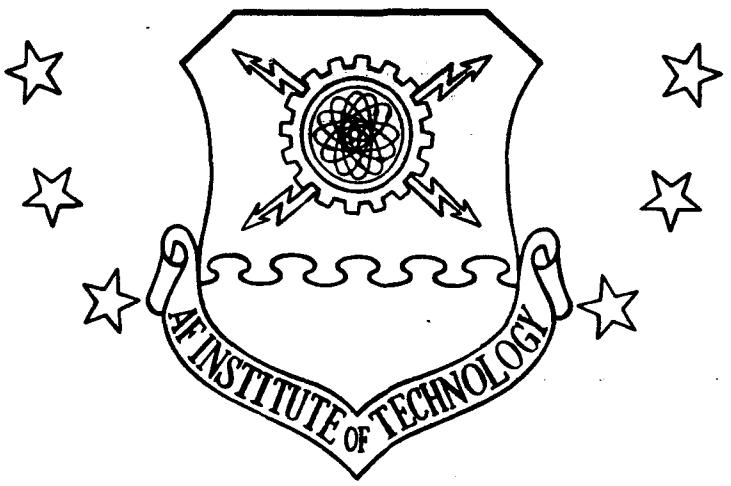


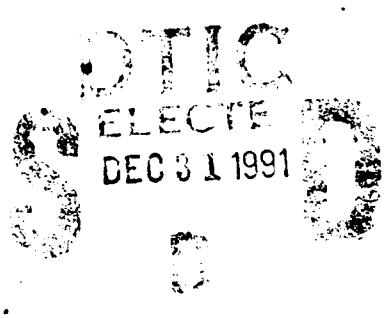
DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GA/ENG/91D-01



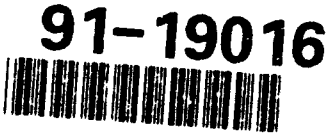
DESIGN AND APPLICATION OF AN OBJECT ORIENTED GRAPHICAL DATABASE MANAGEMENT SYSTEM FOR SYNTHETIC ENVIRONMENTS

THESIS

John A. Brunderman
Captain

AFIT/GA/ENG/91D-01

Approved for public release; distribution unlimited



AFIT/GA/ENG/91D-01

DESIGN AND APPLICATION OF AN OBJECT ORIENTED GRAPHICAL DATABASE MANAGEMENT SYSTEM FOR SYNTHETIC ENVIRONMENTS

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Astronautical Engineering

John A. Brunderman,

Captain

December, 1991

Acknowledgments

I would like to thank all those who have given me support during this thesis effort. First I want to thank Lt. Colonel Phil Amburn, my thesis advisor, who took on the extra responsibility of a cross-departmental student. His enthusiasm, guidance and sense of humor kept me on track and made the whole experience a reasonably pleasant one. I wish to also thank the other thesis students in the graphics lab. Their critiques and comments helped in the design and implementation of my software system. I have a special thanks to give to the Rorabacher's and the Parrott's, whose prayers and encouragement lifted me up during some difficult times.

My deepest 'thank you' goes out to my family. To my two sons, Eric and Adam, who understood when I did not have time to wrestle. And most of all to my wife, Cynthia, who kept the faith and persevered during this past year and a half. She was my confidant, my counsellor, my proof reader, my best friend, and so much more. I dedicate this thesis to you with love.

John A. Brunderman

Table of Contents

List of Figures

List of Tables

Abstract

The Air Force Institute of Technology (AFIT) is investigating the use of synthetic environments for military applications under the sponsorship of Rome Laboratories (RL). Areas under investigation include mission planning, battle management and flight simulation.

The work reported in this thesis focuses on the object-oriented design and implementation of the Graphical Database Management System (GDMS) used to support research in these areas. GDMS provides the data structures, file formats and algorithms to manage and render hierarchical, three-dimensional, polygonal models. Flexibility and adaptability were key factors in its design.

A secondary objective of this research was to demonstrate the functionality of GDMS through the development of a characteristic application. This led to the design and implementation of a DataBase Generation System (DBGen) for the construction and manipulation of synthetic environments. DBGen allows a user to orient, scale, move, delete and add multi-resolution objects to synthetic environments interactively. It also provides a real time fly-through capability for immediate feedback on the dynamic response of the database.

The development of the Graphical Database Management System was successful. The design and implementation of four major synthetic environment applications based on GDMS tested its flexibility, while the rapid incorporation of a number of special effects algorithms in the final stages of the development cycle demonstrated its adaptability.

The development of the Database Generation System was also successful. In addition to providing an excellent platform for testing GDMS funtionality, it proved to be an effective tool for placing 3-D objects on terrain.

DESIGN AND APPLICATION OF AN OBJECT ORIENTED GRAPHICAL DATABASE MANAGEMENT SYSTEM FOR SYNTHETIC ENVIRONMENTS

I. Introduction

1.1 Overview

In military operations, the quality of the planning has a considerable effect on the success of the mission. From target selection to mission debrief, an extraordinary amount of training, planning, and coordination takes place to give a pilot the highest possible chance of success. Success is defined not only as hitting the target, but as returning the pilot and aircraft to fly again. We can not, generally, afford to trade a weapon system and crew for each target we need to take out. As our systems and crews get more specialized and expensive, this factor becomes even more critical.

The vast majority of real world tactical planning is done manually with paper, pencil, slide rules and various charts, maps and photographs of the battle area (20:5). The battle field terrain is the central focus. The terrain drives ingress and egress routes, radar masking options, altitude profiles, fuel requirements and numerous other considerations. The shear volume of required information makes manual mission planning time consuming and potentially faulty.

Computer assisted planning aids, like TAC's Computer Assisted Force Management System (CAFMS), were developed to store, retrieve and process information for the planner (4). These early computerized planning aids presented information in statistical textual formats, wherein the user had to visualize the battle arena. Unfortunately, the human mind is limited as to the amount of abstract data it can track at once (3). To manage and present more information to the user, researchers have investigated the use of 3-Dimensional graphical representations. These 3-D, graphically generated depictions of the 'real' world are called "synthetic environments". Terrain-based synthetic environments include, but are not limited to, the following components: 1) the basic surface terrain in the area of interest, 2) features like rivers, forests, fields and roads, 3) objects like cars, trains, and buildings, and 4) indistinct items like fog, visibility and radar envelopes.

Synthetic environments are not new to the military. They have been used in high fidelity flight simulators for some time. What is new is that the cost of the hardware needed to generate and manipulate a synthetic environment has dropped dramatically.

With the advent of general purpose graphic workstations, like the Silicon Graphics Iris 4D, synthetic environments are finding their way into a large number of other applications. Recent years have seen its application to military planning. Current systems using synthetic environments include:

The Route Evaluation Module (REM) for route selection and planning (2).

SCENARIO, a 3-D stereoscopic mission planning system (16).

And the Battle Management Visualization System, an experimental system for viewing Red Flag data (21).

For these applications, the objects and the terrain comprising the synthetic environment are maintained within a graphical database. A graphical database is comprised of geometric descriptions containing vertices, normals, and attributes that approximate how light interacts with the surface. Several graphical standards have emerged in recent years for dealing with the complexities and management of graphical databases. Of particular interest is International Standards Organization (ISO) standard graphical package called PHIGS (10). PHIGS (Programmer's Hierarchical Interactive Graphics System) provides an extensive set of graphical database manipulations, while promoting portability between applications (7). PHIGS includes the following features and capabilities: it uses a 3D, floating-point coordinate system and implements the standard 3D viewing pipeline (7); it maintains a database of objects which can be manipulated through simple editing calls; and it operates in an abstract 3-D world coordinate system, not 2-D screen space (10). These capabilities match well with the requirements for a synthetic environment graphical system. PHIGS does not address all our concerns, but it provides a good baseline from which to draw capabilities.

1.2 Problem Statement

The utility of the synthetic environment interface to applications in mission planning and battle management is an open research issue. The Air Force Institute of Technology (AFIT)

is developing its Synthetic Environments Laboratory (SEL), under the sponsorship of Rome Laboratories (RL), to address these issues.

AFIT developed the specific requirements for this research. To meet them, the SEL needed a general purpose graphical database management system from which to base synthetic environment applications. The system had to be adaptable to allow for experimentation and changing research objectives. Commercial systems were unacceptable on this point due to their proprietary status. Thus, an in-house program to provide the baseline capability for future investigations was developed.

Additionally, the capability to generate test environments were also needed. An application based on the above database management system had to be constructed to generate the files for use in future application.

1.3 Research Objectives

The main objective of this thesis was to design and implement an object-oriented Graphical Database Management System (GDMS) used to support SEL research related to mission planning, battle management and flight simulation. The GDMS must provide the data structures, file formats and algorithms to manage and render hierarchical, three-dimensional, polygonal models within a synthetic environment. Flexibility and adaptability were key requirements in the design.

The system must accommodate the graphical database storage and rendering needs of the other efforts in the SEL. These efforts included the development of an object oriented flight simulator (19), a synthetic environment battle management system (8), a terrain generation system (5), a geometry viewing program (15), and a synthetic environment generation system (See Chapter 4). The GDMS is the glue holding all these applications together. It acts as the integrator, insuring compatibility and interaction between them.The second objective of this research was to demonstrate the functionality of GDMS through the development of an application for the construction and manipulation of synthetic environments. This led to the design and implementation of the DataBase Generation System (DBGen). DBGen had to allow a user to orient, scale, move, delete and add multi-resolution objects to synthetic environments interactively. It also had to provide a real time fly-through capability for immediate feedback on the dynamic response of the database.

DBGen was not intended to be a modeler, just an organizer. Objects descriptions are selected from a list of predefined geometric models, set up by the user in a special file, and added to the terrain descriptions provided by Duckett's Terrain Generation system (5). The resulting synthetic environments are saved to disk for the demonstration, analysis and testing of other SEL application.

Figure 1 shows the relationship of the data files, GDMS, and the major applications within the SEL. The Terrain Generation System constructs the terrain files (GEOM, LINK, and PLACEMENT) for use by GDMS. GDMS is built around reading and manipulating these data files. Applications then use GDMS to handle their graphical database requirements.

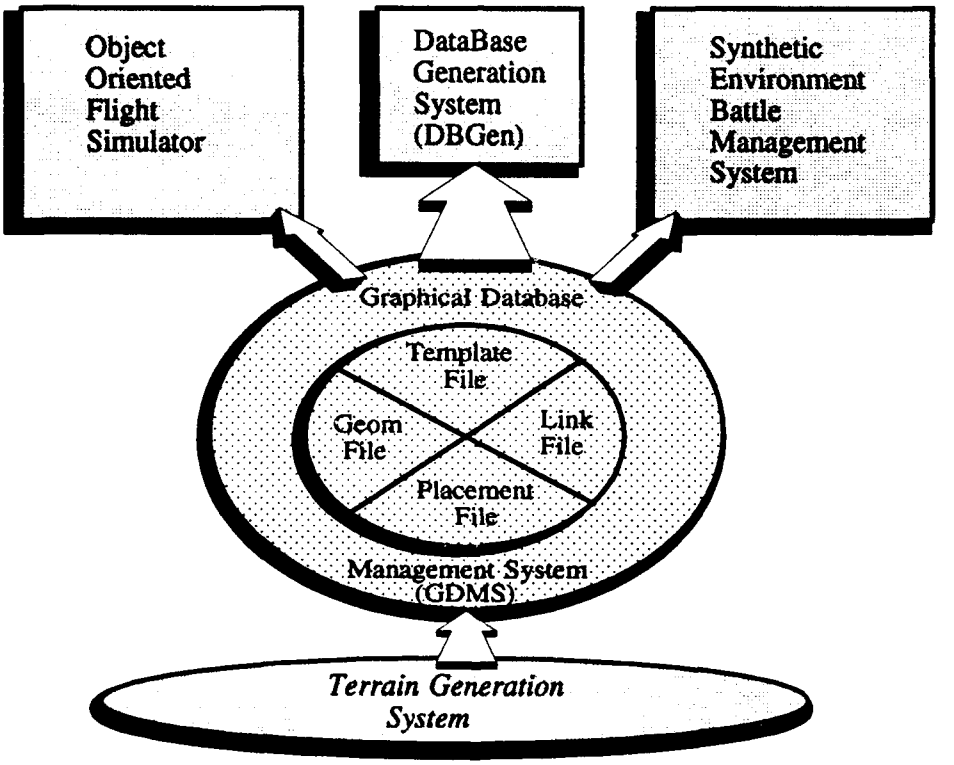


Figure 1. Relation between various thesis efforts.

1.4 Requirements

Three common requirements surface in applications using synthetic environments. Each application must 1) visualize the environment, 2) manipulate objects in the environment, and 3)

interactively move through the environment. Based on the particular application, these requirements might change order of importance. The following paragraphs examine each of these in more detail.

1.4.1 Visualization The first requirement is to visualize the environment. Environment visualization is subdivided into three areas, projection, resolution, and realism. Projection deals with how the environment is mapped to the display device to produce images. Two approaches are used, perspective and orthographic. The perspective view gives the environment a 3-D quality, thus displaying it in a manner similar to the way in which the eye is accustomed. Distant objects appear smaller than near objects. This provides motion cues to the viewer, thus helping our cognitive understanding of speed and distance in a moving scene (7). An orthographic view provides direct correlation between screen space and world space. It is used for measurement and positioning purposes. DBGen uses an orthographic view in an overhead look at the terrain for object placement purposes.

Resolution is a measure of the detail in a scene. The required level of detail is based on the task at hand. Mission planning requires a high level of detail, so a planner can account for as many factors as possible. Battle management, however, might get by with less, since it is primarily an overview system and tends to use symbolic representations of objects in the environment (12). Another factor in the resolution required is the distance from eyepoint to terrain. Close objects require higher detail than objects far away. The graphical database management system should be able to handle the diverse resolution requirements for our intended applications.

Realism deals with the type of information presented to the user. It can be divided into two categories, classification and density (20). Classification deals with the ability of the system to render the synthetic environment so a user can recognize land use and generic features, such as forests, fields, towns, etc. In our intended applications, this allows a user to rapidly identify terrain types and potential routes. Object density is a measure of the number of surface objects placed over a given area of terrain. Higher object densities help with spacial perception, like altitude control in flight simulators (11).

The proficiency of a graphical database management system in providing classification cues and high object densities is a function of the hardware available and the data structures used. The

push is always on to increase the realism of the scenes. For our system, a means to try out various rendering algorithms was required.

1.4.2 Manipulation The second requirement is to create and manipulate objects in a graphical database. Whether inventing a synthetic environment, or trying to match a specific battle area, every object needed, from terrain to truck, has to be modeled and inserted into the database. The management of these objects (storing, creating, editing, rendering) is the major design criterion of the Graphical Database Management System.

This requisite drives the following requirements. The system must be able to build up complex objects in PHIGS-like hierarchies from geometric primitives. The primitives in this case will not be individual polygons, but previously created models on disk. To manage the polygonal count rendered for each frame, the system must be able to incorporate multiple levels of resolution for each object. To provide flexibility in object construction, a hierarchical structure is needed. An additional structure is required to break down the synthetic environment into smaller blocks for efficient access and rendering in the database. A mechanism for viewing the database from different angles with different perspectives is also needed. Finally, the ability to store and retrieve all this information from disk is required for portability between applications.

1.4.3 Interactive Movement The final requirement for a graphical database management system involves moving through the environment, viewing it from different angles. The performance of the system effects the smoothness and flow of the images, which determines its usability for particular applications.

Graphical performance is often measured by frame rate, i.e., how many image frames per second the system can produce of the given environment. It is inversely proportional to the number of geometric primitives processed through the graphical pipeline. The higher the number of polygons and pixels covered, the more time it takes to draw each frame, and the slower it goes.

For flight simulation, the update rate of a "fly-through" capability is very important (20). The goal is to display images at a rate comparable to a motion picture. Motion pictures use a standard rate of 24 frames per second (13). 30 frames per second is targeted for smooth animation because it falls just above the normal eye's threshold for observing display flicker (13). High

end flight simulation systems use even faster frame rates, 60 Hz and above, to reduce eye fatigue and provide highly accurate animation. 10 Hz is the minimum frame rate for using animation as interactive feedback (20). Update rates below beyond this level introduce unacceptable lags in the response of the image to inputs from the user, causing over-correction.

In the mission planning application, flying the route, or observing it from enemy positions is still important, however the update rate has much less effect. The limiting requirement for mission planning systems is on level of detail. Route selection, radar ground clutter, line-of-sight consideration and nav-point identification are all significantly impacted by the resolution of the terrain. Balancing these two factors is a large part of a mission planning design.

The database management system should be able to meet the performance requirements of the intended applications. Data structures and algorithms should be designed with speed in mind.

1.5 Constraints

There were several significant constraints effecting the overall system. Each of these constraints, to varying degrees, effected the speed and efficiency of the design.

The first constraint dealt with money. The capability described above is not unique. It is available on the commercial market at high levels of fidelity. Unfortunately, it also comes at a high price. These systems can build up detailed synthetic environments, but they require special purpose hardware to render. The SEL needed a public domain graphical database management system that could be modified at necessary to test out new concepts and algorithms.

Given the development of a custom system, the next concern was the format for the basic geometric description of an object. Over the course of several years, AFIT has developed a series of tools to help with the modeling process. Tools to generate surfaces of revolution, extrude shapes, scale, rotate, translate and combine files. This capability works through a special file format designated the AFIT Standard Geometry File Format (GEOM) (6). The GEOM format is not the most efficient in terms of disk space or speed of translation, being ASCII based, but a whole library of GEOM file descriptions already exists from which to draw objects. Since replacing these tools and models would require extensive time and effort, I chose to keep the GEOM format as the baseline model description language.

The third constraint dealt with the hardware platform available and the software language used for the development. The hardware platform was a Silicon Graphics Iris 4D workstation. It uses special hardware components in conjunction with its own graphical library to speed up polygonal throughput. The models used, 85/GT and 310/GTX, are capable systems, yet still lacking in certain areas, like hardware texture mapping and hardware alpha blending. This limits some of the display techniques usable in the rendering system.

For software, the choice was between standard C and C++, an object oriented language. These are the standards for the Graphics Laboratory at AFIT. Although C would probably execute faster due to the reduced overhead, the advantages in C++, like encapsulation, inheritance and polymorphism, far outweighed the slight reduction in speed.

The fourth constraint was one of timing. The Graphical Database Management System was the basic software platform upon which most other SEL applications were being built. The Flight Simulator needed it for the out-of-cockpit view. The Battle Management system needed it for battle field display. The Special Effects project needed it as its baseline for rendering enhancements. AFIT required each of these efforts to be completed within the 1991 thesis cycle. Therefore, I structured my research effort to provide a basic capability as quickly as possible, with incremental enhancements coming later. These and other design considerations are discussed in detail in Chapter III.

1.6 Thesis Overview

This document is composed of five chapters. The first is the introduction, which contains the background, problem, research objectives, requirements, and constraints. The second chapter covers applicable background information, to include commercial system, graphical standards, and other military applications related to this topic. Chapter III provides a detailed review of the design and implementation of the Graphical Database Management System. Chapter IV describes the design and implementation of the Synthetic Environment Database Generation System. This is an application of GDMS. The final chapter provides performance measurements, strengths and weaknesses, with recommendations for future work.

II. Background

2.1 Overview

This chapter provides the background to put this thesis effort in context to other work in the field. It involves research into synthetic environments, which require two basic capabilities. The first is the ability to manage the graphical database making up the synthetic environment. The second is the ability to construct synthetic environments for use in research applications. The Synthetic Environments Laboratory (SEL) at AFIT has been set up to coordinate the research based on these capabilities in the areas of mission planning, battle management, and flight simulation.

Section 2.2 describes graphical standards, from which the core of GDMS capabilities were patterned. Sections 2.3 and 2.4 give a quick history of the hardware used to produce synthetic environments, from Computer Image Generators to graphic workstations. Section 2.5 provides an overview of several recent military applications in synthetic environments using graphic workstation. Section 2.6 examines drawing interfaces to extract a common paradigm for use in DBGen. Section 2.7, the summary, ends the chapter by showing how all this information relates to the task at hand.

2.2 Graphical Standards

The primary function of GDMS is to manage the graphical information comprising the synthetic environment. Research into the management of graphical information has produced standards like PHIGS and PHIGS PLUS. These packages offer significant capability for the storage, manipulation and rendering of 3-Dimensional graphical objects. Although AFIT did not have an implementation of a PHIGS package for the target hardware, nor the money to purchase one, the graphical standards provided a pool of capability from which to draw. The following is a short description of these packages.

PHIGS descended from the CORE and GKS standards and their functions closely resemble one another (9). PHIGS is an ISO standard describing a graphical reference model that provides functions for application modeling and 3-Dimensional interactive computer graphics (10). PHIGS PLUS is an enhancement of PHIGS to allow for higher quality 3-D picture rendering techniques (lighting, color and shading) as well as more sophisticated geometries (tesselated surfaces) (10).

PHIGS is a hierarchically modeled system. It maintains a mechanism, called a Central Structure Store (CSS), to hold information about objects. Objects are defined geometrically in a local modeling coordinate system. Modeling transforms are applied to position and orient components relative to one another. A local transformation positions an object relative to its parent's origin and coordinate system. Since these transforms are inherited from parent structures, complex hierarchical transformations can be created (10). The CSS stores all this geometric information to facilitate rapid display traversal during screen regeneration. PHIGS uses a 3-D, floating-point coordinate system and implements a conventional 3-D viewing pipeline (7:Chapter 6). It operates totally in this abstract 3-D world coordinate system, not in a 2-D screen space. PHIGS does not address all our concerns, but it provides a good baseline from which to draw capabilities. Any graphical management system implemented should match much of the capability, if not the format of these standards.

2.3 Computer Image Generators

Synthetic environment technology has been around for some time. Its first application was in early flight simulators where it was used to visualize the approach and landing phase. Miniature models of the runway environment, a Terrain Model Board (TMB) were built up by hand, and placed near the simulator. A special camera was then slaved to the virtual position of the aircraft to feed images of the TMB to the pilot. The models were expensive to build and maintain, and could not easily be modified to provide different environments.

To overcome these drawbacks, the military funded much of the development of the Computer Image Generator (CIG). Imagery from CIGs first appeared in flight simulators as tools to display night-only scenes using point light sources for distant objects. A very limited capability existed to display three-dimensional objects like runways or buildings. The synthetic environments, being computer generated, allowed the user some choice of the areas over which to fly. The need for a variety of environments quickly grew, which gave birth to a class of software for generating the graphical databases for the hardware.

Advances continued in CIG technology to the point where computers today can update 16000 textured, anti-aliased, shaded faces at 60 Hz, and provide weather effects, too (20:9). The supporting set of database programs also kept pace over the years. These systems generate the

CIG database files directly from Digital Terrain Elevation Data (DTED) and cultural feature data, as well as from human input.

This capability comes at a price, though. Typical top-of-the-line CIGs fall in the multi-million dollar range (20:9). The support package needed to generate databases adds additional costs. In addition, these systems are proprietary to the manufacturer, and can not be modified for research purposes. Alternatives do exist, however.

2.4 *Graphical Workstations*

Recent advances in computer chip technology has brought synthetic environment applications down to the level of the graphic workstation. Several student projects originating at the Naval Postgraduate School have illustrated the utility of graphic workstations for command and control visualizations (20)(1)(22).

Graphic workstations offer several capabilities useful to our effort (20:17). They provide a mechanism for fast drawing of polygonal meshes using either Gouraud or flat shading. They have significant integer and floating point processing capacity. And, they provide network connectivity.

The Silicon Graphics IRIS/4D is the target platform for this thesis effort. It provides the basic capabilities needed for manipulation of 3-Dimensional geometric models, which are used to build synthetic environments. A library of graphic support routines is provided to interface to the machine's graphic engine. It furnishes the basic rendering functions for all the graphic primitives; polygons, lines, points, etc. It also provides a simple object management routine for storing, editing and rendering higher level objects. This latter capability is quite limited, however, and was not used in the development of the GDMS software. Of all the primitives available, the hardware is optimized for polygonal throughput.

2.5 *Application of Graphic Workstations in the Military*

Recent years have seen an increase in the application of synthetic environments to military tasks, particularly those involving military planning. Military planning systems cover a wide range of topics, from threat analysis, to battle management, to mission planning, to mission rehearsal.

Since the graphical database management system should be able to handle the requirements of these systems, it is beneficial to describe what several of these systems do.

2.5.1 Route Evaluation Module (REM) The primary purpose of REM is to allow Tactical Air Control Center Combat personnel to rapidly and effectively assess alternatives for unit tasking during Air Task Order development (2). The system allows users to select bases and targets, and then performs path optimizations for threat and fuel considerations. Although the system maintains the terrain information in three dimensions, it only provides a two-dimensional contour map image to the user. The software was developed in C.

2.5.2 3-D Stereoscopic Display System (Scenario) Scenario was developed to provide a demonstration of 3-D stereoscopic display techniques in the context of an operational Air Force task (14). The user's task was to select a set of waypoints from a friendly airfield to an enemy target, with minimal exposure to threats, such as enemy radar and contaminated areas. The system was developed in C.

2.5.3 Battle Management Visualization System The Battle Management Visualization System (BMVS) was developed at AFIT to establish the concept of previewing Air Tasking Orders (ATO) with a three-dimensional virtual environment (21). It presents users with a miniature battle environment complete with aircraft, threat regions, targets, etc. The software was developed using C.

2.6 Drawing Interfaces

The Database Generation System is similar to many drawing programs on the commercial market. Just like these commercial programs, DBGen's purpose is to manipulate graphical primitives on computer screens. To establish a set of guidelines for the design of DBGen, this section describes the general interface used in these various programs. For illustration, I will concentrate on DrawPerfect, a package from the WordPerfect Corporation.

DrawPerfect is a presentation graphics package for use on IBM PCs and compatibles. It is designed for mouse input through a screen interface. Figure 2 shows the screen layout for DrawPerfect. The icons to the left of the screen provide quick access to features of the program.

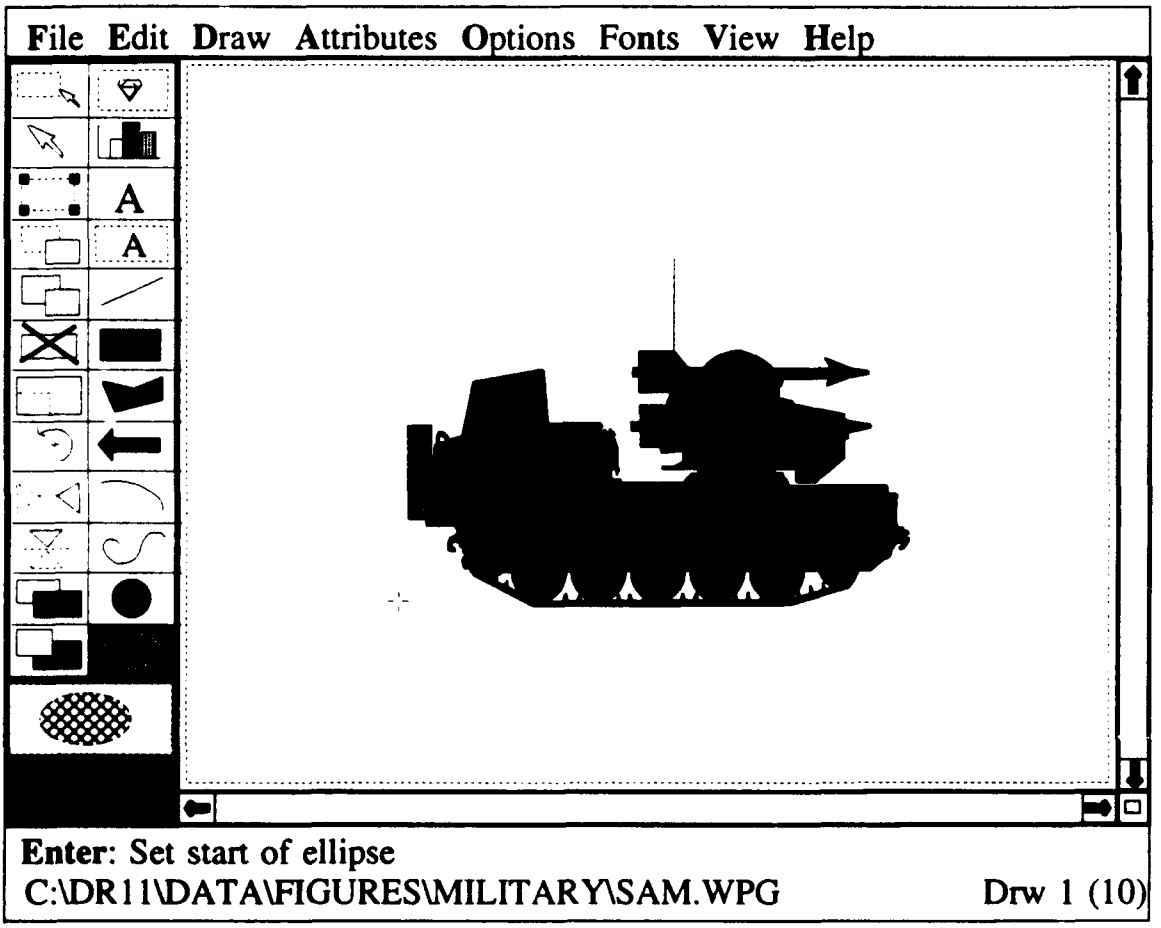


Figure 2. Screen layout for DrawPerfect presentation program.

Half of these icons select geometric primitives to insert in the drawing, like lines, polygons, boxes, circles, etc. The other half activate modes, like move, copy, and delete, for manipulating the objects already inserted in the drawing. The lower part of the screen provides status information to the user. The majority of the screen is taken up by the drawing area, with scroll bars to provide a means for moving the eyepoint. The final item to note is the menu bar at the top of the screen. This provides access to less frequently used commands, like the file management functions for saving and retrieving the objects created. The DBGen format is similar (See Chapter IV).

2.7 Summary

The information provided in this chapter formed the basic framework for the design and development of the Graphical Database Management System and the Database Generation System. The historical sections provide the reader with the background work in this area to put this effort in context. The PHIGS graphical standard influenced the general capability and interface of GDMS, whereas commercial drawing programs motivated the design of the layout and features in DBGen. The next two chapters describe the details of the design and implementation of GDMS and DBgen.

III. Graphical Database Management System (GDMS) Design

3.1 Overview

This chapter discusses the detailed design and implementation issues that were factors in the development of GDMS. It explains the rationale for the decisions made in the structuring of the software. The specific requirements for GDMS were developed by the author to address the graphical management and rendering needs of synthetic environment applications. The goal was to develop a library of routines to provide this capability. Figure 10 at the end of the chapter outlines the C++ classes that were constructed for this purpose.

The chapter is organized as follows. The first section describes general requirements and constraints, which apply across the whole effort. The second section is a detailed requirements and decisions section to cover each major area of the design. The third section deals with implementation specific details of the class structures making up the database system. The final section gives a summary.

Throughout this chapter, the term 'user' refers to an individual with access and knowledge of the file structures defining the database. This is usually the individual setting up the database files for the specific GDMS-based application.

3.2 General Requirements and Constraints

Various pressures existed to shape and constrain the design of this system. The Graphical Database Management System (GDMS) is part of an overall effort in the Synthetic Environments Laboratory (SEL) to develop a test bed for synthetic environment applications, such as the Object Oriented Flight Simulator, the synthetic environment Battle Management System and the synthetic environment Database Generation System. Figure 3 provides a general overview of how GDMS relates to the other SEL applications. The needs of these applications influenced the design and implementation of GDMS.

This section covers those requirements and constraints that had an impact on the entire design process. These include the geometry format, the language selection, the impact of prior software, and the overall timing requirements for a working system.

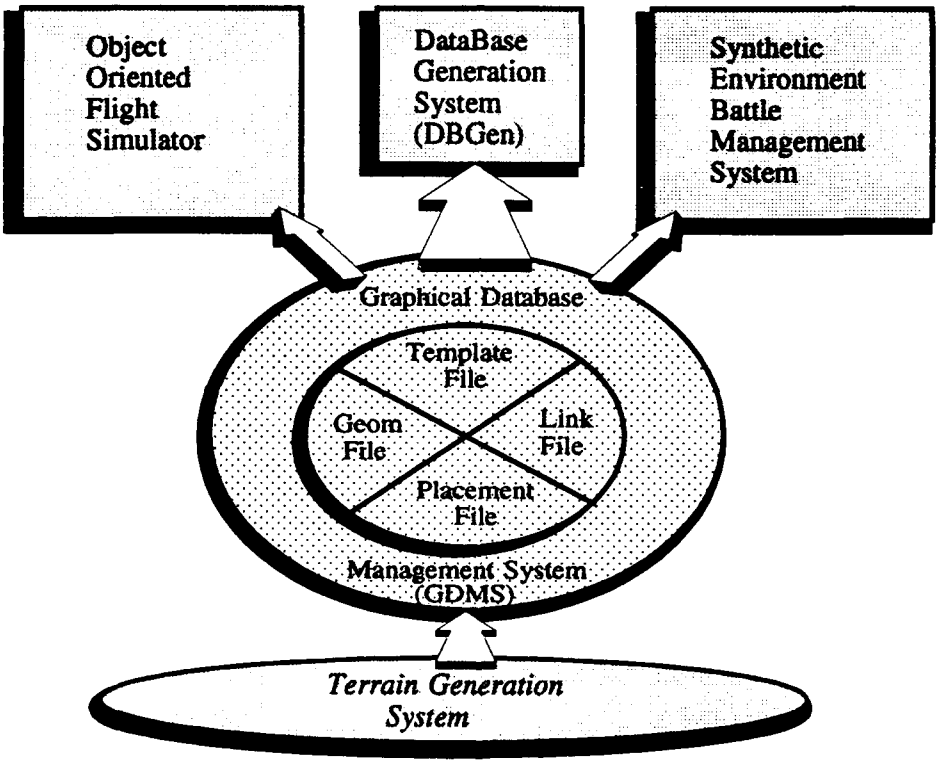


Figure 3. Synthetic Environments Laboratory Efforts.

3.2.1 Geometry Format Several geometries are available for modeling objects in three-space. Procedural models, fractals, grammar-based models, particle systems, spline-based models and polygonal models have all been used to represent a wide variety of things (7). The most common of these, and the easiest to implement, is the polygon model. Polygon manipulation is well understood in graphics community. The hardware platform, Silicon Graphics Iris 4D, is optimized for polygon throughput. Consequently, to limit the scope of this projects, I designed GDMS to handle polygon models only.

3.2.2 Language Selection There are five separate individuals involved this thesis cycle in the SEL effort. Much of our work is tied closely together. We needed a common language that would allow us to develop our modules separately, and then share the resulting code with relative ease. Since these modules would be intermixed freely, we needed the language to provide tight control over its procedures and variables. We also wanted the ability to extend the functionality of a module, without necessarily changing its code directly. We decided to use an Object Oriented Programming (OOP) language.

OOP is a method of programming that seeks to mimic the natural tendency we have to classify and abstract things (17). OOP languages are characterized by three main properties; encapsulation, inheritance and polymorphism. Encapsulation is the combining of data with dedicated functions to manipulate it (17). Inheritance is the building of new, derived classes that inherit the data and functions from one or more previously defined base classes, while possibly redefining or adding new data and actions (17). Polymorphism is the giving of one name or symbol to an action that is shared up and down a class hierarchy, with each class in the hierarchy implementing the action in a way appropriate to itself (17). OOP languages provide the functionality we were looking for. Captain Simpson's thesis effort, itself, is on the design of a flight simulator using an object oriented paradigm (19).

We chose C++ as the implementation language because it was a stable, reliable translator. As a translator, instead of a native compiler, the standard debugging tools for C were also available. Additionally, only C or C++ were available in the Graphics lab at AFIT.

All the mechanisms for the graphical database system were designed using object oriented constructs. The code is encapsulated and can be updated, replaced or extended as needed to

increase speed and/or efficiency.

3.2.3 Prior Software Impact The second major influence on the design was the existence of a specific set of modeling software. Over the years , AFIT has developed a series of tools to help in the modeling process. These tools manipulate files in a format called the AFIT Geometry File Format (GEOM) (6). This format contains the geometric description of the object being modeled. The GEOM files are ASCII based, using keywords to differentiate between entries. This allows modelers to read and manipulate the data directly. One does pay a price for this convenience, however. The ASCII files are much bigger than equivalent files using binary formats, and take more time to read in and parse them.

3.2.4 Development Timing The third major influence on the design was the timing constraint. As mentioned before, five thesis efforts for the 1991 cycle depended in some way on the database system. Getting a working version as quickly as possible was critical to these other efforts. Sufficient time was needed after initial development to integrate the database system into the other applications, making updates as required. For this reason, many of the algorithms and constructs in the design are as straightforward as possible. Future work could address more sophisticated techniques.

3.3 Detailed Design Requirements and Decisions

The graphical database management system is intended to provide all the functionality needed for the storage, manipulation and rendering of objects in a synthetic environment. I patterned the system after hierarchically based graphical models like that used in PHIGS. The software needed the following functionality: 1) to read geometric descriptions from disk into memory for display on the hardware, 2) to take these geometric descriptions and group them together to form more complex object descriptions, 3) to merge up to three complex object descriptions into a single, multi-resolution object, 4) to group any number of multi-resolution objects into a single complex multi-resolution object, 5) to place multiple instances of these complex multi-resolution objects on a uniform grid of terrain objects, and 6) to provide a viewing mechanism for interacting with the grid. Table 1 relates these requirements to the file and data structures used to address them.

Table 1. Requirements and constructs within GDMS

REQUIREMENTS	FILE STRUCTURE	CLASS STRUCTURE
Geometric Description	GEOM	GeomClass
Complex Object Descriptions	TEMPLATE	Template
Object Abstraction	LINK	Translator
Multi-Resolution	LINK	Translator/Placement
Object Placement	PLACEMENT	Placement/TGrid
Object Management	N/A	TGrid/PhigsList/ PhigsNode/Genlist
Viewing	N/A	View3D/World_Window

The remaining sections will follow this general break down of functional areas as described above. Specifically, these sections deal the requirements in the areas of geometric descriptions, templates, multi-resolution object abstraction, hierarchical construction, object placement, object management, and viewing. Each section will cover the storage, manipulation, and rendering requirements, as applicable.

Storage covers both the storage of data on disk and the data structures in memory. The information stored to disk must be sufficient to recreate the environment desired. The data stored in memory must be kept at a minimum, while providing rapid access.

Manipulation deals with the ability to specify initial and subsequent positions, orientations, scales and visible attributes for each object in the database. This applies to both static and dynamic objects.

Rendering is the display of the objects on target hardware. All hierarchical relationships must be preserved during this phase.

The following section are ordered in a "bottom up" approach, similar to the development cycle, starting with the low-level requirements and working up to the high-end ones.

3.3.1 Geometric Descriptions The first requirement was for the storage, retrieval, management and rendering of geometric descriptions. This formed the basis for the design of the C++ data structure class, GeomClass. (See Section 3.4.1). The GeomClass provides the functionality for reading geometric descriptions from disk and displaying them on the hardware.

The AFIT GEOM format was chosen as the storage format for geometric descriptions on disk. It has support for a variety of geometric constructs, like polygons and bi-cubic surfaces. However, in scoping my effort, I limited the allowable descriptions to polygonal models only. In most polygonal models, where a large portion of the polygons are connected, over half of the vertex and normal information is shared with other polygons. The GEOM disk file format reduces this duplication by separating the vertex information from the connectivity information. With the large amount of information contained in a geometric description, this is greatly needed. Unfortunately, in doing so, it also obscures the spacial relationship between any two polygons in the object. This makes it very difficult to construct triangular meshes, an efficient rendering technique on the Silicon Graphics Iris 4D workstations, or to do collision detection between objects on an individual polygon level. Nevertheless, in designing the data structure to hold the geometric information in memory, I decided that the memory savings were more critical to the design, so I patterned it after this format.

Similarly, the duplication of descriptions in memory had to be avoided to limit memory usage. GEOM descriptions are the basic building blocks for all the objects in the database. The same description can be used across multiple objects, or a number of times within the same object. A mechanism for managing these descriptions was required.

Another technique for the management of geometric data is the swapping of object descriptions in and out of main memory based on their visibility to the eyepoint. This is required for large gaming areas where the terrain information alone fills all the available memory in the hardware. Unfortunately, with our target applications centering on synthetic environment concepts using helmet mounted displays, the entire visible gaming area is just a head turn away. Additionally, the GEOM file format is not well suited for rapid retrieval from disk due to the size and parsing requirements of ASCII files. Thus, I decided to forego explicit memory swapping, and keep all geometric descriptions in memory at one time.

3.3.2 Templates The next step in building up a synthetic environment is the capability to hierarchically group individual component objects together to form complex objects, called templates, and to store and retrieve them from disk as single objects. Speed and flexibility were the key considerations in the design. These requirements led to the development of the Template class and TEMPLATE file format. (See Section 3.4.2 and Appendix A).

In the construction of complex objects, or templates, a hierarchical structure was needed for maximum flexibility. Hierarchical system are used in most standard graphical packages, like PHIGS. I decided to use the PHIGS model as a guide to these requirements.

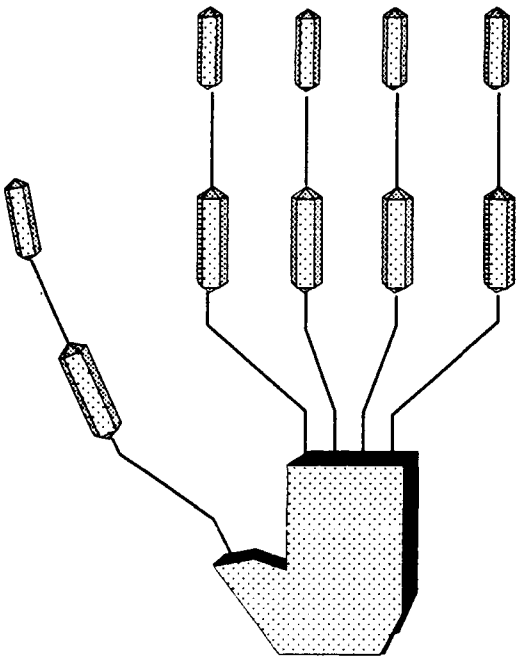


Figure 4. Pictorial representation of a HAND Template object.

In hierarchical systems, component objects can be positioned relative to a local coordinate frame, or to another component, called its parent. The structure is tree-like, in that only one parent can exist for any component, although a component can be the parent to any number of children. It is also like a list, in that there can be any number of parent nodes in the template, positioned independently.

A template structure is required to use GEOM files as its basic building blocks. The system is not intended for use on a modeling level, to manipulate individual polygons, but from a much higher level, to manipulate complete objects. A special TEMPLATE file is used to define the components and their relationships. Along these lines, the number of components in a given object, is fixed in the TEMPLATE file, and can only be changed there. Thus, templates are static objects, although their components can move.

A hand template is an example of this kind of object (See Figure 4). A hand can be broken down into a palm and five fingers, where each finger is modeled by only two finger segments for simplicity. The palm would be one GEOM description, and a finger segment would be the other. From these, every component in the hand can be constructed. The first finger segments would be defined relative to the palm, and the second finger segments relative to the first. Of course, each component must be independently scaled and rotated, a required capability of the Template class. The result is a single object, the hand, with potentially movable parts. The Template format can also be used to group multiple, independent objects together into a single object, like an airfield.

3.3.3 Multiple Resolution Object Abstraction The next requirement in the GDMS is to take several template descriptions of the same object, at various levels of detail, and combine them into a single multi-resolution object. The definition of these objects is handled through the Translator class and the LINK file format (See Section 3.5.7 and Appendix A).

Multiple levels of detail can speed up rendering by controlling the polygonal throughput to the hardware [See Olson (15)]. To provide a suitable mix between modeling, storage, and rendering requirements, I limited the number of resolution levels to three, corresponding to low, medium and high, with a fourth option being a bounding volume around the object. Three levels limit the demand on the modeler, the memory storage, the disk storage, and the algorithms, while providing flexibility in the system test their effectiveness [See Olson (15)].

The specific description files, GEOM and/or TEMPLATE, making up an object's three levels of detail must be specified in some way. An application must know which description files to load in memory, and how to link them to object instantiations in the database. There were two goals for this portion of the design. The first was to isolate the programmer from the task of positioning each level of resolution individually. The second was to minimize the complexities of linking the

multiple resolutions together. I came up with a system using three layers of abstraction to meeting these requirements; object categories, type ID numbers and resolution levels.

An object category is a label representing a group of objects with similar functions, like aircraft, or buildings. A type ID number is an identifier that corresponds to a particular object defined in one of the categories. A resolution level is used to indicate the particular description object for the low, medium or high level of detail.

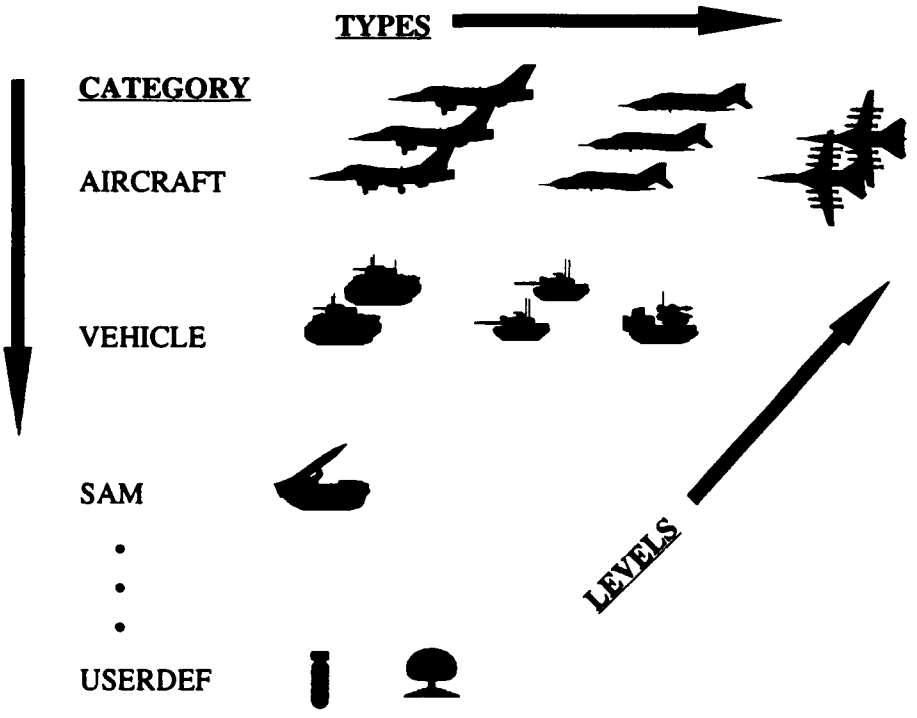


Figure 5. Object abstraction in GDMS.

A special LINK file is used to define these relationships in the GDMS. The file does two things. First, it establishes an object category in the database, and second, it links each resolution file to the appropriate object type defined. The Translator class must match up the resolution levels for each object, regardless of the order they are defined in. Specifying all three levels of resolution for any given object type is not required. Figure 5 illustrates the concept.

A programmer identifies an object definition by specifying its object category and Type

ID number. Rarely will an application have need to refer to individual object resolutions, but if necessary, the capability is provided. The programmer needs to know nothing of the actual file names used to define the object, but these are also provided on request.

An advantage of this type of object abstraction is that users, by switching the LINK file, can switch the images, without redoing all the object placements. This allows a battle manager to manipulate a database using symbolic icons, and a flight simulator to use it with 'real' images.

3.3.4 Hierarchical Construction In keeping with a hierarchical model, complex, multi-resolution objects must also be able to be grouped into even more complex objects. Just like the templates described above, a PHIGS-like structure is used to accomplish this. Unlike the templates, though, this level of construction is totally dynamic. Objects can be dynamically altered by adding or deleting components from the structure at run-time. The GenList and Phigs classes were developed to manage this complexity. (See Sections 3.5.2 and 3.5.3).

The structures were designed using linked-list algorithms. Linked-list mechanisms provide the dynamic element needed. The interface follows the PHIGS metaphor of opening and closing elements. This hierarchical capability should be inherited by any structure needing it. All objects placed in a synthetic environment using GDMS are derived from the PhigsNode, which is derived from the GenListNode (Refer to Figure 10).

To see how this might work, imagine an aircraft in a flight simulator application. The list capability allows an aircraft to be loaded with as many missiles or bombs as desired, without having to supply separate TEMPLATE files for each possible configuration. The individual objects are created and attached to the aircraft as appropriate. To "fire" a missile, the application simply detaches it from the aircraft and reinserts it as a separate object in the database.

3.3.5 Object Placement The construction of our abstract, multi-resolution, hierarchical object has now reached the point where we can place it within the synthetic environment. The Placement class handles this requirement (See Section 3.4.6). This type of object is called a Placement object.

The actual Template descriptions for a given object type are defined at initialization. To place, size, orient and color each occurrence of an object independent of the others, a unique

Placement object must exist for each occurrence.

To illustrate this, suppose one created two Placement objects of a SAM battery, where the SAM missiles could rotate relative to its base through Template calls. If the same Template descriptions were used in the object, rather than separate copies, then both missile racks would rotate whenever one was moved. To avoid this, separate copies of each Template resolution for each Placement object must be created. Terrain is an exception to this rule. Since terrain objects only exist once in the database, their Template descriptions can be used directly.

A Placement object is at the pinnacle of the object hierarchy. It must manage all the manipulation and rendering requirements for an individual object. These include control over the dynamic movement of components, switching between levels of detail, resolution blending, coloring, and coarse clipping options. The resolution blending and coarse clipping algorithms were developed by Olson (15).

3.3.6 Object Management The next step in the construction of the database management system is to take all the Placement objects generated, both terrain and other, and store them in some type of structure for efficient access and rendering. The storage, in this case, includes both the data structures, for active use, and disk files, for permanent storage and retrieval. The TGrid, or Terrain Grid, class is responsible for these requirements (See Section 3.4.7).

In terrain-based environments, for which the GDMS is designed, a grid-like structure works well for segmenting the database. Although a uniform grid is not required, it does simplify the implementation, and is so designed. Individual terrain blocks are created, of a uniform size, and linked together to form the gaming area. If the grid size is defined to match that of the terrain blocks, then objects placed on the terrain naturally fall in the proper grid block. This leaves a much smaller set to search for a particular item, so long as you know which block it is in. In most visual applications, this is the case.

Another advantage of this approach is that it provides another level of coarse clipping. A given terrain block might have several objects grouped with it. The block can be checked against the viewpoint in the same manner as that of Placement objects. Large areas of the database can thus be excluded from the pipeline to increase throughput.

Of course, the whole point of a database management system is to be able to manipulate the objects in it, and save the changes. In designing the disk storage routines, I included the ability to separate out the terrain information from the rest of the objects. This is useful in a research environment, where one might want several different object files for any given terrain file. The mechanism for permanent storage uses ASCII based files, called PLACEMENT files. (See Appendix A).

3.3.7 Viewing The final function of a graphical database management system is to provide the means to view the synthetic environment. To interact visually with the graphical database, two items must be accomplished; a viewing volume/transformation must be defined and it must be attached to a specific window on the display.

A viewing transformation takes objects within an imaginary volume in world space and transforms their three-dimensional descriptions into screen coordinates for display on a viewing device. There are two basic types of viewing volumes, orthographic and perspective.

An orthographic projection defines a viewing space using parallel rays. For non- skewed volumes, the sides of the volume are perpendicular to the viewing plane. An object far away appears the same as an object close up, all other factors being the same. This means that a position defined in an orthographic window in screen space will correlate linearly in two dimensions with world space coordinates. By aligning the viewing volume with the world coordinate axes, a programmer can determine an object's position directly from the screen presentation. For instance, by using an overhead view of the terrain, one can specify an exact location in X and Y by selecting a point on the screen, and applying offset and scaling factors. A program can use this to place or retrieve objects from the database. Of course, another mechanism must be used to specify the Z position.

A perspective representation defines a viewing space as a pyramid, with the eyepoint residing at the apex. Objects further away will appear smaller than objects close up. The distance away from the eyepoint and the angle of ray divergence, or field- of-view (FOV), determine the relative image size of an object. These views are useful for understanding spacial relationships between objects. A perspective view is used for out-of-the-cockpit views in flight simulators, or any views where the movement of the eyepoint or 3-D effects are needed.

To allow for multiple viewports into the graphical database, a way was needed to set up and control each one. This capability had to handle both orthographic and perspective viewing volumes, and allow for dynamic positioning in world space. The View3D class was designed for this (See Section 3.4.8).

It was also necessary to link a viewing volumetransformation to a window on screen, and be able to render the database in this window on command. A generic capability was needed which would leave the manipulation of the terrain grid entirely to the programmer, but which would handle all the requirements for rendering and viewport manipulation. In addition, the system also needed the ability to render a separate list of objects for use at the discretion of the programmer. This list might include other aircraft or missiles in a flight simulator, or include objects being manipulated in a battle management system. The goal was to relieve the using programmer from as many of the direct viewport manipulations as possible by providing a higher level interface. The World_Window class was designed to do this (See Section 3.4.9).

3.3.8 Summary In all, the requirements outlined above provide the basic capability needed for a graphical database management system. I discussed the flow of structures from the individual GeomClass component, to the complex Template object, to the multi- resolution object, to the hierarchical Placement object to the Terrain Grid, to the viewing requirements. Each level builds upon, or uses the levels below it. A similar relationship exists in the data files, from GEOM file, to TEMPLATE file, to LINK file to PLACEMENT file.

The next section discusses the implementation issues involved in actually creating the GDMS. The discussion is centered around the construction of the C++ classes used in the implementation.

3.4 ImplementationClass Construction

Each of the requirements discussed above are implemented through one or more C++ classes described below. The classes tend to align with the required areas in GDMS. Specifically, geometric descriptions are handled by the GeomClass class, templates by the Template class, multi-resolution object abstraction by the Translator class, hierarchical construction by the GenListNode and PhigsList classes, object placement by the Placement class, object management by the TGrid

class, and viewing by the View3d and World_Window classes. Figure 6 illustrates the general flow of the classes and data files, from the top level down.

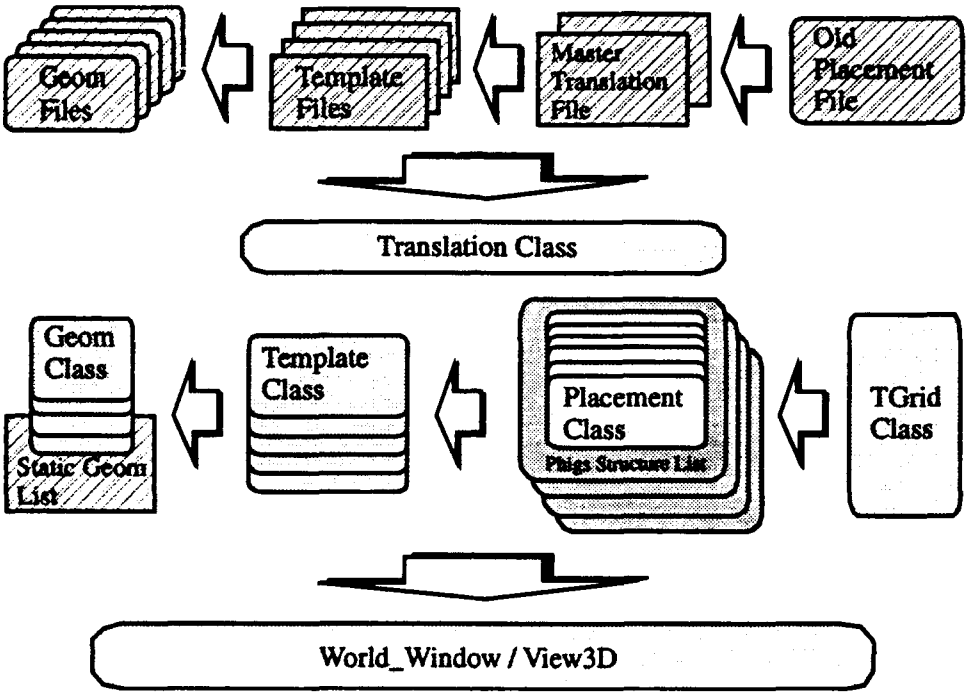


Figure 6. Data and Class alignments in GDMS.

3.4.1 GeomClass Class The GeomClass is the class structure responsible for managing the low level geometric descriptions. It holds all the information needed for rendering a geometric object. In addition to the information on the polygonal structure, it maintains attribute information, face normals, and statistical information, like number of vertices, polygons, and attributes. I had two major concerns in the implementation this class. I needed to minimize the amount of memory used, while maximizing the rendering speed. I used arrays for both speed and compactness.

The structure is set up to minimize memory for objects with shared vertices. It does this by maintaining arrays for all the vertices, normals, and vertex colors (if applicable), which are then indexed through a separate connectivity array for polygons. The connectivity array specifies how the entries in the vertex and normal arrays connect to form each polygon. In this way, points that would normally be duplicated in two or more polygons, can be stored in a single entry in memory.

This assumes, of course, that the modeler took advantage of this feature within the GEOM file, and did not duplicate the points.

The GeomClass is by far the largest user of memory within the GDMS. I needed a mechanism to prevent duplication of the GEOM files used during the construction of various objects within the database. I developed a specialized class constructor, MakeNewGeom, which is used instead of the normal constructor. This method crosschecks the GEOM filename against a static list of previously loaded GEOM files to insure it had not already been loaded. If a match is found, the pointer to the existing description is returned, otherwise, the file is loaded and its name is added to the list. A given GeomClass can be removed entirely from memory through a special RemoveGeom method, which also updates the static list.

For speed in rendering, I use simple loops with array indexing for access to the information. I also sort the polygon array by attribute index to help minimize the amount of attribute switching required during the rendering process. It is the modeler's responsibility to insure duplicate attributes are not specified in the GEOM file, as this reduces efficiency.

To make the generation of bounding volumes more efficient in classes higher up the chain, a min/max value along each axes is computed and saved. When queried for its bounds, the object simply returns these values, since they can not be changed without changing the model.

The remainder of this section is devoted to miscellaneous information related to the functionality of this class.

The rendering method is tied closely to the specific hardware of the SGI 4D. All the parameters describing an image must be passed to the hardware using library calls. The lighting model requires predefined material types, describing surface attributes for each different type of polygon. The GEOM files contain this information, but it is difficult to compare attributes across files to avoid duplication. The material types have to be defined on a GEOM file by GEOM file basis.

To define any of these attributes, the SGI 4D graphics system must be initialized. This forces the programmer to open some type of window, which initializes the graphics, before reading in any GEOM file information. It is the programmer's responsibility to insure this occurs. A system crash will result if this rule is not followed.

During loading of the first file, nine standard material types and three transparency patterns are predefined to give programmers a basic set of colors and patterns for special rendering. Their ID numbers range from one to nine, and will overwrite any definitions using these labels. In addition, the polygon attributes pick up at ten and continue until all are defined. This leaves only those ID numbers after the total number of attributes defined for use by the programmer. Of course, the preset values can be overwritten once the first file has been read.

3.4.2 Template Class The Template class is responsible for managing the complex template objects made up of GeomClass components. This class holds the information relating the object descriptions to the components, and the relative orientation of each. The class is constructed so any number of components can be defined for any number of GeomClass objects. The design goals stress both flexibility of object construction and speed of rendering.

To accommodate the hierarchical structure required for complex objects, a data structure containing pointers to child and sibling nodes, along with a modeling matrix and description pointer, was created. The pointers, in this case, are actually indices into component and GeomClass arrays. The Template class uses array structures to minimize access speed. As a consequence, the physical number of components is fixed upon initialization. Neither addition nor deletion of components is allowed, although they can be deactivated (meaning flagged so they would not be rendered).

The transformation matrices are 4x4 floating point arrays used to translate, scale, and rotate a given component relative to its parent. The transformations are applied in pre-multiplication order, which means they are formed with commands given in the opposite order from the intended application. Since templates are in local modeling coordinates, the Placement class is used to position the object in world coordinates.

All object resolutions in the database are Template objects. Templates are constructed from either an individual GEOM file, or a special TEMPLATE file, which uses multiple GEOM files. The TEMPLATE format is described in Appendix A.

The way the database is set up, a series of Template instances for each defined object are maintained by the Translator class, and copied into an instantiation of a Placement object when requested. The Placement object can then manipulate its own Templates without effecting the

other objects in the database. An overloaded equality operator is provided to accommodate the copying process.

Although speed was a goal of the design, rendering for a template object was implemented recursively due to the nature of the hierarchical structure and ease of programming. Performance might be increased slightly by restructuring this method in an iterative approach.

Finally, like the GeomClass objects above, a bounding volume is computed and maintained which encloses each component in the object. This is passed to the Placement class for coarse clipping purposes.

3.4.3 Translator Class The Translator class provides a level of abstraction in the database system to separate object descriptions from object identification. It allows programmers to work with multi-resolution objects without tracking the individual filenames for each description. The primary goal in the implementation was concerned with the programming interface to the other classes.

The Translator class loads and stores a Template object for each resolution of each object defined for the synthetic environment. A particular object, with its associated templates, is identified through an object category and type number. The templates correspond to individual resolution levels.

An object category is a special, predefined label designating a group of similar objects. For this effort, I defined a particular set of object categories for applications in battle management and flight simulation. These include TERRAIN, RADIOTWR, WATERTWR, HANGAR, SAM, AAGUN, AIRCRAFT, VEHICLE, BUILDING, MISSILE, and USERDEF. The list covers most of the objects found in these type of applications, with USERDEF being a catch all. Additional categories can be added only by changing the code. This involves adding or changing entries to the enumerated type 'objectType', and revising the methods for each class that uses it.

The type ID number is an integer corresponding to a particular item in an object category. Users define the maximum number of object types for each object category that will be defined in the file. The type numbers must be less than this maximum number, since they are simply indices into an object array. The maximum number is used to allocate the array needed in memory.

Resolution indicators provide the final link to individual descriptions used for low, medium and high levels of detail for any given object. The actual filename to a specific description is passed to the database management system by indicating the object category, type ID number and level of resolution. The Translator class places this information within a two-dimensional array for easy access.

In addition to the Templates defined for each object, the Translator class also holds a short name for descriptive purposes. A label of 'USERDEF, Type 1' is not very helpful for identifying what the object actually looks like. The short name string, which is user definable, can provide this type of information.

All this information is defined off-line in a special LINK file defined by the user. The LINK file format is described in Appendix A. It is the user's responsibility to set up this file to define all the objects which will be used in the database.

3.4.4 GenListNode Class The GenListNode class provides the links and methods to add to, delete from, and iterate through a generalized list. This class is used to provide the basis for the PHIGS-like hierarchical capability of Placement objects. Arrays are unsuitable for this due to the requirement for dynamic insertion and deletion of objects in the database. A main goal of this implementation was adaptability.

A generalized list is a special list structure that allows any list node to hold either a pointer to another list, or data on a particular item. For the GenListNode class, the model is modified slightly and allows both data and pointers to coexist in a node at the same time. This was done to increase the speed of access through all data nodes, and to simplify the implementation.

The GenListNode structure provides all the capability of a tree structure, with the versatility of a doubly-linked list. The generalized list structure is quite versatile, and can be applied to a wide range of applications. By using the inheritance and polymorphism of C++, the data structure can be extended to track virtually any kind of data. In the GDMS, the GenListNode class is extended to the PhigsNode class, which is then extended to the Placement class. Thus, the Placement objects get the full functionality of a generalized list, without duplicating any of the code.

The GenListNode uses a child-sibling metaphor for the structure of the list. Any node can

be a parent, sibling or child of another node. A node can have either zero or one parents, but can have an indefinite number of children or siblings. The sibling of a child is also a child. See Figure 7.

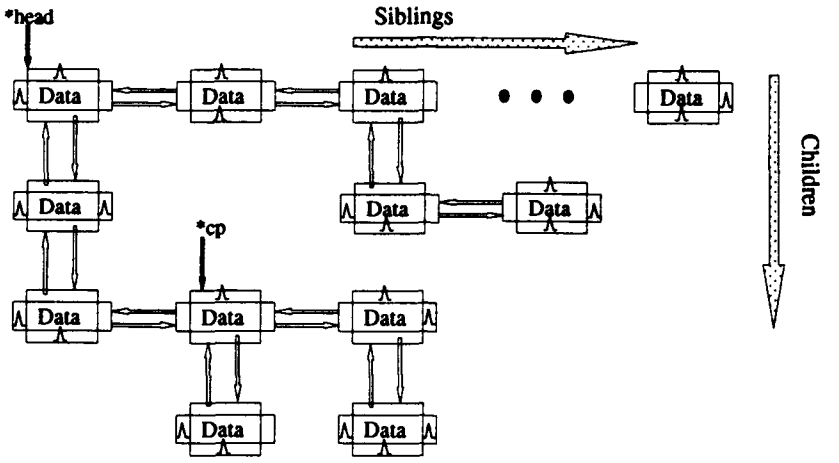


Figure 7. Generalized list structure within the PhigsList Class.

The hierarchical nature of the list allows for easy tail recursive methods for adding, deleting or walking through the list. Nodes can be removed non-destructively if desired, for manipulation and reinsertion, perhaps at a different level. The back-pointing links in the node structure simplify the insertion and deletion algorithms. They also allow for the implementation of algorithms where recursion is unacceptable.

A special iterator is provided to walk through the list, while performing a programmer defined action. By taking advantage of the polymorphic nature of inherited objects, specific actions on derived objects are possible. The iteration scheme follows a modified depth first algorithm, where the action is performed first on the current node before descending to the next node. This is to allow for proper pre-multiplication of matrices during object rendering.

3.4.5 PhigsNode/PhigsList Classes The purpose of these classes is to provide the list management functions needed to implement a PHIGS-like structure using a generalized list. The PhigsNode class simply extends the GenListNode class to include a node ID number and a PHIGS status flag. The PhigsList class provides PHIGS-like calls for managing the generalized list of nodes. The goal of this implementation was to isolate the programmer from as many of the internal details of the PHIGS hierarchy as possible.

These two classes work hand in hand with GenListNode class to provide a PHIGS-like interface. The PhigsList maintains the head of the list, as well as a current pointer into the list. Nodes are manipulated relative to the current pointer through PHIGS-like calls of Open, Close, Create, Attach and Detach. Global methods are provided to clear the list, locate the current position, return the head or perform an action. The PhigsList can manage any type of node derived from the PhigsNode class.

As an example of how the PhigsList class is used, I will go through the steps of building up a placement object in the database (See Figure 8). The example will be of an F-16 with two missiles attached to the wings. This assumes that the F-16 and missile descriptions have already been loaded through the LINK file.

The programmer must first instantiate the placement object for the F-16 using 'new' operator on the Placement class. The resulting pointer to the object is then loaded into the database with a unique ID number using the 'Create' command. The unique object number applies only on the level assigned (among siblings) and is needed to retrieve the object at a later time. The current pointer now refers to the F-16 node, which is considered 'OPEN'.

Now, a missile can be created in the same way. After positioning it relative to the F-16 with Placement class methods, it is loaded on the aircraft with another 'Create' call. This time, however, a 'Close' call is made to close the child and move the current pointer back the parent F-16. The same steps apply for the second missile.

With the missiles loaded, a final 'Close' call will close the F-16 object and return the current pointer to NULL status. Additional missiles or bombs can be added later by first opening the F-16 object with the 'Open' command, and following the same procedure. In this way, a hierarchical object list like that shown in Figure 9 can be made.

```
        // OList is ptr to Translator class
        // Figs is a ptr to a PhigsList class
        // Object is a ptr for a Placement class

                        // ******F16******
        // 2 below is a reference to the F16 type description
Object = new Placement (AIRCRAFT, 2, OList);
Object->SetPosition (1000, -2500, 10000);

        // 1 below indicates the ID number for later reference
Figs->Create (1, Object);                    // loads F16 on list

                        // ******Missile 1******
Object = new Placement (MISSILE, 1, OList);
pushmatrix ();                               // uses library calls to
loadmatrix (idmat);                          // construct matrix which
translate (-10, 7.5, -.5);                   // which positions missile
rotate ('z', 90.0);                          // relative to F16 parent
getmatrix (m);
popmatrix ();
Object->LoadPositionMatrix (m);
Figs->Create (1, Object);                    // loads M1 as child of F16
Figs->Close ();                              // closes M1

                        // ******Missile 2******
Object = new Placement (MISSILE, 1, OList);
pushmatrix ();
loadmatrix (idmat);
translate (-10, -7.5, -.5);
rotate ('z', 90.0);
getmatrix (m);
popmatrix ();
Object->LoadPositionMatrix (m);
Figs->Create (2, Object);                    // loads M2 as child of F16
Figs->Close ();                              // closes M2

Figs->Close ();                              // closes F16
```

Figure 8. Example of method calls for the constructing F-16.

For maximum flexibility and speed, the class provides methods to make node pointers directly available to the programmer. They are provided to allow programmers to make calls of the given object class directly. The programmer has the responsibility not to corrupt the internal linkages. For instance, if a programmer gets a node from the list directly, and then deletes it himself without first detaching it, the system might crash. This could occur because the PhigsList pointers to the head and current node might now point to unallocated memory. Actions like this should be accomplished with the provided PhigsList methods.

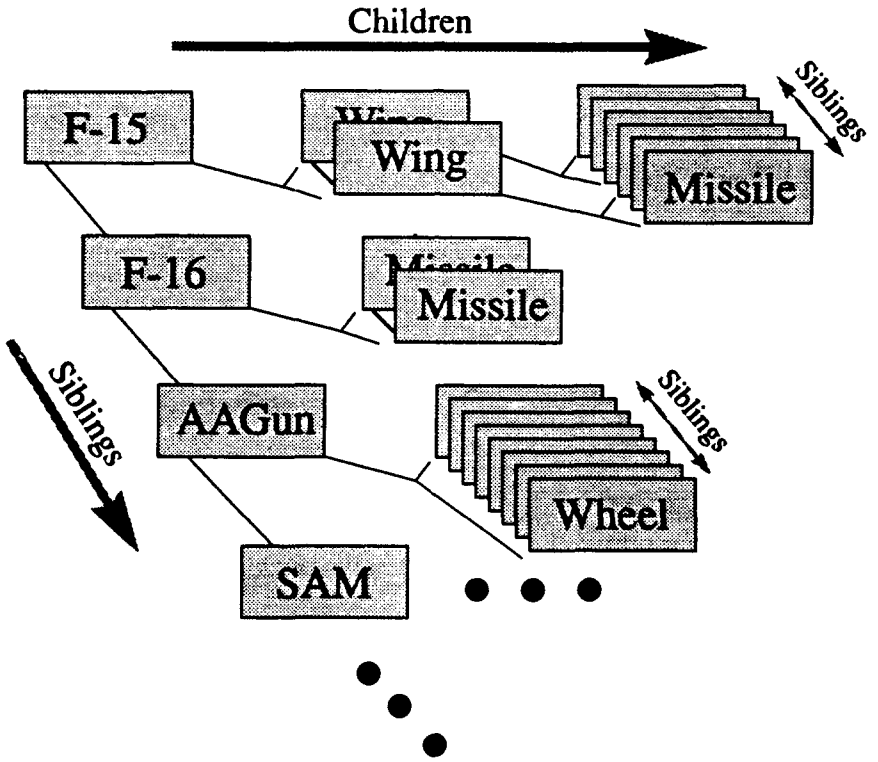


Figure 9. Possible PhigsList construction of Placement objects.

3.4.6 Placement Class The Placement class provides the structure and methods to define multi- resolution, hierarchically based objects, or placement objects, to load in a synthetic environment. Placement objects are classified into one of two possible groups, static or dynamic. Static objects consist of the TERRAIN and the objects on it that do not move. Dynamic objects are items like aircraft and missiles that do. Static objects will far outnumber the dynamic objects in most

scenarios. The only difference between the two is that the programmer must maintain the pointers to the dynamic objects separately to be able to manipulate the objects efficiently.

The major components of a Placement structure include three Template objects, a transformation matrix, resolution switching information and a bounding volume. The design of this class is centered about the efficient management and rendering of the multiple levels of detail.

There are two methods for creating a Placement class object. The first is to pass in an object category and type identifier to the constructor, followed by up to three Template pointers corresponding to the three image resolutions. These Templates can be generated independently or retrieved singly from the Translator class (See Section 3.4.3). For objects already defined in the database, additional levels of detail can be added or replaced with the LogTemplate method. Of course, there is still a maximum of three resolutions per object.

The second approach is to pass the address to the Translator class, itself, instead of the three Template pointers, and let the constructor extract the available levels of detail on its own. This eases the programming interface for creating objects directly from the Translator class.

The Placement class is a derived type of PhigsNode, so multiple objects can be linked and managed by the PhigsList. As such, it extends the hierarchy beyond the Template class, to include multiple resolution components, while allowing for dynamic insertion and deletion of components. Care must be taken during dynamic insertions to place the child component relative to the parent, as opposed to the world coordinate system. See the F-16 and missile example in Section 3.4.5. By the same token, objects dynamically detached must be converted back to world coordinates to allow for proper placement on their own in the database. Consequently, there is more overhead involved in this process.

Each Placement object, whether parent or child, has a transformation matrix to position, rotate and scale its three dimensional description. Just as in the Template class, this matrix is a 4x4 floating point array. The difference is in how it is used. The transformation matrix is applied to whichever resolution is active in the component at the time of its rendering. In addition, for the base node, or parent object, the matrix is applied relative to the world coordinate system, instead of the parent's local coordinate system. This is how entire objects are place or moved in the synthetic environment, through the transformation matrix of the base node.

Before rendering a placement object, the class must determine which resolution to choose of those available. There are two ways to do this. The first is the simplest, where the programmer just specifies the resolution desired. For objects that do not have the particular resolution defined, the rendering method switches to the next most detailed resolution available. The second method involves selecting resolutions automatically based on the distance from the eyepoint. The distance calculated is from the origin of the object's local coordinate system, not necessarily from its centroid. The effect of this is that for objects modeled with the origin off-center, the resolutions will switch at different apparent distances from the user, depending on the direction of approach. The rendering method compares this distance with user definable switching distances to determine which resolution to render. These switching distances define two hysterisis bands for transitioning between levels of detail. The hysterisis prevents cyclic image toggling when objects are very near the specified distances.

The Placement class has two ways of setting the switching distances, manually and computationally. The first method allows users to override the default settings in the system. The second sets them based on a single field-of-view (FOV) parameter. This method uses the FOV to determine what percentage of a window the object takes up at certain distances. Or, inversely, given the percentages desired, it finds the distances to which they correspond, which are the switching distances. These percentages were determined empirically using a 'that looks about right' approach and are coded directly into the program.

The switching distances can be used to simply toggle between resolutions, or used in a more advanced technique of blending (See Olson (15)).

Bounding volume information is the last block of data maintained in the class on each object. Bounds are set by querying the three resolution templates for their individual volumes to establish an overall volume. This volume is used for collision detection and coarse clipping purposes. The volume vertices are kept in world coordinates to speed up the coarse clipping algorithm. This was done to optimize the handling of static objects in the database. A consequence of this, however, is that for dynamic objects, the bounding volume has to be recomputed whenever the object has moved.

I have already mentioned many of the factors impacting the rendering process, now let us

look at the different rendering options. Rendering is activated through a suite of four overloaded Render methods. Each involves a recursive procedure due to the hierarchical design of the class objects. One set of Render methods provides coarse clipping of all objects. The other set relegates clipping of each polygon entirely to the hardware. Within either set, one method activates automatic resolution switching, the other allows the programmer to set a specific resolution, including an option to render just the bounding volume.

As a final service, a recursive method can save the structure of the Placement objects to disk. The recursive nature of this method allows a single call from the head node object in a PhigsList to save the entire contents of the list. An open file handle must be supplied, along with a mode for saving objects. The programmer can select ALL objects, ALL but TERRAIN, or TERRAIN only objects for saving. This allows the programmer to separate the terrain and object placements into different files, or merge them all into one.

3.4.7 TGrid Class The TGrid class, or Terrain Grid, maintains the information and methods needed to populate and manipulate a two-dimensional grid of static objects. This includes both terrain and cultural features. The class was designed to meet the requirements for database object management (See Section 3.3.6). The thrust of the implementation was on efficient access and rendering of the entire static database.

The grid structure consists of a dynamically allocated 2-D array of grid nodes spanning the gaming area. Each node contain a PhigsList and a bounding box. The PhigsList holds the objects, while the bounding box surrounds them all for coarse clipping purposes. The TGrid is intended to hold only Placement objects.

The grid is defined in world coordinates on the X-Y plane, using the minimum and maximum values, and a spacing parameter. The terrain must be modeled with the Z coordinate up to fall properly on the grid. The array dimensions are computed using the grid size parameter, to form a uniform grid. Dividing the length of each side by the grid size determines the number of nodes in that direction. The (0,0) position of the 2-D array is always the lower left corner of the grid (South-West). The grid can be defined anywhere within the XY-plane. A method is provided to convert world coordinates into uniform X,Y grid coordinates.

All objects are controlled by the grid node in the lower, left corner of the grid square they fall

on, with the exception of objects placed outside the grid. These are controlled by an appropriate exterior grid node. The process is entirely automatic. The programmer simply sets the object's position matrix to the desired location in world coordinates, and adds it to the grid. The grid method assigns it to the appropriate grid node. This method uses the location of the object's local origin, not necessarily its centroid. The programmer must also provide a unique ID number for the object to be able to retrieve it from the grid, so a special GetUniqueID method is provided.

The TGrid class is designed to allow the programmers to interact with the database in a direct manner. The methods are visually based in that they expect the application to provide the general position of the objects requiring access. A reference position, or a designated area in world coordinates must be passed to the methods providing access into the grid.

One method, GetNearestObjectBeyond, searches for the nearest object beyond a specified radius, relative to the reference position, and returns a single object pointer. Only the current grid square, and the immediate squares surrounding it are checked. This method does not actually remove the object found from the grid, but it does give the programmer access to it for functions like selection highlighting, etc. The method is designed to walk the radius out until the proper object has been selected, where it can then be detached and manipulated.

The other method, GetObjectsInArea, gathers all objects within a specified area and puts them in a separate PhigsList for return to the application. Only those grid squares intersecting the area of interest are searched. Unlike the method above, this method actually removes the objects from the grid. Area selection is intended for large scale manipulation, such as area deletes, copies, or moves.

Both methods exclude terrain objects from the selection searches. This approach is taken since terrain objects are tied to a specific location in world space, and must remain fixed to match with the other terrain blocks.

As an interface concern, to minimize the requirements on the programmer, and maintain an effective coarse clipping algorithm, the appropriate grid square bounding volume is automatically recomputed whenever an object is added or removed. The total grid min and max altitudes are updated at this time, also.

To provide a permanent storage and retrieval mechanism, this class contains the methods for reading and writing object placement information to disk. The PLACEMENT file format is defined in Appendix A. The methods are LoadDatabaseFile and SaveDatabaseFile.

The LoadDatabaseFile method allows programmers to take the object descriptions from the file, and either append them to the current grid or create a new one. The programmer can specify the type of objects to be loaded: ALL, ALL but TERRAIN, or TERRAIN only. This method is used exclusively for loading static objects into the grid, although the code can easily be adapted for loading dynamic objects.

The SaveDatabaseFile method is used to save the grid objects to disk. It also has the same options for specifying the object types as the load method. The method can be made to append its objects to the end of a file, or overwrite a new file. The combined capability gives the programmer considerable flexibility.

Four rendering options exist, corresponding to the Placement object rendering options (See Section 3.4.6). These are integrated within the World_Window class to simply the selection options for the programmer. Similarly, a display option to draw the grid lines at some specified altitude is also integrated with the World_Window class. Of the options available, the most time efficient rendering mode uses coarse clipping and automatic resolution selection.

3.4.8 View3D Class The View3D class is responsible for the structures and methods used to define and manipulate a viewing volume, which in turn defines the viewing transformation. This class is used in conjunction with the World_Window class to provide a viewport into the synthetic environment for the graphical database management system.

This class was originally designed by Olson (15) during development of a specialized viewing application for GEOM files. Major design changes in the operation and structure of this class were made to adapt it to the GDMS. The class was restructured to follow more of a classic object oriented approach, verses a procedural approach. The design concentrated on efficiency and functionality.

The View3D class uses the 3-D viewing model described in Chapter 6 of (7). All the parameters can be set and retrieved by the programmer. Both perspective and orthographic

projections are supported. The SGI architecture splits the viewing transformation defined in Foley et al (7) into two pieces, a static projection matrix and a viewing matrix. It uses the double matrix mode to specify the volume constraints separately from the positioning information. This is to increase the speed of rendering since the projection matrix generally remains unchanged (18).

The View3D calls fall into two categories: those that set up the viewing volume, or projection matrix, and those that position and orient it in world space, the viewing matrix.

The first set defines size and shape of the viewing volume. Methods exist to define the viewing window, set the projection reference point (PRP), establish the front and back clipping planes, and designate the type of projection to use, orthographic or perspective. The ComputeProjection method takes this information and computes and loads a projection matrix for use with the SGI. It is the programmer's responsibility to call the ComputeProjection method after the volume is defined or changed, but before any objects are rendered.

The second set directs the orientation of the volume in world space. Methods exist to set the view reference point (VRP), the up vector (VUP), and the view plane normal (VPN). The Compute3Dview method takes this information, computes the viewing matrix, and loads it on the hardware. The Compute3Dview method is called from within the World_Window class prior to each rendering of the database. Programmers that use this class, but do not use the World_Window class to attach the eyepoint to a window on screen, must insure the Compute3Dview method is called for each frame update.

This class also maintains special parameters and methods for the rapid coarse clipping of bounding volumes. This ties in with the rendering options in both the TGrid and Placement classes. The IsVisible method accepts an entire bounding box, instead of just a single point, for efficiency reasons, and returns its visibility with respect to the viewing volume. The algorithm was developed by Olson (15).

3.4.9 World_Window Class The World_Window class contains the structures and methods needed to bind a terrain grid and an object list, to a window and viewing volume. This class is derived from both the View3D class, described above, and the Window class, developed by Simpson (19). The class was designed to handle all the rendering and viewport manipulations

needed in most applications. The implementation effort concentrated on the class flexibility and programming interface.

The World_Window class manages the position and orientation of the viewing volume with respect to the window on the screen, and renders the database as appropriate. The methods break down into two major areas; rendering of the images, and control of the viewing volume.

The class keeps two pointers for rendering of the database; one to a terrain grid, and one to a PhigsList. The terrain grid holds all static objects in a database. The PhigsList holds all the dynamic objects. It is the programmer's responsibility to load and manage these items. They are kept for rendering purposes only. Methods exist to specify the rendering options available with these items.

Programmers can also select a number of display options, such as overlay information on the view volume, axes display and text, grid line drawing, and lighting model.

The remaining set of methods deal with manipulating the viewing volume relative to the images displayed in the window. These include Pan, MoveIn, Swivel, Roll, Zoom, and Rotate-AboutOrigin methods. These methods compute all the parameters needed for the View3D class to perform the desired action. Programmers may also manipulate the viewing volume directly through inheritance using View3D calls.

A programmer updates the images on screen by simply calling the Redraw_Window method, after changing any of the parameters. This, in turn, calls the Draw_Window method which makes the appropriate calls to the hardware, and applicable classes.

3.5 Summary

The nine classes described above work together to provide a flexible, yet powerful, graphical database management system. The classes are diagramed in Figure 10. GDMS was designed primarily for synthetic environment applications, like mission planning, battle management or flight simulation, but can also be used for many other types of graphical database applications. GDMS gives a programmer a set of classes that provide a high level interface for storing, manipulating and rendering three-dimensional graphical objects on an Iris 4D workstation. The next chapter

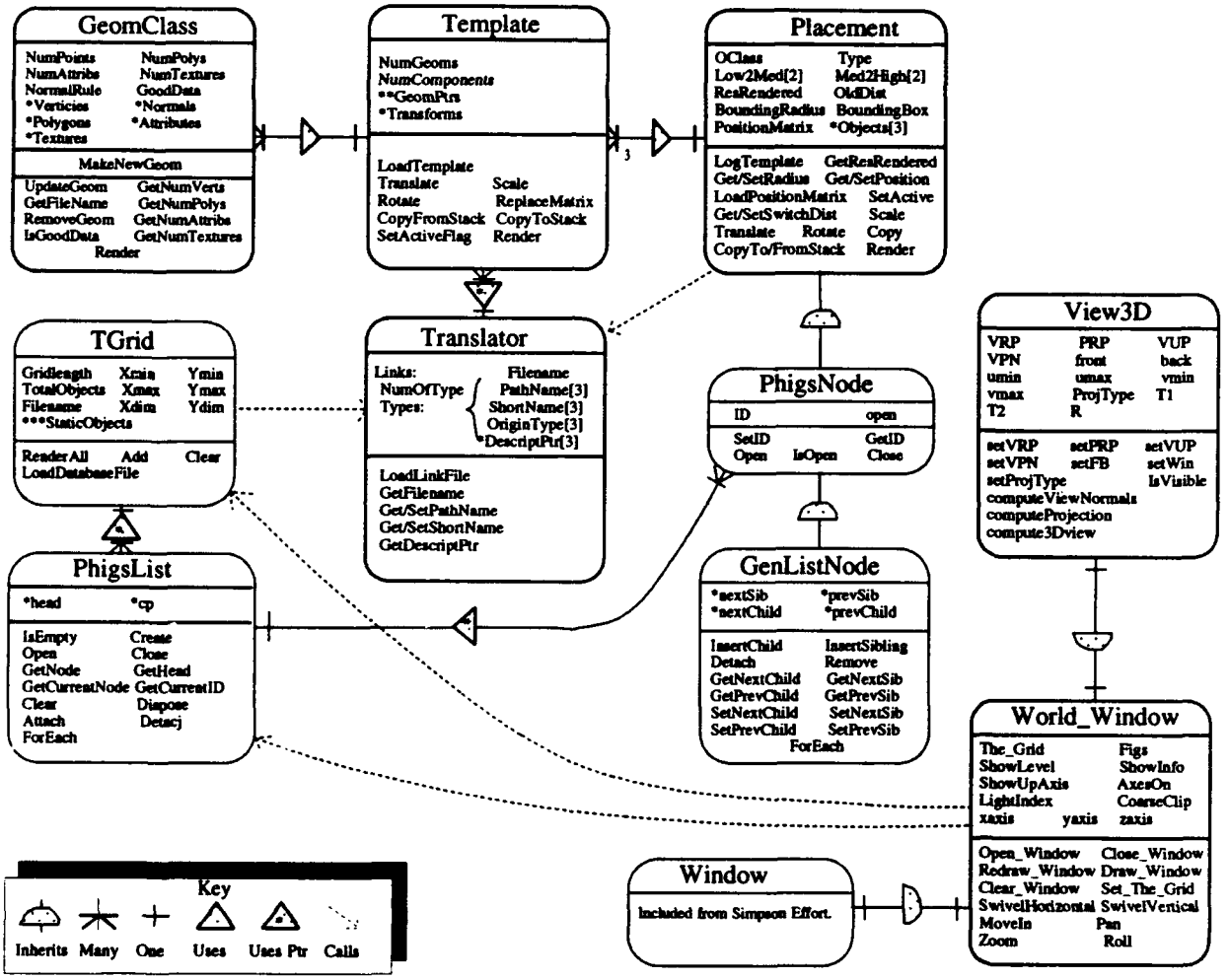


Figure 10. Class diagram for GDMS.

discusses the design and implementation of an application using GDMS, a synthetic environment database generation system.

IV. The Database Generation System (DBGen)

4.1 Overview

This chapter discusses the detailed design and implementation issues effecting the development of the Database Generation System (DBGen). It reviews the decisions made in the structuring of this application. The specific requirements for DBGen were developed by the author to test GDMS, and provide a capable database generation system. The goal was to develop a tool to allow users to take previously defined models and place and manipulate them on a terrain grid interactively. DBGen is not considered a modeler, because it does not create or manipulate the primitives making up individual object descriptions. The modeling is left to other programs. This program takes their output and combines them into synthetic environments. A driver program, GDMS, and a few extra classes were needed to do this. Figure 11 shows all the classes incorporated into DBGen. The asterisked items are part of the GDMS library.

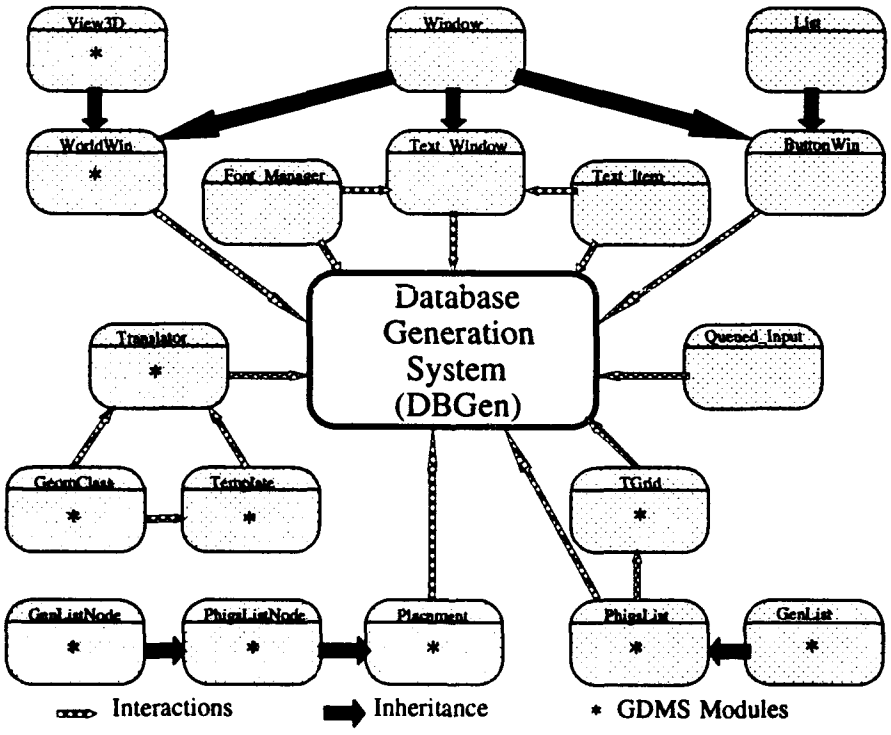


Figure 11. Classes used within DBGen.

DBGen was designed to use as many of the features of the graphical database management system as reasonable. Its implementation fulfilled three primary functions. First, it tested the majority of the methods used in the GDMS software, and forced the refinement of the class interfaces for general use. Second, it provided a platform to develop and test many of the special algorithms needed for the Battle Management System and the Object Oriented Flight Simulator. Third, it provided the means to generate synthetic environments for use in SEL research.

Figure 12 shows the general flow of data between applications in the SEL. The Terrain Construction Tool generates the GDMS terrain files from Digital Terrain Elevation Data (DTED). The Database Generation System uses these files, along with additional geometric descriptions, to populate the terrain with objects, and outputs the resulting GDMS object files. The Flight Simulator and Battle Management System then use these files to build up their gaming area for their particular application. Each application uses GDMS to manage the graphical database.

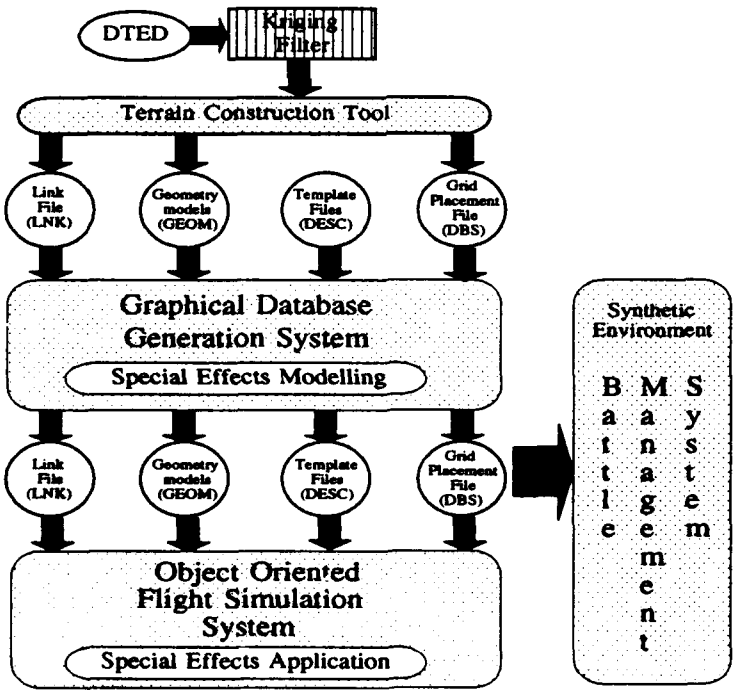


Figure 12. Overview of data flow between SEL applications.

This chapter has three sections. The first section discusses the general requirements desired

in a synthetic environment Database Generation System. The second section covers some of the decisions made in the design and implementation. The last section outlines the specific class and driver structures developed during the implementation.

4.2 General Requirements

The requirements for the Database Generation System (DBGen) drove many of the design decisions in the graphical database management system. By the same token, the other efforts in the SEL impacted the design of DBGen. As these applications developed their own requirements, the database management system was updated/modified to accommodate them, after which the changes were incorporated into DBGen, if reasonable. As a general goal, the Database Generation System was designed to incorporate as many of the basic features required by both the Flight Simulator and the Battle Management System as possible. Of course, it also needed to meet the requirements unique to its own application. These requirements broke into four broad categories; file management, object editing, environment control, and user interface.

4.2.1 File Management All SEL applications are terrain based. The terrain files used within this application, (GEOM, LINK and PLACEMENT), are generated by Duckett's system (5). During updates or additions to a given terrain block, his system overwrites these files with the new information. To preclude the loss of any overlaid static object information placed by DBGen, the terrain information is separated from the rest of the database objects, which drove the capability in GDMS (See Section 3.3.8). The terrain files can be updated at will, changing resolution or total area, without affecting the objects already in place. Multiple object files can also be developed to give different object densities/positioning over the same set of terrain, without duplicating all the terrain information on disk.

4.2.2 Object Editing The Database Generation System is primarily a visually oriented program, as opposed to a text-based program. A user manipulates objects in the database by observing the change in the images on screen due to inputs from the mouse or keyboard. Most settings are changed through on-screen buttons or pop-up menus. The format for DBGen is similar to many of the common presentation programs available on personal computers, such as Corel Draw (TM), Draw Perfect (TM), etc. These programs manipulate elements like points, lines,

boxes, or even entire objects on screen to form complex 2-D images. DBGen manipulates complex three-dimensional, multi-resolution models, to form 3-D synthetic environments.

Both these applications create and manipulate what is essentially a database of objects. In each, the basic editing requirements are the same. The program must provide a set of building blocks to be used for database construction. It must allow the user to add or delete these blocks to the database at will. It must also allow the user to move, orient or resize these blocks either during or after insertion. For convenience, the user should also be able to copy or mirror existing blocks in the database.

Inherent in many of these features is the requirement to be able to identify and retrieve objects from the database, which demands two capabilities. The first is a picking option, where the user can select, or deselect, an individual object in the database. This is usually relative to some type of pointing reference, like a mouse cursor. Multiple objects can be specified in this manner. The second is an area selection option, where the user can specify a region on the screen, and all objects falling within the boundaries of this region would be selected. This block of objects would then be treated as a single object for manipulation purposes.

Finally, the program must be able to store the final product to disk and retrieve it again at a later date for further modifications. The files generated in DBGen are the same files used and manipulated in the other SEL applications.

All these requirements are strongly tied to the capabilities of the GDMS, upon which this application is built. As the requirements were implemented, additional methods and restructuring were often required of the database management system. Thus, the design and development of both DBGen and the GDMS proceeded iteratively.

The information used to define the database is provided in external files. These include the LINK and PLACEMENT files for the terrain, the master LINK file for object descriptions, the PLACEMENT file for positioning information, and all the TEMPLATE and GEOM files modeling the objects themselves. Filenames for the LINK and PLACEMENT files used for DBGen are required to have 'lnk' and 'dbs' extensions, respectively. All these files are ASCII based. DBGen's primary function is to create or modify the object PLACEMENT file, which it does through the graphical interface. To give the user some type of editing capability in the other

areas, DBGen provides a pop up shell to 'vi'. The program is suspended until the 'vi' session is completed, after which shell window is automatically closed. Changes will only take effect if the file is specifically reloaded by the user.

4.2.3 Environment Control Environment control allows a user to manipulate parameters global to the whole program environment. Categories include parameters that effect how the program interacts with the database and how the user interacts with the program.

Figure 13. Terrain viewed at MORNING setting in DBGen.

Interactions with the database include lighting control and rendering options. In a lighting model, direction, color and intensity have a significant impact on the images formed from three-dimensional databases. For terrain based environments, this coincides with the time of day (See Figures 13 and 14). By varying the direction to the light source, one can simulate the position of the 'sun'. Variations in color and intensity also contribute to the effect. The light is assumed to be at infinity to reduce rendering time. To give the user a look at the terrain under different lighting conditions, DBGen needed some means of varying the time of day. It uses four settings, morning,

noon, afternoon and twilight, as characteristic times. Suitable parameters are passed to the lighting model when an option is selected.

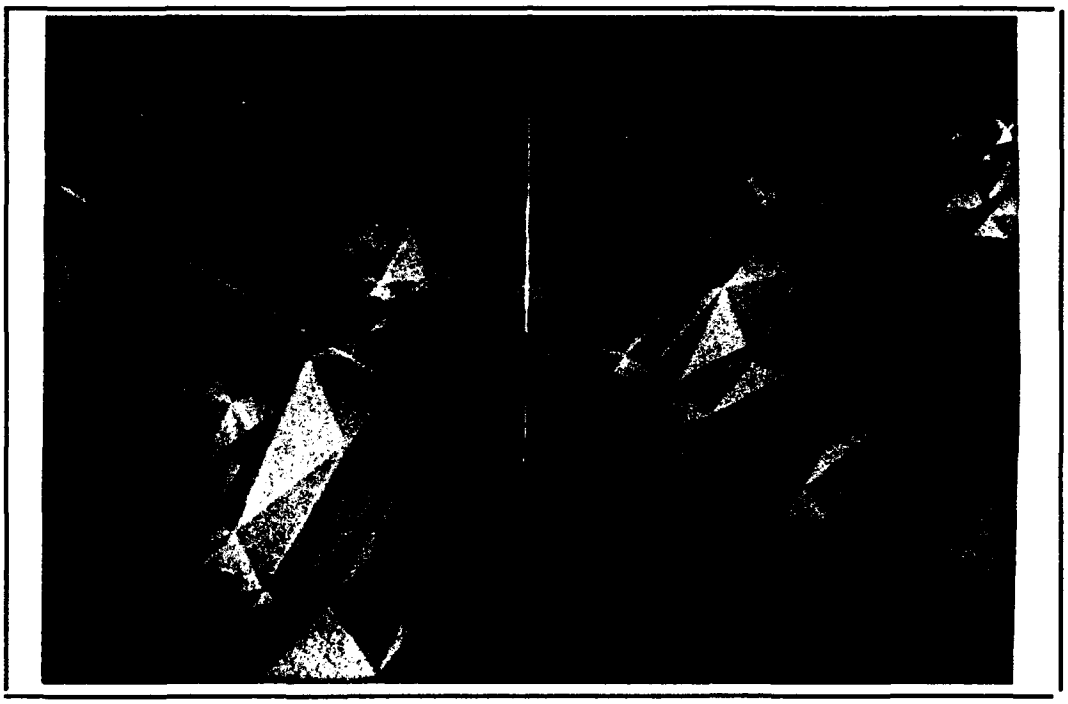


Figure 14. Terrain viewed at AFTERNOON setting in DBGen.

For rendering purposes, the user has several more options. The database management system provides four methods for rendering objects (See Section 3.3.6). The user needs the ability to select these different modes. They include auto or manual resolution, with or without coarse clipping. This allows the user to gather measures of the effectiveness of each mode, for modeling and analysis purposes. Similarly, the blending feature between resolutions in auto mode must be user selectable. A popup menu is provided to accomplish this.

The second category of user interaction with the program includes control of viewing modes, view volumes, and information displays.

DBGen has two primary viewing modes, an object placement mode and a fly-through mode. Given these two capabilities, a method for switching between them is obviously required. In addition, they are sufficiently different in purpose to warrant separate window formats (See Figures 15, 16, and 17).

The first mode deals with object placement. Placing three-dimensional objects onto three-dimensional terrain in a graphical environment is difficult to accomplish, especially without cues like stereopsis, shadows, sound, pressure, and many other inputs we use in the real world. DBGen needed a purely visual mechanism for orienting objects with the terrain, one better than a single perspective window could provide. To do this, three separate views, one on each axis of the object being manipulated, looking back at the object, is provided (See Figure 15). This helps the user control the placement much better, even on sloping terrain or when a particular viewpoint is blocked by other items in the database.

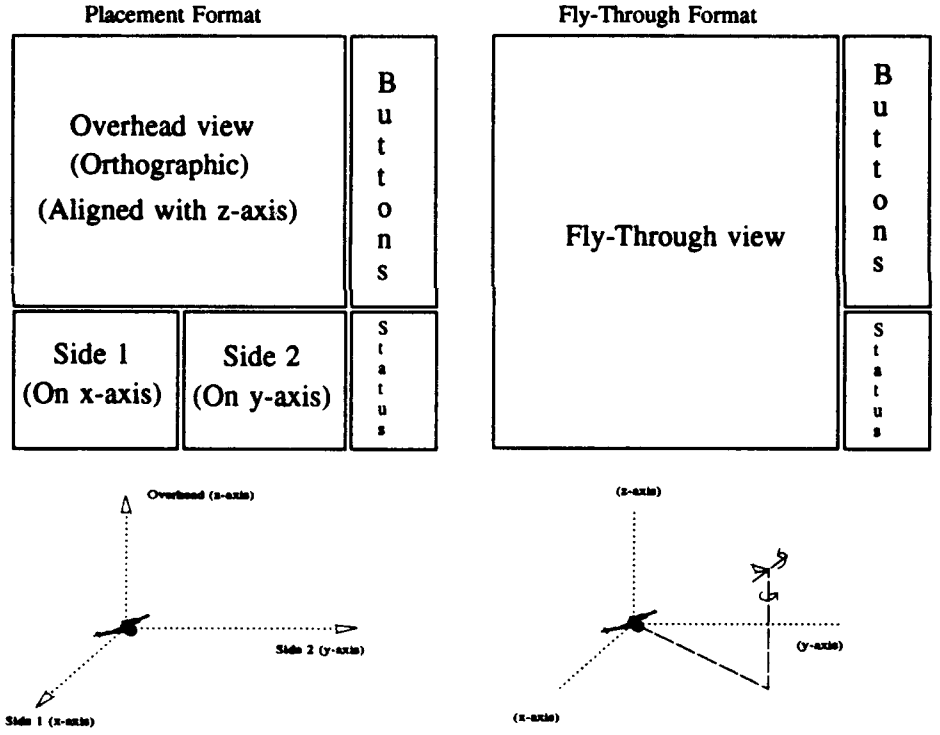


Figure 15. Window Layouts with corresponding views in DBGen.

The second mode is the fly-through mode. The user needs one large screen, with an attitude reference indicator for positional awareness. This mode gives the user direct feedback on what a flight simulator view of the database would be like (See Figure 15). It also provides feedback on the update rates possible for various scene contents.

Each window used in positioning and fly-through modes is generated from the World_Window class of GDMS. This gives each window its own viewport into the synthetic environment, which can be manipulated to provide various views of the database. DBGen requires a means to do this, as well as the option to reset the variables default values.

Related to this is the display of information detailing the current settings for each window. This information is displayed directly in the window, through a World_Window function. The user needs the option to turn off this display to avoid clutter.

Other display options include object axis and grid square lines. A special status window was used to present information global the entire application. A Text_Window [See Simpson (19)] was used for this purpose.

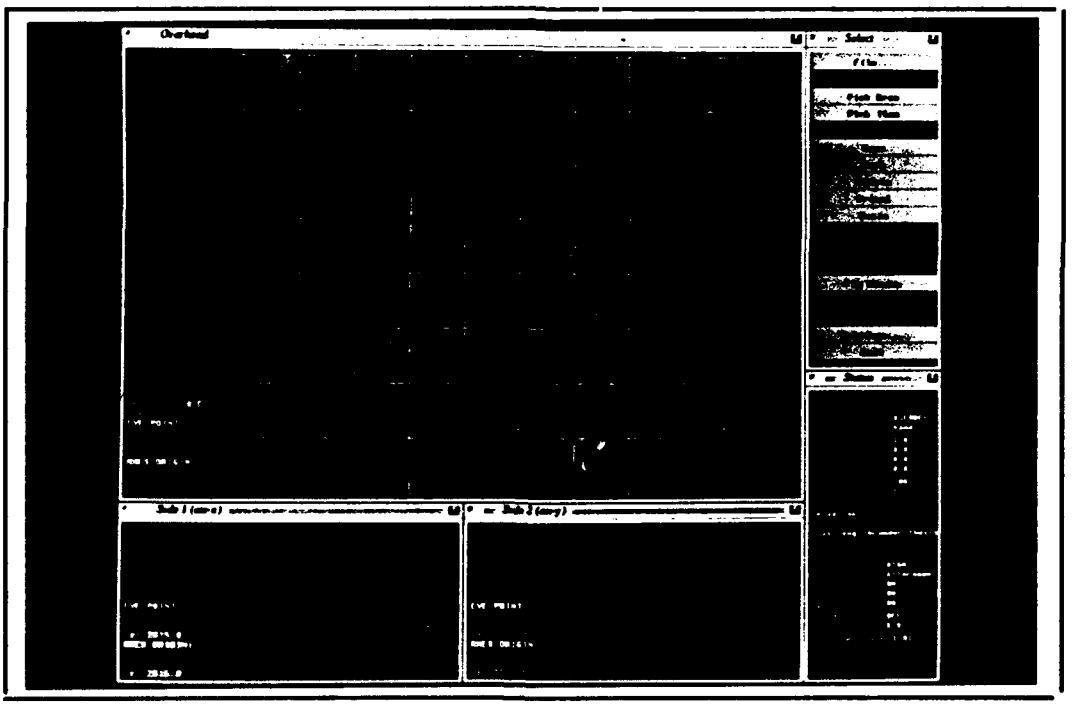


Figure 16. Picture of Placement window format in DBGen.

The window layouts for the two different formats are shown in Figure 16 and 17.

4.2.4 User Interface A user interface permits the selection of program functions. How intuitive that interaction is relates to the usability of the program. A user interface, alone, can

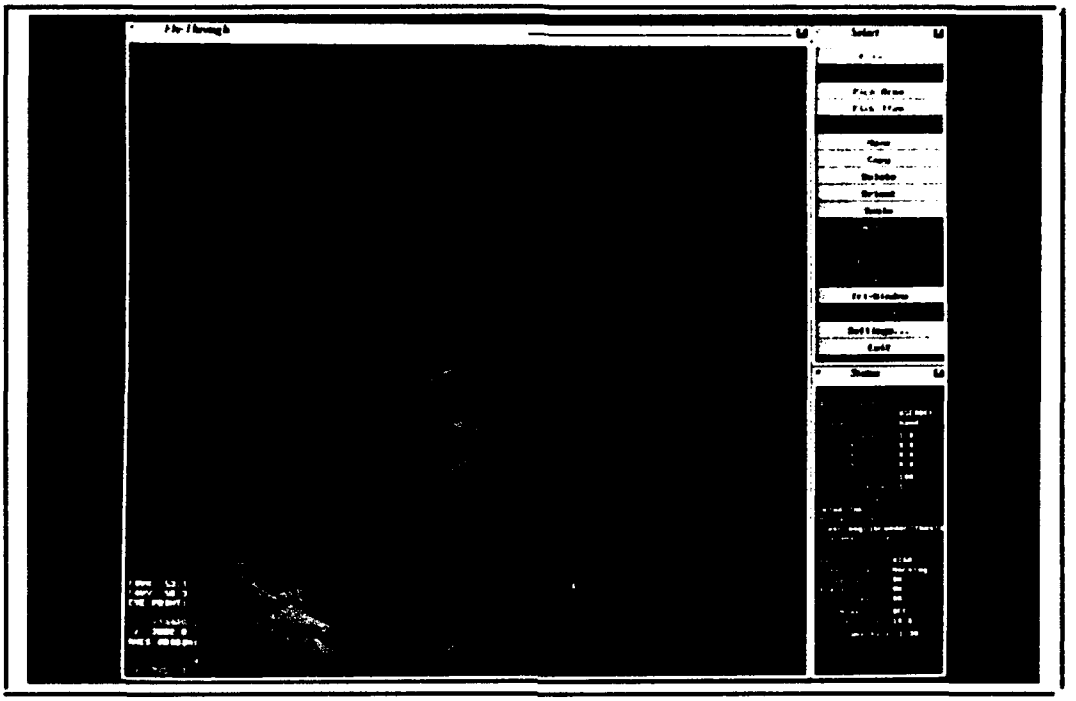


Figure 17. Picture of Fly-Through window format in DBGen.

often leave a user cursing or praising a program. As a guide, I returned to commercial drawing programs.

Successful commercial programs generally use a menu bar and a series of icons to activate commands or select options. Icons give the user immediate access to the most commonly used commands. Menu bars minimize the clutter on the screen, while allowing access to secondary or configuration type commands. A button bar window can be implemented to serve both purposes. Buttons can activate a command option directly, switch a mode, or call up a menu for more detailed options. The button color can be used to indicate the status of a particular button option. Additionally, the button title should be changeable to toggle modes. See Section 4.4.1 concerning the ButtonWindow class.

It is important to be consistent throughout the user interface. The user should only have to learn one methodology, and then be able to use it to access the entire program. That methodology should be as simple as the application allows. The hardware system influences what user interface options are supportable in a program. The SGI 4D provides a tightly integrated mechanism for

linking inputs, from either the supplied three-button mouse or keyboard, to a particular window on the screen. Other input types include joysticks, spaceballs, or datagloves, but their control mechanism is through separate RS-232 ports. The structures for controlling these alternate input devices were still in development by other thesis students during the implementation of DBGen. Consequently, they are excluded from the design. DBGen was implemented using a predominately screen based, mouse driven approach, with keyboard augmentation when necessary or convenient.

4.3 Design and Implementation

In the following sections, I will cover the major elements in the design of DBGen. I will describe object selection, manipulation, type identification, insertion and interface issues. Where applicable, I will indicate the C++ classes that were developed to attain the design goals. These classes are described in Section 4.4.

4.3.1 Object Selection Objects must be selected from the database before any type of manipulation can occur. The GDMS TGrid class provides this capability. Whether objects are individually selected, or gathered from a specified region, they must be designated in some way for identification by the user. Each object uses a color index parameter for this purpose. Within DBGen, an object can only have three states; normal, selected, and active. An object is in a normal state before it has been selected. In this state, objects are rendered using whatever colors are specified in the GEOM files. The selected state occurs when the user selects an object in the database, which places it on a selection list. The selection list is rendered in red (See Figure 18).

The user can select as many objects as are in the database, excluding terrain. Terrain is not selectable for modification because it must remain fixed with respect to all the other terrain blocks in the database. Once the user has selected the objects desired for manipulation, he changes their state to active. This is indicated by changing their color to green (See Figure 19). The selection list now becomes the active list, and no other objects may be added until the objects are reinserted in the database, or the list is returned to the selection state. For a list in the selection state, switching editing modes from one state to another will automatically change the list to active.

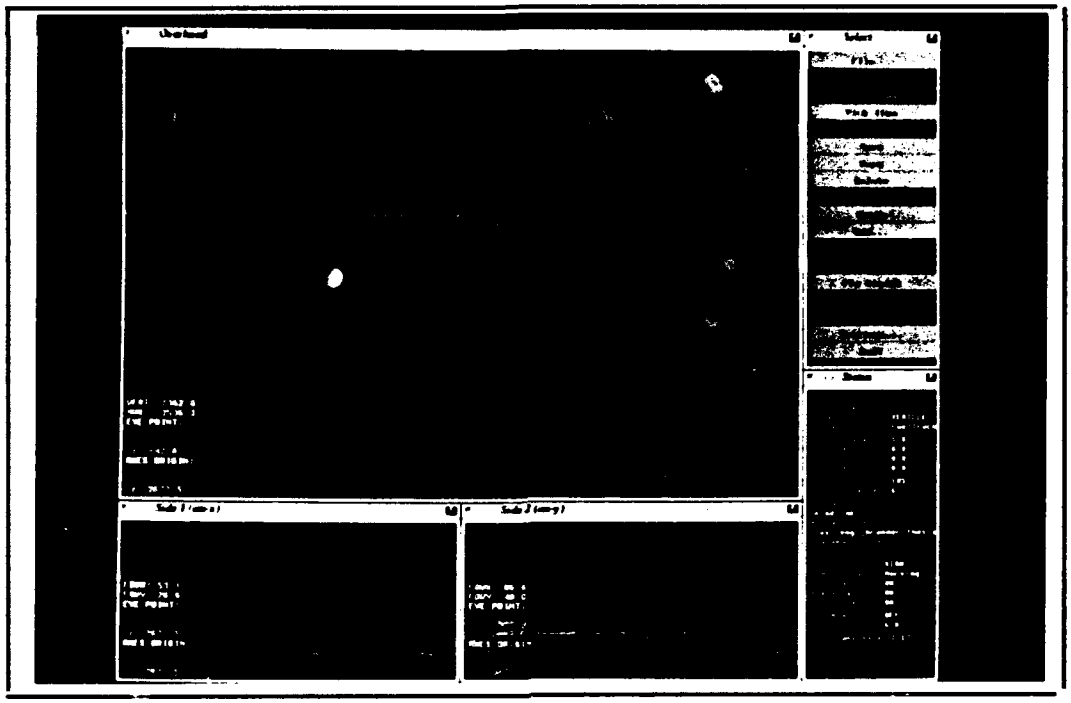


Figure 18. Picture of objects selected in the database.

4.3.2 Object Manipulation The list of active objects is treated as a single complex object for manipulation purposes. A center point in three dimensions is computed by averaging the positions of all the objects on the list. Rotations, scales, and translations are all relative to this center point. This allows the user to drag an entire collection of objects to another position and reorient it. A user can quickly build up multiple groupings, like airports or townships, using the copy function and the drag feature. Individual object manipulation is a special case with a single object on the list. In the active state, users can switch between editing modes, like scale, move, orient, or copy. Selecting add mode, however, will cause all items on the list to be reinserted automatically. The list must be cleared to allow individual objects to be added and oriented on the terrain. After the user has completed any desired manipulations of the active list, he can reinsert the objects in the database, or return the list to the selection state. The mechanics of selection and activation are discussed in the User's Manual (See Appendix B).

4.3.3 Object Type Identification Creating an object for insertion in the database is somewhat different than direct manipulation of an existing object. It requires the user to have explicit

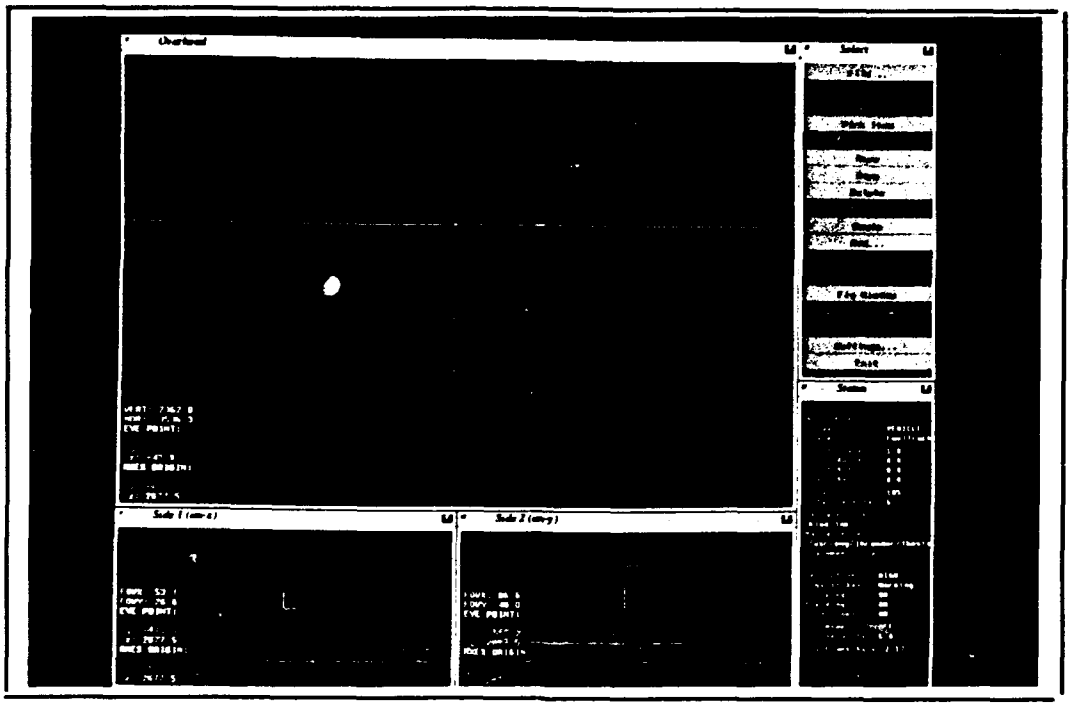


Figure 19. Picture of objects activated for modification in the database.

knowledge of the available objects for insertion. In DBGen, these objects are not defined until run-time. This is unlike most commercial drawing programs where the primitives are known before hand. This requires DBGen to build up an object description interface dynamically, from the master link file. It queries GDMS about the object definitions available, and builds up a multi-level menu tree to walk through available selections, from category to type ID to level of resolution, as appropriate. By linking this menu structure to a visible button on screen, the user gains quick access and a familiar menu format for selecting objects for insertion in the database (See Figures 20 and 21). The same methodology is used throughout the Button Window Panel, popping up menu trees for selection by the user as needed. Buttons with menu popups are indicated with trailing ellipsis (...).

4.3.4 Object Insertion Objects are inserted in the database by selecting a particular object definition, and then indicating the position on the terrain with the mouse. Object placements are accomplished only in the overhead, orthographic view of the placement window interface. An instance of the object definition is created and activated for manipulation by the user. The user

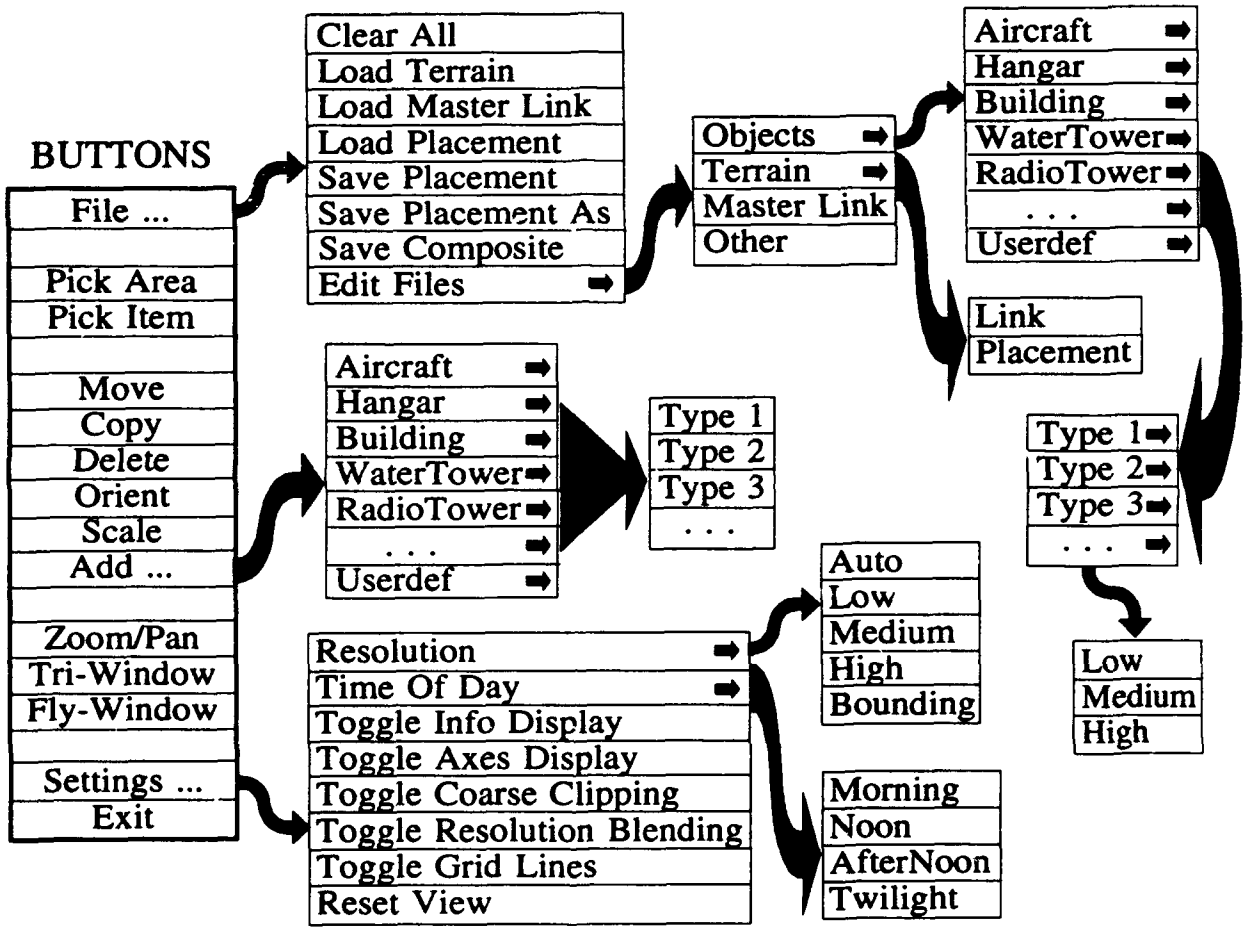


Figure 20. Cascading menu structure accessed through the Button Panel.

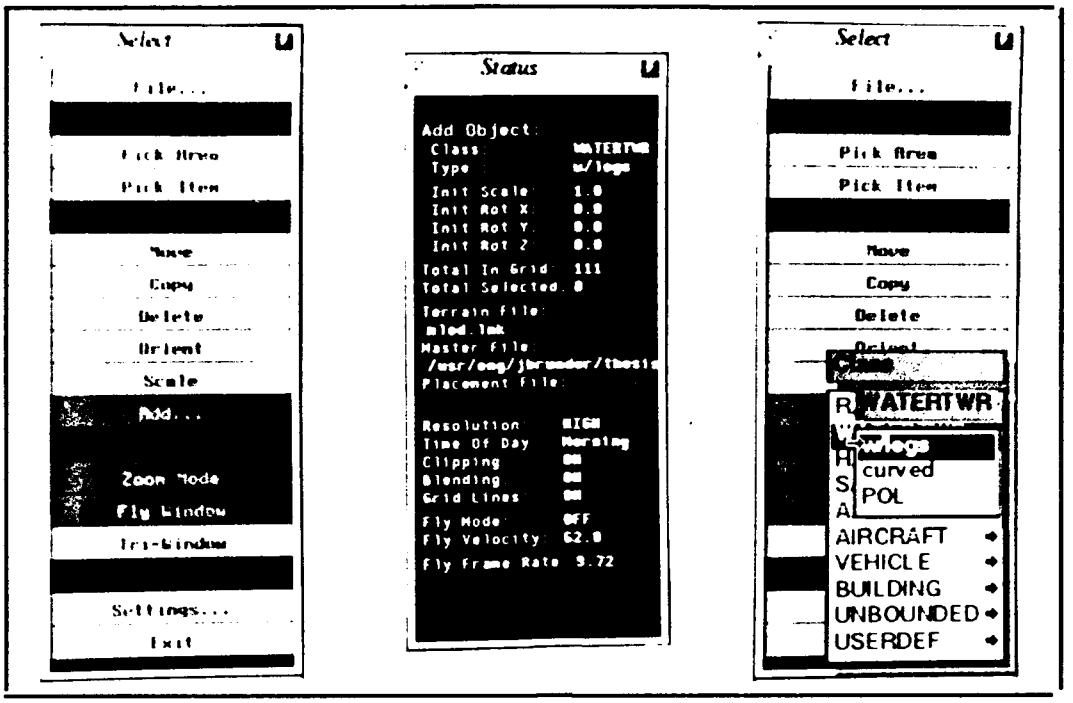


Figure 21. Pictures of Button Window and Status Window panels in DBGen.

is required to orient, and deactivate the object, before adding another object. To increase the efficiency of inserting objects in the database, the previous scale, rotation and altitude values are kept to apply to subsequent objects of the same type. If the user changes object types, however, the scale and rotation values are reset, but not the altitude. This is done because different object types might not be defined in the same local coordinate frame, nor with the same scaling factors. The altitude is a world coordinate parameter, and applies to all objects. Saving it helps greatly with placing multiple objects on terrain that is above sea level. Objects created with the add command are manipulated using the orient mode. The status window, show in Figure 21, provides information on these values. See the User's Manual in Appendix B for detailed mode interactions and Status Window description.

4.3.5 Mouse Interface The mouse is the primary means of user input in DBGen. With the numerous windows and functions required, the methodology was designed to be as consistent as possible across the various windows. Each button on the mouse controls a general type of action. The particular window and the modes selected determine the specific actions that are accomplished.

The middle mouse button manipulates the viewing volumes of each window. It allows the user to zoom or pan the view, depending on the mode set in the Button_Window. Zooming varies the shape of the volume, while panning varies the position.

The right mouse button is used to toggle states. In the Placement windows, it changes the objects on the selection list to active, or from active to normal. In the Fly- Through window, it activates the fly mode, starting and stopping the flight through the synthetic environment. In the Button_Window, it changes the modes or selects the items in the pull down menu.

The left mouse button carries out the actions of the modes selected. In the Placement windows, it performs two separate functions, depending on the status of the object list. If the list is in selection mode, left mouse perform a selection function, either putting objects on the list, or taking them off. If the list has been activated, the mouse movements manipulate the object as applicable to the window and the editing mode active. See the User's Manual for specifics. In the Fly-Through window, the left mouse button performs the function of the flight controls. In fly mode, it controls roll and pitch rates during flight. In stationary mode, it swivels the eyepoint around a fixed point in space. In the Button Window, it serves to select the actions available.

4.4 Class/Driver Construction

Only two additional classes had to be developed to support this application, other than the driver itself. These were the button panel routines. Figure 22 illustrates the interaction of the classes making up DBGen. Solid lines are inheritance lines. Hashed lines indicate message passing. The items with an asterisk are GDMS modules.

4.4.1 ButtonNode/ButtonWindow Class The button panel used in DBGen was designed for general use to allow inclusion into other SEL applications. It relies on other classes developed by Simpson (19). The button panel is actually constructed using two separate classes, ButtonNode and ButtonWindow. The ButtonNode class defines and manipulates an individual button on the screen, while the ButtonWindow class manages the panel and the list of buttons. The combination provides the user with 2-D, optionally shadowed buttons, with color toggle selection (See Figure 21).

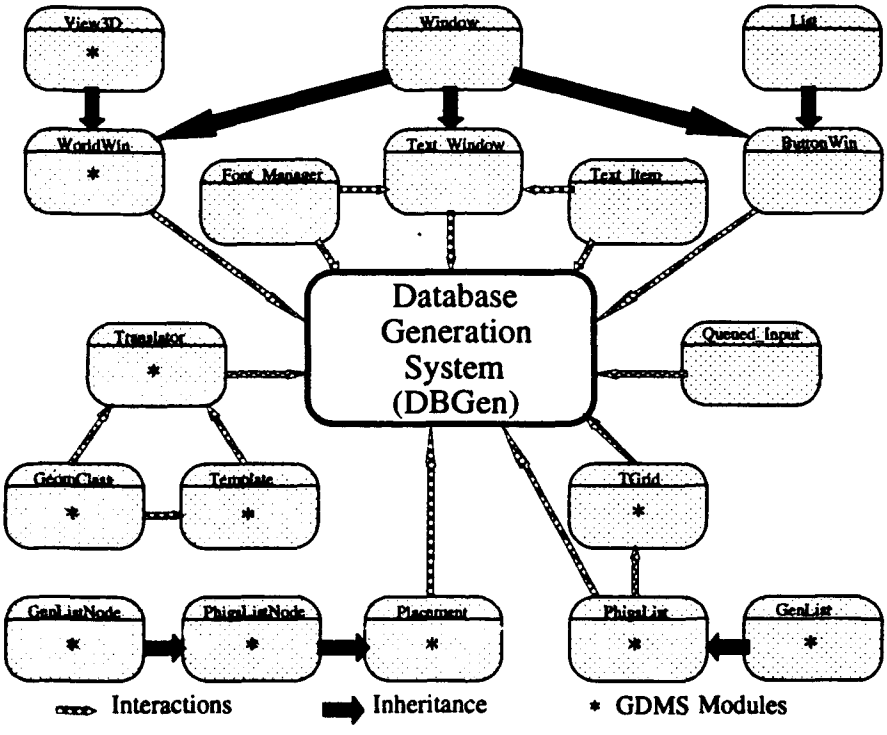


Figure 22. Class interaction in the construction of DBGen.

A ButtonNode is a derived class of a doubly-linked ListNode. It holds the color, size, position and title of a button as defined by the user. Methods give the user access to all its parameters.

A ButtonWindow is a derived class of both Simpson's (19) Window class, and a doubly-linked List class. The latter manages the list of buttons in a way similar to the GenList class. This code was developed early in the thesis cycle and has not been replaced with the generalized list code.

A user defines the size of the window, and the contents of each button, indicating a unique ID number, and the ButtonWindow class can set up and display them. A special method exists, WhichButton, that returns the button ID of the particular button 'pressed'. The programmer must provide the mouse coordinates relative to the ButtonWindow origin, for the proper button to be returned. The driver can use the returned ID number to process the desired action.

4.4.2 Driver The driver brings together a myriad of pieces to create the end product. It acts as a traffic light, directing messages from one class to the next. In addition to the classes already described in this thesis, it uses Queued_Input, Text_Item, Text_Window, and STimer classes for a variety of purposes. [See Simpson (19)]. The driver defines and initializes the entire application before settling down to process user input.

This processing is structured in sections. It checks, in order, the graphical buttons, keyboard input, the Fly-Through window, and each of the Placement windows; overhead, X-axis and Y-axis views. The code is structured so as to minimize looping time during dynamic updates of the scene. Manipulation of objects in one window are also updated in the other windows, if they are visible. Figure 23 is a psuedo-code rendition of the driver program.

4.5 Summary

DBGen proved to be an excellent platform for testing GDMS functionality. It contained most of the interfaces to the GDMS library that were required in the other SEL applications. It demonstrated the usefulness of GDMS for developing rapid prototypes.

It also proved to be an excellent application in its own right. Synthetic environments could be built and examined with relative ease. The next chapter shows some of the scenes generated

```
Parse Command Line;
Initialization {
        Load Terrain, Master Object, and Placement files.
        Construct Dynamic Menus;
        Setup Window Formats;
        initialize variables;
        Queue Inputs (mouse, keys);
}

Do {  //Main Loop
        Process Command Buttons;
        Process Key Inputs;

        If (Fly_Window Active) {
              Do {
                  Process Right Mouse Button;
                  Process Middle Mouse Button;
                  Process Left Mouse Button;
                  if (Fly Mode Set) Process KeyBoard;
                  Update Screen;
                  Update Frame Rate;
              }While (Flying OR Left Down OR Middle Down);
        }

        If (OverHead Window Active) {
                Process Right Mouse Button;
                Process Middle Mouse Button;
                Process Left Mouse Button;
                Update All Three Windows;
        }

        If (Side 1 Window Active) {
                Process Right Mouse Button;
                Process Middle Mouse Button;
                Process Left Mouse Button;
                Update All Three Windows;
        }

        If (Side 2 Window Active) {
                Process Right Mouse Button;
                Process Middle Mouse Button;
                Process Left Mouse Button;
                Update All Three Windows;
        }

} until (Command to Exit);
```

Figure 23. Psuedo-Code of DBGen driver.

with DBGen. The performance of the program is also examined.

V. Results, Conclusions and Recommendations

Overall, this thesis effort accomplished what it set out to do. The Synthetic Environments Laboratory, through the Graphical Database Management System (GDMS), now has a set of C++ classes for constructing applications related to mission planning, battle management and flight simulation. GDMS provides the basic requirements for visualizing the environment, manipulating objects in the environment and moving through the environment. Three applications have already been implemented using GDMS; a flight simulator, a battle management system and the Database Generation System (DBGen).

DBGen also achieved its desired goals. It was an excellent platform for testing the features of the GDMS. As an application, it provided the means to generate synthetic environments for use in other SEL applications.

The remaining sections provide some measures of system performance, for both GDMS and DBGen, followed by a discussion of their strengths and weaknesses. Recommendations for future work are also provided.

5.1 GDMS

5.1.1 Results The Graphical Database Management System is a complex set of software. Many factors effect its overall throughput performance, such as the rendering options selected (blending, resolution mode, coarse clipping), the number of polygons in an object, the number of objects in a grid square, the number and size of grid squares in the database, the field of view, etc. Test cases were run, using DBGen, to extract the performance gain/penalty for these GDMS features.

Test runs were made diagonally across the grid, starting from the southwest corner and ending at the northeast corner. The images covered approximately 40 percent of the pixels within the fly-through window of DBGen. This window is 1048 pixels wide by 984 pixels high. Applications using the helmet-mounted display, which requires a much smaller window (512 x 512), should see performance increases over the results shown here due to fewer pixels drawn per frame. The tables below show frame rates for both the Silicon Graphics Iris 4D/85GT and the 4D/310GTX.

Table 2. Frame rate response of GDMS with various options set

36 sq miles - 6x6 grid
3 Levels of Detail
  Low   - 2 polygons/sq mile (72 Total)
  Medium - 8 polygons/sq mile (288 Total)
  High   - 50 polygons/sq mile (1800 Total)
53x50 degree Field-of-View (X by Y)
Terrain ONLY

COARSE CLIPPING	BLENDING	RESOLUTION	4D/85GT	4D/310GTX
NO	N/A	LOW	20	24
NO	N/A	MEDIUM	15	20
NO	N/A	HIGH	7.5	10
YES	N/A	LOW	20-25	24-30
YES	N/A	MEDIUM	15-20	20-30
YES	N/A	HIGH	7.5-20	10-29
NO	NO	AUTO	17 - 19	20
YES	NO	AUTO	17 - 22	20-30
NO	YES	AUTO	12 - 15	14.5-17.5
YES	YES	AUTO	12 - 19	15-20

NOTE: With no polygons visible (all clipped), the maximum update rates were:
  4D/85GT - 30 hz
  4D/310GTX - 45 hz

Table 2 compares the performance of the rendering options in GDMS. The first group of numbers provide a baseline for absolute throughput of the system in this configuration. The second group shows the effects of the coarse clipping algorithm. Notice that the initial value for the range, when all grids are in view, is the same as the baseline, indicating negligible cost for performing the clipping algorithm. The benefits of coarse clipping are obvious from the increased frame rate during the later part of the run. The third group measures the effects of 'AUTO' resolution switching. In AUTO mode, the average frame rate falls near the MEDIUM value. GDMS trades the HIGH resolution in the foreground for the low in the background. The effect is a higher apparent throughput for the system. The final group shows the effect of resolution 'blending' on performance. In blending mode, two resolutions are rendered simultaneously with separate transparency factors (in this case pattern overlays) during a narrow transition band. Since the polygon count varies depending on the distances to each object in the grid, the frame rate varies rapidly during traversal of the environment. The total effect of blending is a general reduction in frame rate. Having all the options on yields frame rates somewhere between the HIGH and MEDIUM values.

This table can also give us a comparison between the hardware performance. The 4D/310GTX has a performance advantage over the 4D/85GT of approximately 20 to 30

Table 3 attempts to isolate the effect of the gri[illegible] on performance. The total number of polygons in the scene is constant for the two resolut[illegible]own. In the cases without clipping, the grid size definitely has a detrimental effect. It is more apparent for LOW resolutions because a larger percentage of the rendering time is spent walking the grid structure over rendering polygons. With clipping on, however, the results are mixed. When a large number of grid squares are in the field of view, as at the start of the run, the effect is the same as without clipping. But as the view moves across the terrain, the higher grid count becomes favorable because the smaller squares can be clipped away sooner.

Table 4 shows the effect of Field-of-View variations with coarse clipping on the frame rate. Field-of-View (FOV) is the total angle observed in the X-direction as defined by the viewing volume. As the FOV decreases, more of the grid will be clipped out, so an increase in frame rate is expected. This is generally what occurred. The effects, however, were n[illegible]t as significant as anticipated. This was primarily due to the flight path over the terrain. By fly[illegible]g diagonally,

Table 3. Frame rate response of GDMS varying Grid dimensions

```
120x120 sq miles
Levels of Detail
  LOW - 800 polygons total
  HI  - 7200 polygons total
Blending - N/A
53x50 degree Field-of-View (X by Y)
Terrain ONLY
```

GRID	RESOLUTION	COARSE CLIPPING	4D/85GT	4D/310GTX
4x4	LOW (50 poly/grid)	NO	12.0	15.0
10x10	LOW (8 poly/grid)	NO	10.0	12.0
4x4	HIGH (450 poly/grid)	NO	2.6	3.7
10x10	HIGH (72 poly/grid)	NO	2.5	3.5
4x4	LOW (50 poly/grid)	YES	12.0-20.0	15.0-24.0
10x10	LOW (8 poly/grid)	YES	10.0-20.0	12.0-30.0
4x4	HIGH (450 poly/grid)	YES	2.9- 8.6	4.0-10.0
10x10	HIGH (72 poly/grid)	YES	3.0-13.0	4.0-15.0

Table 4. Frame rate response of GDMS varying Field-Of-View

```
10x10 grid - 120x120 sq miles
Levels of Detail
  LOW - 8 polygons/grid  (800 total)
  HI  - 72 polygons/grid (7200 total)
Coarse Clipping - ON
Blending        - N/A
Terrain ONLY
```

FIELD-OF-VIEW	RESOLUTION	4D/85GT	4D/310GTX
90	LOW	9.0-20.0	12.0-30.0
53	LOW	10.0-20.0	12.0-26.0
30	LOW	11.0-23.0	13.0-30.0
10	LOW	13.0-25.0	15.0-30.0
90	HIGH	2.5-11.0	3.5-13.0
53	HIGH	3.0-12.0	4.0-15.0
30	HIGH	3.7-15.0	4.5-15.0
10	HIGH	5.0-13.0	6.0-15.0

directly through the grid square intersections, the largest number of squares remain visible at any given time. This reduces the clipping effects. Additionally, as FOV is decreased, the image zooms and a higher proportion of the window is covered with image, which decreases frame rate. Since this is purely a function of the graphic hardware, it was not investigated in depth.

Table 5. Frame rate response of GDMS varying Grid dimensions

```
1x1 Grid
53x50 degree Field-of-View (X by Y)
Coarse Clipping - OFF
Blending        - N/A
```

	FRAME RATE		THROUGHPUT	
TOTAL POLYGONS	4D/85GT	4D/310GTX	4D/85GT	4D/310GTX
558	17.0	23.0	9,486	12,838
1060	13.0	20.0	13,780	21,200
1915	7.5	12.0	14,362	22,980
2989	6.0	8.6	17,934	25,705
4040	4.6	7.0	18,584	28,280

To get a better measure of the polygonal throughput of the system, with minimal effect of grid structure, a terrain grid with only one node was set up, and objects of varying polygon count were added. The results are shown in Table 5. The overall throughput gets better with more polygons. This is because a higher percentage of the work is being done by the hardware, as opposed to the software. From these numbers, the 310GTX is performing with about a 50

5.1.2 Strengths A strong point for GDMS is in its ability to manage and render blended, multi- resolution, hierarchical objects, in both perspective and orthographic viewports. Figure 24 shows a hand object constructed hierarchically with the TEMPLATE and GEOM file formats (See Section 3.5.1). GDMS gives the programmer the flexibility to construct and manipulate any kind of object based on polygons. It was designed with synthetic environments in mind, so is particularly well suited to these type of applications.

Another strength is that GDMS relieves the programmer from most of the details of managing the database, while giving the application user the flexibility to directly manipulate the objects in

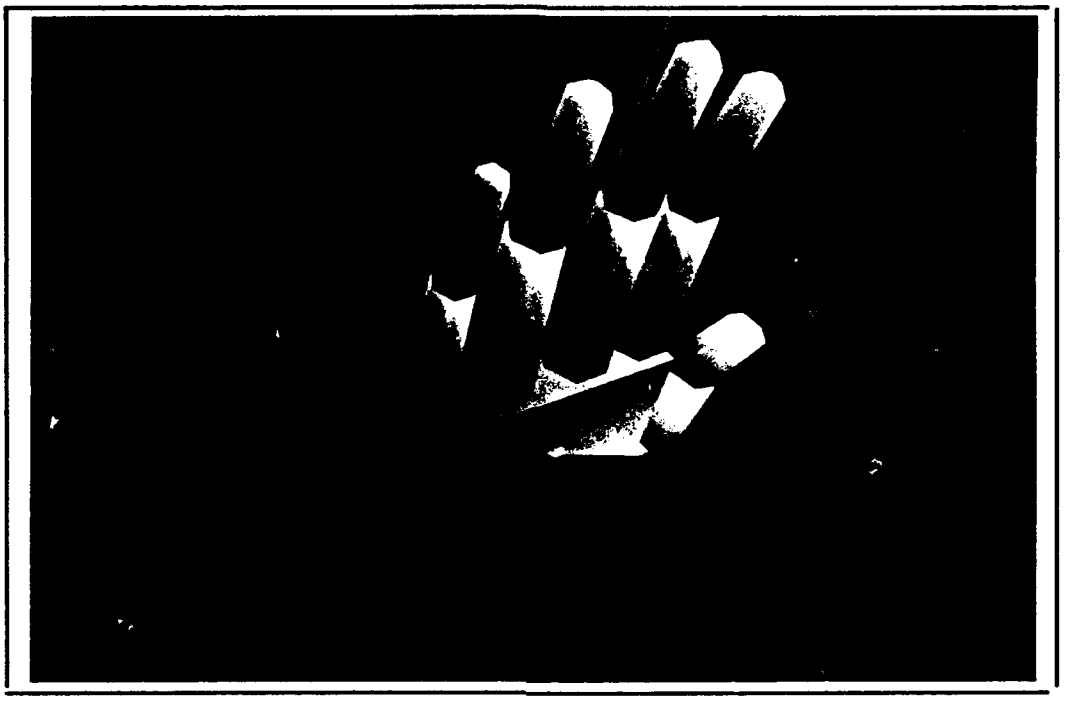


Figure 24. Picture of HAND object constructed hierarchically with GDMS.

the database through ASCII files. The file formats were set up to minimize the support programs needed to generate a database. Simple environments can even be constructed by hand.

A third strength is the object oriented design of GDMS. It allows programmers to extend the classes through inheritance, or, if necessary, replace the internal algorithms with minimal impact on the rest of the system.

In essence, GDMS provides everything but the control structures needed for coordinating between the user and the database.

5.1.3 Weaknesses and Recommendations GDMS is not the perfect system. It has its limitations and plenty of areas for improvement.

One of these is that GDMS is limited to models with polygonal descriptions only. Although other modeling primitives exist, like bi-cubic surfaces and textured ellipsoids, GDMS cannot handle them. Incorporating these structures into GDMS would greatly increase its flexibility. The GEOM file format already supports these two options, which could be used for various purposes

(clouds, spheres, etc), so a good portion of the work is already done.

The cost for these options, of course, will be reduced frame rates. The hardware is optimized for rendering polygons, so processing other geometric primitives is bound to be less efficient, in time, if not in the memory needed to hold the structures.

Another improvement to GDMS would be to implement texture mapping. GDMS does not currently support the texture mapping option of the GEOM file format, even if the hardware would support it. Hardware systems are now available which do real time texture mapping, so GDMS should be updated for this capability. The realism of the images produced would be far better with texture mapping, so the polygon count could be lowered accordingly, increasing frame rate.

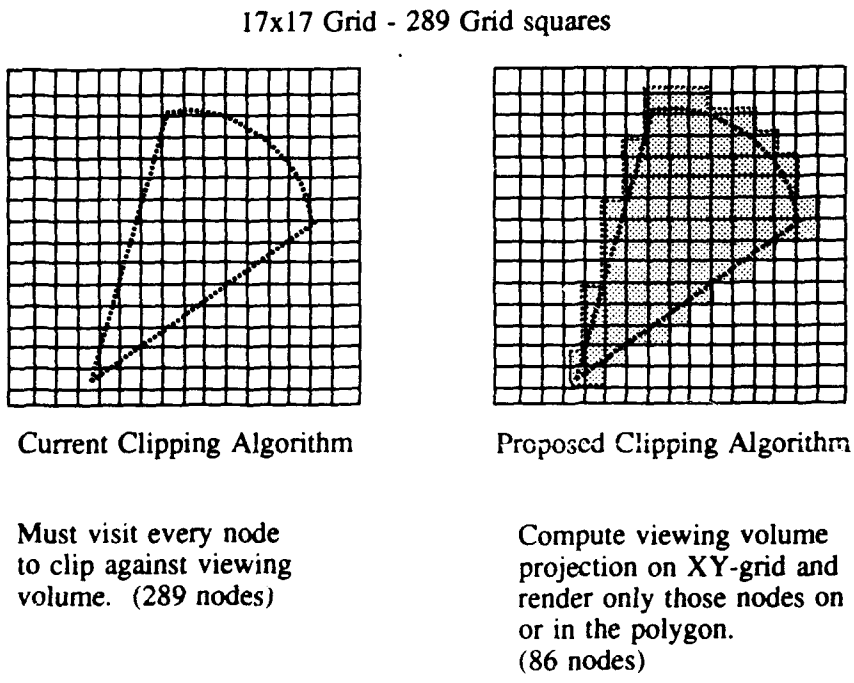


Figure 25. Coarse clipping algorithm comparisons.

As seen from Table 3, the grid dimensions have a marked impact on frame rate when large portions of the scene is visible. This is largely due to the algorithm used for coarse clipping. Currently every node is visited to determine its visibility. This number goes up as the square of

the grid dimension. Soon, the cost in time of processing every node becomes too expensive. For a solution, a modification to the algorithm could be implemented to quickly discard a large portion of the grid nodes from ever being processed (See Figure 25). This would allow the user to define large gaming areas with high grid square counts, subject to the limitations of memory, without a substantial frame rate reduction. This benefit becomes even more pronounced for environments with high object densities, since an even larger number of polygons are quickly rejected, along with the overhead of walking through the structures. This approach is generally better than the algorithm currently used, where every node must be rejected individually, but it does raise the issue of memory constraints.

Testing of large grid sizes revealed a limitation on the total number of files which can be loaded into memory at one time. The current hardware platforms are configured with only eight megabytes of RAM. During initialization of the geometry files, the machine reaches a point where excessive disk swapping takes place. This effectively limits the program to a maximum of about 400 GEOM files (20x20 grid) in memory at any one time. This varies somewhat depending on the size of the files. To allow GDMS to handle large gaming areas with minimal performance degradation, a mechanism is needed to intelligently swap the terrain grids in and out of memory as they come in and out of the visible area, overriding the automatic virtual memory feature of Irix (SGI Unix).

Throughput is always a weakness in rendering software. One can never have too fast a frame rate. Although the current rate is acceptable in some regimes, it does bog down at times. The bottlenecks must be found and corrected to get the best performance from GDMS. A profiler, if one should ever becomes available for the Iris workstations, is an excellent way to find these, and should be purchased.

For the system as is, one known way to increase the overall throughput of the rendering pipeline is to replace the recursive procedures in the Placement, Template, PhigsList and GenList classes, with iterative procedures. Recursion is generally slower than iterative procedures, and it is used extensively in the rendering methods to walk through the hierarchical structure. Restructuring these methods is bound to improve throughput.

5.2 *DBGen*

5.2.1 Results DBGen was used to as the platform for measuring the performance of GDMS. The frame rates quoted above were generated during simulated flight through DBGen's Fly-Through window.

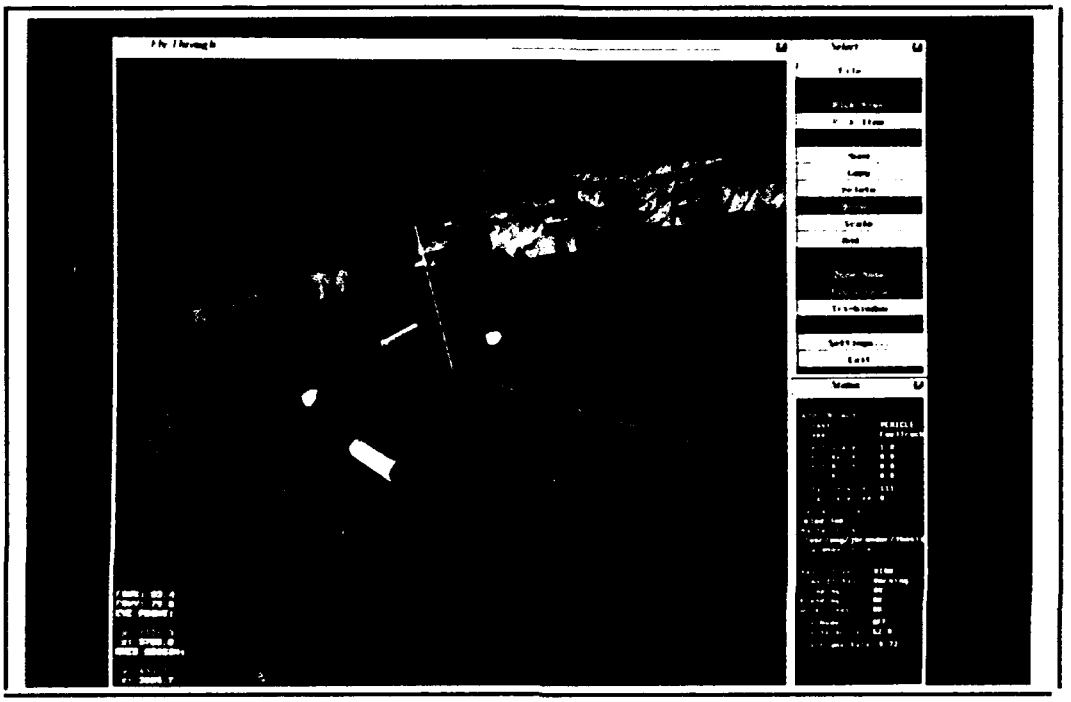


Figure 26. Synthetic environment created with DBGen (Grand Canyon area).

As an application, several environments were created from within DBGen for testing and demonstration purposes. The following figures show various databases built with the program.

Figure 26 shows an area around the Grand Canyon. Figure 27 shows the Death Valley area being populated with objects in the Tri-Window. Figure 28 is shows the front range near the Denver area.

5.2.2 Strengths The Database Generation System has proven to be a useful addition to the Synthetic Environments Laboratory. Its Tri-Window object placement system allows for intuitive placement of objects on terrain, while the fly-through window provides valuable instantaneous feedback on the real-time performance of the constructed database. The ease of switching between

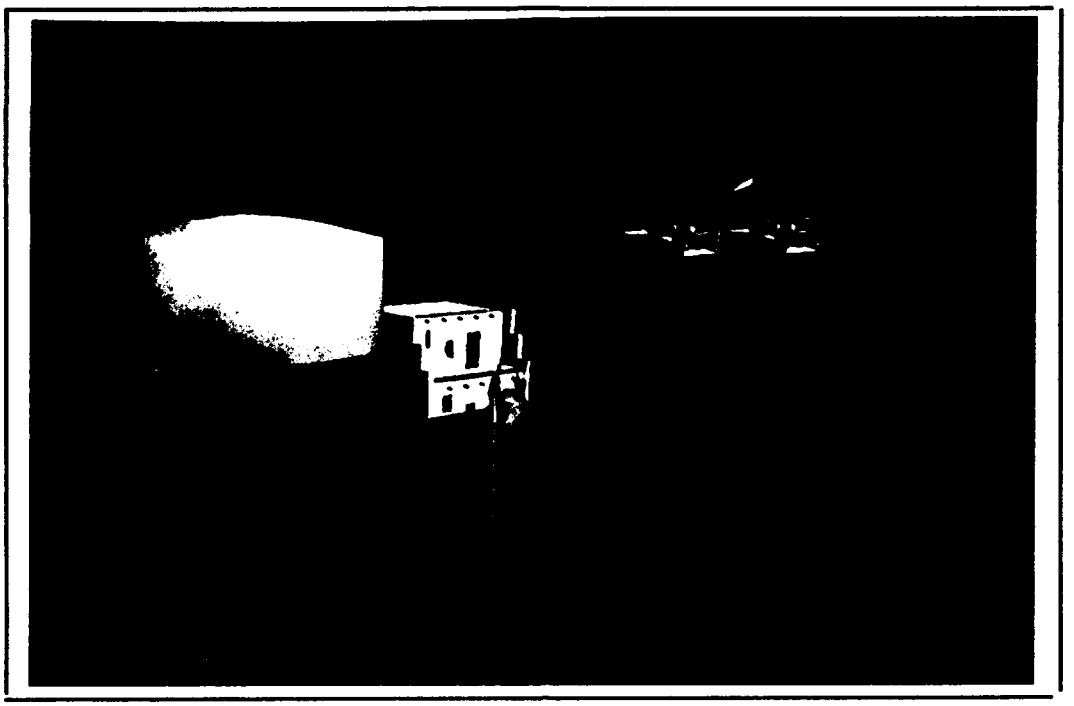


Figure 27. Synthetic environment created with DBGen (Death Valley area).

the two formats increases the productivity of the user and his confidence in the end product. The overall interface seems well suited to the intended task. The control the user is given over the program environment is very useful in the development of terrain databases. It is not perfect, though.

5.2.3 Weaknesses and Recommendations Although DBGen is good for what it does, several features are still missing.

DBGen cannot dynamically change an object's polygon colors, attributes, vertex positions or normals. A built in modeling capability to directly change these values would be quite useful. Additionally, an interface to Duckett's (5) terrain generation system would also be useful. A user could then generate the terrain files interactively, altering the grid spacing and polygon resolution as needed to optimize throughput.

Another area for improvement is in the information presented in the Status Window of DBGen. This is sufficient for controlling the program, but more is needed for a thorough analysis

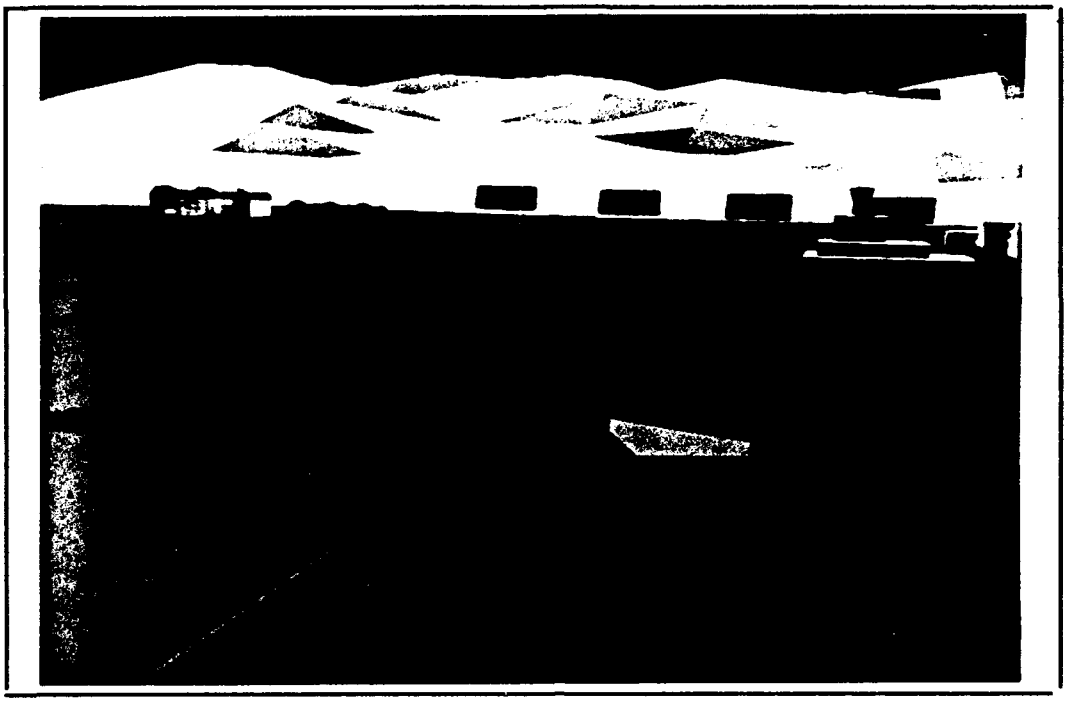


Figure 28. Synthetic environment created with DBGen (Denver Area area).

of a synthetic environment. Information global to the entire database, as well as the current scene, would be useful. In working with the system, the polygon counts for each object to be placed in the database is needed. Similarly, the polygon count for each grid, and for the entire database would be useful. Other information needed is the number and size of the grid squares, how many polygons in the terrain, the number of objects placed on the terrain, how many objects per grid, etc. Unfortunately, the space in the window is limited, so pop up boxes might be used to display information upon request. Other useful parameters would be dynamic information on the number of grid squares rendered or clipped, a measure on the number of polygons in any given scene, and the resolution switching distances for objects and terrain.

Two additional editing features would also be useful. The first is an automatic altitude/orientation mode which computes the initial placement on the terrain, regardless of the terrain's slope. The second is an area fill mode in which objects can be placed automatically at some specified density. The latter feature requires the former to be viable.

The next stage in the development of this application would be to extend the functionality

of DBGen into a mission planning or battle management arena. A thorough analysis of the requirements in such a system must be accomplished first. Any additional features required could then be implemented in piecemeal, using DBGen as the base. In this way, a application prototype could be developed for the Synthetic Environments Laboratory in a short time.

Appendix A. *File Formats*

In designing the database system, four separate data types emerged for the storage of a synthetic environment to disk. The first was the actual geometric descriptions of the objects themselves. The second was the construction of multiple objects into more complex objects. The third identified object file descriptions with object types. The fourth defined each object instance and placed it in world coordinates, while also allowing for hierarchical object construction.

The definitions of how these functions are stored is contained in the GEOM, TEMPLATE, LINK and PLACEMENT file formats, respectively. The GEOM format is described in (6). The others were developed specifically for this effort and are described in the sections below.

All the formats are ASCII based to allow for convenient editing and readability. Individual lines within files are limited to an arbitrary length of 1024 characters. A new line is started after each 'newline' character. When specified, either '/*' or '//' can be used to designate a comment line, provided they are the first none blank characters on the line. Blank lines are also permitted. Preprocessor commands are not supported.

The ASCII format was used for two reasons. First, the GEOM format was already ASCII based, and consistency in format eases understanding. Second, the conversions from disk to memory are designed as a preprocessing step, so file retrieval times were only a minor consideration.

A.1 TEMPLATE File

A TEMPLATE file is a special ASCII file used for creating Templates of complex objects out of several standard GEOM files. The format provides the information needed by the Template class to generate this information. In this format, comments or blank lines are permitted only after the first two lines. By convention, TEMPLATE files should have a 'desc' extension.

An excerpt from the TEMPLATE file used to construct a HAND object is shown if Figure 29. The file format is as follows:

```
LINE 1: Comment Line.  Can be anything up to 1024 characters.  This
line must exist.  It can not be blank.  This line is normally
used for file identification.
```

Table 6. TEMPLATE file format

LINE	KEYWORDS	COMMENTS
1	None	Up to 1024 characters
2	files <# of GEOM files> components <# of components>	File and object component counts
3+	file <filename>	GEOM filename
4+	component <n> parent <n> file <n> [begin] [end] [translate <f> <f> <f>] [rotx <f>] [roty <f>] [rotz <f>] [scale <f> <f> <f>]	Beware circular references References 3. Transformations bracketed by 'begin' and 'end'. Can extend beyond one line. Pre-multiplication order.

NOTE: Comments allowed after 2nd line.

```
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 0.00 -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 29. Excerpt from "hand.desc", a TEMPLATE file.

LINE 2: Must contain:

Keyword "files" followed by a count of the GEOM file names to be used in the Template. It is the users responsibility to insure the number of files actually used matches this number.

Keyword "components" followed by a count of the total number of components used in the object. For instance, a hand made up of four fingers and a thumb, each having two parts, plus the palm makes eleven components. This number must match the actual components defined later in the file.

LINE 3 - \# of files: Must contain:

Keyword "file" followed by a pathname to a geom file. This should include the full path to insure files could be read from any startup directory. List as many files as specified in LINE 2.

REMAINDER: Must contain:

Keyword "component" followed by an integer ID number between 1 and the number of "components" specified. Each component must have a unique ID number. The program warns the user if duplicate ID numbers are used. Ordering is unimportant.

Keyword "parent" followed by the component ID number of the parent component. Specifying a "parent 0" makes coordinate transformations relative to the local coordinate frame. Local coordinate frames should be modeled with Z being up to align with GDMS applications. Circular references can be constructed, but will result in an infinite loop if used. Users are responsible for insuring the links of parent ID's do not circle back to the original component.

Keyword "file" followed by the GEOM file number to use. "file 1" specifies the first pathname listed starting from LINE 3 above.

The remaining entries on this line position and orient the components relative to the specified parent. The keywords "begin" and "end" must bracket all these commands. Component entries can extend for as many lines as needed. The optional entries include:

"translate" followed by three floats (X,Y,Z).

"scale" followed by three floats (X,Y,Z).

"rotx" followed by float to specify angle about X in degrees.

"roty" followed by float to specify angle about Y in degrees.

"rotz" followed by float to specify angle about Z in degrees.

All transformations are constructed by pre-multiplication. The user should arrange calls in opposite order from desired execution.

A.2 LINK File

A LINK file is a special ASCII file used for creating the links between object types and appropriate object descriptions. The format provides the information needed by the Translator class to generate this information. In a LINK file, entries can not extend beyond one line (1024 chars). Comments or blank lines are permitted, but only after the first line. For convention, LINK files should have an 'lnk' extension.

An example from a LINK file is shown in Figure 30. The file format is as follows:

LINE 1: Comment Line. Can be anything up to 1024 characters. This line must exist. It must not be blank.

Table 7. LINK file format

LINE	KEYWORDS	COMMENTS
1	None	Up to 1024 characters
2+	<CATEGORY>	[TERRAIN] [RADIOTWR] [WATERTWR] [HANGAR] [AIRCRAFT] [BUILDING] [SAM] [AAGUN] [MISSILE] [USERDEF]
	maxtypes <n>	Type count in file.
3+	<CATEGORY>	Same as above.
	type <n>	
	level <1-3>	
	[name] <shortname>	Last defined resolution wins.
	[gpath <GEOM filename>]	Either gpath or dpath required.
	[dpath <TEMPLATE filename>]	

NOTE: Comments allowed after 1st line.

```
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 30. Example of simple LINK file.

LINE 2-?: The next block of entries define the maximum number of each category of object allowed in the database. Object categories can be any of the following:

(TERRAIN, RADIOTWR, WATERTWR, HANGAR, SAM, AAGUN,
AIRCRAFT, VEHICLE, BUILDING, MISSILE, and USERDEF).

An entry must start off with the object category (all capital letters), followed by the keyword "maxtypes", followed by the maximum number of object types to allocate for the given category. The program will not allow object types designated with ID numbers above this value. Each type designation can hold up to three resolutions.

LINE ?-eof: The final block of entries specify a short name and a path name for each object category, type number, and level of resolution. Entries must begin with the object category, listed above. The remaining keywords are:

Keyword "type" followed by an integer ID number (mandatory). Values above the "maxtypes" designated will be rejected, with a warning given.

Keyword "level" followed by a resolution index (mandatory). The index is an integer corresponding to resolution values as follows:

1 = LOW, 2 = MEDIUM, 3 = HIGH.

Polygon counts within resolutions are at the descretion of the user. The user must specify at least one resolution entry for an object to be defined. Multiple resolution entries are entered in separate lines. The user must insure that separate description files intended for a single object are designated with different resolution indices. If only two resolutions are entered for an object, they should be designated with consecutive resolutions for blending purposes. I recommend using LOW and MED for efficiency, if less than three resolutions are defined. Other values are rejected with a warning given.

Keyword "name" followed by a simple name for the object type (Optional). This should be specified for only one of an object's resolutions. It is a descriptive name to aid in object identification.

Keyword "gpath" or "dpath" followed by the full path name to the appropriate type image file. "gpath" indicates a GEOM file. "dpath" indicates a TEMPLATE file. Users must insure the file type matches the file designation. By convention, GEOM files have "geom" extensions, and TEMPLATE files have "desc" extensions, but this is not required. It is the users responsibility to insure the pathnames specified correspond to actual files.

A.3 PLACEMENT File

A PLACEMENT file is a special ASCII file used for creating instances of object types and placing them in a synthetic environment database, structured on a grid. The grid dimensions are specified in world coordinates. Objects placed outside of the grid are attached to the nearest node in the grid. The format provides the information needed by the TGrid and Placement classes to generate this information. It is also used in Duckett's Terrain Generation System (5). Comments or blank lines are permitted only after the second line. By convention, PLACEMENT files should have a 'dbs' extension.

An example of a terrain PLACEMENT file and an object PLACEMENT file are shown in Figures 31 and 32.

The file format is as follows:

LINE 1: Comment Line. Can be anything up to 1024 characters. This line must exist. It must not be blank.

LINE 2: Grid Definition. This line contains all the information needed to fully specify a grid in world coordinates. The 'Z'

Table 8. PLACEMENT file format

LINE	KEYWORDS	COMMENTS
1	None	Up to 1024 characters
2	[originLat <f> [N][S]]	For Terrain Builder Program
	[originLong <f> [W][E]	
	minx <n>	Grid dimensions (longs).
	miny <n>	
	maxx <n>	
	maxy <n>	
	gsize <n>	Grid square width
3+	create	Creates Phigs node
	<CATEGORY>	[TERRAIN] [RADIOTWR] [WATERTWR] [HANGAR] [AIRCRAFT] [BUILDING] [SAM] [AAGUN] [MISSILE] [USERDEF]
	type <n>	Reference to definition in LINK.
	[begin]	Transformations bracketed by
	[end]	'begin' and 'end'. Can extend
	[translate <f> <f> <f>]	beyond one line.
	[rotx <f>]	Pre-multiplication order.
	[roty <f>]	
	[rotz <f>]	
	[scale <f> <f> <f>]	
	[matrix <f> <f> <f> <f> <f> <f> <f> <f> <f>]	Upper 3x3 - Left->Right, Top->Bottom.
	[low2med <f> <f>]	IN, OUT switch distances
	[med2high <f> <f>]	
	[create]	Child construction allowed inside
	[close]	the definition of another object.
	close	

NOTE: Comments allowed after 2nd line.

```
terrain\_builder
originLat 36.0000 N originLong 118.0000 W minx 0 miny 0 maxx 11160 maxy 11160 gsize 1860

create TERRAIN type 1 begin translate 0 0 0 scale 1860.000000 1860.000000 1.000000 end close
create TERRAIN type 2 begin translate 0 1860 0 scale 1860.000000 1860.000000 1.000000 end close
create TERRAIN type 3 begin translate 0 3720 0 scale 1860.000000 1860.000000 1.000000 end close
create TERRAIN type 4 begin translate 0 5580 0 scale 1860.000000 1860.000000 1.000000 end close
create TERRAIN type 5 begin translate 0 7440 0 scale 1860.000000 1860.000000 1.000000 end close
create TERRAIN type 6 begin translate 0 9300 0 scale 1860.000000 1860.000000 1.000000 end close
create TERRAIN type 7 begin translate 1860 0 0 scale 1860.000000 1860.000000 1.000000 end close
create TERRAIN type 8 begin translate 1860 1860 0 scale 1860.000000 1860.000000 1.000000 end close
create TERRAIN type 9 begin translate 1860 3720 0 scale 1860.000000 1860.000000 1.000000 end close
```

Figure 31. Excerpt from PLACEMENT file (terrain.dbs).

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

create AAGUN type 1 close
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
close
create USERDEF type 1 begin translate -37.00 -204.00 0.00
    matrix 1.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 1.00 end
close
create AAGUN type 1 begin translate -128.00 -86.00 0.00
    matrix 1.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 1.00 end
close
```

Figure 32. Excerpt from PLACEMENT file (test.dbs).

axis is assumed to be up. Entries can not extend to next line. Legal entries for this line include:

Keyword "minx" followed by a long integer representing the farthest extent of the grid in the '-X' direction (Westerly).

Keyword "maxx" followed by a long integer representing the farthest extent of the grid in the '+X' direction (Easterly).

Keyword "miny" followed by a long integer representing the farthest extent of the grid in the '-Y' direction (Southerly).

Keyword "maxy" followed by a long integer representing the farthest extent of the grid in the '+Y' direction (Northerly).

Keyword "gsize" followed by an integer specifying the incremental length of each grid square over the entire area.

NOTE: Invalid entries from above will cause the system to abort the load of a PLACEMENT file with an accompanying message.

The following entries are used only in the Terrain Generation System for placing multiple terrain blocks appropriately.

Keyword "originLat" followed by a float representing latitude in degrees and character of 'N' or 'S'.

Keyword "originLong" followed by a float representing longitude in degrees and character of 'E' or 'W'.

LINE 3-eof: The remaining lines are used to instantiate objects and place them in the database. The format follows a PHIGS-like structure by allowing hierarchical constructions. Objects are identified by category and ID number. The user is responsible

for insuring the object type is included in the active LINK file. An error message results if the object can not be identified. An object entry definition creates an object instance, positions and orients it, sets resolution switching parameter, and then closes it. Hierarchically structured objects can be constructed by creating a new object before closing the old. The user must insure all open objects are closed properly. Object transformations are done between two keywords, "begin" and "end". Entries can extend beyond one line. The keywords for this block are:

Keyword "create" followed by one of the object categories described in the LINK format.

Keyword "type" followed by the integer ID number corresponding to the object type defined in the LINK format.

Keyword "begin" to initiate transformation processing (Optional).

Keyword "end" to terminate transformation processing. Required if "begin" is used. Transformation must be concluded before creating any new child objects.

Keywords "begin" and "end" are must bracket all object manipulation commands. The optional entries include:

"translate" followed by three floats (X,Y,Z).

"scale" followed by three floats (X,Y,Z).

"rotx" followed by a float to specify angle about X in degrees.

"roty" followed by a float to specify angle about Y in degrees.

"rotz" followed by a float to specify angle about Z in degrees.

"matrix" followed by nine floats specifying the upper 3x3 matrix of the 4x4 modeling matrix. Entries are listed in left to right, top to bottom order.

"low2med" followed by two floats. The first specifies the distance to switch from LOW to MEDIUM, the second from MEDIUM to LOW.

"med2high" followed by two floats. The first specifies the distance to switch from MEDIUM to HIGH, the second from HIGH to MEDIUM.

Keyword "close" to match the "create" and terminate object definition.

All transformations above are constructed by pre-multiplication. Users should arrange calls in opposite order from desired execution. For child objects, transformations are relative to the parent object instead of the world coordinate system.

Appendix B. *DBGen USER's MANUAL*

The Database Generation System (DBGen) is a tool for populating polygonal terrain models with polygonal object models to form synthetic environments. The following section describes the command line interface, the window layout, mouse interface, and keyboard commands.

B.1 Command Line Interface

Parameters can be passed into the program to load terrain files, object link files, and object placement files (See Appendix A for file formats). The format for the command line is shown below.

```
dbgen [-h/?] [-t <terrain>] [-m <object definitions>]
      [-d <object placements>]

where: -h/? option          - prints the usage line above
   <terrain>            - pathname to terrain 'lnk' \& 'dbs' files
   <object definitions> - pathname to master 'lnk' file of objects
   <object placement>   - pathname to object 'dbs' file

NOTE:  All pathnames are specified without extensions.
```

Any or all of these parameters can be passed into the program. Options exist within the program to change these at any time.

B.2 Window Layout

The screen is broken down into three areas, a command button area, a status area, and a viewing/manipulation area. The command Button window provides access to all the commands and modes available in DBGen. The Status window displays pertinent information on the state of the program. The viewing/manipulation windows are where the work is done.

B.2.1 Button Window The Button window consist of a series of graphical buttons used to activate commands, pull down menus or change modes in the program. There are four sections of

buttons, file commands, editing modes, window modes, and setup commands. Figure 33 shows the relationship between the buttons and the pull down menu. The following paragraphs will explain what each function does.

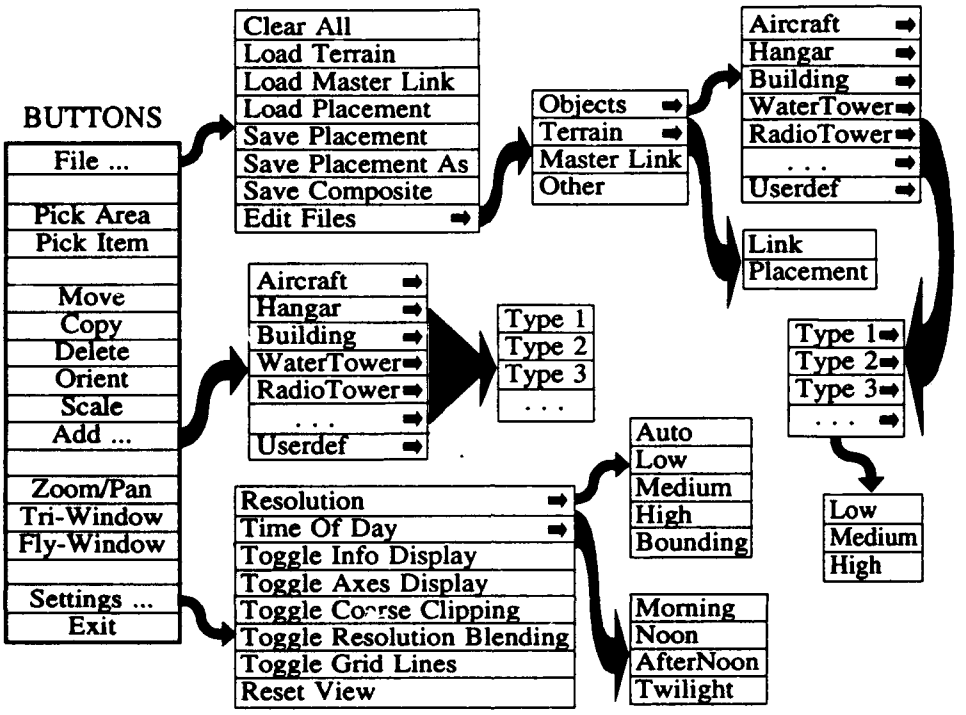


Figure 33. Button window with cascading menu relationships.

B.2.1.1 File Commands The 'FILE ...' button pops up a menu related to the files DBGen uses on disk. The options are as follows:

```
'Clear All' - purges current database from memory.  (Requests
   Confirmation before proceeding)

'Load Terrain' - Requests a single pathname for terrain files.
   The filename must have no extension.  The program automatically
   appends '.lnk' and '.dbs' to the name given.  Terrain files may be
   created through the Terrain Generation System (See Duckett [14]).
   Old terrain information is removed from memory before loading the
```

new files.

'Load Master Link' - Requests pathname for master object file. The filename must have no extension. The program automatically appends '.lnk' to the name given. The master object file must be constructed by hand (See Appendix A for LINK fileformat). Old object description information is removed from memory before loading the new file.

'Load Placement' - Requests pathname for object placement file. The filename must have no extension. The program automatically appends '.dbs' to the name given. The object file is generated by DBGen through the 'Save Placement' command. Old placement objects are removed from memory before loading the new file.

'Save Placement' - Saves the current state of the database to the filename loaded with the '-d' option of the command line or the 'Load Placement' command. If thefilename has not been provided, the program will ask for one.

'Save Placement As' - Saves the current state of the database to the filename specified by the user. CAUTION - The program will overwrite a file of the same name on disk without any warning.

'Edit Files' - This option calls up another menu for file selection. Once the file has been selected, a shell to 'vi' is invoked. You must exit the shell to continue work in the program. The program is suspended until this occurs.

'Objects' - Pops up a series of dynamically constructed menus based on the object descriptions loaded in the master link file. The user must select the object category, type ID, and level of resolution to get access to a particular file for editing or

viewing (See Figure ~\ref{figB1}). Be advised, the normal UNIX permissions on files apply.

'Terrain' - Pops up a menu to allow you to select either the terrain LINK or PLACEMENT file for editing. This uses the names loaded from the command line or the 'Load Terrain' option above.

'Master Link File' - Invokes 'vi' with the name of the master object file loaded from the command line or the 'Load Master Link' option above.

'Other' - This option invokes 'vi' after requesting a filename. Extensions must beprovided here.

B.2.1.2 Picking Modes The Picking Modes determine how objects are selected in the overhead view of the TRI-WINDOW format.

'PICK AREA'

When the mode is active in this window, objects in the database are selected from an area swept out by the mouse. One corner of the box is anchored by pressing in the left mouse button. The opposite corner is specified upon release. Objects within the region formed by the two corners are selected. Pick Area and Pick Item are mutually exclusive modes.

'PICK ITEM'

When the mode is active in this window, objects in the database are selected/deselected based upon their X-Y distance from the position specified. Only the specified grid square and the eight surrounding squares are searched for the 'nearest' object. Pick Item and Pick Area are mutually exclusive modes.

B.2.1.3 Editing Modes Editing buttons toggle MODES which specify how objects are manipulated within the Tri-Window format. The tables in Section B.3 give specifics for each window and each mode. In addition to setting the mode, these buttons can also activate or deactivate the list, depending on its state. If items are selected on the list (in RED), toggling an

editing mode will automatically activate them (in GREEN) for manipulation. Conversely, if the list is active, selecting the same mode button will return the list to the selection state. Changing modes with active objects does not alter their state. The editing buttons follow:

'MOVE'

Objects activated with this mode set are moved about in the plane of the active window. Movement follows the mouse pointer movement.

'COPY'

This feature requires some elaboration. When a user activates the list with copy mode set, a new list of objects are created, offset slightly, and activated. The original list is returned to its normal state in the database. The user must drag the new objects to a new location, or overlapping objects will occur, reducing frame rate. Manipulation works identically to the move mode. In order to make additional copies, or add objects to the selection list, reselect the 'COPY' button, returning the active objects to the selection state, and repeat the process.

'DELETE'

Objects are deleted from the database when activated with this mode set. Multiple objects can be deleted individually, or selected as a group and deleted all together. Caution must be used when selecting this mode from the panel, because any items on the list will be immediately deleted.

'ORIENT'

Objects activated with this mode set are rotated about the local coordinate axis corresponding to the View Plane Normal of the active window. Additionally, this mode allows fine adjustments of the object's altitude. Movement follows the mouse pointer movement; horizontal rotates, vertical elevates. The ARROW, and PAGE keys also adjust altitude, allowing larger increments.

```
Page Up           - Adds 1000 units
Page Down         - Subtracts 1000 units
Up Arrow          - Adds 10 units
Dn Arrow          - Subtracts 10 units
Shift-Page Up     - Adds 100 units
Shift-Page Down   - Subtracts 100 units
Shift-Up Arrow    - Adds 1 units
```

```
Shift-Dn Arrow      - Subtracts 1 units
```

'SCALE'

Objects activated with this mode set are scaled symmetrically about their center. Multiple objects are scaled together from the group center. Negative scaling is possible, causing mirroring of the objects.

'ADD'

Selecting 'ADD' automatically deselect and reinsert all objects on the list, and pops up a menu of available object categories to choose from. This menu is constructed dynamically from the master LINK file specified at startup, or from the 'FILE ...' menu. Each entry calls up the set of objects available for that category. Selecting one of these object types activates the 'ADD' mode for that object description, and initializes the global rotation and scale parameters. (See Section B.3). Controls are the same as for 'ORIENT' mode.

B.2.1.4 Window Modes Three buttons exist for changing modes related to the window formats, 'ZOOM/PAN', 'TRI-WINDOW', and 'FLY-WINDOW'.

'ZOOM/PAN'

This button toggles the effect of the middle mouse button on the viewing volumes in each window. The mode is indicated by the title on the button, either 'ZOOM' or 'PAN'.

'TRI-WINDOW'

This button pops the three window format to the front of the screen to allow for database editing. TRI-WINDOW and FLY-WINDOW are mutually exclusive modes.

'FLY-WINDOW'

This button pops the fly-through window to the front of the screen to allow interactive fly through capability. The viewpoint is initially set looking at the last object placed in the database. TRI-WINDOW and FLY-WINDOW are mutually exclusive modes.

B.2.1.5 Setup Commands The 'SETTINGS ...' button pops up a menu controlling the rendering options in the program. The settings are global to every viewing window. The options are as follows:

'Resolution' - This selection calls up another menu allowing the user to set the level of detail to be rendered. Five modes exits. The first is 'Auto', where the program automatically selects the resolution to render based the object's distance from the eyepoint. The next three select Low, Medium, or High levels of detail only. The final setting is for Bounding Boxes, which renders the bounding volumes instead of the objects.

'Time of Day' - This calls up a menu to select four general time of day options for the lighting model, Morning, Noon, Afternoon, and Twilight.

'Toggle Info Display' - This toggles the textual information in the lower left corner of each viewing window on or off.

'Toggle Axes Display' - This toggles the display of the axes on or off. The information displayed for axes origin is also toggled.

'Toggle Coarse Clipping' - This toggles coarse clipping on and off for rendering of the database. The 'C' key also does this.

'Toggle Resolution Blending' - This toggles the blending option between levels of detail. The 'B' key also does this.

'Toggle Grid Display' - This toggles the display of grid lines on the synthetic environment. The 'G' key also does this.

'Reset Views' - This resets the viewing volume and eyepoint of each window to their initial values.

B.2.2 Status Window The Status Window provides information pertinent to the state of the program. Most of the entries are self explanatory. The following is a short description of the potentially confusing entries:

The 'Add Object' entries indicate the current mode for adding objects to the database. If 'NILL' is indicated, no object is selected.

The 'Init Scale', 'Init Rot X', 'Y', and 'Z' entries relate what initial values are applied to objects first added to the database. These values are obtained directly from the object being manipulated when in Add mode.

The 'Total in Grid' entry indicates how many objects are contained in the database, not counting those objects selected. This number includes the terrain objects (i.e. - a 6x6 terrain grid has a minimum of 36 objects indicated).

The 'Fly Velocity' entry is a relative parameter indicating how many units are traveled per frame when in fly mode. The "actual" speed over the terrain is totally dependent on frame rate and the units defined in the terrain grid. This velocity can be varied using 'A' and 'S' keys. 'A' decrements velocity by one unit per frame. 'S' increments by one unit per frame.

B.2.3 Viewing/Manipulation Windows The viewing area serves two different purposes, depending on the mode set from the button window. The first is to allow the user to place and orient objects in the environment. The second is to allow the user to see how the environment looks from a moving platform. Each of these functions use its own window format. (See Figure 34 for window formats.)

The first format, or placement format, uses three separate windows, one for each coordinate axis. The overhead view (aligned with the z-axis) uses a wide area orthographic projection for placing or retrieving objects on the terrain. The view centerline remains fixed during object insertion. This is the only window that can be used for selecting or placing objects. The two side views (looking down the x and y-axes, respectively), are perspective projections to help with object orientation in three dimensions. These views are slaved to the object being manipulated, so that it always remains fixed in the center of the window. Coordinate axes are displayed in each

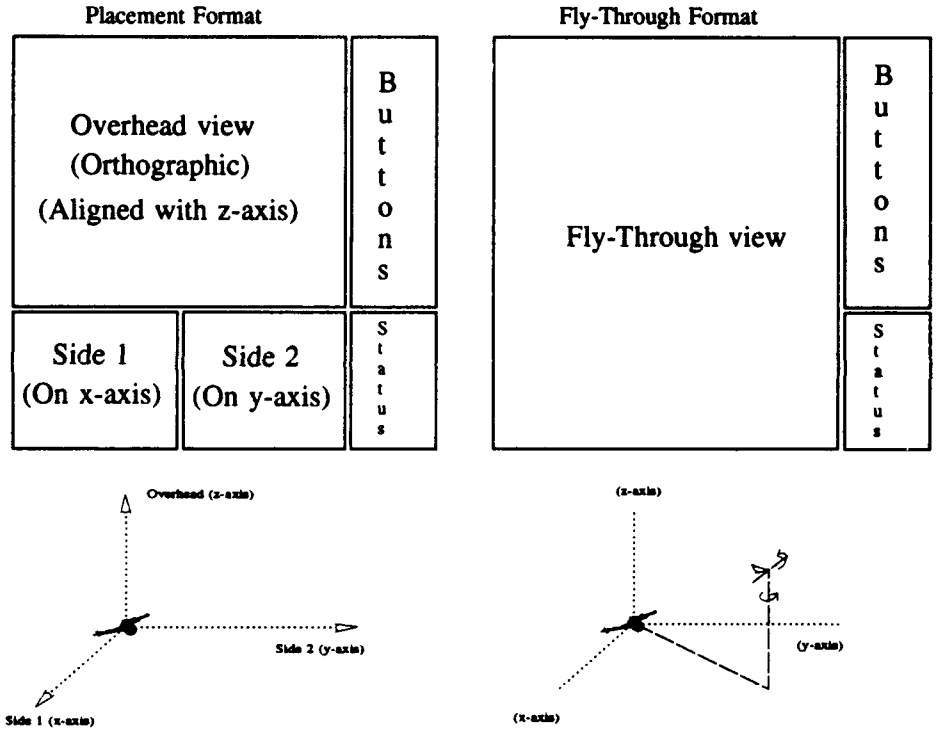


Figure 34. Window formats in DBGen.

window for reference. Red is used for the X, green for the Y, and blue for the Z axis. The XYZ axes correspond to the RGB colors for a memory aid.

The second format, or fly-through format, uses a single window occupying the same space as the three placement windows. It uses a perspective projection to simulate a view from a cockpit. The user can select a fly mode or look-around mode for observing the environment. A simple reference guide is used to help with attitude orientation. The coordinate axes are projected out in front of the 'plane', so that the origin is always centered in the window. The blue colored axis (Z), always points to the sky.

The specifics on how the interaction takes place in each of these windows is covered in the mouse interface section (B.3)

B.3 Mouse Interface

The mouse is the primary means of user input in DBGen. Each button on the mouse controls a general type of action. The particular window and the modes selected determine the specific actions that are accomplished.

B.3.1 Middle Mouse Button

The middle mouse button manipulates viewing volumes of windows in the database interaction area. The user clicks and holds the button down while in the desired window, then moves the mouse pointer to zoom, or pan the view, depending on the mode set in the button window.

In ZOOM mode, moving left zooms out and moving right zooms in. This changes the view volume's u-v dimensions to give both the orthographic and perspective views a zoom capability. Moving the mouse down and up moves the eyepoint in and out, respectively, along the view plane normal (VPN). This effects only the resolution switching of objects in the overhead, orthographic window, though it also zooms the views in the perspective windows.

In PAN mode, the movement of the mouse, with the button held down, appears to drag the object images in the direction of movement, relative to the plane of the window. In actuality, the eyepoint is moving opposite to the apparent object movement.

B.3.1 Right Mouse Button The right mouse button is used to toggle states. This button effects the Placement windows, the Fly-through window, and the Button window.

In the Placement windows, it toggles the objects on the selection list to active, or from active to normal. If an object list is in the selection state (colored red), a click of the right button will change their status to active (green) for manipulation. A subsequent click will switch to a normal status, reinserting the objects in the database, and clearing the list. If the list is already clear, the right mouse button has no effect.

In the Fly-Through window, the right mouse button toggles the fly mode, starting or stopping the flight through the synthetic environment. The mode set determines the actions that will be accomplished with the left mouse button. Moving the mouse pointer outside the window area will turn off the fly mode also.

For the Button window, the right mouse button activates the particular graphical button under the cursor. Graphical buttons are used to toggle modes, pull down menus, or execute commands. Entries in pull-down menus are selected by releasing right mouse button on desired selection.

B.3.2 Left Mouse Button The left mouse button performs different actions in each window, depending on the modes selected. Modes are set through the Button window and with the right mouse button.

Table 9. Left mouse effects in Overhead window.

MODE	ACTION	COMMENT
MOVE	L/R	Drags group left/right wrt view (-+X-axis)
	U/D	Drags group up/down wrt view (+-Y-axis)
COPY		Same as MOVE
DELETE	N/A	Objects in list are deleted upon activation
ORIENT	L/R	rotates group about Z-axis
	U/D	moves group up/down relative to terrain
SCALE	L/R	N/A
	U/D	scales group up/down uniformly in x, y & z
		scaling group to negative values will mirror image
ADD		Same as ORIENT

B.3.2.1 Overhead Window In the overhead window of the Placement format, the left mouse button performs three separate functions. First, if the list is either empty or in selection mode (red), and the program is not set to insert objects (ADD mode), left mouse clicks will select or deselect objects in the database. The program looks for the nearest object to the mouse pointer, in the grid square selected and in eight surrounding grid squares.

Second, if the program is in ADD mode, a click will insert an object at the point indicated, and activate the object for modification. Items on the list at the time of mode selection will be reinserted in the database.

Third, if the list is activated (green), clicking and holding the left button down will manipulate the object on the list as applicable to the window and the editing mode selected. See Table 9.

B.3.2.2 Side 1 Window (On x-axis) In side window 1, the left mouse button serves to manipulate objects with respect to the X-axis. No selection or insertion capability exists. See Table 10 for mode actions.

Table 10. Left mouse effects in Side 1 window.

MODE	ACTION	COMMENT
MOVE	L/R	Drags group left/right wrt view (-+Y-axis)
	U/D	Drags group up/down wrt view (+-Z-axis)
COPY		Same as MOVE
DELETE	N/A	Objects in list are deleted upon activation
ORIENT	L/R	rotates group about X-axis
	U/D	moves group up/down relative to terrain
SCALE	L/R	N/A
	U/D	scales group up/down uniformly in x, y & z
		scaling group to negative values will mirror image
ADD		Same as ORIENT

B.3.2.3 Side 2 Window (On y-axis) In side window 2, the left mouse button serves to manipulate objects with respect to the Y-axis. No selection or insertion capability exists. See Table 11 for mode actions.

In the Fly-Through window, the left mouse button serves two functions.

Table 11. Left mouse effects in Side 2 window.

MODE	ACTION	COMMENT
MOVE	L/R	Drags group left/right wrt view (+-X-axis)
	U/D	Drags group up/down wrt view (+-Z-axis)
COPY		Same as MOVE
DELETE	N/A	Objects in list are deleted upon activation
ORIENT	L/R	rotates group about Y-axis
	U/D	moves group up/down relative to terrain
SCALE	L/R	N/A
	U/D	scales group up/down uniformly in x, y & z
		scaling group to negative values will mirror image
ADD		Same as ORIENT

In one mode, as the user "flys" through the environment, the left mouse button controls the pitch and roll rates of a simulated 'aircraft' in flight. Pressing and holding the left button enters roll and pitch rate commands based on the distance from the center reference point. Left of center rolls the 'plane' left, while right of center rolls to the right. Similarly, above the center point pitches the 'plane' down, executing a pushover, while below center pulls the 'plane' up. The further from the center, the faster the rate. These commands are similar to the inputs used in the Silicon Graphics 'flight' program.

In the second mode, when flight is turned off, the left button allows the user to swivel around in his position. Left movement swivels the view volume to the left and right movement swivels the volume to the right. Up movement swivels the volume up and down movement swivels the volume down. Flight will resume along the new line of sight when activated.

Appendix C. *Unix Manual Page*

dbgen(1) USER COMMANDS dbgen(1)

NAME

dbgen - places objects in synthetic environments interactively.

SYNOPSIS

dbgen [-t terrainfile] [-m masterobjectfile] [-d placementfile]
[-h] [-?]

DESCRIPTION

The DataBase Generation System (DBGen) is an application of the Graphical Database Management System (GDMS). It is used for generating synthetic environments. DBGen allows a user to orient, scale, move, delete and add multi-resolution objects to synthetic environments interactively. It also provides a fly-through capability for immediate feedback on the dynamic response of the database. The program uses the mouse and keyboard for inputs. Keyboard response is as follows:

1) Effects object altitude:

Page Up	- Adds 1000 units
Shift-Page Up	- Adds 100 units
Up Arrow	- Adds 10 units
Shift-Up Arrow	- Adds 1 units
Page Down	- Subtracts 1000 units
Shift-Page Down	- Subtracts 100 units
Dn Arrow	- Subtracts 10 units
Shift-Dn Arrow	- Subtracts 1 units

2) Effects fly-through velocity

AKEY	- Decreases velocity by 1 unit
SKEY	- Increases velocity by 1 unit

3) Effects environment settings

GKEY	- Toggles the display of grid lines
BKEY	- Toggles multi-resolution blending

CKEY - Toggles coarse clipping in software

The mouse is used for the majority of interaction with the program. Its effect is dependent on the active window and the mode of the program. The active window is the window in which the mouse pointer resides when the mouse button is depressed. The effects break down generally as follows:

1) Middle Mouse Button - This effects the view volume of the active window, zooming or panning based on the mode selected in the button panel.
2) Right Mouse Button - Toggles states on or off, depending on the window.
3) Left Mouse Button - Selects and performs actions on objects in the tri-window. Acts as flight control stick in fly-through mode.

OPTIONS

-h -?

Prints the synopsis from above.

-t terrainfile

Load the terrain information included in the "terrainfile" LINK and PLACEMENT files (*.lnk \& *.dbs).

-m masterobjectfile

Load the LINK file specified with "masterobjectfile". This includes the definitions of all available objects to be active in the application (*.lnk).

-d placementfile

Load the PLACEMENT file specified with "placementfile". This holds any predefined object placement information (*.dbs).

NOTE: All pathnames are specified without extensions.

FILES

terrainfile

specifies the geometry files, grid and placement information for multi-resolution terrain.

masterobjectfile

specifies GEOM and TEMPLATE files defining each resolution of each object available in the database.

placementfile

specifies the placement information for any objects placed on the terrain specifed above.

BUGS

Any attempt to reload identically named geometry files within the program does not correctly update the descriptions in memory. The old descriptions remain.

Bibliography

1. Breden, W. O. and J. J. Zanoli. *Visualization of High-Resolution Digital Terrain.* MS thesis, NPS52-89-38, Naval Postgraduate School, Monterey, CA, June 1989.
2. Computer Systems and Technology Division, Electronic and Computer Systems Laboratory, Georgia Tech Reasearch Institute. *(Draft) Route Evealuation Module (REM), Software Users Manual*, August 1989.
3. Dahn, D. A. *A Low-Cost part-Task Flight Training System: An Application of a Head Mounted Display.* MS thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1990.
4. DCS for Communications-Computer Systems, Directorate of Computers, Systems Support, HQ TAC. *Computer-Assisted Force Management System (CAFMS)* (Ver. 6 Edition), November 1988.
5. Duckett, D. P. *Application of Statistical Estimation Techniques to Terrain Modeling.* MS thesis, AFIT/GCE/ENG/91D-02, Air Force institute of Technology, Wright-Patterson AFB, OH, 1991.
6. Filer, R. E. *A 3-D Virtual Environment Display System.* MS thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1989.
7. Foley, J., et al. *Computer Graphics: Principles and Practice.* Addison-Wesley Publishing Company, 1990.
8. Gerken, M. *An Event Driven State-Based Interface for Synthetic Environments.* MS thesis, AFIT/GCS/ENG/91D-07, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1991.
9. Howard, T. L. J. "An Annotated PHIGS Bibliography." In *Computer Graphics Forum*, pages 262–265, December 1989.
10. Howard, T. L. J. "PHIGS and PHIGS PLUS Tutorial." In *In Proceedings Eurographics*, June 1990.
11. Kleiss, J. A. and others. *Effect of Three-Dimensional Object Type and Density in Simulated Low-Level Flight.* Technical Report AFHRL-TR-88-66, Williams AFB, AZ: Operations Training Division, Air Force Human Resources Laboratory, May 1989.
12. Lewis, H. V. and J. J. Fallesen, "Human Factors Guidelines for Command and Control Systems: Battlefield and Decision Graphic Guidelines." Research Product 89-01, Systems Research Laboratory, US Army Research Institute for the Behavioral and Social Sciences, Alexandria, VA, March 1989.
13. McGhee, R. B. and others, "An Inexpensive Real-Time nteractive Three-Dimensional Flight Simulation System." Summary Report NPS 52-87-034. Naval Postgraduate School, August 1987.
14. Mikel, Russel D. *3-D Application Study.* Technical Report RADC-TR-89-305, Griffis AFB, NY: ITT Research Institute, November 1989.

15. Olson, R. *Techniques to Enhance the Visual Realism of a Synthetic Environment Flight Simulator.* MS thesis, AFIT/GCS/ENG/91D-16, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1991.

16. Rome Air Development Center, Griffis AFB, NY. *SCENARIO User's/Operator's Manual,* June 1989.

17. Rumbaugh, J. and others. *Object-Oriented Modeling and Design.* Englewood Cliffs, NJ: Prentice Hall, 1991.

18. Silicon Graphics Inc., Mountain View, CA. *Silicon Graphics Iris Reference Manuals* (C edition Edition), 1991.

19. Simpson, D. J. *An Application of the Object Oriented Paradigm to a Flight Simulator.* MS thesis, AFIT/GCS/ENG/91D-22, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1991.

20. Southard, D. A. *Superworkstations for Terrain Visualization: A Prospectus.* Technical Report MTR 11080, Rome Laboratories, January 1991.

21. Wardin, C. L. *Battle management Visualization System.* MS thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1989.

22. Zyda, M. J. and others. "Flight Simulators for Under $100,000," *Computer Graphics and Applications*, *8*(1):19–27 (January 1988).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1991	Master's Thesis

4. TITLE AND SUBTITLE
DESIGN AND APPLICATION OF AN OBJECT ORIENTED GRAPHICAL DATABASE MANAGEMENT SYSTEM FOR SYNTHETIC ENVIRONMENTS

5. FUNDING NUMBERS

6. AUTHOR(S)
John A. Brunderman, Captain, USAF

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

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GA/ENG/91D-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Richard Slavinski
RL/COAA
Griffis AFB NY 13441-5700
DSN 587-7764

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

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

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This investigation deals with the development and application of a software system to manage and render three-dimensional synthetic environments for use in mission planning, battle management and low end flight simulation systems. The work focuses on the object-oriented design and implementation of the Graphical Database Management System (GDMS). This system provides the data structures, file formats and algorithms to manage and render hierarchical, three-dimensional, polygonal models. A DataBase Generation System (DBGen) was designed and implemented using GDMS. DBGen allows a user to orient, scale, move, delete and add multi-resolution objects to synthetic environments interactively. It also provides a fly-through capability for immediate feedback on the dynamic response of the database. The results indicate acceptable performance for GDMS, while DBGen proved to be an effective tool for placing 3-D objects on terrain.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Graphical Database, Synthetic Environment, Virtual World, Graphic Workstations, Battle Management, Mission Planning, Flight Simulation, Object Oriented	107
	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

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to **stay within the lines to meet optical scanning requirements.**

Block 1. Agency Use Only (Leave Blank)

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency. Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ..., To be published in .... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availablity Statement. Denote public availability or limitation. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR)

DOD - See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE - See authorities
NASA - See Handbook NHB 2200.2.
NTIS - Leave blank.

Block 12b. Distribution Code.

DOD - DOD - Leave blank
DOE - DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports
NASA - NASA - Leave blank
NTIS - NTIS - Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.