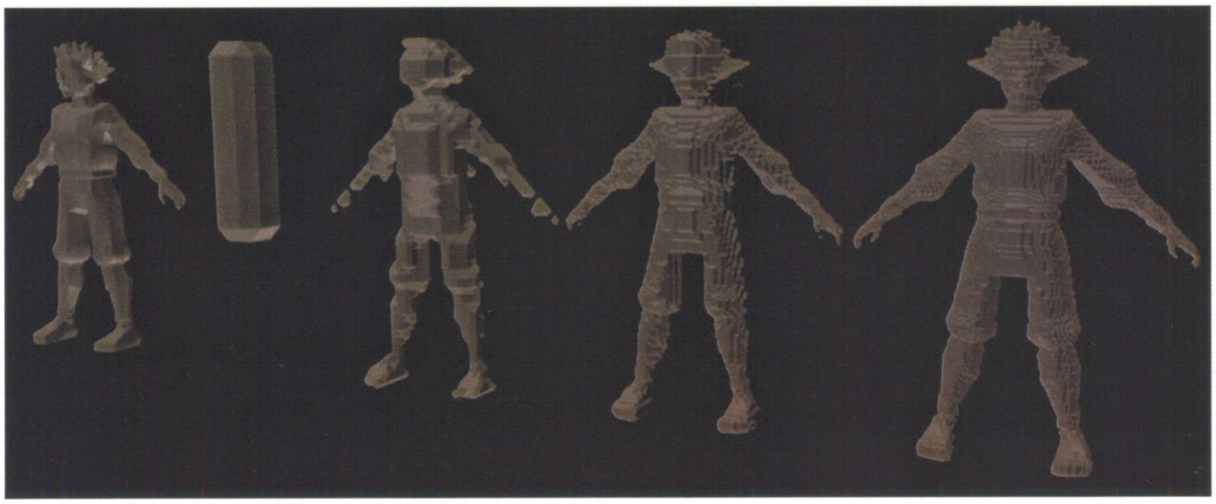


Figure 149: Original humanoid and 10x10, 50x50, 150x150, 250x250 resolution models at 90 projections

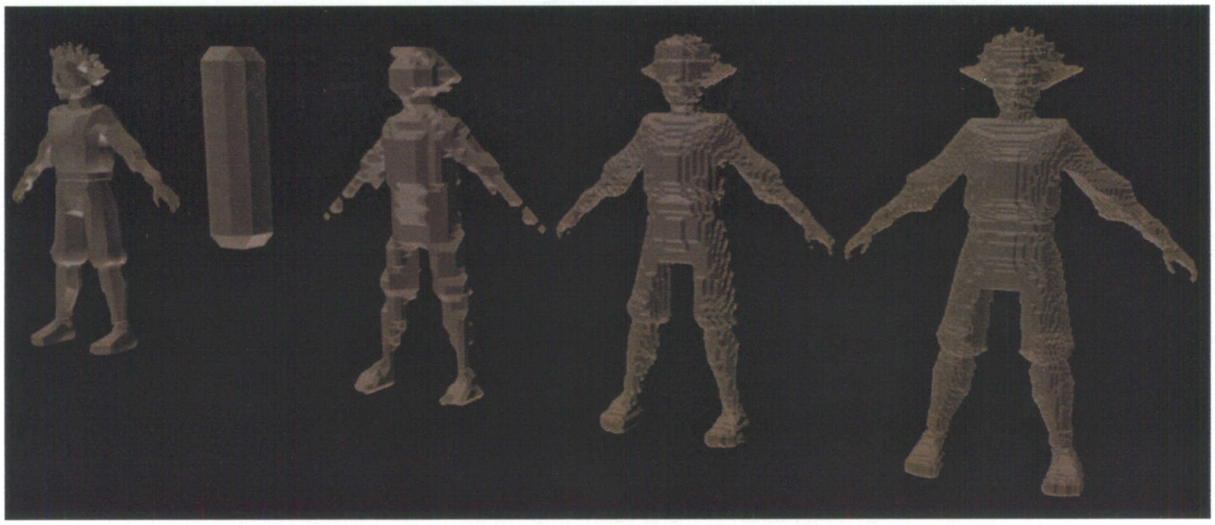


Figure 150: Original humanoid and 10x10, 50x50, 150x150, 250x250 resolution models at 180 projections.

After 60 projections, there are almost no changes in the shape of the models. This may be due to the fact that the reconstructions after 60 do not have the necessary resolution to incorporate additional projections. This is illustrated in Figure 151, where the resolution of the voxel grid is not at a high enough resolution to benefit from small increment projections.

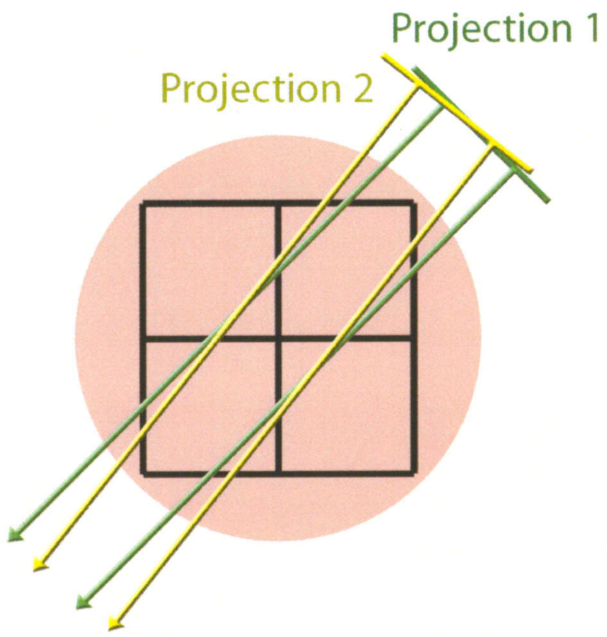


Figure 151: With small increments between projections and a low resolution, additional projections add little information to the final model. This can lead to higher projection reconstructions looking the same as lower projection reconstructions at the same resolution.

Comparing the final three projections of the humanoid side-by-side supports the idea that additional projections are not required in this instance. Figure 152 shows the 180, 90 and 60 degree projections next to one another with no noticeable changes to the geometry. Careful attention to the facetted vertices of the model at each resolution show that there is no change in the final reconstructed model regardless of whether 180, 90 or 60 projections were utilized. This is true across all of the resolutions from 10x10 to 250x250.

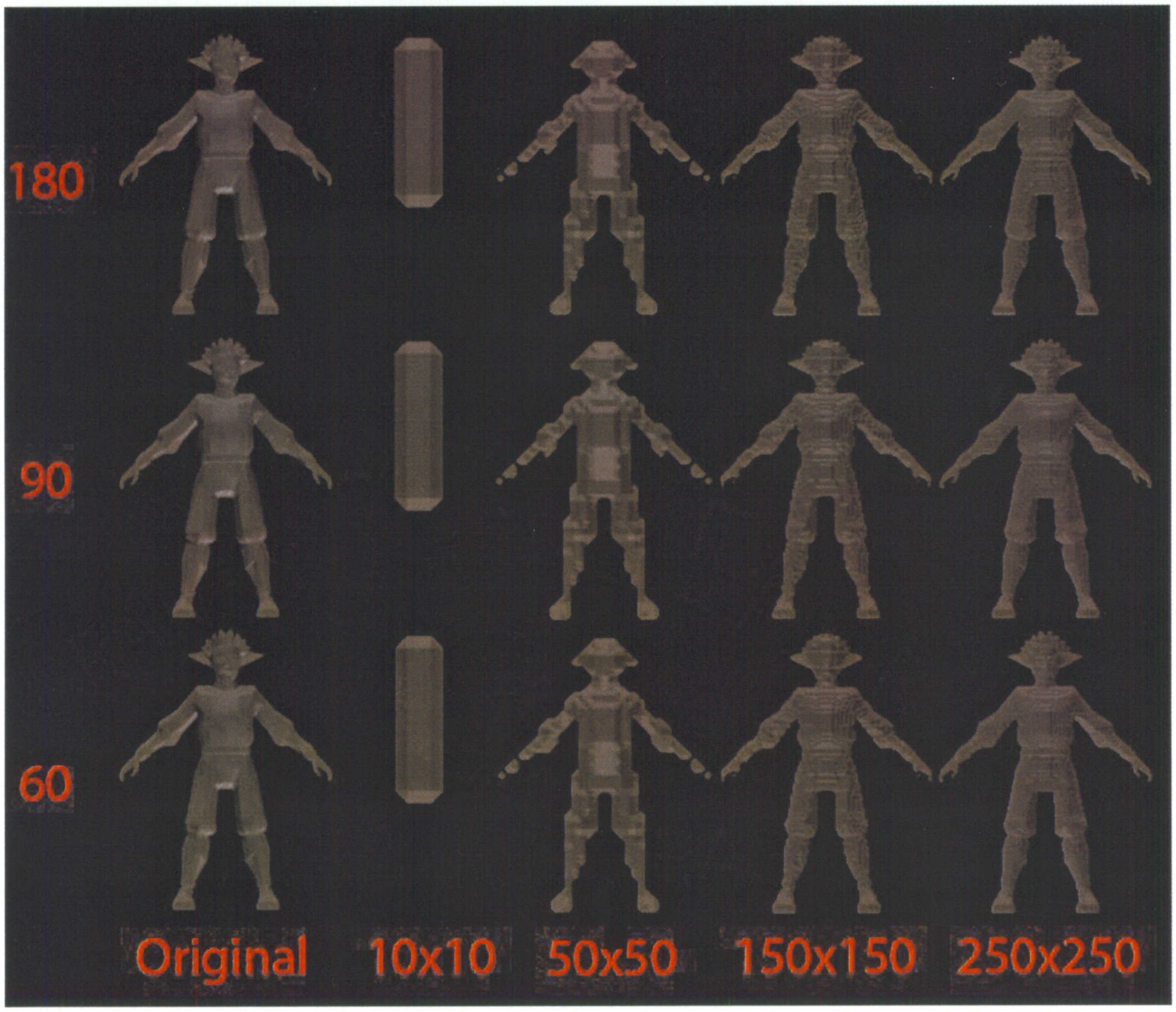


Figure 152: Humanoid comparison chart of 180 projections, 90 projections and 60 projections with the original model, 10x10 resolution, 50x50 resolution, 150x150 resolution and 250x250 resolution.

Figure 153 shows the models with resolutions between 10x10 and 250x250 pixels. In each progression of these images there is a clear refinement of the model. Working from bottom to top, each reconstruction begins to chisel out the shape of the model.

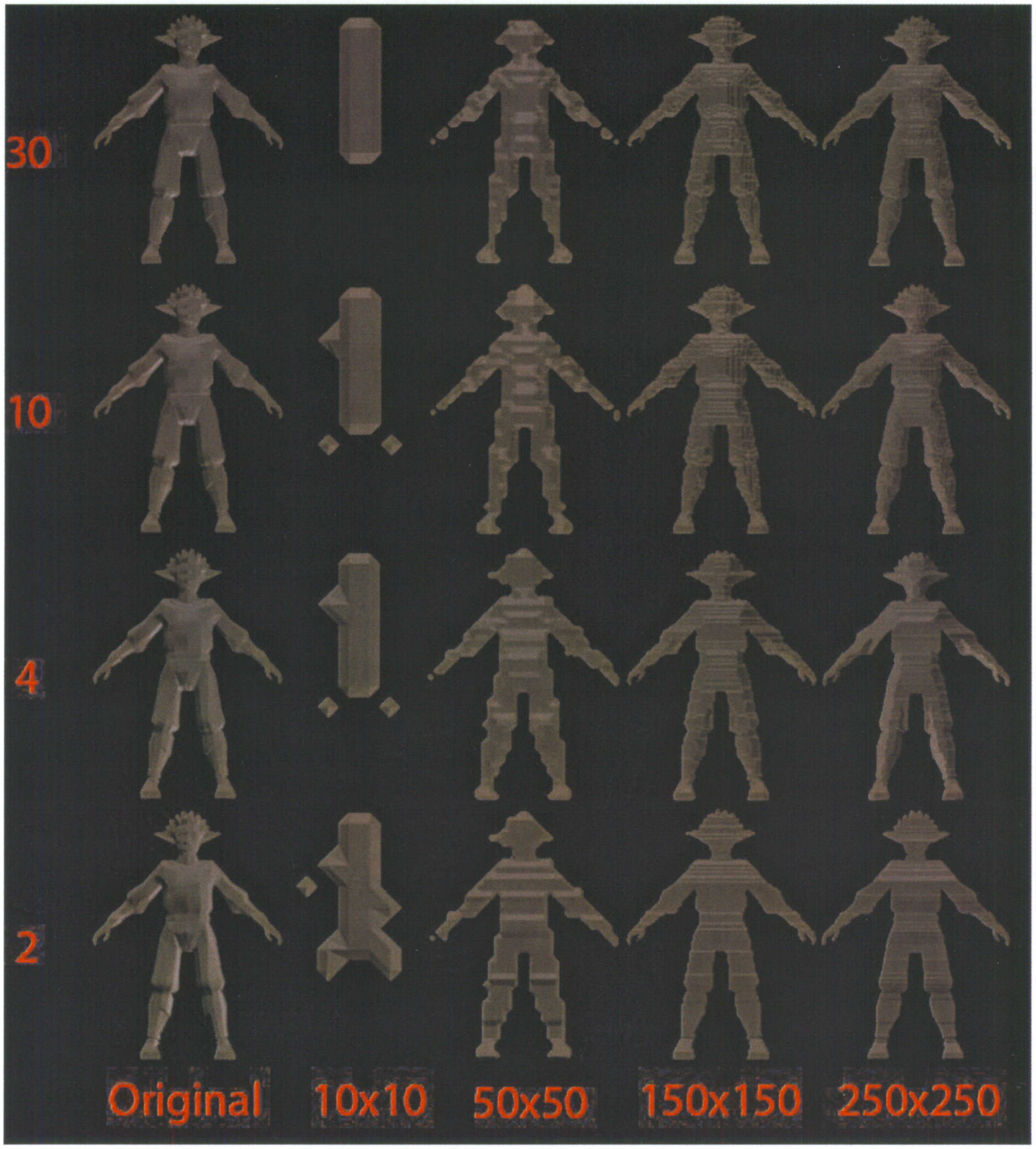


Figure 153:Humanoid comparison chart of 30 projections, 10 projections, 4 projections and 2 projections with the original model, 10x10 resolution, 50x50 resolution, 150x150 resolution and 250x250 resolution.

There are issues with this new reconstruction algorithm. A magnified comparison of the highest resolution model next to the original model can be seen in Figure 154.

This reconstruction algorithm utilizes images that quantize an object into discreet blocks. It will never be possible to attain the smooth and simple original model through this method. The best that can be achieved is a higher resolution with a higher polygon count that approximates the original surface.

Also, many details are lost in the algorithm. As previously mentioned, self-occluded objects can cause reconstruction problems. Any concave section of an object will pose challenges to this reconstruction algorithm. This is clearly seen in the close up along the face and chest region.

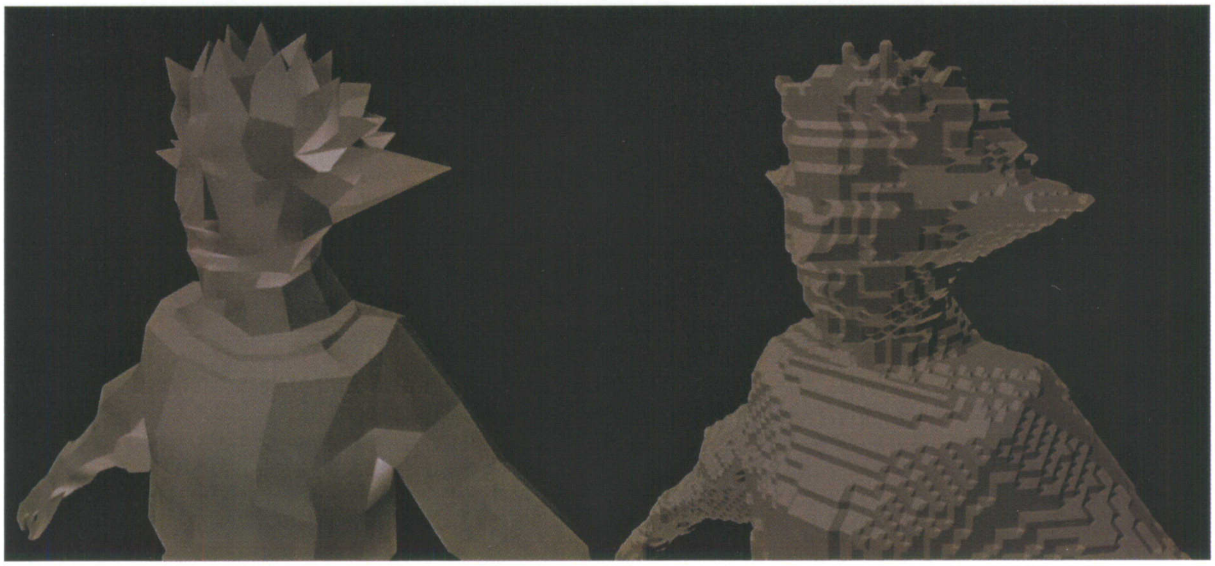


Figure 154: Magnified comparison of original model to reconstructed model.

The following series of figures help to demonstrate the differences between the original model and the reconstructed model in terms of volume. It is difficult to have a numeric value which indicates how wrong a reconstruction model is compared to the original model. These images are an attempt to visually show the differences at different levels of projections with the highest available resolution of 250x250 pixels.

Figure 155 compares the original model in red to the reconstruction model with 2 projections in blue. There is significant excess volume resulting from the use of only two projections.

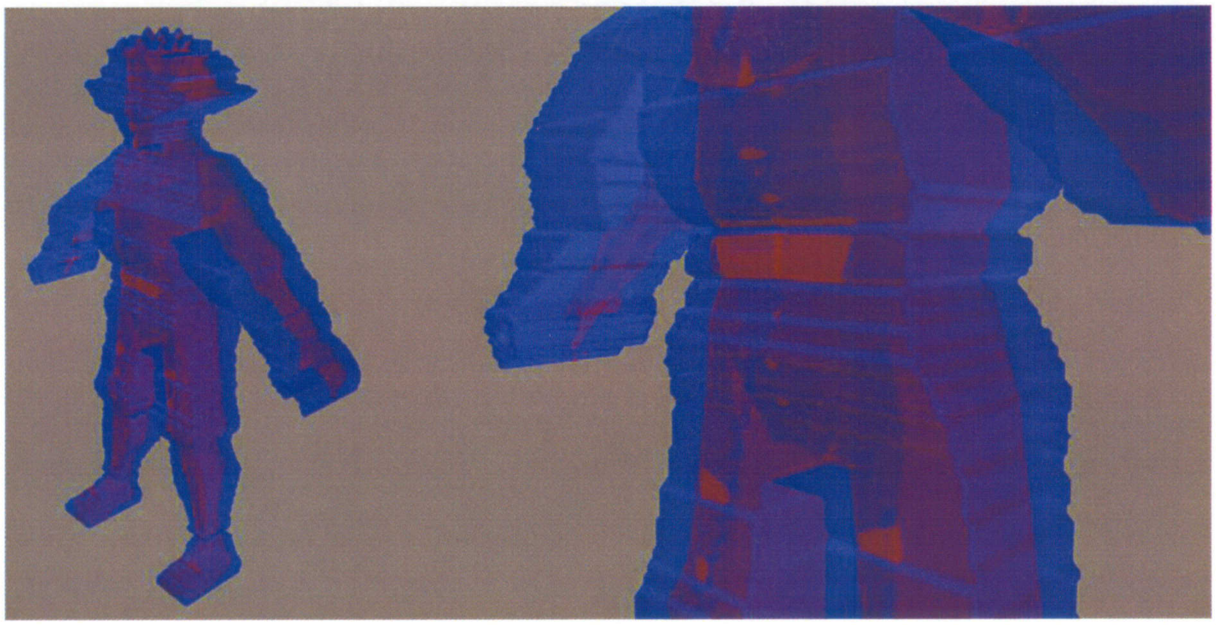


Figure 155: Overlaid comparison of original model to the 250x250 resolution at 2 projections model.

Figure 156 shows the result of the original model overlaid with a 4 projections reconstruction. By doubling the number of projections, there has been a significant amount of data culled from the model. However, once again self-occlusion has caused several segments between the arms and legs to persist.

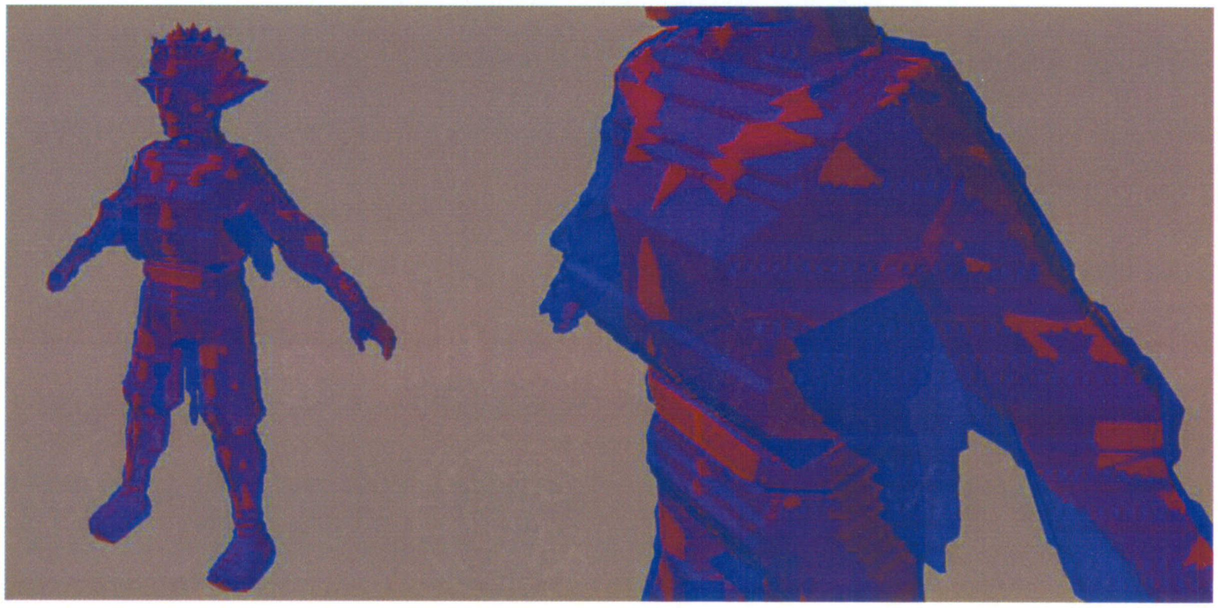


Figure 156: Overlaid comparison of original model to the 250x250 resolution at 4 projections model.

With 10 projections, most of the excess material between the arm and chest has been culled away as seen in Figure 157.

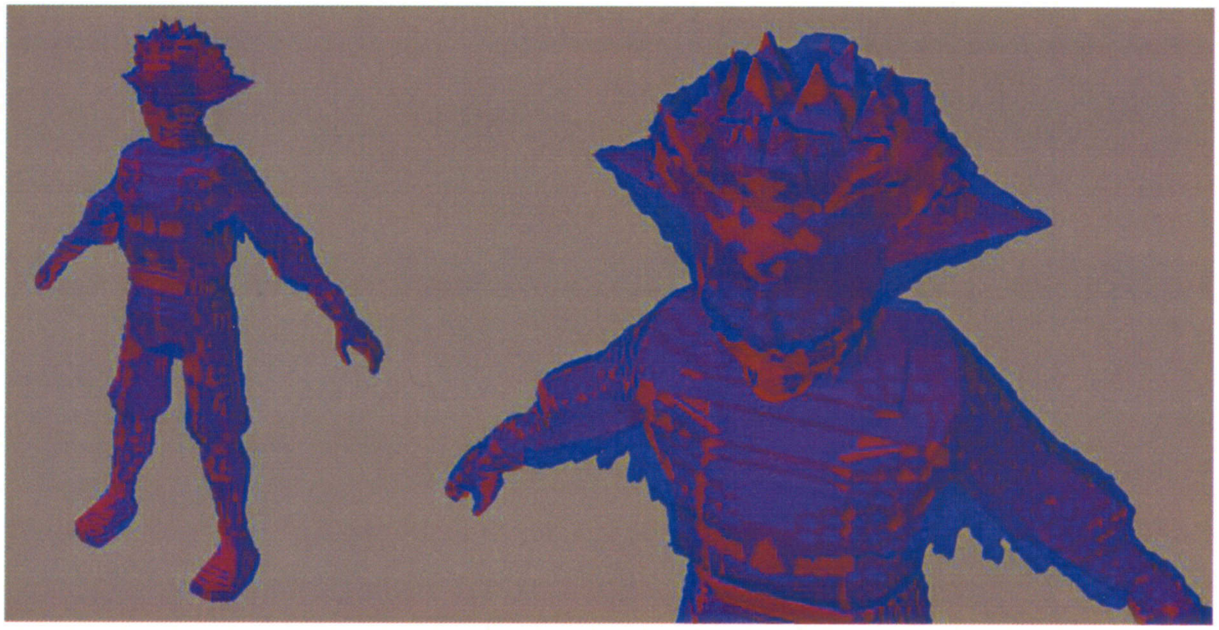


Figure 157: Overlaid comparison of original model to the 250x250 resolution at 10 projections model.

The final projection level resulting in changes to the model can be seen in Figure 158. The 30 projections have removed nearly all excess information from the reconstruction. Several noticeable areas are around the head due to the sharp topology and self-occlusion.

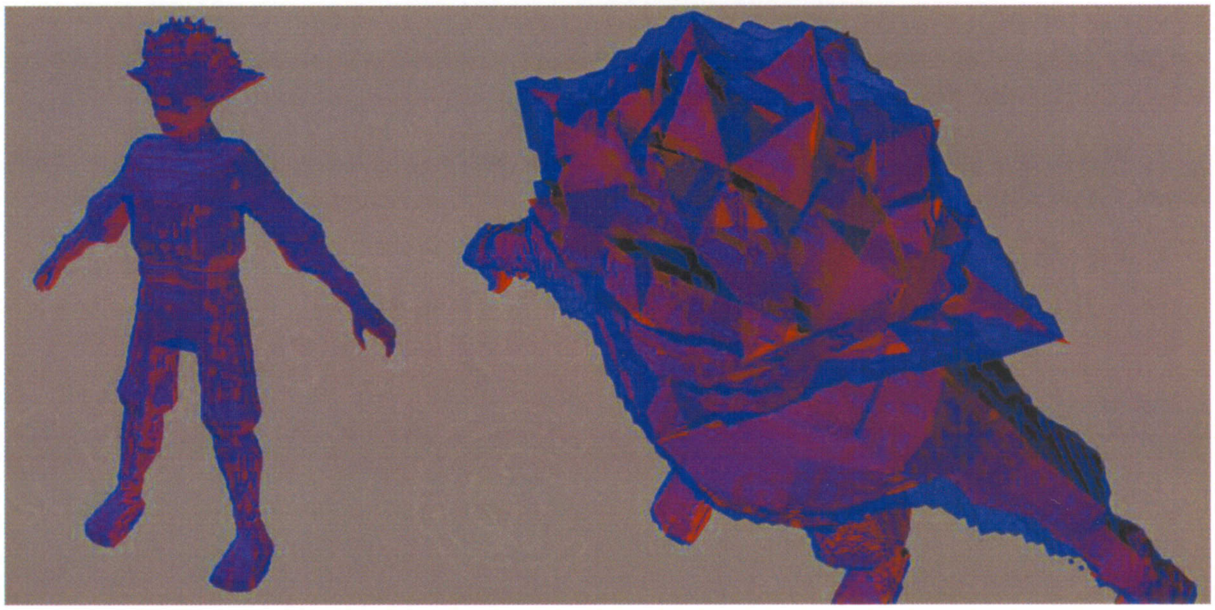
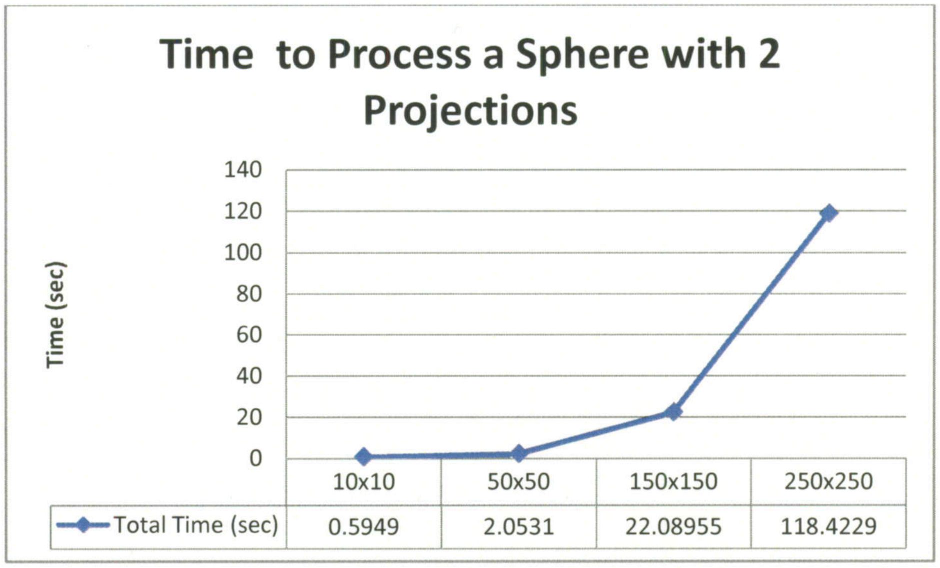


Figure 158: Overlaid comparison of original model to the 250x250 resolution at 30 projections model.

4.5.1.4 Processing Times

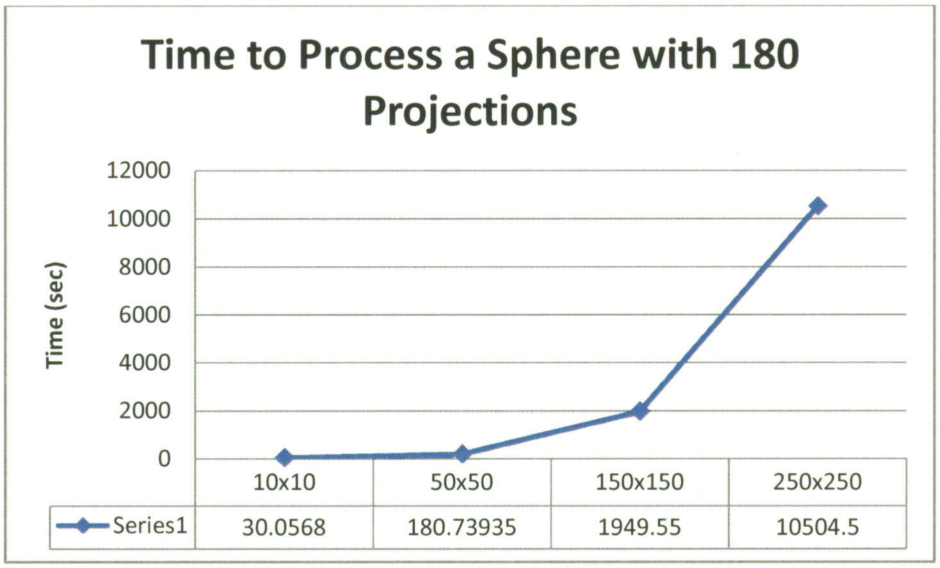
Figure 159 shows the time to process a sphere reconstruction with only 2 projections. The fastest of the reconstructions can be accomplished in half of a second; however this yields the poorest quality. With each increase in resolution a larger amount of time must be dedicated to the algorithm.



	10x10	50x50	150x150	250x250
Total Time (sec)	0.5949	2.0531	22.08955	118.4229

Figure 159: The time to process a sphere using 2 projections at 90 degree increments.

This trend is the same regardless of how many projections are utilized. Figure 160 shows the time trials for a sphere reconstruction at 180 projections and the same general increase in processing time.



	10x10	50x50	150x150	250x250
Series1	30.0568	180.73935	1949.55	10504.5

Figure 160: The time to process a sphere using 180 projections at 1 degree increments.

Plotting this on a base ten logarithmic scale produces a nearly linear graph following an exponential progression as shown in Figure 161.

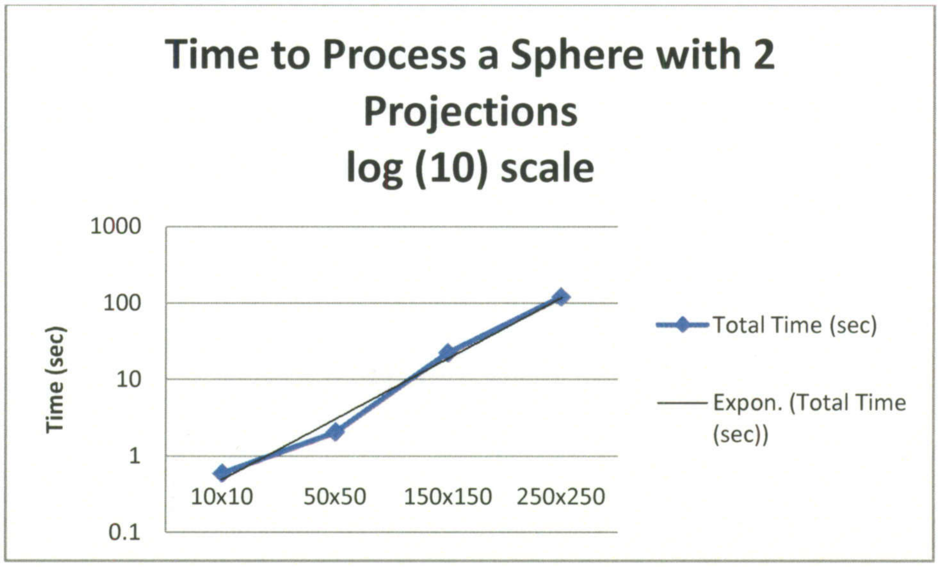


Figure 161: The time to process a sphere using 2 projections at 90 degree increments on a logarithmic scale with an exponential trend line.

Plotting the 180 projection data on a base ten log scale creates an even more linear shape, nearly matching the exponential trend line. The reason for the slight discrepancy between the two reconstructions is from the differences in the computational speeds of each run. Every time the algorithm is processed a slightly different time is taken. Over a small time step of a half a second, this can result in small differences having a large effect on the shape of the graph. For the longer reconstructions utilizing 180 projections and taking a minimum of 30 seconds, this has little impact.

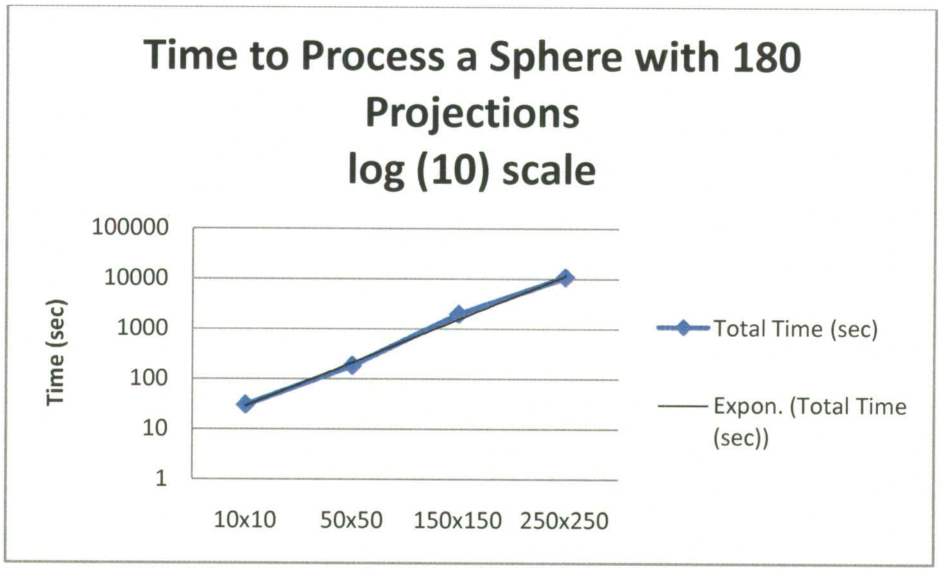


Figure 162: The time to process a sphere using 180 projections at 1 degree increments on a logarithmic scale with an exponential trend line.

The next series of figures (Figure 163, Figure 164, Figure 165 and Figure 166) provide the results of time trials for each of the three models. Regardless of the type of model being reconstructed the final time is nearly the same. For any model being created, regardless of the complexity of the model, the algorithm time will be the same.

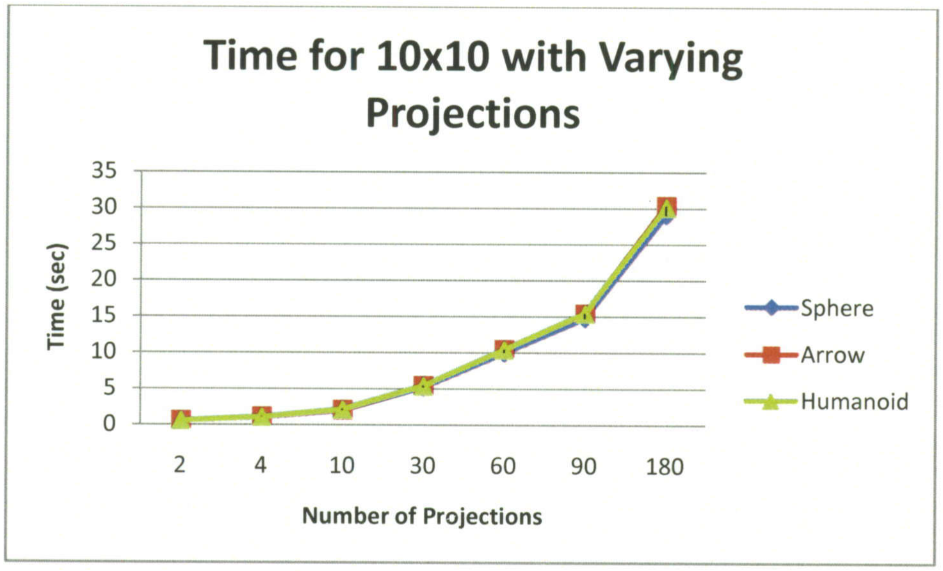


Figure 163: The time to process three models using 10x10 images of various projections.

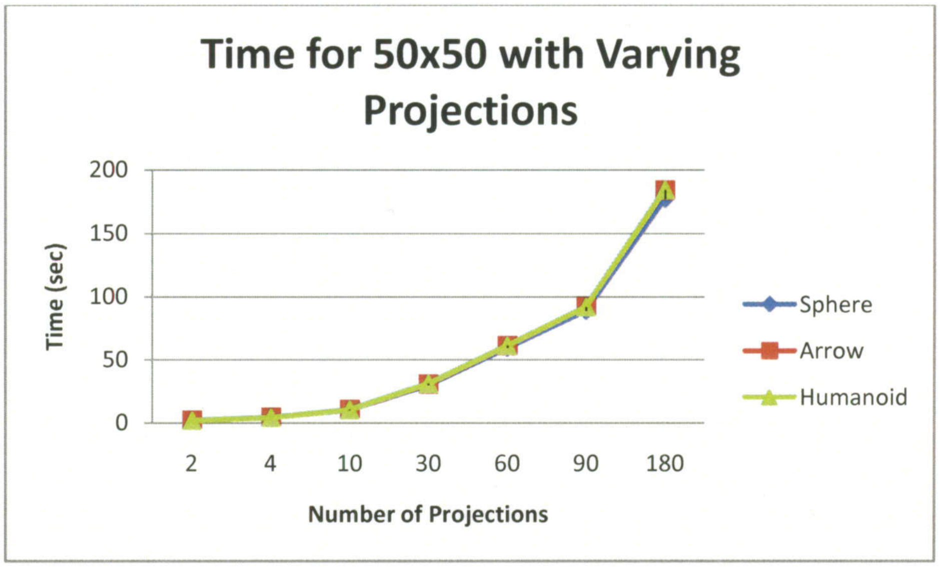


Figure 164: The time to process three models using 50x50 images of various projections.

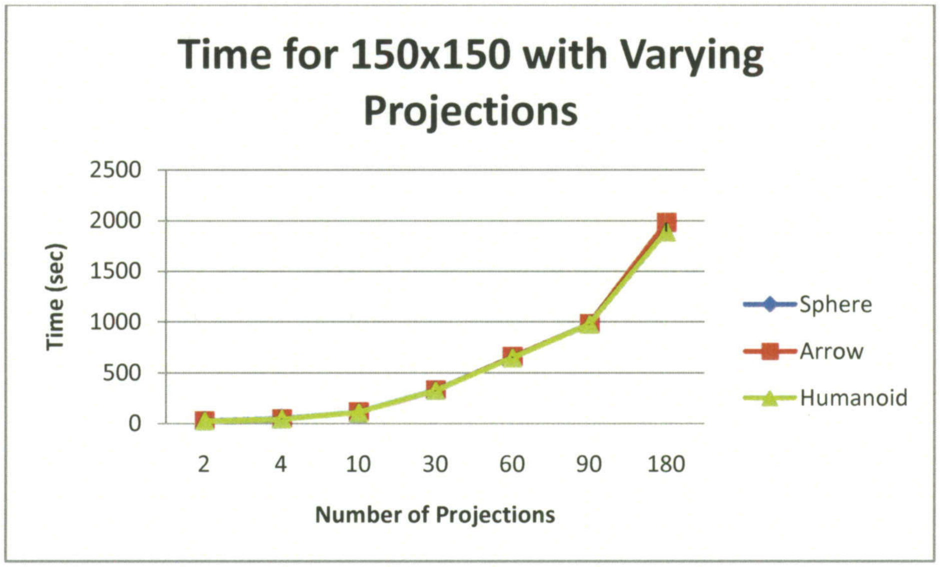


Figure 165: The time to process three models using 150x150 images of various projections.

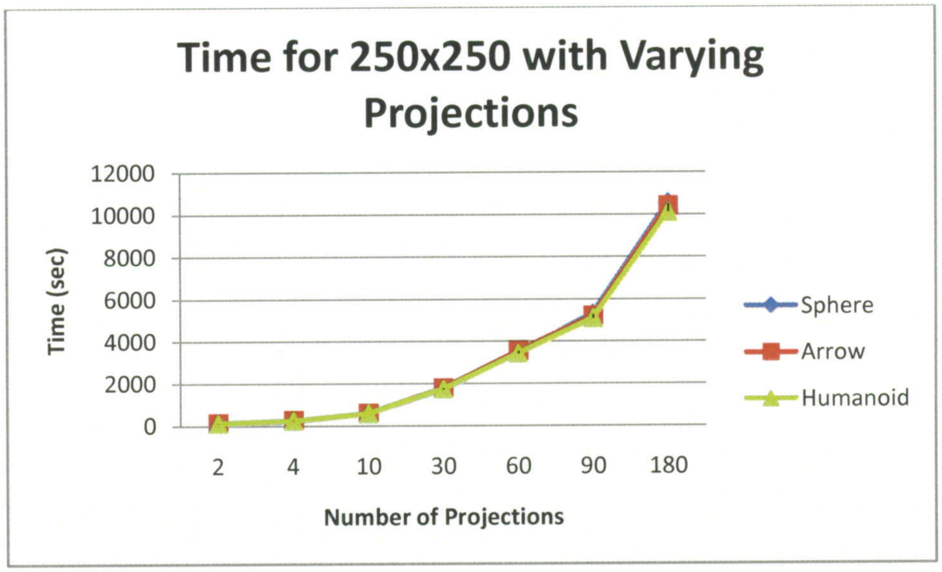


Figure 166: The time to process three models using 250x250 images of various projections.

A base ten log plot for 10x10 resolution projections shows a similar exponential increase in the time to reconstruct a model as seen in Figure 167.

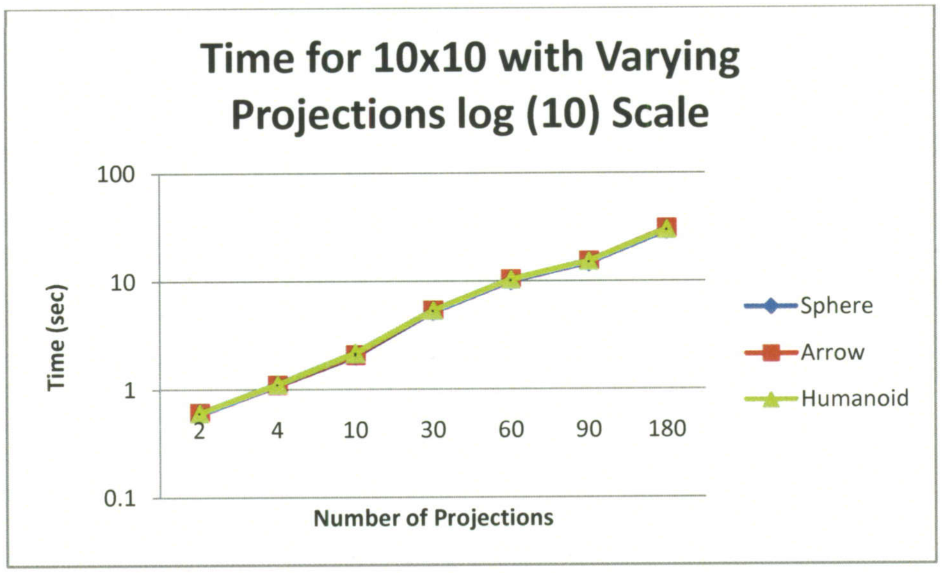


Figure 167: Time for 10x10 resolution reconstruction at varying projections on a base ten log scale.

These time trials show that there is an exponential increase as the number of projections or the resolution is increased. However, the switch from a lower resolution model to a higher resolution model takes significantly more time than increasing the number of projections. Based upon the synthetic data shown here, a resolution of approximately 150x150 with 30 projections will in most cases produce a suitable reconstruction model in a minimum of time.

4.5.1.5 3-D Fourier Descriptors of Synthetic Models

The following ellipsoid plots were created for the sphere and arrow reconstructed models. The resolution was held constant in each plot and contains seven ellipses showing the increase in projections from 2 to 180.

Figure 168 shows a 3-D isometric view of the three descriptors calculated from the sphere reconstruction at a resolution of 50 pixels. The first three ellipsoids of 2, 4, and 10 projections can clearly be seen as the blue, red and green ellipses, respectively. Distinguishing the ellipses of 30, 60, 90 and 180 projections is more difficult in this three dimensional view.

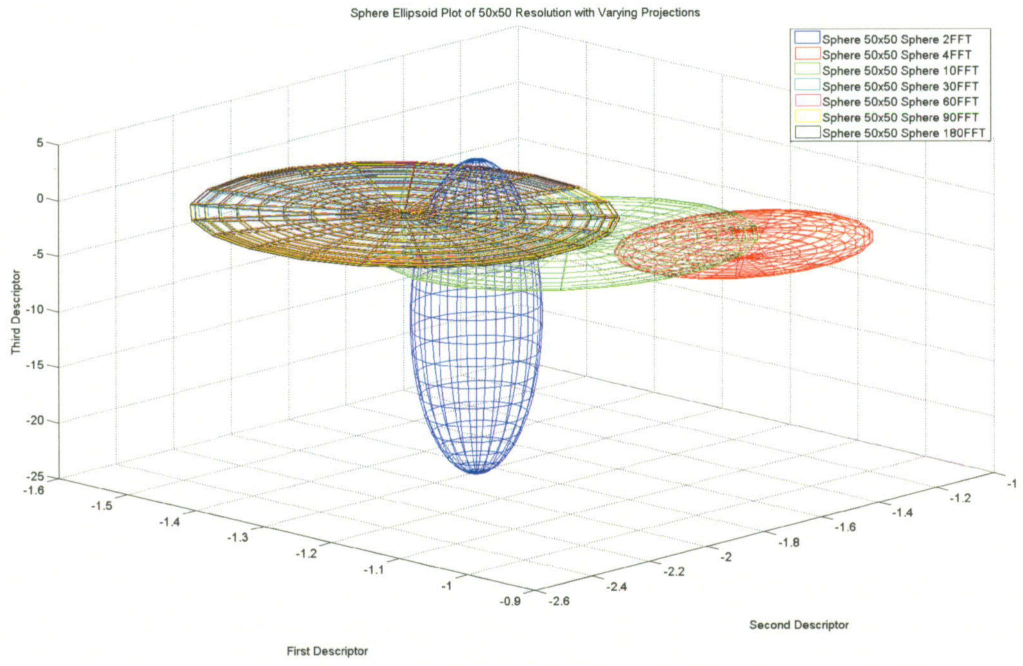


Figure 168: 3-D shape descriptors of 50x50 pixel resolution spheres.

Figure 169 compares the first and third descriptor values. Progressing down the list of ellipses yields a decrease in the third descriptor decreases until you arrive at 30 projections or more. Here, the variance is so small the ellipses appear as flat lines stacked in the same location.

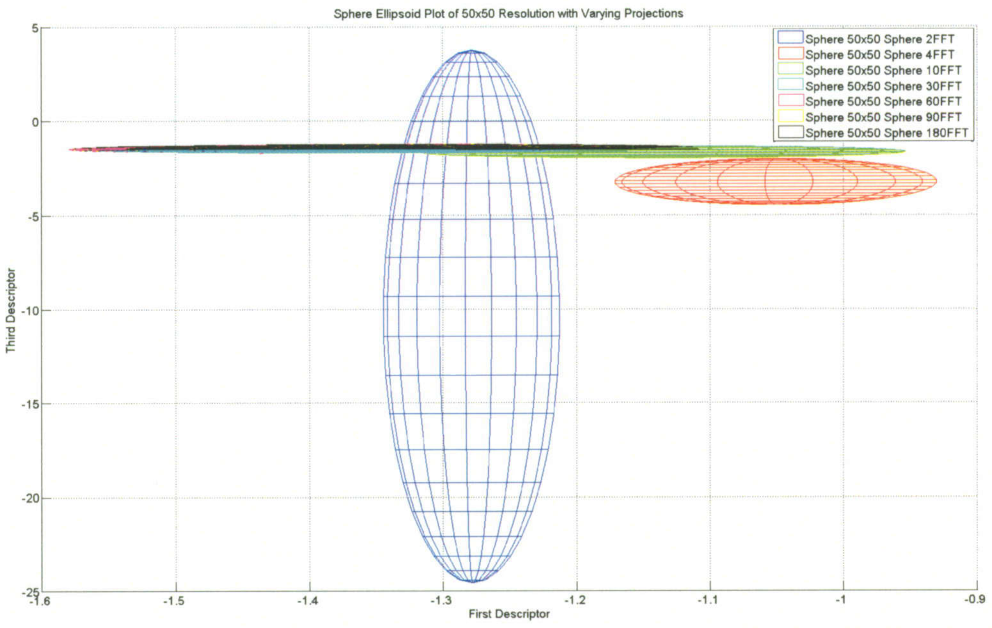


Figure 169: Third and first 3-D shape descriptors of 50x50 pixel resolution spheres.

Figure 170 compares the first and second descriptors. From this view, a distinction between the mean values of the different projections can be observed. While there is definitely some variation between projections above 10, the question becomes: is it worth the extra processing time to gain a slight refinement in variance? For each increase in the number of projections, the time to reconstruct is doubled. The decrease in variance is hardly distinguishable at hardly appears beneficial at this low resolution.

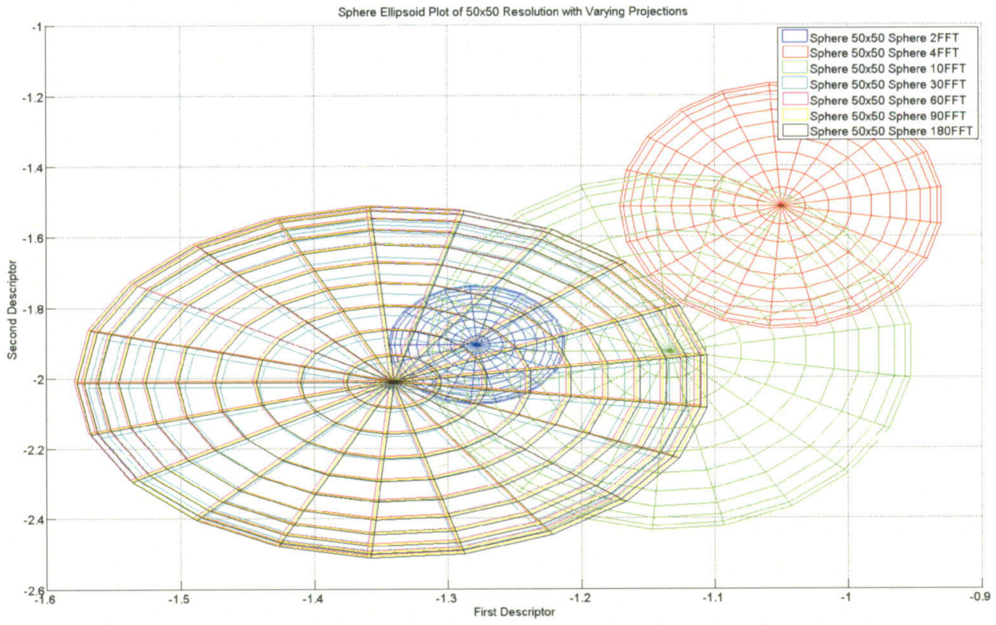


Figure 170: Second and first 3-D shape descriptors of 50x50 pixel resolution spheres.

Figure 171 compares seven ellipsoids at a resolution of 150x150 pixels. There is a distinct difference in mean and variance between a sphere reconstructed with 2 projections, a sphere reconstructed with 4 or 10 projections and spheres reconstructed with 30 projections or more.

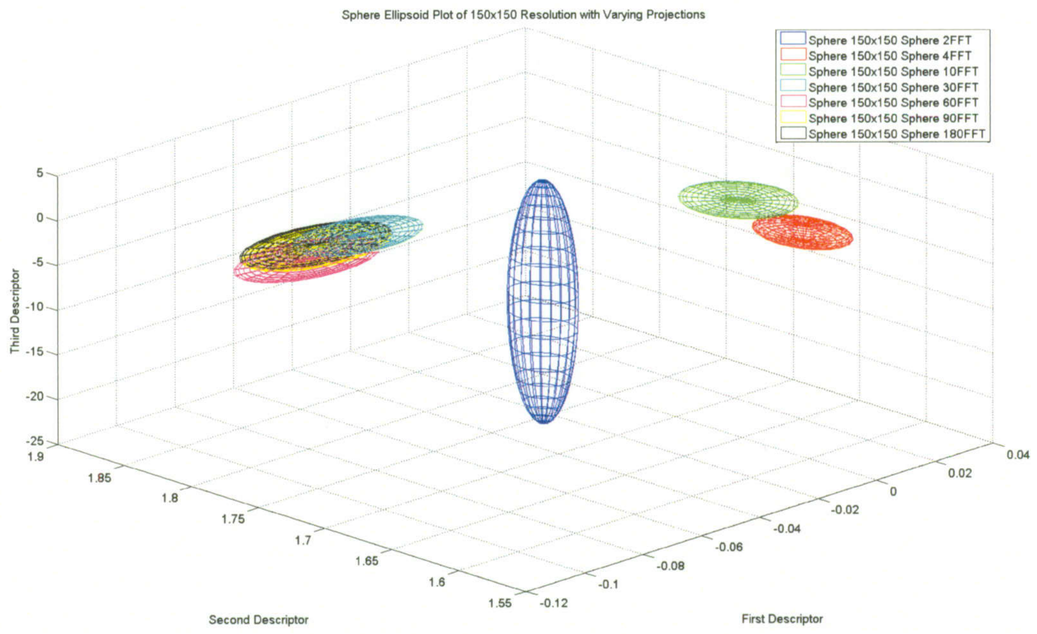


Figure 171: 3-D shape descriptors of 150x150 pixel resolution spheres.

Figure 172 is a plot of the third and first descriptor for a 150x150 sphere. Just as before with the 50x50 resolution images, the variance of the third descriptor decreases after utilizing 10 projections.

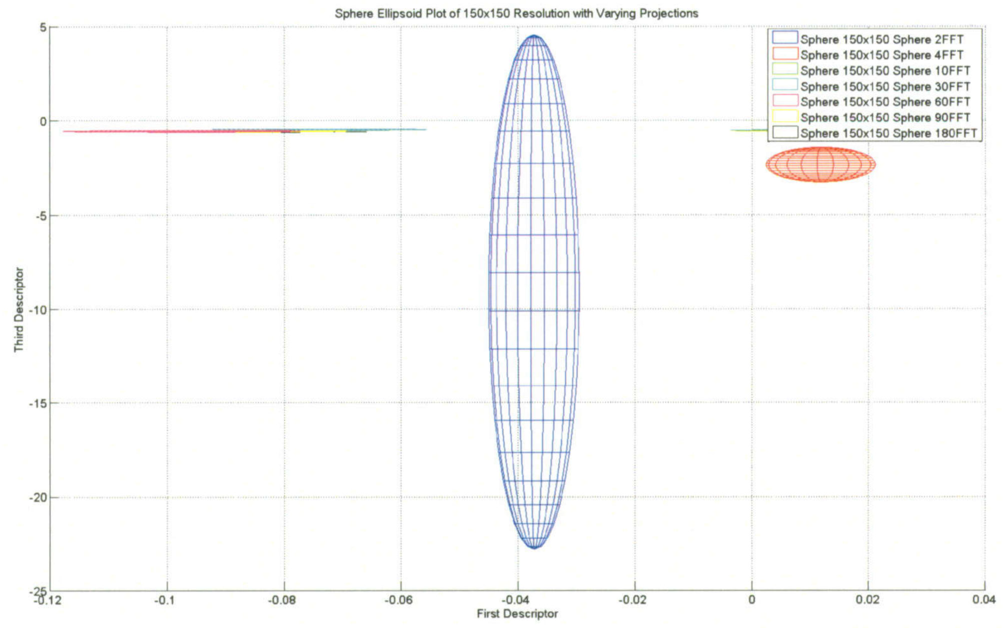


Figure 172: Third and first 3-D shape descriptors of 150x150 pixel resolution spheres.

Figure 173 is a plot of the second and first descriptors, which show a clustering of the 4 and 10 projection sphere and those of at least 30 projections or more. With a higher resolution image, additional refinement of the variance can be seen up until 90 projections. However, 90 and 180 projections fall almost completely on top of one another. This concludes that using 180 projections is a wasteful choice when at a resolution of 150x150 pixels.

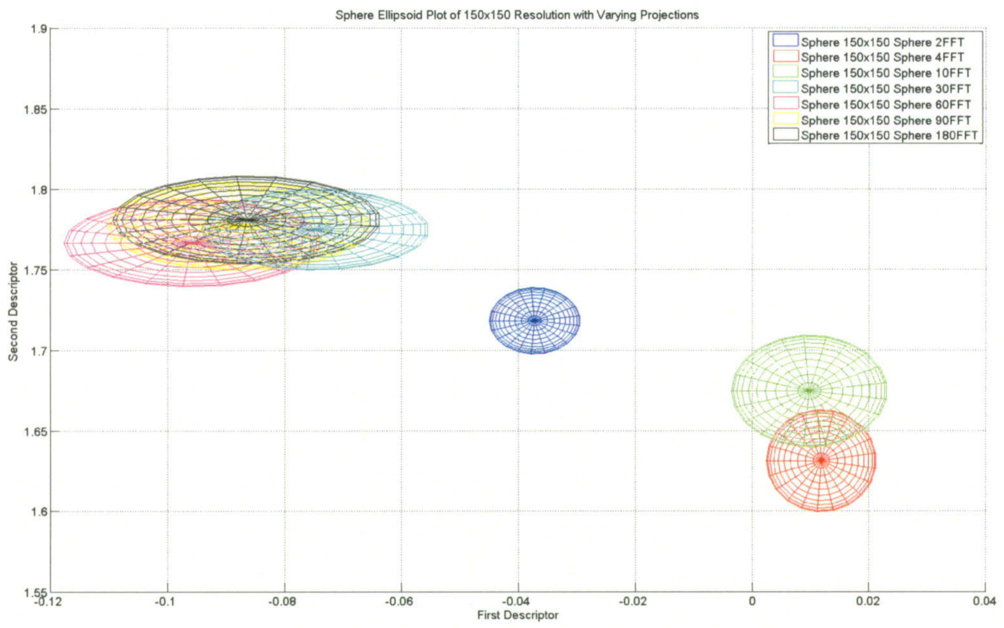


Figure 173: Second and first 3-D shape descriptors of 150x150 pixel resolution spheres.

Figure 174 is the final resolution tested with the spheres at 250x250 pixels. This resolution provided ellipsoids with the most variance between each of the projections. Clear differences between the yellow and black ellipses can be seen, indicating that 180 projections can be beneficial when sampling at such a high resolution. However, this means a significant amount of time will have to be spent on the reconstruction.

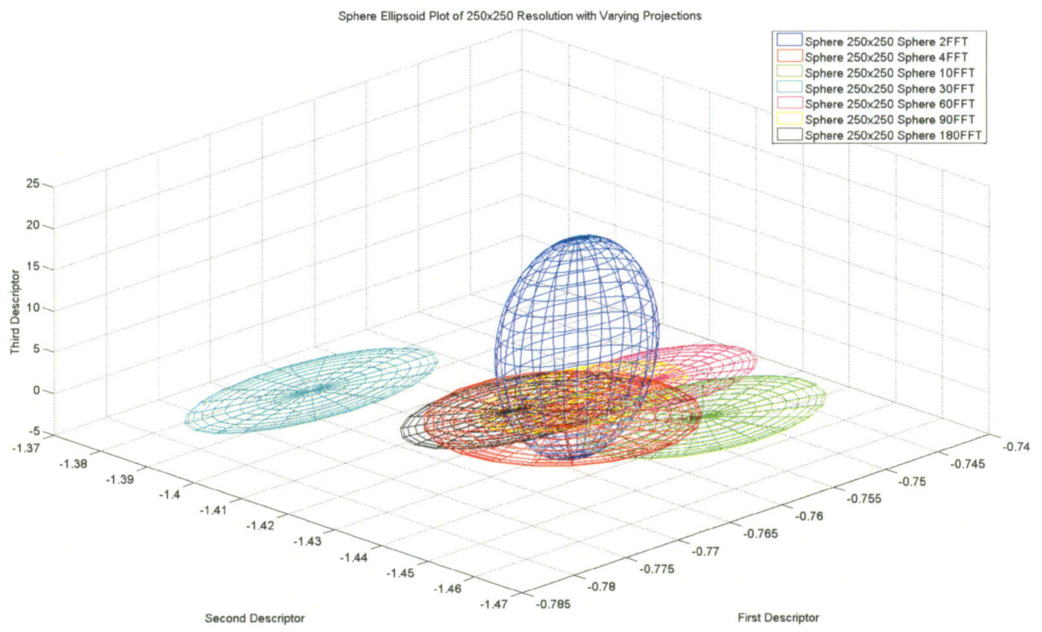


Figure 174: 3-D shape descriptors of 250250 pixel resolution spheres.

Figure 175 shows the third and first descriptors and the eventual drop in variance to nearly zero. A significant drop in variance can be seen when moving from 2 to 4 projections.

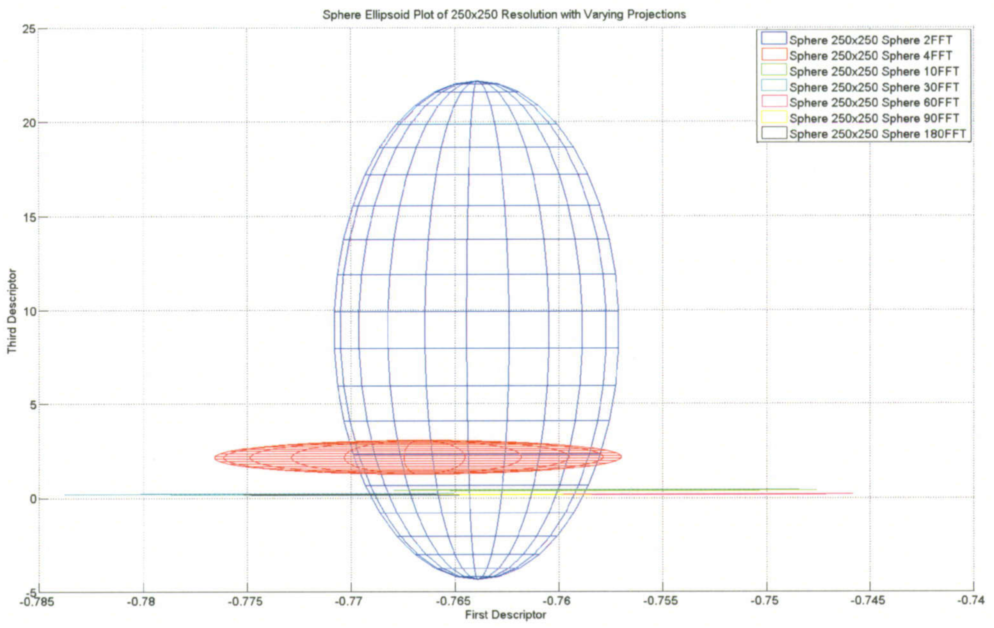


Figure 175: Third and first 3-D shape descriptors of 250x250 pixel resolution spheres.

Figure 176 is significant in that it depicts the first and second descriptors, which show the deviation of ellipsoids as resolution increases. Distinct differences in mean (ellipsoid center) and variance (ellipsoid size) can be noticed when moving down the legend all the way up to a full 180 degrees of rotation. However, an important point to notice is the continually shrinking scale of the first and second descriptor axes. While the ellipsoids are more distinct as one moves up in resolution we are also dealing with finer and finer scales (variances). This means that the differences might only be distinct due to the narrowing of the range we are viewing for the 1-3 descriptors. To human eye, these differences are indistinguishable as previously shown in the visual comparisons.

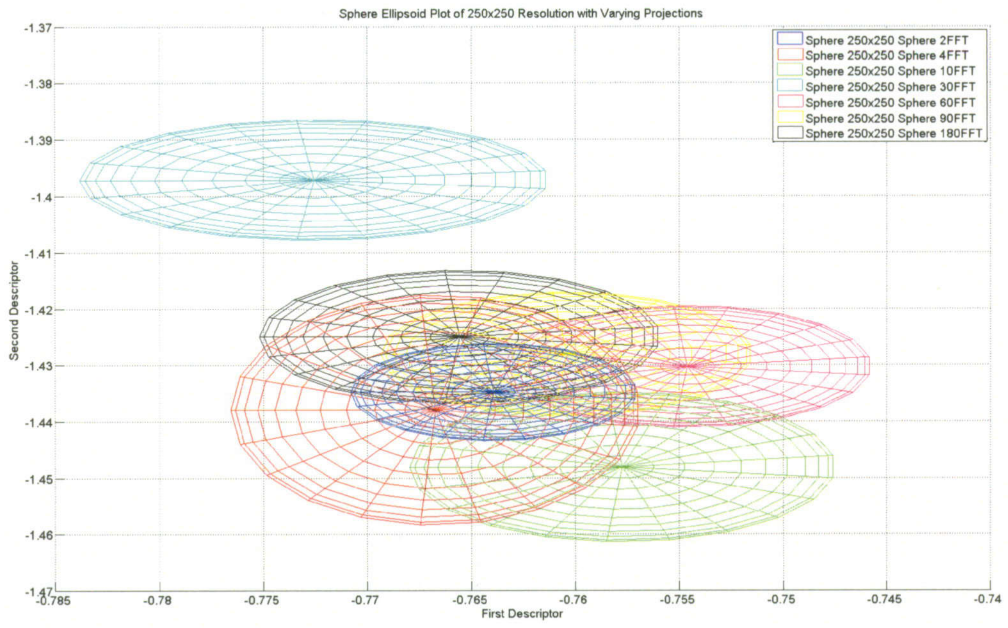


Figure 176: Second and first 3-D shape descriptors of 50x50 pixel resolution spheres.

The reconstructed arrows were a more complex shape than the aforementioned spheres. Due to this complexity, and overall difference in geometry from projection to projection, we will see a difference in the descriptors. Figure 177 is the 3-D plot of all projections tested at a resolution of 50x50 pixels. From this initial image it can be seen that all of the projections capture similar data as they have a similar mean value. The major difference between the ellipses is in the variance identified by the ellipsoid size.

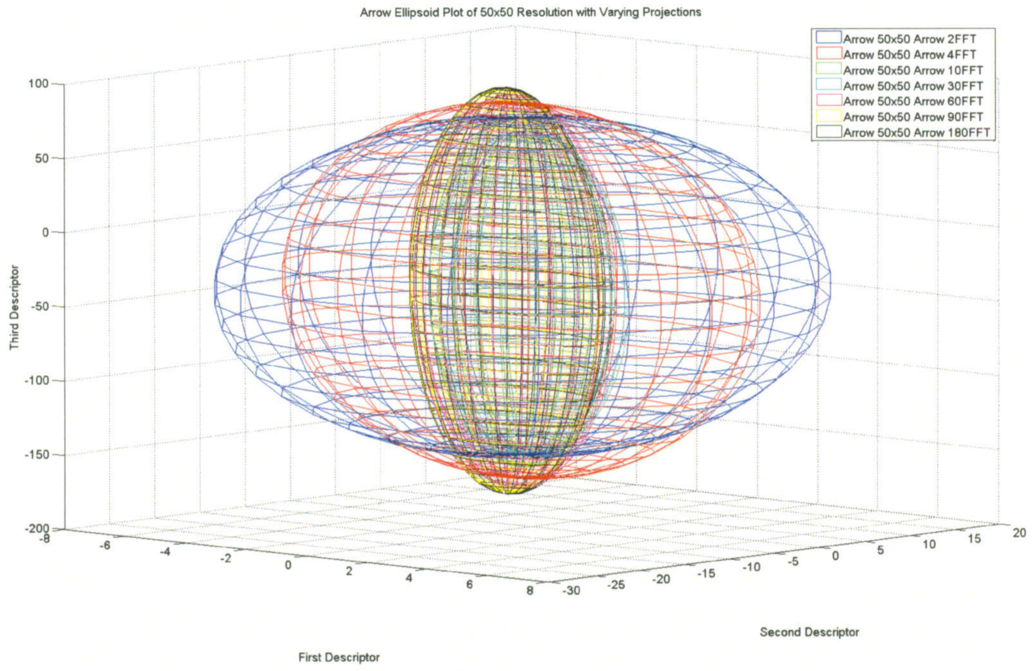


Figure 177: 3-D shape descriptors of 50x50 resolution arrows.

Figure 178 is a plot of the third and first descriptor. The value of the third descriptor stays relatively constant across all projections. However, a clear reduction in the variance across the first descriptor can be seen. The first descriptor remains approximately constant after the 10 projection ellipse.

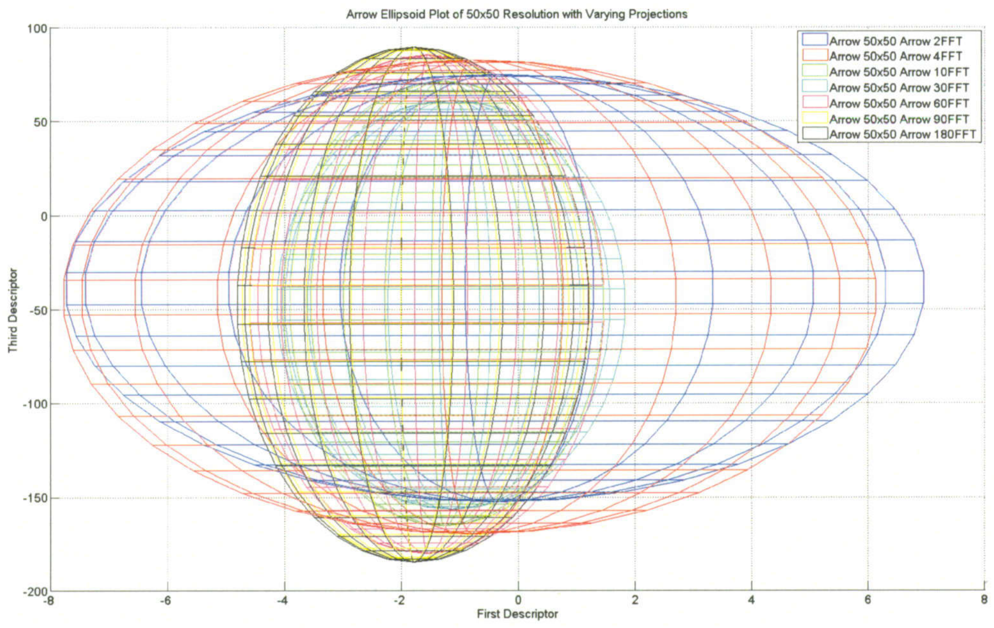


Figure 178: Third and first 3-D shape descriptors of 50x50 resolution arrows.

Figure 179 shows the reduction in the variance across the first and second descriptor. As with the sphere at a resolution of 50x50 pixels, little benefit is gained from using more than 10 projections.

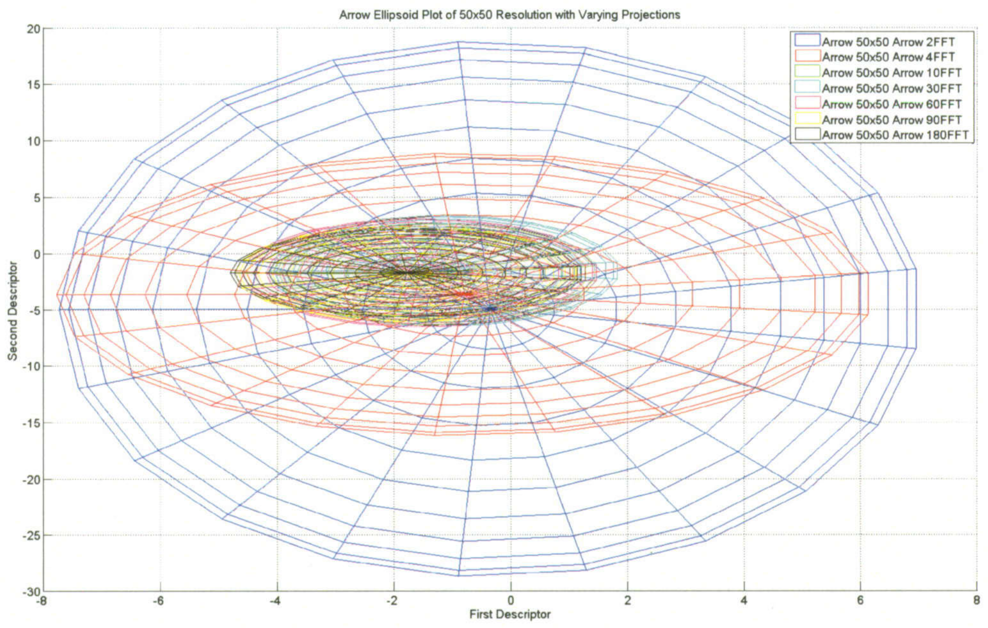


Figure 179: Second and first 3-D shape descriptors of 50x50 resolution arrows.

Moving to a higher resolution of 150x150 pixels in Figure 180 shows similar initial results to the 50x50 resolution plot. The ellipses are centered near the same point (indicating a common mean), with a reduction in variance as the number of projections is increased. Figure 181 shows the third and first descriptors, and as with the spheres a reduction in the variance can be seen up to the 30 projection model.

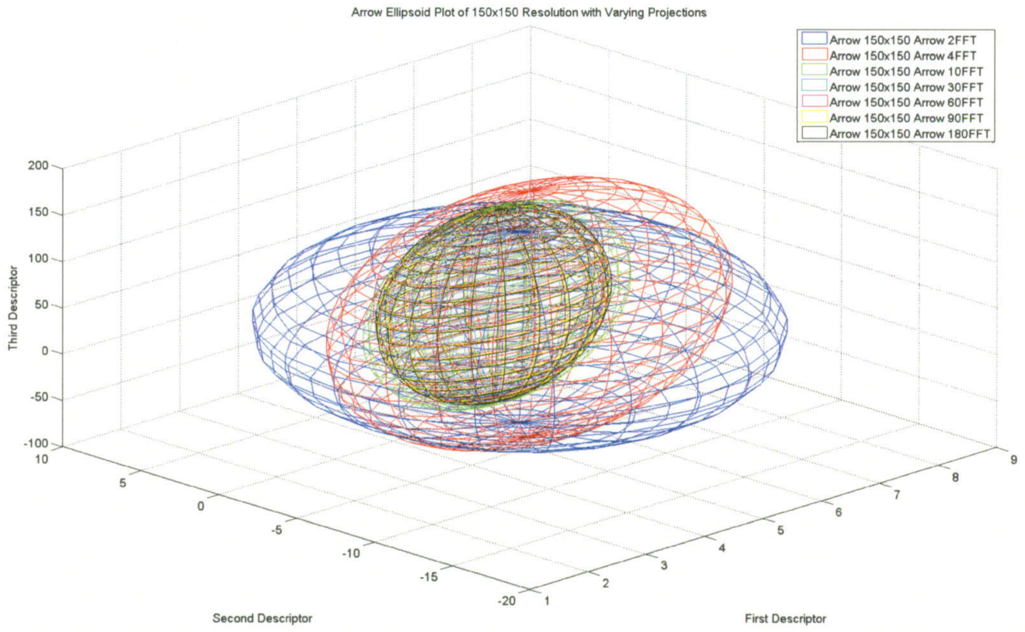


Figure 180: 3-D shape descriptors of 150x150 resolution arrows.

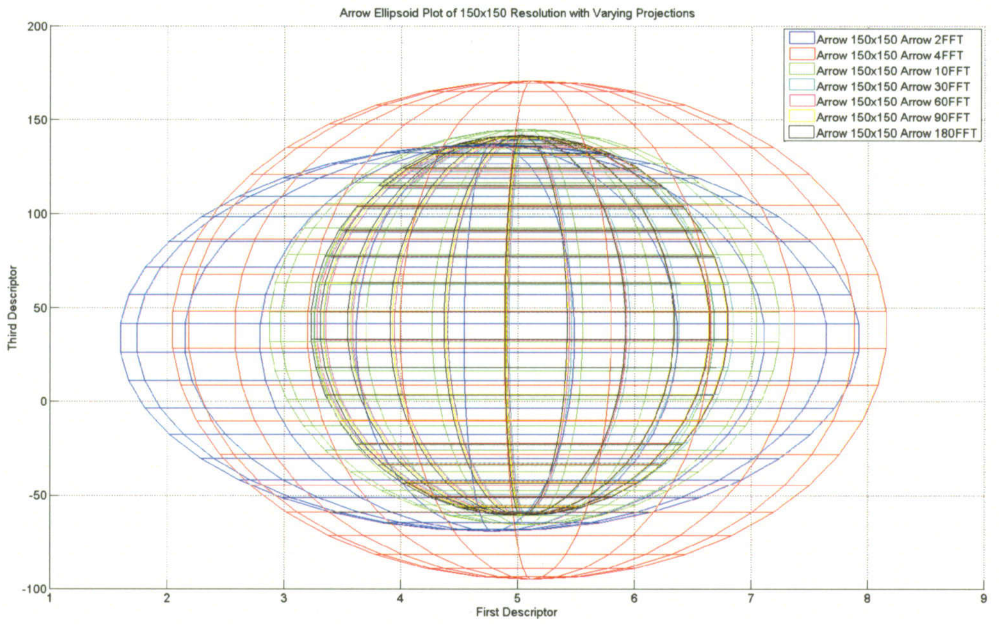


Figure 181: Third and first 3-D shape descriptors of 150x150 resolution arrows.

In Figure 182, a plot of the first and second descriptors shows that the best projection to stop would be 30 projections for a resolution of 150x150 pixels. While there is some gain from utilizing 60 projections across the variance of the first descriptor, it would also mean twice as much time in reconstructing the particle.

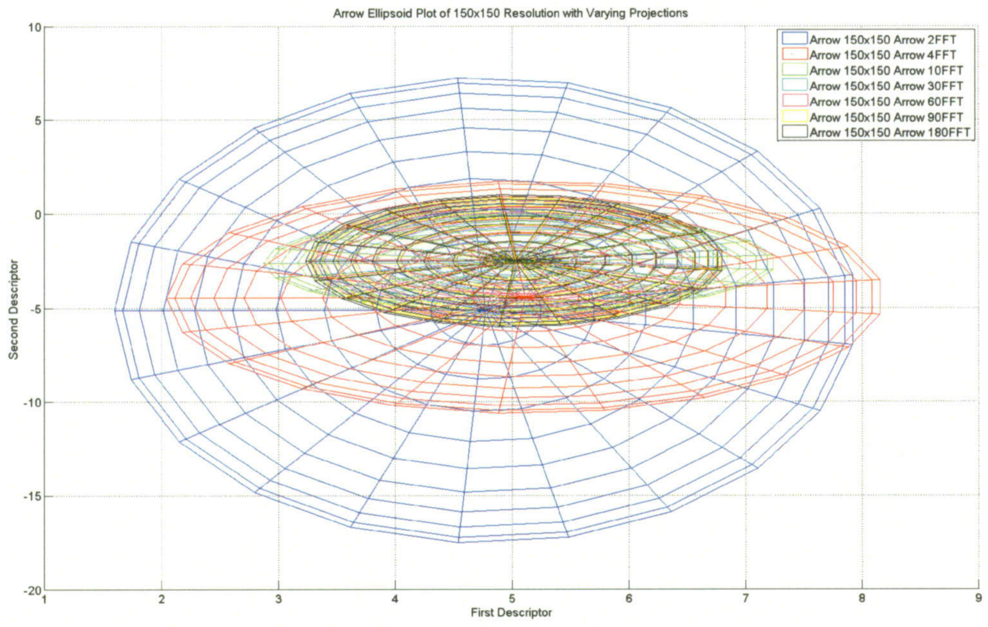


Figure 182: Second and first 3-D shape descriptors of 150x150 resolution arrows.

Figure 183 is a 3-D plot of the maximum resolution used for the arrows at 250x250 pixels. As with all the other plots, the spheres consistently fell within a similar mean and the variance decreased as the number of projections increased. The plot of the first and third descriptors looked very similar to the 150x150 ellipse plot shown in Figure 184.

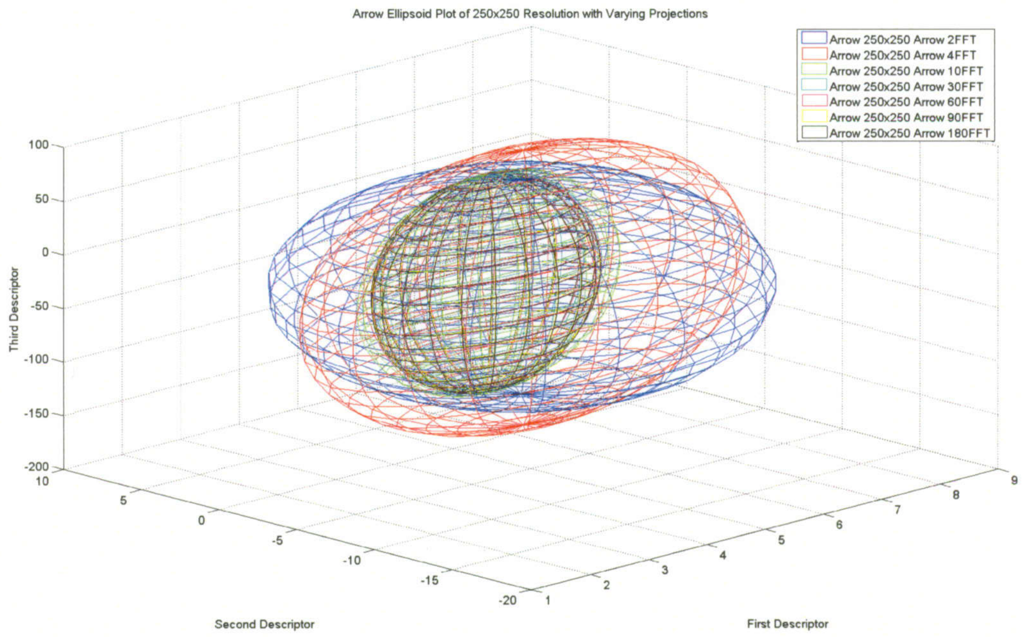


Figure 183: 3-D shape descriptors of 250x250 resolution arrows.

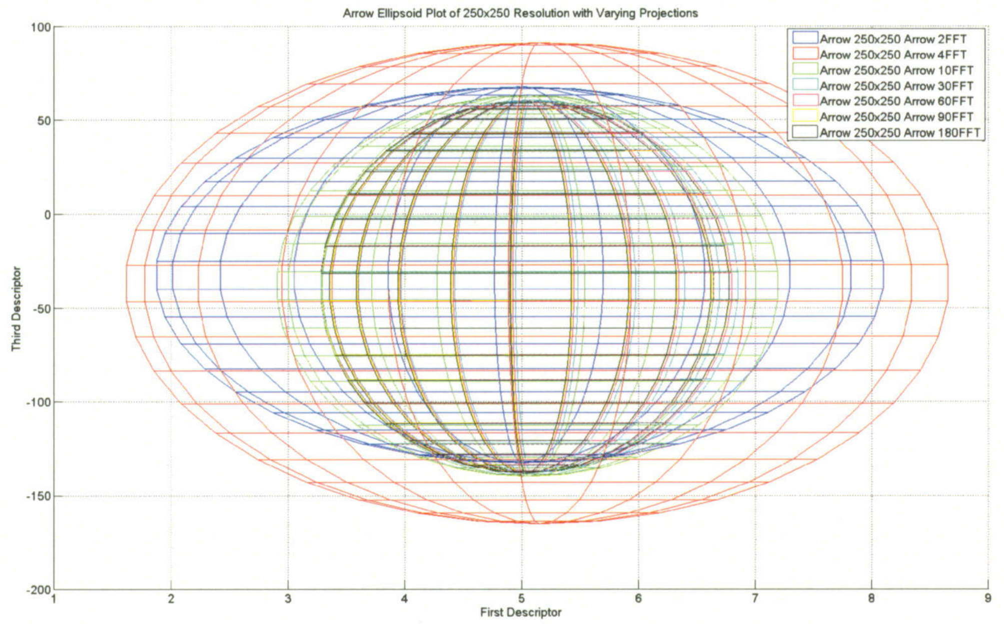


Figure 184: Third and first 3-D shape descriptors of 250x250 resolution arrows.

One very interesting thing to note with Figure 185, which is a plot of the second and first descriptors, is that it is almost exactly the same as the 150x150 resolution version. Even the scaling of the axes are the same across both plots. In the case of the arrows, anything more than 30 projections would result in little reduction in variance and mean compared to the time it would take to reconstruct the object.

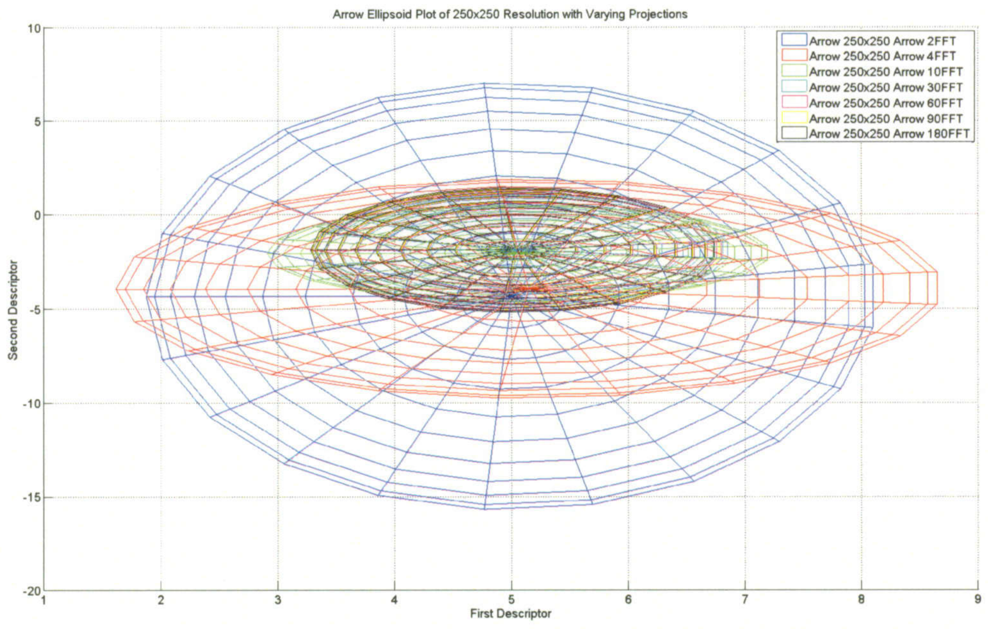


Figure 185: Second and first 3-D shape descriptors of 250x250 resolution arrows.

A humanoid ellipsoid plot was not feasible due to the complex shape involved. Many of the features of the human were concave, and the algorithms designed by Corriveau, Barrot and Giordano were meant for convex spherical shapes [5]. Any attempt to represent the humanoid as an ellipsoid plot would have required extensive morphological image processing and severely reduced the fidelity of the images to the initial reconstruction.

4.5.1.6 3-D Particle Reconstruction

Particle reconstructions from optical data were performed on two different types of sand, a Dry #1 and Michigan Dune particle. The two particles were chosen because they were visually distinguishable so that upon inspection, it would be clear to see how the algorithm performed.

The first particle reconstructed was a Michigan Dune sample taken on 5/31/05. A date is issued to distinguish it against any other reconstruction. Figure 186 shows a comparison between the X-ray reconstruction, the old ART reconstruction and the new reconstruction algorithm.

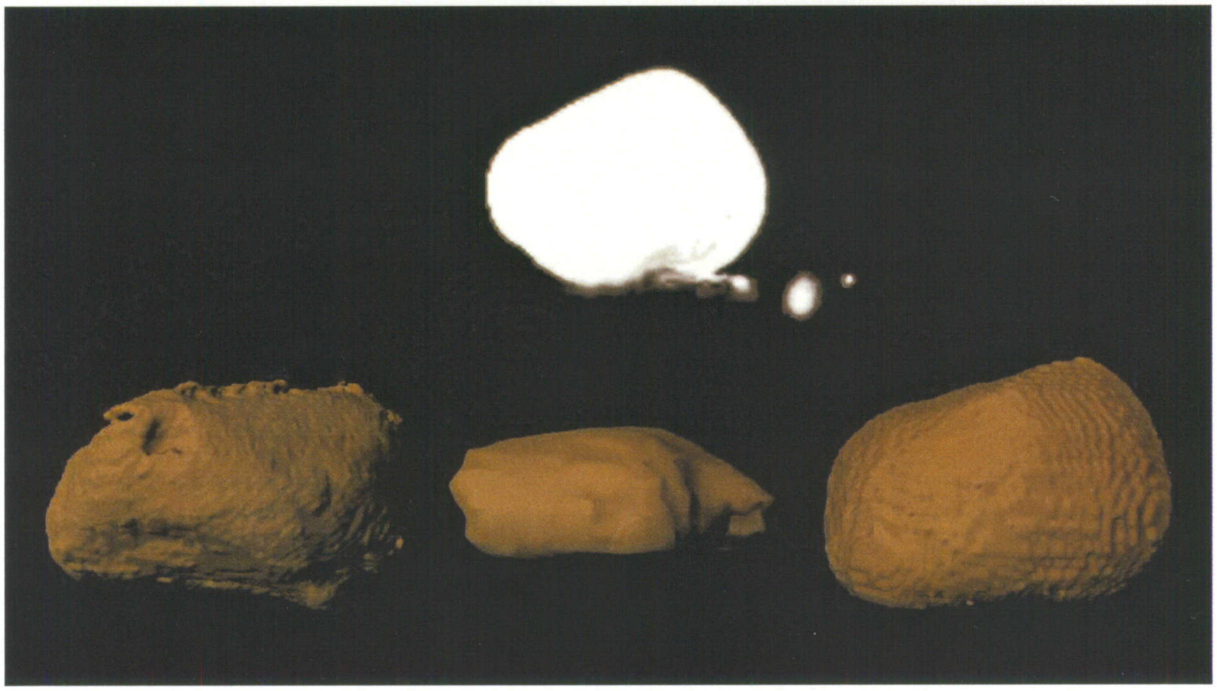


Figure 186: Michigan Dune sample from 5/31/05. On the far left is the X-ray reconstruction, followed by the old optical reconstruction algorithm and then the new reconstruction algorithm.

At the top of the image is a reference of the sand particle at its initial state when it was photographed. For the new reconstruction particle on the far right, a simple rotation about the x and y axis results in an immediate match. The other particles, however, had to be hand manipulated to find an angle that resembled the image. This process is very subjective and due to several reconstruction differences can make it impossible to find an exact angle for comparison. An additional top-down look at the particles can be seen in Figure 187.

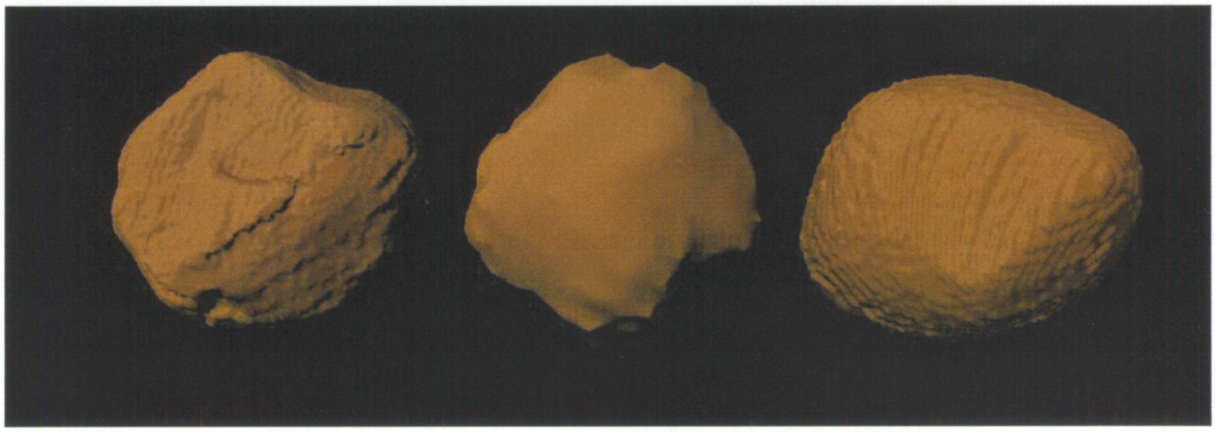


Figure 187: A planar view of the Michigan Dune Sample from 5/31/05. From left to right: X-ray reconstruction, old optical reconstruction, new optical reconstruction.

When scanning the X-ray models, the particle has to be suspended with a clear adhesive. This adhesive has a density that can approach the sand particles when bunched up, cause reconstruction artifacts. The old ART reconstruction algorithm does not directly reconstruct from the captured optical images. Instead, the algorithm utilizes the first three Fourier descriptors to create approximations of the sand particle. From these approximated images the final model is reconstructed. This means that while the general shape should hold, many of the details of the object will be missing. Also, there appears to be scaling issues with the old algorithm. Many of the particles are squashed along the y-axis. This could be due to improper graphing of the particle during the Matlab iso-surface reconstruction. All three axes must be set to the same scale to ensure a properly scaled object.

Figure 188 is the second particle reconstruction utilizing images from the online database from the Dry #1 series taken on 6/27/05.

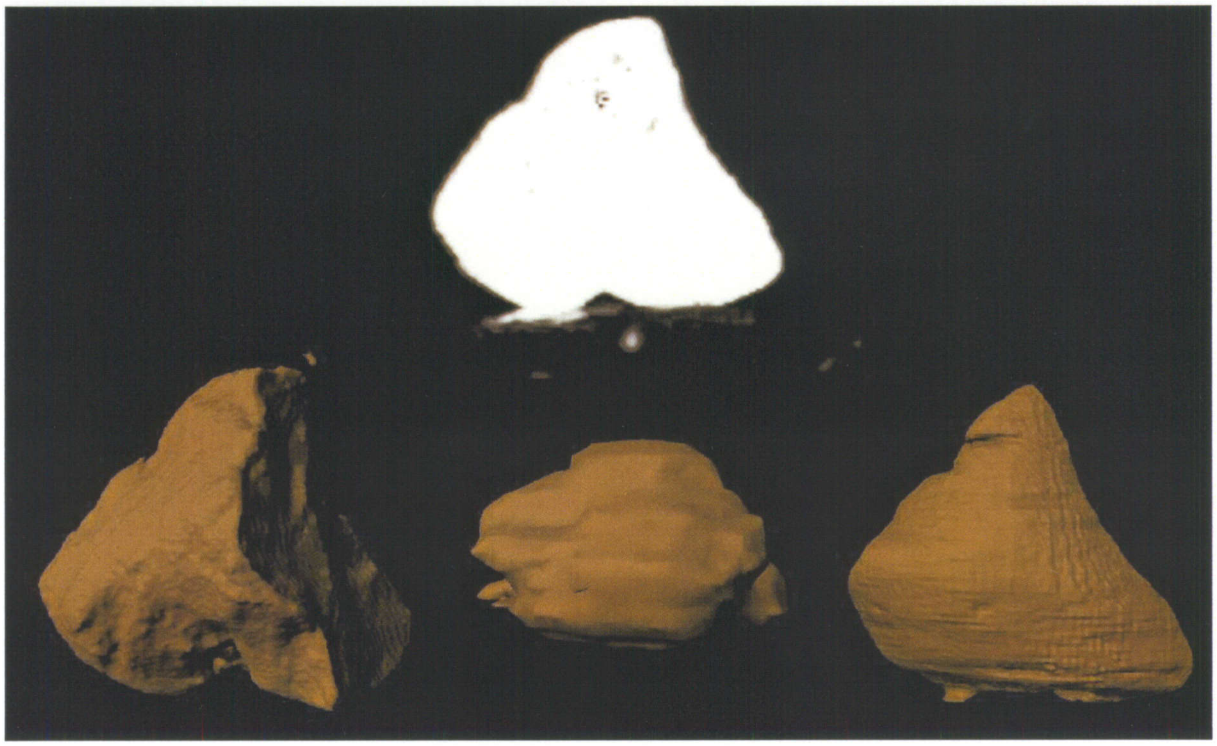


Figure 188: Dry #1 sample from 6/27/05. On the far left is the X-ray reconstruction, followed by the old optical reconstruction algorithm and then the new reconstruction algorithm.

When looking at the top reference image, some noise can be seen inside and around the particle. Due to the lighting setup, at times the stand becomes lit and shows up in the final reconstruction. When looking at the bottom of the right most particles, small outcroppings can be seen which partial reconstructions of the rotating stand are. In addition, since many sand particles are translucent, inner details appear black. This can cause reconstruction problems in the object and can be seen on the rightmost reconstruction at the tip of the particle.

Figure 189 is a top-down view of the particles. Once again the old reconstruction particle in the center has been squashed and flattened. The X-ray on the left and the new reconstruction algorithm on the right retain a proper shape.

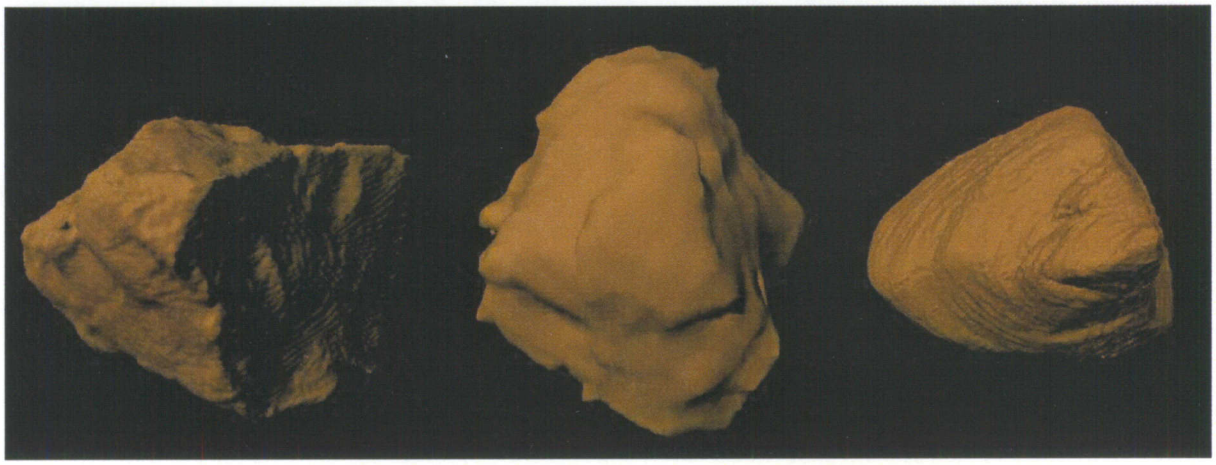


Figure 189: A planar view of the Dry #1 Sample from 6/27/05. From left to right: X-ray reconstruction, old optical reconstruction, new optical reconstruction.

One final issue with reconstructions for all the algorithms is the problem of scale. When the reconstruction algorithms are made, each voxel unit corresponds to one world coordinate. That means that in the case of high resolution reconstructions, the objects will appear in 3-D modeling programs to be extremely large when compared to the low resolution reconstructions. Currently, a manual re-scaling is performed for all the particle reconstructions from X-ray, the old algorithm or the new one.

CHAPTER 5 : CONCLUSIONS

Forerunners in the field of visualization have demonstrated its potential for measuring exploring areas that are difficult to accurately characterize. Furthermore, studies have indicated that human perception can be enhanced when utilizing a 3-D representation of data. This can provide additional perspective, display meaningful patterns, enhance grouping ability and augment the capacity to form relationships between data sets. This thesis has presented a detailed investigation of techniques for visualizing multiple data sets, specifically – graphical, measurement and health data inside a 3-D VR environment. The focus of this thesis has been to develop pathways to generate the required 3-D models efficiently without sacrificing visual fidelity. The tasks include creating the visual representation for the virtual world, integrating multi-sensor measurements, creating multiple user-specific visualizations and a performance evaluation of the completed virtual environment.

The primary focus for utilizing VR as a visual interface is to provide the best graphical interface to the user. The basic three tenets of VR: immersion, navigation and interaction, allow for a visually detailed and spatially accurate environment. The primary objectives of the research work discussed in this thesis are reproduced below along with a description of the specific implementation.

This thesis has addressed –

1. A survey of current techniques for:

    a) The design, development and implementation of immersive, navigable and interactive VR environments for advanced scientific visualization – Previous work describing applications in the areas of visualizations, VR, finite element analysis, fluid dynamics, geological research, medical procedures, psychology were cited.

b) The design, development and implementation of 3-D modeling algorithms in a VR environment – Previous work describing applications in the areas of contour maps, color maps, height-field plots, voxel representations, iso-surfaces, arrow plots and hedgehog plots were cited.

2. Specific contributions in the area of designing, developing and implementing 3-D models inside of a VR environment, specifically for:
    a) Integrated awareness models of graphical, measurement and health data – multiple pathways were investigated for generating the required 3-D models efficiently without sacrificing visual fidelity. Flowcharts were developed for creating 3-D immersive environments for a variety of applications, irrespective of variations in source data formats.
    b) 3-D reconstruction models of tomography data – an efficient and accurate algorithm has been developed for model synthesis from multiple 2-D projections. The algorithm provides for the synthesis of particle shapes inside a homogeneous aggregate mixture.
    c) Applications and validations of methods in a sufficient number of example VR environments – these include the visualization of a rocket engine test stand and computed tomographic reconstructions of optical image projections of geomaterial aggregates.

5.1 Summary of Accomplishments

In this thesis, varying forms of data were transformed into visualizations for viewing. Virtual reality was utilized in several of these applications to create immersive, interactive and navigable environments. The following list contains a concise description of the accomplishments in this thesis as well as the benefits and disadvantages.

I. **Multi-dimensional Data Visualizations** – The visualizations presented here covered one, two and three-dimensional solutions to viewing information of either scalar or vector form.

A. The benefits demonstrated in this research work include –

1. Visualization pathways for scalar and vector data of 1, 2 or 3 dimensions.
2. Functions are implemented inside of a single call for easy portability.
3. Visualizations can use either color and/or magnitude to describe what is happening with the data.
4. Provides visualizations for graphs, surfaces and time-varying vector data.

B. The disadvantages of the method are –

1. Not all visualizations could be created for real-time application. Iso-surface generations still require external creation so that the resulting surfaces can be seen inside of the VR environment.
2. Surface generation requires an algorithm for computing vertex normals, which will allow proper lighting to be displayed across the surface.

3. Advanced graphical features such as graph and axis scale/size still need to be implemented to create a truly robust visualization.

II. **Integration of Graphical, Measurement and Health Data in a VR Environment -** Several virtual environments were created for various forms of data to create simplistic visualizations. Several process flowcharts were devised to handle the different forms of information inside of the virtual world. These demonstrations covered the visualization of a rocket engine test stand during test firing and the comparison of different types of sand from across the world.

A. The benefits demonstrated in this research work include –

1. Provides a proven process path for handling graphical, measurement and health data that has been utilized in over five VR simulations.

2. Multiple data visualizations for various types of sensor data including: temperature, pressure, strain, current and valve state.

B. The disadvantages of the method are –

1. Minimal health information restricted the number of visualizations that could be produced for this type of data.

2. Depending on the format of the graphical data, modeling the objects can take an extensive amount of time. Automated methods for handling this information have not yet been developed.

III. **3-D Reconstruction Models of Tomographic Data –**

A. The benefits demonstrated in this research work include –

1. A fast and easily understood implementation of 3-D reconstruction from various projections around an axis.

2. Reconstructions were evaluated for processing time from a standpoint of projections vs. resolution.

3. Fourier analysis and subsequent plotting of the descriptors showed the benefits for utilizing resolutions above 50x50 pixels but no higher than 30 projections.

4. Showed a logarithmic progression in the time to reconstruct models as either resolution or number of projections rose.

B. The disadvantages of the method are –

1. Concave surfaces cannot be properly reconstructed by the algorithm.

2. When plotting the Fourier descriptors of a reconstruction, sufficiently concave projections will cause the algorithm to fail. Only through morphological digital image processing can most non-circular projections be utilized.

Virtual Reality was implemented as a visualization tool for displaying various types of data. Information was visualized though the use of several visualization methods to allow the user to select their own preferences. Depending on the requirements, the visualizations could be changed to provide information either quickly and generally or more precise and exact. The interface also provided the user with several options for navigation around the environment as well as a map to easily determine relative location.

1. The VR environment allows for visualization of multiple sensor data sets that might only have been previously viewed as two-dimensional spreadsheets, allowing for more complete and easier understanding of what is occurring.

2. VR enables a visualization method that includes the spatial relationship of the 3-D models, a feature that can be lost with other two-dimensional approaches.

5.2 Recommendations for Future Work

Each of the three topics presented in this thesis can be further refined. In the area of data visualization, the implementation of a real-time iso-surface would decrease the time it takes to create 3-D surfaces. A simple algorithm needs to be implemented that would update the normals on the surfaces of the polygons so that lighting information could be correctly performed. Additional visualizations could also be implemented such as cone plots or seeding functions when observing vector quantities that deal with flow.

In the area of the MTTP trailer, further work needs to be done in the area of ISHM and VR integration. At this point Vizard has limited access to health data from the Gensym G2 software package, which is responsible for root cause analysis and anomaly detection. Through the use of either a novel communication format or the integration of an existing format such as OSA-CBM a pipeline could be created that would allow for visualization of sophisticated health information.

Also, visualizations pertaining to gas or liquid flow through pipe ways could be added to the simulation based on available pressure information. Analysis data of the rocket plume could be added to the visualization, but only if it is first preprocessed and added later as a graphical object.

CHAPTER 6 : LIST OF REFERENCES

[1] S. Papson, "An Investigation of Multi-Dimensional Evolutionary Algorithms for Virtual Reality Scenario Development," M.S. thesis, College of Eng., Rowan Univ., Glassboro, NJ, 2004.

[2] P. Freymuth, "Flow visualization in fluid mechanics," *Review of Scientific Instruments*, vol. 64, no. 1, pp. 1-18, Jan., 1993.

[3] M. Schulz, T. Reuding and T. Ertl, "Analyzing engineering simulations in a virtual environment," *IEEE Computer Graphics and Applications*, vol. 18, no. 6, pp. 46-52, Nov./Dec., 1998.

[4] F. Fish, L. Howle, M. Murray, "Hydrodynamic flow control in marine mammals," *Integrative and Comparative Biology*, vol. 48, pp. 788-800, May, 2007.

[5] P. Giordano, "Optimization of optical computed tomography techniques for the synthesis of particle aggregate models," M.S. thesis, College of Eng., Rowan Univ., Glassboro, NJ, 2007.

[6] J. Corriveau, "Three-dimensional shape characterization for particle aggregates using multiple projective representations," M.S. thesis, College of Eng., Rowan Univ., Glassboro, NJ, 2004.

[7] D. Barrot, "An algebraic reconstruction technique (ART) for the synthesis of three-dimensional models of particle aggregates from projective representations," M.S. thesis, College of Eng., Rowan Univ., Glassboro, NJ, 2005.

[8] R. Spence, *Information Visualization: Design for Interaction*, 2nd ed. Essex, England: Education Limited, 2007.

[9] H. Simon, *The Sciences of the Artificial*, 3rd ed. Cambridge, Maine: MIT Press, 1996.

[10] S. Bryson, "Virtual reality in scientific visualization," *Communications of the ACM*, vol. 39, no. 5, pp. 62-71, May, 1996.

[11] C. Ware, *Information Visualization: Perception for Design*, 2nd ed. San Francisco, California: Morgan Kaufmann, 2004.

[12] A. Dam, A. Forsberg, D. Laidlaw, J. LaViola, and R. Simpson, "Immersive VR for scientific visualization: a progress report," *IEEE Computer Graphics and Applications*, pp. 26-52, Nov. /Dec., 2000.

[13] F. Brooks, "What's real about virtual reality," *IEEE Computer Graphics and Applications*, vol. 19, no. 6, pp. 16-27, Nov. /Dec., 1999.

[14] E. Castronova, *Synthetic Worlds: The Business and Culture of Online Games*, Chicago, Illinois: The University of Chicago Press, 2005.

[15] T. Hong, "A virtual environment for displaying gas transmission pipeline NDE data," M.S. thesis, College of Eng., Iowa State Univ., Iowa, 1997.

[16] M. Slater, A. Steed and Y. Chrysanthou, *Computer Graphics and Virtual Environments*. Addison Wesley, 2002.

[17] Mechdyne Corporation, *Mechdyne Corporation 3D and Advanced Visualization Solutions* [Online].Available: http://www.mechdyne.com, May, 2008.

[18] N. Valys, *Stereoscopy*, London: The Focal Press, 1966, (Translated from the Original Text).

[19] L. Chittaro, R. Ranon and L. Ieronutti, "VU-flow: a visualization tool for analyzing navigation in virtual environments," *IEEE Transactions on*

Visualization and Computer Graphics, vol. 12, no. 6, pp. 1475-1485, Nov. /Dec., 2006.

[20] J. Thurmond, P. Drzewiecki and X. Xueming, "Building simple multi-scale visualizations of outcrop geology using virtual reality modeling language (VRML)," *Computer & Geosciences*, vol. 31, no. 7, pp. 913-919, March, 2005.

[21] R. Bernhard, B. Alexander, B. Reinhard and S. Dieter, "Liver surgery planning using virtual reality," *IEEE Computer Graphics and Applications*, vol. 26, no. 6, pp. 36-47, Nov. /Dec., 2006.

[22] M. Slater, D. Pertaub and A. Steed, "Public speaking in virtual reality: facing an audience of avatars," *IEEE Computer Graphics and Applications*, vol. 19, no. 2, pp. 6-9, March, 1999.

[23] M. Tory, A. Kirkpatrick, M. Atkins, and T. Moller, "Visualization task performance with 2D, 3D, and combinational displays," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 1, pp. 2-13, Jan./Feb., 2006.

[24] R. Pausch, D. Proffitt and G. Williams, "Quantifying immersion in virtual reality," *International Conference on Computer Graphics and Interactive Techniques*, Charlottesville, VA., 1997. pp. 13-18.

[25] D. Hearn and M. Baker, *Computer Graphics with OpenGL*, 3rd ed. Upper Saddle River, NJ: Pearson Prentice Hall, 1986.

[26] D. Astle and K. Hawkins, *Beginning OpenGL: Game Programming*, Boston, MA: Premiere Press, 2004.

[27] VRCO Inc, *VGeo User's Guide*, version 1.7. Virginia Beach, VA: VRCA Inc, 2002.

[28] S. Govil-Pai, *Principles of Computer Graphics: Theory and Practice Using OpenGL and Maya*, New York, NY: Springer Science + Business Media, 2004.

[29] J. Choi, *Maya Character Animation*, 2nd ed. Seoul, Korea: Sybex, 2004.

[30] *(2008, May). Autodesk Maya Help* [Online]Available: http://www.autodesk.com.

[31] R. Wright, B. Lipchak, and N. Haemel, "OpenGL Superbible: Comprehensive Tutorial and References," 4th ed. Upper Saddle River, NJ: Addison-Wesley, 2007.

[32] H. Wright, *Introduction to Scientific Visualization*, Hull, United Kingdom: Springer Sciences + Business Media, 2007.

[33] K. Ma, "Visualizing time-varying volume data," *Computing in Science & Engineering*, vol. 5, no. 2, pp. 34-42, March, 2003.

[34] A. Helgeland and T. Elboth, "High-quality and interactive animations of 3D time-varying vector fields," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 6, pp. 1535-1543, Nov., 2006.

[35] T. Urness, V. Interrante, E. Longmire, I. Marusic, S. Neill and T.W. Jones, "Strategies for visualization of multiple 2D vector fields," *IEEE Computer Graphics and Applications*, vol. 26, no. 4, pp. 74-81, July, 2006.

[36] J. Berkley, G. Turkiyyah, D. Berg, M. Ganter and S. Weghorst, "Real-time element modeling for surgery simulation: an application to virtual suturing," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 3, pp. 314-325, May/June, 2004.

[37] R. Sharp, J. Adams, R. Lee, R. Crane, and R. Machiraju, "Physics-based subsurface visualization of human tissue," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 3, pp. 620-629, May, 2007.

[38] S. Babu, A. Ulinski, B. Lok, L. Hodges and C. Zanbaka, "Comparison of path visualizations and cognitive measures relative to travel techniques in a virtual environment," *IEEE Transactions on Visualization and Computer Graphics*, vol. 11, no. 6, pp. 694-705, Nov. /Dec., 2005.

[39] K. Garrison, "Exploration of multiple pathways for the development of immersive virtual reality environments," M.S. thesis, College of Eng., Rowan Univ., Glassboro, NJ, 2008.

[40] G. Lecakes, J. Morris, J. Schmalzel and S. Mandayam, "Virtual reality platforms for integrated systems health management in a portable rocket engine test stand," *International Instrumentation and Measurement Technology Conference*, Victoria, May, 2008, pp. 388-392