AFIT/GCS/ENG/93M-04

RENDERING THE OUT-THE-WINDOW VIEW FOR THE AFIT VIRTUAL COCKPIT

THESIS

W. Dean McCarty

AFIT/GCS/ENG/93M-04

Approved for public release; distribution unlimited

AFIT/GCS/ENG/93M-04

RENDERING THE OUT-THE-WINDOW VIEW FOR THE AFIT VIRTUAL COCKPIT

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science (Computer Systems)

W. Dean McCarty, B.S.

March, 1993

Approved for public release; distribution unlimited

Acknowledgments

I would like to express my gratitude to everyone who helped me with this thesis effort. Sincere thanks go to my thesis advisor, Lt Col Phil Amburn, for his patience and for his understanding of computer graphics and people. Lt Col Amburn always had time to share the joy and the excitement of the most trivial successes made in this project. I would also like to thank the 92D graphics students, Capts Rex Haddix, Bruce Hobbs, David Pond, Richard Remington, and David Tisdale, for pulling cables, finding mistakes, and supporting each other. The other two-thirds of the Virtual Cockpit project, Capt Steven Sheasby and Capt John Switzer, also deserve a special thank you for their determination to make the project fly.

I would also like to thank David Neyland and the people at DARPA for sponsoring the Virtual Cockpit. This project would not have been possible without their support.

Finally and with my deepest gratitude, I want to thank my family. Thanks to my wife, Carol, for her support and for her strength to keep the family on course while I was absorbed in virtual worlds. And my children, Sarah, Jared, Joshua, Anna, and Rachel, deserve a thank you for their youthful comments which always reminded me of my priorities and my eternal responsibilities.

W Dean McCarty

Table of Contents

Page

List of Figures

List of Tables

AFIT/GCS/ENG/93M-04

Abstract

The Air Force Institute of Technology (AFIT) is developing a distributed interactive flight simulator, the Virtual Cockpit, using commercial graphics workstations and helmet mounted displays. The Virtual Cockpit communicates with other simulators via local and long-haul networks using the SIMNET protocol.

The work reported in this thesis focuses on developing the terrain database for the synthetic environment and on rendering the pilot's view of the database.

There are many different file formats for describing 3-dimensional geometric polygonal objects. An analysis of three formats, AFIT GEOM, the Naval Postgraduate School (NPS) DRP, and Software Systems' Flight, is presented. Each file format is described, and their attributes are compared in a decision table.

A series of C++ classes were developed to render the file format used by the terrain database. These C++ classes, which include classes for stationary and moving entities, and for textures, are discussed. A technique for increasing the rendering speed is also presented.

RENDERING THE OUT-THE-WINDOW VIEW FOR THE AFIT VIRTUAL COCKPIT

I. Introduction

1.1 Overview

Flight simulators are an integral part of military flight training. Although instructors can not replace all flying time with simulator time, simulators do have many advantages. For example, flight simulators "permit close observation of pilot performance, they provide immediate feedback of information, and they permit a pilot to experience malfunctions and flight conditions rarely encountered in actual flight" (8). Additionally, increasing aircraft fuel costs and decreasing budgets will *force* air crews to train more in simulators and less in the aircraft.

Along with this increasing interest in flight simulators, the Defense Advanced Research Projects Agency (DARPA) gave the Air Force Institute of Technology (AFIT) Graphics Laboratory a grant to build a flight simulator based on commercial off-the-shelf computer graphics workstations and helmet mounted display systems. The simulator would be connected to local and long haul networks enabling it to interact with other simulators on the DARPA SIMNET network. The goal was to have version one of the AFIT Virtual Cockpit participate in DARPA's Zealous pursuit exercises in December 1992.

Zealous Pursuit is the first phase in development of DARPA's WAR BREAKER simulation environment. The WAR BREAKER program is to develop and demonstrate "capabilities enabling an integrated, end-to-end system that detects, identifies, targets, and neutralizes time-critical targets". The simulation environment will be one of the system engineering tools used to analyze WAR BREAKER system requirements (13).

1.2 Problem Statement

The purpose of the AFIT Virtual Cockpit was to prototype a flight simulator using a commercial off-the-shelf graphics workstation that costs under $250,000. This prototype would demonstrate the ability to fly through a synthetic environment using a believable flight dynamics model, while interacting with other vehicle simulators appearing in the environment.

My task was to develop the database for the synthetic environment and to render the pilot's view of the environment. The database required models of the terrain elevation and of the cultural features, i.e., vehicles, buildings, bridges, etc. The images rendered from the resulting database would provide all of the feedback information to the pilot.

The Virtual Cockpit project included two additional research areas. Capt. John Switzer developed the aircraft dynamics and instrumentation model (12), and Capt. Steven Sheasby developed the network software allowing us to communicate with other simulators operating in the same synthetic environment(10).

1.3 Objectives

In approaching this project, there were three main objectives. First, the environment had to contain both stationary and moving objects. Second, the ability to texture map polygons was needed to increase the visual fidelity of the rendered images. And finally, the image frame rate needed to be high enough to prevent a sluggish response to the pilot's input.

1.3.1 Static and Dynamic Models The database needed to contain both static geometric models and dynamic geometric models. The static models are the houses, trees, and other stationary objects that give the database visual complexity. This complexity makes the environment appear more realistic. The models also give the pilot visual clues about his altitude and speed. On the other hand, the dynamic models are the aircraft and ground vehicles that are moving around in the environment. For us, other simulators connected to the network would be controlling the dynamic models in our world.

Obviously, the static models could be positioned in the environment prior to running the simulator and that location would not change. However, the Virtual Cockpit would need the ability to change both the position and the orientation of the dynamic models during runtime. The network messages from the other simulators would provide the position and the orientation in the world database.

1.3.2 Textures The Graphics Lab's new workstations have the ability to render texture maps in realtime. The second objective was to incorporate this realtime texturing into the rendering of the database. The texture maps would increase the visual fidelity and complexity of the database without increasing the polygon count of the models.

1.3.3 Frame Rate The last objective was to render at least ten to fifteen images (frames) per second. A slow frame rate, one to two frames per second, would give the Virtual Cockpit a sluggish response. Thus, the pilot would overreact to the images and lose control of the aircraft.

Even though a satisfactory frame rate is critical to successfully fly the Virtual Cockpit, I decided to analyze the frame rate last. My approach was to analyze the geometry file formats and to implement the rendering software first. Then, I would optimize the software to improve the frame rate.

1.4 Constraints

There were three major constraints that affected the development of this project.

The first constraint was the hardware available in the Graphics Lab. Since the lab is equipped with Silicon Graphics workstations, all software development and optimization would be done on those workstations. In 1992, the Graphics Lab purchased Silicon Graphics 4D/VGX and VGXT workstations that have the ability to texture map in realtime.

Secondly, the languages available for developing the Virtual Cockpit were C and C++. We decided to use C++ in order to take advantage of the reusability of the object oriented C++ classes. Also, previous students had created a base of many C++ classes which could be used in developing the Virtual Cockpit.

Finally, the greatest constraint was time. Money, which is often a constraint, was available for new hardware and software through a DARPA grant. However, the acquisition process takes time and patience; neither of which are overly abundant in students. Many times, the development process was delayed or inhibited because of the hardware and software delivery date changes and postponements.

1.5 Thesis Overview

This thesis consists of five chapters. The first chapter includes a discussion of the problem statement, the research objectives, the constraints on the research, and on overview of this document. Chapter II contains background information on the PHIGS geometry format, techniques for achieving realistic images at interactive processing speeds, and the use of graphics workstations in flight simulators. Chapter III analyzes three geometry file formats and the method used to select one of the formats. Chapter IV discusses the C++ classes developed for the geometry format, and the extensions made to the format. The final chapter discusses the results and conclusions of the project, and recommendations for further research.

II. Background

2.1 Overview

This chapter lays the groundwork for the remainder of this thesis. The Programmer's Hierarchical Interactive Graphics System (PHIGS) for specifying graphics data is described in Section 2.2. The next section looks at two techniques, textures and levels of detail, that can be used in interactive simulators to increase the visual realism. Section 2.4 describes a flight simulator developed by the Naval Postgraduate School (NPS) that runs on a Silicon Graphics graphics workstation.

2.2 PHIGS

PHIGS is one of several graphics specifications standards. PHIGS is "an interface between an application program and a graphics system that controls graphics devices" (11).

The basic building blocks of PHIGS graphics data are called *structures.* Structures may contain graphics primitives, attributes, viewing selections, modeling transformations, and references to other structures. Since structures may be nested, it is possible to create complex hierarchical models from simple structures. Each structure is defined in it's own modeling coordinate system. Parent structures may apply transformations of scaling, rotation, and translation to child structures.

The PHIGS database is dynamic. Any element of a structure may be edited while executing the application. Elements may even be added or deleted. This provides extensive flexibility at runtime. For example, changing the transformation matrices will move a child structure relative to the parent structure.

PHIGS also binds attributes to primitives when the structure is traversed at rendering time instead of when the structure is created. This allows child structures to inherit attributes from parent structures. Since attributes may be changed while executing the application, traversal-time binding is a necessity. Otherwise, the new attributes would not take effect since the old attributes were bound to the primitive when it was created.

These characteristics allow PHIGS to effectively model 3-dimensional objects and to manipulate those objects in world coordinates. PHIGS, therefore, enables the computer graphics user to move from the 2-dimensional realm of only manipulating the 2D graphics image into the 3-dimensional world of virtual environments and manipulating objects within those environments (4).

2.3 Interactive Techniques for Increased Realism

As flight simulator images become more and more realistic, pilots feel more like they are actually flying and less like they are playing a game. This section discusses some of the techniques available for creating realistic images. However, some techniques that create photo realistic images, such as, ray tracing and radiosity, require more time than is feasible in an interactive application such as a flight simulator. Therefore, the scope of this section concentrates on two techniques, texture mapping and levels of detail, which can improve the image realism in interactive applications.

2.3.1 Textures Early computer generated images were too simple and plain. For an image to look realistic, it must have the appearance of complexity. "Texture mapping is a relatively efficient way to create the appearance of complexity without the tedium of modeling and rendering every 3D detail of a surface" (5). Modeling every detail of a surface would also require more polygons in the model. The extra polygons would increase the rendering time and, thus, decrease the frame rate. Of course, rendering textured polygons takes longer than rendering Gouraud shaded polygons. However, the increased realism is well worth the extra time spent in processing.

In addition to a more realistic image, texture maps increase the visual cues pilots need while flying in flight simulators (2). These visual cues help to determine the altitude and speed without constant watching of the instruments. Therefore, the flight simulator would more closely simulate an actual flight, and the effectiveness of the training would increase.

2.3.2 Levels of Detail Image rendering time is directly related to the number of polygons being rendered. Therefore, as the polygon count decreases, the frame rate will increase. This is the fundamental concept behind having multiple levels of detail for the geometric models. As a viewer moves away from an object, the viewer can distinguish less detail about that object. Therefore, at that distance, the database can use a simpler model with fewer polygons to represent the object.

Three things to consider about levels of detail are: how many levels of detail are necessary, how much simpler should the next model be, and what is the switching distance, the distance from the eye to the model, between levels of detail. A common approach has been to have three levels of detail, high, medium, and low. However, the difficulty is trying to minimize the visual discrepancies between levels of detail by varying the number of resolution levels and the transition distances. Michael A. Cosman has proposed as many as eight levels of detail for representing terrain. He suggests that as the distance doubles, the polygon count should decrease by half (3). This will give more levels of detail, but the change between each level will be less noticeable.

On the other hand, for models of objects, Roy Latham uses the rule of thumb that the polygon count from one level of detail to the next should differ by an order of magnitude (7). The use of multiple levels of detail is still subject to personal preference and artistic license.

Choosing the correct transition distance is another tradeoff. This time the tradeoff is between visual discrepancy and polygon count. The switching distance should be great enough to minimize the visual distraction of the model changing, but it should also be small enough to help reduce the overall polygon count and to increase the frame rate.

To alleviate the effect of one model "popping" or suddenly switching to the next level of detail, we can blend the two models using transparency. Mr. Latham suggests completely fading in one model before fading out the other model. This will prevent the possibility of a light, that is behind the object, becoming visible when both of the transitioning models are partially transparent.

Of course, the switching distance triggers the beginning of a transition between levels of detail. However, Mr. Latham says once the transition starts then it needs to complete in a given amount of time. If the completion of the transition is also triggered by distance, then it is possible to circle around an object, such as a control tower, at the switching distance, and the object will stay a conglomerate of the two transitioning models. However, if the transition completes in a given amount of time, then both levels of detail will be visible only for that short amount of time (7).

2.4 Flight Simulators on Graphics Workstations

Most commercial flight simulators used by the military and by airlines for training are large and expensive. The simulators require special facilities for the projection systems and multiple computer image generators (CIGs). The CIGs use expensive proprietary hardware to generate multiple channels for display. Due to the expense of these commercial flight simulators, only large organizations can afford to own them. However, with the cost of graphics workstations dropping, the possibilities exist for developing low-cost flight simulators, where the hardware costs less than $250,000.

NPS has demonstrated this capability. They developed a prototype flight simulation of an Army weapon system called the Fiber Optically Guided Missile (14). The initial terrain database is a 35 x 36 kilometer area of Fort Hunter Liggett, CA.

However, the terrain database for the missile flight is a 10 kilometer square subset of the total terrain. The missile launch site and the target site determine the area extracted from the terrain database.

Once the terrain is extracted, launch occurs, and the missile flight simulation starts. From this point until missile impact, the monitor displays the view from the missile's onboard "camera". The operator may pan, tilt, and zoom the camera view. He may also input navigation corrections to ensure hitting the target.

The Naval Postgraduate School implemented this prototype on the Silicon Graphics Iris 3120 graphics workstation. The missile flight simulation can render between 1500 and 2000 polygons per frame at a rate of three to four frames per second. Since this was published in 1988, Silicon Graphics has increased the polygon throughput of their internal graphics engine, and they have added real-time texture capability. These improvements to the graphics workstation would increase the frame rate and improve the visual realism of the simulation.

2.5 Summary

The information in this chapter gave a baseline for comparison and analysis of geometry formats and flight simulator requirements. The discussion on PHIGS, texture maps, and levels of detail described features to look for when analyzing and choosing a format for our geometric models. The requirements for the AFIT flight simulator were based on the accomplishments of the NPS low-cost simulators.

III. Geometry File Formats

3.1 Overview

This chapter gives a comparison of three different file formats for describing three-dimensional polygonal objects. The AFIT GEOM format, developed by and used at AFIT, is described in Section 3.2. The next section gives a description of the Naval Postgraduate School's DRP format which is used to describe terrain databases. Section 3.4 describes the Flight file format, version 11, used by Software Systems in their commercial modeler, MultiGen. A comparison of these three geometry file formats is in Section 3.5.

3.2 AFIT GEOM Format

The AFIT GEOM format is the geometry file format currently used at AFIT. The object description is stored in ASCII files, and the file format was designed to be simple but also flexible. Since GEOM files are ASCII files, any text editor may be used to create and to edit the files. Along with text editors, tools are available which revolve, rotate, scale, translate, and combine GEOM files. By using these tools, simple objects may be modified and combined into more complex objects and written to a single GEOM file.

3.2.1 GEOM File Description The geometry information in a GEOM file is position dependent. The vertices (points) are grouped together and sequentially numbered. The vertices are then referred to by number in the description of an individual polygon. Polygons (pa'ches), attributes, and textures are also listed together in the file and are referred to by their position number.

Figure 1 is an example of a simple GEOM file. The first line is a comment and is ignored by all rendering software. The second line says vertices are listed counter-clockwise in polygon descriptions. It also tells the rendering software to cull

```
Five sides of a cube with some transparency and some texture maps
ccw purge
points 20 patches 5 attributes 4 textures 3
/* bottom face of cube will have checkerboard on it */
0.  0.  0.  normal 0.  0.  1.  tindex 0.  0.
1.  0.  0.  normal 0.  0.  1.  tindex 1.  0.
1.  1.  0.  normal 0.  0.  1.  tindex 1.  1.
0.  1.  0.  normal 0.  0.  1.  tindex 0.  1.
/*top face of cube will be opaque red */
0.  0.  1.  normal 0.  0.  1.
1.  0.  1.  normal 0.  0.  1.
1.  1.  1.  normal 0.  0.  1.
0.  1.  1.  normal 0.  0.  1.
/* y=0 face, will have Lenna on it */
0.  0.  0.  normal 0.  1.  0.  tindex 1.  0.
1.  0.  0.  normal 0.  1.  0.  tindex 0.  0.
1.  0.  1.  normal 0.  1.  0.  tindex 0.  1.
0.  0.  1.  normal 0.  1.  0.  tindex 1.  1.
/* x=1 face, will have mandrill on it */
1.  0.  0.  normal 1.  0.  0.  tindex 0.  0.
1.  1.  0.  normal 1.  0.  0.  tindex 1.  0.
1.  1.  1.  normal 1.  0.  0.  tindex 1.  1.
1.  0.  1.  normal 1.  0.  0.  tindex 0.  1.
/* x=0 face, will be transparent blue */
0.  0.  0.  normal 1.  0.  0.
0.  1.  0.  normal 1.  0.  0.
0.  1.  1.  normal 1.  0.  0.
0.  0.  1.  normal 1.  0.  0.
4 1 2 3 4 attribute 1 texture 1 type TEXTURE
4 5 6 7 8 attribute 2 type PLAIN
4 9 10 11 12 attribute 3 texture 2 type TEXTURE
4 13 14 15 16 attribute 3 texture 3 type TEXTURE
4 17 18 19 20 attribute 4 type PLAIN
shading FLAT reflectance FLAT kd .3 ks .6 n 10 color 1 0 0 opacity 1
shading FLAT reflectance FLAT kd .3 ks .6 n 10 color 1 0 0 opacity 1
shading FLAT reflectance FLAT kd .3 ks .6 n 10 color 1 0 0 opacity 1
shading FLAT reflectance FLAT kd .3 ks .6 n 10 color 0 0 1 opacity .5
/usr/eng/kfife/rle/checkerboard.rle
/usr/eng/kfife/rle/lenna.rle
/usr/eng/kfife/rle/mandrill.rle
```

Figure 1. Cube description using AFIT GEOM format.

polygons that are facing away froni the viewer. The third line is the first line with required information. It states the number of vertices, polygons, attributes, and textures contained in the file.

From the information in line three, we know the next 20 lines, excluding comments, will be vertex lines. Each vertex is defined by the XYZ coordinates and optional normal, color, and texture indices.

After the vertex information, are five lines describing the polygons. The first polygon is defined by four vertices, vertex one, two, three, and four. Attribute line one provides the color and shading information, and the first texture map is also applied to the polygon.

Each attribute line specifies the shading and the reflectance model to use in rendering the image. The *kd*, *ks*, and the *n* are the diffuse lighting component, the specular component, and the power coefficient for rendering specular highlights. This attribute also defines the color and the transparency of the polygon.

The last lines of the GEOM file are the paths to the texture image files. These texture maps must be stored in the run-length encoded (rle) format of the Utah Raster Toolkit.

3.2.2 Extensions to the AFIT Format The previous section gave a basic description of the AFIT GEOM format. However, as AFIT moved into research in virtual environments, the GEOM format did not provide enough information. Additional information was necessary in order to move objects through the environment and to create multiple instances of an object without actually duplicating the geometric description. Capt. John Brunderman filled this gap in the format by creating three additional file formats, TEMPLATE, LINK, and PLACEMENT (1). The three new formats are also ASCII based.

The TEMPLATE format provides the information to create more complex objects from simpler GEOM files. The compone⁻ts are combined in a hierarchical

```
description file for a dataglove
files 2 components 11

file palm.geom
file finger_part.geom

component 1 parent 0 file 1 begin translate 0.0 0.0 0.0 end

// Thumb
component 2 parent 1 file 2 begin translate -0.25 -0.30 0.1
   roty -25.0 rotx 30.0 scale 0.25 0.25 0.24 end
component 3 parent 2 file 2 begin translate 1.0 0.0 0.0 rotx 10.0
   scale 0.7 0.7 0.8 end

// Index finger
component 4 parent 1 file 2 begin translate 0.0 -0.19 0.0 roty 3.0
   scale 0.25 0.24 0.24 end
component 5 parent 4 file 2 begin translate 1.0 0.0 0.0 roty 9.0
   scale 0.8 0.7 0.7 end

// Middle finger
      .
      .
      .
```

Figure 2. Excerpt from a TEMPLATE file (hand.desc).

structure, and each component may be scaled, translated, and rotated. Because of the tree structure of the format, whenever a parent component is rotated, the child components will also move. Part of a TEMPLATE file describing a hand is in Figure 2. There are definite similarities between the TEMPLATE format and the GEOM format. In the example, the number of files and components are listed in the second line. Then the file names for the underlying geometric objects are given. The remainder of the TEMPLATE format consists of lines describing the components. The components are given a number, and the parent component and the file number are specified. The rest of the line gives position and orientation relative to the parent.

```
This is a test file for the linking control algorithms
AAGUN maxtypes 1
USERDEF maxtypes 2

USERDEF type 1 level 1 name Hand dpath hand.desc
AAGUN type 1 level 1 name AAA1 gpath gun.geom
AAGUN type 1 level 2 dpath gun.desc
USERDEF type 2 level 1 name Cube gpath cube.geom
```

Figure 3. Example of a LINK file (test.lnk).

```
This is a test file for Grid Class (Placement)
minx -1000 miny -1000 maxx 1000 maxy 1000 gsize 200

create AAGUN type 1 begin translate -200 -200 0 rotx 20.0 end close
create AAGUN type 1 begin translate -2000 -2000 0 end
create USERDEF type 1 begin translate 20 20 20 scale 1.2 1.2 1.2
   end close
```

Figure 4. Excerpt from a PLACEMENT file (test.dbs).

A LINK file creates the links between categories of objects and their geometric descriptions. Figure 3 shows an example of a LINK file. Lines two and three state that the file specifies one object of category AAGUN and two objects of category USERDEF. The objects for each category are numbered in the *type* fielɑ. The *level* determines the resolution or the level of detail provided by this geometric description. The rest of the line gives a name which may be used in referencing this object and the TEMPLATE or GEOM file that describes the object.

The PLACEMENT format places instances of object types on a grid in a virtual environment database. Figure 4 shows part of a PLACEMENT file. The second line describes the size of the grid. The following lines create instances of the object types and specify positions and orientations for those instances.

3.3 NPS DRP Format

The NPS DRP format was developed by David R. Pratt at the Naval Postgraduate School for describing terrain databases (9). Geometry information is stored in eight different file formats. Two of the file formats are the same, but used for different purposes. Only the six unique file formats will be discussed here.

Terrain polygon information is contained in two different file formats. Files in Terrain Polygon File Format One are processed to produce Terrain Polygon File Format Two files and NPSOFF Model files. Figure 5 shows a sample of the Terrain Polygon File Format One. The lines beginning with *C* make the color table. The first line is color zero; the second line is color one. The C line contains the RGB values and a user defined index. One common use of the index is to link a texture map with this particular color. The remaining lines give polygon information. The *P* line gives the number of vertices in the polygon and an index into the color table. The *D* line gives the XYZ coordinates for the associated vertex. NPS terrain files assume positive X to be east, positive Y to be up, and positive Z to be south. The origin for the terrain is at the northwest corner of the database.

The second terrain format is Terrain Polygon File Format Two. A sample of this format is in Figure 6. The polygons in this file format create a regular grid. The first line of the polygon information specifies the X index, the Z index, the number of vertices, the index into the material/color file, the polygon priority, and a "u" or "l" for upper or lower triangle. The normal to the polygon is found in the next line. And then the vertex information follows in XYZ coordinates. The polygon priority determines the order of rendering. Priority one polygons are rendered first, and the other polygons are then rendered on top.

The Material and Color File Format sets up a color table which is indexed by the Terrain File Format Two files and by the NPSOFF Model files. Figure 7 shows two colors from a material/color file. Comments may be added to the file in the normal "C" style. The *defmaterial* and the *defend* determine the beginning and

```
C 100 100 100 102
C 50 50 50 54
C 50 50 50 43
P 3 0
D 24000.000000 560.832031 16000.000000
D 24125.000000 565.708801 16000.000000
D 24000.000000 606.247192 16125.000000
P 3 0
D 24125.000000 565.708801 16000.000000
D 24000.000000 606.247192 16125.000000
D 24125.000000 560.832031 16125.000000
P 3 1
D 25477.865234 377.649994 23386.757813
D 25367.464844 377.649994 23309.585938
D 25463.337891 377.649994 23354.072266
```

Figure 5. NPS Terrain Polygon File Format 1.

```
128 271 3 12 1 u
-0.004827 0.989798 0.142398
16000.000000 970.483215 34000.000000
16000.000000 988.466431 33875.000000
16125.000000 971.092834 34000.000000
128 271 3 14 1 1
-0.078355 0.973753 0.213696
16000.000000 988.466431 33875.000000
16125.000000 971.092834 34000.000000
16125.000000 998.524841 33875.000000
244 398 4 6 10 u
30613.970703 191.380005 49863.968750
30601.439453 190.279999 49875.000000
30619.599609 190.800003 49875.000000
30622.470703 191.050003 49872.468750
```

Figure 6. NPS Terrain Polygon File Format 2.

```
/* color 7 */
defmaterial
lt_green
emission 0.000000 0.000000 0.000000
ambient 0.064314 0.109020 0.064314
diffuse 0.321569 0.545098 0.321569
specular 0.000000 0.000000 0.000000
shininess 0.000000
alpha 1.000000
defend
/* color 20 */
defmaterial window
emission 0.000000 0.000000 0.000000
ambient 0.521463 0.521463 0.521463
diffuse 0.571542 0.571542 0.571542
specular 1.000000 1.000000 0.936407
shininess 108.712250
alpha 0.707472
defend
```

Figure 7. NPS Material/Color File Format.

end of a material definition. Following the *defmaterial* is the name of this particular material. The RGB values are then given for the emissive, ambient, diffuse, and specular components of the material. The *shininess* coefficient determines the size of the reflective highlight. Transparency is specified by the alpha value.

Individual objects and components of vehicles are described in NPSOFF Model Format. Figure 8 shows an excerpt from a NPSOFF file. The *setmaterial* and the color name specify the color and material for the following polygons until the next *setmaterial.* The *defpoly* starts the polygon definition. It is followed by the coordinates of the normal, the number of vertices, and the coordinates of each vertex. The *defdecal*, *underlay*, *overlay*, and *enddecal* describe coplanar or *decal* polygons. The *underlay* polygons are rendered first, and then the *overlay* polygons are place on top.

```
setmaterial
m106mat8

/* poly #94 */
defpoly
0.000000 1.000000 0.000000
3
1.550000 1.790000 -1.350000
0.313333 1.790000 -1.350000
1.550000 1.790000 0.000000

defdecal
underlay
/* poly #95 */
defpoly
0.000000 1.000000 0.000000
3
-2.160000 1.790000 1.350000
0.313333 1.790000 -1.350000
-2.160000 1.790000 -1.350000

/* poly #96 */
defpoly
0.000000 1.000000 0.000000
3
0.313333 1.790000 1.350000
0.313333 1.790000 -1.350000
-2.160000 1.790000 1.350000

overlay

setmaterial
m106mat4

/* poly #59 */
defpoly
0.000000 1.000000 0.000000
3
-2.000000 1.790000 1.200000
-0.200000 1.790000 -1.200000
-2.000000 1.790000 -1.200000

enddecal
```

Figure 8. NPSOFF Model File Format.

```
O m1 1 101 3
F 0.000000 1.510000 0.000000
F 1.655800 0.380000 0.000000
P 1 0
M -4.345000 0.000000 -2.240000
M 5.695801 2.370000 2.240000

O m106 3 103 1
P 1 0
M -3.430000 0.000000 -1.850000
M 2.920000 2.230000 1.850000
```

Figure 9. NPS Vehicle Parameter File Format.

Vehicle Parameter files combine individual components into objects. An example is shown in Figure 9. The *O* line starts the vehicle description. It contains the vehicle name, the vehicle type index, the destroyed vehicle index, and the number of components in the vehicle. The *F* lines give a translational offset for each subcomponent. The line beginning with *P* contains two binary flags. The first flag specifies if this vehicle contains weapons, and the second flag determines if this vehicle can resupply other vehicles. The two *M* lines contain minimum and maximum XYZ values for the vehicle. Those values may be used to create a bounding box for the vehicle.

Finally, the Model Instance file puts the database together. Figure 10 is a simple example. The *O* line gives the object name. The instance (*I*) lines specify the instantiations of the models. They contain the XYZ translation and an optional rotation value for each instance.

3.4 Flight Format

Unlike the other two geometry file formats which are public domain, Flight format is a proprietary format developed by Software Systems for their commercial

```
O carport
I 16000.00000 20100.00000
I 16000.0000C 20200.00000
O watertower
I 16100.00000 20000.00000
I 16150.00000 20000.00000
O Zville_sign
I 17000.00000 20200.00000
O church
I 18000.00000 20100.00000
o house3
I 18100.00000 20000.00000
I 18150.00000 20000.00000
I 18200.00000 20000.00000
I 18250.00000 20000.00000
I 18300.00000 20000.00000
```

Figure 10. NPS Model Instance File Format.

modeling package, MultiGen. Flight files are also binary files, whereas the other two formats use ASCII files.

3.4.1 Flight Geometry Hierarchy In a Flight file, the geometric description is stored in a hierarchical structure. Figure 11 shows an example of a simple Flight file. The root node is called the *header* and, appropriately, stores information relating to the entire file. The header node has a *group* node for a child. Groups may contain other group nodes, *level of detail* nodes, or *object* nodes. Level of detail nodes are similar to group nodes except they also have switching distance information. The subtree below a level of detail node is only visible when the distance from the eye point to the modeled object is within the switching distance range. Object nodes contain *polygon* nodes. And polygons are composed of *vertex* nodes.

Because of the tree structure, attributes for a node will also apply to the subtree of that node. The color table, the material table, and the texture palette are stored with the header node.

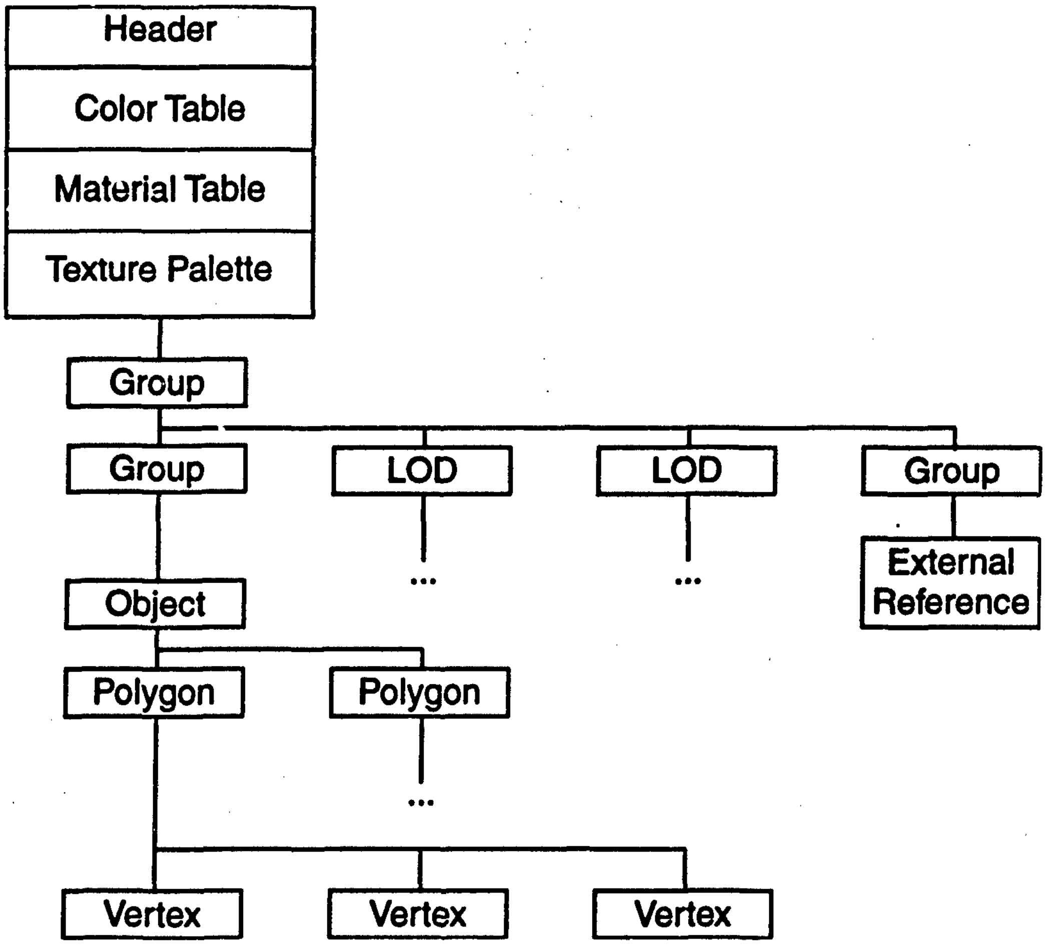

Figure 11. Example of the hierarchy in a Flight file.

ATTRIBUTE	WEIGHT	FLIGHT	NPS	AFIT
File Structure				
Binary	1.0	−		
ASCII	1.0		+	+
Levels of Detail	2.0	+	+	+
Instancing	1.5	+	+	+
Materials				
Emission	0.1	+	+	−
Ambient	0.1	+	+	+
Diffuse	0.1	+	+	+
Specular	0.1	+	+	+
Shininess	0.1	+	+	+
Alpha	1.5	+	+	+
Texture	2.0	+	+	+
Minification	0.1	+	−	−
Magnification	0.1	+	−	−
Wrap Method	0.1	+	−	−
Environment	0.1	+	−	−
Single File or	1.0	−		
Multiple Files	0.8		−	+
Subface Polygons	1.0	+	+	−
Z-axis up	0.5	+	−	+
de facto Standard	3.0	+	−	−
SGI Performer compatibility	2.0	+	−	−
Total Weight		14.4	9.5	9.7

Table 1. Decision table for geometry file formats.

3.5 Geometry Format Comparison

In order to compare the three formats, attributes for each format were given weights and then combined into a decision table. The format was marked with a plus (+) if the format had the attribute and it was an asset. If the format did not have the attribute or if the attribute was a disadvantage, then the format was marked with a minus (−). The total weight for a format was the sum of the attributes marked with a plus. The resulting decision table is Table 1.

The binary or ASCII file structure could have gone either way. Both have advantages and disadvantages. For example, the ASCII formats can be edited with common text editors, which cannot be used on a binary format. However, a binary format should have the advantage of being smaller and, therefore, using less disk storage. But Flight files have extra padding for future enhancements. Thus, Flight files are still about the same size as the corresponding AFIT GEOM files.

There were many similarities. Both Flight and AFIT GEOM formats can represent multiple levels of detail. All three formats can have instances of the same model. And they all have materials and textures. However, Flight contains some additional information to slightly modify the texture. NPS and Flight both have the ability to prioritize coplanar polygons (subface polygons), so rendering occurs in a definite order without Z-buffering problems.

The differences ranged from subtle to major. The NPS format already has the ability to page large terrain databases in and out of memory. Flight and AFIT GEOM would need that extension.

Another difference was the number of different individual file formats. Flight has one format and one file, but the file tends to be large. AFIT GEOM has four file formats. So, the GEOM files are smaller, but they are more numerous. NPS has nine file formats, which can be unwieldy.

The major difference between the formats was the close cooperation between the developers of Flight format and Silicon Graphics (SGI). In the Silicon Graphics world, Flight format is quickly becoming a de facto standard. This is even more apparent since SGI included the ability to read Flight files into its Performer software. Performer is a high performance, rapid rendering environment for prototyping real-time simulation applications (6).

On the basis of the decision table, Software Systems' Flight format was chosen as the geometry file format for the Virtual Cockpit.

IV. Renderinq Flight Format

4.1 Overview

This chapter discusses my implementation for rendering geometric models in Software Systems' Flight format, version 11. A description of the C++ classes and methods is in section 4.2. Section 4.3 discusses the additions made to the Flight format.

4.2 Classes

The C++ classes can be divided into two groups: classes for the nodes of the Flight hierarchy tree, and classes for the tables and pallettes of information. Classes for AFIT extensions to Flight format will be discussed in another section.

4.2.1 Node Classes Each tree node in the Flight hierarchy has a particular record type, i.e., header, group, LOD, etc., and a corresponding C++ class. Table 2 shows the Flight nodes and their classes.

Initially, I duplicated the Flight hierarchy in memory by using the node classes. This was a quick and simple recursive approach to rendering Flight files. To render a group node, render the subtree below it. To render an object node, render the polygons below it. Eventually, the "problem" of rendering a geometric object was divided into smaller pieces which were easily handled. For example, a vertex is rendered by sending the coordinates to the rendering pipeline.

This approach worked, however, the rendering speed was too slow. When flying the Virtual Cockpit with instruments, the frame rate was only one to two frames per second. After "flying" various configurations of the cockpit: with instruments, without instruments, with many polygons, with few polygons, etc., the rendering bottleneck appeared to be the tree traversal time.

FLIGHT NODE	C++ CLASS
bead	MGBead
header	MGHeader
group	MGGroup
level of detail	MGLOD
external reference	MGExtern
object	MGObject
polygon	MGPolygon
vertex bead	MGVertexBead
vertex S	MGVertexS
vertex C	MGVertexC
vertex V	MGVertexV

Table 2. Flight nodes and their C++ classes.

Therefore to increase the frame rate, all of the rendering information is now pushed up into the header node during preprocessing. After the Flight file is read into memory, the tree is traversed, and rendering control flags and coordinates are written into two arrays in the header. Rendering control information is stored in an array of integers and the vertex coordinates and normals are stored in an array of floats. To render the header node, two pointers are incremented down the rendering arrays, and the information is sent to the pipeline. This method increased the frame rate to seven to nine frames per second.

4.2.2 Table and Palette Classes Separate C++ classes were created for the Flight records that describe the color table, the material table, and the texture palette. Another class was also created to contain the transformation matrix found in the Flight transformation record. The names of the C++ classes and their associated Flight records are listed in Table 3.

Flight files may contain only one each of color and material tables, and texture palettes. These records are located immediately after the header information. The classes for the color table, the material table, and the texture palette are primarily

FLIGHT RECORD	C++ CLASS
color table	MGColorTable
material table	MGMaterialTable
texture palette	MGTexturePalette
transformation matrix	MGMatrix

Table 3. Flight records and their C++ classes.

arrays of the information. For materials and textures, the only information that is saved in the array is the index number used to define the material or the texture. After the material or the texture is defined, the index number is used to bind (activate) or unbind (deactivate) the material or texture.

I created the MGMatrix class purely for convenience. It gave me the flexibility to define matrix multiplication, rotations about an axis, translating and scaling operations.

4.3 AFIT Extensions to Flight Format

To the original Flight format, I added two extensions and provisions for a third. This was simple since Flight has two user defined fields, *Special 1* and *Special 2*, in group, level of detail, and object nodes. Additionally, header, group, level of detail, object, and polygon nodes have an optional comment field that may be used for special processing.

4.3.1 Triangle Mesh The Zealous Pursuit terrain files provide terrain elevation on a regularly spaced grid. Figure 12 shows the easiest method for creating a polygonal skin over the terrain data. By connecting the elevation points along the grid lines and then dividing each square into two triangles, we can create a geometric model of the terrain. The resulting terrain model is a *triangle mesh* since it is constructed entirely from adjacent triangles.

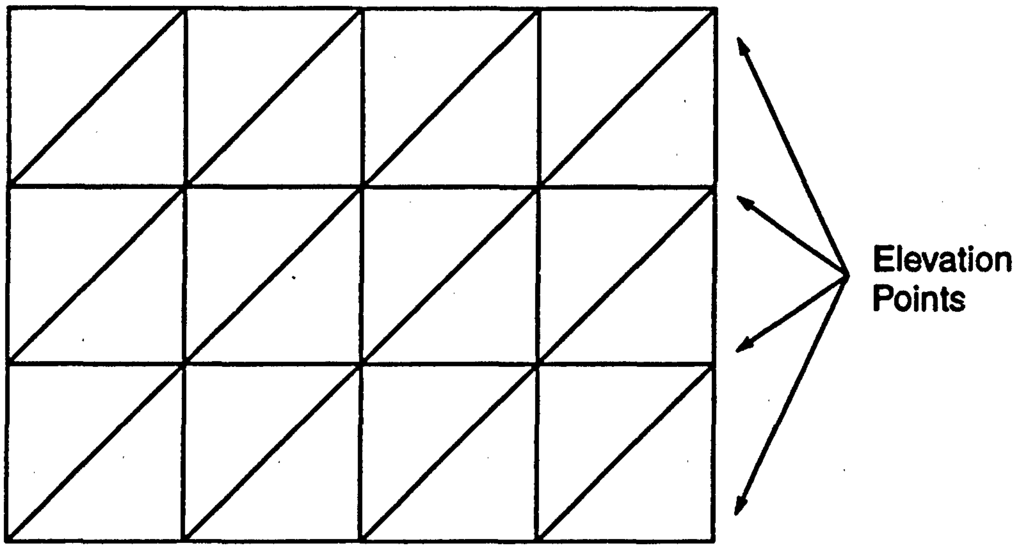

Figure 12. Terrain model created from adjacent triangles.

To render the terrain, each triangle may be sent individually through the rendering pipeline. This means, however, three vertices are sent through the pipeline for each triangle. A vertex in the center of the terrain grid is shared by six triangles, and, therefore, it is sent through the pipeline six times. On the other hand, if the vertices are sent to the pipeline in a different order, information may be shared between adjacent triangles. By sending the vertices in Figure 13 to the pipeline in numerical order, the previous two vertices can be used with the new vertex to render the next triangle. The Silicon Graphics GL library allows us to take advantage of this geometry.

In order to describe a triangle mesh in Flight format, the *Special 1* field of the object node was used. When the field is set to one, the polygons and vertices below this object form a triangle mesh. To render the triangle mesh, the three vertices of the first triangle are sent to the pipeline. Then the first vertex of each subsequent triangle is sent. Therefore, the ordering of the polygons and vertices is critical.

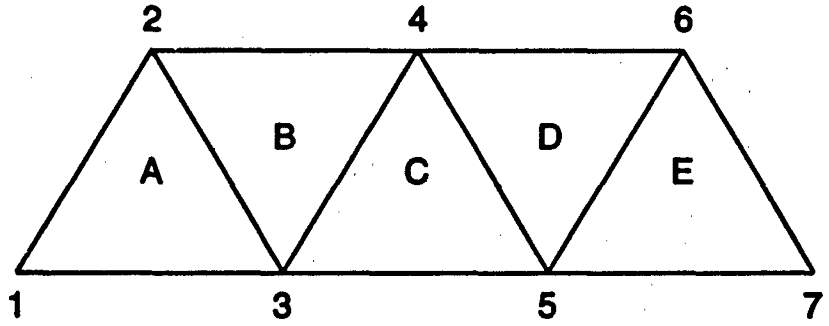

Figure 13. Simple triangle mesh.

For Figure 13, the triangles would be ordered alphabetically in the Flight hierarchy. The vertices for triangle A would be listed numerically, 1, 2, 3. However, the first vertex for triangle B must be vertex 4 because it is the vertex that is not shared with triangle A. So, the vertices for triangle B would be 4, 3, 2. The vertices for triangle C would be 5, 3, 4, etc. Figure 14 shows part of the Flight hierarchy for Figure 13.

4.3.2 Dynamic Models Dynamic models are those objects in the virtual environment that either have moving parts or the entire object moves. For all of the dynamic models, the constraint was made that the moving part must be described in a separate Flight file and referenced as an external reference file.

This constraint allowed the use of the *Special 1* field in the group node above each external reference node. If the field is set to one, then the external reference is dynamic and may move relative to the rest of the model. If the field is zero, then the external reference is static and may not move.

In order to move the dynamic models, an array of transformation matrices is stored with each instantiation of the model. The first matrix transforms the entire model, and the following matrices transform the dynamic external references.

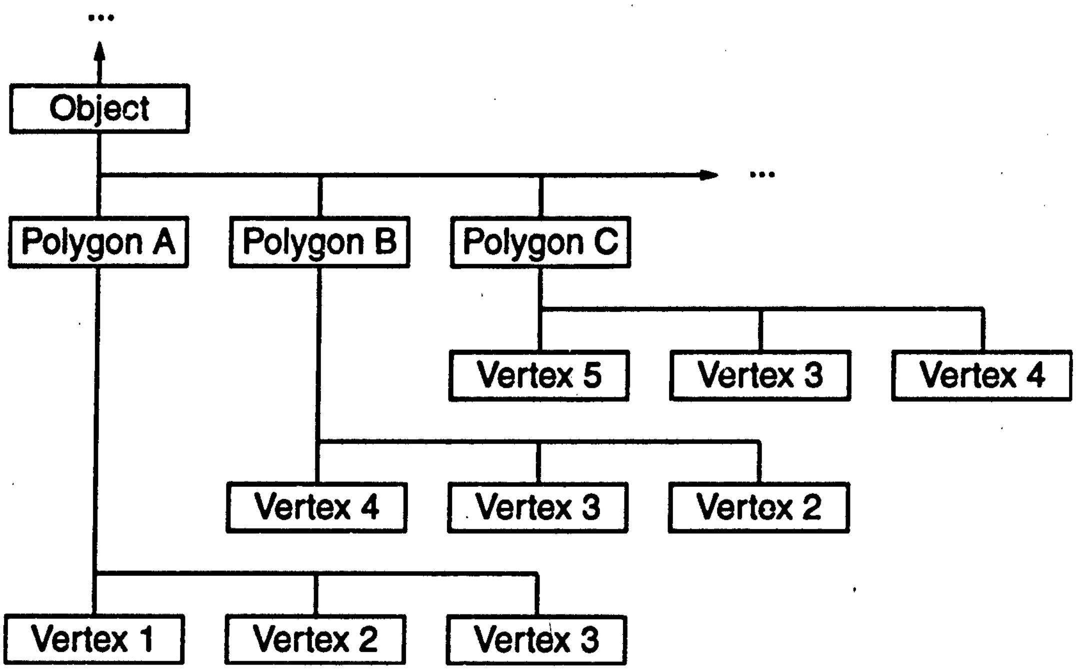

Figure 14. Part of the Flight hierarchy for a triangle mesh.

4.3.3 Terrain Paging The MGTgrid class is an extension that was not fully implemented. Another hierarchical layer was added above the Flight header node. This layer would be a two-dimensional grid of terrain models. Each grid point would reference the header node of a Flight terrain file. However, only the files for the nine closest grid points would be read into memory at one time. By moving across the grid, new terrain files would be paged into memory and old files would be paged out.

Currently, this class behaves like the MGHeader class. It is only a placeholder for future research.

V. Conclusions and Recommendations

Overall, the rendering component of the Virtual Cockpit was a success. We are able to fly through the database and to render the cockpit and other textured objects at an acceptable frame rate. Of course, what is acceptable for the initial version may not be acceptable for the next version.

Using an object oriented approach to the rendering helped tremendously. Flight format quickly separated into definable objects and C++ classes. Once the initial version of the classes was working, it was simple to add additional methods and extend the rendering features one at a time. The object oriented approach also reduced the trauma of integration when I combined my code with the code of Capt. Switzer and Capt. Sheasby.

The following section discusses the results and conclusions for each objective of the project. Recommendations for future work are in the final section.

5.1 Conclusions

There were three objectives for the rendering portion of the Virtual Cockpit. The first objective was to render stationary and moving objects in the synthetic environment. Second, the ability to render texture maps was necessary to increase the visual realism without increasing the total polygon count. The last objective was to maintain a frame rate of at least ten to fifteen frames per second.

5.1.1 Static and Dynamic Models The static models are the terrain and all of the non-moving objects in the database. The rendering of static objects was achieved with the rendering of Flight files in general. Figure 15 shows a close up of the terrain database with external references to a castle model and to a tree model.

The dynamic models are the objects that move through the database. By including a transformation matrix with an external reference, the *static* external

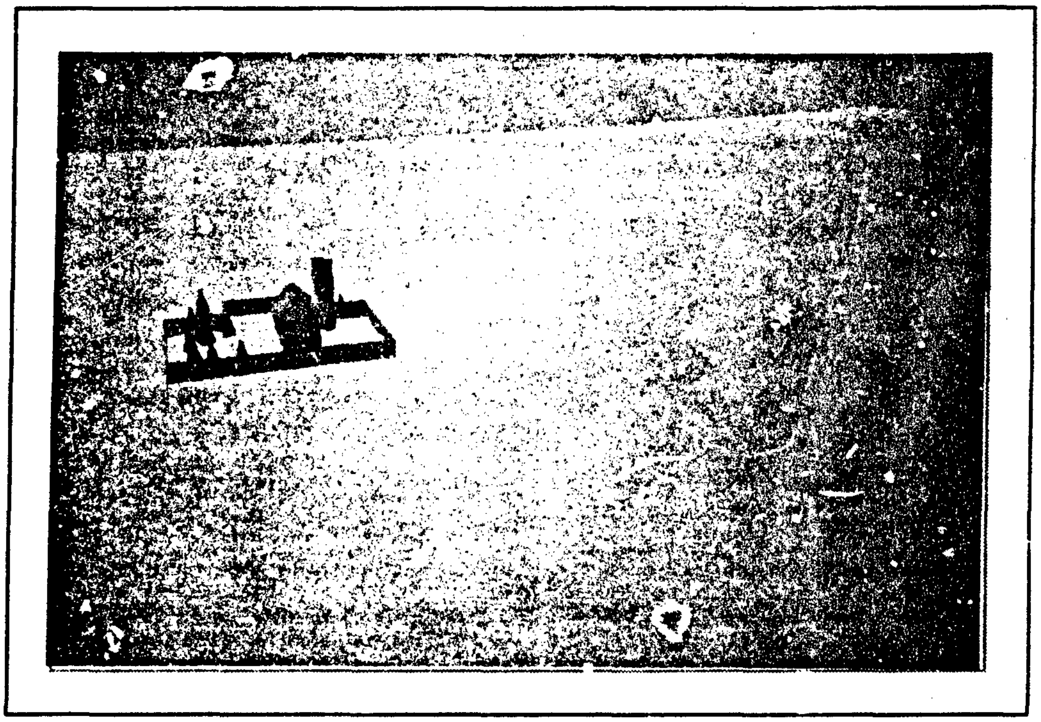

Figure 15. Static model of a castle and trees.

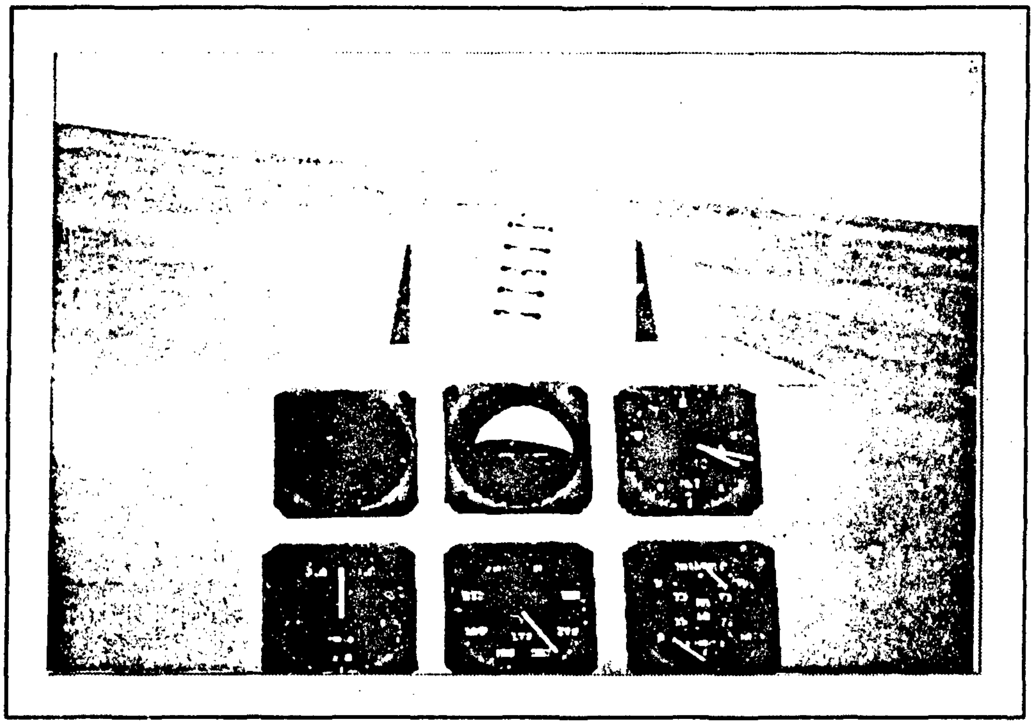

Figure 16. Instruments in the virtual cockpit.

reference became a *dynamic* model. Figure 16 shows the moving dials and objects that make up the instrument panel in the Virtual Cockpit. The F-15 in Figure 17 is another example of a dynamic object.

5.1.2 Textures The proper application of textures can dramatically increase the realistic impression of computer simulations. Textures add detail and variety to the database. Therefore, the ability to use textures was absolutely essential. Figure 18 shows the terrain and the simplified cockpit without an airframe. The terrain was textured to resemble sand. Texturing can also be seen in Figure 16. The instrument panel was textured with one texture image to provide the numbers and the *background* for the dials.

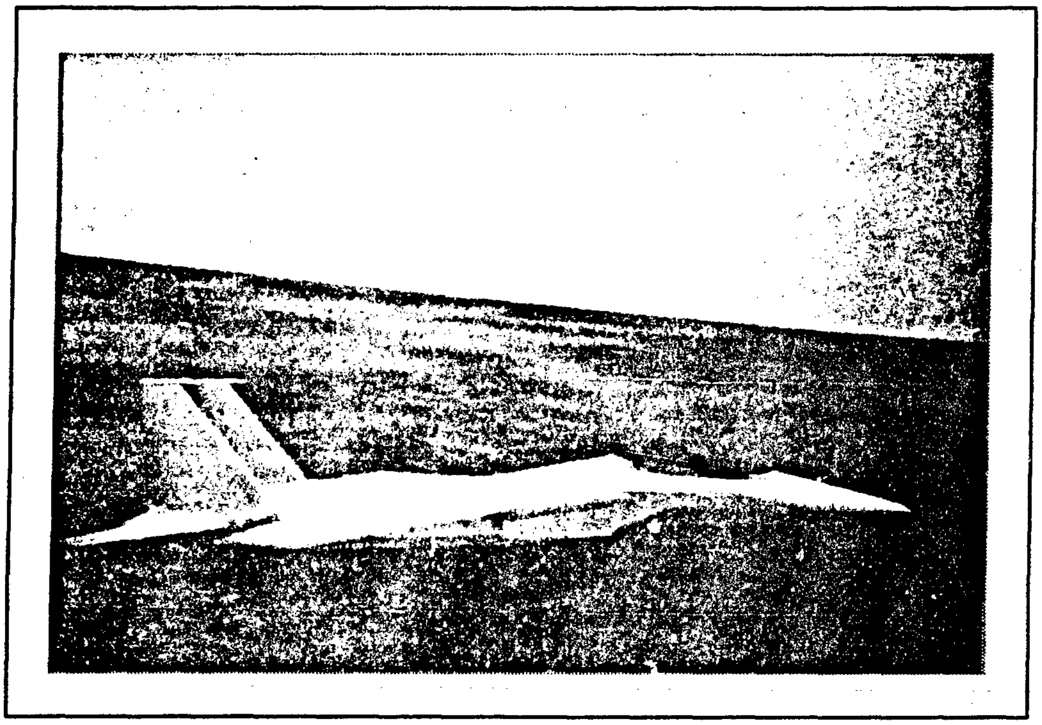

Figure 17. Dynamic model of an F-15.

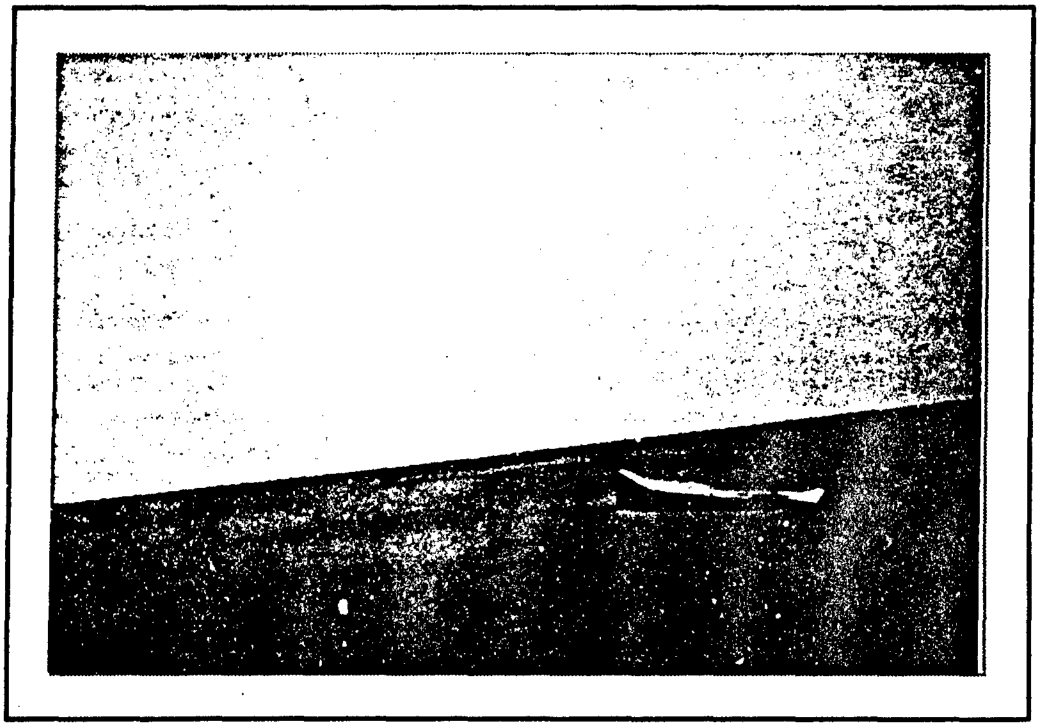

Figure 18. Virtual cockpit without the aircraft.

5.1.3 Frame Rate Although the objective was to achieve at least ten frames per second, the frame rate had the lowest priority for this version of the Virtual Cockpit. The goal was to make it work, first, and then to improve the performance.

The Virtual Cockpit now has a frame rate of seven to nine frames per second. This frame rate did not achieve the objective, but flying the Virtual Cockpit at seven frames per second is not difficult. However, if the frame rate drops to three or four frames per second, the cockpit becomes difficult to fly because of overreacting to the display images. Therefore, the current frame rate for the cockpit can not decrease very much before the cockpit becomes unflyable. Future versions of the Virtual Cockpit will probably require a higher frame rate in order to have more leeway before dropping to an unuseable frame rate.

5.2 Recommendations

Every thesis project has room for future improvements, and this one is certainly no exception. Some recommendations for each of the objective areas are given.

Many of the static and dynamic models need two versions: a *working* one and a *destroyed* one. We also need the ability to switch from the working model to the destroyed model. After a vehicle is hit, we should see flames and smoke, then just smoke, and finally the destroyed model.

The data structure that holds the external references and the dynamic references also needs to be more efficient than the current structure. Currently, it is a simple linked list, which worked well for the number of moving vehicles we had. However, as the number of different referenced models increases, the length of the linked list will increase, and the average search time of the list will also increase. Therefore, the external and dynamic references need to be stored in a data structure which may be searched more efficiently than the linked list.

In the area of texture mapping, having the ability to dynamically change the texture indices of a polygon would create some interesting effects. Normally, the texture indices for a vertex do not change. But, changing the texture indices at runtime would cause the texture map to move on the polygon. By smoothly varying the texture indices, a water texture map would move across a polygon giving the effect of flowing water. The same technique could be used to move a cloud texture across a polygon sky.

Finally, the issues of frame rate and performance will become more critical in future versions of the Virtual Cockpit. Some possibilities for improvement are in the algorithms for coarse clipping and polygon culling. Currently, Flight file objects are clipped if their bounding boxes are outside of the viewing volume. Changing the algorithm to use a bounding sphere instead of a bounding box would reduce the number of dot product calculations from eight to one. At the same time, polygons could be culled according to their priority level. If the frame rate is below a certain threshold, then low priority polygons are not rendered. As the frame rate increases, more of the low priority polygons could be rendered.

Bibliography

1. Brunderman, John A. *Design and Application of an Object Oriented Graphical Database Management System for Synthetic Environments.* MS thesis, AFIT/GA/ENG/91D-01, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1991.
2. Clark, L. Charles and Thomas C. Brown. "Photographic Texture and CIG: Modeling Strategies for Production Databases." *Proceedings of the 9th Interservice/Industry Training Systems Conference.* 274–283. November 1987.
3. Cosman, Michael A. "A System Approach for Marrying Features to Terrain." *Proceedings of the 11th Interservice/Industry Training Systems Conference.* 224–231. November 1989.
4. Foley, James D. and others. *Computer Graphics: Principles and Practice* (second Edition). Reading, MA: Addison-Wesley Publishing Company, 1990.
5. Heckbert, Paul S. "Survey of Texture Mapping," *IEEE Computer Graphics and Applications*, *6*:56–67 (November 1986).
6. *IRIS Performer Programming Guide.* Document Number 007-1680-010. Silicon Graphics Inc, Mountain View CA, 1992.
7. Latham, Roy, President. Personal interview. Computer Graphics Systems Development Corporation, Mountain View, California, 14 September 1992.
8. Orlansky, Jesse and Joseph String. "Reaping the Benefits of Flight Simulation." *Computer Image Generation* edited by Bruce J. Schachter, New York: John Wiley and Sons, 1983.
9. Pratt, David R. and Michael J. Zyda. "NPS Terrain Database Format." Naval Postgraduate School, Monterey, CA, February 1992.
10. Sheasby, Steven Michael. *Management of SIMNET and DIS Entities in Synthetic Environments.* MS thesis, AFIT/GCS/ENG/92D-16, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1992.
11. Shuey, David and others. "PHIGS: A Standard, Dynamic, Interactive Graphics Interface," *IEEE Computer Graphics and Applications*, *6*:50–57 (August 1986).
12. Switzer, John C. *A Synthetic Environment Flight Simulator: The AFIT Virtual Cockpit.* MS thesis, AFIT/GCS/ENG/92D-17, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1992.
13. *WAR RREAKER Simulation Phase One: The Zealous Pursuit Exercise.* Technical Report. Defense Advanced Research Projects Agency, 1992.

14. Zyda, Michael J. and others. "Flight Simulators for Under $100,000," *IEEE Computer Graphics and Applications*, *8*:19–27 (January 1988).

Vita

W. Dean McCarty was born on September 19, 1957 in Urbana, Illinois. In 1975, he graduated from Champaign Central High School. He attended the University of Illinois at Urbana-Champaign, and graduated in 1979 with a Bachelor of Science degree in Computer Science. From 1980-1982, he served as a missionary in San Antonio, Texas for the Church of Jesus Christ of Latter-day Saints. He entered the U.S. Air Force in 1985, and was assigned to HQ TAC at Langley AFB, VA as a communications officer. In 1987, he transferred to HQ First Air Force at Langley AFB where he managed the computer systems for the CONUS NORAD Region Operations Control Center. Following his assignments at Langley AFB, he attended the Air Force Institute of Technology. He separated from the Air Force in 1992, and is employed by a defense consulting firm. He and his wife, Carol Shuldberg McCarty, are the parents of five children: Sarah, Jared, Joshua, Anna, and Rachel.

Permanent address: 504 E. Madison Avenue
Casey, Illinois 62420

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY

March 1993

Master's Thesis

4. TITLE AND SUBTITLE

RENDERING THE OUT-THE-WINDOW VIEW
FOR THE AFIT VIRTUAL COCKPIT

6. AUTHOR(S)

W. Dean McCarty

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology, WPAFB OH 45433-6583

AFIT/GCS/ENG/93M-04

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

DARPA/ASTO
3701 North Fairfax Drive
Arlington, Va 22203

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

Approved for public release; distribution unlimited

13. ABSTRACT

The Air Force Institute of Technology (AFIT) is developing a distributed interactive flight simulator, the Virtual Cockpit, using commercial graphics workstations and helmet mounted displays. The Virtual Cockpit communicates with other simulators via local and long-haul networks using the SIMNET protocol.

The work reported in this thesis focuses on developing the terrain database for the synthetic environment and on rendering the pilot's view of the database.

There are many different file formats for describing 3-dimensional geometric polygonal objects. An analysis of three formats, AFIT GEOM, the Naval Postgraduate School (NPS) DRP, and Software Systems' Flight, is presented. Each file format is described, and their attributes are compared in a decision table.

A series of C++ classes were developed to render the file format used by the terrain database. These C++ classes, which include classes for stationary and moving entities, and for textures, are discussed. A technique for increasing the rendering speed is also presented.

14. SUBJECT TERMS

Distributed Interactive Simulation, Synthetic Environments, Computer Graphics, Flight Simulator

15. NUMBER OF PAGES

40

16. PRICE CODE

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