THESIS

EXTENSIBLE 3D (X3D) GRAPHICS CLOUDS FOR GEOGRAPHIC INFORMATION SYSTEMS

by

Darren W. Murphy

March 2008

Thesis Advisor: Philip A. Durkee
Second Reader: Don Brutzman

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2008	**3. REPORT TYPE AND DATES COVERED** Master's Thesis
4. TITLE AND SUBTITLE Extensible 3d (X3d) Graphics Clouds for Geographic Information Systems		**5. FUNDING NUMBERS**
6. AUTHOR(S) Darren W. Murphy		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.		
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**

13. ABSTRACT

This research evaluates the production of three dimensional (3D) clouds for geospatial viewing programs such as Google Earth, NASA World Wind, and X3D Earth. This thesis took advantage of iso-standard X3D graphics and X3D Edit in conjunction with manually produced image textures to represent 3D clouds. While a 3D geospatial viewing might never completely characterize the current state of the atmosphere, a sufficiently realistic virtual 3D rendering can be created to present current sky coverage given adequate satellite and model data. Various visualization demonstration results are presented that can be rendered and navigated in real time. Further research and development is needed to match a cloud typing model output with a particular method of 3D cloud production. Data-driven adaptation and production of cloud models for web-based delivery is an achievable capability given continued research and development.

14. SUBJECT TERMS X3D Cloud GIS			**15. NUMBER OF PAGES** 101
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

EXTENSIBLE 3D (X3D) GRAPHICS CLOUDS FOR GEOGRAPHIC INFORMATION SYSTEMS

Darren W. Murphy
Captain, United States Air Force
B.S., University of California at Davis, 2000

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN METEOROLOGY

from the

NAVAL POSTGRADUATE SCHOOL
March 2008

Author: Darren W. Murphy

Approved by: Philip A. Durkee
Thesis Advisor

Don Brutzman
Second Reader

Philip A. Durkee
Chair, Department of Meteorology

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This research evaluates the production of three dimensional (3D) clouds for geospatial viewing programs such as Google Earth, NASA World Wind, and X3D Earth. This thesis took advantage of iso-standard X3D graphics and X3D Edit in conjunction with manually produced image textures to represent 3D clouds. While a 3D geospatial viewing might never completely characterize the current state of the atmosphere, a sufficiently realistic virtual 3D rendering can be created to present current sky coverage given adequate satellite and model data. Various visualization demonstration results are presented that can be rendered and navigated in real time. Further research and development is needed to match a cloud typing model output with a particular method of 3D cloud production. Data-driven adaptation and production of cloud models for web-based delivery is an achievable capability given continued research and development.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

It has been a pleasure to work with the Meteorological Department and the MOVES Institute at the Naval Postgraduate School. Many thanks to Professors Philip Durkee, Don Brutzman, and Jeff Weekley for hours of dedicated assistance. Thank you to my family for the encouragement. Above all, thank you to my wife Sydni for living and working with me side by side the whole way.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Cloud visualization continues to present a challenge for both the meteorological and computer visual graphics communities. The ever-increasing ability to have larger amounts of computer memory, as well as allocated memory for graphics cards in a personal desktop or laptop computer are allowing for improvement in cloud generation and visualization techniques. Bringing together computer graphics and meteorological knowledge bases makes it possible to represent cloud phenomena in a manner that is scientifically accurate and aesthetically pleasing.

While meteorologists are trained to conceptually visualize 3D aspects from a 2D cloud rendering or even numerical data, it is difficult to pass this knowledge on to a customer. A 3D visualization that encompasses a certain meteorological correctness and an artistically correct rendering of clouds can be a valuable addition to Air Force weather products. Such visualizations must provide a direct path from numerical model fields to a customer-oriented viewing platform, in order to improve end-user understanding and utilization of weather data. Such a platform can be used at all levels of planning and encompass all types of missions from ground patrol to flight paths. Additionally, such visualizations must permit the military forecaster to note areas of particular interest and focus on the meteorological soundness of a forecast thereby saving time and avoiding possible errors by eliminating the mental conversion to numerical or 2D data.

A theater-wide sensing strategy continues to be a top priority for the Air Force Weather forces. With ingest of available ***insitu*** instrumental, observer, and remote sensing data, perhaps a comprehensive virtual 3D sky rendering can be used for critical mission oriented decisions. By utilizing this study's 3D cloud visualization platform combined with already available model data as a "nowcast," the forecaster in the field, while not having the ability to have eyes on the desired observation area, can get a synthetic approximation of what the cloud conditions are. This will significantly improve or, in part, imitate a remote theater sensing strategy.

A number of questions can be raised as to why such an endeavor with 3D clouds is appropriate for the Air Force or to the US Military in general. One such question is what are the advantages of rendering clouds in this manner rather than the traditional 2D visualizations or simply creating the appearance of clouds for the GIS systems? Very simply we live in a 3D world. Every flight, maneuver, and operation requires the planner or operator to visualize his 3D mission from a 2D map. A whole new era of mission planning and execution is upon us with the ability to conceptualize the plans on a computer with 3D visualization. Creation of full 3D volumetric clouds, rather than an appearance of clouds within a 3D environment, is needed to represent the current state of the measured atmosphere as we would see it. Most importantly these volumetric clouds give a realistic view to customers whose central mission is of an aerial nature.

The presentation of cloud data within a 3D virtual environment is beneficial for more than military or government uses. The consumer use of GIS programs has become varied and broad. Thus the general benefit of 3D cloud rendering as presented by this thesis is also broad. It represents the next step in the evolution of weather visualization products.

The ultimate goal of this thesis is to connect the new and popular consumer 3D visualization programs with meteorological variables, creating a cloud representation using a standard personal computer. With this purpose in mind a collection of fog, stratus, cirrus, and cumulus cloud cases are developed for visual display. Within the program, full analysis of the cloud structures can be accomplished by using predefined scene navigation. User interaction can include a land-based examination of the surrounding sky, or a full flight plan that follows a mission path through different levels and types of clouds.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A. CONCEPT

When looking back on the history of 3D applications for cloud visualization, it is helpful to look at both scientific and visual graphic efforts. While focused on the same topic, these two efforts tend to take a different approach toward 3D cloud visualization. A brief look at these two approaches to designing 3D clouds is essential for understanding the research directions applied in this thesis.

Meteorologists and the scientific community tend to be focused on all aspects of defining areas of clouds precisely and correctly using either data-driven or mathematical modeling techniques. Efforts to more accurately define the cloud base, cloud top, and incremental resolution have been the objectives of much research. Many different sources of data can ultimately provide input for the 3D space occupied by a cloud structure. Satellite, radar, atmospheric soundings, and ground observations can all be sources of input data, with availability dependent on the area and time period under scrutiny. For example, there is a large disparity in data available from the Continental U.S. (CONUS) vs. data sparse regions such as Afghanistan or Iraq.

Regardless of the rendering program, the scientific visualization of environmental data tends to be quite logical showing clear delineation between incremental changes in the data. This often leads to colorful 2D map presentations with a legend that details the data for the user. While perfectly useful for the scientific community, this type of display requires the customer to conceptualize the data as a 3D cloud. Such conceptualization is a learned skill and not inherently intuitive.

It is important to emphasize when data visualization attempts to render a better resolution than the data captures, a certain amount of interpolation is required. In practice, interpolation is seen in the parameterization and grid placement of data variables within a weather model. Ingestion of data from various different locations to a gridded model requires the use of parameterizations to not only fit the data to a computational grid, but also to forecast the variables forward in time. In the visualization of such data the opposite process often occurs. Algorithms are used to modify the gridded data into a more realistic area for the interpolated data to occupy. The result is data values are no longer constrained to fit a gridded mesh, but rather correspond better to geographic features.

B. SCIENTIFIC VIZUALIZATION

The number of available scientific applications used to view meteorological data is large. It is worthwhile to investigate such applications to determine if there are any features desirable for a 3D application as studied by this research. One such application long used by the scientific and academic world is the Vis5D program.

VIS5D software is an OpenGL based volume renderer for 2D as well as 3D graphics. At the heart of the Vis5D software is a process called 3D texture mapping. This process takes advantage of graphics-acceleration hardware to rasterize polygonal slices from texture memory to view a 3D image (Meibner et al, 2000). The process of volume or volumetric rendering can be slightly difficult to understand. The goal of 3D volume visualization is to display an object, or in most cases data, that fills a 3D volume. In order to represent this continuous data field to a viewer, usually an internal surface of interest is the only part of the object that needs to be rendered. Hence, a patchwork of connected polygons in 3D virtual space is used to represent the 3D object. Figure 1 is an example of a Vis5D 3D rendering of a yellow isosurface enclosing constant parameter values and white 2D contours following constant parameter values. Prominent in the figure is the 3D box with an isosurface and volumetric displays overlaid on a raised surface physical map projection. While Vis5D can display large volumes of gridded data, and represent them in a smoothly shaded visualization, rendering of data becomes more complex at higher resolutions. These renderings are primarily used for exterior viewing without the option of being able to navigate through the scene of individual clouds. If this option were available, the fly-through navigation of internal cloud structures would not seem realistic.

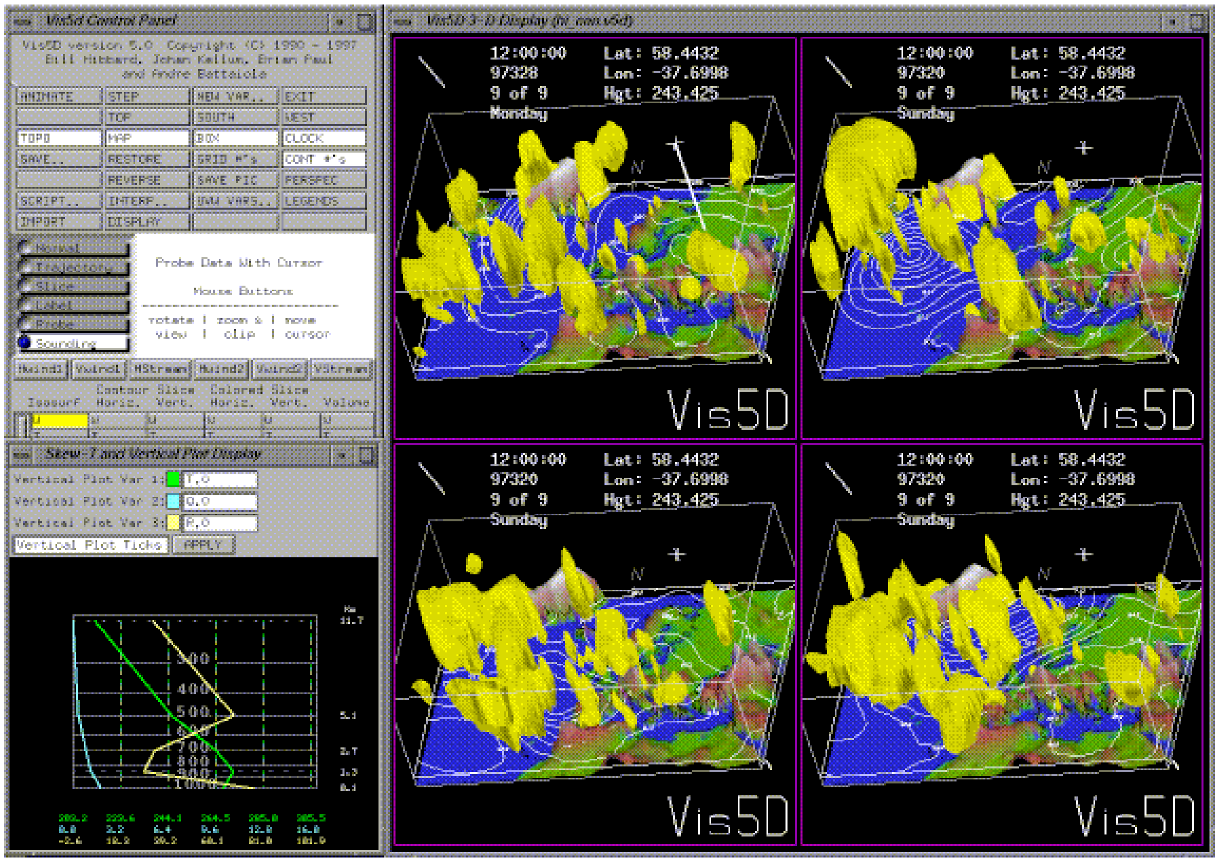


Figure 1. 3D example of Vis5D showing ensemble forecast data at approximately 1:1800000 ratio for vertical exaggeration (SSEC,1998).

Of all the features of the Vis5D software, arguably the most beneficial is open source licensing. While the definition of open source has evolved over the years, the benefits are consistent, primary among them being that the cost is free for any use. Open source coding also allows for open dialog from developers and users alike. This free sharing of ideas encourages development and improvement of the software, adding features and fixing bugs. Such is the case with the Vis5D community. While the software is no longer supported in its original form as created by the

Space Science and Engineering Center of the University of Wisconsin-Madison, the original code has been used and improved in various ways by multiple organizations (Hibbard, 1989).

Regardless of the form of software, algorithms for rendering, or amount of shading all the scientific software visualization tools use data to drive the rendering process. Gridded data can be rendered in 3D independent of file format.

C. VIRTUAL VISUALIZATION

In other visualization applications, the endeavor to represent reality is the challenge. Software developers in gaming and graphics animation companies have come a long way in their ability to visualize a realistic sky. In the case of this thesis, there are many tips and methods that can be acquired from studying these endeavors. One such endeavor is the process of making the video game software - Microsoft Flight Simulator 2004. Figures 2 and 3 are examples of the cloud renderings from the program. These examples are notional renderings in a virtual world for the purposes of the flight simulator game and do not try to represent a real cloud at a particular place and time.

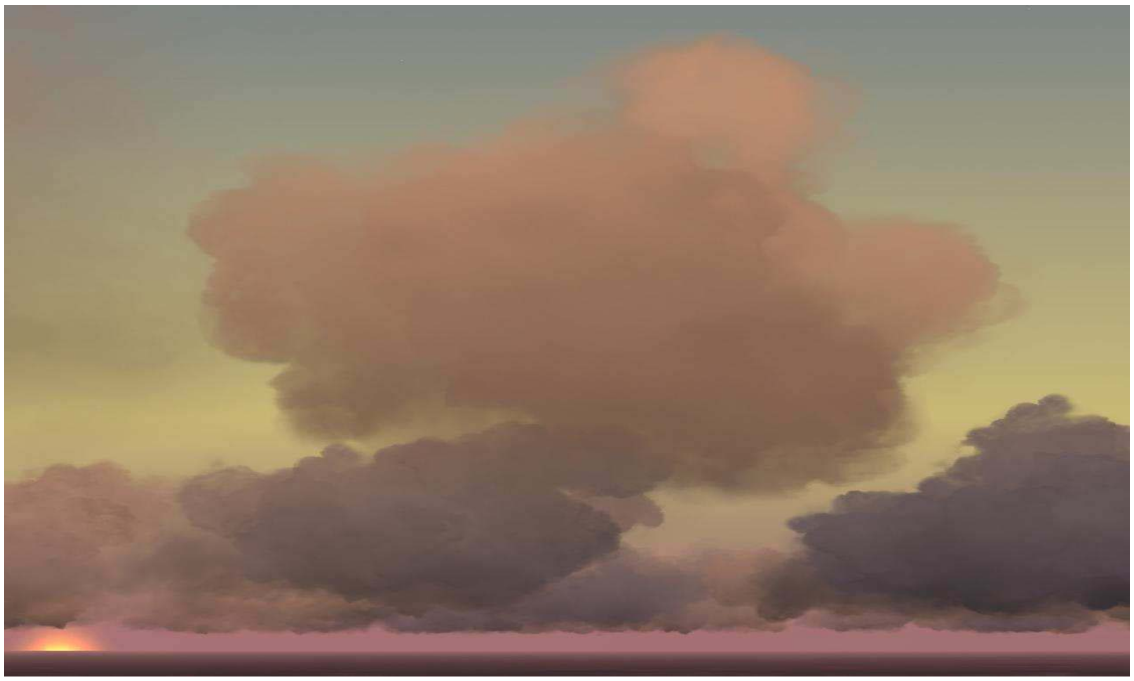


Figure 2. Realistic clouds at sunset produced from textured sprites for Microsoft Flight Simulator (Wang, 2003).

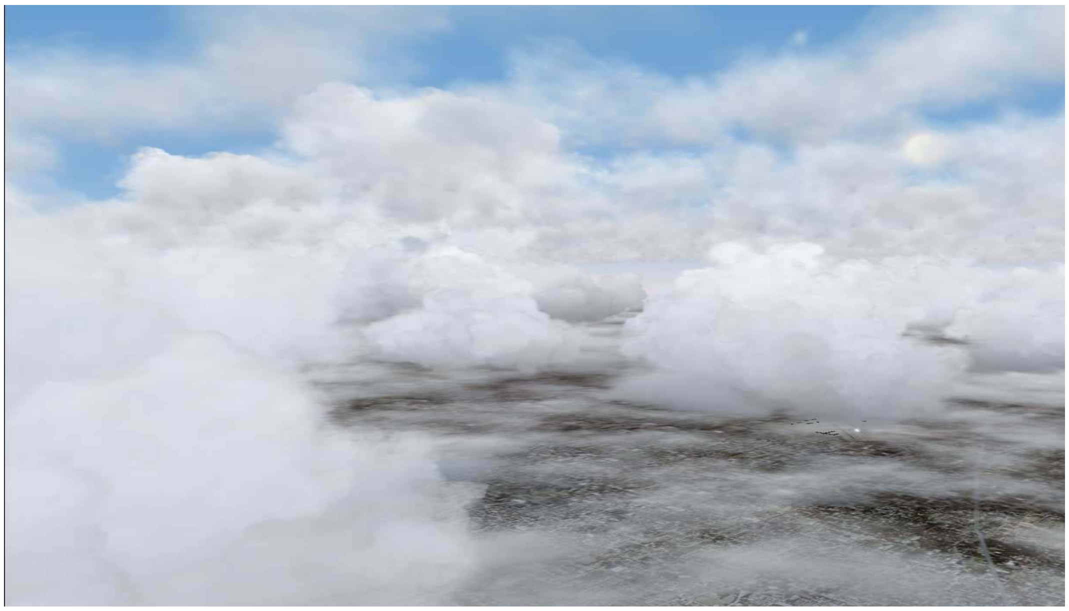


Figure 3. A cloud scene for Flight Simulator showing stratus, cumulus congestus, and altocumulus (Wang,2003).

For virtual flight simulation, it was necessary to develop a process to render realistic clouds while using as little of the computer memory and graphics resources as possible. Likewise, the graphics process of developing clouds for geospatial display programs needs to be computationally inexpensive. Adding data driven clouds to such programs is a capability the programs were not originally designed for.

The Flight Simulator team was successful in creating artistic clouds using a different technical approach when compared to the clouds rendered by Vis5D. No collected data or numerical models were used in the production of Flight Simulator clouds. Ten types of clouds were manually created for the program using image-editing software and proprietary scripting. Bounding boxes, as seen in Figure 4, defining the areas of clouds are placed throughout the 3D scene. These clouds are not the volumetric polygon surface renderings common to 3D scientific data clouds. The clouds are created by assigning locations within the bounding boxes for 16 different 2D images on "sprites" that face toward the camera view at all times. This process as seen in Figure 5 gives the viewer the appearance of a 3D image formation. The patterns of the 2D images, sprite density, location, and rotational aspect of the images assist in the composition of different cloud types. Within the software program, calculations are completed for time of day and depth-of-image location within the cloud for shading purposes, giving an even more realistic appearance. Not all of the cloud structures are of a volumetric design. For faster rendering additional cloud "imposters" were created. Only a certain number of real volumetric clouds are rendered within a finite region around the camera or user view. The imposters are images of clouds applied to a ring surrounding the viewer at the horizon. While adding to the realism of the flight simulator by providing clouds at a distance, there is no interaction with the cloud imposters and the viewer (Wang, 2003).

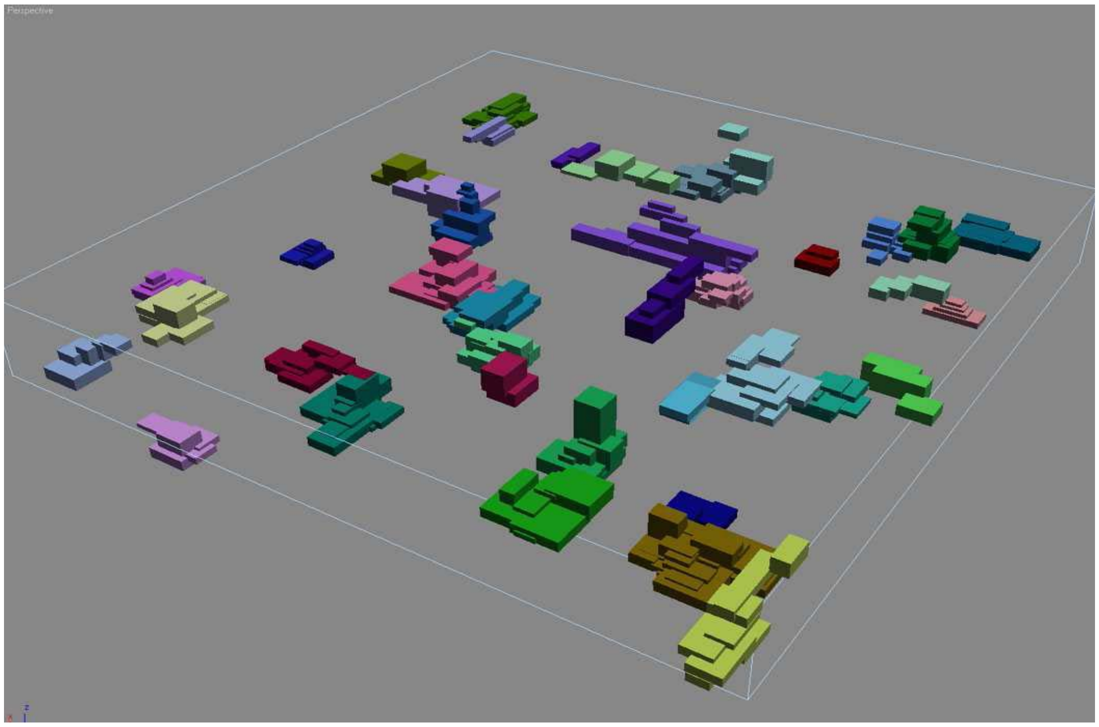


Figure 4. Bounding boxes are created in Microsoft Flight Simulator for regions of clouds (Wang, 2003).

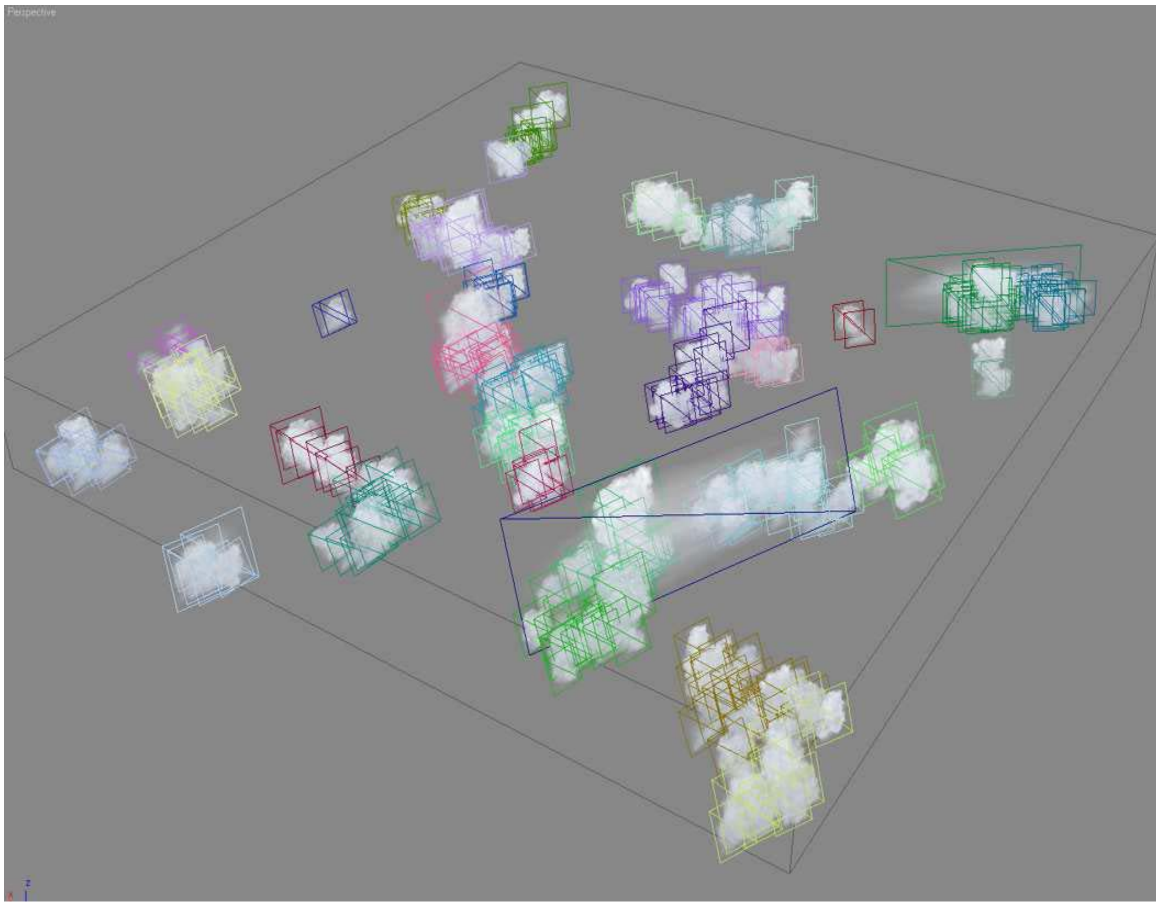


Figure 5. Textured sprites are scattered throughout the bounding boxes to visually represent different cloud types (Wang, 2003).

Other companies have created similar volumetric 3D clouds. Generally the processes of creating volumetric clouds are proprietary, as are the software visualization packages. Likewise, these cloud representations are customarily artistically driven and therefore created "by hand." Once a particularly striking resemblance of a cloud is formed, it can be copied and manually placed in a different location within the 3D environment. The gaming community is most likely satisfied with these cloud-making procedures. The process is computationally inexpensive while the frame rate of displayed images remains high, and most importantly the realism of actual clouds are present. Even while research of possible connections between meteorological data and virtual clouds was being accomplished for this thesis new methods were created.

Sundog Software has recently introduced a product named Silver Lining. Silver Lining is slightly different from the artistically created clouds in that the clouds are procedurally created. There are no copies of clouds. All clouds as well as imposters are dynamically created within the software. Within the Graphical User Interface (GUI) the user can determine selections for meteorological variables, time of day, and the determination of computational limitations. Currently four types of clouds are scripted to be built: cumulus congestus, cumulus mediocris, cumulonimbus stratus and cirrus. Figure 6 shows how realistic procedurally created clouds can be in a 3D geospatial viewer with high resolution surface imagery. The company offers a commercial software development kit (SDK) for software developers to use the Silver Lining application program interface (API) with other programs (Sundog, 2008).

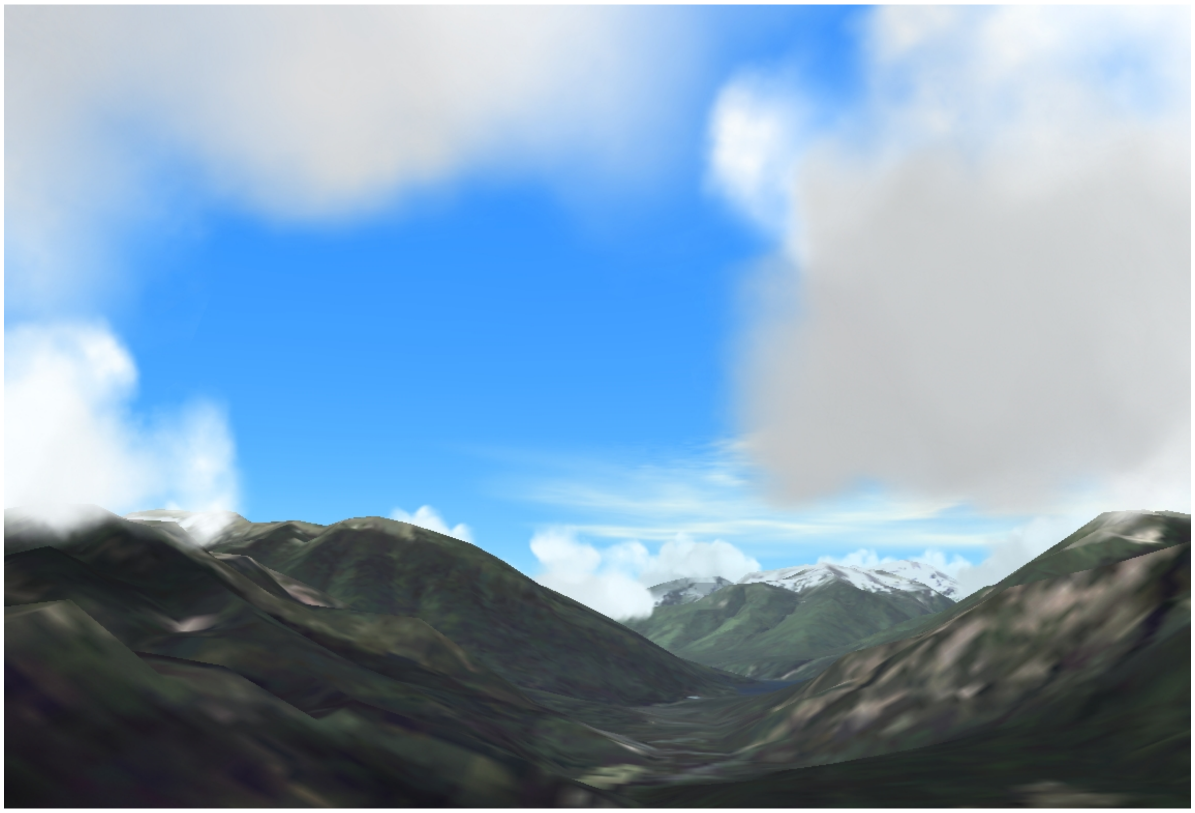


Figure 6. An example of procedurally generated cloud using Silver Lining software (Sundog, 2008).

With the advent of the 3D geospatial display systems such as Google Earth and NASA World Wind the benefit of displaying meteorological data is more apparent. The National Ocean Atmospheric Administration (NOAA) and the National Weather Service (NWS) have embraced the use of such visualization programs especially Google Earth. Satellite, radar images, and weather warnings are just a few of the weather products produced for visualization in Google Earth. While this shows a step past customer-based 3D programs, the images of satellite and radar are still the 2D images the meteorological community is familiar with. The composite radar image for the Monterey Bay area as seen in Figure 7 is a flat rendering of the composite radar data that has been added as a layer to the Google Earth program. This is useful, but still is not quite the 3D aspect of meteorological data outcome for geospatial visualization systems desired by this research.

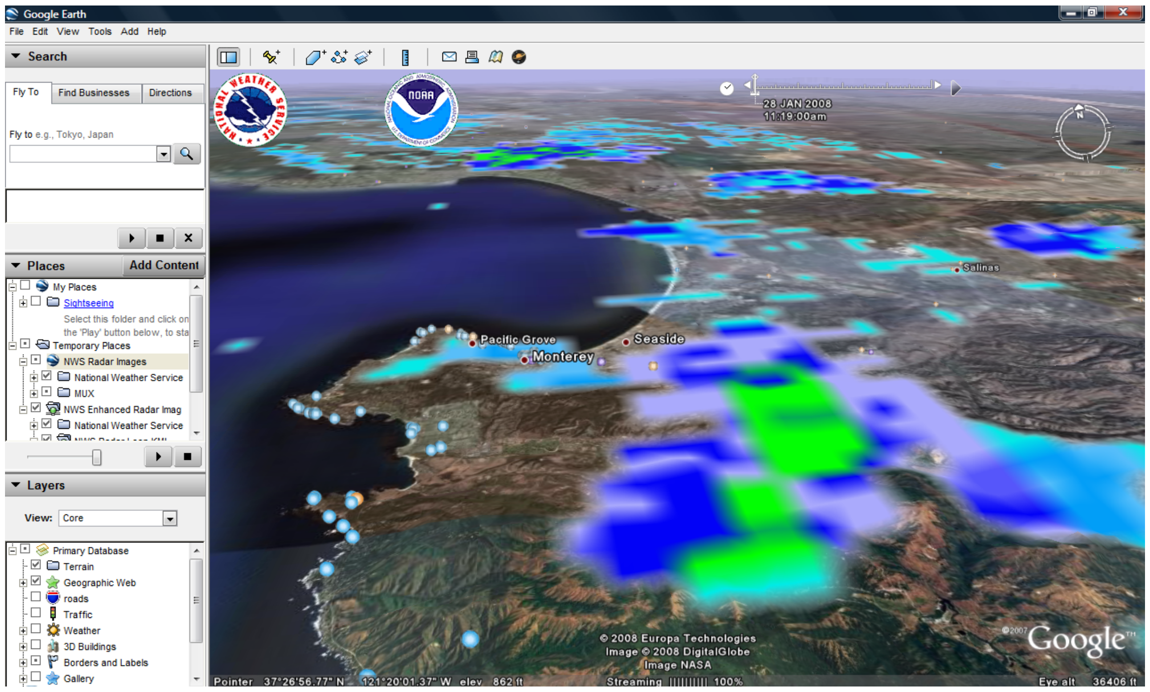


Figure 7. NWS composite radar layered at ground level viewed in Google Earth (Google Earth 2008).

D. GEOSPATIAL DISPLAY APPLICATIONS

With the determination made to research volumetric clouds for a real-time interactive visualization program the question is raised as to which program. Based on successful deployment, currently at the top of the list are, as suggested above, Google Earth and NASA World Wind. Both have their good points as well as bad. Looking at the programs in different areas such as ease of use, features, and the learning curve associated with its use, helps to determine which program is better suited toward the weather customer. The areas examined to determine if certain aspects of this thesis research will be supported by the programs are: detailing, if the program is an open source project, the type of programming language used, and the functionality.

1. Google Earth and NASA World Wind

While Google Earth was the simplest to use and quickest to learn, it does not completely support the methods to be used in the volumetric cloud formation. Additionally it is not an open source project, so software features cannot be scripted in to function with the volumetric clouds as created with a viewer facing image node. NASA World Wind is an open source project with a SDK available so that an application program can be written to allow the functionality necessary for the volumetric cloud method.

2. X3D Earth

There is one aspect of this research area that is a larger undertaking and beyond the scope of this thesis, but which is hopefully of keen interest to the DoD - a cross-service web-based 3D geospatial display system that can cover many roles of planning and training. One such possibility exists via the undertaking of X3D Earth by the Modeling, Virtual Environments and Simulation (MOVES) Institute of NPS as part of the X3D Earth Working Group of the Web3D Consortium. These groups are working to use "Web architecture, XML languages and open protocols to build a standards-based X3D Earth usable by governments, industry, scientists, academia and the general public." (Brutzman, 2006) An open source, scalable, sustainable platform can branch through all services and career fields. The very basis of this plan rests upon Extensible Markup Language (XML) standards, especially Extensible 3D (X3D) Graphics.

E. EXTENSIBLE MARKUP LANGUAGE (XML)

X3D Earth, Google Earth, and NASA World Wind have a common thread in their base languages. XML is a text-based language used to define the data languages that transmit and store data defined by the user or author. The XML language does not actually perform any functions or processes. These abilities are defined by specific tags as allocated by a particular schema. Google Earth's Keyhole Markup Language (KML) is a specialized markup language used in transporting large amounts of geographic data, and implemented by the Google Earth browser. X3D is a royalty-free International Standards Organization (ISO) compliant schema defining features for the viewing of XML data, largely 3D scenes. Due to the similarities between these two programs and their base languages, 3D scenes and objects are often transferable between the two. By applying an Extensible Stylesheet Language Transformation (XSLT), one schema or markup language can convert data into another. This ability is especially beneficial for projects with the magnitude of virtual 3D worlds and geospatial display systems. Additionally, it is beneficial to the prospects of this research.

F. X3D VIEWERS

There is large distinction between the definitions of viewable data models using XML schemas within visual application programs. The schema is a defined set of supported nodes or events. A viewer will then take the information given by the schema and render it to the screen. Both Google Earth and NASA World Wind have their own viewers as part of the program package. X3D has many dedicated viewers and is capable of being rendered in a web browser such as Microsoft Internet Explorer or Netscape using an X3D or VRML supporting plug-in. The benefits of diverse support can cause challenges, however, some functions and capabilities of X3D were not readily available using certain plug-ins, as found during this research. The benefit of a non dedicated viewer allows a user to choose a familiar browser and can far outweigh any difficulty. For this thesis effort Cortona plug-in by Parallel Graphics was the most useful. Publication of exemplars and results continues to improve the capabilities and conformance of all viewers, making X3D the most stable and reliable long-term option.

THIS PAGE INTENTIONALLY LEFT BLANK

III. METHODS AND PROCEDURES

A. VOLUMETRIC PROSPECTS

The first consideration to build volumetric clouds is to define how to fill a 3D area with appropriate geometry and textures. Initially the prospect of particle fields was an interesting option. As meteorologists know a cloud structure is comprised of particles of condensed water or ice, and drawing such objects seems like a most logical way to form a virtual cloud. The first issue was then how to define a particle in the XML and X3D schema chosen to represent as the geospatial display system. Within the X3D scene a particle is given a geometric shape and appearance properties. If a particle is given a radius in the micrometer range, relative to other scaled objects in the displayed scene, the resolution is far below the ability of the computer or graphics to render it. This required the particle size be increased to a viewable size. Filling a 3D area required many particle geometries. As more geometric shapes were added to a scene the graphics hardware needed to render and keep track of locations and appearance. The amount of memory necessary to process the particles and view them is enormous for any virtual GIS scene with clouds. It soon became apparent that the use of particles for volumetric filling was not yet a plausible technique.

While still wanting to completely fill a 3D area for a true volumetric cloud, placing layers of textured pixels was studied next. X3D includes a node with the ability to texture individual pixels or predefined 2D areas. This ability includes both color and transparency which is advantageous to cloud production. A complete sky cover using a limited number of 2D surfaces was developed. Scripting for these surfaces use a white color, but employed random generation of pixel transparency. Figure 8 displays an area of 200km by 200km comprised of 10 pixel textured 2D surfaces layered on top of each other at a distance of 100 meters apart. The surfaces remain static and a flythrough of the cloud structure is somewhat realistic for a cirrus type cloud layer. A visual artifact of this method is if the view position is directly parallel with the textured surfaces, the user can look directly out the side of the cloud structure and see blue sky. This certainly detracts from the realism. Besides the view difficulties, several other challenges arose before fully functional scenes can be developed. The layer structure in Figure 8 was created using 100 by 100 grid spacing for pixels. This implies each surface needs to generate 10,000 random pixel color and/or transparencies. Filling a defined cloud area with any more than approximately 100 surfaces created stack overflow and memory errors within the Java virtual machine engine. Additionally, the visual rendering of the surfaces was dependant on individual display resolution. When a display was given a high resolution, individual pixels were able to be distinguished. However, this created an image that did not visually represent a cloud well at all. Thus it was deemed not practical to fill volumetric areas using pixel layers. Fortunately the methods used by Microsoft in Flight Simulator 2004 were similar to nodes available in X3D, and cloud production turned its course in that direction.

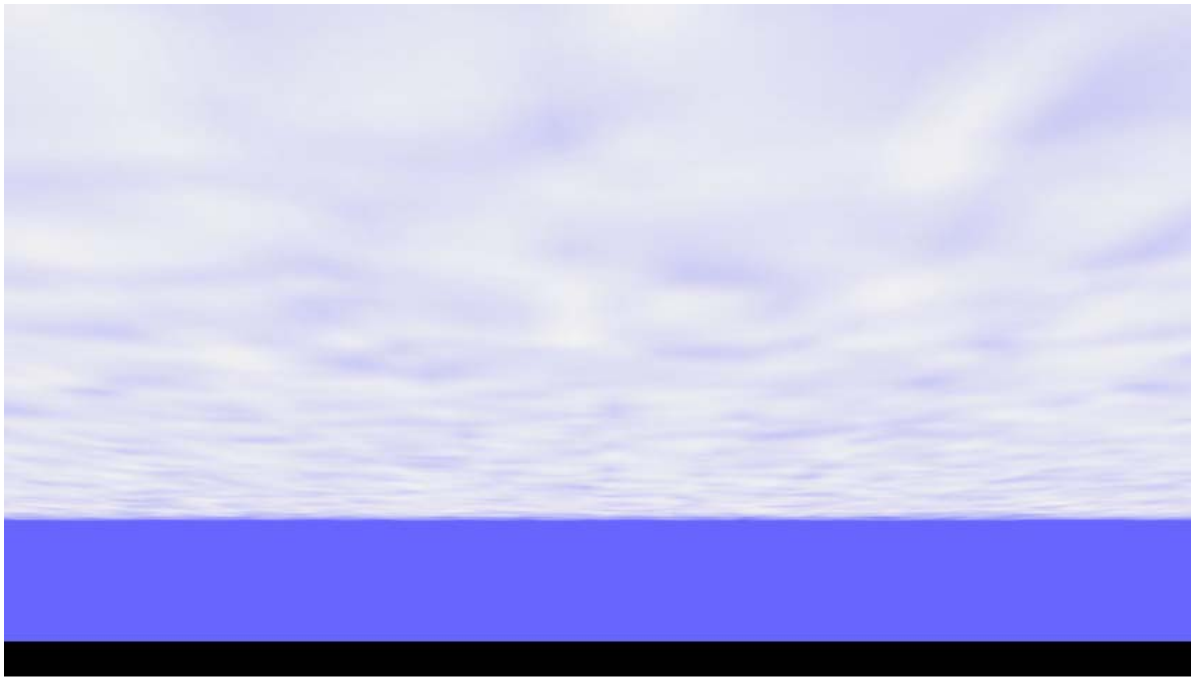


Figure 8. Sky cover layer using pixel texturing formed from 10 100x100 pixel layers.

B. CLOUD PRODUCTION

The general difficulty in producing model-driven clouds in virtual 3D is the process is both scientific and artistic. In Microsoft Flight Simulator 2004 the decision was made to not programmatically generate the clouds through an automated process. Developing the clouds manually gave more control to fine tune the artistic look of the cloud, and the automated process had parameterized equations whose result did not yield desirable results (Wang 2003). While being more difficult to produce virtual cloud structures in 3D for meteorological purposes, it was deemed beneficial to produce clouds procedurally. While preferable for a meteorological program, specialized programming needs to be accomplished for every different type of cloud. The many details of cloud structure and the method to best fill 3D regions for a particular type of cloud are daunting challenges and are more than can be completed in one thesis. However, the procedure outlined by this research cis a good start and be more fully developed to create a better visual representation and a more complete library of images and patterns for cloud production in virtual worlds.

1. Texture Images

Many methods for image generation are viable for the production of clouds as shown in this research. There is an essential understanding that the texture of the image, when rendered with a volumetric placement, must generate a visibly recognizable structure of the cloud it represents. It is these images, when textured and placed in and around the bulk of the cloud, which contain visual information regarding cloud density and texture. Ultimately, in conjunction with placement in a 3D space, these textures need to accurately portray the cloud type.

The most important visual aspect of realistic clouds in a virtual world is shadowing. To the human eye, shadowing brings out the three dimensional aspect of any object. During this thesis research images used for textures were produced manually. The GNU Image Manipulation Program (GIMP) is a robust and free image editing software that allowed many textures to be explored (GIMP, 2008). It also allows for scripting to produce images automatically. The file format chosen for the images was the Portable Network Graphics (PNG) format (PNG Home Site, 2008). This image format is open and royalty free making usage and deployment easier. Testing a variety of image textures in clouds revealed that solid textures did not produce a realistic effect. A certain amount of transparency dispersed throughout the texture especially along the edges of the square image, significantly aided in the cloud realism. PNG file format images, unlike other formats, can be layered with a transparent background allowing for these effects.

Figure 9 shows six images in 512x512 resolution that were created and tested for cloud structure. An additional capability within X3D is the manipulation of images based on an alpha value, which determines the transparency of the image or object being rendered. This benefit can be used for multiple purposes. It can be applied to entire clouds to vary appearances within the browser or viewer. Transparency can be programmed and applied to individual images on billboards to simulate the growth or dissipation in the cloud life cycle. This technique suggests that such a cloud production process can be utilized not only in static environment simulation, but also dynamic simulations.

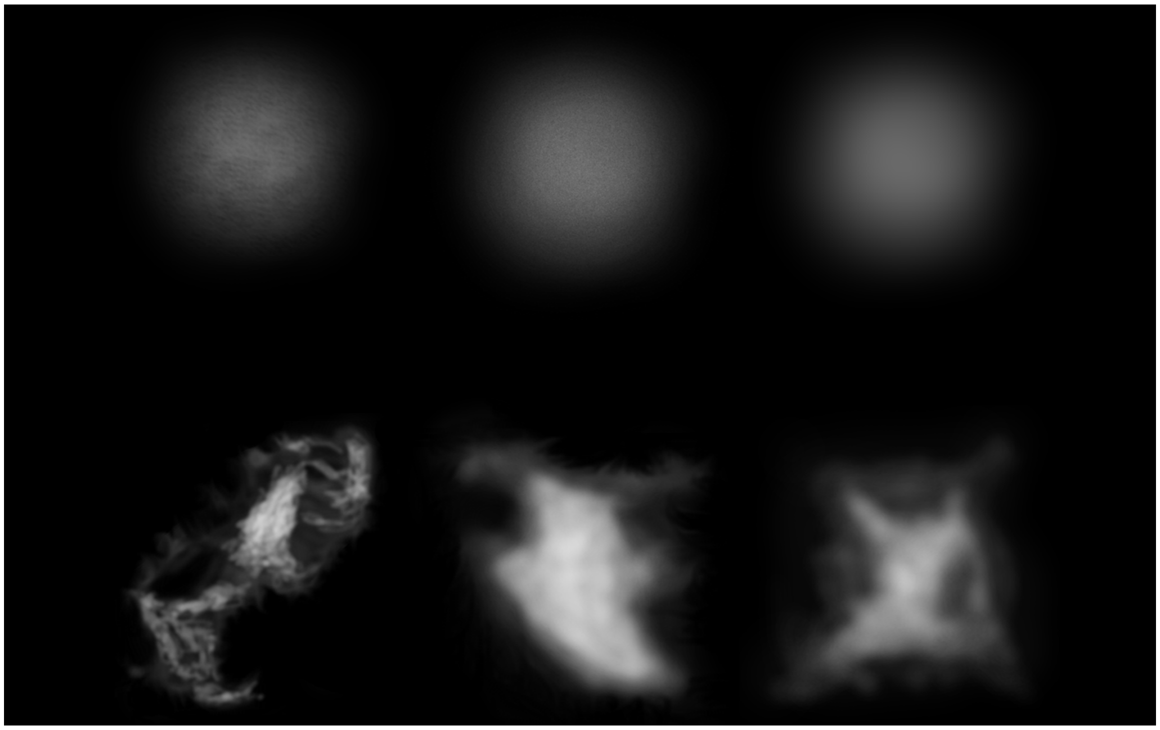
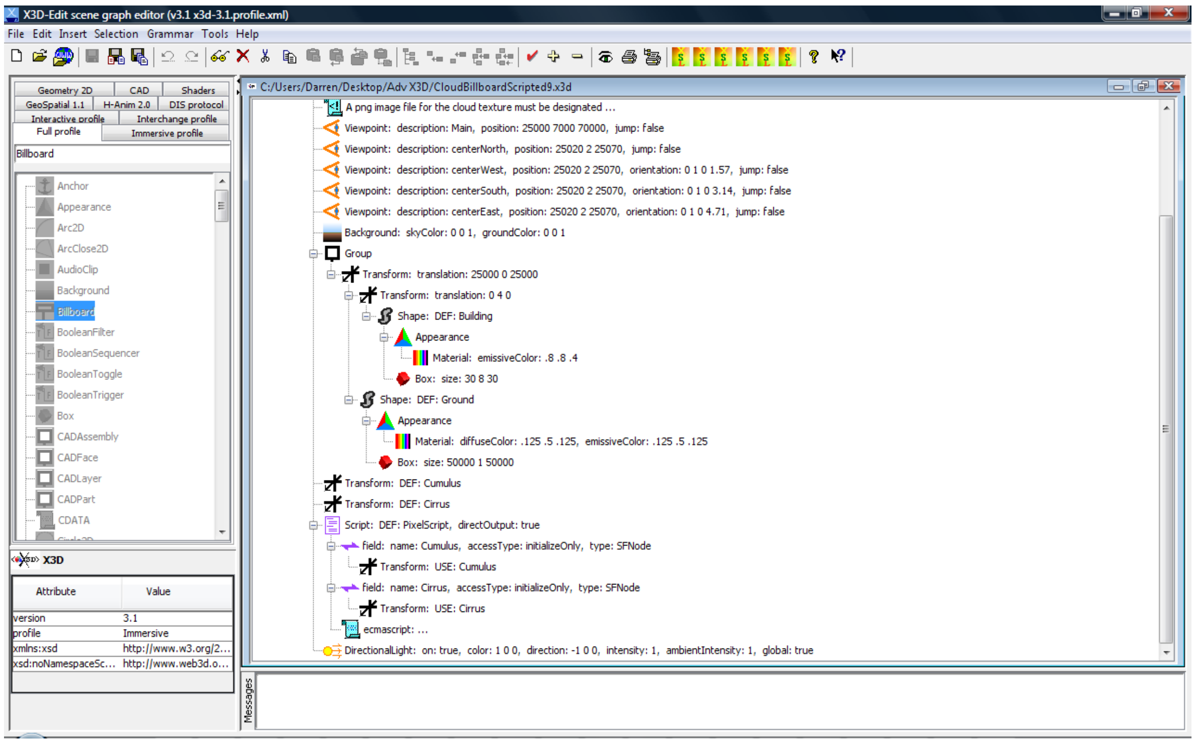


Figure 9. Individual PNG images created for billboard textures with white coloring and alpha layer.

2. X3D-Edit

During the research X3D-Edit version 3.1 was used for cloud production. X3D Edit is a scene graph editor that allows production of X3D scenes conforming to the X3D schema and a viewing renderer. Figure 10 is an example of the X3D Edit application (Brutzman, 2002).



Attribute	Value
version	3.1
profile	Immersive
xmlns:xsd	http://www.w3.org/2...
xsd:noNamespaceSc...	http://www.web3d.o...

```
C:/Users/Darren/Desktop/Adv X3D/CloudBillboardScripted9.x3d
A png image file for the cloud texture must be designated ...
Viewpoint:  description: Main,  position: 25000 7000 70000,  jump: false
Viewpoint:  description: centerNorth,  position: 25020 2 25070,  jump: false
Viewpoint:  description: centerWest,  position: 25020 2 25070,  orientation: 0 1 0 1.57,  jump: false
Viewpoint:  description: centerSouth,  position: 25020 2 25070,  orientation: 0 1 0 3.14,  jump: false
Viewpoint:  description: centerEast,  position: 25020 2 25070,  orientation: 0 1 0 4.71,  jump: false
Background:  skyColor: 0 0 1,  groundColor: 0 0 1
Group
  Transform:  translation: 25000 0 25000
    Transform:  translation: 0 4 0
      Shape:  DEF: Building
        Appearance
          Material:  emissiveColor: .8 .8 .4
        Box:  size: 30 8 30
    Shape:  DEF: Ground
      Appearance
        Material:  diffuseColor: .125 .5 .125,  emissiveColor: .125 .5 .125
      Box:  size: 50000 1 50000
Transform:  DEF: Cumulus
Transform:  DEF: Cirrus
Script:  DEF: PixelScript,  directOutput: true
  field:  name: Cumulus,  accessType: initializeOnly,  type: SFNode
    Transform:  USE: Cumulus
  field:  name: Cirrus,  accessType: initializeOnly,  type: SFNode
    Transform:  USE: Cirrus
  ecmascript: ...
DirectionalLight:  on: true,  color: 1 0 0,  direction: -1 0 0,  intensity: 1,  ambientIntensity: 1,  global: true
```

Figure 10. X3D Edit authoring tool for creation of Extensible 3D (X3D) graphics scenes showing an example of the scene graph for the cloud rending X3D file. (Brutzman, 2008)

Within X3D-Edit is the ability to define other scripting via JavaScript (i.e. EcmaScript) to accomplish various tasks. These scripted tasks represent the building or production of different cloud structures. The script created X3D nodes representing the cloud and in turn presented to the browser as a VRML file. While the methods researched and developed within this thesis are mostly manual, it is possible to completely author either an X3D(XML) or VRML file to produce the same cloud structures. For long-term data-driven production processes, the XML-based X3D encoding is preferable in order to take best advantage of web-based services and best practices.

3. Billboard Placement

The billboard node in X3D transforms the rotation of its geometry to face in a given direction defined by the user. The scripting for billboard placement was designed to not only create a viable 3D cloud, but to also minimize the number of billboards used. By minimizing the number of billboards, the number of surface vertices is minimized and computations requiring graphics resources are reduced. Several placement methods involving various coordinate systems were used in the initial research, and each method has its own relative merits in producing different cloud types and physical appearances. Regardless of which method is used, all computations for billboard placement are completed prior to the scene being rendered. This means graphic computations within the scene are not affected by the complexity of the calculations used to place the billboards.

4. Coordinate Systems

Cloud structure takes on different shapes, suggesting that designing models for their shape ought to utilize different geometric shapes as well. Cubic coordinates are given by the pseudo code:

$$X = C * Random()$$
$$Y = C * Random()$$
$$Z = C * Random()$$

Where C is a maximum value and Random() is a multiplier between zero and one yielding a random number between zero and the maximum value C. It is also important to note that each invocation of Random() provides a different random value. This method may be best used when a larger area of a more uniform cloud structure needs to be created, or a smaller area in which only a few image textures are needed. It is certainly the simplest to define and code. However, as larger quantities of billboards are rendered within the cubic area, its square dimensions become visible in the browser. Cylindrical coordinate's pseudo code is:

```
$\phi = \alpha * Random()$
$\theta = 2\pi * Random()$
$r = H * \tan\phi - Y * \tan\phi$
$then$
$X = r * \cos\theta$
$Y = H * Random()$
$Z = r * \sin\theta$
```

α represents a maximum angle between zero and $\pi/2$ corresponding to the angle ϕ, H is a defined maximum height of the cloud or section, and θ the circular angle placement around the disk at layer y. Conical coding is:

```
$\theta = 2\pi * Random()$
$r = M * Random()$
$then$
$X = r * \cos\theta$
$Y = H * Random()$
$Z = r * \sin\theta$
```

Here the placement of the billboards is random about a disk of radius r at a random height Y. Spherical coordinates can be coded as a complete sphere or half sphere by limiting ϕ to an angle of $\pi/2$:

$$\begin{aligned}
&\theta = 2\pi * Random() \\
&\phi = \pi * Random() \\
&r = M * Random() \\
&then \\
&X = r * \cos\theta * \sin\phi \\
&Y = r * \cos\phi \\
&Z = r * \sin\theta * \cos\phi
\end{aligned}$$

5. Random Placements

During cloud development a random placement of billboards was adopted. This randomness aids in varying the appearance from cloud to cloud or between cloud sections. Alternatively the number of billboards can be reduced by defining either the height Y or the radius r incrementally by dividing the maximum value by the total number of billboards filling the region. This also prevents, to a certain extent, the over layering of billboards within a limited region. Randomness will also become an important factor when it comes to model and data resolution. Without the ability to resolve down to the smallest cloud rendered within a 3D scene, it is still necessary to program for this ability. This aspect is further discussed in the chapter five. This effect is not meant to replace parameterizations, rather at this point in the production process it is merely used to circumvent the lack of resolution and proper parameterizations.

6. Shading

Shading of a cloud structure is an aspect in the cloud production that was not fully developed under this research. While accomplished to some extent to show a more detailed appearance, no calculations for the complexity of scattering were programmed. A darker shade was manually applied to the previously developed images. Within the scripting, as a 3D coordinate is calculated for individual billboards, if the placement is designated within the lower portions of the cloud, the darker image is assigned to that billboard. This method assumes the sunlight was from a purely nadir direction. A limitation of this method of shading an X3D billboard node is for thinner clouds or billboards that have been assigned a position not completely beneath the cloud is rendered as shaded. This is seen in side view of Figure 11 and top view of Figure 12. While not completely discernable from the side, viewing the cloud from above shows individual darker billboards that in the real world appear white. This erroneous shading can be avoided, but in turn complicated the shading algorithms slightly. Chapter IV details future work suitable to cloud shading.

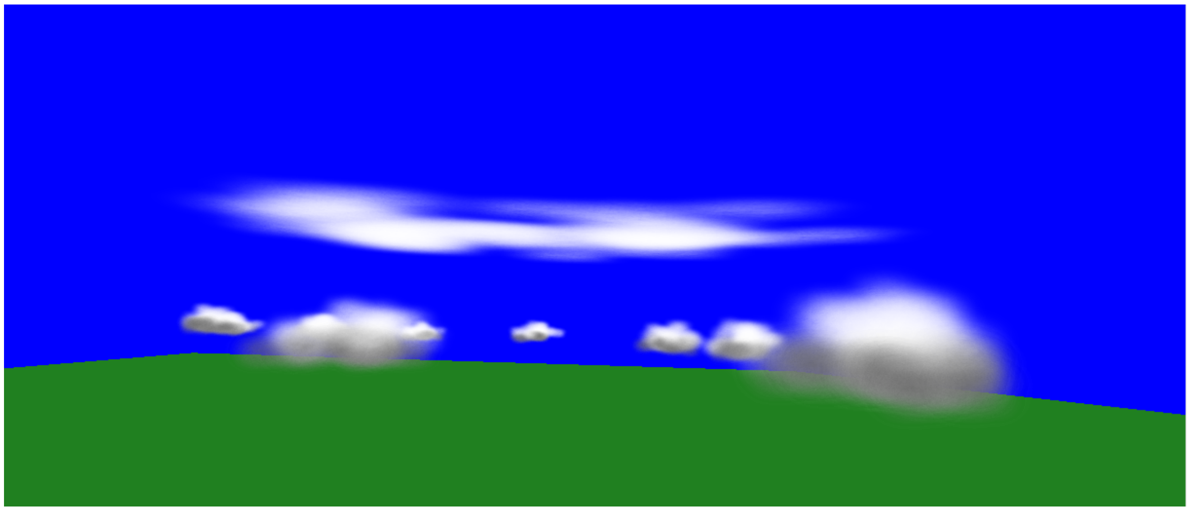


Figure 11. Side view of an X3D scene showing cirrus and multi shaded layer cumulus clouds.

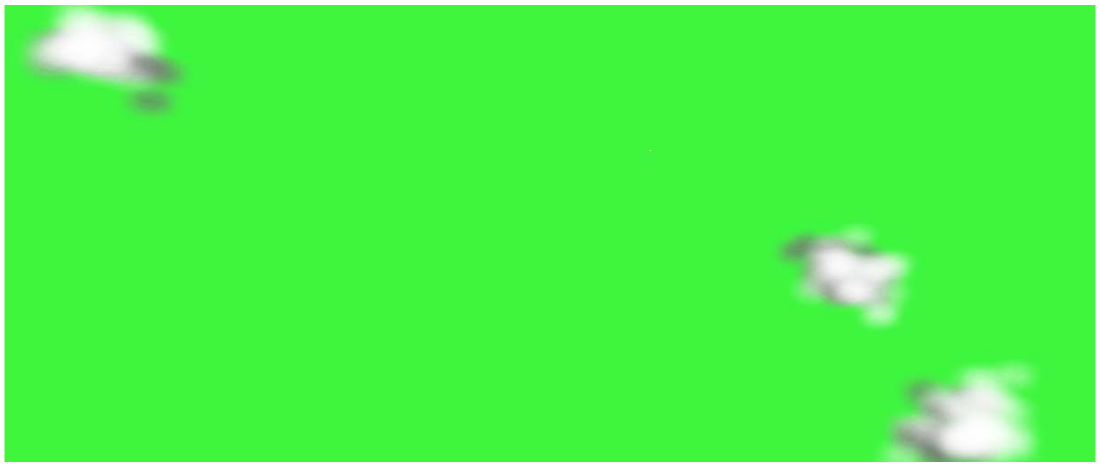


Figure 12. Top view of an X3D scene showing multi shaded layer cumulus clouds

C. PERFORMANCE CALCULATIONS

Performance measurement of the cloud rendering method was accomplished by two processes. The first is by simple calculation of total system memory allocated to the browser program and view plug-in to accomplish the rendering. For comparison purposes of future work the system properties used are an AMD Turion dual core 1.9 processor with 2 Gigabytes(GB) of random access memory(RAM). Internet Explorer 7 is used as the browser with a Cortona plug-in for VRML viewing. Besides system resource programs no other programs were running at the time of performance operations. Initial calculations use one of the predesigned image textures, which are 512x512 pixel resolution, normal scaling on a 100x100 meter flat X3D IndexedFaceSet geometry node. The resulting billboards are rendered in a cubic geometric area of 1 km. Table 1 outlines the results and Figure 13 charts the responses. These results assist in the computation for approximating the total number of billboards and associated cloud structures that can be rendered. Of note during the 10,000 billboard test the listed system occasionally reached its limit and returned a fatal runtime error requiring shutdown of the browser program. These results can vary based on how much system memory is allocated to other minor programs running in the background. Scale modifications were run on the same number of billboards however; scaling had no effect on the memory allocation. This suggests cloud size will have no effect on the scene rendering, but rather simply the number of billboards used will dictate the grid size able to be rendered. A moderately sized cumulus cloud with a base of approximately 1km built by previously described scripting methods using 5 cloud sections with 8 billboards each creates a total of 40 billboards. Thus 15 square km complete cumulus sky coverage requires 225 clouds and 9000 billboards. This is close approaching the limit of 10000 for the test system. It is important to note these tests were for one cloud type production only. This 15 km limit for cumulus does not take into account any additional layers and further billboards that may be needed to complete the rendering. This is a severe limitation to overcome. There are a couple concepts that can be further researched to expand the grid memory limits. The first is the use of imposters or full cloud image on one texture. The second would be to use a Level of Detail X3D node to limit the number of billboards used based on either direction or distance from any given cloud. These concepts are further discussed in Chapter IV.

Table 1. Memory performance for 1 to 10000 billboards.

Billboards	1	10	100	1000	2000	5000	10000
Memory (mb)	31	41	55	150	225	485	900

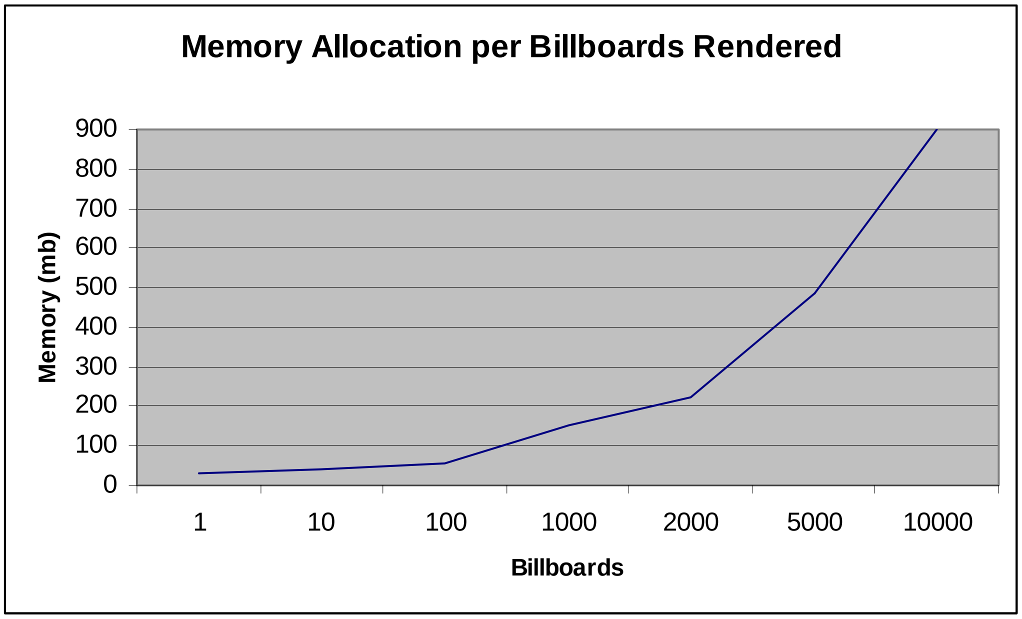


Figure 13. Memory allocation increase with increased billboard production.

A second performance test was completed that applies to the visual notion of navigation through the virtual world. A snapshot of the sky about any location will most likely be preferable to the majority of customers from a meteorological standpoint. However, to be able to navigate through a scene is advantageous for mission planning whether by ground or flight. To test for limitations in navigation, frame rate is computed within the browser. See Appendix 2 for the XML code. The frame rate indicates how often the browser/viewer combination can recalculate and display the 3D scene. The frame rate speed is not completely dependant on the number of billboards used within the API script, but other factors to include the filled pixels within the view frustum or visible screen. The method for frame rate testing of the cloud scene can be accomplished by many methods. The method used for thesis testing was simply layering billboards horizontally behind each other and to automatically navigate through the scene. This method allows for the frames per second (FPS) calculation with an increasing number of filled pixels and layers in the visible window. The Cortona viewer has the ability to designate one of four different navigation speeds in essence changing the navigation FPS rate. Additionally the navigation speed can be increased within the browser by pressing the shift key during movement. This test can not therefore, be used as a finite limitation of frame rate. It is shown as a benchmark for limitations, as well as future work. Speed designated by the user, in addition to whether the navigation is manual or automated will affect the frame rate. Table 2 outlines the results and Figure 14 charts the responses. There are many practical applications to discuss the minimum FPS for a software program. However, the fact is Phase Altering Line (PAL), National Television System Committee (NTSC), Digital Television (DTV), commercial films, computer video cards and screens range from the 20's to hundreds of FPS. The basis for many applications usually rests upon the ability of the human eye to make a distinction between moving images. This distinction can vary based on whether the image is a flicker of light or the blur of a moving object. For the general purposes of this endeavor a common benchmark is approximately 15 FPS. However, during the testing, a measured 11 FPS showed very little "jump" between displayed images. If the same calculation is applied to this speed test for navigation, the acceptable number of billboards is limited to 1,000. This is a far less than memory limitation of 10,000. Applying this number to the 1 km sized cumulus, it seems a mere 25 clouds can be rendered for navigational purposes. For complete sky coverage this is 5 square km.

Table 2. Frame rate performance in frames per second for 1 to 10000 billboards.

Billboards	1	10	100	1000	2000	5000	10000
FPS	35	31	25	11	7	3	1.5

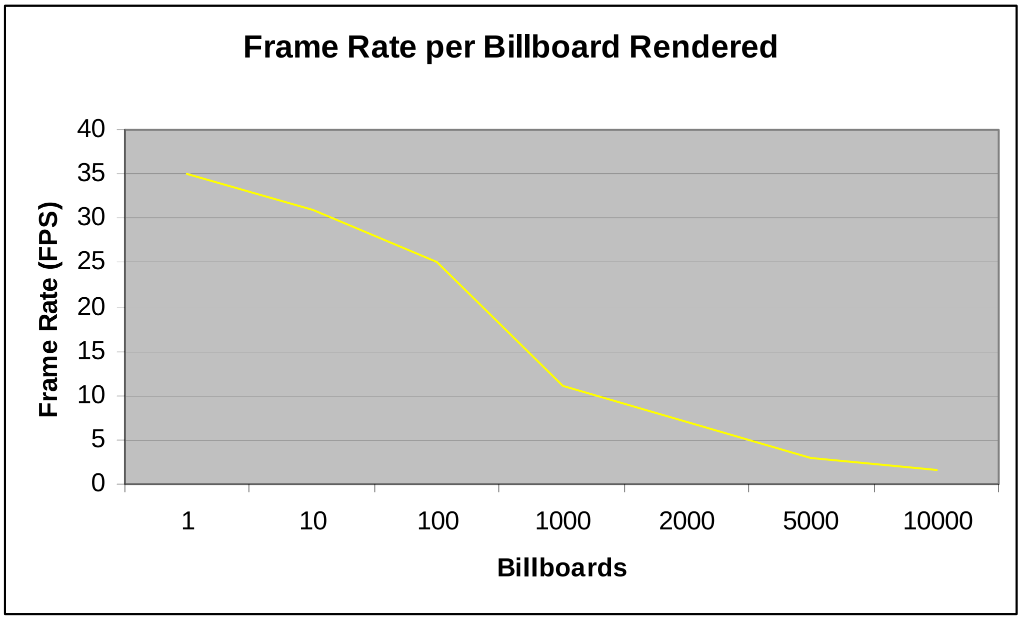


Figure 14. Decline in frame rate with increased billboard production.

While total memory is system independent using the same methods, limitations of memory and frame rate are system dependant. As with all computer software any enhancements in system speed and related technology will improve the individual software performance. With these methods shown there is a definite need to improve the cloud rendering with a fewer number of billboards to maximize the grid area viewed at any given time. Section B in the Chapter V discusses some of these methods and other possibilities.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DATA PROCESS

A. METEOROLOGICAL INPUTS

The process to fill the GIS scene with procedurally created clouds is straight forward and depicted in Figure 15.

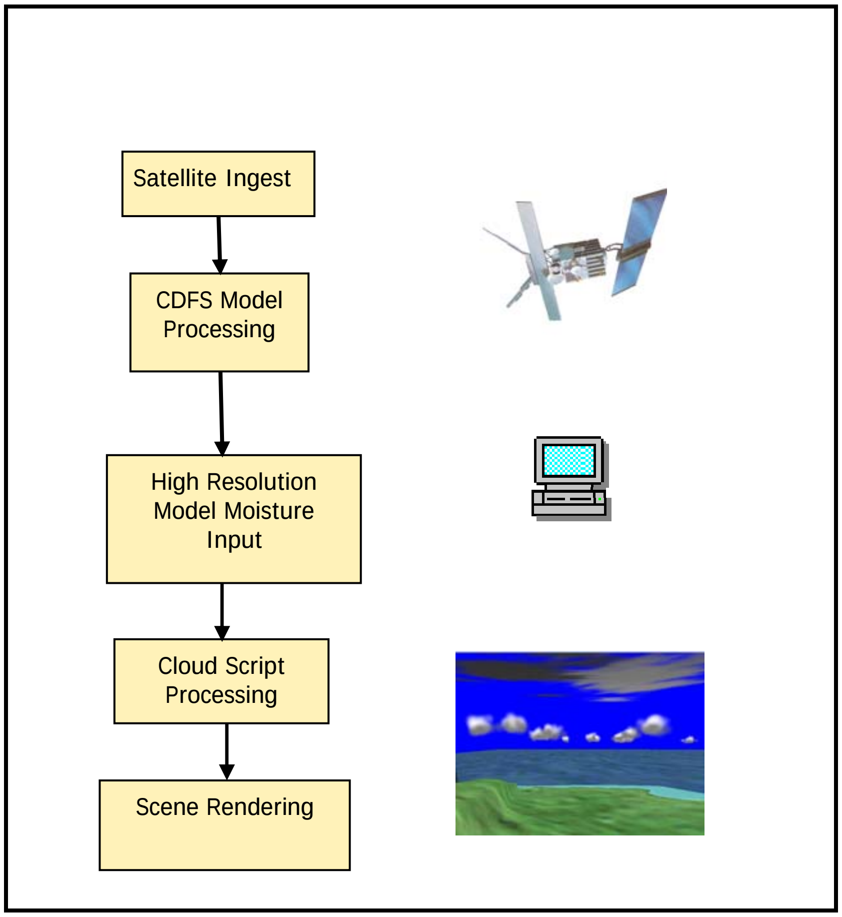


Figure 15. Flowchart depicting the rendering process from raw data to scene visualization on a user side computer.

While creating 3D clouds is the goal of this thesis research, there is a need to locate a data source for their production. While it is beyond the scope of this research to determine the most meteorologically or statistically correct data, certain features of data are desirable. Many different methods can be used to define the 3D region of a cloud, and many have been researched for the purpose of creating 3D clouds. NPS studies have been accomplished using IR satellite data (Owen 1998) and a combination of satellite, upper air soundings, and surface observations (Stone 1992). While other options include radar or stereographic satellite compilations, the realization of available data for any particular area remains a challenge. Ultimate remote sensing capabilities for 3D cloud structure include recent use of cloud profiling radars on satellites.

The CloudSat project images (Figure 16) show cloud and storm structure, intensity and rainfall rates. Never before has such detailed structure been available for such a complete atmospheric profile. However, for thesis research purposes use of this data by itself is not practical as it only represents a thin horizontal slice of atmosphere. Inclusion of this data in future research could prove invaluable. However, for a nearly complete 3D data set that allows for the definition of clouds using a weather forecast model is proposed.

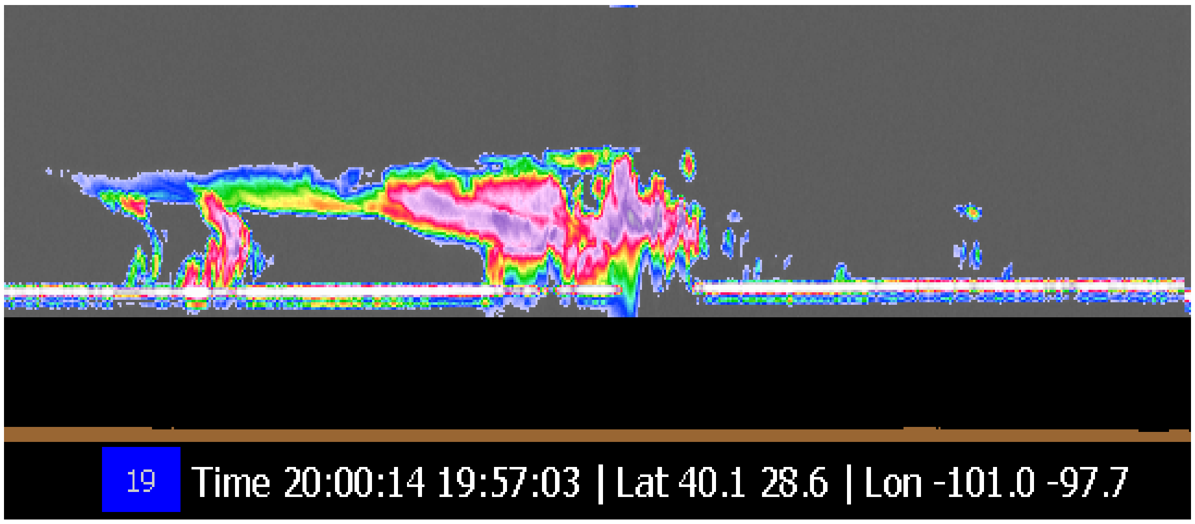


Figure 16. CloudSat profile imagery for storms over Texas (Colorado State University 2008).

While it is beneficial for procedurally generated clouds to be rendered independent of any particular data, the methods researched and produced in XML at minimum need a cloud classification.

1. Cloud Depiction and Forecast System (CDFS) II

The CDFS can serve at least as a partial input to the cloud rendering method. The model incorporates satellite and conventional analysis through four levels of processing in its final output, the Worldwide Merged Analysis as depicted in Figure 17.

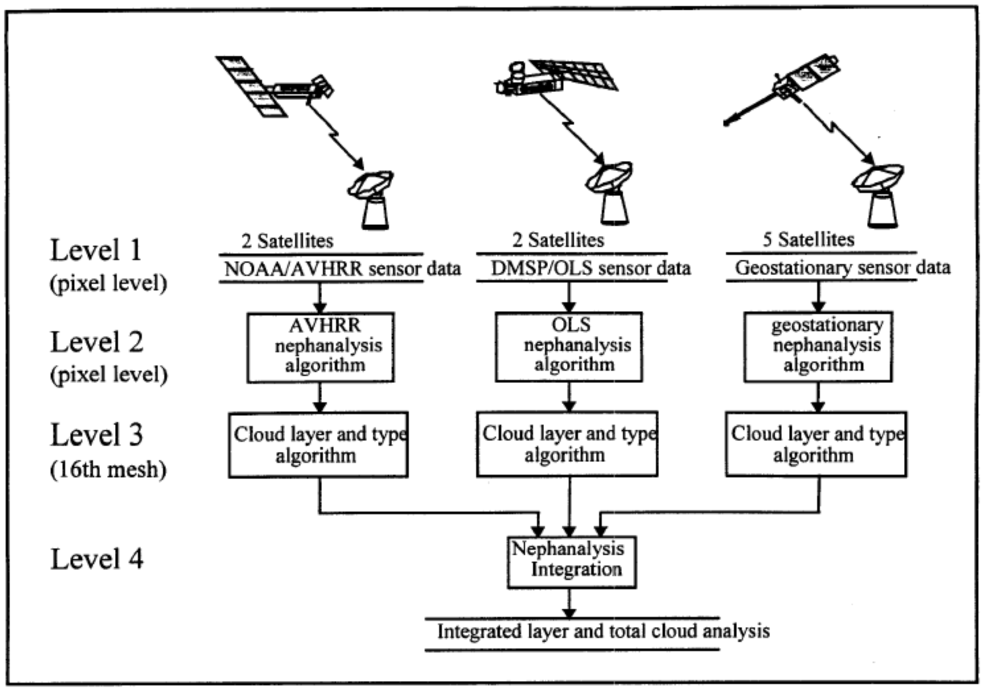

Level 1
(pixel level)
Level 2
(pixel level)
Level 3
(16th mesh)
Level 4
2 Satellites
NOAA/AVHRR sensor data
2 Satellites
DMSP/OLS sensor data
5 Satellites
Geostationary sensor data
AVHRR nephanalysis algorithm
OLS nephanalysis algorithm
geostationary nephanalysis algorithm
Cloud layer and type algorithm
Cloud layer and type algorithm
Cloud layer and type algorithm
Nephanalysis Integration
Integrated layer and total cloud analysis

Figure 2: CDFS II Cloud Analysis

Figure 17. CDFS multilevel data, processing, and analysis methods for creating the Worldwide Merged Cloud Analysis (Plonski 1998).

Cloud typing begins in level two where clouds are detected from the raw data and classified into three initial groups of cirrus, cumulonimbus, and fog based on spectral analysis. In level three, further typing of cumuliform or stratiform is calculated based on spatial extent of the cloud formation. Finally, based on data, height of the cloud is classified as low, medium or high cloud. The model then classifies the cloud as one of nine cloud types. Cloud layer thickness is predefined and assigned based on classification. The cloud base is then calculated by subtracting the assigned thickness from the cloud top height, and can later be changed when level four processing is accomplished with real world observations and other conventional reports are merged.

2. Combined Sources

This model discussion does not imply there are not superior algorithmic methods for the determination of cloud regions in 3D space. It is assumed beneficial and practical for cloud rendering methods to have a high resolution model in both the horizontal and vertical. Using a high resolution model saves further extrapolations to be accomplished within the cloud rendering API, introducing further error. Regardless of model resolution, a certain amount of approximation might be necessary. The quantitative resolution visible within a 3D virtual world is essentially in the sub meter range. As discussed the method of filling 3D space is guided by a particular cloud type structure. Different types of clouds are associated with different methods and calculations as well as base images. Thus, determination of cloud typing within the model is useful. These two processes of cloud region and cloud type determination can also presumably be obtained from different sources, so there is no need to find a single model or data set defining the two aspects. Variables of cloud mixing ratio and ice mixing ratio are common to many forecast models. A separate model can provide the high resolution spatial input and a typing model such as CDFS, or 3DCAS can drive the rendering API.

This thesis concept initially was to research the ability to create clouds near real time for the purpose of a theater sensing strategy. The proof of concept question becomes how to quantify the accurate depiction of 3D GIS or virtual world clouds. Ultimately the closer the virtual clouds appear to the real world clouds of the same time, the more accurate the production rendering can be considered. This comparison is essential in a cloud rendering process proof of concept for operational use. In addition to approximated areas of clouds being accurate, relaying to a customer the visual aspect of a weather scenario associated with a certain type of cloud ought to be accomplished by the rendering. The comparison is essentially a model and rendering API verification.

The process of cloud rendering for meteorological purposes is only limited by model input. As discussed, the dual model input is currently seen as the most logical choice. However, the process can be employed for nowcast or forecast purposes. A single model would need to be developed to drive both region and method. The algorithms as used by CDFS for cloud typing are derived from current satellite data. Therefore algorithms for cloud typing would need to be developed in conjunction to the spatial regions associated with clouds.

B. INPUT METHODS

Under current design, meteorological input occurs in multiple areas dependent on type of model data used. Table 3 is a general outline for the cloud production process. The cloud rendering process and model input will vary slightly depending on the cloud type.

Table 3. Individual steps of the cloud production process programming.

Step 1	Origin Designation
Step 2	Cloud Location
Step 3	Cloud Type
Step 4	Cloud Quantity
Step 5	Cloud Size - Cloud Sections - Section Size - Billboard Size - Billboard Quantity
Step 6	Cloud Production - Coordinate System Used - Billboard Texture Selection - Billboard Rotation - Billboard Geometry

Step 1 – Origin Designation: The first step entails designating the general grid location for display. If the model file name or header data contains location information, automatic retrieval and designation within the API is suggested. The X3D XML schema allows for a curved earth surface or orientation on a flat plane, based on user needs. Additionally, objects can be assigned a location based on simple scene coordinate system or in geospatial coordinates. Regardless of how the earth's surface is displayed X3D schema uses a coordinate location designated in meters. This is useful when using model data gridded in meters or km. For the cloud rendering, a corner of the grid can be designated as the scene “origin” for the local coordinate system.

Step 2 - Cloud Location: Based on their origin, clouds will be placed by coordinates designated in meters and heights relative to the surface. Care must be taken here to adjust for height if elevation data is used to display the surface. Surface height must be added to model data height if the model layer is terrain following. There will invariably be some randomness or artistic license taken in placement. Display resolution in the virtual world will simply be higher than the mixture of model data used.

Step 3 - Cloud Type: Based on the data from CDFS or similar cloud typing model the API will search for the cataloged method. This implies a minimum of nine different cloud creation methods be created. Research methods of cloud typing and differentiating the visual aspects of clouds are virtually endless.

Step 4 - Cloud Quantity: Related to the placement is the cloud quantity. The current method design is for user input to define the number of clouds placed within a quadrant. As model data is included the number of clouds will vary based on cloud type, number of adjacent model grid sections designated with clouds, and cross referenced with cloud size. Step four would likely benefit from the dual model input as higher resolution input from humidity, ice mixing ratio or water mixing ratio can assist in the quantity determination.

Step 5 - Cloud Size: Cloud size is again directly related to cloud type and can partially be defined by the layer depth from CDFS. Additional steps are required within the API as it was found that construction of the cloud structure was best handled with multiple sections. The number of sections will be designated based on cloud type and overall cloud size. Section size will be calculated as a percentage of the cloud size and can be standard or given a range for visual diversity. Part of defining the size of the cloud is related to the size of the textured billboard. Thought and testing must be given to the size of the billboard. When textures are applied to the stated size, the image will be displayed as created. Cloud size can also be varied by scaling. Scaling, as seen in testing, does not affect the performance, but it will affect the appearance of the texture. This affect can be favorable or not, which dictates methods must be tested in various and multiple ways for a desirable final visualization. Billboard quantity will also assist in the cloud size, but also the cloud density. Some correlation is needed between the amounts of moisture variable as described in step four, as well as reference the texture used in step six.

Step 6 - Cloud Production: The rest of the cloud rendering process will be defined by model inputs already used in previous steps. The cloud type input from CDFS will drive the coordinate system used for billboard placement, billboard geometry, and rotation. Billboard selection is guided by the dual input of type and moisture variables as well as shading principles discussed below in this section. The proposal exists that some cloud types may require other inputs. For example, which direction an overshooting top is developed would be dependant on wind direction and speed. Wind patterns can also be used in the general tilt of a larger cumulonimbus structure. Cloud physics and dynamics are certainly a useful knowledge in this area, but once again how to visually represent the cloud phenomena with meteorological inputs rests upon the data available.

V. RENDERING IMPROVEMENTS

A. SHADING

The shading of cloud structures in computer graphics terms is related to the meteorological principle of the scattering of visible light by the water or ice particles within the cloud. The computer graphics chore is the simulation of these principles of scattering. This principle of shading takes in several different aspects and can be defined as complex or as simply as the designer likes. In practice, the shading should represent a realistic cloud to pass along the intended weather effects of the cloud structure to the viewer. In Chapter III, example of a simple characteristic for clouds to be darker in lower levels is shown. This, of course, is not a color difference, but rather the lack of visible light at these levels. Figure 18 shows light from a source is scattered away and toward the path to the eye, and progressive distances along this path have fewer photons available to observe. Realistically if given meteorological information to render the virtual scene, the sun position would be defined by the time of day as defined in the data set. The shading would then be darkened in positions of the cloud that are opposite the position of the sun. Additionally, the algorithm for cloud shading can be calculated to increase darkness based on cloud depth in relation to the sun. The shading algorithm can be further developed by including computation of ambient light scattered into the direction of the camera view angle. A certain amount of shading control can be accomplished through texture manipulation.

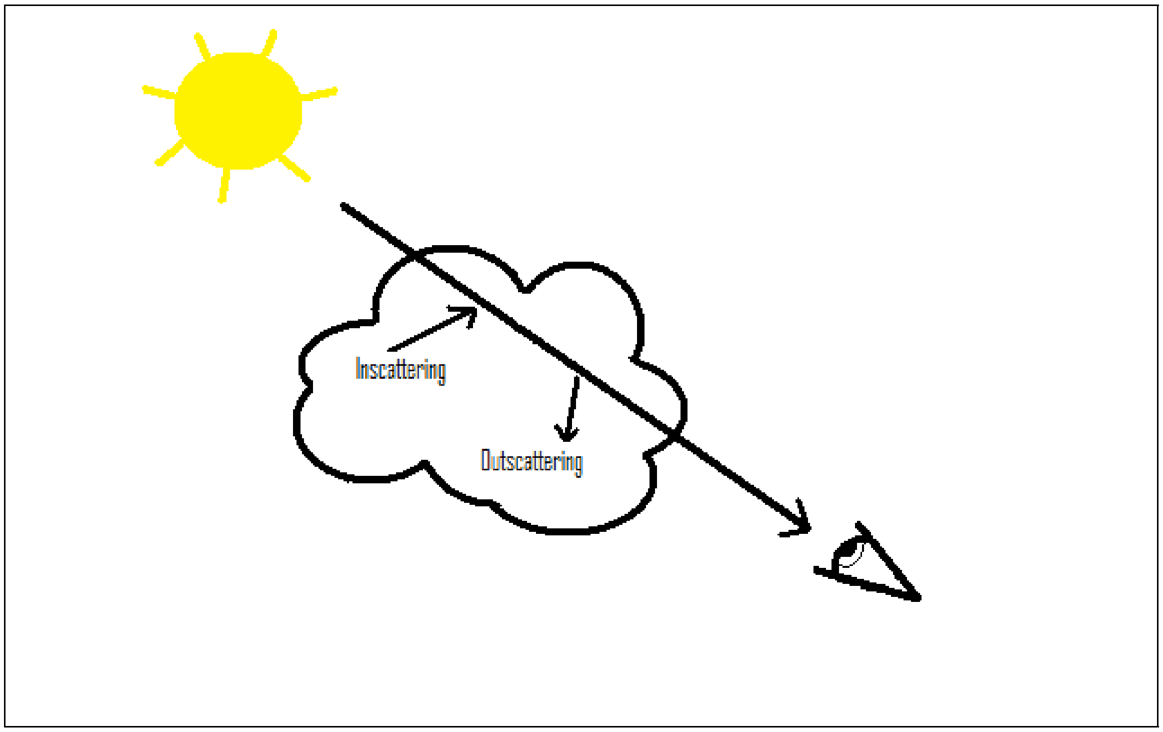


Figure 18. Single source light principles for shading indicating scatterings along the light path to the viewer.

Further control can be accomplished within X3D by one of three different light nodes: directional light, point light, and spot light. With introduction of internal scene lighting, consideration must be taken into account for shape of the textured billboards. Internal lighting is accomplished on geometry surface normal. Given a flat surface any internal lighting will be rendered the same across the billboard texture and across the entire cloud structure. This effect can be changed slightly by giving a certain rotation to the defined geometry prior to designating the billboard node. The geometric shape allocated for the billboard can also affect this internal lighting. Due to the changing normal across a curved surface, the billboard will show a difference of applied internal lighting. Testing was attempted with different categories of curved surfaces. When mapping to a complete sphere structure, image textures must be edited by scaling the image smaller. When texture mapping occurs the smaller image is then mapped to only one side of the sphere, which is preferable for the billboard rendering. Additionally the sphere node within X3D is not editable to increase the number of vertices defining the sphere. This presents an unfavorable visual artifact of seams and vertices from the defined sphere visible on the image texture. Half spheres were also tested using extrusions of arcs, however in texture mapping the image is applied to the arc instead of the entire hemisphere, rendering this geometry unusable.

Non-Uniform Rational B-Splines (NURBS) curves are an available node within the X3D schema. The surfaces formed by NURBS present a smoother surface than a defined sphere. Testing was attempted, however the node was not supported by the Cortona plug-in. More research is needed to determine shading control of the billboard surface.

A separate approach would be to utilize the ShaderProgram and ProgramShader nodes available in X3D. These nodes give separate programs the ability to define shading per vertex or per fragment within the X3D scene. Many different methods can be utilized in this separate program, and no suggested method can be made at this time. Research points to using Open Graphics Library Shading Language (GLSL) or Phong shaders. These shaders can be used to replace manual production of image textures representing a more realistic cloud, however, this is acknowledged as a more advanced approach.

B. PERFORMANCE

As can be seen from the performance tests, memory allocations and frame rate are very limiting to the display of large spatial regions of clouds. Further research must employ system memory cost saving measures to display as many visual aspects of current weather and cloud conditions while maximizing system resources.

1. Imposters

Taking a cue from the previous work of Microsoft and Sun Dog Software, complete images of clouds can be displayed on one dedicated piece of geometry. Questions include: At what range from the user or camera view would using the imposter process begin? This is surely a question that must be tested within the designated browser/viewer. This question of range may be influenced by the purpose of the rendering whether the view be static or for navigational purposes. If a static view for the geolocation is used, imposters can simply be images of clouds. This significantly decreases further effort in texture production. These images communicate the weather associated within the region, at a fraction of the system memory cost. If future design incorporates the option to navigate through the virtual world, imposter creation becomes more challenging. One idea is to create procedural imposters. With procedural cloud production, every cloud structure is slightly different. At some distance from the cloud structure the imposter is replaced by a 3D procedural cloud. For visual accuracy the imposter needs to be very similar in size and shape to the imposter. A second option is to encompass the view position with a textured ring of imposters. The ring of imposters moves in conjunction with the view position. This option is slightly limiting in that these "distant" cloud structures never actually become closer to the camera view. Thus if the model data suggests a different cloud pattern at some point the images on the surrounding ring would need to be replaced. If the cloud structures have been in position throughout the entire navigation and such an exchange can appear abrupt.

2. Texture Switching

Another option exists, but is likely not as computationally cost effective as single image imposters. This alternative is to limit the number of rendered billboards dependant on the camera view direction and range. If a normal cloud structure is made of 40 texture images only a certain number of images are actually seen at any given angle. If the images are semi transparent there is a certain perceived color meshing between the front and rear image giving the appearance of thickness. This density can be compensated for by simply assigning a less transparent or nontransparent image to the few billboards in the view of the camera. As the camera view came within a defined distance from the cloud structure, the remaining billboards are rendered and the images replaced by the regular 3D textures. X3D and GIS systems calculate or perform this function as a Level of Detail (LOD) determination as defined by a node. LOD is a method of loading or unloading data to improve visualization and performance.

C. DISPLAY

While these methods are developed for the purpose of insertion into present or future GIS systems, it is not too difficult to design a general display system unique for meteorological purposes. A concern over insertion into a GIS platform is the focus of the application. Certain GIS applications place a larger emphasis on high resolution elevation data and imagery. These attributes are important but also require sharing system resources. As seen by the performance testing, current ability is limited. This being the case, a simpler virtual world design will allow for a greater spatial area to be displayed. Again the option for surface displays is varied. Determination would need to be made between a curved or flat Earth surface and with or without elevation data. Conversely, switching between Earth surfaces can be accomplished as a user directed input.

VI. CONCLUSION AND RECOMMENDATIONS

A. CONCLUSIONS

Through the general methods researched and developed here, a simple XML language representation of 3D clouds has been shown. At the present time, X3D is the most useful and accessible schema to use for the cloud structure production. The available nodes in the X3D schema are fully compatible with production methods. Texture images were produced manually using image editing software while cloud structure, position, and lighting are automated within the file scripting. An interesting concept of this method is that the visualization processing step currently in use by the Joint Air Force and Army Information Network (JAAWIN) can be bypassed. The rendering API can be programmed to pull data directly from a server thus saving processing power on the JAAWIN server side.

Limitations are apparent in current versions of the cloud rendering. Most significant is the limitation of textures by memory. For these methods to be productive and useful, a balance of volumetric and 2D cloud structures must be used. Additionally a lack of full compatibility between the X3D schema and plug-in viewers hampered the efforts. To take full advantage of available X3D nodes a fully X3D compliant viewer plug-in must be used.

With further research and development this 3D cloud rendering process can be utilized as the next generation of meteorological visualization products.

B. RECOMMENDATIONS FOR FUTURE WORK

To further improve production methods the following tasks are suggested:

- Edit the API script to ingest and display clouds from model data: CDFS typing and high spatial resolution moisture model data.
- Explore the option of using Sun Dog Software's Silver Lining SDK. A further option is to contract the company to develop the processes. For a more rapid production of 3D cloud rendering, this would be the preferred method.
- Further develop a minimum of 9 methods of cloud production to match the types designated by CDFS. This process would include image texture improvement for an accurate cloud representation. In contrast artistically created clouds of the nine types can be used. The limitations of memory would still be present, but this can enhance visual features and simplify production process.
- Research an automated texture production process. If automated this can assist in the generation of 2D cloud structures
- Select or design a rendering scene for the visualization. This selection can be through working with Google Earth, NASA World Wind, or X3D Earth, Sun Dog Software, or implementing a low resolution elevation and Earth surface image package. Based on this decision it may or may not be necessary to also choose a viewer plug-in to negate the X3D compatibility issues.
- Develop a testing for proof of concept purposes. The concept challenge will be an accurate depiction of the current sky conditions and atmospheric state that they represent.

APPENDIX A. EXEMPLAR X3D SCENES

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE                    X3D                    PUBLIC
"http://www.web3d.org/specifications/x3d-3.1.dtd"

"file:///www.web3d.org/TaskGroups/x3d/translation/x3d-
3.1.dtd">
<!--Warning:  transitional DOCTYPE in source .x3d file-->
<X3D profile="Immersive" version="3.1"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
xsd:noNamespaceSchemaLocation="http://www.web3d.org/specific
ations/x3d-3.1.xsd">
  <head>
    <meta content="CloudScene.x3d" name="title"/>
    <meta
      content="X3D  utilizing  ecmascript  to  develop  quasi
volumetric  3D  clouds  from  png  image  textured  billboard
nodes." name="description"/>
    <meta content="Capt Darren W. Murphy" name="creator"/>
    <meta content="1 Nov 2007" name="created"/>
    <meta content="23 March 2008" name="modified"/>
    <meta

content="https://savage.nps.edu/Savage/Environment/Meteorolo
gy/CloudScene.x3d" name="identifier"/>
    <meta
      content="Additional   png   images   to   use   in   cloud
rendering:&#10;https://savage.nps.edu/Savage/Environment/Met
eorology/Cloudtexture1.png&#10;https://savage.nps.edu/Savage
/Environment/Meteorology/Cloudtexture1_1.png&#10;https://sav
age.nps.edu/Savage/Environment/Meteorology/Cloudtexture1_2.p
ng&#10;https://savage.nps.edu/Savage/Environment/Meteorology
/Cloudtexture1_3.png&#10;https://savage.nps.edu/Savage/Envir
onment/Meteorology/Cloudtexture2.png&#10;https://savage.nps.
edu/Savage/Environment/Meteorology/Cloudtexture3.png&#10;htt
ps://savage.nps.edu/Savage/Environment/Meteorology/Cloudtext
ure4.png&#10;https://savage.nps.edu/Savage/Environment/Meteo
rology/Cloudtexture6.png&#10;https://savage.nps.edu/Savage/E
```

```
nvironment/Meteorology/Cloudtexture7.png&#10;https://savage.
nps.edu/Savage/Environment/Meteorology/Spheretexture.png&#10
;" name="image"/>
    <meta
      content="The ecmascript used was found to renderable
only by Cortona plug-in.&#10;Care should be taken when using
CreateVrmlFromString node.&#10;\ before quotations in the
string, and must be in accurate vrml form.&#10;"
name="bugs"/>
    <meta
      content="X3D-Edit,
http://www.web3d.org/x3d/content/README.X3D-Edit.html"
name="generator"/>
    <meta content="../../license.html" name="license"/>
  </head>
  <Scene><!--png image files for the cloud textures must be
designated
      in the ecmascript node. Cloud sizes and shapes are
designated in
      the ecmascript node. Scaling, translations, and size
of geometry
      controls sizes and shapes.--><Viewpoint
description="Main"
      jump="false" orientation="0 1 0 1.57" position="50000
1000 42000"/>
    <Viewpoint description="Light House Tower" jump="false"
      orientation="0 1 0 1.3" position="45000 1290 44000"/>
    <Viewpoint description="centerWest" jump="false"
      orientation="0 1 0 2.5" position="48000 1000 20000"/>
    <Background groundColor="0 0 1" skyColor="0 0 1"/>
    <DirectionalLight ambientIntensity="1" color="1 1 1"
      direction="-1 0 0" global="true" intensity="1"
on="true"/>
    <Group DEF="Terrain">
      <Transform scale="50 50 50" translation="25000 0
25000">
```

```
        <Inline
url="&quot;MontereyBayLargeMesh.wrl&quot;&#10;&quot;../../Lo
cations/MontereyBayCalifornia/MontereyBayLargeMesh.x3d&quot;
&#10;&quot;https://savage.nps.edu/Savage/Locations/MontereyB
ayCalifornia/MontereyBayLargeMesh.x3d&quot;&#10;&quot;../../
Locations/MontereyBayCalifornia/MontereyBayLargeMesh.wrl&quo
t;&#10;&quot;https://savage.nps.edu/Savage/Locations/Montere
yBayCalifornia/MontereyBayLargeMesh.wrl&quot;"/>
      </Transform>
      <Transform rotation="1 0 0 1.57" translation="25000 0
25000">
        <Shape>
          <Rectangle2D size="77000 55000" solid="false"/>
          <Appearance>
            <ImageTexture
url="&quot;ocean.png&quot;&#10;&quot;https://savage.nps.edu/
Savage/Environment/Meteorology/ocean.png&quot;"/>
          </Appearance>
        </Shape>
      </Transform>
    </Group>
    <Group DEF="Placemarks">
      <Transform scale="50 50 50" translation="45000 30
44000">
        <Inline
url="&quot;Lighthouse.wrl&quot;&#10;&quot;../../Locations/Mo
ntereyBayCalifornia/Lighthouse.wrl&quot;&#10;&quot;https://s
avage.nps.edu/Savage/Locations/ShipIslandMississippi/LightHo
use.wrl&quot;&#10;&quot;../../Locations/MontereyBayCaliforni
a/LightHouse.x3d&quot;&#10;&quot;https://savage.nps.edu/Sava
ge/Locations/MontereyBayCalifornia/LightHouse.x3d&quot;&#10;
"/>
      </Transform>
    </Group>
    <Group DEF="Clouds">
      <Transform DEF="Cumulus"/>
      <Transform DEF="Cirrus"/>
      <Transform DEF="Fog"/>
      <Transform DEF="Stratus"/>
      <Script DEF="PixelScript" directOutput="true">
        <field  accessType="initializeOnly"  name="Cumulus"
type="SFNode">
          <Transform USE="Cumulus"/>
        </field>
        <field  accessType="initializeOnly"  name="Cirrus"
type="SFNode">
          <Transform USE="Cirrus"/>
        </field>
        <field   accessType="initializeOnly"   name="Fog"
type="SFNode">
          <Transform USE="Fog"/>
        </field>
        <field  accessType="initializeOnly"  name="Stratus"
type="SFNode">
          <Transform USE="Stratus"/>
</field><![CDATA[ecmascript:


function cumulustranslation() // These values designate the
boundary location of the cloud
{
     X = 50000*Math.random();             //  X horizontal
range
     Y = 1000 + 300*Math.random();  //   Y vertical base +
range
     Z = 50000*Math.random();         // z horizontal range


     randomt = new String(X+' '+Y+' '+Z);
     return randomt;
}
```

```
function cumulusscale() // these values scale a cloud within
a designated size
{
     maxscale = 1;
     scale = Math.round(9+maxscale*Math.random());
     X = 1.5*scale;
     Y = scale;
     Z = scale;

     randomscale = new String(X+' '+Y+' '+Z);
     return randomscale;
}



function cumulussectiontranslation() // These random values
place another portion of cumulus type cloud
{
     randomtheta = 6.28319*Math.random();
     randomphi = .7854*Math.random();
     randomradius = 90 + 5*Math.random();//the first whole
number should be close to the sectionradius
     X                                                        =
randomradius*Math.cos(randomtheta)*Math.sin(randomphi);
     Z                                                        =
randomradius*Math.sin(randomtheta)*Math.sin(randomphi);
     Y = randomradius*Math.cos(randomphi);

     randomt = new String(X+' '+Y+' '+Z);
     return randomt;
}

function cirrustranslation()
{
     X = 50000*Math.random();
     Y = 8000 + 1000*Math.random();
     Z = 50000*Math.random();

     randomt = new String(X+' '+Y+' '+Z);
     return randomt;
}

function cirrusscale()
{
     maxscale = 1;
     scale = Math.round(9+maxscale*Math.random());
     X = 1.5*scale;
     Y = 1+0.5*Math.random();
     Z = 1.5*scale;

     randomscale = new String(X+' '+Y+' '+Z);
     return randomscale;
}

function cirrussectiontranslation()
{
     randomtheta = 6.28319*Math.random();
     randomphi = .7854*Math.random();
     randomradius = 90 + 5*Math.random();

     X                                                   =
randomradius*Math.cos(randomtheta)*Math.sin(randomphi);
     Z                                                   =
randomradius*Math.sin(randomtheta)*Math.sin(randomphi);
     Y = randomradius*Math.cos(randomphi);

     randomt = new String(X+' '+Y+' '+Z);
     return randomt;
}

function fogtranslation()
{
     X = 40000+5000*Math.random();
     Y = 300*Math.random();
     Z = 40000+5000*Math.random();

     randomt = new String(X+' '+Y+' '+Z);
     return randomt;
}

function fogscale()
{
     X = 7;
     Y = 7;
     Z = 7;

     randomscale = new String(X+' '+Y+' '+Z);
     return randomscale;
}

function fogsectiontranslation()
{
     randomdistance = 950 + 100*Math.random();

     X = randomdistance;
     Z = randomdistance;
     Y = 100*Math.random();

     randomt = new String(X+' '+Y+' '+Z);
     return randomt;
}

function stratustranslation()
{
     X = 50000*Math.random();
     Y = 2000 + 1000*Math.random();
     Z = 50000*Math.random();

     randomt = new String(X+' '+Y+' '+Z);
     return randomt;
}

function stratusscale()
{
     maxscale = 10;
     scale = Math.round(30+maxscale*Math.random());
     X = scale;
     Y = 2+0.5*Math.random();
     Z = scale;

     randomscale = new String(X+' '+Y+' '+Z);
     return randomscale;
}

function stratussectiontranslation()
{
     randomtheta = 6.28319*Math.random();
     randomphi = .7854*Math.random();
     randomradius = 90 + 5*Math.random();
     X                                                         =
randomradius*Math.cos(randomtheta)*Math.sin(randomphi);
     Z                                                         =
randomradius*Math.sin(randomtheta)*Math.sin(randomphi);
     Y = randomradius*Math.cos(randomphi);

     randomt = new String(X+' '+Y+' '+Z);
     return randomt;
}

function rotation()
{
     radians = 6.28*Math.random();
     randomr = new String('0 0 1 ' + radians );
     return randomr;
}

function cumulus()
{
maxi = 20;  // number of clouds
maxj = 5; // denotes how many portions affecting the size of
the cloud
maxk = 8;  // number of billboards indicating cloud density
sectionradius = 100;  //radius of individual cloud sections

for (var i=0; i < maxi; i++)
{

CloudStringA = '    Transform {         \n' +
'    scale '+ cumulusscale() + '                  \n' +
'    translation '+ cumulustranslation() + '    \n' +    //
cloud placement
'    children [                                \n';
```

```
CloudStringB = new Array();
CloudStringF = new Array();

     for (var j=0; j < maxj; j++)
     {

     radius = 0;

     CloudStringB[j]= '  Transform {                  \n' +
     '      translation '+ cumulussectiontranslation() + '
\n' +     // section placement
     '    children [                              \n';

     CloudStringC = new Array();
     image = new String();

               for (var k=1; k < maxk; k++)   // maxk value
denotes how many textured billboards make up the cloud
               {

          randomtheta = 6.28319*Math.random();
          randomphi = 1.5708*Math.random();
          radius  =  radius+(sectionradius/maxk);  //  radius
incremental steps based on billow radius and max billboards

          X                                                  =
radius*Math.cos(randomtheta)*Math.sin(randomphi);
          Z                                                  =
radius*Math.sin(randomtheta)*Math.sin(randomphi);
          Y = radius*Math.cos(randomphi);

          if (Y <= 30) //cloud shading and lighting control
     {
```

```
     image         =         '        \"CloudTexture1_5.png\"
\"https://savage.nps.edu/Savage/Environment//Meteorology/Clo
udTexture1_5.png\" \n';
     }


          else
     {
     image         =         '        \"CloudTexture1_4.png\"
\"https://savage.nps.edu/Savage/Environment//Meteorology/Clo
udTexture1_4.png\" \n';
     }


          Billboardtranslation = new String(X+' '+Y+' '+Z);


          CloudStringC[k] = ' Transform {             \n' +
          'translation '+ Billboardtranslation + '     \n' +
// random billboard placement within radius designated above
          '      children [                            \n' +
          '          Billboard {                      \n' +
          '            axisOfRotation 0 0 0           \n'  +
// 0 0 0 designates rotation on all axis
          '      children [                            \n' +
          '       Transform {                          \n' +
          '          rotation  0 0 0 0               \n'   +
// a rotation of the individual billboards can be defined
          '   children [                         \n' +
          '        Shape {                            \n' +
          '          appearance Appearance {         \n' +
          '                   material Material {      \n' +
          '                                  }        \n' +
          '           texture ImageTexture {         \n' +
          '            url [ ' + image + ' ]           \n' +
          '       }                                 \n' +
          '       }                                 \n' +
          '        geometry IndexedFaceSet {          \n'   +
// define type of geometry to texture
          '         coordIndex [ 0, 1, 2, 3 ]         \n' +
          '         solid FALSE                  \n' +
          '               coord Coordinate {         \n' +
          '         point [ 50 50 0,             \n' +     //
define size of the geometry. Here 100 meter 2D square.
          '                        50 -50 0,          \n' +
          '                      -50 -50 0,           \n' +
          '                        -50 50 0 ]         \n' +
          '                          }                 \n' +
          '                        }                    \n' +
          '                      }                     \n' +
          '                  ]                         \n' +
          '                }                           \n' +
          '           ]                             \n' +
          '       }                                   \n' +
          '      ]                                    \n' +
          '     }                                         \n';

          }

     CloudStringD = CloudStringC.join(' ');

     CloudStringE = '   ]                 \n' +
     '    }                              \n';

     CloudStringF[j]   =   CloudStringB[j]   +   CloudStringD
+CloudStringE;

     }

CloudStringG = CloudStringF.join(' ');
```

```
CloudStringH = '      ]                               \n' +
'     }                                                \n' +
'#####################################################  \n';

CloudString = CloudStringA + CloudStringG + CloudStringH;

newNode = Browser.createVrmlFromString(CloudString);
Cumulus.children[i] = newNode[0];

   }
}

function cirrus()

{

maxi = 2;
maxj = 5;
maxk = 8;
sectionradius = 1000;

for (var i=0; i < maxi; i++)
{

CloudStringA = '    Transform {          \n' +
'    scale '+ cirrusscale() + '                    \n' +
'    translation '+ cirrustranslation() + '      \n' +
'    children [                                \n';

CloudStringB = new Array();
CloudStringF = new Array();
```

```
for (var j=0; j < maxj; j++)
{
radius = 0;

CloudStringB[j]= '  Transform {                  \n' +
'translation '+ cirrussectiontranslation() + '    \n' +
'    children [                                \n';

CloudStringC = new Array();

          for (var k=1; k < maxk; k++)
          {

     randomtheta = 6.28319*Math.random();
     randomphi = 1.5708*Math.random();
     radius = radius+(sectionradius/maxk);

     X                                                 =
radius*Math.cos(randomtheta)*Math.sin(randomphi);
     Z                                                 =
radius*Math.sin(randomtheta)*Math.sin(randomphi);
     Y = radius*Math.cos(randomphi);

     Billboardtranslation = new String(X+' '+Y+' '+Z);

     CloudStringC[k] = ' Transform {               \n' +
     'translation '+ Billboardtranslation   + '    \n' +
     '      children [                             \n' +
     '          Billboard {                  \n' +
     '            axisOfRotation 0 0 0           \n' +
     '            children [                     \n' +
     '       Transform {                           \n' +
          '      rotation '  + rotation() + '        \n' +
          '         children [                        \n' +
          '           Shape {                         \n' +
          '          appearance Appearance {         \n' +
          '                 material Material {        \n' +
          '                  }                         \n' +
          '            texture ImageTexture {        \n' +
          '                            url 
[\"cloudtexture3.png\"
\"https://savage.nps.edu/Savage/Environment/Meteorology/clou
dtexture1_4.png\" ] \n' +
          '        }                                   \n' +
          '        }                                  \n' +
          '        geometry IndexedFaceSet {          \n' +
          '         coordIndex [ 0, 1, 2, 3 ]         \n' +
          '                    solid FALSE            \n' +
          '              coord Coordinate {           \n' +
          '             point [ 500 500 0,            \n' +
          '                      500 -500 0,          \n' +
          '                     -500 -500 0,          \n' +
          '                     -500 500 0 ]          \n' +
          '                          }                 \n' +
          '                        }                  \n' +
          '                      }                    \n' +
          '                  ]                         \n' +
          '                }                          \n' +
          '           ]                               \n' +
          '       }                                   \n' +
          '      ]                                     \n' +
          '     }                                   \n';


          }
```

```
     CloudStringD = CloudStringC.join(' ');

     CloudStringE = '   ]                 \n' +
     '    }                             \n';

     CloudStringF[j]   =   CloudStringB[j]   +   CloudStringD
+CloudStringE;

     }

CloudStringG = CloudStringF.join(' ');

CloudStringH = '      ]                              \n' +
'     }                                            \n' +
'#########################################################
\n';

CloudString = CloudStringA + CloudStringG + CloudStringH;

newNode = Browser.createVrmlFromString(CloudString);
Cirrus.children[i] = newNode[0];
  }
}

function fog()
{

maxi = 2;
maxj = 5;
maxk = 8;
sectionlength = 1000;
```

```
for (var i=0; i < maxi; i++)
{

CloudStringA = '    Transform {         \n' +
'    scale '+ fogscale() + '                \n' +
'    translation '+ fogtranslation() + '        \n' +
'    children [                                \n';

CloudStringB = new Array();
CloudStringF = new Array();

     for (var j=0; j < maxj; j++)
     {

     length = 0;

     CloudStringB[j]= '  Transform {                  \n' +
     '    translation '+ fogsectiontranslation() + '  \n' +
     '    children [                              \n';

     CloudStringC = new Array();
     image = new String();

               for (var k=1; k < maxk; k++)
               {

          length = length+(sectionlength/maxk);

          X = length;
          Z = 100*Math.random();
          Y = 100*Math.random();
     image          =          '          \"CloudTexture1_4.png\"
\"https://savage.nps.edu/Savage/Environment/Meteorology/Clou
dTexture1_4.png\" \n';

          Billboardtranslation = new String(X+' '+Y+' '+Z);

          CloudStringC[k] = ' Transform {             \n' +
          ' translation '+ Billboardtranslation + '    \n' +
          '      children [                           \n' +
          '          Billboard {                       \n' +
          '       axisOfRotation 0 0 0                 \n' +
          '    children [                              \n' +
          '        Transform {                         \n' +
          '        rotation  0 0 0 0                 \n' +
          '         children [                        \n' +
          '           Shape {                         \n' +
          '          appearance Appearance {         \n' +
          '                   material Material {      \n' +
          '                                   }        \n' +
          '           texture ImageTexture {         \n' +
          '           url [ ' + image + ' ]           \n' +
          '       }                                \n' +
          '       }                                \n' +
          '         geometry IndexedFaceSet {        \n' +
          '          coordIndex [ 0, 1, 2, 3 ]         \n' +
          '                    solid FALSE            \n' +
          '           coord Coordinate {         \n' +
          '              point [ 500 500 0,          \n' +
          '                      500 -500 0,         \n' +
          '                     -500 -500 0,         \n' +
          '                     -500 500 0 ]         \n' +
          '                         }                 \n' +
          '                       }                    \n' +
          '                       }                     \n' +
          '                   ]                          \n' +
          '                 }                            \n' +
          '            ]                              \n' +
          '       }                                    \n' +
          '      ]                                      \n' +
          '     }                                       \n';

          }

     CloudStringD = CloudStringC.join(' ');



     CloudStringE = '   ]                 \n' +
     '    }                            \n';

     CloudStringF[j]   =   CloudStringB[j]   +   CloudStringD
+CloudStringE;

     }

CloudStringG = CloudStringF.join(' ');

CloudStringH = '      ]                               \n' +
'     }                                      \n' +
'#########################################################
\n';

CloudString = CloudStringA + CloudStringG + CloudStringH;

newNode = Browser.createVrmlFromString(CloudString);
Fog.children[i] = newNode[0];
   }
```

```
}

function stratus()
{

maxi = 1;
maxj = 10;
maxk = 10;
sectionlength = 1000;

for (var i=0; i < maxi; i++)
{

CloudStringA = '    Transform {          \n' +
'    scale '+ stratusscale() + '                   \n' +
'    translation '+ stratustranslation() + '    \n' +
'    children [                              \n';

CloudStringB = new Array();
CloudStringF = new Array();

     for (var j=0; j < maxj; j++)
     {

     length = 0;

     CloudStringB[j]= '  Transform {                   \n' +
     ' translation '+ stratussectiontranslation() + '  \n' +
     '    children [                             \n';

     CloudStringC = new Array();
     image = new String();
```

```
          for (var k=1; k < maxk; k++)
          {

     length = length+(sectionlength/maxk);

     X = length;
     Z = 1000*Math.random();
     Y = 1000*Math.random();

image         =         '         \"CloudTexture1_4.png\"
\"https://savage.nps.edu/Savage/Environment/Meteorology/Clou
dTexture1_4.png\" \n';

     Billboardtranslation = new String(X+' '+Y+' '+Z);

     CloudStringC[k] = ' Transform {              \n' +
     '  translation '+ Billboardtranslation   + ' \n' +
     '      children [                            \n' +
     '          Billboard {                      \n' +
     '            axisOfRotation 0 0 0            \n' +
     '            children [                      \n' +
     '       Transform {                          \n' +
     '        rotation  0 0 0 0                 \n' +
     '        children [                         \n' +
     '         Shape {                           \n' +
     '           appearance Appearance {        \n' +
     '                   material Material {      \n' +
     '                                   }        \n' +
     '            texture ImageTexture {        \n' +
     '            url [ ' + image + ' ]           \n' +
     '        }                                 \n' +
     '            }                              \n' +
     '          geometry IndexedFaceSet {        \n' +
     '         coordIndex [ 0, 1, 2, 3 ]         \n' +
     '                     solid FALSE            \n' +
     '                 coord Coordinate {        \n' +
     '                      point [ 500 500 0, \n' +
     '                             500 -500 0,  \n' +
     '                           -500 -500 0,     \n' +
     '                            -500 500 0 ]   \n' +
     '                           }                  \n' +
     '                         }                  \n' +
     '                       }                    \n' +
     '                   ]                        \n' +
     '                 }                          \n' +
     '            ]                             \n' +
     '        }                                   \n' +
     '       ]                                    \n' +
     '      }                                      \n';

     }

CloudStringD = CloudStringC.join(' ');

CloudStringE = '   ]                \n' +
'    }                            \n';

CloudStringF[j]   =   CloudStringB[j]   +   CloudStringD
+CloudStringE;

}

CloudStringG = CloudStringF.join(' ');
```

```
CloudStringH = '      ]                                  \n' +
'     }                                             \n' +
'#######################################################
\n';

CloudString = CloudStringA + CloudStringG + CloudStringH;

newNode = Browser.createVrmlFromString(CloudString);
Stratus.children[i] = newNode[0];
   }
}

function initialize()
{

cumulus();
cirrus();
fog();
stratus();

}]]></Script>
      <DirectionalLight ambientIntensity="1" color="1 1 1"
        direction="-1  -1  0"  global="true"  intensity="1"
on="true"/>
    </Group>
  </Scene>
</X3D>
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. MODEL AVAILABILITY

The X3D files with included scripts and png image files are available for download and viewing online at https://savage.nps.edu/Savage/Environment/Oceanography/index.html. The Monterey elevation and lighthouse files are also available in the Savage archives.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Bouthors, A., Neyret, 2004: Modeling Cloud Shapes. Eurographics Digital Library, 5 pp. [Available online at http://www.eg.org/EG/DL/Conf/EG2004/short/short54.pdf]. Last accessed Feb 08.

Brutzman, D. X3D-Edit Authoring Tool for Extensible 3D (X3D) Graphics, 7 pp. [Available online at http://www.web3d.org/x3d/content/X3D-EditAuthoringTool.pdf]. Last accessed Mar 08.

Brutzman, D., Daly, L., X3D: Extensible 3D Graphics for Web Authors. Morgan Kaufman, 441 pp.

GNU Image Manipulation Program [Available online at http://www.gimp.org]. Last accessed Mar 08.

Harris, M.J., 2003: Real-Time Cloud Simulation and Rendering. PhD dissertation. Department of Computer Science, University of North Carolina at Chapel Hill, 173 pp.

Hibbard, W., Santek, D., Interactivity is the Key. Proceedings of the 1989 Chapel Hill Workshop on Volume Visualization, 39-43.

Meibner, M., Huang, J., Bartz, D. Mueller, K., Crawfis, R., A Practical Evaluation of Popular Volume Rendering Algorithms. October 2000, IEEE Symposium on Volume Visualization 2000, 81-89.

Plonski, M., 1998: Cloud Depiction and Forecast System II Overview, 1998 IEEE Aerospace Conference 7 pp. [Available at http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel4/5608/15054/00685807.pdf?tp=&isnumber=&arnumber=685807.]. Last accessed Feb 08.

Portable Network Graphics Home Site [Available online at http://www.libpng.org/pub/png]. Last accessed Mar 08.

Schpok, J., Simons, J., Ebert, D.S., Hansen, C., 2003: A Real-Time Cloud Modeling, Rendering and Animation System. SIGRAPH Symposium on Computer Animation. 8 pp. [Available online at www.ecn.purdue.edu/purpl/level2/papers/ scaclouds.pdf]. Last accessed Feb 08.

Tarantilis, G.E., 2004: Simulating Clouds with Procedural Texturing Techniques Using the GPU. M.S. Thesis. MOVES Institute, Naval Postgraduate School, 69 pp.

Taylor, M.G. Evaluation of a Cloud Fraction Analysis Product. M.S. Thesis. Dept. of Meteorology, Air Force Institute of Technology, 93 pp.

Treinish, L., 1999: Task-Specific Visualization Design. IEEE Computer Graphics and Applications. October 1999, 72-77.

Wang, N., Realistic and Fast Cloud Modeling. 17 pp. [Available online at http://ofb.net/~niniane/clouds-jgt.pdf]. Last accessed Jan 08.

INITIAL DISTRIBUTION LIST