COMPUTER IMAGE GENERATION

by
Michael Fortin
and
Olen Atkins

FLIGHT AND GROUND VEHICLE SIMULATION UPDATE - 2015

Table of Contents

COMPUTER IMAGE GENERATION

CAE Medallion

1. INTRODUCTION

The purpose of these notes, and the associated presentation, is to expose system users, and potential users to the characteristics of computer image generation (CIG) technology as they apply to real-time simulation for training, engineering and research purposes. It is hoped that this information will equip the user with an understanding of the existing and potential capabilities of CIG systems so that better use may be made of a current system and/or a more knowledgeable decision process undertaken in any future acquisitions.

We will not extend into the detailed system architecture of any current systems, or an in-depth survey of available products. Competitive considerations often limit an in-depth knowledge of a large number of systems. Rather than slight or misrepresent any of the available systems we will consider various architectural approaches and their resultant capabilities in a more generic sense.

Most system application examples and illustrations will tend to be aviation oriented, primarily due to the author's experience and, in fact, a large percentage of the CIG industry's collective experience, and expenditure. As we will see, however, this bias toward aviation is rapidly changing and an ever larger number of non-aviation applications are rapidly appearing. This is particularly true of the many new applications that have been facilitated by the ongoing surge in personal computer (PC) based graphics capabilities. Some of these will be discussed as well.

Throughout the presentation an attempt will be made to cover new techniques, new technology and future trends in the field of computer image generation. Please bear in mind, however, that what the industry will do next is even harder to determine than what it is doing currently.

1.1 What is CIG?

Strictly, computer image generation is any process that uses a computer to produce an image. Many different types of computer imagery are being used in an ever growing number of applications all around us. There is a wide and quickly growing range of software products and tools aimed at taking advantage of the graphics capability of PCs that include powerful graphics processor units (GPUs), often referred to as “game cards”.

One of the earliest applications of computers to create images was the computer aided design and computer aided manufacturing (CAD/CAM) area. These tools and workstations developed from wire frame imagery into very powerful, full color solid graphics with shading, texture and hidden surface removal. Computer Based Training has evolved into a science of its own, making use of CIG at all levels of the education process. Imagery from non real-time frame buffer type systems, which create imagery for the sake of imagery, is more and more evident in advertisements and other features on television as well as special effects for motion pictures. Also, computer imagery is well on its way to being a legitimate art form in its own right.

While all of these forms of CIG have things in common with what we will call a visual system or, more properly, an image generator (IG), they are generally not suited for use in immersive simulation applications. We will restrict ourselves primarily to the discussion of CIG systems that can take a position and attitude input from an operator (e.g. pilot) and compute and display a perspective image of a 3D database as it would be viewed from that location. The operator can then perform his task (e.g. flying the plane) by referencing that visual image. This is referred to as "man in the loop simulation". Further, to make use of the device meaningful, the operator must be able to perform that task in the same or similar manner as in the actual vehicle or device being simulated. This is referred to as “transfer of training”. A typical example is the aircraft simulator pilot who, upon visually sensing a change in altitude or attitude, will subconsciously correct the flight path with no other input than the visual scene (i.e. not looking at the flight instruments).

The two main issues here are: 1) how complex is the scene (i.e. system capacity) and 2) how fast will the system compute and display the new image (i.e. update rate). In the above example, if the update rate is too slow the pilot will be late in recognizing the attitude problem and entering the appropriate correction, and late again in taking it out. An even greater counter correction will then be needed, possibly resulting in pilot induced oscillation (PIO). Similarly, if the scene lacks sufficient detail, he may not see that a correction is required until much later than he would in a real world situation. This is generally due to a lack of height and speed cues in the visual environment. As we will see, these two related questions - how much?/how fast? - are the key to all real-time CIG applications, but they are not the only issues which must be considered.

1.1.1 The Image Generator

Image generator (IG) is the term generally used to describe the hardware and associated software that creates the image. This is almost never a pure computer in the historical sense of the word. Even modern general purpose computers are too slow to perform all of the calculations required for a complex scene at the rates required for a smooth representation of motion, without special purpose hardware of some type. Most IGs will include one or more computers or microprocessors that are combined with specialized, high speed, digital graphics processors to produce the images. In some systems this special purpose hardware is itself a computer of sorts. In the large, purpose built image generators all of the functions were carried out in hardware custom designed for the task. In the current trend toward PC based systems the special purpose hardware resides on the graphics boards, and their incorporated GPU chips.

It is the amount, type and performance of this special purpose hardware - be it custom VLSI, off the shelf processors, or commodity game cards - that will tend to determine the cost of the IG and its suitability for a given simulation application.

1.1.2 The IG and the Visual System

The image generator makes up a large portion of any CIG visual system. So much so, in fact, that the system is often referred to by the brand name of the IG. You will see or hear that a simulator has a CAE Medallion visual system or an ESIG 5500 visual system. There are, however, many other contributors to the visual system. In particular, the display device(s) are a major factor in the overall application and may be entirely independent of which IG has been applied. The database, although closely related to the IG, will have many independent aspects of its own. This is the reason that a given image generator type may be used in many varied applications. Other areas include the visual control functions resident in the simulator host computer and the interface between the host and the IG. Beautiful scenes from IG are of no use if they do not respond to the user's inputs correctly. More mundane features such as system diagnostics and utilities, and database modeling tools, are also important considerations in the long term support of a visual system.

1.2 CIG History

In an historical sense, the first applications of CIG in the mid to late 60's tended to be in the areas of engineering and research. This had more to do with the high cost and low performance of these early systems than a lack of interest from the training community. It is also of interest that these early devices were considered "daylight" systems although some of them were monochrome (i.e. black and white).

It was not until the early 70's, with the advent of the lower cost dusk/night systems, that CIG equipped simulators began to be used as training devices in large numbers. Compared to the closed circuit television (CCTV) model board and film projection systems widely used at the time, these systems offered low purchase and life cycle cost, relatively unlimited motion envelopes and very precise and perspectively accurate night/dusk images with calligraphic light points making up much of the image content. Over the period of a few years they almost totally replaced the earlier technology in most civil and some military training applications. Some of these early CIG systems are still in use today although customer expectations, market pressures and regulatory requirements have driven the technology onward at a rapid pace.

In the late 70's these dusk/night systems evolved into day/dusk/night systems, still with calligraphic capabilities. Throughout this evolutionary process the original "big" CIG systems continued to grow as well. These tended to be built and utilized in small numbers until the early 80's when higher system performance, lower system cost and higher real world training cost (i.e. fuel cost) all came together to make larger numbers of these devices affordable to meet demanding military training requirements.

In the early ‘90s a growing number of smaller systems appeared. Some of these had their roots in CAD/CAM or workstation technology and others were designed from the ground up as image generators. As these systems became more powerful and utilized better and smaller dedicated graphics hardware (much of it VLSI), their performance became more and more suitable for general simulation tasks. These systems tended to be raster only (i.e. non-calligraphic), low cost (relative to the “big” systems) and very strong in selected capabilities. The lower cost of these devices was partly due to the much higher manufacturing volume allowed by the many applications, other than simulation, for these more general purpose devices. The downside of this benefit was that the systems included a large number of features not suitable to, or needed for simulation. It was also much more difficult to coerce these manufacturers into incorporating simulation features as we were a smaller percentage of their overall market.

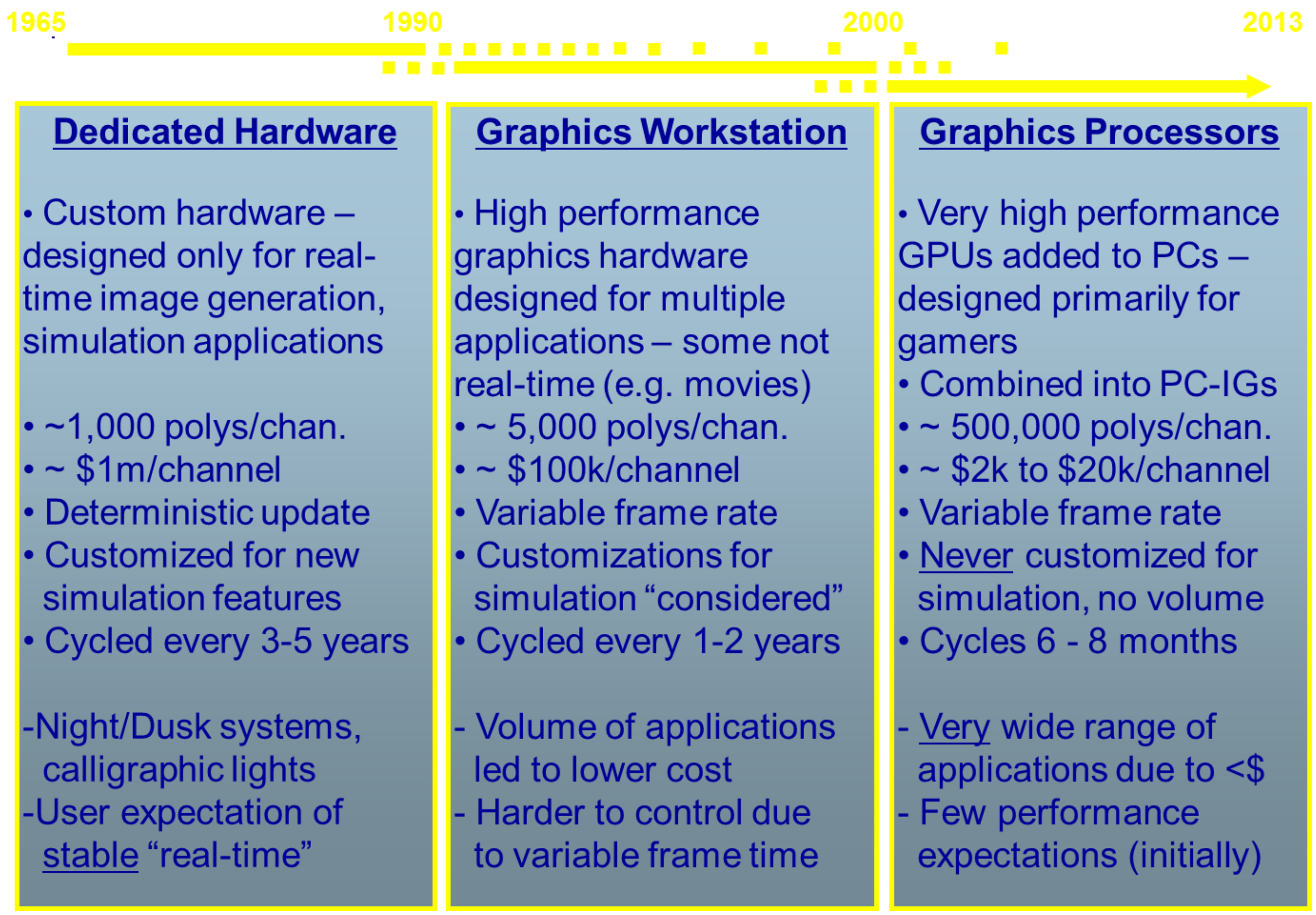
More recently an even lower cost range of products has become available. These are generally referred to as “PC graphics” or “PC-IGs”. These systems offer amazing imagery at even more amazingly low costs and have quickly found their way into “serious” simulation. The graphics accelerator boards for the PC have been developed almost exclusively for the home gaming market. Whereas the dedicated IG manufactures were happy to sell 100 systems a year, and the general purpose graphics computer people to sell 1,000 a year, the creators of the graphics accelerator boards expect to sell <u>millions</u> a year. This volume again translates into phenomenally low cost but means that the simulation industry has little significant input for dedicated features. In order to take advantage of this cost and performance the industry must find ways of adapting this technology to its very demanding specifications without compromising either.

When PC-IGs first became available they were found to be lacking performance in a number of areas that generally made them unsuitable for “serious” simulation applications, regardless of the very low costs. The graphics hardware was generally capable of a large number of polygons, many more than larger systems of the time. Where they were found lacking was in the pixel processing areas. This meant low resolution images with little or no anti-aliasing. The resulting poor image quality was “good enough” for the less indiscriminating home game market whose users had never been exposed to the image quantity and quality provided by dedicated image generation hardware. Each subsequent generation of the PC graphics cards has provided better and better performance.

After several generations of graphics accelerator performance improvements the developers of this hardware began to see improved image quality as a discriminator in this very competitive market. It was at this point that these systems began to be candidates for sophisticated simulation applications. There are still some missing elements in the current systems but various IG vendors are finding ways to work around these limits. This is generally accomplished by adding special purpose hardware (again) to make the low cost PC based system into a true “image generator”. Another area where progress prepared the technology for real-time simulation was the operating systems. The PC-IG systems normally use either Windows or Linux. Originally neither of these operating systems was able to sustain a <u>stable</u> update rate. Various utilities (e.g. disk access) would interrupt the efficiency required for steady performance and the system would glitch slightly. This was another area that the home gaming market was tolerant of, but full up simulation applications were not.

The latest technology leap in PC graphics boards is the addition of vertex and pixel/fragment shaders. Shaders are small programs that are loaded directly to the graphics card. These programs are called from the database to perform very specific functions. The vertex shaders manipulate polygon vertexes within the scene while the pixel/fragment shaders can determine how the pixels covered by the polygon are rendered. Shaders are discussed more in 4.10, including some of the potential pros and cons of this emerging technology.

As a result of the PC-IG progress the potential user/purchaser of a CIG device has a very wide range of systems to choose from. They include the PC systems at less than $10,000 per channel, the more complex workstation based systems in the $100,000 or less price range, to the specialized top of the line systems (some with calligraphic lights) for as much as $1 million or more. How to decide?



Dedicated Hardware

• Custom hardware – designed only for real-time image generation, simulation applications

• ~1,000 polys/chan.
• ~ $1m/channel
• Deterministic update
• Customized for new simulation features
• Cycled every 3-5 years

-Night/Dusk systems, calligraphic lights
-User expectation of stable "real-time"

Graphics Workstation

• High performance graphics hardware designed for multiple applications – some not real-time (e.g. movies)
• ~ 5,000 polys/chan.
• ~ $100k/channel
• Variable frame rate
• Customizations for simulation "considered"
• Cycled every 1-2 years

- Volume of applications led to lower cost
- Harder to control due to variable frame time

Graphics Processors

• Very high performance GPUs added to PCs – designed primarily for gamers
• Combined into PC-IGs
• ~ 500,000 polys/chan.
• ~ $2k to $20k/channel
• Variable frame rate
• Never customized for simulation, no volume
• Cycles 6 - 8 months

- Very wide range of applications due to <$
- Few performance expectations (initially)

fig. 1-1 – Image Generator evolution and approximate time-line

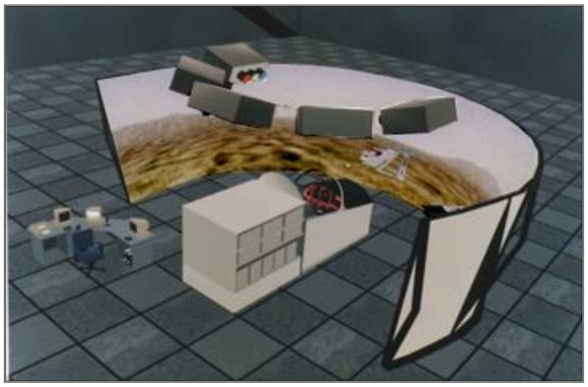
In the ideal simulation world, led by pilots and engineers, the big, top end systems that are the most powerful, are the obvious choice. In the real simulation world, however, adherence to budgets and fiscal prudence are generally expected if not required. The end result is compromise. The first step in a workable compromise is to thoroughly understand the requirements of the device for the intended application.

1.3 Applications

Historically, the largest single application of CIG has been aviation oriented simulators. There may not be more aircraft visual systems than any other type, but it can certainly be said that more money and development effort has been applied in this area. The annual survey of civil airline simulators by Flight International magazine lists well over 650 CIG systems in use around the world. There are undoubtedly more than this in the many and varied military applications world wide.

1.3.1 Military

Hughes

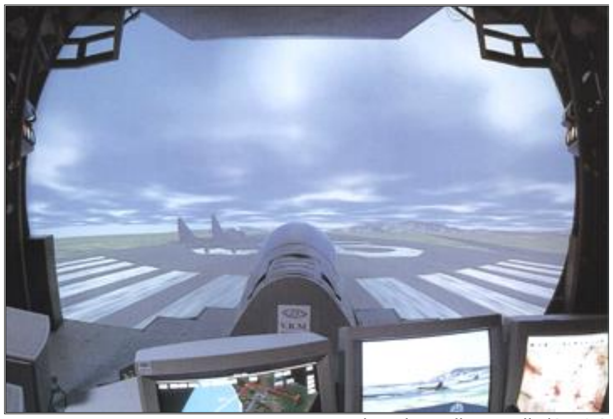

Hughes

The military, due to its varied and unique training requirements, tends to acquire the most complex and expensive CIG equipment available. The top end of these systems are called Weapons Systems Trainers (WSTs) or Weapons Tactics Trainers (WTTs), which are intended to train the student in the all of the many different aspects of combat flying in his or her particular aircraft. In addition to basic takeoff and landing type skills, these may also include air-to-air, air-to-ground, various sensors, low level navigation and attack, and in some cases all of these combined.

In addition to being a very complex image generator and database problem, the need for a very large field of view (up to 360° x 200°) can make the display configuration extremely complex. Some tuypes of display require that the image generator pre-distort the image so that it will appear correctly on the curved surface of a dome or screen. This can be very impactive on the IG hardware and wherever possible is performed in the projection devices or in post processors between the IG and the projectors.

The databases for WST and WTT simulators tend to be very large geographic areas, and include enough other aircraft, ground targets and threats that tactics as well as basic weapons delivery can be practiced. These databases generally represent some real-world area and are developed using data from sources such as the National Geospatial-intelligence Agency (NGA), or some similar organization. Correlation with various navigation, radar and optical sensor simulations is also often required. The impact of these requirements will be discussed below.

Thomson/Hughes


Virtual Reality Media/SEOS

Another major type of military simulator is the Operational Flight Trainer (OFT). These are less complex systems, typically with smaller field of view and database requirements. The main purpose of these systems is to train the student to fly the aircraft, primarily in the landing and takeoff modes, as well as some forms of navigation. Since this can include confined area landing sites and small ships for helicopters, or aircraft carriers and tactical airstrips for fixed wing aircraft, the problems can still be complex. The databases to satisfy these requirements may be small in area but must contain a high degree of detail to provide sufficient height and speed cues for a relatively inexperienced pilot to learn how to fly the airplane.

A much more general simulator category is the Part Task Trainer (PTT) that is intended to train the student in a limited portion of his mission. This could involve a single avionics system such as a heads up display, infrared sensor, a single weapon such as a Maverick missile, or a limited flight envelope such as in-flight refueling. Alternatively, the PTT may be designed to train one member of a mufti person crew in his specific duties. Although many and varied, these systems tend to have small fields of view, often a single channel, and relatively simple image generators and databases. Since the object is to teach the operation of a single system or task, the emphasis may be on flexibility rather than scene quantity and image detail. Real time operation may not be a requirement and in the past many of these systems used non CIG devices such as videodisk for their image source. This is also the category where most non-aviation simulators fall. These include tank driver and tank gunnery simulators as well as some maritime applications such as ship handling and periscope simulators.

In recent years there has been a great deal of interest in large numbers of low fidelity simulators networked together to provide team training. This is referred to as Distributed Interactive Simulation (DIS) because the computing power to supply these complex capabilities is distributed throughout the network rather than one large central processor. The prototype of this type of simulation was the SIMNET program, sponsored by the U.S. Army and DARPA (now ARPA), which networked a large number of tank and infantry fighting vehicle simulators. The network makes use of satellites as well as land lines to link units at different bases, or even on different contents. The individual simulators and visual systems may be relatively basic in performance since they are not intended to teach fundamental driving or gunnery skills. They are designed to teach a large number of tank crews to operate as a team against an even larger number of computer driven opposing armored vehicles. The opposing tanks are provided by another part of the system referred to as Semi-Automated Forces (SAFORS).

The technology breakthroughs with this type of device include the network itself which must operate literally around the world, and the need to provide sufficient fidelity at the low cost per unit demanded by the large number of total units. Since the first phase of SIMNET was considered a success, the Army developed a much improved system called the Close Combat Tactical Trainer (CCTT) that has additional capabilities. More recently the aviation portion of the combat environment was added with the AVCATT-A system that supports multiple helicopter types. The U.S. Navy and Air Force have recently established similar systems for F-15, F-16, F-18, and other aircraft. Many other services in the world are closely monitoring these programs and have plans of their own for similar systems.

ECC/E&S

Another recent military application of simulators that is often intended or required is mission rehearsal. The first of these programs was the U.S.A.F. Special Operations Forces Aircrew Training System (SOFATS). This capability has been discussed for many years, but the technology is now becoming available to make it a reality. In addition to the very high scene capacity required to represent a given mission scenario, the means must also be available to quickly create or update a database from current intelligence data. This implies, at the very least, sophisticated database architectures, protocols and tools that will allow the required database changes to be made with little or no user input (i.e. semi-automatically). This requirement is also having an effect on the basic architecture of some image generators themselves. The only way to build, or even update, a database in as little as 48 hours is to develop methods of storing the source data (i.e. NDA data, satellite imagery, a library of feature models, etc.) on the system disk so that the image generator can "assemble" the database in real-time.

1.3.2 Engineering and Research

Engineering simulators (and their visual systems) may be purchased (or developed) by the manufacturers of airframes or large subsystems for aircraft or other vehicles. They aid in the design and evaluation of various aspects of vehicle handling and performance during the development phase. These are normally large systems, similar to the WSTs discussed above. This is because they require a great deal of flexibility as well as high scene content. Also, they are purchased with aircraft development budgets rather than training budgets. The databases for these systems may be large areas to support long navigation and weapons delivery test flights, but will also contain small areas of high scene detail designed to test specific features of the aircraft. These databases and some of the IG software will often be designed and/or modified by the user activity in order to support various tests which may be required. Relatively little is known publicly about the utilization of these systems as they tend to be applied to proprietary and/or classified projects within the user organizations.

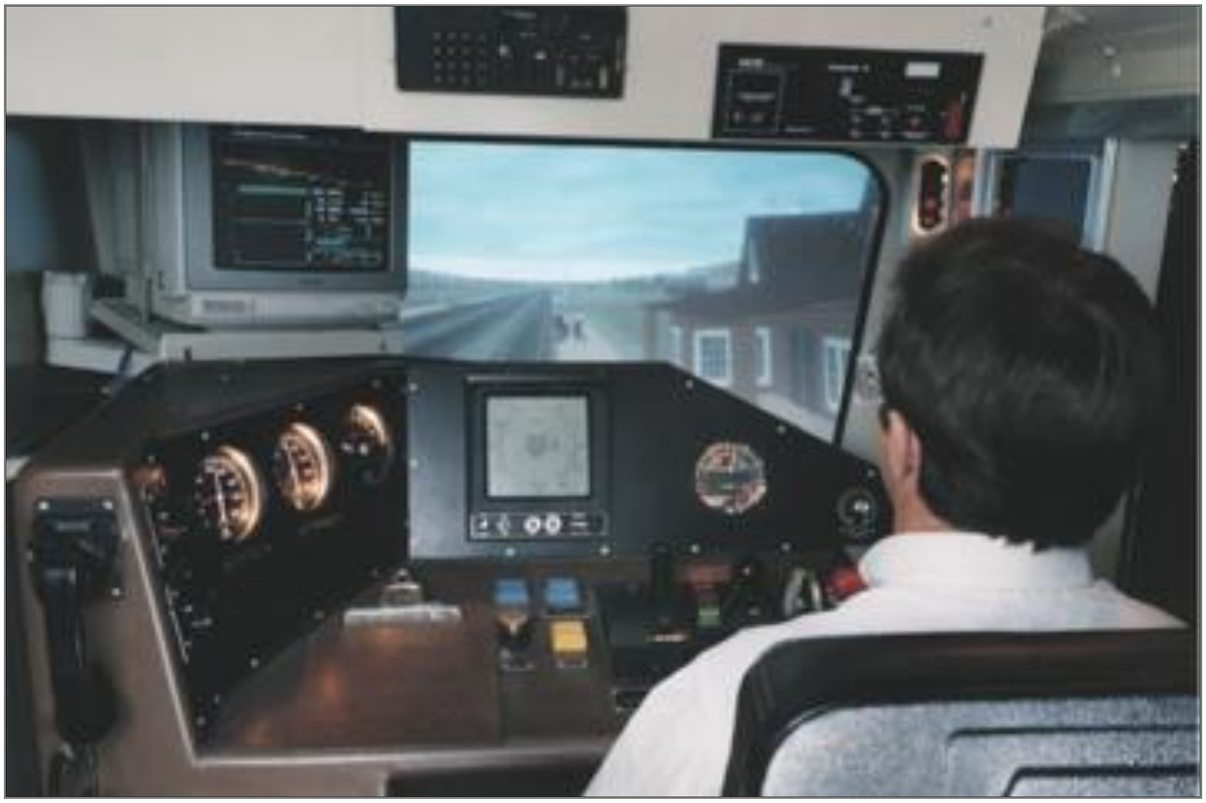
Research simulators are generally owned and operated by large government organizations, such as the Air Force Human Resources Laboratory at the ex-Williams AFB, the Navy Training Systems Center in Orlando, Fla. NASA, or the FAA. They may also be owned by universities or other organizations doing human performance related research. The CIG systems on these simulators are used to task load the operator so that his performance with a certain configuration of the vehicle and/or environment can be evaluated. Some applications test the effectiveness of portions of a simulator, such as the display device or control loading system, while others may attempt to evaluate what type of information the operator may need for a specific maneuver or task. Many types of systems are used in this work. They do need to be flexible, however, since they will be put to many different uses, often in parallel.

1.3.3 Civil or Commercial Simulators

Flight Safety

Civil airlines operate the most concise categories of simulators and visual systems. This is, of course, due to the regulatory authorities such as the Federal Aviation Administration, and similar organizations around the world, that dictate what type of training the airlines must do. The approval from these organizations allows the airline to do some or all of their training in a simulator as opposed to the actual aircraft, thereby greatly reducing the cost of the training. This economy made a great contribution to the rapid progress of simulator technology in the early days of CGI technology, and continues to be a factor today.

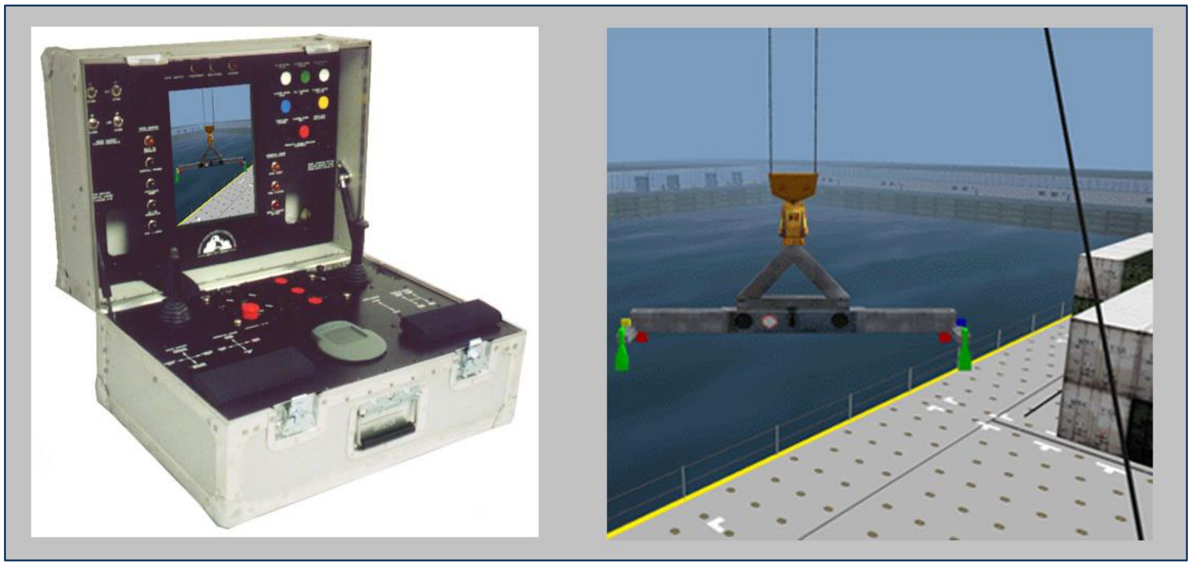
In the late 70s and early 80s the FAA has allowed more and more of the required training to be done in the simulator, assuming the equipment was up to an appropriate standard. This process, aggravated by rising fuel cost, made better and better simulators and visual systems affordable to the airlines as they became available on the market. In June, 1980 the FAA published the Advanced Simulation Plan (originally FAR 121), which allowed an airline with certified equipment and training syllabus to train a pilot new to the company (although not necessarily a new pilot) entirely in the simulator so that the first time he flew his assigned aircraft there were passengers aboard. This level of training and equipment was originally called FAA Phase III, but is now referred to as Level D. There is also a Phase I (Level B) and Phase II (Level C) which allow recurrent training and conversion and upgrade training to be accomplished in the simulator.
.
The FAA regulation was supplemented in January 1983 by Advisory Circular 120-40 that covered FAR 121 operators (i.e. airlines) as well as FAR 135 operators (i.e. regional airlines). This circular was modified to AC 120-40A in July 1986. AC 120-40B further defined some of the acceptance tests for simulators and visual systems, and the regulation continues to be updated today. Many other countries have developed similar regulations. In general, any airline flying into a country must submit their training plans and equipment to approval by that country's regulating authority. Systems acquired by the military services are sometimes required to meet FAA Level D, making use of the detailed requirements as a system performance specification.



IITRI/SGI/Hughes

fig. 1-2 - Locomotive CIG system

The ongoing spiral of reduced hardware costs, with the same or greater performance, has allowed CIG systems to find their way into more and more commercial training applications. Some of these new customers previously used other visual devices (e.g. video disk based systems), or had no simulators and/or visual systems. In the latter case training was performed on the actual equipment, sometimes at great cost or risk. Fig. 1-2 shows a CIG system applied to a locomotive simulator. In general, any system or vehicle that is operated visually (partially or in total) is a candidate for visual simulation using CIG techniques.



Jason Associates

fig. 1-3 – Low cost portable crane simulator

The simulator shown in fig. 1-3 is another example of a visually operated device that has benefited from CIG based training. This is a portable simulator designed to train the operators of large cranes. Here again, recent developments have allowed a small system (cost and size) to create relatively complex images that provide useful training. In addition to a dense database with dynamic object priority, the system also makes use of real-time "shadow" effects in order to judge the height of the crane's hook above the ground and other features.

1.3.4 Entertainment

This is undoubtedly the major application of "real-time" computer graphics today. As mentioned earlier, devices in this category are driving the technology that the simulation industry has chosen to make use of, for the most part. This market originally took the form of games to be played either on a personal computer, with graphics accelerators, or on a console box that made use of the home TV as the system display. These applications have now grown to be multi-player, or even many players across the internet.

While the imagery on the display may be very similar to what one expects to see on a simulation device used for training, the motivation and technique behind that imagery is generally somewhat different. The general approach to game system is to make the entertainment factor the highest priority in the design of the system. While this is understandable, it does cause some artifacts that are not typically tolerated, or even allowed in simulation for training purposes.

One such area is the system stability with regard to update rate. The industries history of special purpose devices, developed exclusively for real-time simulation, instilled an expectation of a very stable system with no glitches or jumps. The collective experience of the game market has been continually improved performance as the GPU hardware increased in capability. The glitches and jumps that they experience during game play are getting fewer and less severe with each new generation of hardware. The simulation industry and users have always insisted on no glitches and have had to adapt the PC and graphics technology to achieve that goal.

Another area where major differences are seen is in the database design and content. Again, the game systems are in the entertainment business and create their environments to achieve that purpose. For the most part, they are also able to determine the size and content of the database. That means they can make choices that not only provide entertainment to the user, but also optimize the performance of the IG to provide as much content as possible. One way of doing this is to have very small databases so that the density of features can be very high, and so that system resources are not spent loading new areas from the system disk. The Database Engineers working in simulation are forced to create very large areas and must include content that matches the real-world as closely as possible.

For better or worse the simulation industry has tied itself to the products developed for the gaming market. While this continues to be a good thing, more or less, it does make us dependent on hardware that continues to be developed for a different application. While the differences between the gaming application and simulation are manageable today, that may not always be the case. In recent years many of the makers of the large, special purpose graphics devices that simulation cut its teeth on have moved on to other types of hardware, or gone out of business. If the gaming interest and technology heads in a different direction the suppliers of simulation devices may find themselves scrambling to find something else that will satisfy their requirements.

EAT SLEEP PLAY

2. THEORY AND ARCHITECTURE

As was stated in the previous section, there are many different CIG systems available today. They all differ somewhat in their approach to solving the difficult problem of creating a complex scene as rapidly as possible. We will not attempt to analyze all of the different system architectures but will look at a symplified system block diagram (fig. 2-1) to explain the processes that must be performed.

fig. 2-1 - Typical Image Generator Architecture

The components of the typical CIG system which are described below are designed to process the visual database as efficiently as possible. We will discuss the many features and capabilities which are included in the database in greater detail in subsequent chapters. Bear in mind that the database normally includes a large (sometimes very large) amount of geometry data which defines the size and location of every visual feature that will appear in the resultant scene. It also contains the various controlling mechanisms which are used to create complex imagery.

2.1 Front End Computer

Every CIG device has some type of identifiable computer as the first step in its "pipe line". This has been true since the early days of the technology and is undoubtedly one of the reasons that the field is called computer image generation. Over the years, however, as the overall expectations for the system's performance increased, more and more of the tasks originally carried out in the computer's software were moved to special purpose hardware. Since this hardware was designed to perform these functions and nothing else, it was able to accomplish much more in the allowed time. Unlike the computer, however, which has to do everything in a serial mode, some aspects of the hardware can be paralleled so that even more iterations may be performed. In particular, the large number of repetitive mathematical calculations required to convert the three dimensional geometry of the database into two dimensional perspective values for eventual representation by the display was one of the first series of calculations designed into Geometric Processor hardware.

In general, the various functions and calculations which are better done in special hardware have been moved out of the computer, and tasks the computer does well have been left there. As we will see, this has had considerable effect on the size and type of the front end computer which is used in modern systems. As discussed above, the computer in current systems is generally a PC, which typically include high speed graphics processors. Even though these are in the same “box” the processes shown above and discussed below are conducted separately.

2.1.1 Host Interface

One of the essential tasks of the front end computer is the interface with the simulation device computer, generally referred to as the host. In some recent systems the host may be another CPU within the same system component. Each time the host computer cycles it sends a block of data to the visual system computer. The critical data in that block is the position (X, Y, and Z or latitude, longitude and altitude) and the attitude (heading, pitch and roll) of the eyepoint. The primary task of the visual system is to display a perspective image of the database environment as it should appear from that position and attitude. There may be data for several other aspects of the scene as well, such as visibility, ambient illumination and similar position and attitude data for one or more dynamic coordinate systems (DCSs) which may be moving in the scene relative to the database coordinate system (i.e. model origin).

In most early systems the host-to-visual interface was one way, with the visual receiving the data and processing the images at its own rate. Most current image generators, however, are capable of calculating data which is of use to the simulator (and the instructor and/or operator). The height of the eyepoint above the modeled terrain may be returned to the simulator to drive the radar altimeter, for example (see 4.5). Also, the visual may detect if a portion of the simulated vehicle contacts something in the database (see 4.6). The position of moving objects in the scene such as a missile and its target may be calculated in the visual and the results sent to the host for scoring purposes.

The hardware utilized for this interface has taken several different forms over the years. Common terminal interfaces such as RS232 were generally found to be too slow as system rates and the amount of data to be transferred both increased. Direct memory access (DMA) was utilized by many systems due to its high speed and the relatively small impact it has on either the host or the visual software. Many of the current systems make use of a standard Ethernet interface operated in a broadcast mode. Ethernet is very fast in this configuration, common to almost all of the computers and micro-processors currently being used for both the host and the visual computers, and is designed to handle variable length data blocks and two way communications.

In systems where the visual and host tasks are performed by separate CPUs, or groups of CPUs, the interface can be done through shared memory. There are several advantages to this configuration.

2.1.2 Extrapolation

Since the host computer and the visual computer have very different tasks to perform, it is not surprising that they may cycle at different rates. The host rate depends on the type of vehicle being simulated and in some cases different functions in the simulator will be calculated at different rates. The aerodynamic calculations from which the position data is derived is typically among the fastest, but may be anywhere from 15 hz. to 60 hz. Some very slow vehicles, such as ship simulators, may cycle as low as 1 to 2 hz. The visual systems on the other hand will normally cycle between 30 hz. and 60 hz., with some applications being as low as 5 to 10 hz. As discussed below, some visual systems will cycle at different rates depending on the ambient condition they are depicting. This difference in rates means, among other things, that when the host sends the current position and attitude data for the eyepoint and any dynamic models, the visual system will not be ready to make use of it, and when the visual systems is ready, the most recent data available will not be current.

If this situation is not accounted for the image will be calculated from an inaccurate position which may be erratic. The result will be a small but objectionable jitter of the image. This is particularly noticeable when the eyepoint is close to objects in the scene and moving at slow speed such as an aircraft taxiing around an airport. In the case of formation flying the problem is aggravated by both the eyepoint data and the position data for the other aircraft being erratic.

The method used to minimize this effect is for the front end computer to record the time at which the latest position data arrived from the host, compare this new data with that received in the previous two or three data blocks (which have been stored) and extrapolate a predicted position for the exact point in time when the visual system is ready to start calculating a new image. In the case where the simulator update rate is less than half of the visual rate, the visual would otherwise calculate two or more images with the same data, resulting in the same image.

One concern with the extrapolation process is its accuracy. Until recently it could be shown with analysis that the maximum error possible was very small. This was due in part to the fact that the controls on most vehicles would not provide a true step function. In the short time between even the slower updates the control surface of an aircraft, for example, was limited in how much it could change due to mechanical considerations. The fly by wire systems in some modern aircraft, particularly the high performance fighters, can create a step input which the extrapolation routines cannot account for as accurately.

2.1.3 Database Manager

Database management is the process of refreshing the database as the eyepoint moves through the gaming area. This process is omitted in some systems, done in software (in the Front End Computer) in others or may have major sections of the system hardware dedicated to it.

The total database may contain hundreds of thousands of polygons and lights representing thousands of square miles of terrain and many thousand objects (and a great deal of on-line memory). The database management process must determine which of those polygons may soon be in view, retrieve them from the system disk and pass them on to the geometric processor for further manipulation (see 3.2.2). Regardless of where the function is performed it will be closely integrated with the format and structure of the database and possibly with the disk storage as well.

With the availability of very large memory in recent systems it is possible to consider (again) retaining the entire database on-line. This assumes that the application requires a relatively contained gaming area and/or limited polygon density. There is still a need to cull the polygons before they are sent to the Geometric Processor (see below), but the requirement to page data from the system disk, and manage the on-line memory to receive it, might be eliminated.

Whether the task is done in software or hardware, it is difficult to speed it up through parallel processing. This is because of the hierarchical nature of most databases. Since the entire database structure must be evaluated, the work cannot be efficiently shared by additional processors

There are two basic management strategies. One partitions the database into objects of various sizes and calculates their distance from the eyepoint to determine if each one should be retrieved. A hierarchical structure is used so that as few range tests as possible are required. The other approach divides the database into a grid structure. The eyepoint position relative to the grid is used to determine which of the surrounding squares should be retrieved from the disk (see fig. 3-9 & 10).

2.1.3.1 Software

When the Front End Computer performs the management of the database the function must be considered a part of the system load. The sub-model switching discussed in 3.2.2 or simple implementations of a range based system are often considered part of the real-time program. These may work as a background task where the process will only examine as much of the database as it has time for in a cycle and continue where it left off during the next cycle. It may take several system cycles for the program to determine which scene elements are required from the disk. The performance of this type of database management must be understood and incorporated into the database design as one of its parameters.

2.1.3.2 Hardware

In systems which will require large quantities of data at very high rates, special purpose hardware may be incorporated which insures that the database management process can adequately refresh the active database. This hardware will consist mainly of arithmetic processors which can calculate the range from the eyepoint position to the many objects in the database and direct the appropriate ones to be retrieved from the system disk. The range calculations may also be used to order the objects, and consequently the polygons they contain, into the order required for the priority process.

2.1.3.3 Disk Interface

Some of the larger systems will bring the model data directly from the system disk into the database manager and/or the geometric processor. In these systems the database information is separated into the management portions which the front end processor or the database manager evaluates and the actual polygon and light data, which is routed to the geometric processor without being manipulated in the front end computer. In this configuration, the database management portions of the model effectively become pointers which, when found to be active, activate the appropriate portion of the visual database. The advantage of this implementation is that the resultant data can be transferred at much higher speed and with less impact on the remainder of the system if the data path is direct to the point where the information will be used.

2.1.4 Mass Media Storage and Interface

In the distant past the system software and databases were stored on tape or even punch cards. Later, large disk platters, or combinations of platters were used as databases got larger, in both geographic area and content. The magnitude of the data and the speed of these disk units sometimes required complex storage strategies in order to minimize the time required to access the database.

Today’s systems make use of the large commercially available disks that are available with the PCs that form the bulk of the system. As we will discuss below, many of the larger systems consists of more than one PC, and in some cases a very large number of computers. Some system architectures, and/or their operating systems, may require that every PC in the system have a copy of the software and database in local storage. In a large system this can be very cumbersome when it comes to updating the software or the database. If the system is being used in a classified facility it may be necessary to frequently remove the disk units from the system for security reasons. Other systems make use of a single disk unit in one PC and then distribute the software and database content over the internal network.

2.1.5 System Management Task

One of the more critical functions of the visual computer is the overall management of the image generator. This includes the monitoring of the load in the various processing centers of the system and implementing the appropriate form of overload management when and where required. It is essential for several reasons, not the least of which is the health and sanity of the observer, that all of the system's channels operate at the same rate regardless of their load.

Other less critical system tasks may include the monitoring of special effects, such as flashing lights and strings of strobe lights, so that they will be consistent in all channels and amongst themselves. Also, the movement of objects (DCSs) in the scene may be performed in the visual computer as opposed to the host.

The paths for these objects may be defined in the database.

2.1.6 Utilities

In addition to the real-time visual programs, the computer may also handle utility programs critical to the maintenance and support of the system. These include a powerful diagnostic program which will test the various boards and assemblies in the hardware. Data is sent to a circuit board or components on a board, and the returning data compared with expected results. When the results differ from those expected, further tests are run and/or messages are sent to the system terminal for interpretation by maintenance personnel. In some cases, the technician may manipulate the test and monitor the results, to further trouble shoot a problem.

With the advent of the PC based systems most diagnostics of this type have pretty much gone away. In the current systems the main "diagnostic" is that some or all of a system channel is not working, i.e. there is no image. In this case the entire PC is generally replaced and may or may not be repaired as a stand along, off-line activity.

Database generation tools are another important utility which may be available in the visual computer. These will be discussed in section 6.3. Data transfer utilities should also be available to move data from the tape media to the system disk and vice versa. Also, of course, any of the standard utilities which come with the computer operating system should be available for off-line task which may have nothing to do with the simulation task.

2.1.7 Minicomputers, Microprocessors

Historically the front end computer for a CIG system was a "name brand" minicomputer. These tended to be a significant portion of the total system cost and floor space. The size and capacity of these machines was generally matched to the image generator, with the larger systems requiring a larger minicomputer. Despite the wide range of computers available on the market for the system designers to choose from, the performance of the computer was seldom a perfect match to the rest of the system. Normally the minicomputer would include capabilities and utilities which were wasted in the CIG system.

In later years manufacturers built their own front end computers or used commercially available single board computers. By combining one or more of the commercially available microprocessors with an appropriate size and type of memory and other processors ("number crunchers") they were able to customize a front end computer which much more closely matched the requirements of the total CIG system. This approach also provided modularity and growth flexibility to the system architecture. More computer boards could be added to handle additional tasks. Also, when a better version of the microprocessors became available it was possible to substitute it into the system to provide increased performance, assuming the original bus structure connecting the memories and processors could handle the increased data traffic.

As stated above, today's systems make use of the CPU that is native to the PC itself. In a small system there may only be one PC, and therefore one, maybe two, CPUs. In these cases the CPU(s) must perform all of the tasks required of the system front end computer. In larger system with many PCs the tasks are generally distributed throughout the system and the processing requirements shared.

2.2 Geometric Processor

The basic function of the geometric processor (GP) is to convert the various three dimensional database elements into a two dimensional perspective image as seen from the current eyepoint and at the current attitude. The bulk of this task involves performing the matrix mathematics required to convert the three dimensional (X,Y,Z) position of every vertice (see 5.2) into the X,Y and depth information needed for the perspective image. The scene is calculated for a flat image plane within the GP. The number of these calculations which can be performed in one field time or system iteration is directly related to the overall capacity of the system. Increases in this performance have been accomplished by utilizing faster and faster processors as they became available, or by creating several parallel data paths, or both.

In the wide range of PC graphics accelerators available a major performance issue is how much of these perspective geometry calculations are done on the graphics card as opposed to the PC processor (usually a high power Pentium). If the calculations are left in the PC there will be a trade off with the other software application processes. If the graphics card does the calculation (as well as the pixel fill below) the cost of the card will be higher. Even in the worst case, however, this cost is a fraction of traditional workstations and image generators.

In order to limit the number of perspective calculations to the absolute minimum required for the accurate rendering of the database, the GP also performs several additional culling processes (fig. 2-2). Backfaced polygons, those facing away from the viewer, are discarded. This normally involves a quick test which compares a vector from the eyepoint (viewing vector) with a vector which is perpendicular to each polygon (usually stored with the polygon). Another quick examination of the vertices of the polygon or the ends of the light strings will determine if the polygon or string is outside the field of view. A less efficient culling mechanism, but still useful, will discard polygons which are too perspectively small to be seen.

Another task normally performed in the GP is polygon clipping. When the above field of view test finds that some of the vertices are outside or that all are outside, but out different sides, the polygons and light strings will have to be "clipped" at the boundaries. This involves creating artificial edges for the polygons and end points for the light strings. This is one of the more computationally intensive tasks for the GP. There has been a lot of system design effort devoted to more efficient clipping algorithms and/or IG architectures which reduce or eliminate the need for clipping.

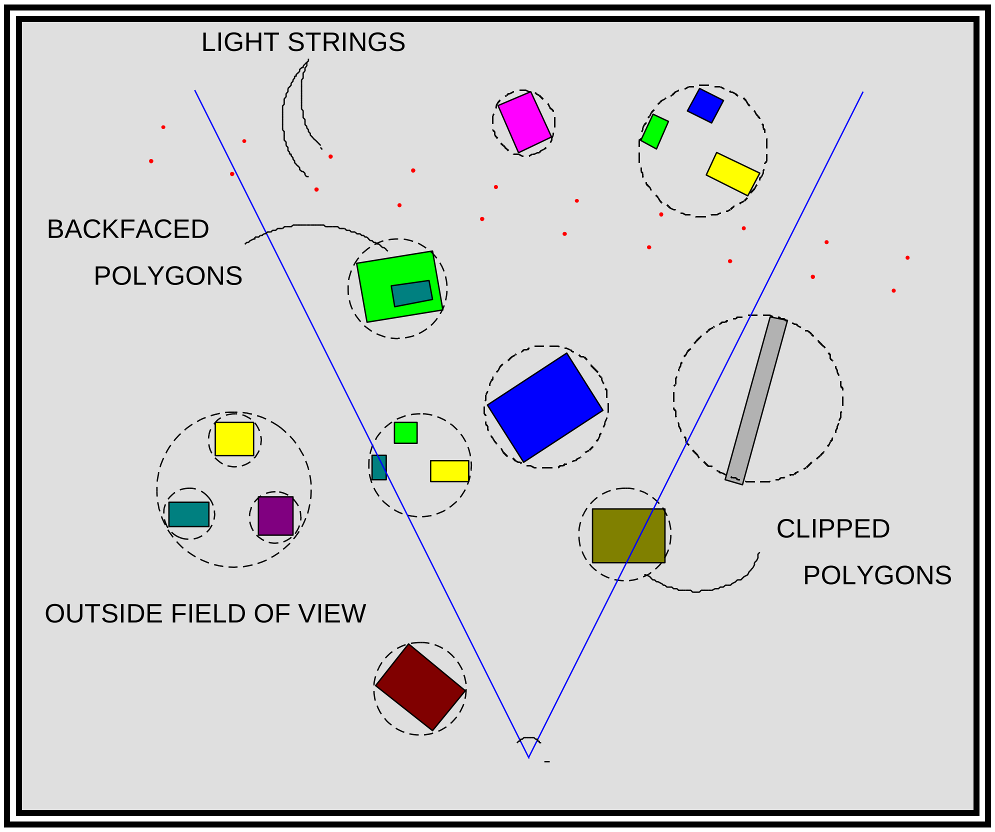


fig. 2-2 - Geometry Processor Functions

Lights are normally stored in a string format which includes the coordinates for the first point, the last point and the number of lights to be evenly distributed between these points (see 5.4). The expansion of this data into individual light points is performed in the geometric processor. Light strings must also be clipped in that the GP will create the appropriate beginning and/or end point for any string which extends out of the field of view. Other light point related tasks which may be included in the GP are a same point test which removes lights that occupy the same position on the display (important for calligraphic lights), light point directionality and flashing or rotating lights.

2.3 Display Processor

The display processor (DP) is typically the largest portion of the image generator. It performs many of the functions which determine how the final image will appear on the display. The basic task of the DP is to take the perspective image of each polygon and light as they come from the GP and map them into the appropriate pixels. This data is then converted into analog signals which drive the display device.

A typical CIG image is made up of picture elements, or pixels, arranged in raster lines. A system will be configured to produce a fixed number of pixels. This is the smallest element of color in the picture. It is analogous to the black dots which become visible when a newspaper photograph is enlarged. Although there are other factors, the number of pixels which the DP produces relates directly to the resolution of the entire system (see 3.3). In some systems the number of pixels can be configured to match a wide range of display devices. Early image generators used the same output formats as conventional television (525 lines of 640 pixels), but more recent systems will provide millions of pixels.

Many older systems used an interlaced format which meant that only half of the raster lines (and pixels) were calculated and displayed each field. Assuming the update rate was fast enough, the human eye could merge the two fields into a relatively stable image. The total of the odd and even fields is called a frame. The number of pixels was normally specified per frame and, combined with anti-aliasing (see 4.1) and limitations imposed by the displays and optics, determined the resolution of the system.

Newer systems can produce a millions of pixels in a non-interlaced format. This configuration provides a more stable image without some of the aliasing like artifacts imposed by the interlace process. A growing number of display devices which can take advantage of these advancements are also available.

Another aspect of a system's pixel resolution which must be considered is the average pixel fill rate. This relates to the number of times that the DP can address (or "touch") each pixel during one field. As each polygon is mapped into the raster structure the pixels which they cover must be calculated, examined and/or filled. Depending on the priority mechanism and other system architecture issues there will be a limit on the total number of touches which can be performed. This value is normally specified as pixels per second and must be divided by the number of pixels in each image and the update rate to determine the "pixel touch ratio", "over-write limit" or "depth complexity limit". For example:

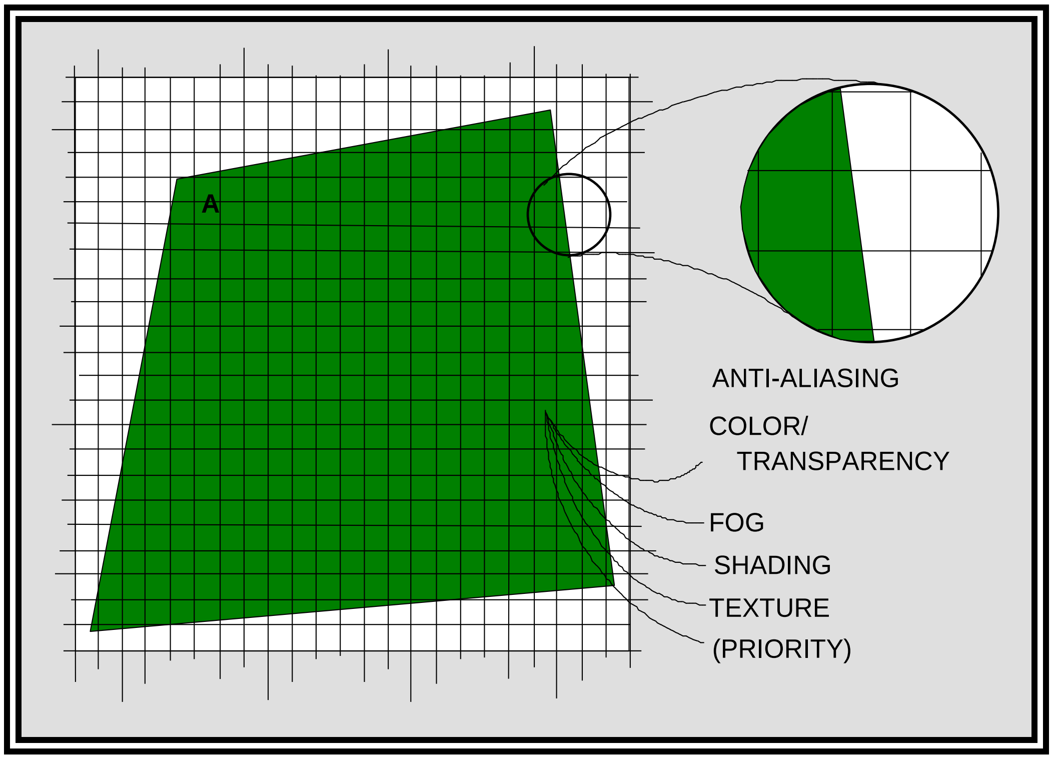


fig. 2-3 - Per Pixel Functions in the Display Processor

(1280 x 1024 = 1.3m pixels) x 5 hits/pix. = 6.5m pix/frame *6.5m pix/frame @ 30hz = 195m pixels/sec.*
@ 60hz = 390m pixels/sec.

The values specified in these areas may also imply a limit on the average size of the polygons. The depth complexity or number of pixel touches required will also depend on the application. The number five in the above example implies a ground based application where more polygons will be in line with each other so that their pixels overlap the same areas of the screen. All of these factors must be considered together to determine if the specified system capacity and/or system resolution are appropriate values for the intended application.

One of the early approaches to this scan conversion was for the DP to compute the contents of each raster line in synchronism with the display of the raster line. Each completed raster line was sent immediately to the display to be drawn. Each subsequent line had to be completed when the display was ready to draw it. The advantages to this approach were that relatively little hardware was required and there was very little time added to the system transport delay. The disadvantage was that this approach forced some assumptions, and limitations, about how much detail was on any given raster line. This in turn forced some very difficult capacity constraints on the database.

The modeler had to attempt to predict and manage display space issues in model space. This meant that a potential overload would depend on the altitude of the eyepoint as well as the position since a low altitude will tend to compress more polygons into the raster lines near the horizon. Some systems adopted a vertical raster structure in an attempt to alleviate this problem. Also, increasing the resolution of this type of system was difficult due to the serial nature of the process. Faster processing was the only way to add pixels to the relatively fixed line time.

The more common approach to display processing involves the incorporation of a dual pixel or display buffer. Each half of this buffer contains enough memory capacity for the contents of all of the pixels in the image. During one iteration of the system, the entire image is assembled into one half of this buffer. During the next iteration the contents of this half of the buffer are sent to the display while the other half is being filled with the next image. Database capacity limitations are much easier to manage because the total polygons on screen is the issue rather than where they are on the screen. The disadvantages to this approach are that it requires additional hardware (the dual buffer) and an additional field is added to the transport delay. These penalties are accepted in the interest of a more stable and predicable system.

Other important aspects of the system (fig. 2-3) which may be manifested in the DP include visibility effects, anti-aliasing, texture, color, shading and priority (occulting). Potential implementations of some of these effects will be discussed in the appropriate sections. The details of how the display processor of any given system incorporates these many functions is beyond the scope of these notes and may, in fact, require a complete design disclosure and a considerable amount of study on the part of the potential user. It is hoped that the discussions in the following sections covering individual DP operations will at least make the user aware of the critical issues.

2.4 System Considerations

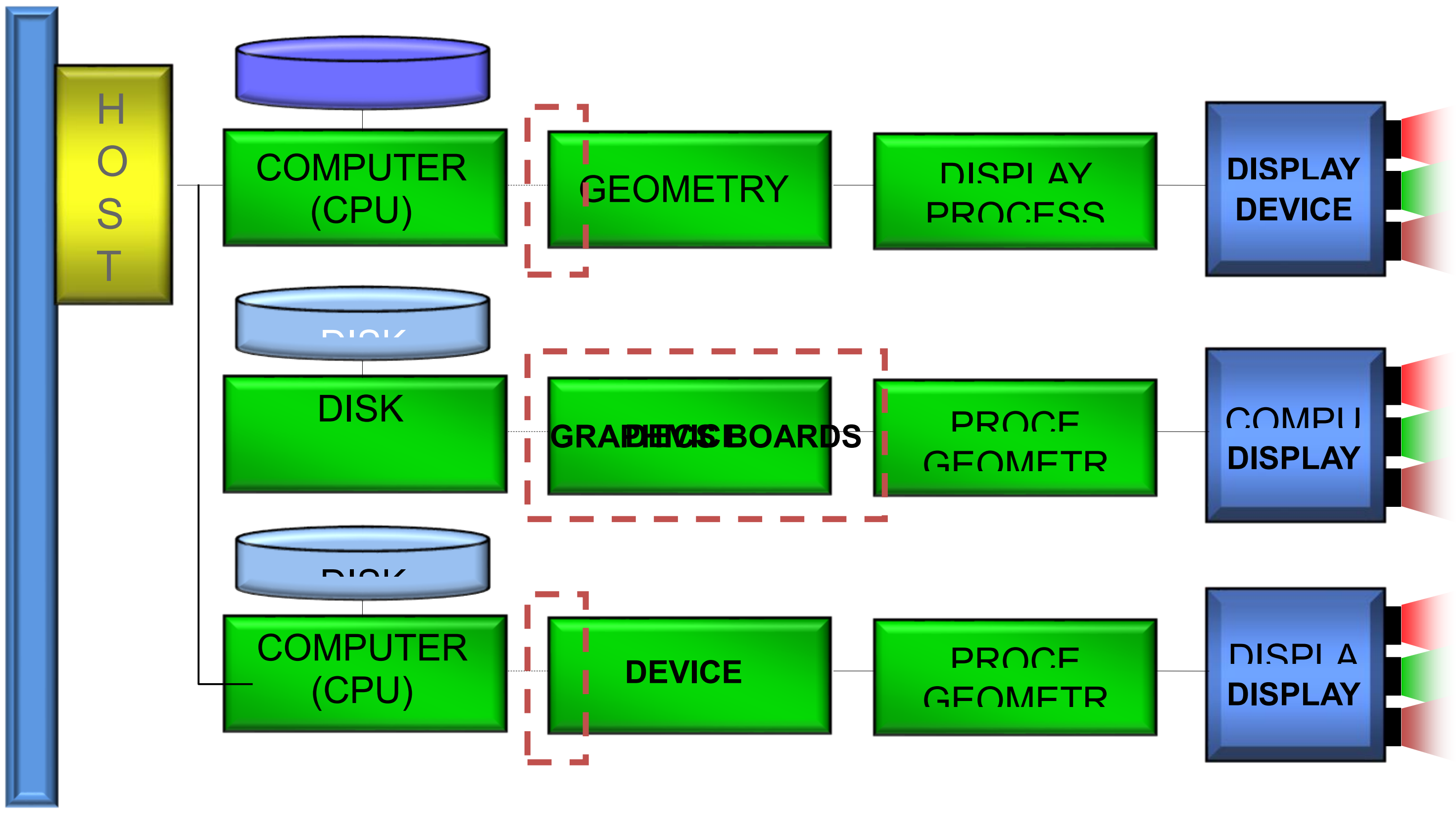
So far we have looked, briefly, at one channel of a typical system. Depending on the application, a great deal more than this may be required. The effect on the system of operating several channels or of adding additional channels is very important in large, multi window simulations. How long the image generator takes to complete the picture and get it on the display (i.e. transport delay) is also an important consideration.

2.4.1 Multi-Channel Systems

In recent years, more and more applications of CIG hardware have required more than one channel of image generator with their associated display devices. Early simulations, particularly within the civil airline community, consisted of a single channel with two monitors displaying the same image (Captain and copilot). As with the increase in system capacities brought on by market pressures, a larger overall field of view was also seen to be a requirement. Three or five channel systems are now the norm on airline systems.

Military systems have always tended to be large, but recent trends have increased them even further. This is aggravated by the addition of sensor and guided weapons channels to the already large visual fields of view. Systems with eight channels or more are often more and more for the WST or WTT simulators as the cost of IG channels goes down. This trend will probably continue until the advent of head and/or eye tracked area of interest display devices.

The way in which the system is affected by additional channels may have a severe impact on the various capacities and capabilities of the image generator. To some degree these effects, and the system's ability to work within them, will depend on where the division into channels takes place. In earlier image generators it was possible to share the GP and/or the DP tasks between channels. These compromises were done primarily to reduce the overall cost of the system hardware at the expense (or lack of) of performance in one area or another. Current PC based systems do not generally benefit from sharing capacity among



channels, either economically or technically.
The system depicted in fig. 2-4 consists of three PCs networked together to form a three channel visual system. Each PC, and their associated graphics processors, creates a single image which is displayed to the operator through some type of display device. In this system each PC has a disk which contains the system software and the database, although as mentioned above, it is generally possible to only have one disk and let the “master” PC distribute the data to the other channels. It may also be necessary to have that PC perform some or all of the system type functions that are required to keep the overall scene together so that it forms a continuous image.

fig. 2-4 – “Simple” multiple channel PC based system

One area where the commercial game boards are very weak is in providing some method of synchronizing these three channels so that the image are all drawn at the same time. There are a number of good reasons to do this, one of which is the synchronization of the scene content that was mentioned above, i.e. flashing lights, the movement of dynamic scene elements, etc. Other reasons have to do with the way that the human eye perceives images that are not drawn at the same time. One example is the appearance of the horizon line during a rapid roll. In the system above, without any form of synchronization, it is likely that the horizon will appear to break at the channel boundaries. This is because the images are not being drawn together and the raster lines that make up the area around the horizon are being drawn at different times. The more rabid the roll rate, the more apparent the discontinuity will appear.

A more subtle problem with images displayed at slightly different times is the physiological response of the observer. Even though the artifacts are virtually invisible on the screen, the observer my experience eye strain, headaches, and even simulator sickness after viewing the disjoint images for an extended period. The time required and the degree of detrimental response by the user, if any, varies considerable from person to person, making the effect very difficult to quantify.

The commercially available game boards do not typically include a synchronization mechanism because a great majority of their market is a single game player using a single PC. There are a number of techniques, software and/or hardware, for providing some form of synchronization or "frame lock". This is one of the areas where IG manufacturers and suppliers must augment the capabilities of PCs with graphics capability in order to create a PC based image generator, or "PC-IG".

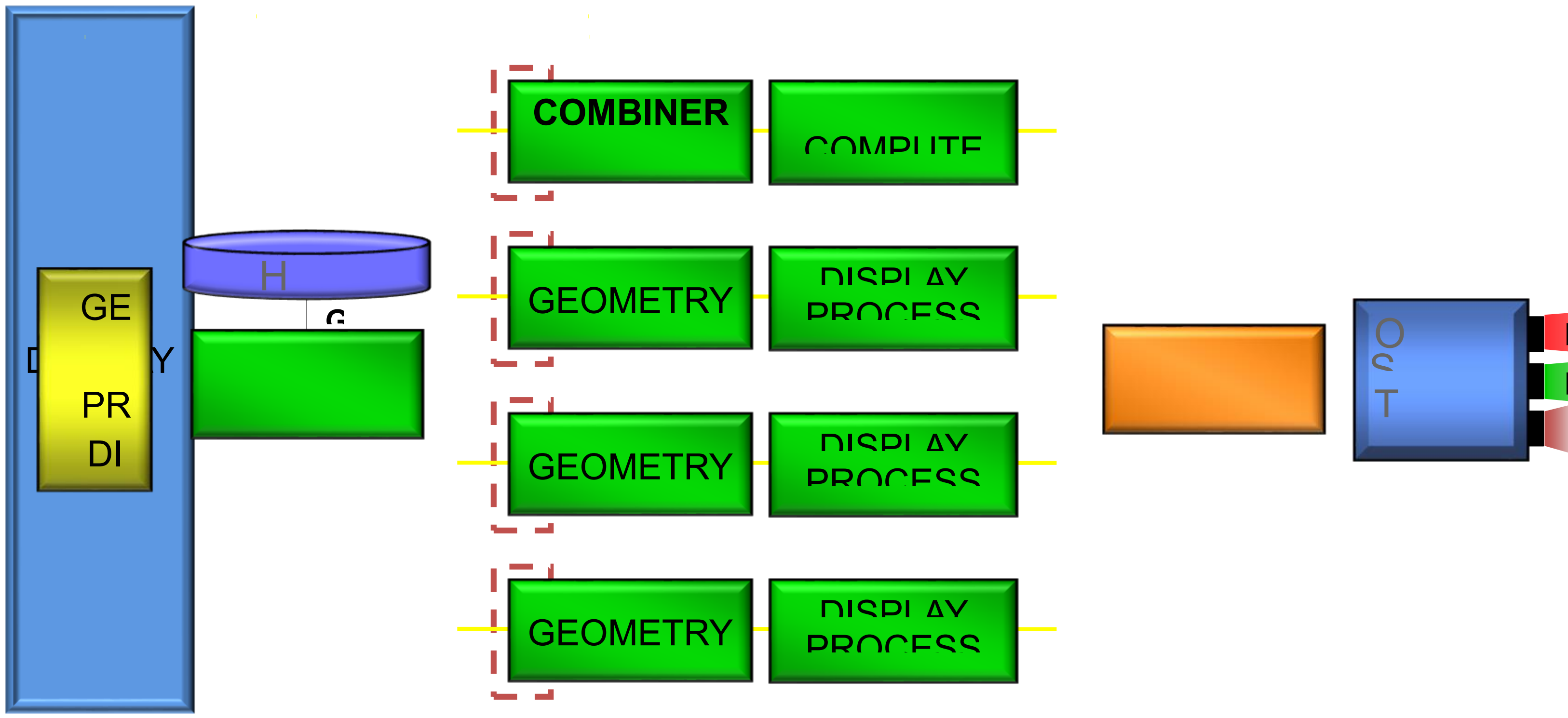


Figure 2.5 shows a different approach to increasing the quality of the image provided by a single channel of a PC-1G. In the configuration depicted the processing power of four GPUs is combined to create a single image to be displayed to the observer. This can be accomplished by configuring a system with four graphics cards, as shown above, or by actually using four PCs. In either case the images processed by the four GPUs must be combined in some manner before being sent to the display device. Obviously, the

fig. 2-5 – Multiple PC and/or GPU channel architecture

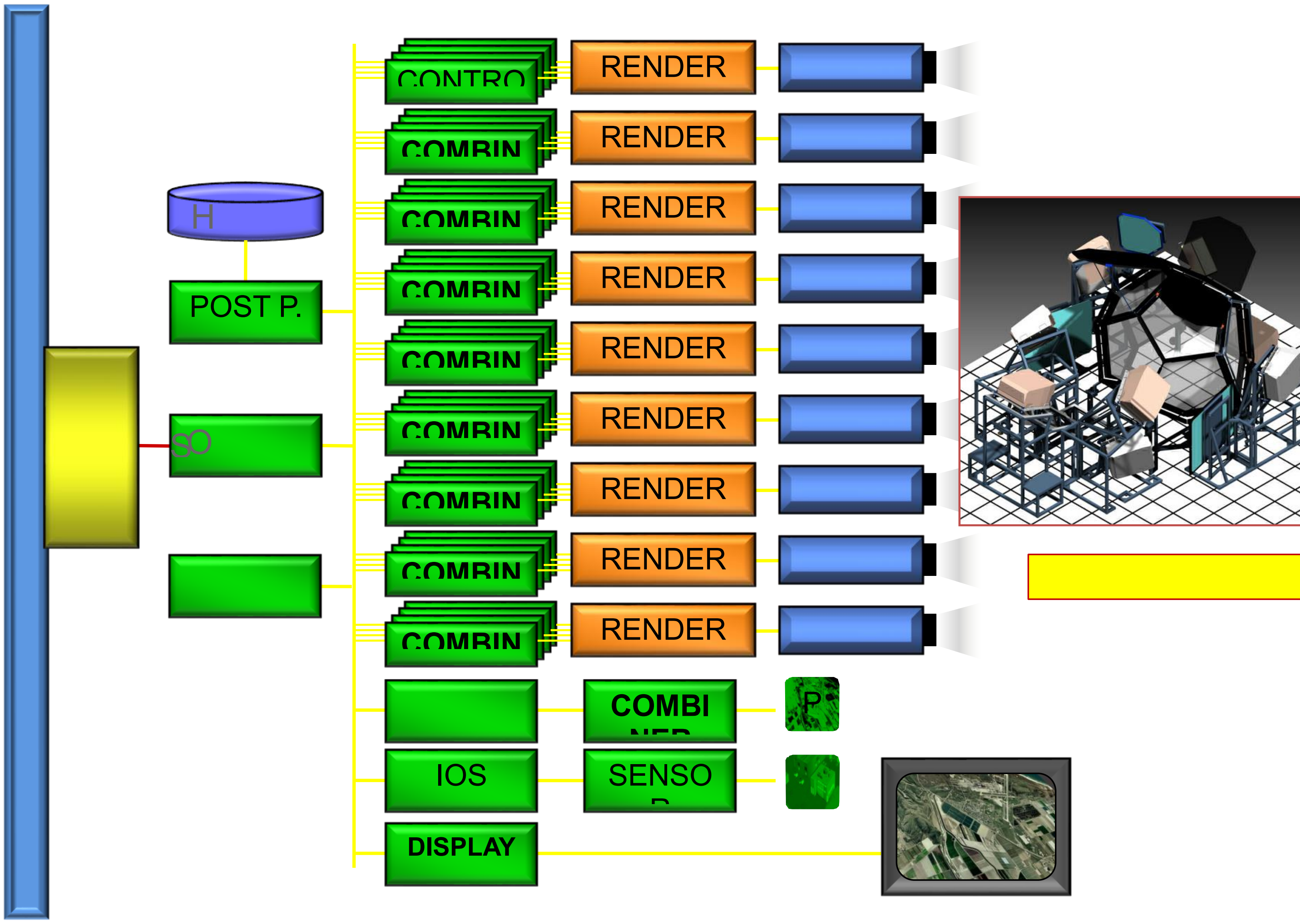
"combiner" function is not included in commercially available PCs or graphics boards (yet) so this is another area where the supplier must add to the capability of the basic system.

The reasons that systems of this type are commonly available can be traced to both early and recent history of simulation technology. As was discussed in chapter one, long ago the simulation industry developed very high standards for system stability and performance during the early years when very expensive special purpose hardware was used for the IGs. When workstation technology with much lower costs came along the industry was anxious to benefit from the economy of these devices, but were unwilling to give up many of the "standards" they were accustomed to. This was also the case some years later when PCs with GPUs became available at even lower costs.

In order to achieve the historical expectations of what an IG should provide, it was necessary for the early PCs to be grouped in the manner shown so that performance could be increased to an acceptable level. Most of the perceived shortfall in performance that led to the combining of GPU capabilities was in the equivalent of the DP portion of the technology. The new PCs could not process enough high quality pixels to provide the high resolution images that the industry required. The basic problem was a combination of not enough pixels and simplistic anti-aliasing capabilities. There were also issues with the texture capacity of the new technology.

It could now be argued that later generations of PC based graphics met or exceeded the capabilities of the old IG hardware in these problem areas. It should be possible to return to a single GPU configuration similar to fig. 2-4. It is, however, the nature of the industry to apply the additional capability available with each new technology leap to new and/or more scene content. There is also a non-recurring cost factor related to changing the architecture of a system once it is established.

fig. 2-6 – Large multi-channel PC-IG



The system depicted in fig. 2-6 is the result of a full up visual system with a channel configuration similar to fig. 2-5. As mentioned above, one of the benefits of much lower hardware cost associated with commercial (commodity) PC hardware with advanced graphics cards is that many more top end systems have very large fields of view, requiring many more channels than have historically been used. This system has nine high resolution channels to provide imagery for a 360° display system. It also includes two sensor channels that may provide infrared imagery. These channels include post processors to alter the image from the PCs to include the features and artifacts associated with each sensor. There is also a PC channel that provides imagery to the instructor's station. The ISECT computer will handle mission function such as height above terrain or collision detection. There is also a computer dedicated to assessing which portions of the database are required and distributing it to the rest of the system (note single disk unit).

While a visual system of this type with 50+ PCs may seem overly complex to those unfamiliar with the technology, it is actually a very manageable configuration. The PCs used in a system of this type are not the desk side units most people think of, but are much smaller rack mounted configuration. Despite the imposing number of PCs in a system of this type, the actual footprint of the total system is smaller than a similarly capable system based on earlier technology.

2.4.2 Transport Delay

Transport delay generally refers to one of two different but related parameters in the simulation system. In the theoretical sense it refers to the time it takes an image generator to complete the full calculation and display of an image. This value tends to be an analytical measure due to the variables involved. The effect of the host operating asynchronous with the visual means that the time from the data input until the first use of that data is different each field. Also, many current systems will overlap the various processes in the system. As soon as the GP completes its first calculations, for example, it will send the results on to the DP so that it can start. The effect of this overlap on the system performance may depend on such variables as the system load or the database architecture.

It must also be considered that the field times for CIG systems can vary considerably. A dusk/night system which runs at 30 hz., and therefore has a nominal field time of 33 msec., cannot be expected to complete the visual process as quickly as a day system which runs at 60 hz. and has a field time of 16.6 msec. The total number of fields will depend on a large number of factors in the architecture of the system. The worst case (when not in overload) will be one field for each of the processes as shown in the upper part of fig 2-7. This has generally come to be unacceptable, however, and most current systems will include design optimizations to improve on this. The effect of overlapping the processes is shown in the lower part of fig 2-7.

The second consideration of visual system transport delay is its contribution to the entire simulator's delay. This is the time period measured from a control input by the user until the visual scene moves in response

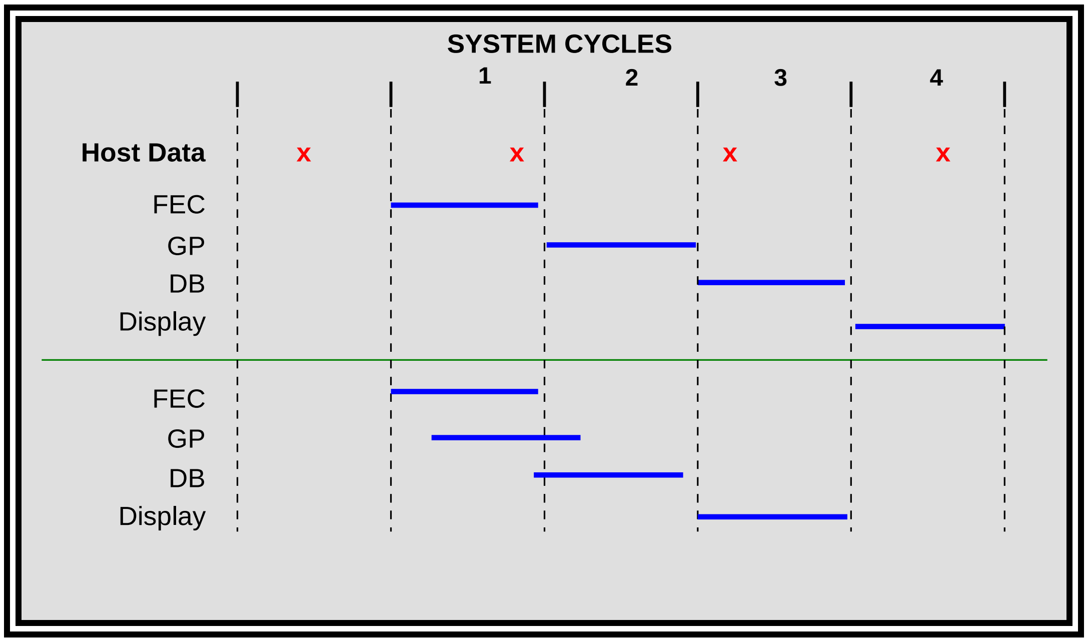


fig. 2-7 – Transport Delay in terms of system cycles

to that input. This number is normally measured, with some difficulty, as part of the acceptance test for the simulator. In order to be meaningful, the transport delay of the system must be specified and measured relative to the delay of the actual aircraft or vehicle. One of the difficulties with these tests is determining when an event starts in the real aircraft so that its duration can be measured. In practice, some agreements are reached and the same conditions applied, as closely as possible, to both the aircraft data and the test results from the simulator. This area is particularly critical in high performance aircraft where high rates may combine with digital control inputs to create flight conditions which are difficult for the visual extrapolation routines to cope with. This situation is aggravated by the fact that most of these aircraft also require very complex imagery to support high speed, low altitude flight maneuvers.

One method often suggested to reduce the visual system's contribution to the overall transport delay is to run the visual synchronously with the host, which would remove the "dead time" between the host input and the visual system's use of the data. Although the resultant position errors are "corrected" by the extrapolation routines discussed above, there is still a variable time delay of up to one field.

Even if the extrapolation process were eliminated the resultant image is still two to four fields out of date relative to the simulator. In order to limit the effects of this delay some systems include prediction routines in their visual drive programs. These routines make use of second and third order acceleration terms to approximate where the eyepoint will actually be when the visual system creates the image. The degree of accuracy associated with these routines will greatly depend on the nature and magnitude of the control inputs from the user.

2.4.3 System Upgrades and Support

A concern often expressed by CIG users during the procurement processes is the possibility of future upgrades to the visual system. This refers to adding more capabilities to the existing technology, more channels for example, and to the possibilities for improving or increasing the technology at some later date. Manufacturers of CIG systems have attempted to keep as many "hooks" as possible in their designs to allow for future expansion in these areas. This sometimes has a negative effect in that those hooks add to the cost of the basic system.

Another aspect is system modularity. This relates to the ease with which system features and options may be added or removed. While the addition of capabilities at some later date is a consideration, the real value of this type of flexibility is in the fine tuning of an "off the shelf" system to a specific application (as in the system shown in fig 2-6).

The inclusion of commercial PC hardware in current systems has brought its own issues with upgrade and support. The PC components, the GPU in particular, are very short lived when compared to the typical simulator life cycle. The possibility of replacing a GPU two or three year after it was installed is very slim unless a “life time” supply of the original processors has been purchased and stored somewhere. It will probably not be possible to purchase a new GPU of the same model and configuration. Unfortunately, the technology does not allow for simple upgrades either. A newer GPU, two or three generations beyond the original, may require a significantly different configuration of both software and hardware. There will also be a number of capabilities available in the newer hardware that the original software and/or database will not take advantage of. A better approach might be to plan for regular updates to the entire system. The hardware for such an upgrade would be relatively inexpensive (compared to historical systems) and the benefits of adding features and content would increase the training opportunities.

3. SYSTEM CHARACTERISTICS

The following is a collection of system characteristics which will help to define the nature of available systems. In many respects these will be the broad measures by which one system may be compared to another. The finer measures will be the system features and options discussed in the following sections.

Where it is applicable, the many varied tradeoffs between system performance and/or system requirements will be discussed. It will become apparent that this is a very complex factor in determining the right system for a given application.

3.1 Performance Categories

In earlier systems that incorporated special purpose hardware it was common to find significant differences the each manufactures approach to image generation. The different types of priority mechanism, for example, each had their pros and cons that had to be understood and evaluated prior to making a decision on which type of system to acquire. Also, in that era the “numbers game” associated with how many polygons a system could produce was one of the major factors in the system specification.

Today’s PC based systems do not exhibit the dramatic differences of the older technology. This is partly because the basic hardware which makes up the graphics boards is, if not the identical GPU, very similar in capabilities. The performance differences are much more subtle now and are related to how each manufacturer makes use of the features and performance inherent in the commercial GPU. As mentioned above, some differences may also be found in the techniques used to adapt the gaming hardware to a simulation application.

3.2 System Capacity

Capacity is a system characteristic which is used by both the manufacturer and the customer to describe a CIG device. Over the years this has come to be known as "the numbers game". This is not to imply that the total system capacity is not an important consideration when determining a system for a given application. It is not, however, the only consideration. One of the main reasons for the high interest in capacity is that a count of polygons and lights is one of the few objective measures in what tends to be a very subjective field. As we will see, even these "hard" numbers are not easy to compare.

Some systems (i.e. manufacturers) define polygons or surfaces differently than others. In some cases edge counts are more significant than polygons. The most important thing to understand when comparing capacity is what the numbers really mean for each system and how they relate to your specific application. The following are some of the typical categories of capacity numbers and some questions which should be considered.

> Polygons in view - Are back faced polygons (those facing the other way) included in the specified capacity? How many sides can a polygon have? Is the size of the polygons on screen a factor in the total number (see 2.4)? Are there other limitations, such as edge crossings per raster line, which may override the polygon limit? Are <u>all</u> system features, such as texture and shading, applicable to <u>all</u> of the polygons simultaneously?

.

Polygons per channel - How does this relate to system capacity in a multiple channel configuration? Is the application such that a large number of polygons will often be grouped into a single channel?

Polygons per system - How does this relate to channel capacity? Will the system support a high content, homogeneous database? What will adding a channel
at a later date do to the system and the database?

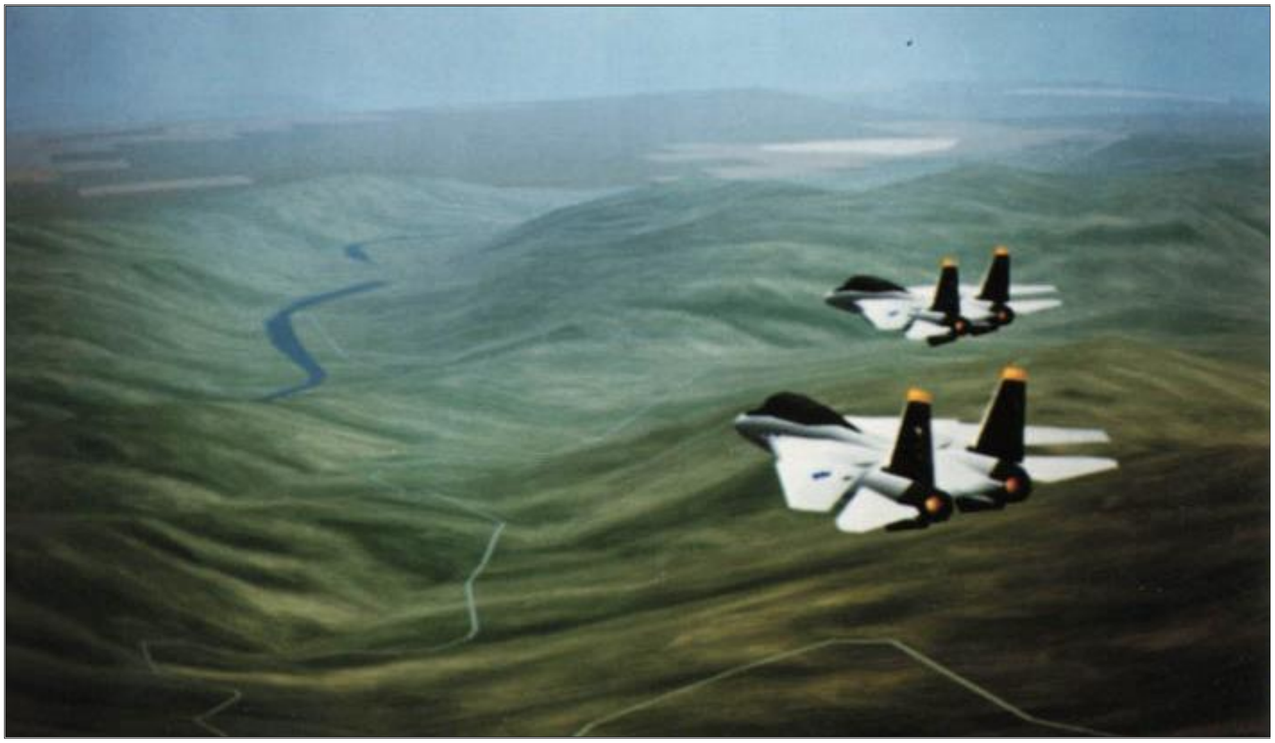


fig. 3-1 - High end system with detailed models and high density terrain

Polygons and lights - Is this a simultaneous capacity or is it polygons <u>or</u> lights? What is the tradeoff of lights per polygon?

Three system characteristics which are closely related to capacity are iteration rate, overload performance and database management capabilities. These will be discussed below but, in general, the more time the system has, the more features it can calculate and display. Also, if the system fails catastrophically when the specified capacity is exceeded, then the database must be designed so that it <u>never</u> reaches that number. The usable system capacity is then somewhat lower than the stated figure. If the system has no way of managing in new polygons and lights as the eyepoint moves through the gaming area, the applications of the system may be severely limited.

3.2.1 Iteration Rate

The update and/or refresh rate of a CIG system is a very critical characteristic. The longer the system can spend calculating polygons, lights and other features the more complex the image can be. On the other hand, if it takes too long, flicker and/or stepping will occur. We have already seen that one solution to part of this problem is to add more hardware to the system and parallel some of the calculations. This solution also adds cost and complexity to the system.

Update rate is a measure of how often the image generator calculates a new picture. Refresh rate is determined by how often the display device refreshes (or redraws) the image. It is possible for the display to draw the same image twice (or more), thus giving the IG twice as long to calculate the new scene. Many systems operate this way, either as their normal mode or as a response to overload. This is sometimes called running at “frame rate” as opposed to “field rate” (terms left over from display interlace techniques). There are two difficulties with this approach. If the update rate (at least half the display refresh rate) is too low, stepping will occur, particularly when the eyepoint is moving or rotating rapidly. Also, double (or more) imaging of rapidly moving objects in the scene will occur. This is a phenomenon similar to field tracking (see 4.1.2) which has to do with how the image is received on the retina of the eye and interpreted by the brain. These limitations should be weighed against the intended application of the CIG to evaluate their impact, if any.

Some current systems are including features to overcome or limit the double imaging caused by multiple copies of the same image. This is particularly true for systems designed to simulate ground warfare applications. Due to the relatively slow speed of armored vehicles and foot soldiers it is acceptable for these systems to update at 15 hz. Since the display devices refresh at 60 hz it is therefore necessary to draw each image four times. One of the places where this economy creates a problem is when the view from the turret of the armored vehicle rotates in azimuth (yaw) and the eye perceives four copies of everything in the scene.

One method for overcoming this anomaly is to calculate a slightly larger image than is being displayed. When the eyepoint rotates it is then possible to slide the image in the same direction by a proportional amount so that the next copy includes "new" pixels that were not in the original displayed image. This can be repeated for each copy of the image. At the next update of the scene the display window will return to the center of the calculated image and the process is repeated for the next three refreshes. There is a slight reduction in effective image resolution because some portion of the calculated image is not displayed. The size of this buffer area around the edges of the displayed image will depend on the maximum expected rotation rates of the eyepoint and the number of repeats expected.

This method will work for changes in eyepoint pitch as well as yaw. It will not work for rapid translation of the eyepoint, however, because new image geometry is required as the eyepoint's position relative to features in the database changes.

Another consideration when evaluating the iteration rates of various systems is the associated display device. In some sensor simulations the actual display device must be used. Most of these devices, particularly those intended for aircraft use, do not have flexible frame rates and the CIG system will have to be matched to the display. Also, some video projectors, such as light valves, may offer very little variation in cycle time.

3.2.2 Database Management

Database management (DBM), or paging, is the process of bringing new scene elements (polygons and lights, maybe texture) from the system disk and placing them in the image generator where they may be processed for display at the appropriate time. In other words, the polygons and lights are moved from the available database (the total available on the disk) to the active database (those active in the system memory) for eventual inclusion in the display database (those actually being processed). DBM normally implies an automatic process, based on the position of the eyepoint in the gaming area, to move pieces of the database as opposed to some action by the instructor or operator. Model selection from a control panel, for example, is not DBM.

In some of the low cost systems and in some of the early versions of the more complex systems, there is little or no DBM. The total database is loaded into the system at the beginning of an exercise and only those scene elements are displayed, regardless of where the eyepoint moves. This means that the system must include a very large memory for database storage (environment memory) and that it must evaluate every scene element in the database to determine which ones must be displayed. For most applications of CIG this imposes an unnecessarily heavy task on the front end of the system unless there is a sophisticated

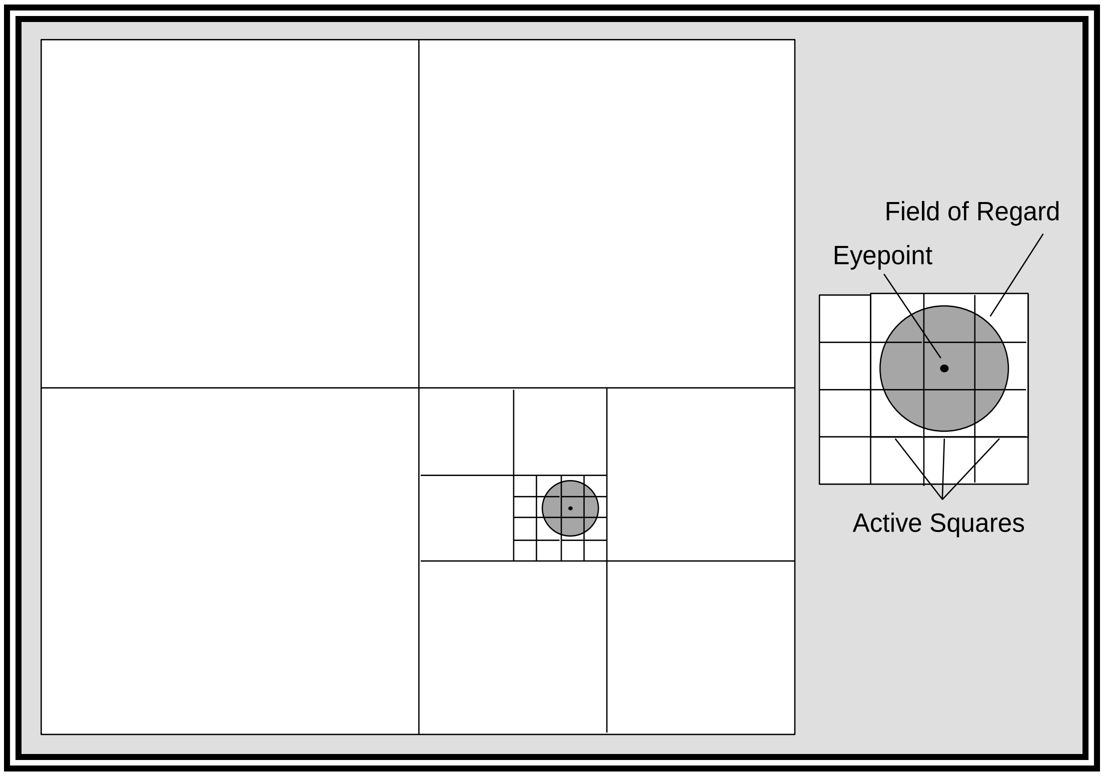


fig. 3-9 - Hierarchical Quad-tree Structure

database culling mechanism in place to efficiently limit the polygons to be processed. The many polygons which are invisible because they are too small and/or too far away must be ignored. Also, forcing the entire database into the image generator may eventually restrict the size and/or density of the database. While this may not be apparent immediately, one of the basic axioms of CIG technology is that the database will outgrow the system. Without database management this can happen very quickly.

Most database management implementations utilize some form of hierarchical database architecture. This means that small portions of the database are grouped into larger portions which are in turn grouped into yet larger portions. The top of the resultant "tree" structure will contain relatively few references to pieces of the database but those pieces will contain references to more pieces and eventually the entire database. Each piece will include geographic information which is evaluated relative to the eyepoint location. If the eyepoint is not in (or near) a piece of the database then that piece and all of the smaller pieces to which it refers may be disregarded for that particular computation frame. If the piece is included then the process moves down that "branch" of the tree and begins evaluating smaller pieces and omitting them or, if they are determined to be candidates for display, retrieving their data from the system disk (normally via the front end processor). This technique works particularly well for large, homogeneous databases where the eyepoint will travel large distances at high speed. The details of how wide and how deep the tree should be for optimum system performance will depend on the individual image generator and the database being accessed. Fig. 3-9 shows a simplistic version of a quad-tree database where each square area is divided into four smaller areas and so on. In the example the eyepoint is in the lower right quadrant. A quick comparison of the eyepoint position to the other quadrants allows 75% of the gaming area to be ignored.

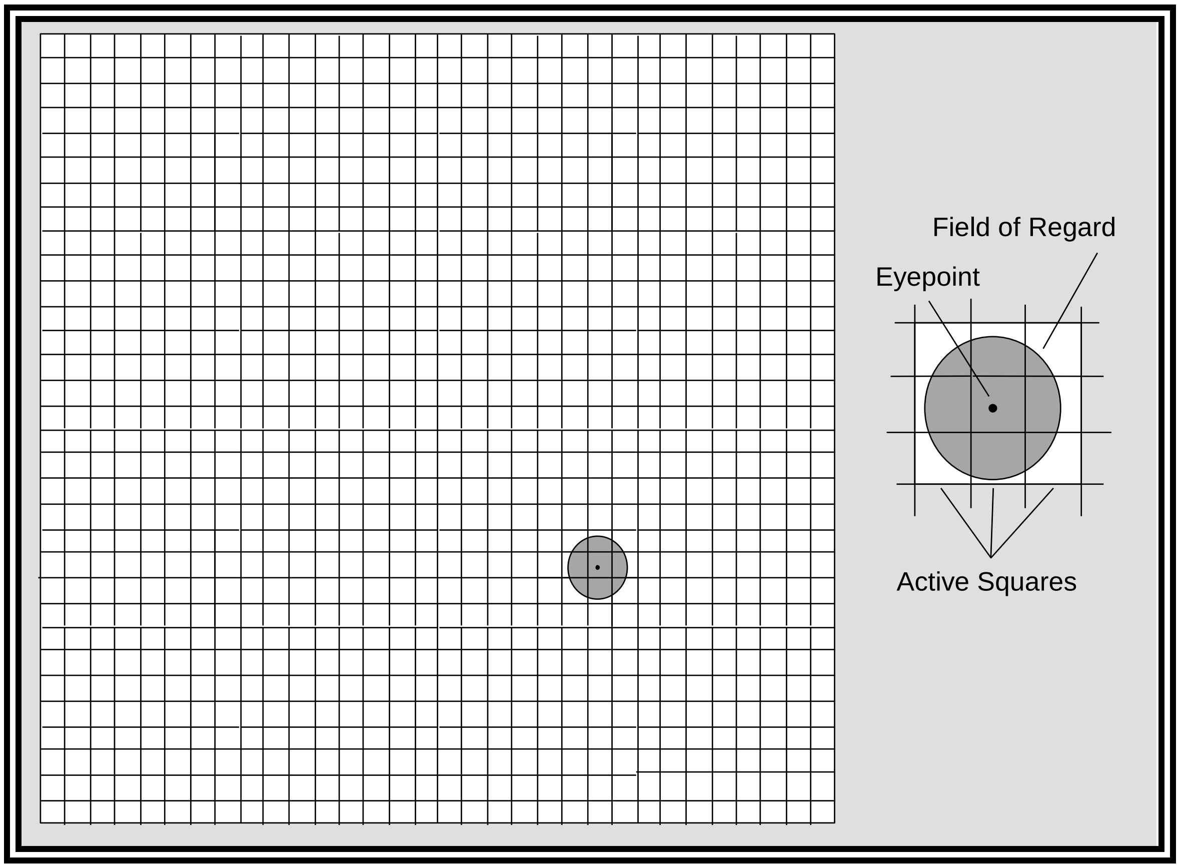


fig. 3-10 - Management Grid Structure

Fig 3-10 shows a grid structure which is sometimes used. In this implementation the front end computer compares the eyepoint position to the rows and columns of a known grid and brings in the nine squares (in this example) which are within the field of regard. This process requires that the front end computer have some knows the extent and layout of the gaming area.

Another aspect of database management is its effect on attempts to quantify the database for specification and costing purposes. In the early days when a database was only as large as the system's memory could hold, there was a finite number of polygons and lights which could be included. With efficient database management techniques the size of the database is limited more by the application, the modeler's ability to squeeze performance out of the image generator and, of course, the database cost.

3.2.2.1 Level of Detail

An extension of the database management process may also be used to change the number of polygons used to depict areas or objects. When an object is far away and just entering the scene, it may be represented by a few polygons. As the eyepoint approaches the object more polygons are added, or substituted, so that adequate detail is provided. This capability, called level of detail switching, may be repeated several times so that a complex object will gracefully increase in detail as it is approached. This insures that the majority of the polygons being displayed are concentrated close to the eyepoint where they are most effective.

Systems which include a level of detail capability differ considerably in implementation. Some will allow each object to include a transition range which will determine the distance from the eyepoint at which the level of detail change will occur. Others will have a fixed number of levels available and a fixed range for each level. Still others will calculate the size of an object on the screen (i.e. the number of pixels it subtends) and adjust the level of detail in accordance with pre-determined sizes. This approach works well with sensor channels where more than one magnification may be required.

In either case, the modeler must design the polygons and lights for each level in such a way that the change is not distracting to the user. The trade off here is in database density. Consider a homogeneous database made up of objects which must be represented in as much detail as possible and are limited to two levels of detail. Each object may consist of one or two polygons when it first enters the scene. The range at which each object must be changed to several polygons becomes the radius of an arc which will contain the balance of the polygons that the system can provide. The shorter that range can be made, the higher the density (and therefore the detail) within the resultant arc.

If the range is made too short the user will see the increase in detail as a distracting "pop" in the scene. If the range is too large the polygons will be spread too thinly and not enough detail will be available in the objects nearby. If the system includes a transparency capability it can be utilized to fade the new version of the object into the scene thereby reducing the distraction of the viewer (see 4.3.1).

The modeler may also reduce the visual impact of the level of detail change by careful selection of the color and geometry of each version of the object. The color of the low detail object, for example, should be an average of the colors used in the high detail. Similarly, the low detail polygons should be shaped so that the addition or substitution of the high detail polygons will have the minimum effect on the apparent size and/or shape of the object in the scene.

On the other hand, the modeler may wish to make the low detail versions of some objects considerably different from the higher or full detail version. This is in order to make up for basic limitations in all visual systems due to the limited (compared to the real world) resolution of the displays and/or raster structure. Distant objects like aircraft or ships (i.e. targets) will not be visible at the correct distance because they have been blurred by the anti-aliasing process (see 4.1) or are too small to contribute sufficiently to the pixel that they are included in. The modeler may choose to make the low detail version of such objects somewhat larger than the actual size, a lighter or darker color to increase the contrast to the background, add low intensity calligraphic light points to "outline" the object, or a combination of several similar tricks. This works best in systems which use transparency to bring higher levels of detail into the scene.

While the techniques discussed above work well for objects such as aircraft models, buildings or trees, level of detail for terrain is more difficult. If the simulation application requires the eyepoint to be on or near the ground a detailed representation of the terrain profile is normally required. This data may be obtained from NGA DTED (see 6.4.1) or similar sources. The problem is that the number of polygons required for even a crude representation of the terrain surface can only be supported for a relatively small area around the eyepoint. Outside of that area a robust level of detail process must be at work.

One of the major problems is that some terrain features are very large and can be seen from a great distance. It may be necessary, for example, to see the shape of a large hill or mountain much further away than other flat terrain. Also, if the outline is not sufficiently represented in the lower levels of detail it will be hard to hide the changes to the higher levels as they occur.

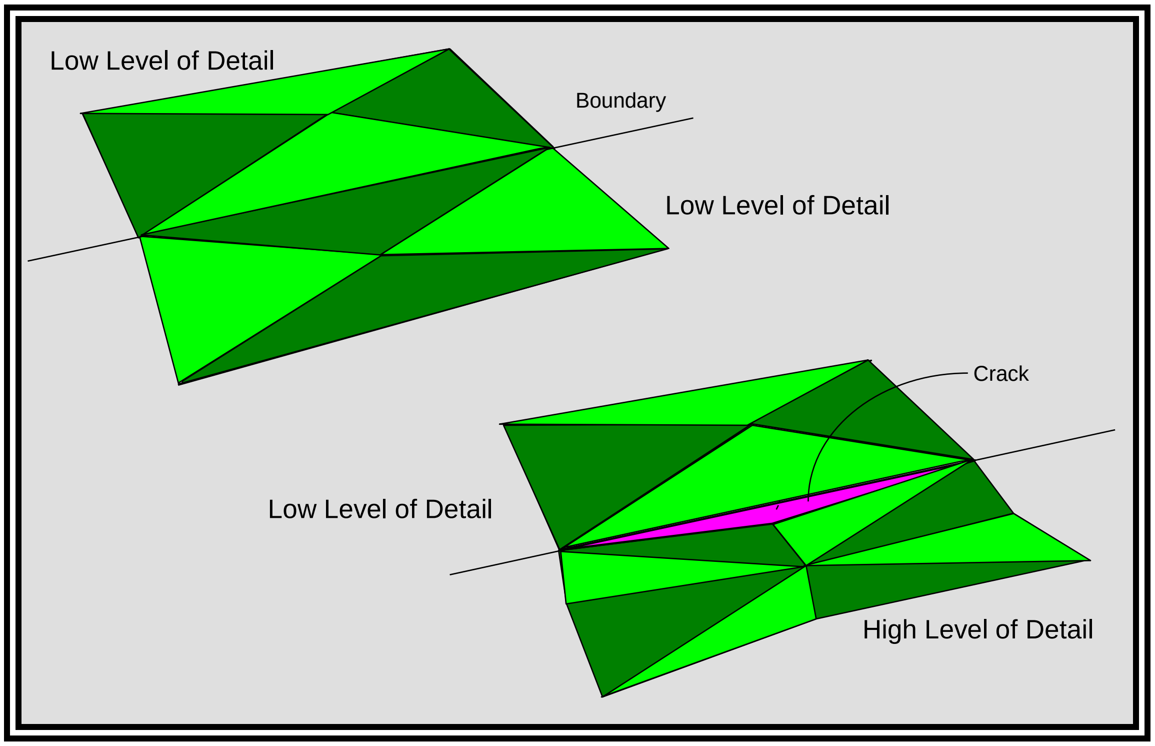


fig. 3-11 - Terrain crack between levels of detail

A more serious, or at least more common problem is cracks appearing between the low and high levels of detail. The simple case illustrated in fig. 3-11 shows how a crack is opened up when the boundary polygon of the near area is divided into two polygons to provide a more detailed terrain profile. In an actual database the low level polygon would probably be divided into more than two polygons and, of course, there would be a large number of these events occurring at a rapid rate. The typical solution is to build "skirt" polygons of the same, or similar, color as the terrain. These may be done manually or the database tools which convert the NGA data into polygons may build the skirt polygons as well. By definition these cracks occur at some distance from the eyepoint so any anomalies caused by the vertical polygons is normally not obvious to the viewer.

Some newer systems solve this problem by first dividing the low level polygon into smaller pieces which are still in the plane of the original polygon. As the eyepoint approaches the vertices of the new polygons or moved vertically, up or down, to create the required terrain profile. With this mechanism the crack is left closed until a matching polygon in the next area is available.

3.2.2.2 Sub-Model Switching

Another less structured form of DBM used in non-homogeneous databases, such as airports, is sometimes referred to as sub-model switching. In this case large pieces of the database are assigned to specific geographical locations in the gaming area. When the eyepoint enters one of these areas, the polygons and lights associated with that area are brought into the system. This technique handles areas of interest in the database. The typical example is an airport database which has several runways. Since the simulator can only approach one runway at a time, it is wasteful to spend system resources to depict the stripes, tire marks, numerals and lights for all of the runways simultaneously. Each set of markings is therefore treated as a sub-model and is only displayed when the eyepoint approaches that runway. The remaining polygons may then be applied to other important features in the airport environment.

3.2.2.3 Automatic Model Selection

Another type of DBM is automatic model selection or large area database management. This is different from those discussed above in that the latitude and longitude to which the database origin is referenced by the host computer is changed. This is necessary because the CIG database is normally built on a flat X, Y, Z grid and most simulators work in round world lat./long. coordinates, particularly for the radio aids portion of the simulation. If the training scenario calls for taking off from one airport and flying to another several hundred miles away it will be impossible for the position and heading of the second airport to be depicted correctly if it is built in the same coordinate system as the first airport. It will therefore be necessary to change the model origin at some point during the flight so that the new origin is on or near that airport.

This is normally accomplished by referencing a table contained in the database or in the host computer software which relates a series of latitude and longitudes to specific database sections. As the simulator moves around its gaming area (typically most of the world) it loads the portion of the database which is associated in the table with the lat./long. entry that it is closest to. This is a relatively simple process if it can be done above cloud or otherwise out of view of the observers. If the ground must be in view at all times, however, it becomes complicated.

In order to avoid distracting discontinuities in scene content, small portions of each database section must be displayed at the same time. This means that database sections referencing different model origins must be displayed simultaneously. It may also be necessary to include several interim database sections between the two airports so that these multi-origin areas are minimized.

3.2.3 Overload Management

Overload occurs in a CIG system in a number of places and for a number of reasons. The most common reason is that the database designer, in his zeal to include as much as possible, has exceeded the system capacity. This is aggravated by the fact that it is impossible to check all of the near infinite positions which the database may be viewed from. The only way to insure that this type of overload never occurs is to build the database well below the systems specified capacity (unheard of!). Another reason for overload may be that the system is operated outside the parameters for which the database was designed. One or more dynamic models may be moved into an area not designed for them or polygon intensive weather effects, such as cloud models, may be selected in an area not designed to support them.

An overload may occur in any of the processing centers of the image generator discussed in the previous chapter. Too many dynamic coordinate systems or other special effects may overload the front end processor. Also, the database management techniques described above may overtax the front end of the system in some configurations. Too many polygons and lights in the active database (even though the display database is within limits) may overload the geometric processor which must evaluate each potential scene element. Finally, the display processor may overload if there are too many transparent polygons or calligraphic lights in the scene or if the system architecture is sensitive to polygon size, number of over-writes or grouping on the screen (edge crossings per raster line for example).

Some systems handle overload by slowing down the update rate relative to the refresh rate. In this case, overload is only a problem if the iteration rate of the system is important to keep the eyepoint moving smoothly through the database. Calligraphic systems may reduce the update and the refresh rate slightly in order to give the IG more time to complete the image. Some of these systems may also reduce the number of raster lines being drawn to make more time for the display to draw light points.

The high end systems handle overload in various ways. Some will extend the immediate field to allow the scene to be completed and will then remove polygons from the display database selectively. This allows the update rate to return to normal after only a few fields. Once the load returns to normal the polygons are returned to the display database. The critical issue is that the polygons which are removed be the least significant (the most distant) so that the observer is not distracted by the disappearance of obvious scene elements. The normal method of accomplishing this is to adjust the ranges used in the level of detail process. A somewhat more selective approach is to classify scene objects in such a way that certain ones can be "turned off" to temporarily reduce the system load.

Overload management is one of the areas where PC based systems suffer from the design bias toward gaming. Like synchronization discussed above, overload management is apparently not a concern to game players. Consequently there is little or no capability built into the hardware to feed performance data back to the front end computer, or anywhere else. There is no way to know that the system load is about to exceed capabilities so few if any of the above methods for reducing the load before overload occurs no possible. Management must then be performed off-line in the sense that the database and application must be carefully engineered to avoid “hot spots” where the system is required to process more than it is capable of.

3.3 Resolution

The resolution of the image generator (as opposed to the resolution of the display system) generally refers to the number of pixels the system will produce. Since the content of each pixel must be computed individually within the display processor hardware (see 2.4), the total number available in the system tends to be one of the important cost/performance tradeoffs which must be evaluated.

Most applications for CIG require a standard 3 to 4 ratio display format and this is generally provided. Some systems, however, have programmable formats so that the pixels per raster line may be tuned to specific display devices and applications. Modular systems are generally available with more than one resolution configuration.

The ability of the specified number of pixels and raster lines to provide the necessary image fidelity must be evaluated against the type and degree of anti-aliasing in the system. Two types of resolution values are considered.

The objective resolution is determined mathematically by "measuring" the angle subtended by pixels in relevant areas of the field of view and averaging them. This measurement is done using a test pattern of equally spaced black and white lines which is reduced in the image until the separate lines are barely detectable. Once the measurements have been made, a Kell factor (see Glossary) is applied to approximate the actual pixel size.

The subjective resolution is measured by selecting a significant object of known size in the database (runway stripes, for example) and moving away from it until it is no longer detectable. The angle which the object subtends at that distance is then calculated. It has been agreed (by the FAA among others) that this is a useable measure of system resolution as long as the database objects being evaluated are operationally significant.

The reason that the objective and subjective resolutions may differ is that the anti-aliasing process (see 4.1) samples the database several times for each pixel. Depending on the color and geometry of the portion of the scene being considered, this is similar to adding more pixels to the system. With current anti-aliasing techniques the subjective resolution typically measures about 25% to 50% better than the objective resolution.

3.4 Priority Mechanism

In a CIG system, as in any computer graphics device, the system must have a way of determining which polygons are in front and which are being occulted (partially or totally). Two major approaches to this problem have been developed.

3.4.1 Z Buffer

The Z buffer method evaluates the distance from the viewer (Z depth) to the contents of each pixel in the scene. An additional buffer is included which stores the current Z depth for each pixel. When a polygon is processed the Z depth of each of the pixels it covers is compared with those already in this buffer. If the new value is greater than that already in the buffer they are discarded on the assumption that the old polygon should occult the new one. If the new value is less than that stored in the buffer the new Z depth replaces the old and the contents of the new pixel (i.e. color, texture, etc.) replaces the old values in the display buffer. When all of the polygons have been processed the contents of the display buffer will be in correct priority order and are sent to the display as the completed image.

Fig. 3-12 shows three polygons being drawn in the order they appear in the database. Since polygon B is in front of A, B's pixel data will replace A's data in the pixels that they share. When C is drawn it is between A and B. C's data will replace A's data, but not polygon B's.

The advantage to this approach is that little or nothing must be done in the database to implement the priority solution. The disadvantages are that it requires the additional buffer ("memory is cheap") and there are problems with anti-aliasing and transparency (see 4.3). Also, every pixel must be evaluated several times, some of them many times, to achieve the final image. The average number of times that each pixel can be processed (or "touched") is a significant system capacity issue (see 2.4).

Many early systems used the Z buffer approach but abandoned it when polygon capacities became large and anti-aliasing was required. Now, with faster electronics such as VLSI and relatively inexpensive memories available many manufacturers are returning to Z buffer technology to benefit from the efficiencies of database construction available with this approach.

It should be noted that the Z buffer approach normally requires some form of limited list priority. This is to allow co-planar polygons such as stripes on runways or windows on buildings to be processed without causing conflicts within the Z buffer.

Some recent systems use the actual range to the pixel's content rather than the Z depth, which is the Z component (in display terms) of the range. If the pixel is in the center of the channel the Z depth and the range are the same. At the edges of the field of view, particularly a large field of view, the two values will differ considerably. At fields of view approaching 180 degrees, or larger, per channel the Z depth becomes

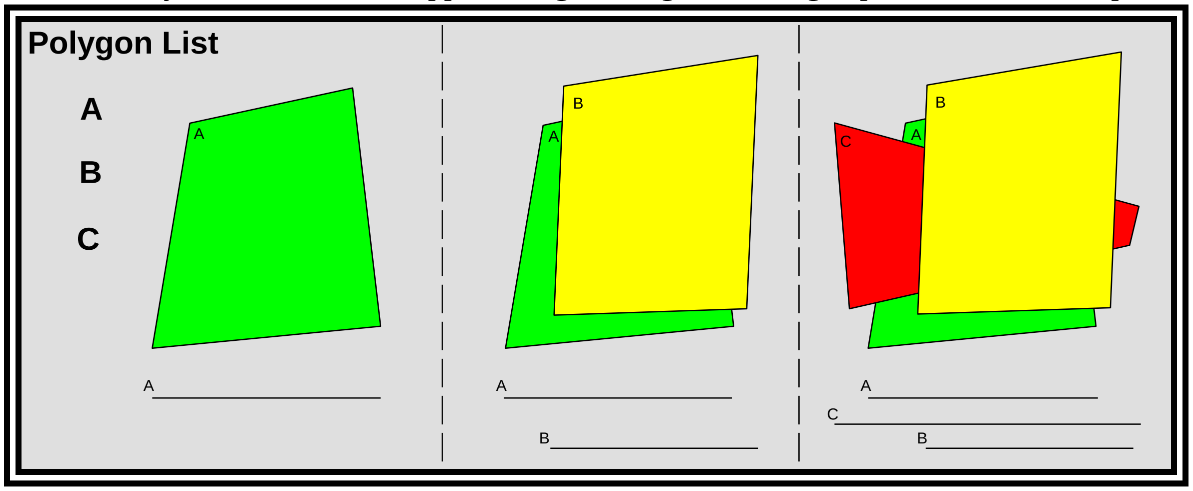


fig. 3-12 - Polygons displayed in database order.

unusable and only the true range can be used.

3.4.2 List Priority

The other approach to priority is referred to as list priority. With this approach the polygons are presented to the display processor in the order that they are to be displayed. Some systems draw the image back to front and therefore begin the list with the farthest polygons (typically the sky and ground underlayment). In this way, each polygon is drawn into the display buffer without regard to what is already there. The disadvantage to this approach is that the pixels may be filled several times before the image is complete.

Most systems construct the image from front to back by beginning with the closest polygons. The main advantage to this approach is that when a pixel has been filled it no longer needs to be considered for additional processing. The problem with either the back to front or front to back approach is that the system must first construct the list of polygons in the correct order. To aid this process the database generally contains separating planes which relate groups of polygons to other groups of polygons on the opposite side of the plane.

These planes are normally implemented with an hierarchical approach similar to the database management technique discussed above and, in some cases, makes use of the same sub-divisions of the database. During each processing field, the eyepoint's location in the database is evaluated relative to these planes and a list of polygons assembled with the closest polygons first (or last). These are then submitted to the geometric processor and the display processor in this order. The inclusion of these separating planes adds considerably to the database design effort but contributes to the overall organization of the database as well as the management. The evaluation of the eyepoint position against the potentially large number of planes is a significant computational load which may be a factor in the system's performance.

Current modeling tools make the design, construction and data entry for the separating planes somewhat straight forward. Some tools will evaluate a collection of polygons and calculate the position and relationships of most of the required planes. It is necessary for the modeler to resolve some polygon arrangements which the tools cannot sort out.

Fig. 3-13 again shows the same three polygons used in fig. 3-12. This time they have (somehow) been sorted in front to back order. When polygon C is drawn the pixels occulted by B can be ignored. Similarly, when A is drawn there are very few pixels which are not already covered by B and C. This kind of pixel fill efficiency allows list priority systems to have smaller display processors. Since the DP is typically the largest cost driver in the system this can be a major benefit. There can be problems with pixels which are touched, but not filled. This happens on the edges of polygons and with transparent polygons (see 4.3).

A limitation of list priority systems is that they do not display two or more intersecting polygons properly. The arrangement shown in fig 3-14 is a simple case where two polygons intersect. In a Z buffer system this conflict would be resolved at the pixel level with the result shown in the upper portion of fig 3-14. Note that there may be problems with this solution if the intersection is not anti-aliased. In a list priority system one of the two polygons must be listed above the other. Including a plane test will only cause the resultant image to flip flop from one solution to the other. The polygons would have to be divided so that four are required vice the two shown.

One application where intersecting polygons are useful is when planting feature models (like buildings) onto sloped terrain. The Z buffer approach would be to extend the model below the nominal lower edge, as if including the basement, and let the Z buffer "cut off" the excess. This approach should be used sparingly since it adds pixel over-writes. In the list priority system the database tools may be made to shape the bottom of each building correctly for the terrain it will be placed on.

Another limitation of list priority systems is the inability to easily determine the relative priority of dynamic models with each other and portions of the static database. Techniques have been developed for resolving

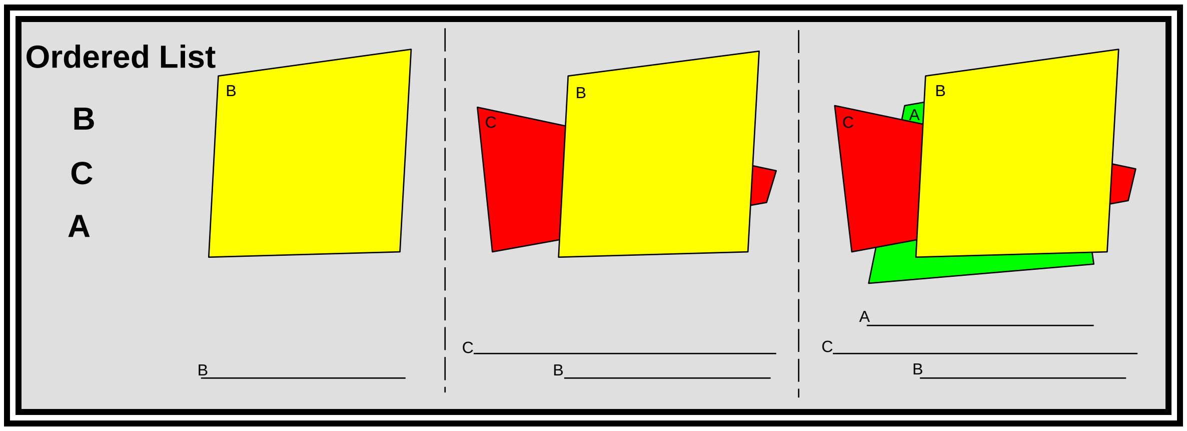


fig. 3-13 - Polygons reordered by depth.

this problem for special applications, but not in the general case.

3.4.3 Hybrid Systems

Some recent priority solutions use a hybrid approach to which attempts to include the best parts of both the list and the Z buffer techniques. One approach is to calculate the priority of the static components of the database (i.e. terrain, buildings, trees, etc.) using the conventional list techniques described above. A by-product of this process is the Z depth (or range) to each pixel in the image. These values may be used for pixel fog (see 4.2) among other things. The Z depth of the pixels occupied by the dynamic models can then be compared to those of the static scene elements using a normal Z buffer type process. This allows the "random" placement of moving models in the scene, but provides some of the pixel fill efficiencies of the list priority approach. These benefits may be limited if there are a large number of dynamic models and/or they are large on the screen (i.e. they cover a lot of pixels). In these cases the classic Z buffer over-write issues becomes the limiting factor.

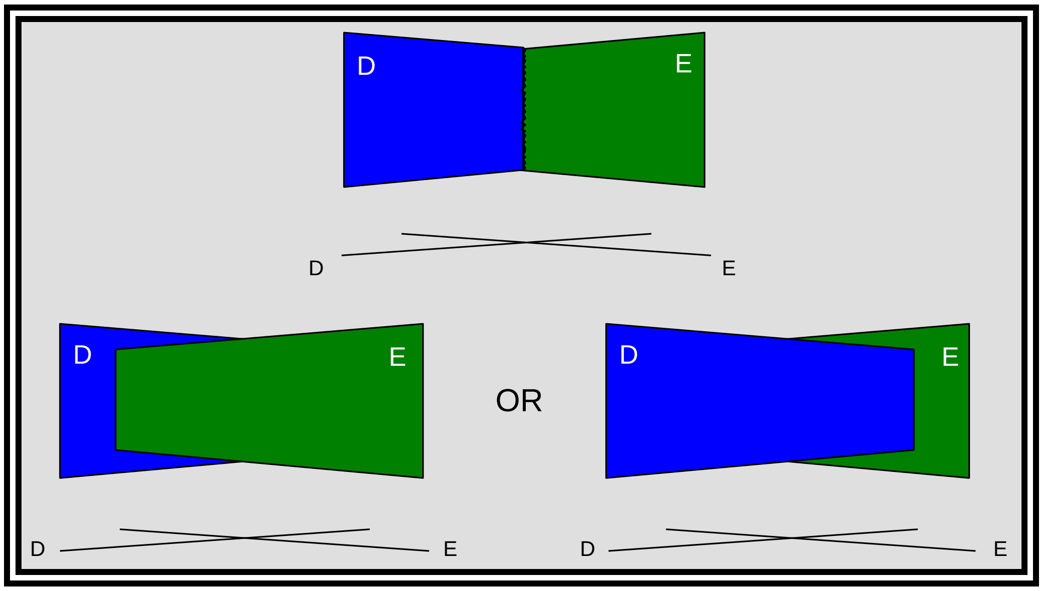


fig. 3-14 - Interpenetrating polygons, Z-buffer (top) and list priority (bottom).

Another hybrid approach is to pre-sort some or all of the polygons in a Z-buffer type implementation so that some pixels can be ignored. Several systems of this type are becoming available. The relative efficiency may depend on the application. Ground vehicles, for example, with low eyepoints tend to require more pixel fill than aviation systems. Algorithms which can sort through the many layers of polygons at ground level and somehow determine which ones can be ignored would be of benefit in reducing the amount of hardware required or freeing up the available hardware to do more useful (i.e. visible) polygons and pixels. Sorts of this type are generally low resolution and leave a number of pixel test still to be performed.

SYSTEM FEATURES

The CIG features and capabilities described below are generally available in the industry. In some systems these will be standard and in others they may be options or not available at all. Some manufacturers may refer to a capability by a different name than is used here.

The inclusion, or addition, of any of these capabilities to a system will probably imply additional cost. It may also imply a negative impact on some other portion of the system. Tradeoffs again. The inverse is also generally true. When comparing the cost of various systems, a significant difference in price will normally indicate the absence of one or more of these features, or some difference in the characteristics discussed above.

4.1 Anti-aliasing

Aliasing, scintillation, stair-stepping or quantization are some of the names applied to the lack of stability in a CIG image caused by the discrete nature of the pixel sampling. When a television camera looks at the real world each pixel is an average of the color and intensity contained in a small cone subtended by that pixel. When a basic CIG device creates an image it must construct each pixel artificially and must assign the color and intensity to each pixel based on the polygon that is present at a single sample point somewhere within each pixel (at the center in the examples). If we consider an image made up of two polygons, one black and one white, with the boundary between them being a diagonal edge through the picture we can see the problem.

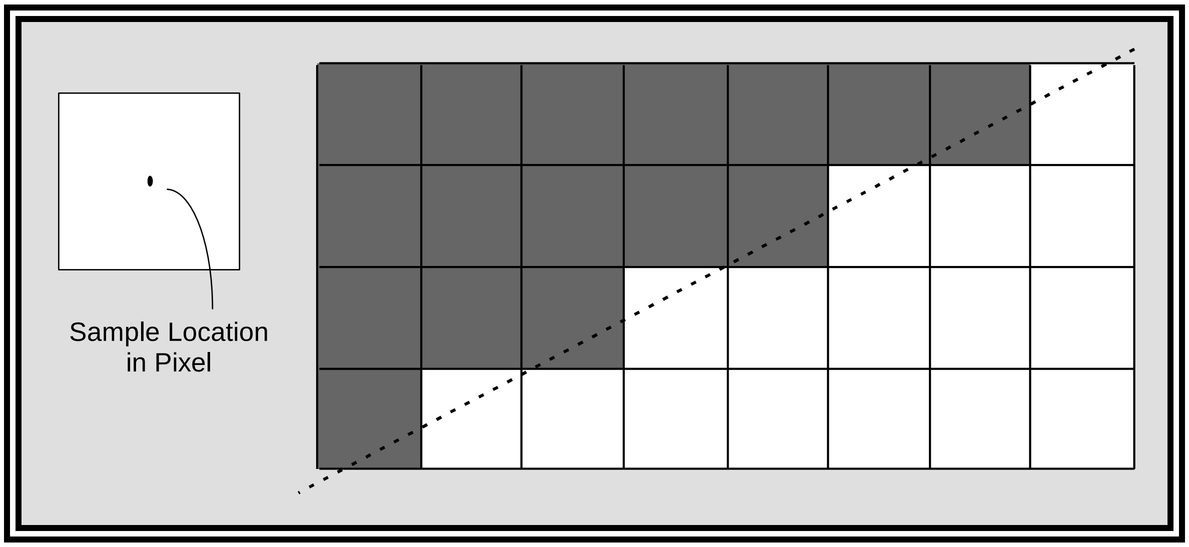


fig. 4-1 - Edge boundary with single sample pixels.

The pixels which the edge passes through must be colored either black or white depending on where the edge falls relative to the center of the pixel. The result is a jagged edge when what is desired is a smooth slope (fig. 4-1). This is bad enough in a static case but, if the edge is moving through the scene, the color assignment will change with each new field resulting in a swimming effect which runs up and down the edge. Now consider a relatively complex scene with a large number of these edges and it is easy to imagine that the resultant image would be unusable. There is a similar problem with polygons which are the size of a pixel or smaller and long thin polygons where the width may be less than a pixel (fig. 4-2). The small polygons will sometimes be missed and will flash off and on as they move through the scene. The thin polygon will appear as a dashed line with the dashes running up and down the polygon.

One approach to improving this situation is to produce more (and smaller) pixels. While this does help, it may add considerably to the hardware required (and the cost) as well as have an impact on the display devices available. To date, no real time system has been able to produce enough pixels to completely eliminate aliasing or the need for some anti-aliasing technique.

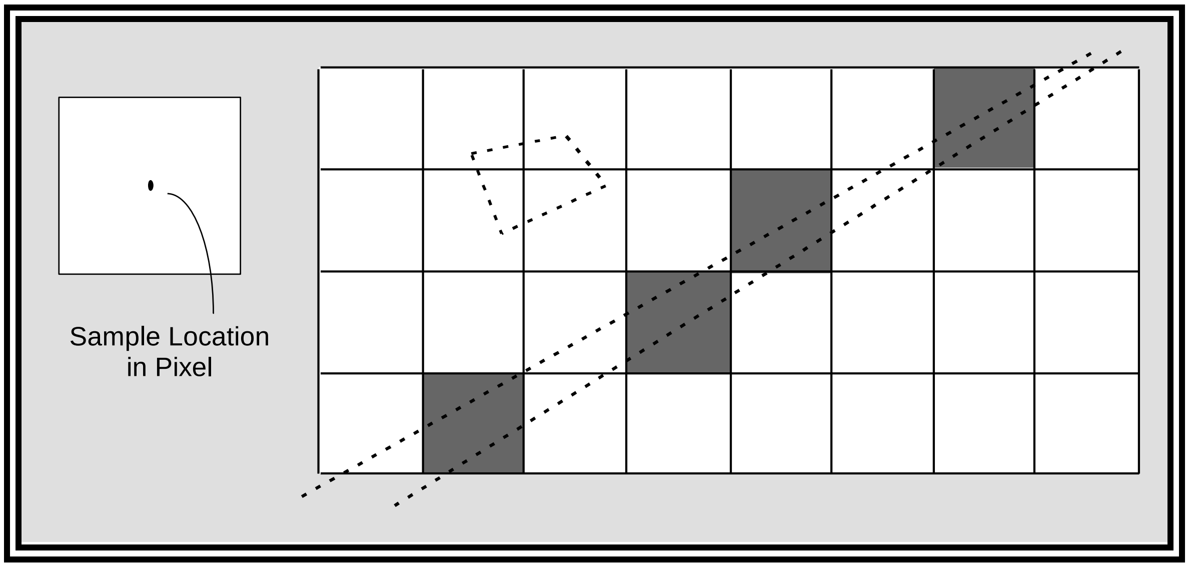


fig. 4-2 - Small and narrow polygons represented with single sample pixels.

Another approach used in some of the early mid-range systems is called sub-scanline averaging. Rather than compute only one set of raster lines, multiple sets (usually two or four) are processed and the solutions

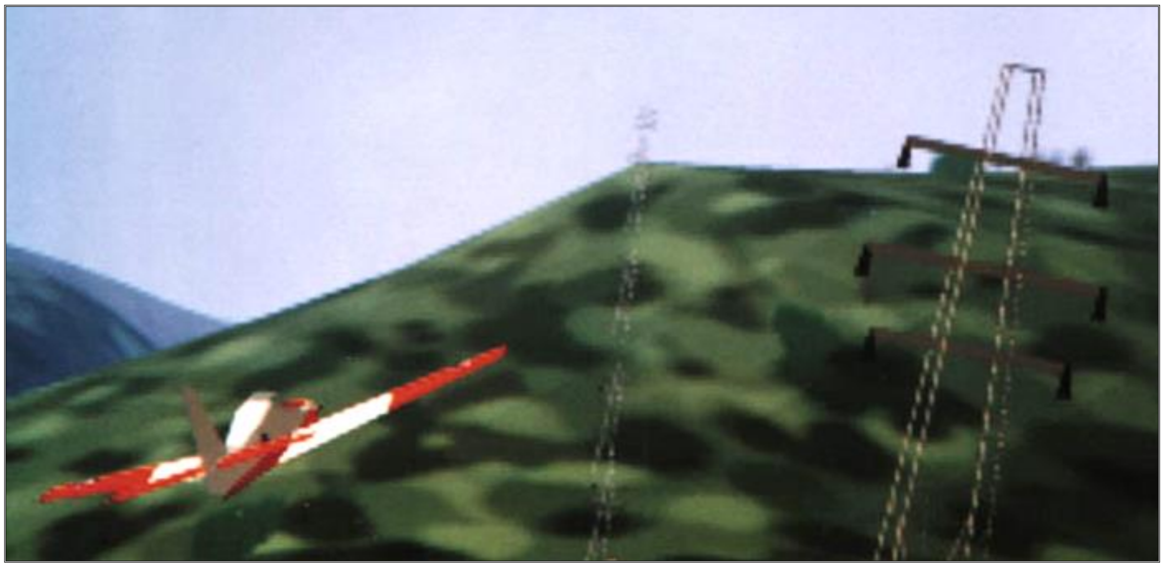


fig. 4-2a - Early CIG system without anti-aliasing.

for each sub-pixel are averaged to produce one pixel which is then displayed. If we again consider the example above with the black and white polygons some of the pixels which the edge passes through would consist of one black sample and three white ones (assuming a four sub-scanline approach) and the resultant pixel would be light gray. Other pixels along the boundary would be medium gray or dark gray. The result will be a blurred edge between the two polygons which is much more stable and less objectionable than in the previous example (fig. 4-3). When applied to an entire scene this process does tend to soften the image somewhat but is generally more acceptable than the alternative. Small polygons will make a contribution to the content of a pixel before they are large enough to fill an entire pixel. This helps these scene elements to emerge into the scene as they are approached. Similarly, a long thin polygon will appear as a slightly blurred line rather than a dashed one.

The sub-scanline approach, `while greatly reducing aliasing, still demonstrates some quantization effects. One difficulty is that the performance of this anti-aliasing may be effected by the angle of an edge relative to the raster structure. In other words, near vertical edges may appear smoother than near horizontal ones, or vice versa, depending on the implementation. Another factor which must be considered is how the image will be effected by the alternating raster lines if an interlaced display is used.

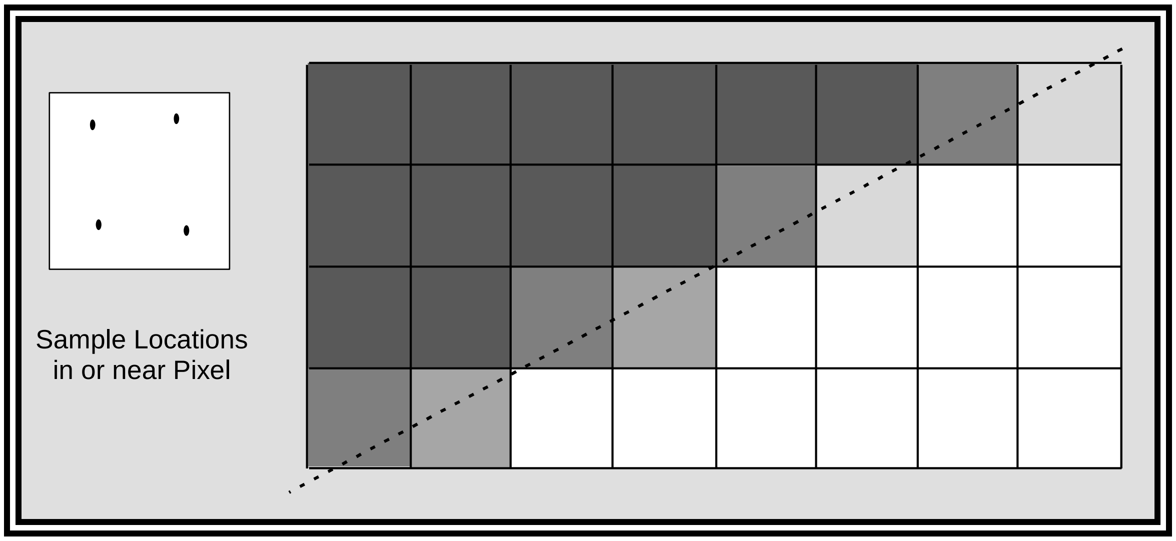


fig. 4-3 - Polygon boundary with four samples (sub-pixels) per pixel.

Many of the current CIG systems available today make use of an even more thorough approach to the problem. This involves taking a large number of samples in and around the pixel being processed. The results from these samples are weighted depending on their distance from the center of the pixel. One goal of this approach is that the total contribution to the image of a small scene element should be the same regardless of where it falls in the raster structure. Also, the positioning of the sample points around the pixel may appear somewhat random as opposed to an n x n grid. This helps to prevent large scene content changes when a vertical or horizontal edge is missed by an entire row or column of sample points. Since this technique considers the pixel as the center of an average of samples there is little or no effect from the orientation of the raster structure itself assuming the pixels are approximately square.

Many current systems allow some flexibility in the number of pixels and the number of sample points. This provides yet another tradeoff to be considered. The extremes of the choices are clearly not the way to go. A maximum number of pixels with only a single sample point each (i.e. no anti-aliasing) does not provide an acceptable dynamic image, no matter how many pixels this allows. Also, a single pixel with thousands of sample points does not make sense because they would average to a single color. The problems in making this selection is in the mid ranges. It can be demonstrated (depending on the application) that fewer pixels with more sample points will sometimes provide a more pleasing image than the same scene with more pixels (i.e. higher resolution) and fewer samples.

4.1.1 Light Point Anti-aliasing

Light points in raster systems (i.e. raster lights) are treated differently than the polygons in the scene. They are normally displayed at the same size regardless of their position with only their intensity being affected by their distance from the eyepoint. While it is important that the light points be as small as possible to represent distant light sources in the simulated environment there are other limitations which require that they must cover several pixels (see fig. 4-3a).

If we consider a light point that is only one pixel in size then a problem will occur if an interlaced display is used. During one field the light is visible, but during the alternate field it would disappear when that raster line is not displayed. The result is a light that seems to flash off and on (or flicker) at a rapid rate. Even in a non-interlaced system the light may blink as it moves from one pixel to the next.

A two pixel light point which overcomes the interlace problem will not be symmetrical so a 2 x 2 pixel light is the next choice. This however, suffers some of the same effects as the edges discussed above and will appear to alias as it moves from one pair of raster lines to the next. It will also appear as a small square rather than a round light source.

To overcome these problems light points are typically displayed as 3 x 3 pixel points but, because the outer pixels are displayed at a lower intensity than the center one, the visual effect is of a rounded, slightly blurred spot approximately 2 x 2 pixels in apparent size. This is referred to as 3 x 3 pixels, "smoothed" to 2 x 2 pixels.

This limitation on light points, combined with the limited dynamic brightness which the raster can generate in a small area of the display, are the reasons that calligraphic light points are utilized whenever high quality night scenes and/or special purpose lights are required. Calligraphic lights are not related to the raster structure in any way and, since their positioning on the screen is at a much higher resolution (the equivalent of 4,000 raster lines), they do not exhibit objectionable aliasing effects.

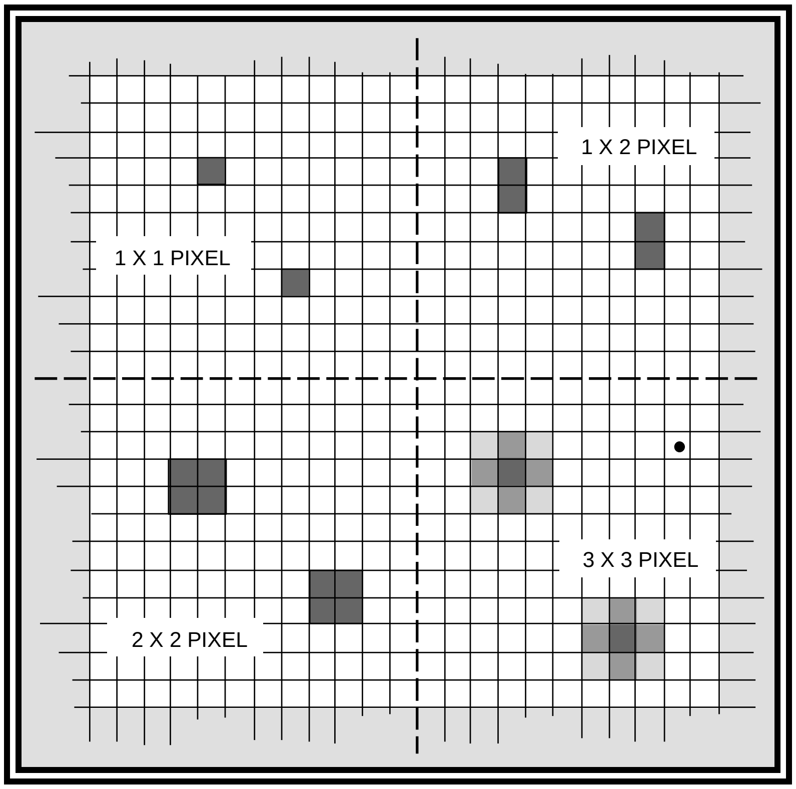


fig. 4-3a - Different raster light point approaches.

4.1.2 Field Tracking

Field tracking is a phenomenon present in all interlaced displays and often confused with aliasing because it manifests itself in very similar ways. It is, however, a physiological effect which happens in the eye of the observer. Little can be done in the image generator to overcome this effect except to use non-interlace display techniques.

Field tracking occurs when the eye locks onto an object which is moving through the scene approximately perpendicular to the raster structure. As the eye follows the object moving at a specific rate, the distance the eye moves in a single field will coincide with the distance between raster lines. The result is that the two raster fields are seen by the eye as overlaying each other. The subjective effect is that there are half the number of raster lines creating the image with all of the apparent aliasing effects one would expect in that situation.

When the eyepoint is translating across the database, field tracking may be apparent in small bands parallel with the raster structure. The location and width of the bands will depend on the speed of the translation. The degree of distraction will depend on the scene content in this area of the display. During slow, constant rate pitch maneuvers (or yaw in a vertical raster system) the entire scene may be subject to this effect. Fortunately, the constant rates which aggravate this phenomenon do not occur often during normal use of typical simulation devices.

While field tracking cannot be eliminated in the image generator the effects are somewhat minimized by positioning anti-aliasing samples in the area above and below the pixel. Since these areas are in the alternate field's raster lines they tend to provide some continuity to the image. Field tracking is not beautiful by any means, but it is very much in the eye of the beholder.

4.2 Visibility Effects

The main visibility effect considered essential for almost all simulation tasks is fog. Other related effects such as haze, ground fog, scud and in-cloud conditions are normally handled as variations to the fog implementation. The mathematical approximation of the fog value for a given distance will probably not be an exact analog of the real world. The functions effecting real world visibility are very complex and, even if they could be incorporated into the CIG system, might not appear correct due to various limitations in the display and optic devices being used. A better approach to the problem is to incorporate a relatively

E&S/Hughes

simple fog function which incorporates sufficient flexibility that it can be tuned to appear correct for a given application.

Fogging of the polygons is generally implemented by combining a predetermined fog color with the

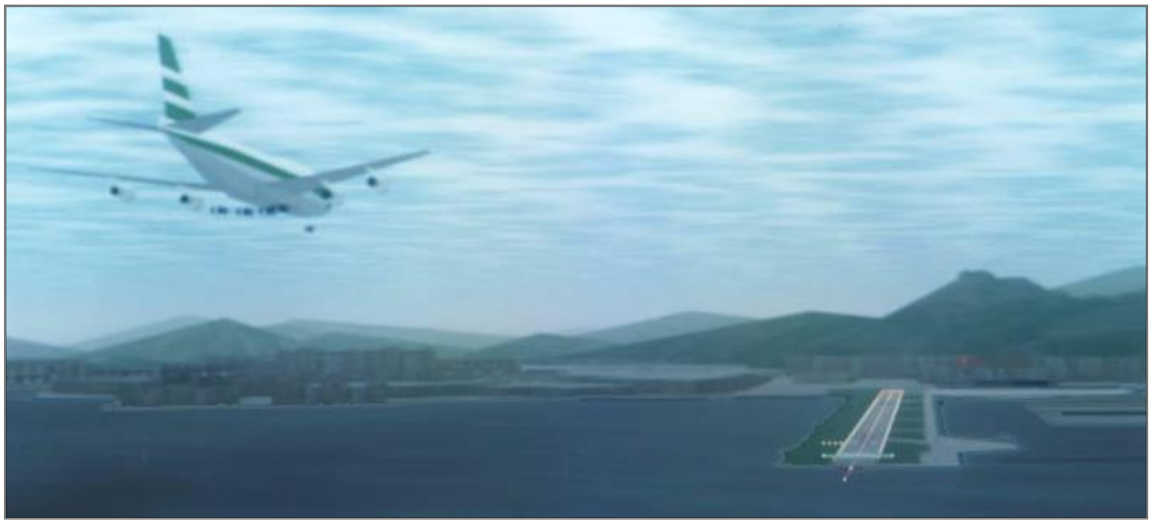


fig. 4-4 - Clear Visibility Approach

polygon color. The ratio of fog color to polygon color will depend on the visibility setting and the distance from the eyepoint. The fog color is normally white or light gray or, in beam penetration displays, as close as the available phosphor colors will allow. Some systems have a limited selection of fog colors so that different environmental effects may be simulated.

Some low cost systems do not include a fog implementation. This is not one of the features which is easily

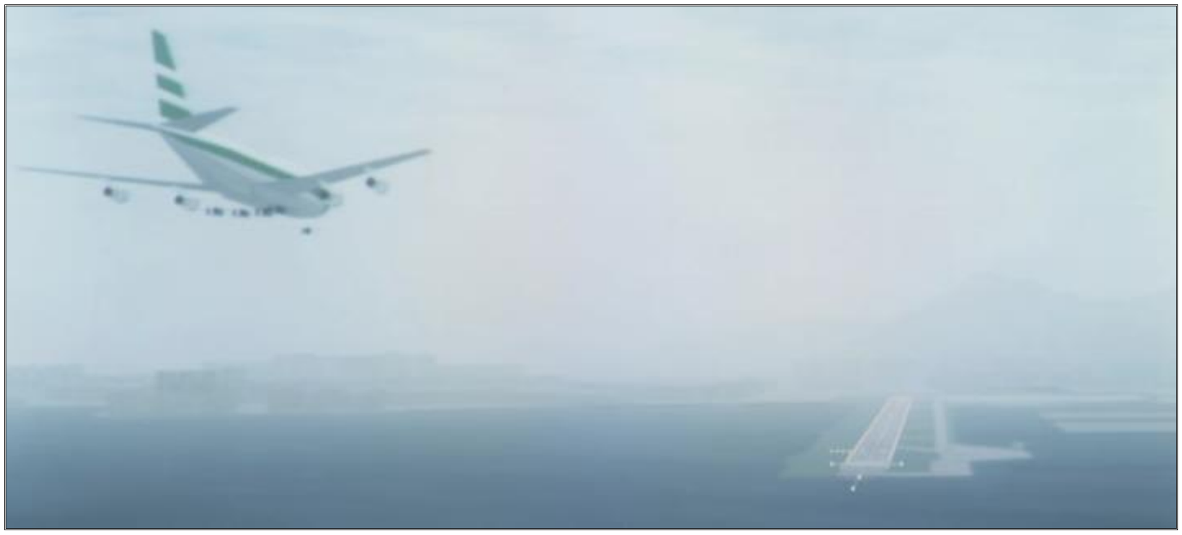


fig. 4-5 - Same approach with pixel fog.

added to a system if it was not included in the basic design of the architecture. Nor is it a feature which can be approximated in the database as a night scene may be approximated by utilizing low intensity colors for the polygons. It is technically possible to divide a large (256^3) color palette into several areas, each of which have identical basic color definitions but with increasing amounts of a fog color added to each area.

The system would then calculate the distance to the polygon each field and retrieve the appropriate version of the basic polygon color from the color palette. The problem with this approach, in addition to the processing overhead, is that the entire polygon would receive the same "fog" solution. For large polygons, such as a runway surface, this is not a viable solution and breaking each large polygon into a sufficient number of smaller ones will greatly reduce the resultant detail.

In systems which include a dedicated fog implementation it is typically accomplished in the display processor after the perspective image is complete. In itself fog will not reduce the polygon and light content of a scene. The polygons and lights are still there, their color and/or intensity is not visible due to the large amount of fog color which has been added to that portion of the display. Software and database management techniques can be implemented which will allow the system to take advantage of a reduced visual range but they are not normally part of the fog implementation. Also, in order to maximize scene density, databases are sometimes designed to operate at a reduced visibility. This is accomplished by assigning the database management and level of detail parameters to ranges compatible with a predetermined maximum visibility setting.

In some older calligraphic systems the fog is computed and displayed on a per raster line basis. This is possible because these systems use a rolled raster technique which always draws the raster lines parallel to the horizon. An entire raster line is therefore approximately the same distance away from the viewer and should have the same degree of fog applied to it. Beginning at the bottom of the display, each subsequent raster line will be further away and will therefore have more fog applied to it until some maximum is reached after which the remaining lines are totally fog color. This allows for a very cost effective visibility simulation and works quite well in some dusk/night systems. This solution has some difficulty with the day/dusk/night systems however. Not because of the day mode, but because these systems tend to incorporate more vertical objects in their databases.

The raster line approach assumes that everything on a raster line is the same distance away. This is not the case if a vertical object (a building or a mountain) rises from the horizontal plane. The amount of fog applied to a vertical object should be constant regardless of which raster lines it falls on and therefore must be handled separately from the ground plane fog. Techniques for fogging a limited number of vertical objects have been implemented but they tend to be on a per object basis which means that the entire object receives the same visibility solution. This appears incorrect for vertical objects which are large and/or very close to the eyepoint.

Another technique is to examine the distance to the vertices of each polygon and interpolate a fog effect between the near and far values. While this is a significant improvement over the implementations discussed above, it may demonstrate some inconsistencies between polygons. This is particularly true where large polygons (the runway again) are in close proximity to smaller polygons (a building near the runway). Also, if one or more of the vertices of the polygon are behind the eyepoint the system may have trouble determining where to "start" the interpolation. Again, breaking large polygons into several small ones improves this situation, but at the expense of scene complexity.

The more complex systems solve the visibility problem at the pixel level. This is a much more accurate representation of the real world but it requires the system to evaluate the distance to the content of each pixel (fig. 4-4, 5). In Z buffer systems this data may already be available and can be applied with little additional effort. For non-Z buffer systems the Z depth must be calculated and/or accessed before the fog can be applied correctly. Once this data is available it may be used for other purposes such as adding texture to the polygons.

A further refinement to the Z-buffer or Z-depth fog is for the system to consider the distance along the view vector for each pixel in the image and calculate the amount of fog along that line. The fog can then be considered as layers (as in the real world) and the portion of each view vector which passes through the layer(s) will determine the amount of fog color to be blended with the pixel color. This is very effective for creating ground fog effects (see fig. 4-5a).

Light points, either calligraphic or raster, require their own fog equations. Scene elements which provide

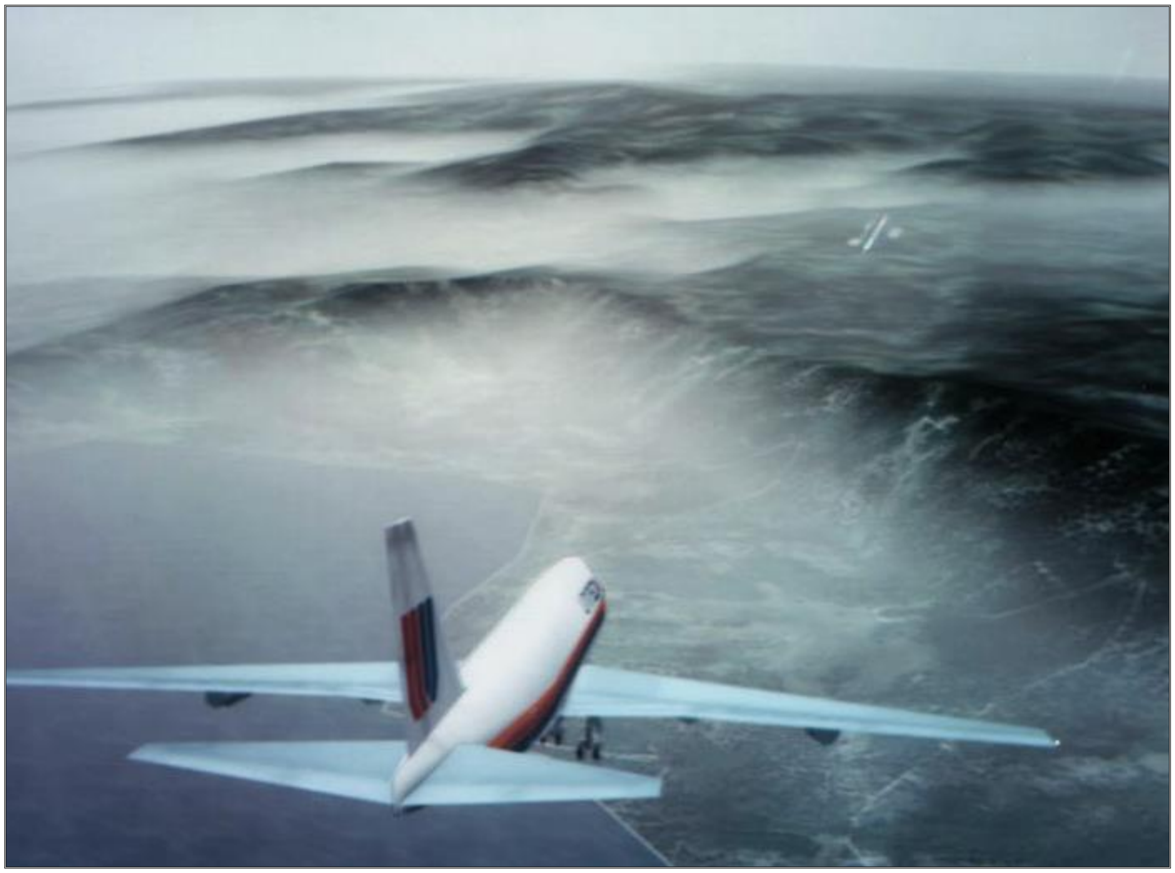


fig. 4-5a - Layered or patchy fog

their own illumination must be visible further in fog than a non-illuminated polygon at the same range. The intensity of each light point must be calculated separately based on the intensity defined in the database light string (where strings are used), any intensity control (i.e. switch setting), the amount of fog selected and the distance (slant range) between the light and the eyepoint.

When simulating sensors such as infrared, the visibility of the sensor channel will probably need to be controlled separately from the visual channel to provide the correct effects.

4.3 Transparency

The availability of transparency in a CIG system makes possible some very nice, although seldom necessary, effects. Simulating smoke or dust may be important for some military applications. Also, prop or rotor disk and canopies on aircraft models are typical applications (fig. 4-6).

A limited transparency effect can be achieved by making use of display interlace. By having certain polygons displayed in one field and omitted (or a different version displayed) in the other, small transparent objects can be represented. If these objects become too large on the display they may appear to flicker because they are being drawn at half the normal iteration rate. This technique is particularly suited to helicopter rotor disk since they should flicker somewhat.

True transparency in a CIG system is accomplished in the display processor by combining the color of the transparent polygon to that of the polygons behind it rather than overwriting it as would be the case for normal, opaque polygons. Polygons are defined as transparent in the database and are assigned a transparency mask or value. This mask will define one of available levels of transparency (some systems have up to 256 levels) which will determine how the colors are combined. In systems which are designed to disregard a pixel once it has been filled this process creates a problem. If the pixel has been filled by a transparent polygon it cannot be disregarded because another polygon may be visible at the same point. If the transparent polygons cover a large percentage of the screen the benefits of this approach are defeated and an overload may occur. For this reason transparency is not generally applicable to wide ranging visibility effects such as ground fog or patchy clouds.

A Z buffer implementation may have a different problem with transparency. The Z buffer (see 3.4.1) will average the color of the transparent polygon with the color that it finds in the frame buffer for a given pixel. This solution will be correct until a subsequent polygon is placed between the transparent polygon and the background. The correct procedure would be to now average the transparent polygon with the new background, but the original values of the transparent polygon's color are no longer in the frame buffer because they were averaged with the original background. One solution to this problem is for the system to do a course sort of the polygons into range order so that these conflicts rarely, if ever, happen. Another solution is for the frame buffer to maintain a "history" of the pixel colors it has processed.

4.3.1 Transparent Detail Switching

By far the most important and common use of transparency in a CIG system is for bringing new detail into the display database. The database management and level of detail process discussed in 3.2.2.1 can be accomplished much closer to the eyepoint if the polygons being added to the scene are faded in through several levels of transparency rather than "popped" in all at once (fig. 4-7). In a similar manner, low detail

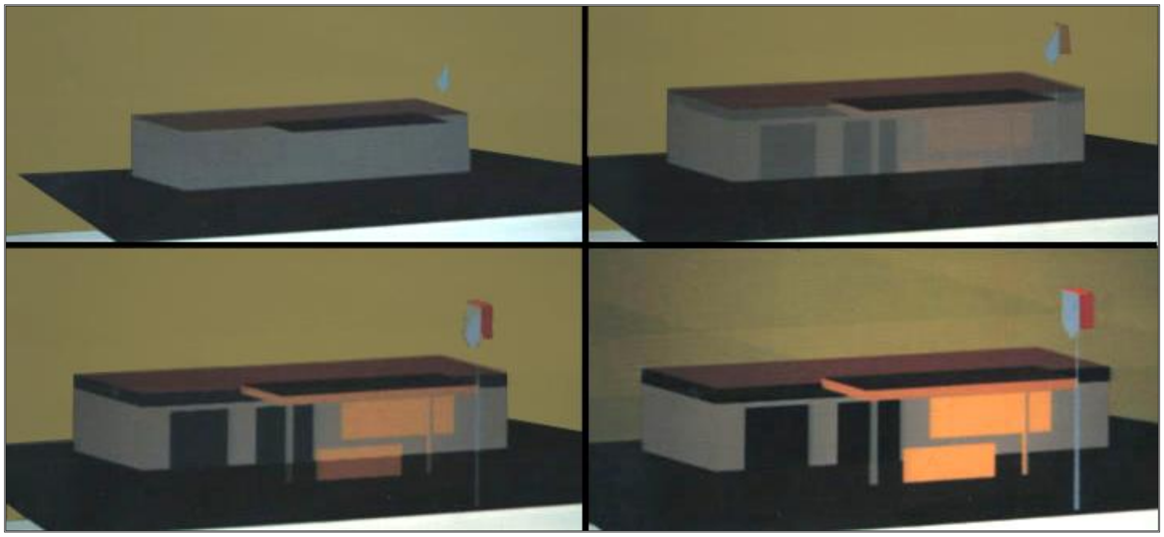


fig. 4-7 - Level of detail change made less obvious by use of transparency.

representations can be faded out at the same time that the high detail version is coming in. The levels of transparency are normally complimentary in this situation so that the sum of the low and high polygons is opaque. If this use of transparency allows the effective range of the transition to be cut in half, which is achievable for much of the scene, the resultant area which must contain high detailed objects is 25% of the original area. This means that the polygon density in this area may be increased by as much as a factor of four.

4.4 Texture

No addition to CIG technology has had more impact on the industry and the images that it creates than the advent of hardware texture. Almost every type of image generator offers some form of texture. Some of the early workstation based systems offered a screen oriented "texture" which was a mask applied to selected polygons in the display plane. This "texture" did not display true perspective effects and did not move correctly for the purpose of adding detail to the CIG scene. Although no figures are readily available, very few CIG devices which offer a texture option have been sold without it in recent years.

Texture is not new to computer graphics. Some method of modulating the surface of the otherwise featureless polygons has been sought since the mid 60s. It is only recently however that advances in the electronics have allowed cost effective implementations. These modulations have amplitude which is seen as contrast on the polygon and frequency which is a measure of how "busy" the pattern is. Two approaches to adding modulation to polygons have been explored.

One approach is to apply mathematical functions to the polygon after its perspective image has been calculated. This approach requires relatively little additional hardware in the image generator but adds to the processing time since it is a serial calculation. It may consequently reduce the number of polygons available. Although very complex math functions can be applied to create interesting and useable texture effects, these functions are somewhat difficult to work with. This is particularly true if a specific real world effect, such as windows on a building, is desired. The biggest advantage to the mathematical approach is that little or no anti-aliasing of the texture pattern is required since the texture is added after the perspective calculations and the effects of distance on the amplitude and frequency of the pattern can be handled as part of the basic process.

Most of the suppliers of CIG equipment have opted for the texture map approach. A number of texture patterns or maps are stored in special memories (additional hardware). The polygons in the database will include a reference to one or more of these patterns and other information, such as scale and position, to instruct the system on how the patterns are to be mapped onto the polygon. If the map is scaled smaller than the polygon, which is often the case, the pattern is repeated until it reaches the end of the polygon or is managed out of the image by an anti-aliasing process. The additional hardware to perform these functions tends to be a parallel implementation which may not effect the number of polygons available in the system. Depending on the implementation, however, it may be that all polygons cannot be textured.

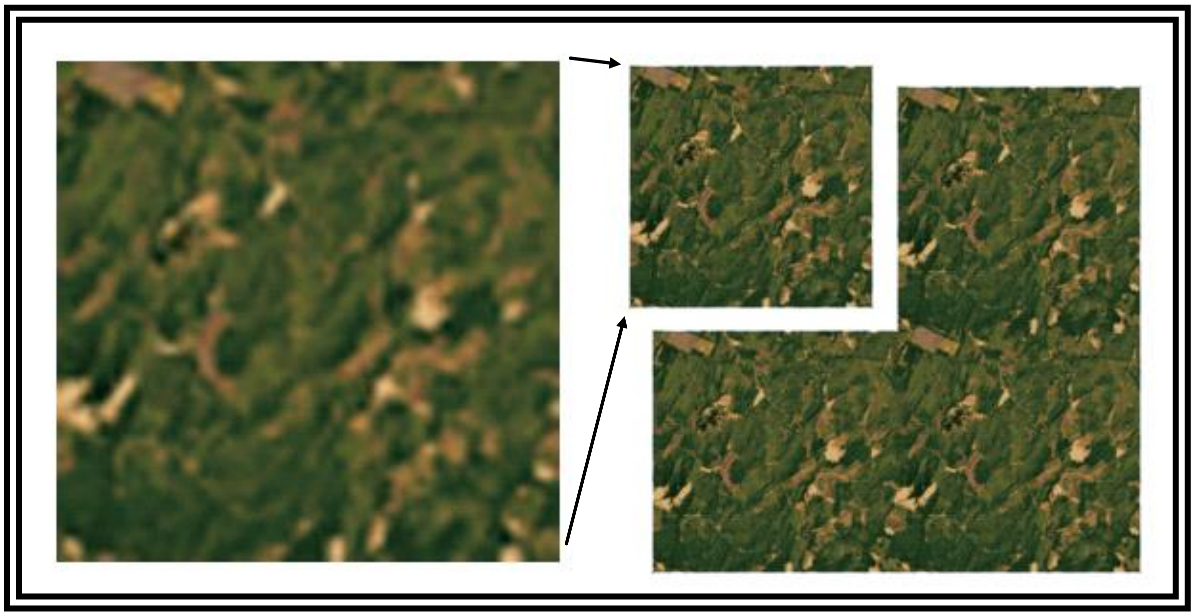


fig 4-10 - Texture map used for geo-typical terrain

The texture maps are stored as an array. Typical sizes are 32^2, 64^2, 128^2, 256^2 or 512^2. Each cell is referred to as a texture element or texel. The amount of data available for each texel is a function of the number of bits allocated to each. This, in turn, will depend on the implementation and other system features which manipulate the texture image. As we will see below, these values have several applications. It may also be possible to define sub-maps which use only a portion of the above arrays or to combine maps such that rather than repeat a map the processor will go on to a second map.

Texture maps add a huge number of "effective polygons" to any CIG image. A runway depicted with a single polygon in a non-textured system may consist of over 750,000 discernible "polygons" with the application of a high frequency texture map. This kind of density, while not applicable to the entire database, provides the observer with a great deal of speed and height information in the critical landing phase of flight training. The texture map in fig. 4-10 is derived from an aerial photograph. A lower resolution version of the map is shown at left to give an idea of the texel array. The content of the basic photograph is manipulated so that the map will tessellate with itself. The image at right in fig. 4-10 shows four copies of the map at normal resolution. It is very difficult to see the boundary between copies.

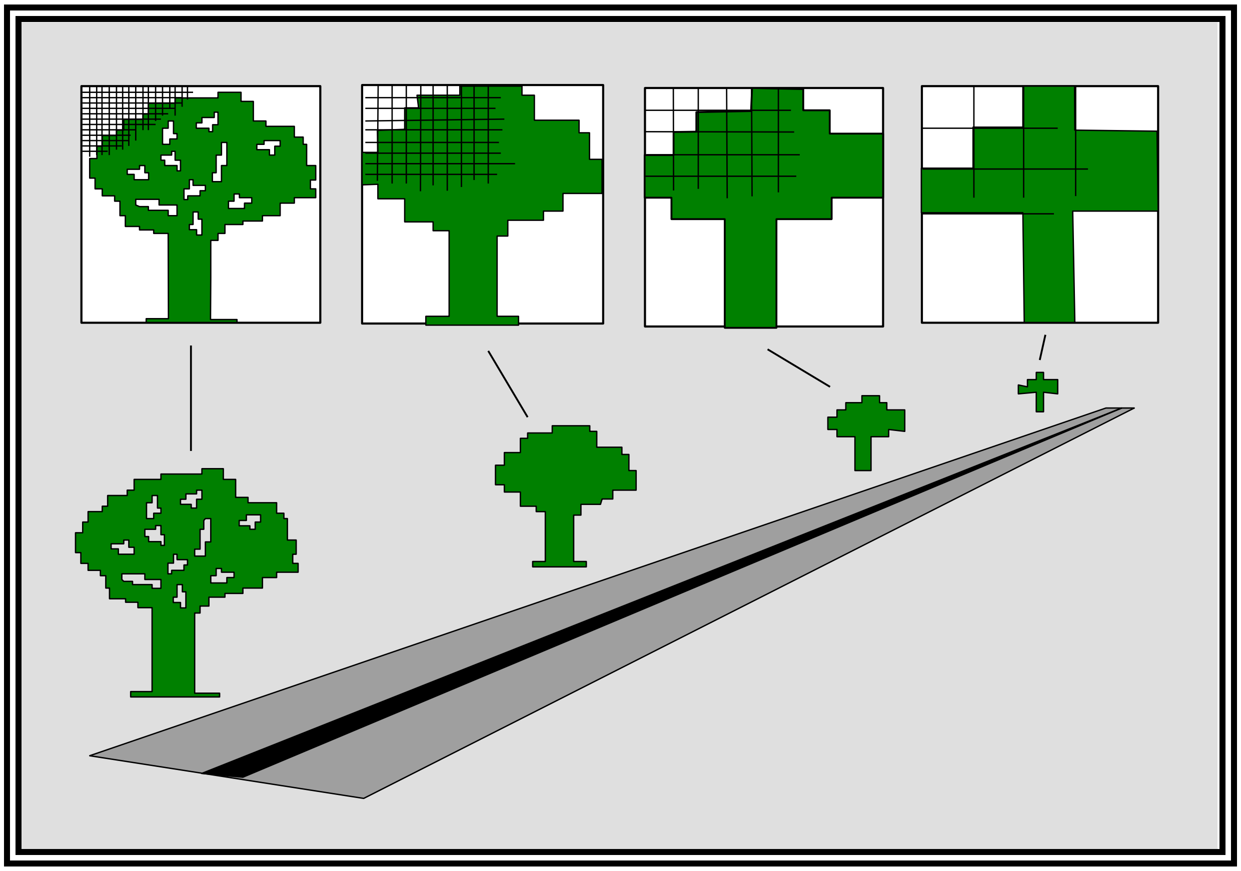


fig. 4-12 - Lower resolution maps used at greater distance.

It is not surprising that this much detail added to the scene creates additional aliasing concerns. The hardware required to provide even a limited texture capability represents a significant increase to the overall system. This hardware typically applies the texture on a per pixel basis so the degree of additional hardware is directly proportional to the amount of texture provided and the number of pixels in the scene. The various anti-aliasing approaches discussed above involve calculating a much larger number of sub-pixels and averaging them to create the actual displayable pixels. The combination of texture and anti-aliasing implies enough texture hardware to calculate texture solutions for all of the sub-pixels being processed. This would require an unreasonable amount of additional hardware (with today's technology) so other approaches to anti-aliasing of texture edges and features are utilized. As with the polygons themselves texture aliasing can be reduced by careful attention to the contrast and/or color changes within the map. One of two approaches to further reduce aliasing is generally applied.

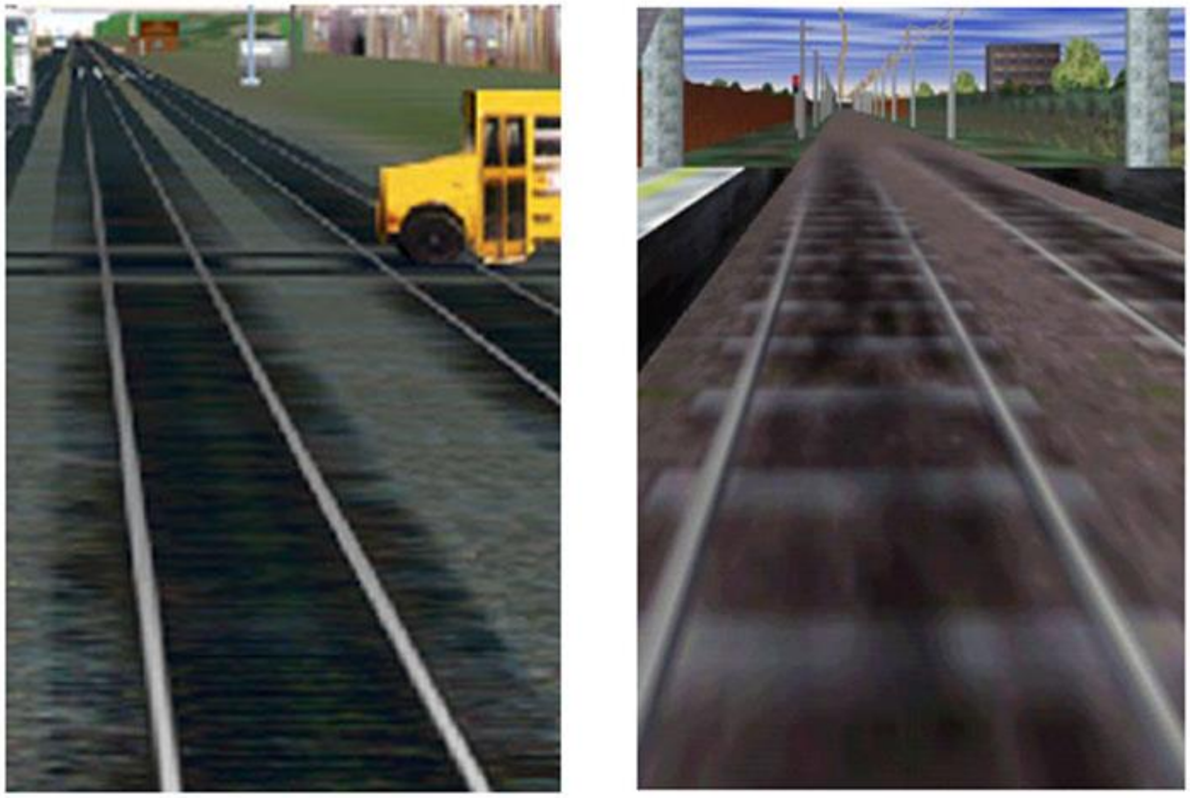


fig. 4-12a - Rail in texture map (right) is blended out, polygon rail (left) remains visible.

A level of detail technique similar to that used for normal objects in the scene may also be used for texture. Several versions of the maps, each with lower contrast and/or frequency, are utilized (fig. 4-12). These lower detailed versions of the map also require less resolution (i.e. fewer texels). As each pixel is processed the system selects texels from the appropriate version of the texture image. In general, the system will not use texels that are perspectively smaller than the pixel. This allows only one texel to contribute to the pixel, thus avoiding color conflicts with could lead to aliasing in the image. This approach requires some additional on-line storage (25%) for the low detail versions of the maps. It may also utilize some form of blending or averaging between levels of detail to hide any distracting effects (see 4.4.5).

This type of texture implementation is referred to as MIP maps. MIP stands for “multum in parvo” which means something like “many in one place”. While MIP mapping does limit texture aliasing, it also removes detail from the image. When the scene is viewed from an elevated angle, as from an aircraft, this effect normally appears to approximate the conditions of the real world. When viewed at shallower angles, as from a ground vehicle, the reduced aspect angle causes the texels to become small at unnaturally short ranges. Any detail included in the texture map is therefore blended out much closer to the eye than may be desired. The example in fig. 4-12a shows two approaches for the railroad track in a locomotive visual system. Both use texture for the track bed (i.e. ties, gravel, surrounded ground surface). In the left image the track itself has been rendered with polygons so that the rail will be visible at greater distances. In the right example the track is included in the track bed texture. The detail of the rail blends into the track bed at a relatively short distance. This also occurs if texture is used to represent lines and markings on runways and taxiways in airport databases.

Some new systems are addressing this problem by doing additional processing of the texels, particularly when they are viewed from a shallow angle, as in the example. The system is able to recognize that a texel that is a pixel wide in one direction may be covering several pixels in the other. If this is the case, an averaging process is activated which will provide a better solution to the texture content of the pixel. This allows the texture to be used for a wider range of applications, such as the rails in fig.4-12a.

Another approach to reducing texture aliasing involves reducing the contrast in the map as a function of the distance to each iteration of the pattern. This function considers both the assigned scale of the map and the perspective size of each copy of the map and reduces the amplitude of the pattern accordingly. The result is a very gradual reduction in contrast as the pattern gets smaller in the scene. Controls may also be available to bias the range at which the clamping occurs. The modeler would assign a high contrast and/or high frequency map a shorter clamping range than that applied to a smoother map which continue to be well behaved when smaller on the screen.

4.4.1 Modulation Texture

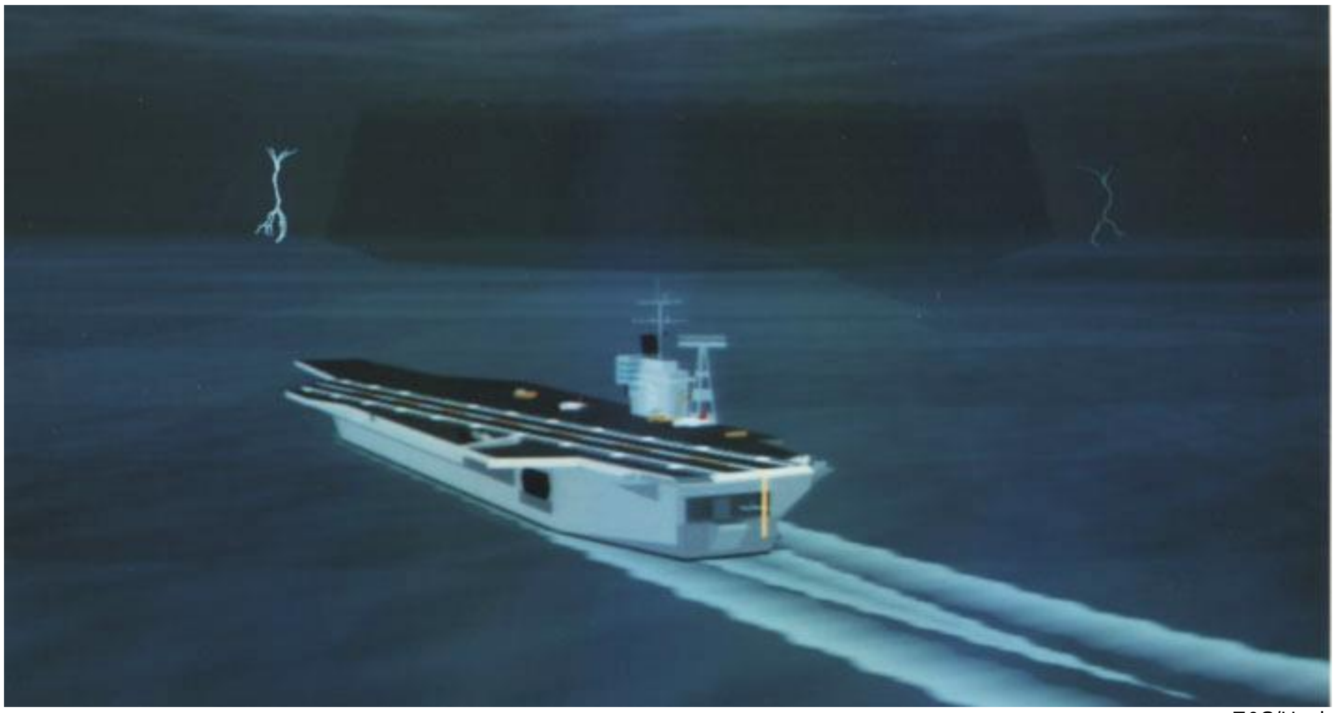

E&S/Hughes

fig. 4-13 - Aircraft carrier model with transparent texture applied to wake.

Modulation texture is the type generally available in all of the CIG systems today, particularly the smaller, mid-range image generators. This technique utilizes the values in each texel to effect some quality of the polygon. Normally this is the intensity of the polygon. The pattern will appear on the polygon as light and dark areas to create the desired effect. Depending on the implementation and the type of anti-aliasing utilized, the texture may add to the intensity of the polygon or subtract from it or both. This will effect the appearance of textured polygons compared to adjacent, non-textured ones.

Some systems will allow the map values to modulate other qualities of the polygon, such as color and transparency. Color blending, or color modulation, changes the color of the polygon according to the values in each texel. Texel values at the low end of the scale are assigned one color and values at the high end are assigned another. Any value in between is then a proportional blend of the two colors. This is a

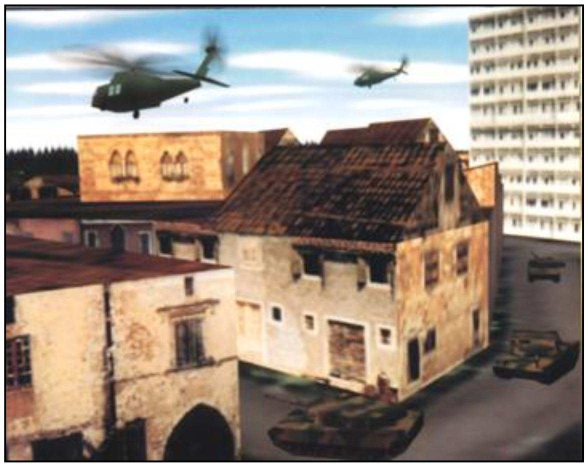


fig. 4-15 - Example of photo-texture.

very useful feature and provides some very realistic effects but is sensitive to aliasing, even with the above techniques applied. Transparency modulation requires that the system have a transparency capability as described in 4.3. This feature provides polygons with holes or translucent areas in them as well as jagged edges (fig. 4-13). It has the same limitations in the display processor as the basic transparency capability discussed above but it only applies to the portions of the polygon which are translucent rather than the entire polygon. Like color blending, this feature provides some very useable effects but tends to alias.

.

4.4.2 Silhouette Texture

This capability is variously referred to as cell texture or contour texture. It is currently only available in some high end systems. The purpose of this type of texture is to create very complex shapes without the aliasing problems associated with normal transparency modulation. The texture maps are the same type of array as the modulation maps discussed above. The utilization of the value in the texels is what is different. The data in each texel serves two purposes. One is to differentiate the texel as being transparent or opaque. The other is to describe how far the texel is from the edge of the silhouette. This latter geometry information is used in an anti-aliasing type process at the pixel level to keep the complex, high detailed shape stable as it moves through the image. Additional modulation maps of some type are normally applied to the silhouette to give it further definition. The "standard" application of this technique is to create "photo quality" trees, normally for military applications (fig. 4-14).

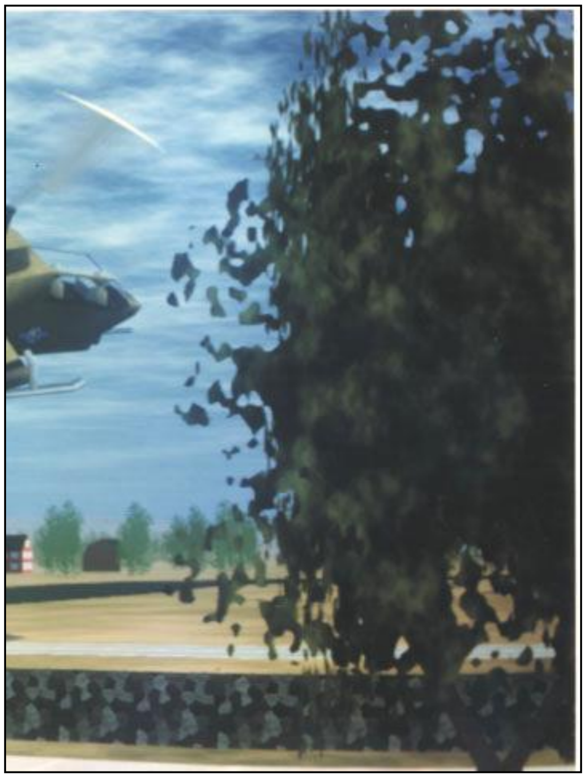


fig. 4-14 - Example of silhouette texture

4.4.3 Color Texture

A third type of texture implementation, sometimes referred to a photo-texture, contains color information in the texture map. The polygon may become a vehicle for inserting the map into the image and provides only the boundaries and the surface plane to which the full color map is applied. The terrain map shown in fig. 4-10 is a full color map where the colors are derived from the aerial photograph. Here again some form of blending may be utilized between levels of detail so that distracting changes in scene content are avoided.

Because these maps contribute the entire color content of the polygon a large number of them may be required. Otherwise large portions of the database will be identical. If the map is utilized to represent the walls and windows of buildings (fig. 4-15) several maps will be required so that all of the buildings are not the same color. Allowing the map to "blend" with the assigned color of the polygon and/or be affected by the intensity of the polygon may alleviate the problem.

Color, or photo-texture, typically requires more storage in texture memory. Whereas modulation maps are essentially monochrome, in a color map the intensity of the red, green and blue components must be stored. I transparency is also included, the color map will require four times the texture memory (for a map of the same size and resolution).

4.4.4 Texture Bump Maps

Bump maps are a form of modulation texture which is only recently making its way into real-time CIG devices. In order for bump mapping to work, and be available as a capability, it is necessary that some form of pixel rate shading be implemented in the system. This will generally take the form of Phong shading (see 5.3.2). All shading effects make use of a normal (to the polygon) vector which is compared to a vector from the light source(s) to calculate the effect of the light(s) on the polygon color and/or intensity. Each texel of a bump map contains angular data with bends the surface normal. It is necessary for the system to calculate the effect on each pixel's color and/or intensity relative to the illumination. This will provide an appearance similar to normal intensity modulation, except that the appearance of the map, and the polygon surface, will change as the eye and/or the illumination source(s) move (fig. 4-16). This technique, when combined with other texture effects, should reduce the flat look that most textured surfaces exhibit.

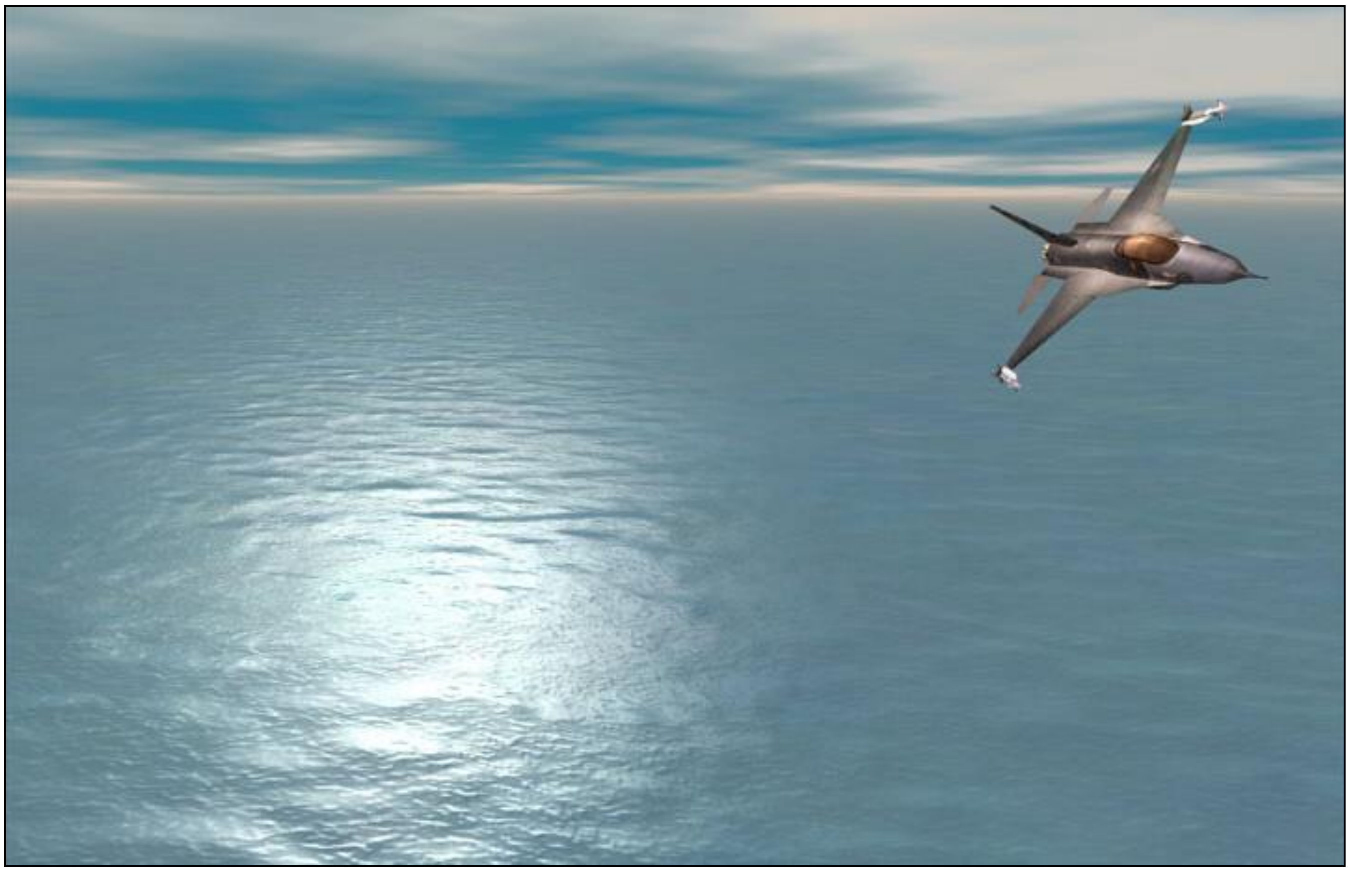

E&S

fig. 4-16 - Bump map texture effect - appearance of map changes with sun/view angle

4.4.5 Texture Positioning

How the texture maps may be assigned to the polygons is an important consideration in the potential applications of texture. Each map will include an origin, typically at one corner. This origin is related to a point relative to the polygon. It may be one corner of the polygon, the model origin to which the polygon itself is related, or some other point inside or outside of the polygon. There are four potential methods of applying texture to polygons.

One approach is to assign a point on each polygon for the texture to "start". A straightforward way of doing this is for the origin of the map to be placed at the first vertex referenced in the polygon and one of the axis of the map to be aligned with the polygon edge between the first vertex and the second. If some other starting point and direction is required the mechanism for defining that point must be included in the database structure.

Another approach is to "wrap" the map around an object constructed from several polygons. In this way the map does not stop at the edge of the polygon but continues onto the next polygon. This implies that the point where the map origin is placed is outside of the second and subsequent polygons and will have to be derived for each polygon. This will hopefully be an automatic process included in the modeling tools or the image generator. Note that these points, although not contained within the polygon, do have to be in the plane of the polygon. This approach will not provide a continuous pattern if the geometry of the polygons is too complex. In particular, if the object bends in more than one direction, problems similar to creating flat maps of the round earth occur.

The third approach is to project the texture onto a polygon from a plane different from that of the polygon(s). A typical example of this technique is to apply a single terrain map to a large number of polygons representing rolling hills. The map is projected onto the polygons from a horizontal plane. This approach does provide a continuous texture map but will have problems if the polygons are vertical (for this example). This technique is also used to apply "decals" to aircraft models or similar complex shapes.

A fourth approach sometimes utilized is to assign the texture to a plane different from that of the polygon. This is a strange effect which can only be depicted as a dynamic case. The texture pattern will move relative to the polygon edges due to the parallax effect between the two planes. This technique is very effective in creating the illusion of reflections such as on a water surface or a glass window. For example, if a subtle wave pattern can be applied to the surface of a lake and a cloud pattern assigned to the lake polygon but perspectively calculated from a plane several thousand feet below the lake surface the result is a water surface which appears to have ripples and to reflect the clouds from above. The cloud pattern should be the same one used in the sky and should be the same distance below the lake as those in the sky are above it. This is Modeling Trick #328.

In some recent systems this out-of-plane technique has been implemented to form an approximation of proper reflectance (which would require ray tracing). The texture “plane” is actually spherical, which gives a better result on anything put large polygons. The model in fig. 4-17 makes extensive use of the effect to create a chrome surface. A close examination of the reflections (which is not possible when the model and/or the eyepoint is dynamic) reveals that the texture used for the reflection may have nothing to do with the environment around the model which is being “reflected”. If the situation allows, a texture map can be created from an image of the local environment. This better supports the illusion of reflection.

4.4.6 Texture Characteristics

With so many available systems offering texture, a wide range of capabilities is to be expected. Some of these have already been discussed in the proceeding sections. Others are covered here.

There are several capacity issues in addition to how many polygons can be textured. The number of maps available at any given time is one concern. Even though the same map may be used to create different effects by changing its scale and applying it to different colored polygons, a relatively large number of maps on-line will help in creating a complex database. Some systems will allow maps to be added to the system on-line as part of the database management process. This must be done very carefully so that maps in view are not replaced or moved.

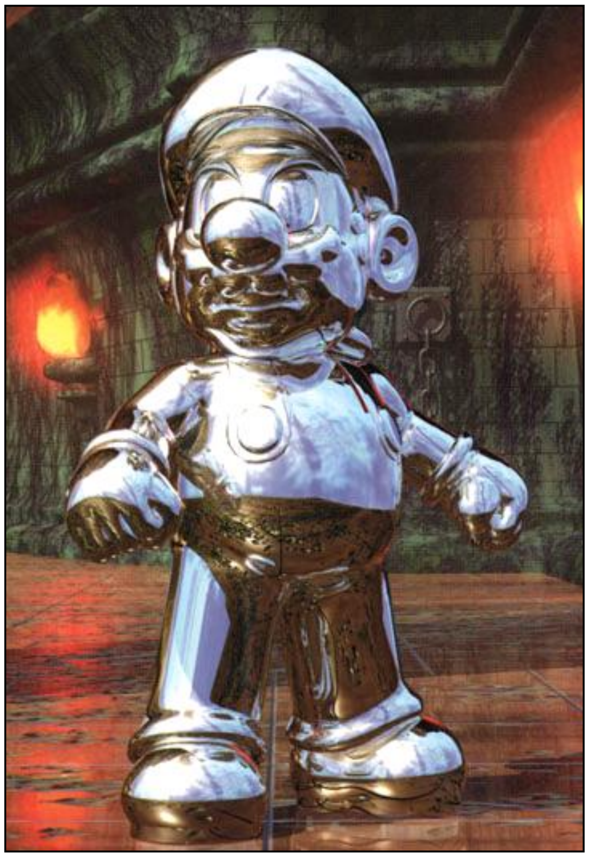


fig. 4-17 - Reflective texture

The capability to apply more than one map to a polygon is also very useful. Two or more maps, each at a different scale, will help to disguise the repetitiveness of the maps when they are applied to a large polygon. Also, a large map will be visible from a greater distance with a smaller one emerging as the polygon is approached and the frequency of the large map no longer provides optimum visual flow.

Texture in the early calligraphic systems was applied only to horizontal surfaces. This was a cost effectiveness issue aimed at the civil airline market rather than a technology problem. While this did limit the applications somewhat, these were very useable systems which provided a significant increase in scene content over the non-textured versions of the same devices. This implementation is often referred to a 2D texture in that the only polygons it can be applied to are flat, ground plane polygons or similar horizontal surfaces. More powerful systems, which can texture any polygon regardless of its orientation are said to provide 3D texture. In both cases it should be understood that the texture itself is a two dimensional effect. Systems which provide true three dimensional texture effects, such as trees or rocks created purely from texture hardware as opposed to a collection of textured polygon, are not yet available.

Texture motion is also a very useable capability. This is normally assigned in the database and will move the map(s) relative to the polygon they are applied to. Typical applications are water surfaces which depict waves or blowing dust or smoke when combined with transparency effects.

Bi-linear interpolation (or blending) is a feature of most current systems which works at the pixel level to blend across the edges created by the texels (fig. 4-18). This is particularly effective when high contrast has been used in maps which appear large on the display. The result is a gradual blend from one intensity or color to another without an edge in between. While this has many advantages it can limit the applications of the texture if it is not a selectable feature. There are some effects, such as runway cracks, which require that the texture edges be visible.

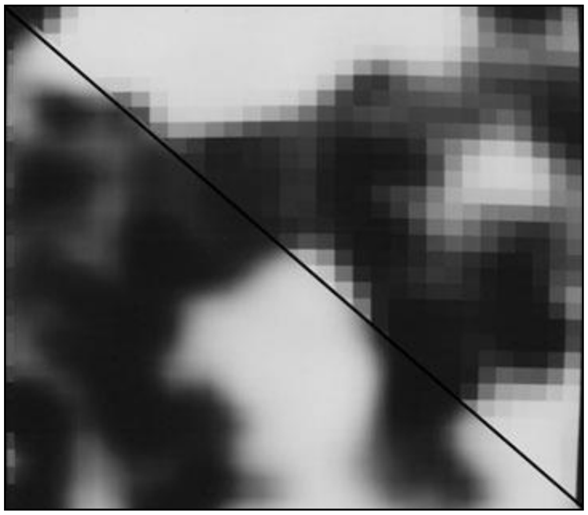


fig. 4-18 - Texture with and without bi-linear interpolation

An extension of bi-linear blending is tri-linear interpolation. This is a similar process intended to reduce texture aliasing. In addition to the blending of texels within the texture map, tri-linear blending also averages the intensity and/or color of the texels between levels of detail in a MIP map implementation. In addition to reducing aliasing this further blending also eliminates the stepping from one MIP level to the next as the eye proceeds through the database.

Skewing the texture map to fit a non-rectangular polygon is also a feature of some systems. Rather than assume that the map is always a square array, for example, this technique may align one axis of the map with one polygon edge and the other axis with an adjacent edge regardless of its angle to the first. This feature can be used to create some very interesting effects, particularly when combined with other features, such as texture motion.

4.5 Height Above Terrain

Height above terrain (HAT) is a CIG feature available in the larger systems which calculates the distance of the eyepoint, or some other point, above the modeled terrain. This may be used by the simulator to drive a radar altimeter or to allow landings on non-flat runways (fig 4-19). If this capability is not provided in the CIG system, the host computer may have to store its own representation of the terrain. Other applications of HAT may be associated with the positioning of dynamic models in the scene. A tank, for example, may be driven in X, Y and altitude from the host with the HAT function providing the Z component and the attitude of the vehicle. Similarly, when a missile is fired from the simulated aircraft, HAT may be used to calculate ground impact.

The incorporation of height above terrain may require additional hardware as well as software. Perhaps as much as an additional channel processor, a portion of a channel or a separate computer. The normal CIG process will not, in itself, provide the required information because the point directly beneath the eyepoint is seldom in the field of view of any of the channels being computed. The plane equation of the polygon may be sent to the host in addition to or in place of the actual height. This equation may easily be solved to determine the height and may be more useful when simulating landings on non-flat surfaces.

As with all other aspects of CIG applications, transport delay may be a consideration in making use of HAT data. For the more critical training tasks, such as landings, it may be necessary to implement a "prediction" mechanism similar to the extrapolation routines used on the host data. The plane equation may be used to predict what the altitude will be in a few frames if it is known, or required, that the plane

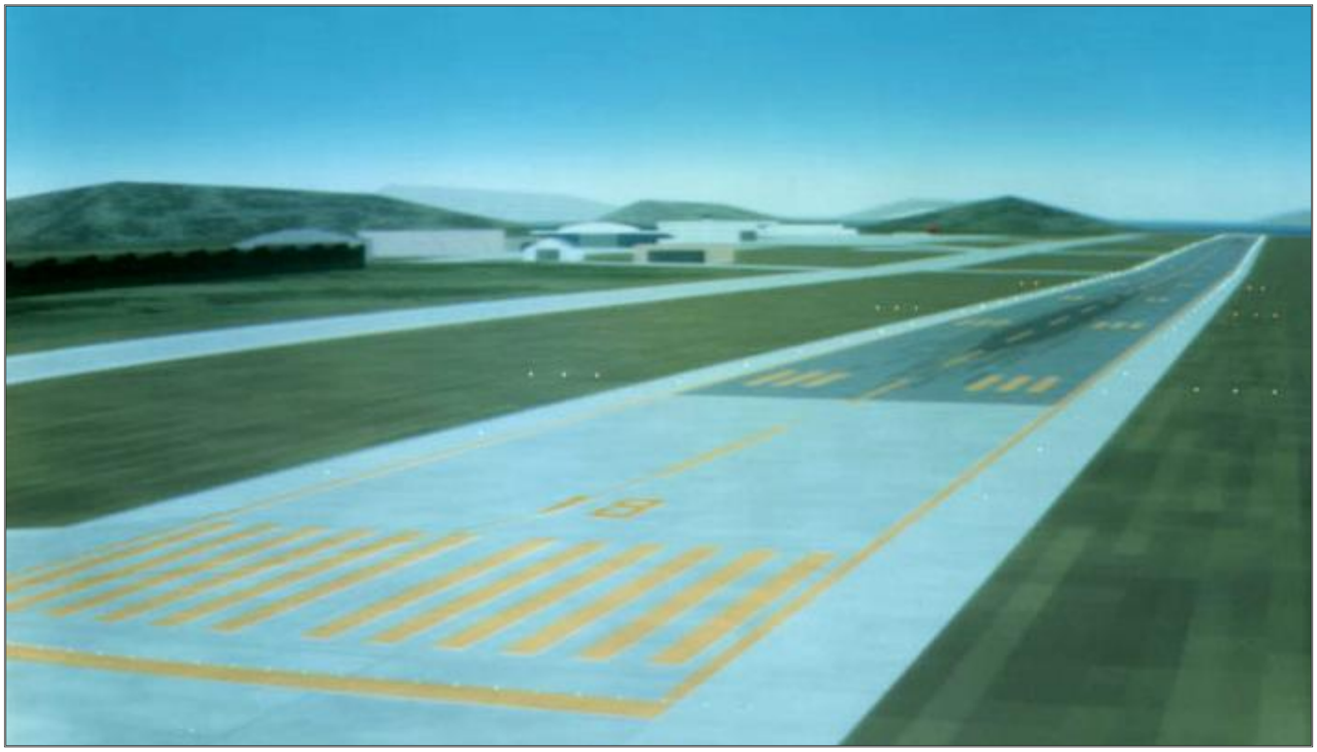

E&S/Hughes

fig. 4-19 - Airfield with non-flat runway surface

will not change in the course of the event. A sloped runway which is otherwise planar could be simulated in this manner. For more complex terrain it may be necessary to position the point for which the height is being calculated a predetermined distance in front of the simulated vehicle. This distance will change depending on the speed of the vehicle. This complex implementation will therefore require that the host be able to calculate and send to the image generator the coordinates of one or more points and then make use of the data it receives back from the visual system to influence the training maneuver in question.

4.6 Collision Detection

Collision detection is somewhat similar to HAT in that it must also test points related to the simulated vehicle against database features which may not be in the field of view of any of the visual channels. Also, the objects in the database are typically simple versions of the visual representations rather than the actual polygons used in the visual database. These are called collision volumes. The points tested against these objects are positioned so that they represent the extremities of the vehicle being simulated.

Each iteration of the image generator the collision detection process will compare as many of these test points as it can against all of the active volumes. During subsequent fields it will test the remainder of the points and then start over again. When one of the test points is found to be inside a collision volume, this information is passed back to the host for its use (to indicate a crash for example). In some systems it will also be possible to determine what type of object was hit based on codes modeled with the volumes and sent to the host, where they are compared to values in a table of available object types. It may also be possible to determine which test point, and therefore which portion of the vehicle, actually collided. These inputs might then be used by the host to determine what type of damage or failure to simulate.

There may also be a limit on the number of volumes which can be active simultaneously. These will be managed into the system via the same database management process that brings in the polygons and lights.

Since the collision volumes are invisible they can be activated close to the eyepoint so a relatively low number of active volumes does not necessarily limit their effectiveness.

The background nature of this serial task implies that it may take several system iterations before every test point is tested against every volume. It is possible then, for a point to enter and exit a volume without being detected. This will depend on the speed of the eyepoint, the number of test points and the number of active volumes. These parameters will have to be considered during the database design phase. Some systems will calculate the path each sample point has traveled between one test and the next to determine if the point has passed through any volumes.

4.7 Line of Sight Ranging

Systems which generate the Z depth of each pixel, either from priority or visibility calculations, may utilize that information to provide the range to a specific pixel in the image. The host will position a cursor within the display image (similar to dynamic coordinate system positioning) and the CIG system returns the distance to the contents of the pixel which the cursor overlays. It may also return the color of the polygon and other information.

This capability is used to aid in the simulation of laser range finders, and other modern weapons systems, which generate range information. By pointing the cursor at a known object in the scene and comparing the polygon/pixel data against expected returns, the host can determine if the object is being occulted by other scene elements. It is required that the object and/or polygons being evaluated are within the field of view of a channel which has this capability. Various aspects of this feature are also very useful to the database designer during model debug and tuning.

4.8 Dual and Offset Eyepoint

In a typical CIG application all of the channels are computed from the same eyepoint with a different look angle and sometimes a different half angle or field of view. There are cases, however, where different channels may require different eyepoints.

The most extreme examples of this are applications where a single CIG device has been used to supply imagery to two simulators. This has been done a few times, normally using large systems, before the smaller day/dusk/night systems were available. The main impact on the system in this configuration is to the database capacity.

As we have seen, a large portion of the image generators task is the retrieval of the active database from the available database on the system disk. The active database is then culled down to the display database in the process of creating the image. If two divergent eyepoints must be supported then there are two active databases and the retrieval and culling process must be done twice. This generally implies that the resultant display database must be half of that originally intended for the system.

The degree of any further impacts on the system performance depends on several issues including where the channel specific hardware splits off from the system hardware. In general, the resultant performance for each eyepoint will be less than half that achieved in the normal, single eyepoint mode. Recent applications which considered this approach have generally opted for two of the smaller systems now available.

A less impactive application is an offset eyepoint which is referenced to, and not far from, the main eyepoint. Recent examples of this implementation include sensor simulations where the sensor is mounted on the vehicle at some significant distance from the viewer. Many helicopters have infrared or other electro-optical sensors mounted on the rotor mast several feet above the pilot. A typical flight tactic would require the pilot to position his aircraft behind a rock or tree while the sensor peers over the top. Obviously, to simulate this maneuver, the sensor channel requires a different view of the environment than the out-the-window visual channels. This is accomplished by applying an offset to the appropriate image generator channel. The difficulty implied by this requirement again depends on the channel architecture of the system.

Some simulations will require an optically guided weapon or RPV which will separate from the simulated vehicle and fly its own path while sending images back to the vehicle. These could very well suffer some of the same difficulties as the dual eyepoint mode discussed above. If the range of the guided vehicle is not too great however, this task can be handled as a dynamic offset eyepoint. A recent trend seems to be to utilize a separate, and usually smaller, CIG device for these applications rather than an additional channel of the out-the-window visual system. Some further implications of sensor simulation are discussed in 5.2.2.

4.9 Non Linear Image Mapping (NLIM)

This system feature, sometimes referred to as Curvilinear Mapping, may be required in any application where the image is projected onto a curved screen and/or the projector is off axis to the screen. Many modern air combat simulators, for example, utilize a large dome with the pilot in the exact center and the projectors as close to the center as possible without creating a conflict with the pilot's head or each other.

The problem is that the perspective image created within the Geometric Processor is calculated for a flat image plane. This works out well when the display device is one or more flat (or nearly flat) CRTs or flat screens. The image of a polygon will have the same straight edges on the display as were calculated within the IG. If the screen is curved, however, the straight edges from the IG will appear curved on the screen. If a calligraphic projector is utilized it may be possible remove most of these distortions with the large number of alignment adjustments available. If a non-calligraphic projector is used, however, the distortions must be removed in the image generator.

In fact, distortions are added to the image in such a way that the projected image will appear correct. One approach is to subdivide the image into a large number of much smaller images. Each of these "sub-field of views" will be a slightly different shape in order to match the curve of the screen. The polygons are then calculated for each of these small areas in the same way it would have been done for the whole channel. The result is a mosaic of small images which combine to form the large, pre-distorted image. This approach will require additional hardware to calculate the large number of different field of view parameters and perform the polygon clipping that each will require. Also, since these sub-areas are not necessarily rectangular, there may be gaps and overlaps which will appear as aliasing in the image. The magnitude of this problem will depend on how many of the sub-areas are required.

Another approach is to calculate the polygon vertices in the normal way and map the resultant screen locations to a "corrected" location for the desired distortion. A look up table is utilized which relates each address (x,y) in the calculated image to a distorted display address. This process will position the vertices of the polygons in the corrected location but the edges between the vertices remain straight (and curved on the screen). The degree of error is directly related to the length of the edge. Up to a certain length the error will not be apparent in the final image. The solution then is to keep all of the edges short by subdividing the polygons. This is an automatic process within the image generator which will segment only those edges which exceed a certain tolerance each field (as opposed to building the database with a large number of very small polygons). This process will also require additional hardware, though not as much as the above solution, and also reduces the polygon capacity of the system.

Any NLIM solution will involve displayed pixels which differ in size. In some cases these differences are extreme and may cause brightness variations, particularly near the edges of the display. Application of texture and fog to the resultant polygons may also be difficult.

4.10 Shaders

Vertex, Fragment/Pixel and Geometry Shaders are relatively new features only recently available with PC graphics boards. These have been used in the past to render very complex images in non real-time applications. The fur seen on some animals in motion picture special effects or the hair on humans in some applications are created with fragment shaders that create a three dimensional effect from a two dimensional texture map. As with many such features throughout the history of computer generated imagery these capabilities are now finding their way into real-time applications of PC-IG systems.

The shaders are small programs that are referenced by polygons in the database. The programs are loaded at system start up and stored in the memory of the graphic processor, similar to texture maps. When the polygons are rendered the programs are activated and create the desired effects. It is also possible to pass information into the shaders that determine how the effect is rendered. These are called "uniforms" and include such things as the time of day, various colors (e.g. fog), the elevation and attitude of the eye point, and other variables that can alter the appearance of the scene.

One current limitation of shaders is that they are very parallel tasks and know nothing about the condition or state of surrounding vertices or pixels. This tends to limit some of the effects were shaders might be applied. The processing load that shaders impose on the overall system throughput is difficult to predict or compensate for (at this time). The addition to a database of scene features and objects that require shaders for their correct rendering will create new problems to the already difficult task of database documentation and version control. This will be particularly problematic when moving databases between applications and/or systems.

Another factor in the use of shaders is the replacement of the normal rendering pipeline. When shaders are not used (as in most previous systems) there is a standard rendering process that performs the "normal" effects (e.g. fog, illumination). If a shader is used to render a pixel this normal pipeline is bypassed. All rendering effects, in addition to those added by the shader, must be performed by shaders as well. This issue is further complicated by the intention of the manufacturers of the graphics processors to eventually remove the standard pipeline from their products and assume that shaders will do everything. This means that the simulation industry will need to find ways of using shaders within the next few years if they plan to continue to use commodity graphics processors in their simulation image generators.

4.10.1 Vertex Shaders

Vertex shaders are used to manipulate polygon vertices. These can be used to create dynamic events such as explosions (particularly effective when combined with particle systems) or other effects that require altering the size and/or shape of polygons. Having the shader manipulate the surface normal at the vertices can also alter the shading effects on the polygon.

4.10.2 Pixel/Fragment Shaders

Pixel shaders operate on each pixel associated with the referenced polygon or object. As one would expect, pixel shaders are most effective on the rendering functions that occur at the pixel level. These include illumination, texture, fog and z-depth. For example, a manipulation of the apparent z-depth is used in the above example of rendering realistic hair or fur. While it is impossible, or impractical, to model each hair with polygons, it is possible to address the pixels that include the hair(s) and visually separate them from the surrounding hairs.

These shaders are sometimes referred to as fragment shaders because they must work on all of the anti-aliasing samples, or fragments of pixels, that form the image.

4.10.3 Geometry Shaders

These are the latest addition to the powerful rendering tools supplied with current graphics processor units. Unlike the vertex shader, which can only manipulate existing vertices, the geometry shader can create polygons or groups of polygons to implement effects. This will allow a significant increase in rendered polygons without adding too much processing to the front end computer. One example of a possible geometry shader application might be for shader based light points. Rather than a light point being programmed as one or two polygons, with a texture map applied to depict the round light source, the light could be a single point that is positioned in the database. The position of each light would be calculated and sent to the geometry processor where it would be expanded into the textured polygon. This would provide a more efficient of light point processing.

4.10.4 The Future of Shaders

It is clear that all futures graphics processors will include more and more capability in these areas. Effects that overload the system today may not do so in the next generation of processors, or the next after that. It is also too soon to understand how shaders might be applied in the simulation industry as a whole. The board manufacturers have created these effects in response to requests from the gaming and non real-time rendering industries. The simulation industry has much different technical and logistic requirements. It will be very interesting to follow the proliferation of these effects in the next few years.

One issue that simulation must address is synchronization across multiple PCs. This can be between channels, or within a channel if the image is assembled from multiple GPUs. Dramatic effects such as explosions, fires, smoke, etc. lose some of their appeal if they are discontinuous at image boundaries.

5. DATABASE IMPLEMENTATIONS

As we have seen in the preceding sections of these notes, CIG systems are extremely complex devices with a wide range of possible features and capabilities. Combinations of these features, and tradeoffs between them, are further complications. When all of the performance evaluations and comparisons of system features have been accomplished, the final determination of a system's suitability for a given application will almost always come down to the visual database or model. The suitability or usefulness of the database will, in turn, often depend on the Database Engineer’s understanding and utilization of the capabilities which the system contains and how they may be applied to best satisfy the simulation requirements. Not only can a proficient modeler create databases which will allow the image generator to fulfill the requirements of a complex training or engineering task, but he can also provide databases which will allow the same, or similar, image generator to fulfill applications which may be very different. For this reason, the same basic CIG hardware may be incorporated into a simulator to train pilots of fast, low level jets as well as a ships bridge simulator to train students in the docking and maneuvering of a destroyer.

In the sections below we will examine some of the basic components of the database and look at some of the options the modeler has in providing solutions to a wide range of simulation problems. In some recent procurements of large military systems, the database cost is very nearly as large as that of the image generator hardware itself. The database also tends to be the area of the CIG device that the user (or owner) will modify or build himself. The system hardware is almost never changed and the software only occasionally, but many simulator operators, in all applications of the technology, will construct their own databases. This is because the area represented by their database will change (a new runway) or their training requirements change (a new flight maneuver) or they need an additional gaming area (a new airport). Often, however, it is because building databases is fun!

In the course of these notes we have discussed many of the components which make up the CIG image. Polygons, lights, texture, etc. all contribute to the eventual scene processed and displayed by the system. The actual process differs from system to system but one thing is constant. All of these components are stored in the database. We will now examine and define these scene elements in more detail.

5.1 The Scene Graph's Relationship to the Visual Database

Creating Databases for real-time 3D rendering is more complicated than just making good looking models. Unlike building a 3D model that is pre-rendered where there are no limits on performance due to time, you also need to consider the special needs of your run-time scene graph rendering process. The most important step toward making a believable 3D environment is to build models that run fast enough so they move smoothly through the field of view of the observer.

For Example, with a lot of care and attention to detail a Database can be built to exactly represent the lines, curves, volumes and size of any scene. After building all the objects needed to represent an environment they can be placed in such a way as to duplicate the actual environment. If the observer is placed at any fixed height, angle, and direction they will see an exact representation of the real world environment would look like as if it is being viewed from the same height, angle, and direction. As far as the observer is concerned it may as well be the real world. However, if this simulation is only an unordered group of polygons extensive enough to represent an environment with no regarded for scene graph structure, than this will be where the real world illusion starts to fall apart.

As soon as you move just one object, be it the observer's eye point or a model, the scene graph rendering process must run its course again and again at very high rates of computation. This seems simple enough for a computer, but like watching a movie, the scene the observer's eye sees must be updated 30 times per second at a minimum. Meaning that the rendering process must run every half a seconds. If you don't have at least 30 frames per second the eye will notice a jumpy change on the position of objects (optimally 120 times per second or higher to make it really smooth). This is not the way objects behave in the real world and therefore starts to add the fakeness of the simulated environment. An environment built with disregard for scene graph structure will force the scene graph to render all the polygons at one time, overloading the process and making it impossible to reach your frames per second goal. This can be elevated by taking the time to understand how the scene graph rendering process works.

The Scene Graph is a small but very important program that is used to render visual databases. A computer can take a calculated environment and using the information laid out by a visual database build and render a scene on a monitor for an observer to view. Although the Scene Graph is small in comparison to other system programs it takes a huge amount of computing time to complete each rendering pass. To add to the scene graphs workload a scene that has moving parts needs to go thought multiple rendering passes to make up the full rendering process. To accomplish this the Scene Graph is built to optimize the speed at which the information in the visual database is processed by a specific combination of hardware and software referred to as the Image Generator (IG).

The problem is that to accomplish this in the most efficient manner the Scene Graph expects the visual database to be laid out in a very specific way. If the visual database is not laid out in such a way as to facilitate the Scene Graph rendering process the opposite affect will accrue. This will causes the Scene Graph to overload with information and slow down below the minimum goal of 30 frames a second.
There for, understanding the relationship between the scene graph and a visual database will assist in structuring a visual database that can greatly improve the performance of the Scene Graph rendering process With an understanding of how the scene graph works a Database or model can be broken down into sections of information that the scene graph can then use to evaluate and render, or not render, based on the eye point of the observer.

Building models so they run smoothly for real time 3D rendering, as you can see, takes just a little more effort and planning than just making them look accurate. Taking into consideration the special needs of the scene graph rendering process, you goal frame rate can be achieved and objects will behave properly adding "reality" to your simulated environment.

A Scene Graph consists of objects, called nodes, arranged in a descending tree structure. The IG can create one or more sub-graphs and attaches them to a hierarchy as needed. The connections between nodes always represent a directed relationship: parent to child. Nodes are refined into two subclasses: Group and Leaf. Group nodes connect together one or more child nodes which can be more Groups or Leafs. A group node can point to zero or more children but can have only one parent. A leaf node has no children and only one parent. The information contained in the leaf nodes are described later and consist of standard graphics library commands use to render the visual scene. OpenGL and DerectX are the two major graphics library used in the industry. To the level this topic will be covered they are similar enough so, we will be use OpenGL as our example.

A scene graph organizes and controls the rendering of its leaf nodes. The scene graph draws in a consistent way that allows for concurrence. The hierarchy of the scene graph encourages a natural spatial grouping of the leaf nodes found at the ends in the graph. Internal nodes act to group their children together. A group node also defines a spatial bound that contains all the geometry defined by its descendants. Spatial grouping allows for efficient implementation of operations such as proximity detection, collision detection, view frustum culling, and occlusion culling. The nodes in a direct path between the scene graph's root and the leaf define the state of the OpenGL commands found in a leaf node. Because a leaf's OpenGL context only relies on a linear path between the root and that node, the scene graph can decide to traverse the hierarchy in whatever order it wishes. It can traverse the hierarchy from left to right and top to bottom. The only exceptions to this rule are spatially bounded attributes such as lights and fog.

Group nodes are the elements used in constructing the hierarchy of a scene graph. All group nodes can have a variable number of child node objects including other group nodes as well as leaf nodes. The group node is a general-purpose grouping node. Operations on Group node objects include adding, removing, and enumerating the children of the Group node. There are also subclasses of Group nodes with additional attributes sometime called object nodes.

A Branch group is the root of a sub-graph of a scene that may be compiled as a unit, attached to a hierarchy, or included as a child of a group node in another sub-graph. A sub-graph, rooted by a Branch Group node, can be thought of as a separate model. For example a river, a building, a vehicle, and even a small as a blade of grass.

The Transformation Group specifies a single spatial transformation that can position, orient, and scale all of its children. The Transformation Group can also be used to externally reference group nodes allowing the Scene Graph to reference a shared Branch. This provides a mechanism for sharing the group in different parts of the allowing for faster processing, from within a branch of the scene graph. Any number of External Reference leaf can refer to the same group node.

The Switch group node allows the Scene Graph application to choose dynamically among a number of sub-graphs. The Switch node contains an ordered list of children and a switch value. The switch value determines which child or children the Scene Graph will render. This can be used for special effects like animations, level of detail, and damage states.

The Leaf nodes define the OpenGL entities such as geometry and lights, provide special linking and instancing capabilities for sharing hierarchy, and provide a view platform for positioning and orienting a view in the virtual world. The leaf node is an abstract class for all scene graph nodes that have no children. The leaf node specifies all geometric models. It contains two OpenGL components: a reference to the model geometry and its appearance component. The model geometry defines a shape's geometric data. The Appearance component specifies the model's appearance attributes, including color, material, texture, and so on.

The Sub-Leaf (or sub face) is a node used for defining decal geometry on top of other geometry. The Sub-leaf specifies that its children should be rendered on top of another leaf model and that they generate coplanar geometry. Examples of this include painted decals or text on surfaces.
The Bounding leaf node defines a bounding region that can be referenced by other leaf nodes. This is to define a region of influence for effects like Fog and Light nodes, or an activation region like Background and Clip nodes. The bounding region is defined in the local coordinate system of the Bounding node.
The Background (or Sky Box) node defines either a background color or a background image that is used to fill the window at the beginning of each new frame. It also specifies an application region in which this Background is active. If multiple Background nodes are active, the Background node that is closest to the eye point will be used.

The Clip node defines the far clipping plane used to clip Group nodes in the scene graph. It also specifies a region in the active viewing frustum for the culling of unseen group nodes. A Clip node is active when its application region intersects the viewing frustum.

The Fog node defines a common set of attributes that control fog, or depth cueing, in the scene. The Fog includes a parameter that specifies the fog color and the boundaries that specify the region of influence for the fog node. Objects whose bounding volumes intersect the Fog node's region of influence have fog applied to their color after lighting and texturing have been applied.

The Light node is a class that defines the properties common to all the lights associated with it. A light has a color, a state; on or off, and boundaries that specifies the region of influence for the light. The light lights objects whose bounding volumes intersect the Light's region of influence.
By following a direct path from the root thought hierarchy to a leaf node the scene graph can accomplish its sole proposes. Which is to establish which OpenGL commands will be needed in the rendering of a graphical model for a scene. There for it is also imperative to understand the relationship between the Scene Graph and OpenGL to assist in structuring a visual database that will run at top performance on the Scene Graph.

OpenGL is a low-level graphics library specification that makes available to the Scene Graph a small set of geometric primitives - points, lines, polygons, images, and bitmaps. OpenGL also provides a set of commands that allow for the manipulation of these geometric objects in three dimensions. Using the provided primitives, together with commands the Scene Graph controls how these objects are rendered into the framebuffer.

OpenGL runs in an event loop. This means that, upon program instantiation, some initialization code may be executed and then OpenGL falls into an infinite loop accepting and handling events. In the event loop there is a series of processing stages called the rendering pipeline. Although this is not a strict rule of how OpenGL is implemented, it provides a reliable guide for predicting what OpenGL will do. Geometric data follow a path through this order of operations that includes evaluators and per-vertex operations, while pixel data such as images and bitmaps are handled by a different part of the process. Both types of data undergo the same final step of raterization before the final pixel data is written to the framebuffer. All data, whether it describes geometry or pixels, can be saved in a display list for current or later use. A display list is a group of OpenGL commands that have been stored for later execution. When a display list is invoked, the commands in it are executed in the order in which they were issued. Most OpenGL commands can be either stored in a display list or issued in immediate mode, which causes them to be executed immediately. Vertices eventually describe all geometry in OpenGL and evaluators are used to convert the vertices into primitives. Spatial coordinates are projected from a position in the 3D world to a position on the screen.

Each geometric object is described by a set of vertices and the type of primitive to be drawn.
The simplest primitive is a Point and is represented by a single vertex. The vertices are specified by x, y and z coordinates assigned to a three dimensional space. Points are the building block for all other geometric object. The term Line, in OpenGL refers to a line segment, not a line that extends to infinity in both directions. Lines are specified in terms of the vertices that make up their endpoints. To save processing time lines may share one of their end points creating a line segment that reduces the amount of data the program must handle. Polygons are the areas enclosed by single closed loop of line segments. Polygons are typically drawn with the pixels in the interior filled in, but you can also draw them as outlines or a set of points.

In OpenGL, there are a few restrictions on what constitutes a primitive polygon for example, the edges of a polygon cannot intersect and they must be convex.

With OpenGL, the description of the shape of an object being drawn is independent of the description state. As a geometric primitive is drawn, each of its vertices is affected by the current OpenGL state variables. These state variables specify information such as line width, line stipple pattern, color, shading method, fog, polygon culling, Textures. Each state variable has a default value. The values of the state variables remain in effect until changed. Changing between states is the single most inefficient process that OpenGL preforms. A good Example of how important state changing is can be seen in the way that the color state is applied. By drawing all the objects with the some color at once the program does not have to waste time changing the states. When a particular geometric object is drawn, it's drawn using the currently specified coloring scheme. OpenGL first sets the color, and then draws the objects. Until the color is changed, all objects are drawn in that color or using that color scheme allowing the program to run efficiently.

For vertex data, the next step is to arrange the objects in three-dimensional space and select the desired vantage point for viewing the composed scene. To change the way an object is viewed the program can either move the object or move the viewer. The two methods for accomplishing this with OpenGL are called modeling transformations and viewing transformations. Viewing transformation is analogous to positioning and aiming a camera. Modeling transformation is analogous to positioning and orienting the model, or objects to be drawn. When modeling transformations are performed, a series of matrix multiplications are actually performed to affect the position, orientation, and scaling of the model. The scaling command multiplies the current matrix by a matrix that stretches, shrinks, or reflects an object along the axes. The translation command multiplies the current matrix by a matrix that moves an object by the given x-, y-, and z- values. The rotation command multiplies the current matrix that rotates an object in a counterclockwise direction about the ray from the origin through the point (x,y,z).(Ferguson,64)
After processing the modeling transformations, objects closer to the viewpoint may obscure objects further from the viewpoint. To draw a realistic scene, hidden surface removal must be performed to ensure that obscured parts of objects are not visible. Hidden surface removal is achieved through the use of a depth buffer and depth testing. When drawing occurs, a depth, or z-value, is associated with each pixel and stored in the depth buffer. A depth test is performed for each vertex in the scene. During a depth test, the depth buffer is checked to see whether another vertex has previously been drawn that would obscure the current vertex. If so, the vertex is not drawn this is call Z-Buffering.

Once the all transformations have calculated the finale layout of the primitives then attributes of the state variables are applied to the polygons. The states are assigned by the Scene Graph and determined from specified lighting conditions. The final step is to obtain and pass textures onto the objects. On a computer screen pixels are controlled by differing amounts of red, green, and blue light mixed to gather and create the desired resultant color. OpenGL use a color-index of values to represent the different color combinations. Each vertex in a primitive is assigned one of these values called its RGB value according to its state.

Light effects are very important in OpenGL, for without the use of lights, a 3D model will still look two-dimensional. OpenGL treats the position and direction of a light source just as it treats the position of a geometric primitive. In other words, a light source is subject to the same matrix transformations as a primitive. Two types of light sources are provided: directional and positional. A directional light source is considered to be an infinite distance away from the objects in the scene. Its rays of light are considered parallel by the time they reach the object. A positional light is near or within the scene and the direction of its rays is taken into account in lighting calculations. Positional lights have a greater performance cost than directional lights due to these additional calculations. Shading can be a calculation intensive state depending on the type of shading needed by the program. With flat shading there is just a single color applied to each polygon that is to be drawn. Flat shading is the fastest because no additional calculations are needed when changing all the vertex colors of a polygon to the same color. However, it is possible to specify a unique color at each vertex. OpenGL will smoothly interpolate the color between the vertices this is known as gouraud shading. OpenGL will linearly interpolate the RGB values for the pixel values along the edges between the vertices causing a gradual shift of color.

Texture mapping guidelines are assigned to polygon so that a texture may rapidly be placed in the scene during the final step of the rendering pipeline. Texture mapping allows for two-dimensional images to be applied to a three-dimensional surface. The only restriction on textures is that there pixel height and width must be a power of two in size. Most IG will allow you to send a non-power of two texture but then OpenGL will need to calculate extra textiles to fill in the texture and remap it, slow down the processing time.

At this point OpenGL has completed all calculations need for converting the primitives, which are transformed with related color, depth values and guidelines for adding textures to the polygons.
The pixel data that describes textures takes a different route thought the OpenGL rendering pipeline. Processed by the Pixel Operations the textures are taken from an array in system memory where they are processed from one of a variety of formats into the proper components. Next the data is scaled and positioned by texture mapping guidelines, and all the data sent to the final step of Rasterization. Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the framebuffer. Line width, point size, shading model, and coverage calculations to support antialiasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square. To be displayed on a monitor, the fragment must then been drawn into the appropriate framebuffer. Thought The OpenGL rendering pipeline the Scene Graph can support three different modes for rendering: immediate mode, retained mode, and compiled-retained mode. These three represent a potentially large variation in graphics processing speed and on-the-fly restructuring. So, it is very important to know exact which mode your IG uses.

Immediate mode allows maximum flexibility at some inherent cost in rendering speed. The application can either use or ignore the scene graph structure. By allowing the program to draw geometry directly a model can be edited and changed without recompiling or the scene graph.

Retained mode allows for a great deal of flexibility but not as much as that provided by immediate mode. The tradeoff is providing a substantial increase in rendering speed. All group nodes defined in the scene graph are accessible and manipulabe. The scene graph itself can be manipulated fully but the OpenGL commands in the leaf nodes are compiled. The application can rapidly construct the scene graph, create and delete leaf nodes, and instantly observe the effect of any Transformation. In retained mode, the Scene Graph knows that the program has defined models, knows how the program has combined these models into compound models, and knows what behaviors or actions the program has attached to objects in the database. This knowledge allows, the Scene Graph to perform many optimizations. It can construct specialized data structures that hold an object's geometry in a manner that enhances the speed at which the, the Scene Graph system can render it. It can compile object behaviors so that they run at maximum speed when invoked.
C

ompiled-retained mode allows the, the Scene Graph to perform an arbitrarily complex series of optimizations including, but not restricted to, geometry compression, scene graph flattening, geometry grouping, and state change clustering. This mode ensures effective graphics rendering speed in yet one more way. A program can request that, the Scene Graph compile a hierarchy or a model. Once compiled, the program has minimal access to the internal structure of the models or scene graph. A compiled model or hierarchy consists of whatever internal structures the Scene Graph wishes to create to ensure that model or hierarchy render at maximal rates. Because the Scene Graph knows that the majority of the compiled model or hierarchy components will not change, it can perform an extraordinary number of optimizations, including the fusing of multiple models into one conceptual model, turning a hierarchy into compressed geometry, or even breaking a hierarchy up into components and reassembling the components into new conceptual hierarchy.

The Scene Graph loop and OpenGL rendering pipeline run independently but are reliant on one another to complete the task of displaying a 3D scene. This high-level view of the render loop permits concurrent implementations of the Scene Graph and OpenGL. First the application must construct its scene graphs. It does this by constructing scene graph nodes and component objects and linking them into self-contained trees. The application next must obtain a reference to any constituent nodes or objects within that branch that it may wish to render. Then OpenGL is able to construct shapes from geometric primitives, thereby creating mathematical descriptions of objects. Then arrange the objects in three-dimensional space and select the desired vantage point for viewing the composed scene. During this stage, OpenGL might perform other operations, such as eliminating parts of objects that are hidden by other objects. The scene is then calculated with the color, lighting and shading of all the objects. The final step is converting the mathematical description of objects and their associated texture information into pixels on the screen. The application must repeat this process for each rendering pass but does not have to wait for a pass to complete to start a new pass.

Every model and environment have unique requirements that can help or hinder the speed of the scene graph rendering process. Through understanding the Scene Graph these situation can be anticipated and models can be prepare for them. There are several different standard methods for constructing visual databases that will assist the application to structure the Scene Graph that will greatly improve the performance of the rendering process.

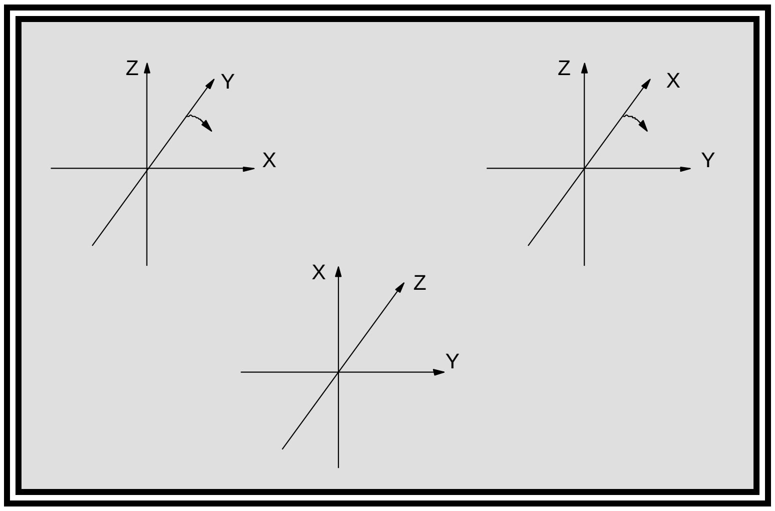
Culling unseen group in the Scene Graph is an extremely fast process when compared with rendering the same group in OpenGL. By balancing the amount of time spent rendering and culling the process, as a whole will be speed up greatly. How to balance cull and draw is best explained in Creating Models for Simulation published by Multigen-Paradigm it says to "adjust the sizes of the group (nodes).... Decrease the group sizes by arranging more groups if the runtime system spends too much time drawing. An entire group must be redraw if even a small area of the group is in the field of view, so it is a good idea to keep the size of groups as small as possible."(3-4)

Nesting group nodes can also utilize the speed that culling provides. This is done by putting many group nodes that have models in the same area of the 3D scene under one group node. This allows the scene graph to cull large section of the hierarchy without having to process every group node. This is usually reserved for large models or whole environments. One way to improve the speed that the Scene Graph can finish a rendering pass is to reduce the amount of information that it has to look through. One way to reduce the database information is to reuse vertices information that is already there. The best way to implement this is explained in this excerpt. "With a t-strip, three vertices are required for the first triangle in the strip. One new vertex is required to complete subsequent triangles. This is because each triangle shares two vertices with the triangle that came before it." ("3Dgate" 16) This method does requires that each triangle be adjacent to the last but the amount of information needed to define the entire set of triangles together is greatly reduced as compared to defining each one separately. If only ever other triangle in a visual database was t-striped it would cut the amount of information need to render the triangles in half. By stringing even more triangles in to the t- striped the database becomes exponentially smaller.

Another way of enhancing the ability of the Scene Graph to process a visual database is to limit the number of state changing. As explained earlier by grouping all the primitives with the same color you can save on state changes. This also hold true for all state changes such as textures and shading. Even paying attention to texture use can improve rendering speeds. Another suggestion from Creating Models for Simulation shows how to do this "Create a single texture of individual textures patterns, or a texture collage, that you can load into the texture palette. Each texture pattern can be fence selected and used as a subtexture. This is useful if you need to apply several different texture patterns, but you want to conserve texture memory."(4-26)

These are only some of the many ways that a visual database can be laid out for efficient rendering. Some improvements are so specific that they only apply to a few cases. There are even cases were these tips will contradict each other. This is seen in the case were you have polygons that are lying adjacent to one in other, some with the same color and some with different colors. You can ether share vertices or try to arrange them to save on state changes but not both. Here it is up to the Database Engineer to make the decision on which tip to use. The case of sharing vertices vs. saving state changing is one of the easer choices to make. As environments get lager the amount of choices can become overwhelming. Setting up a large series of simple test case and modeling them in different combination is the only way to find the best solution for improve efficacy. This is where truly take the time to understand the relationship between the Scene Graph and a visual database will assist in structuring a visual database that can greatly improve the performance of the Scene Graph rendering process. For without this knowledge a modeler has no basis for establishing a testing plan that will relieve the hide abilities of the IG.
The pure magnitude of possible decisions makes it imposable to come up with a standard set of rule to govern all Database creation. This make it extremely important for a Visual Database Engineer to understand how there scene graph rendering process works and how in turn controls it OpenGL commands. This is why building Databases for real time 3D rendering is more complicated than just making good looking models.

5.2 Model Origin, Axes and Units



One aspect of the database which is common to all CIG systems, and in fact, all computer graphics devices, is the model origin. This is the 0,0,0 point from which all database references are made. What may differ, however, is the orientation of the axes which radiate from this point as shown at right.

In some smaller systems the X axis is considered to be vertical, the Y horizontal and Z to extend into the display. This is very similar to the eventual display coordinates the scene features will have after processing and is left over from the CAD/CAM roots of these types of systems.

Larger systems tend to consider the Z axis to be vertical or "up" in the database. Either X or Y may be considered "north" depending on the system. Rotations may be clockwise or counter clockwise, again depending on the system.

One of these approaches is not particularly better than another. A transform may be required to relate the visual system, and its database, to the host coordinate system but this is straight forward. The only difficulty arises when a modeler works with more than one system simultaneously and must keep different system axes sorted out.

The location of the origin in the database will depend on the system, the application and the modeler. In some CIG systems applied to commercial airline simulators, the origin is positioned at the touchdown point of the main runway with the "north" axis extending down the center line of the runway (regardless of the actual runway heading). Within the host computer's visual program, the model origin is equated to the latitude and longitude of the touchdown point, the model runway heading is equated to the actual runway heading and the model elevation is equated to the airfield height. All of these parameters are available from the simulator's radio aids data. This process ensures that the visual database runway and the simulator avionics are aligned for at least one runway in the gaming area.

Aligning additional runways, particularly a crossing one, is sometimes difficult because the position and heading values come from two, or more, different sources. Actual airport landing aids, such as Instrument Landing Systems (ILS), are aligned with the runway through mechanical and electronic means. The actual latitude and longitude of these facilities is therefore not required to any degree of accuracy which means that the position data available for the simulator may be off by many feet. In general, the database, having been constructed from accurate maps and engineering drawings, will be more accurate and the radio aids data should be adjusted to match.

With the advent of Height Above Terrain feedback (see 4.5) in recent systems used by the airlines it has become more common to model the airport at the real world height and heading. These systems may utilize an automatic model load capability which will compare the latitude and longitude of the eyepoint with the lat./long. of the airfield which is one of the model parameters. The closest model (i.e. airport) is then loaded as the simulator "flies" into its proximity. The HAT data will allow the simulator to land on the runway even though it may be a non-flat surface. In principal, the simulator does not need to know which database is active.

In databases which cover a very large area the origin will normally be placed in the center to take full advantage of the storage characteristics of the database values (see below). Some modelers prefer to place the origin in one corner of the gaming area such that only positive values are required. Other origins to be considered are the local origins around which dynamic coordinate systems and objects to be instanced are constructed.

Dynamic coordinate systems (see 5.5) have their own origin which is directly or indirectly referenced to the database origin. The host or the visual computer will provide a position and attitude for the DCS. The placement of the origin within the dynamic objects will depend on how and where the object is designed to move. An aircraft model, for example, should have the origin at the approximate center of gravity so that it will maneuver correctly when it is pitched and rolled around that point. A ship model would have similar requirements. A vehicle on the ground, however, would have the origin in its center, but at ground level rather than the center of gravity. This would allow it to follow non-flat terrain more easily. There are many special cases of DCSs where the origins would be different from these and in some cases, nowhere near the object itself.

Instanced objects are similar to DCS objects in that they are built around their own origin and then moved into place in the database. The translating and/or rotating of these objects may occur in the modeling tools as part of the construction process or it may occur in real time during the processing of the database for display. In the latter case the database will contain a reference to the object and the position(s) where it will be placed. The database manager and/or the geometric processor will take the object from the active database memory and position it in required locations in the gaming area. The placement of the origin for objects of this type have some of the same considerations as the DCSs above. For many applications of this feature the origin will be in the center. This allows relatively easy placement of the object, particularly after it has been scaled and/or rotated.

The database units are generally stored in the system as 16, 24 or 32 bit numbers. The most common seems to be 32 bit. In a 16 bit computer system this will require double precision and all calculations will be done accordingly. The typical scaling of these numbers, when considered in feet, is such that the largest value will be approximately 400 miles and the smallest 1/32 of a foot. This range is generally required so that the gaming area may be made large enough for almost any application. The low end of the range allows very accurate placement of objects in the database but, more importantly, provides smooth motion when the scene elements are moving through the image.

5.3 Vertices and Edges

The vertice is the most basic building block of the visual database. A vertice is an infinitely small point defined by an X, Y and Z value relative to the model
coordinate system. Every database artifact which must be located in the gaming area will refer to at least one vertice. Light points will be equated to a single vertex unless they are stored in strings in which case there may be a vertex defining the first light point and another defining the last point (these could be the same). In a curved light string a third vertex may be used to define the curve. In systems which require separating planes for priority purposes or database management planes to indicate sub-model switching, they are normally defined by three vertices which form the plane. A polygon may be used as the vehicle to define these points.

Some CIG systems use edges as the basic construct. An edge is described by two vertices. The structure of the edge will require a first point and a last point which gives the edge a direction. In some systems the resultant edge extends only from one point to the other while other systems consider the edge an infinite line passing through the two points.

5.4 Polygons

The polygons or surfaces which a system produces will, along with light points, be the primary media of the resultant image. They will also define the capacity of the system as described in 3.2. A polygon is a planar (i.e. two dimensional) feature which is defined within the system by referencing three or more vertices. With few exceptions, most systems require that the resultant polygon be a convex shape. This is so that the polygon will not start and stop more than once along any raster line. Since a non planar or twisted polygon can appear concave from some perspectives they must also be planar to a high degree of precision. The number of vertices which may be used to define the polygon is significant. Some systems limit the number to three and therefore consider that all polygons are triangles (and ensure planarity). Other systems will require four vertices with one of them duplicated if a triangle is desired. Some systems allow a larger number of vertices as long as the result is planar and convex. These systems may define their capacity in terms of the number of vertices rather than polygons. Also, some of their shading and texture features may be adversely effected or unavailable for polygons with a large number of vertices.

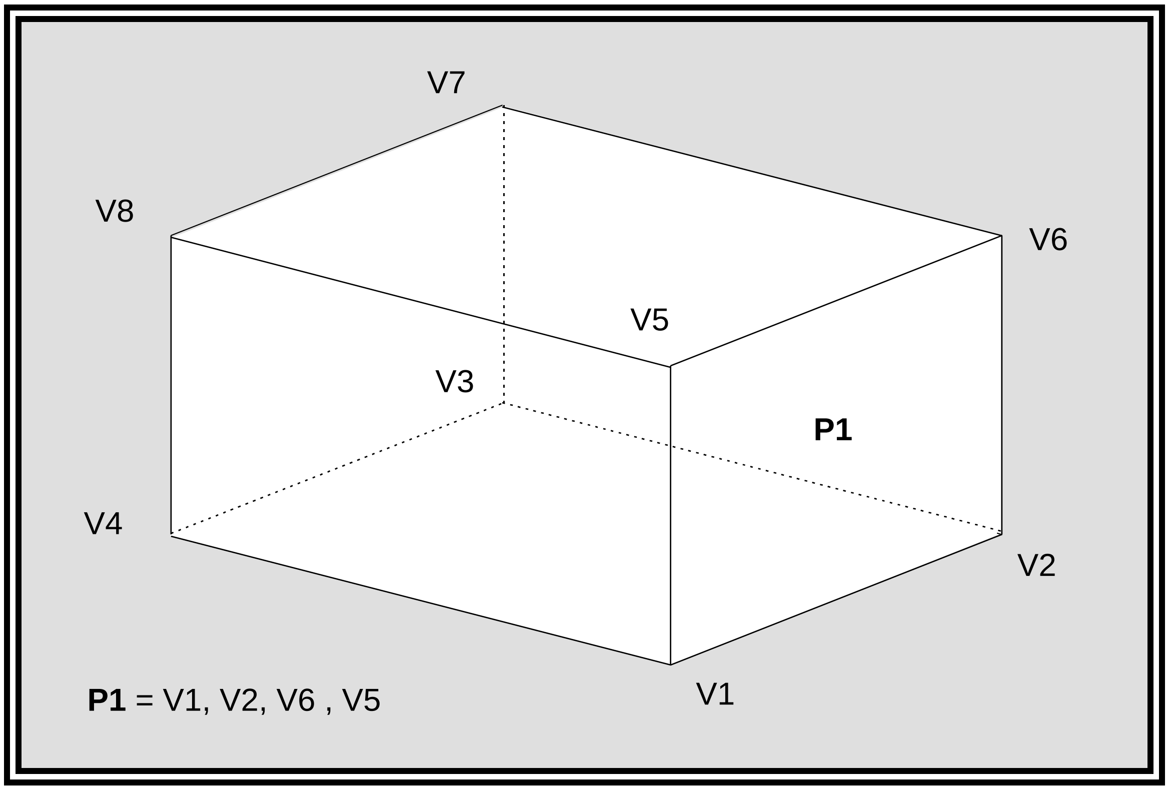


fig. 5-1 - Polygons defined by vertices.

The database entry for a polygon may be in one of several forms. In some systems the vertices are stored separately and each is given a name (or number). The polygon then references the names of the vertices which will define its shape (fig. 5-1). Another approach is for the polygon definition to include the X, Y and Z values for its vertices. While this may appear to be the more straightforward method, it will cause

repetitive data entries when a vertex is used more than once. The order in which the vertices are listed in the polygon definition will determine which side of the polygon will be displayed. A vector perpendicular to the polygon may be calculated by the modeling tools and stored as part of the polygon, or it may be calculated as part of the processing in real time. This vector will be compared in the geometric processor with the viewing vector to determine if the polygon is backfaced (i.e. facing away from the viewer). The same vector may also be used in conjunction with various shading techniques (see 5.3.2).

In systems which use edges rather than vertices a similar approach is taken. Each polygon definition will list the edges which form its boundaries (fig. 5-2). The order of the edges may be used to determine which side of the polygon to draw, as above, or the direction of the edge may be employed to indicate which side of the edge should be filled in. In the latter case, a properly defined polygon will be one in which the inner side of each edge will be indicated, resulting in the interior of the edges forming the polygon. If the wrong side of an edge is indicated the result will be an unbounded polygon which extends infinitely away from the incorrect edge. Another difficulty with edges is that they may not meet in all three dimensions (fig. 5-3). This may be due to data entry errors or to small round off errors in the GP's calculation of the perspective image. The result will be that the polygon will "leak" when viewed from certain angles. Vertices do not share this problem because common positions are used to create the resultant edges for the polygon.

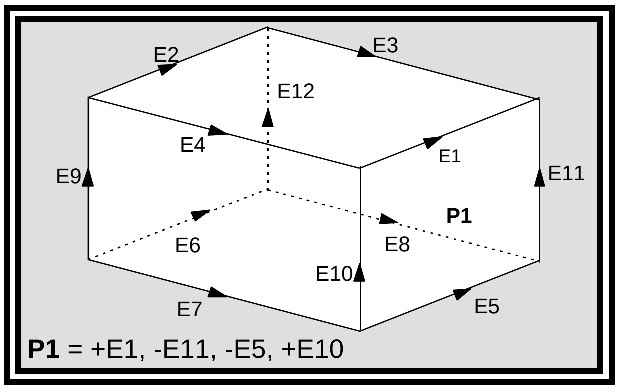


fig. 5-2 - Polygons defined by edges

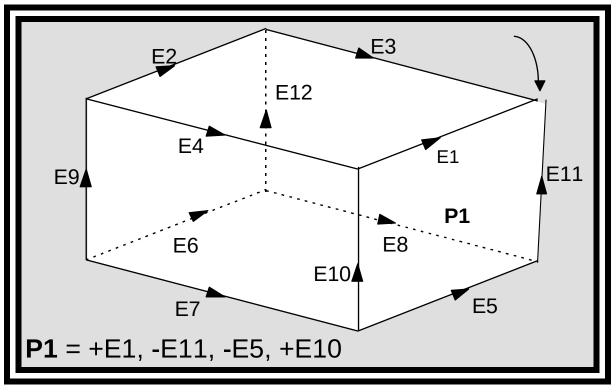


fig. 5-3 - Polygons with disconnected edges

In systems which use bounded edges (defined as a line bounded by two points) these edges can only be shared by polygons which have a common border, as the edges of a cube. Infinite edges may be used anywhere along their length. The stripes on a runway, for example, will make use of the same edge many times because they tend to be aligned with each other (fig. 5-4). This method is efficient for flat areas where all of the edges are in the same plane and may be used several times each. Non-flat areas or

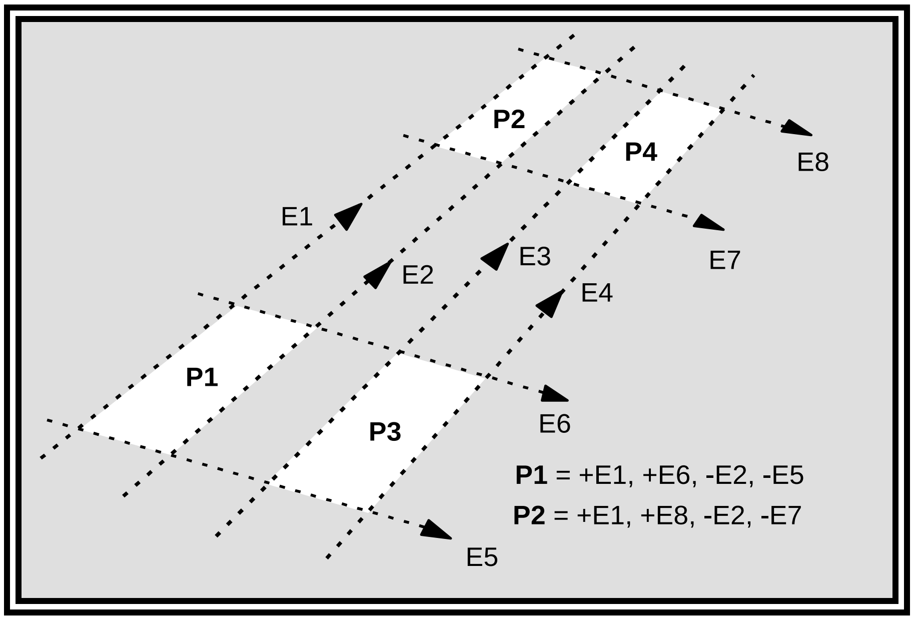


fig. 5-4 - Polygons defined by unbounded edges

databases with many three dimensional features (buildings or hills) will tend to expend all of the available edges with few opportunities to share them, resulting in fewer total polygons.

All of these techniques result in flat polygons with straight edges. A few systems provide curved edges, typically circular segments. Although these are entered in the database in a circular form they are treated by the hardware as a large number of small edges and tend to have a severe impact on the system's capacity. Also, the applications for circles in the database are not as numerous as one might imagine. There have been several approaches discussed which would make use of various three dimensional shapes or volumes as the modeling primitives. These would be scaled, clipped and combined to provide complex objects with much less impact on the image generator than the large numbers of polygons which would be required to approximate the same shapes. No real-time systems which make use of these techniques are yet available.

5.4.1 Color and Intensity

There are several other characteristics of a polygon which determine how it is utilized in the database. The intensity (in a monochrome system) or color of the polygon must also be defined. Two approaches to color are common. One is to define a color map or table which contains all of the colors available to the modeler. The polygon definition will then contain a reference to the table, either a number or a name. In some cases this color table may be accessed and modified as part of the modeling process and in others the colors are in memories in the hardware (normally in the DP) where they are much more difficult to change. The other approach is for each polygon definition to contain the actual values which define the color.

The most common color implementation uses separate values for red, green and blue. There are other measures possible, namely the Munsell color system which refers to hue, chroma and density. Color is a very difficult aspect of the visual database to understand and manipulate and most people find RGB the

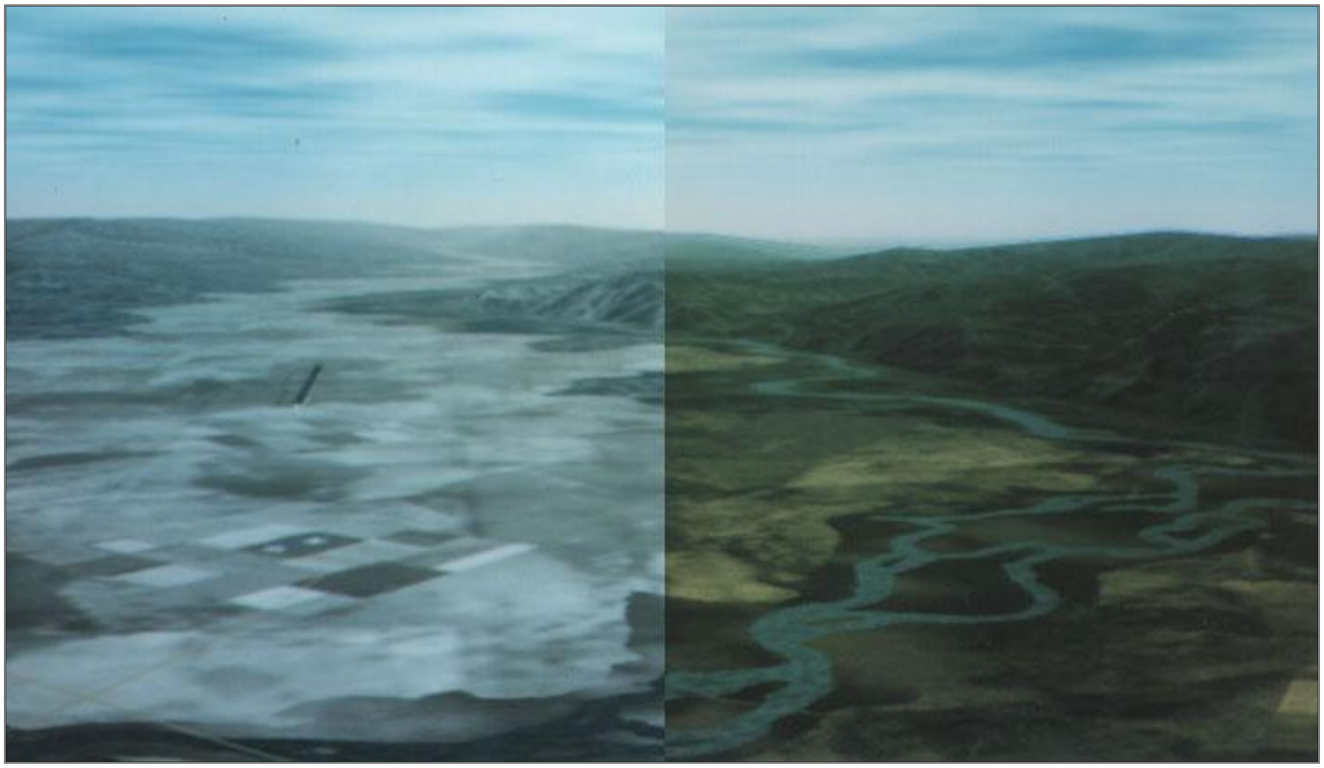

E&S/Hughes

fig. 5-5 - Seasonal effects created by manipulating the color pallet (and texture maps)

easier form to work with. It is also the form that most display systems utilize, probably for the same reason.

In some smaller systems the number of colors available and/or the resolution of the colors may be limited. In addition to restricting the modelers ability to approximate real world colors this can also cause problems with subtle shading effects and texture. The maximum typically provided in any system is 256 shades each of red, green and blue. This results in 16,777,216 possible colors. While many of these are not appropriate for normal database use, it is obvious that there should be no shortage of colors that are. The systems discussed above which utilize a color table will typically allow 256 or 512 of the 256^3 colors to be included. For most applications this is more than sufficient.

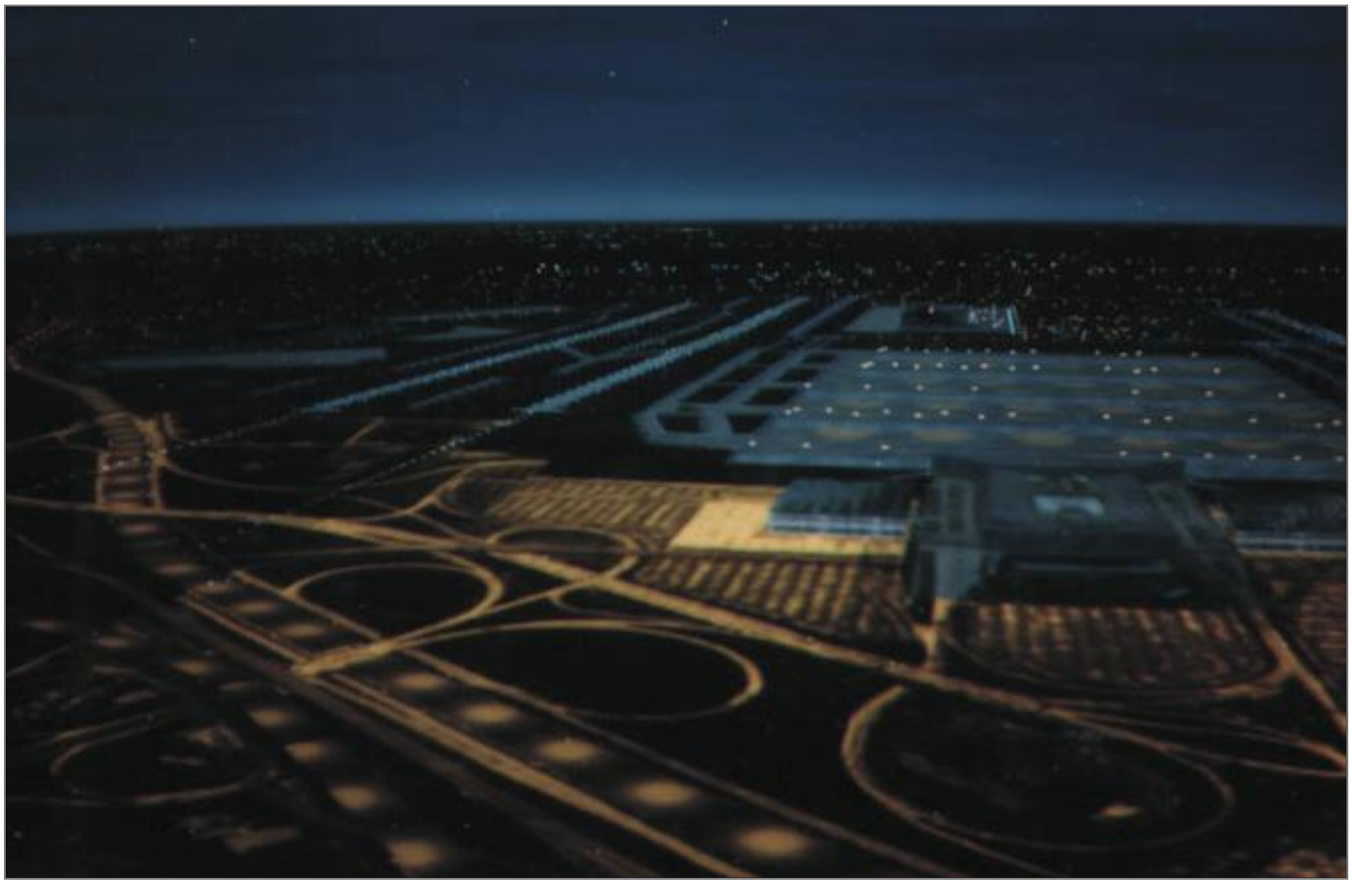

E&S/Hughes

fig. 5-5a - Night scene combining transparency texture with luminous polygons

Another concern which the modeler must address is the effect of the display device on the colors that have been selected. This is particularly true when projectors and monitors are used interchangeably. Many systems provide a gamma correction feature which will alter the color signals which come from the DP in order to standardize what the observer sees. In any case, the modeler will often find that small color changes are required to achieve the desired effects. These changes are much easier if the color map locations can be slightly modified. If the color definition is contained in the polygons then each polygon must be edited to implement a change.

Dusk/night systems which provide monochrome images require that the polygon definition include an intensity value. This will effect the basic gray shade with which the polygon is displayed. The normal number of shades available is 64 with 1 being black and 64 being as close to full white as the beam penetration tube phosphors will allow. The human eye has difficulty differentiating more than 64 gray shades, particularly when the maximum intensity of the image is limited. Less than 32 total shades, however, may begin to effect the choices the modeler has in creating the desired variety in the database.

Some color systems also include an intensity factor in the polygon definition as a further modifier to the color. This greatly extends the effects which can be achieved without adding to the color memory. The modeler is allowed to create subtle differences between adjoining polygons. This is particularly effective when modeling shadows.

Another system feature available with some image generators which should be discussed here is luminous polygons. These are polygons which are displayed at their full (i.e. day mode) intensity when the system is in dusk or night mode. The result is a polygon or object in the image which appears to be illuminated. Windows and doorways on buildings are a good example (fig. 5-5a). The effect of a bright light shining on an object can also be achieved. This is a very powerful feature which adds a great deal of realism to dusk and night scenes. Some method of adjusting the brightness of the illuminated polygons may be required depending on the application. Also, in systems that cycle at 30 hz. in the dusk and night modes the luminous polygons can cause flicker if they cover too large a portion of the display.

5.4.2 Shading

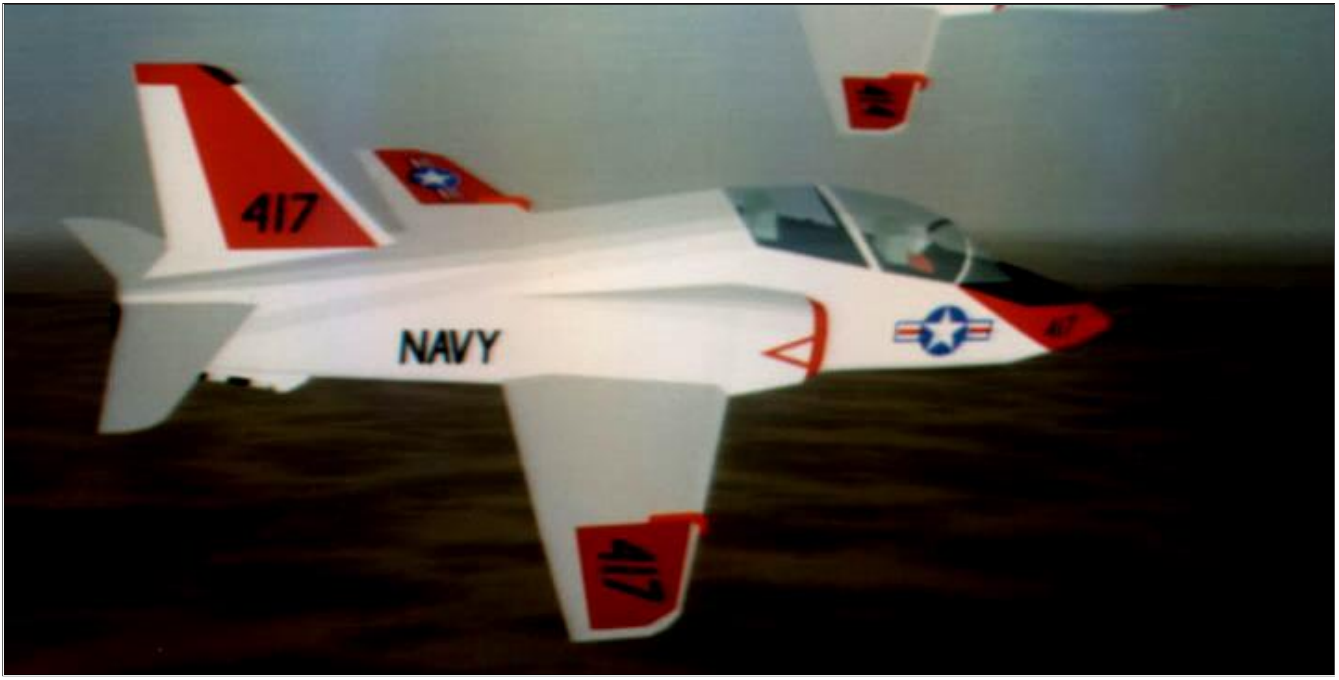


fig. 5-5b - Aircraft model with sun (flat) shading evident on upper fuselage

Another important factor in how the polygons are perceived is the manner in which they are shaded, or illuminated. When you consider that a large portion of what we see in the real world is the result of light reflecting off a surface, you can see why illumination techniques are important. Unfortunately, they are also expensive in processing terms as well as monetary. Lighting models are one of the main differences between the non real-time CIG applications which create images for motion pictures and other near real world scenes. In these applications a great deal of processing time can be spend evaluating all of the light sources which are illuminating a single pixel. This kind of attention to detail produces reflections, glint, and many other effects which we take for granted in our everyday perceptions of the world. A real-time system, however, with only a 60th of a second to calculate the intensity of over a million pixels must be much more careful, and clever, with the illumination effects it provides.

Very low end CIG devices may not provide any shading options other than to display the polygons at the intensity and color as defined in the database. Most systems, however, will have one or more variations available to the modeler. These are normally specified in the polygon definition.

Sun shading assumes an illumination source which will modify the intensity of the polygons based on their angle relative to the source. The illumination vector (from the source) is compared to a surface normal vector for each polygon. The degree of intensity reduction is determined by the difference between the two vectors. Since polygons facing directly away from the illumination source should not be depicted as black (probably), the algorithm normally includes minimum, or ambient values.

This technique is most effective when applied to moving objects in the scene, particularly aircraft. The polygons which make up the airplane would all be assigned the same color (assuming that is appropriate) and the sun shading effect would cause those on the "sunward" side, typically the top, to appear brighter (fig. 5-5b). When the aircraft rolls the intensity of the polygons would be increased as they approach the upper side of the aircraft and decreased as they approach the lower side. This is a very important factor when formation flying is a training requirement.

Sun shading may also be applied to objects on the ground which are not moving. One technique is to define all of the polygons which make up a building or hill as having the same color and intensity and then let the sun shading feature provide the different intensities required to make the facets of the object evident. While this approach is useful for quickly coloring a large number of polygons, it also limits the control the modeler has over the final appearance of the database. He may find that the shading effect did not create enough difference in intensity for the desired effect and that the resultant boundaries between some polygons are not discernible. These difficulties may sometimes be worked around by changing the sun angle but this may move the problem to another portion of the database.

Smooth shading is an extension of sun shading which allows the intensity of two or more polygons (of the same color and texture) to be blended together so that the edges between them cannot be discerned. This technique is generally referred to as Gouraud shading after Henri Gouraud who developed the algorithm in 1971. A vector which is perpendicular to the combined surface is calculated at each intersection of polygons. An interpolation of this vector with those at surrounding intersections is then used to calculate the appropriate intensity for each pixel as the polygons are processed. This technique is used to create objects which appear rounded even though they are a collection of flat polygons. It provides excellent representations of aircraft (fig. 5-6) and other complex objects which are generally round in shape. Since this feature effects several polygons simultaneously those polygons must be defined in such a way that they may be easily related to each other.

While smooth shading adds a great deal of realism by removing the edges from the image, it may also detract from the usefulness of the database in some ways. Rolling hills, for example, which would seem to be a perfect application of this effect, may not provide enough visual flow if the edges are not visible. When the pilot does not have the passing edges to provide speed and height cues he has difficulty sensing his flight conditions. In systems where texture can be added to the hills to provide the flow, smooth shading can be very effective.

A long standing non-real time CIG technique that is finally making its way into real-time CIG systems is Phong shading. Phong shading (also named after its creator, Bui Tuong Phong, in 1975) is an extension of Gouraud shading which provides specular highlights or glint. This is accomplished by evaluating the surface normal with the illumination source(s) for each pixel rather than a value interpolated from the vertices (or corners) of the polygons. This effect, combined with a definition of shininess, allows the

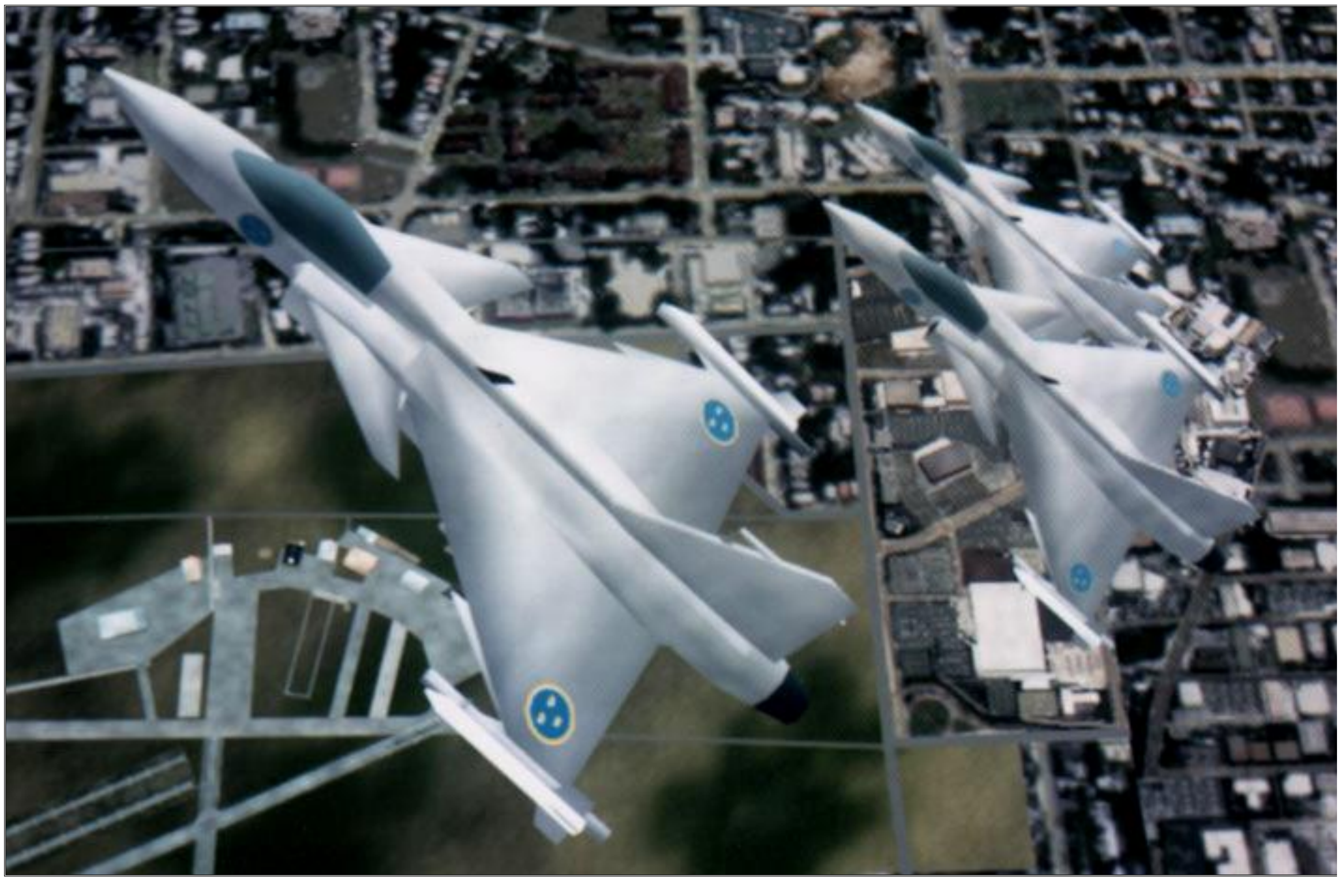


fig. 5-6 - Aircraft model with smooth (Gouraud) shading

surfaces in a database to react very differently (and realistically) to the illumination sources in the environment. Two additional benefits of proper Phong shading are bump texture maps (see 4.4.4), and much better and accurate landing lights and spot lights.

Ray tracing is a simple, but computation intensive, process which looks at the source of the light ray illuminating each pixel. Each ray is traced back to its origin (if there is one) through reflections, transparent surfaces, etc. This method is particularly effective when more than one light source is considered, including the reflection of one light source off a reflective object as a second source. Both of these techniques require that the polygons include some definition of surface reflectivity so that degrees of matte to glossy objects may be depicted. Ray tracing is not currently available in a real-time system, although it is only a matter of time.

Fixed shading allows the modeler to alter the intensity within a single polygon. The polygon definition will include a method for indicating how much intensity is required at each vertex. One corner can be made dark and the other three light, for example. The polygon would then be shaded evenly between the dark corner and the bright ones. Polygons with this feature typically cannot be sun shaded as well as this causes a conflict in the intensity solution across the polygon. This feature is useful when creating smooth shading type effects on flat surfaces. Smudges on a road or runway surface, for example, where dark and light polygons would create an inappropriate hard edge. Other applications can create the illusion of a three dimensional surface with two dimensional polygons, such as water or a plowed field, in the same way that texture does.

Color blending is an effect similar to fixed shading. It is applied to the polygon in the same manner except the values at each vertex will indicate the proportions of the two colors being blended. The utilizations are also similar except that there are many more applications where color blending will add to the realism achieved.

5.4.3 Texture

Texture, and some of its implementations, was discussed in section 4.4. The addition of this very powerful feature to CIG systems has made the modelers job much easier in many ways. Certain database effects, particularly natural ones such as clouds and water, are very difficult if not impossible to represent with polygons and edges. As with most powerful tools there are also some pitfalls which must be avoided.

The texture maps must be applied and moved correctly. The detail inherent in the texture adds a tremendous sense of visual flow to the image. If the maps are not quite in the same plane as the polygon or move in some unexpected way (due to database management, for example) the negative impact on the observer may be as great as the positive impact intended when the texture is correct.

Another potentially distracting aspect of texture is that the maps may repeat themselves across the face of the polygon. Small maps applied to large polygons may be repeated many hundreds of times before they are removed from the image by the anti-aliasing process. This can be particularly noticeable if the maps do not tessellate with themselves and the obvious boundary between one iteration of the map and the next contributes to the repetitiveness of the pattern. Carefully designed maps will help but there is a limit to how many times even a well tessellated map may be repeated without becoming distracting. Combining two or more maps which are of similar design but different sizes can greatly extend the distance between repeats. The scales of the two maps should be selected such that the larger one is not a multiple of the smaller one. The distance between repeats is then the product of the two scales. If this technique can be extended to three or four maps the problem virtually goes away.

Most of the early applications of modulation texture were to represent the natural effects which had not been possible before. Water surfaces, clouds, the grass around runways, tree canopy effects and the concrete surface of runways and taxiways were some of the first utilizations. These all tended to add to the detail and visual flow of the image but nothing recognizable in terms of size was added. Man made or man induced features were included next. Expansion cracks could easily be added to a runway surface map for example. Tire marks, an important feature of any runway, could be implemented with the appropriate map as can windows and bricks on buildings.

The difficulty with these features is that the maps which depict them can be used for little else. A window map, for application to the walls of buildings, is pretty much restricted to that utilization. The natural maps tend to be more generic patterns which might depict water at one scale on a blue polygon and grass at a smaller scale on a green polygon. We again have a performance tradeoff to consider. The application of the texture depends on how many maps are active in the system and/or how many may be managed into the database in real time. Fortunately, most current systems provide sufficient texture memory for a wide variety of maps.

The trend toward representing specific features with texture has resulted in the addition of silhouette texture to the larger systems (fig. 5-9). The requirement for this feature grew out of a need to depict very complex objects without utilizing a lot of polygons and without aliasing. The classic example being trees with very irregular edges and many holes in them. The first attempts to provide tactical databases for helicopter simulators uncovered some basic limitations in systems without texture even though they could produce unprecedented numbers of polygons (at the time). Many of the real world height and speed cues that the helicopter pilot uses to sense small movements during landing and hover maneuvers come from subtle parallax effects provided by seeing one tree through another, for example. Creating these effects with polygons meant very high content per tree and therefore, not many trees. Adding modulation texture to the polygons of the tree does nothing to increase the parallax between trees. Transparency is required.

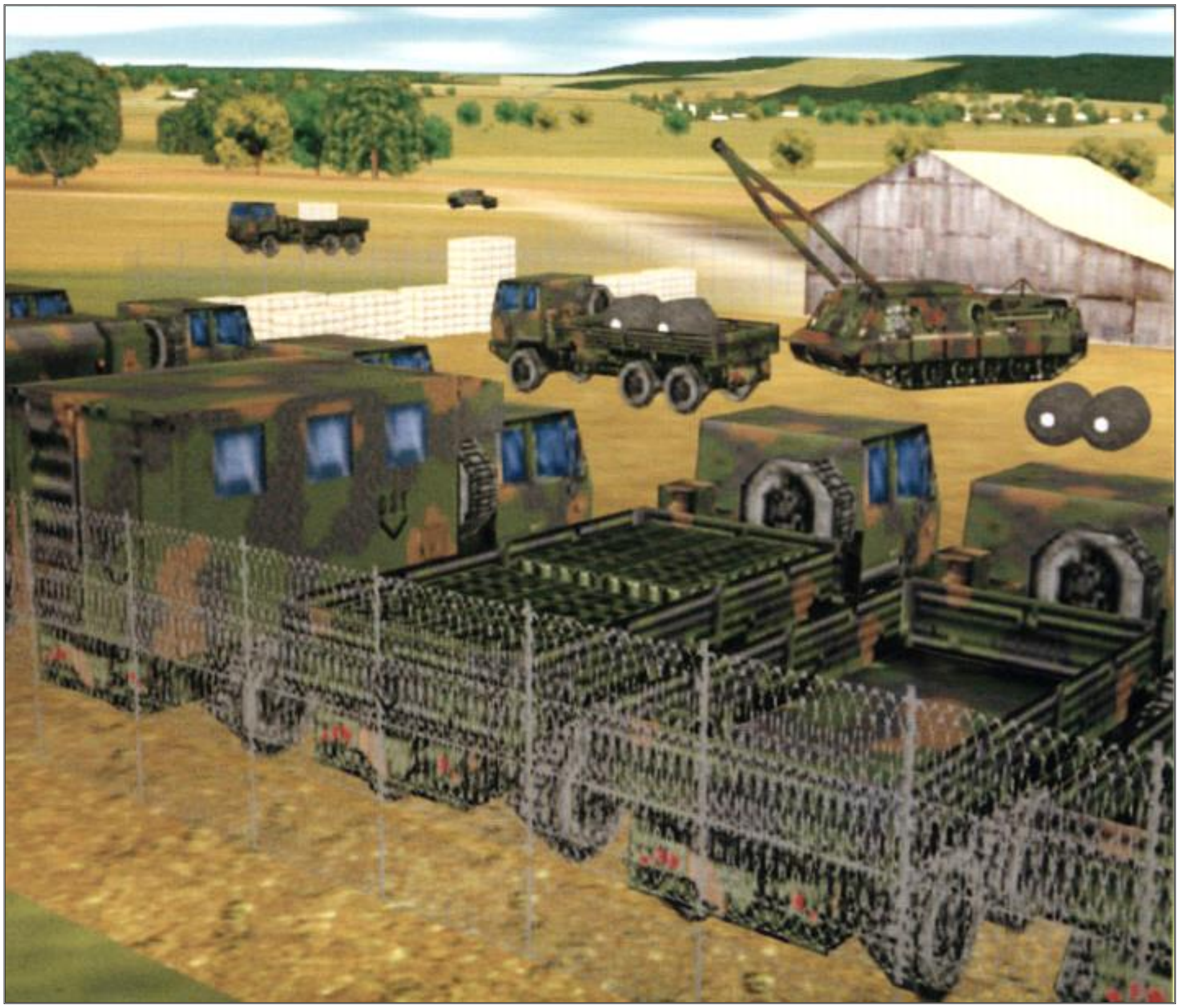
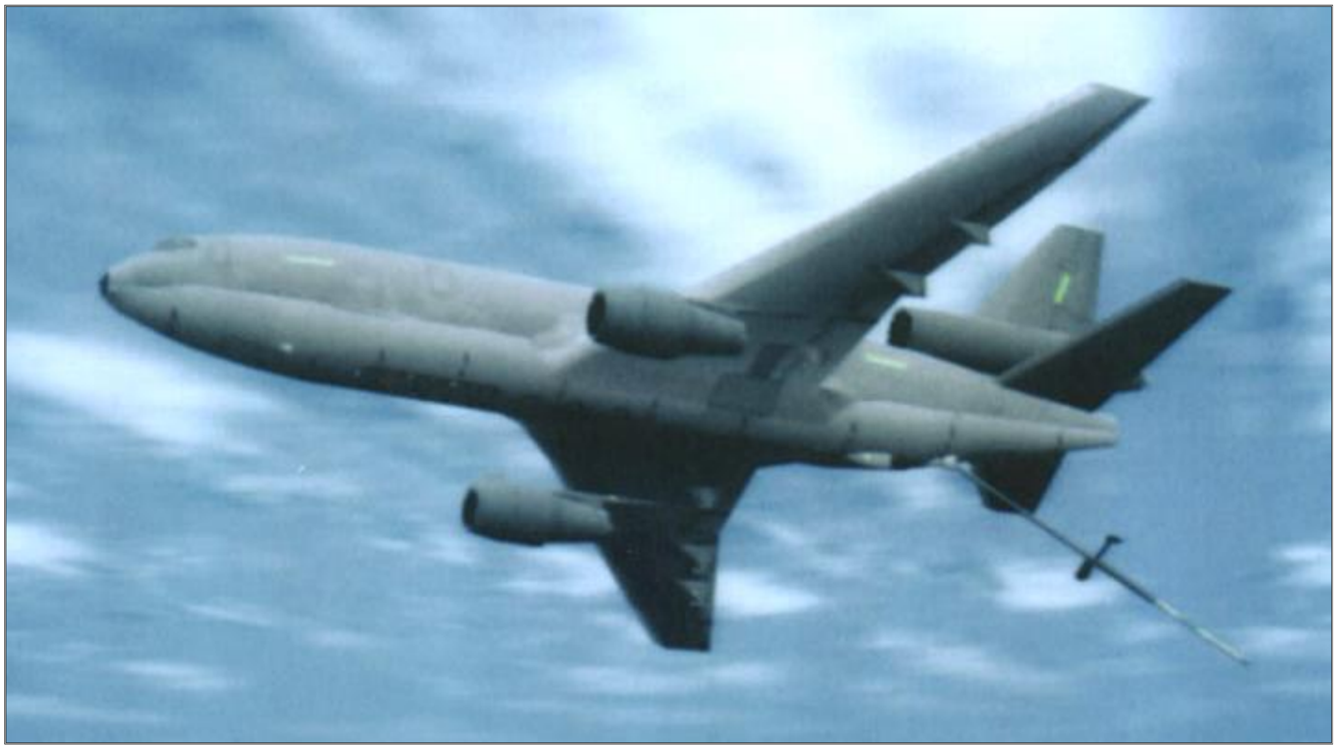

E&S

fig. 5-9 - Database with silhouette texture and photo-texture maps

Image generators which can modulate transparency provide much better trees, and similar objects, but aliasing may limit these applications unless very high resolution maps are used. An alternative is silhouette texture which provides its own anti-aliasing effects. The polygons to which this type of texture is applied are normally sized to match the map as closely as possible. This is to reduce the amount of the polygon which extends beyond the opaque portion of the map and must be made transparent. Here again, the maps are very limited in their flexibility and it is even more critical that there be a large number available in the system. This is a very powerful system feature and adds new applications which may not be possible with modulation texture.

Another texture development is "photo texture" (fig. 5-9). This involves taking a photograph of a real world object (a tree again) and digitizing it to create a texture map. The resultant map may be used as a silhouette map (with appropriate processing) as well as the modulation map to form the interior of the polygon. These maps may be used on large polygons, but they must be modified to ensure tessellation. While this approach to creating maps provides some very complex and compelling imagery, it does have some limitations. A photograph of a very irregular surface or object (like a tree) when applied to a two dimensional surface (like a polygon) does not appear to move correctly. This effect is at its worst when the polygon is nearly edge-on to the observer. This condition may be aggravated by shading effects in the system which may conflict with the "photo" of a rough surface.



Lockheed Martin

fig. 5-10 - Aircraft model with photo texture utilized for surface detail

Also, since photo-texture is more applicable to some types of objects than others, a typical image will be made up of some objects which appear almost "real world" and others which are closer to the normal CIG approximations. Some observers find this inconsistency of realism distracting. Another application of photo texture maps is to project one or more maps onto a limited number of polygons which form the basic shape of an object. In this way complex objects such as aircraft or tanks can be modeled very economically (fig. 5-10, 10a). The texture maps are generated with photo digitizer hardware utilizing a photograph, prepared art work or a scale model of the object. The resultant map data will then be cleaned up with automatic texture map tools and/or by hand. This will ensure that the maps meet at the boundaries and that the color and contrast are consistent. These maps, which include the markings, panel lines and camouflage, are then projected onto the polygons which have been designed to accept them.

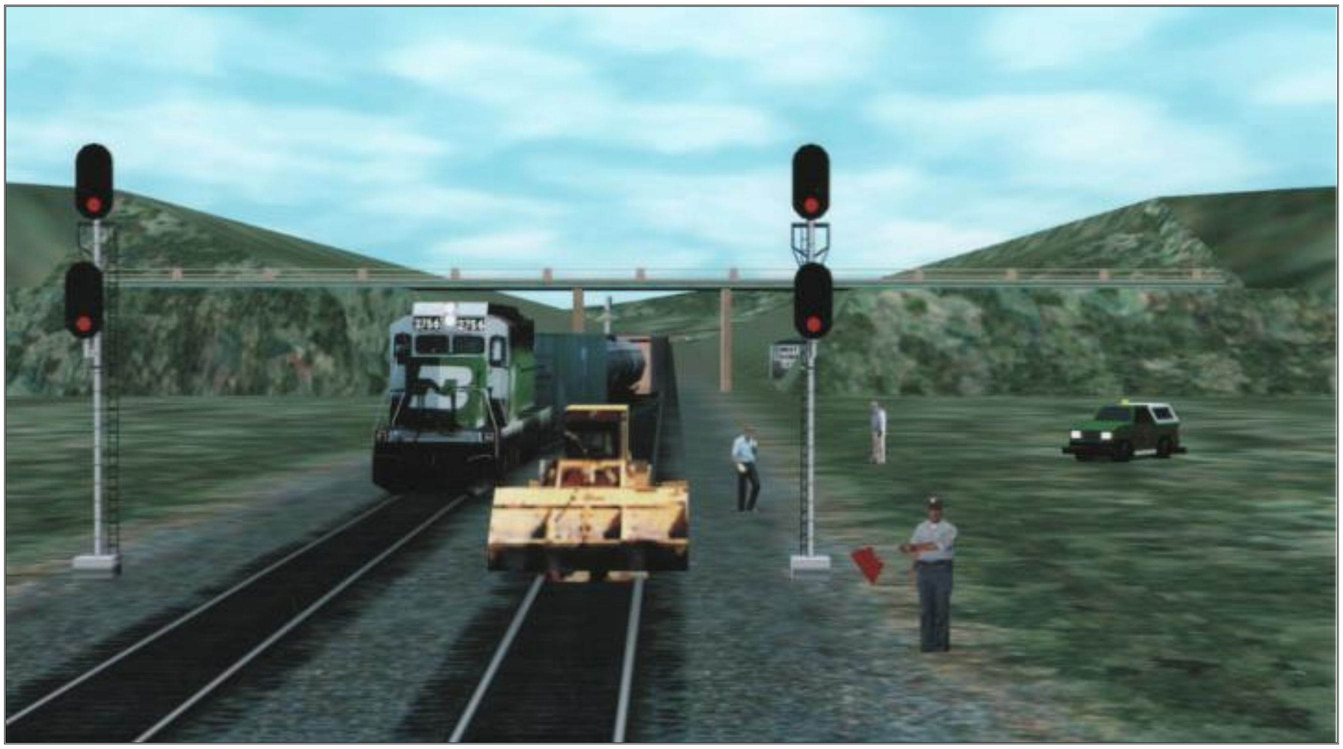

SGI/Hughes

fig. 5-10a - Vehicles and people created with photo texture

This same approach has been applied to terrain using aerial or satellite photography and NGA derived polygons. Geo-specific or global texture of this type creates very good representations of the large areas as seen from altitude (fig. 5-10b). In order to operate at lower altitudes, however, it is necessary to add three dimensional objects to supplement the speed and height cues. The size of the terrain areas involved and the texture maps to cover them requires some form of real-time texture management to bring new detail from the system disk. It is planned (desired, hoped) that databases of this type which can be readily updated with intelligence data in the form of new texture maps (and some objects) will satisfy mission rehearsal requirements.

In the next few years there will probably be more and more demand for systems with some degree of geo-specific texture. In the past the only satellite imagery available for commercial, unclassified applications was of relatively low resolution. The best is 10 meter monochrome imagery from Spot (a French company). The other source, NASA's Landsat provides color imagery (among other kinds), but the resolution is only 30 meters. These basically means that the smallest surface object or feature that is recognizable is 10 or 30 meters on a side. As mentioned earlier, geo-specific texture derived from this imagery looks good from altitude, but become very blurred and useless as scene content at low level. In the next year or two there will be commercial satellites in orbit which will provide imagery in the one to three meter range. This will make geo-specific texture much more useable as a general solution. The systems which make use of this high resolution imagery will also require much more texture memory and processing capability. A texture map derived from one meter data could require 100 times the memory than current maps derived from 10 meter Spot data.

So, while texture has made it possible and/or easier for the modeler to depict many types of features and

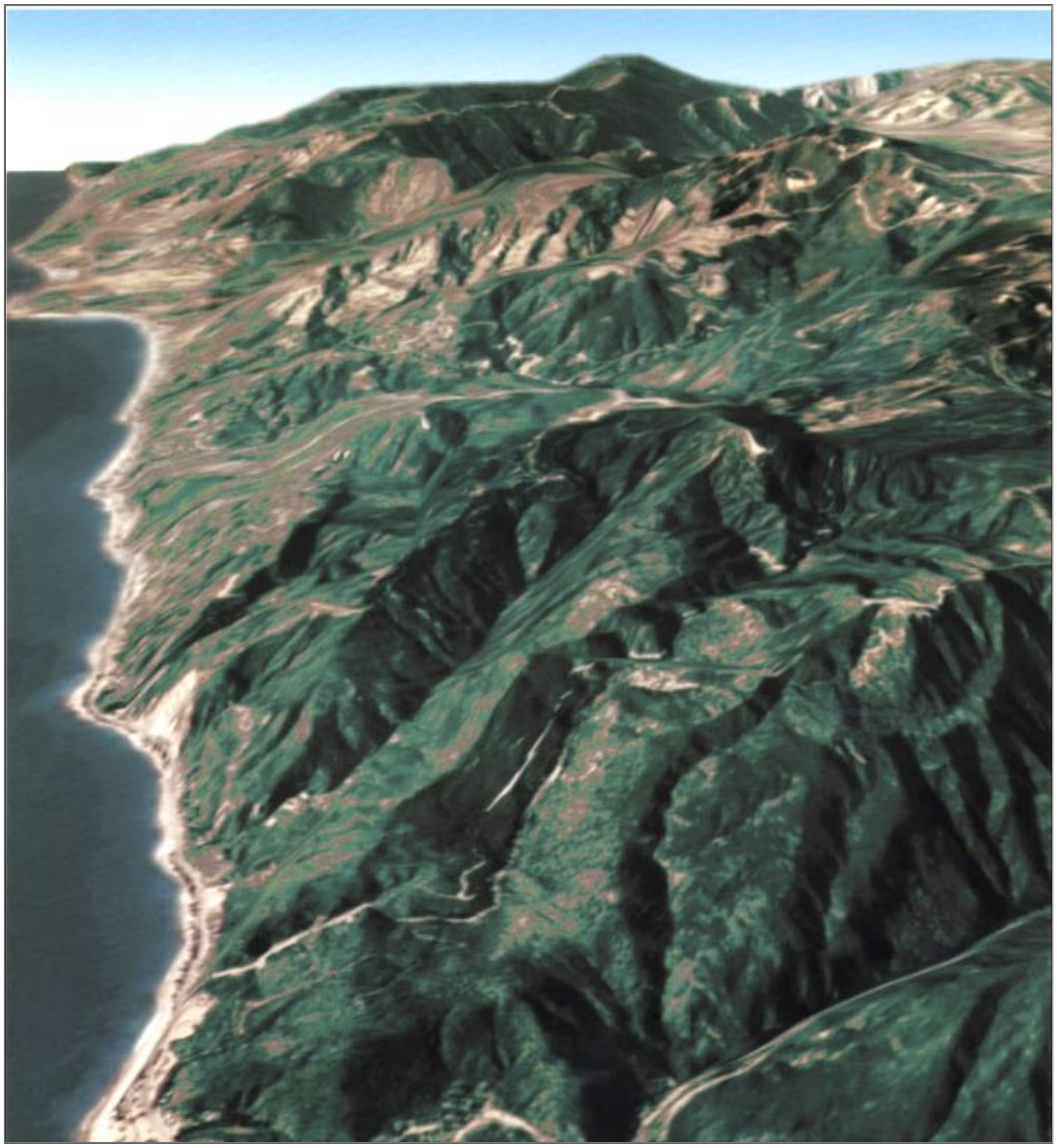


fig. 5-10b - Geo-specific texture derived from aerial/satellite imagery

objects, it has also provided him with a very powerful and complex tool which must be used carefully if its full benefits are to be realized. It is not a magic potion which can be applied to a database to greatly increase the detail and visual flow. It must be designed into the database along with all of the other system features so that it is used in the most efficient manner possible.

5.5 Light Points

Light points are a very important part of simulation imagery. This is, of course, most obvious in the dusk and night scenes where lights make up a large percentage of the image content, but is also true in day scenes, particularly in low visibility. There are many training scenarios which cannot be performed if some simulation of light points is not provided.

Lights are almost always entered in the database in a string format of some sort (fig. 5-11). Some of the smaller systems, which may not produce true light points, require that each point feature is entered individually as a vertex or even as a small polygon. This would be a very tedious task for the modeler and would require a great deal of storage in the system if the database is to contain a significant number of lights. A typical light string definition will include the first point, last point and the total number of lights

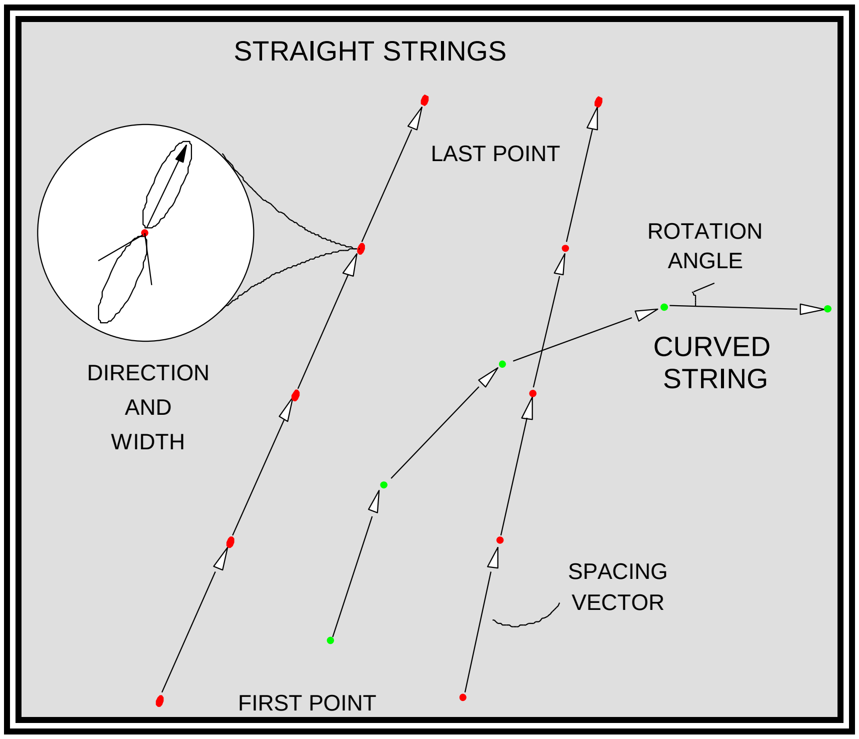


fig. 5-11 - Straight and curved light strings

in the string. It will also contain color, intensity, directionality and several other modifiers depending on the system. Another form of string may contain a vertex for the first light, a vector to the second and the number of lights. A curved light string may require a third point to define the curve or the number of degrees the string rotates between lights may be entered.

Some CIG systems also provide a random string. This construct, which sounds like a contradiction in terms, allows the modeler to describe a rectangular area in the database and indicate how many lights should be included. The system will then randomly scatter the lights within that area. A random seed number is also included so that the modeler may alter the arrangement if it is not suitable for some reason. All of these different string formats are expanded to form the individual lights in the CIG hardware, typically in the geometric processor.

The major factor in how lights will appear in the image and how they may be used in the database is the display technique. Calligraphic lights have very different capabilities and limitations than raster lights and certain effects which are possible with one may not be available from the other. Some systems are capable of both types of light points if an appropriate display device is utilized.

5.5.1 Light Point Color and Intensity

In systems with raster lights the color of the lights may be entered into the string data in the same manner, and with the same selection, that is used for the polygons. Calligraphic lights, however, must be handled differently. The display devices which are compatible with calligraphic lights, particularly beam penetration devices, do not lend themselves to rapid or frequent changes of color. Also, the resultant lights, due to their brightness, would not benefit from subtle color differences. Calligraphic systems therefore tend to offer a limited number of color choices; as few as five in the dusk/night systems (no blue phosphor) up to 16 in some of the full color systems.

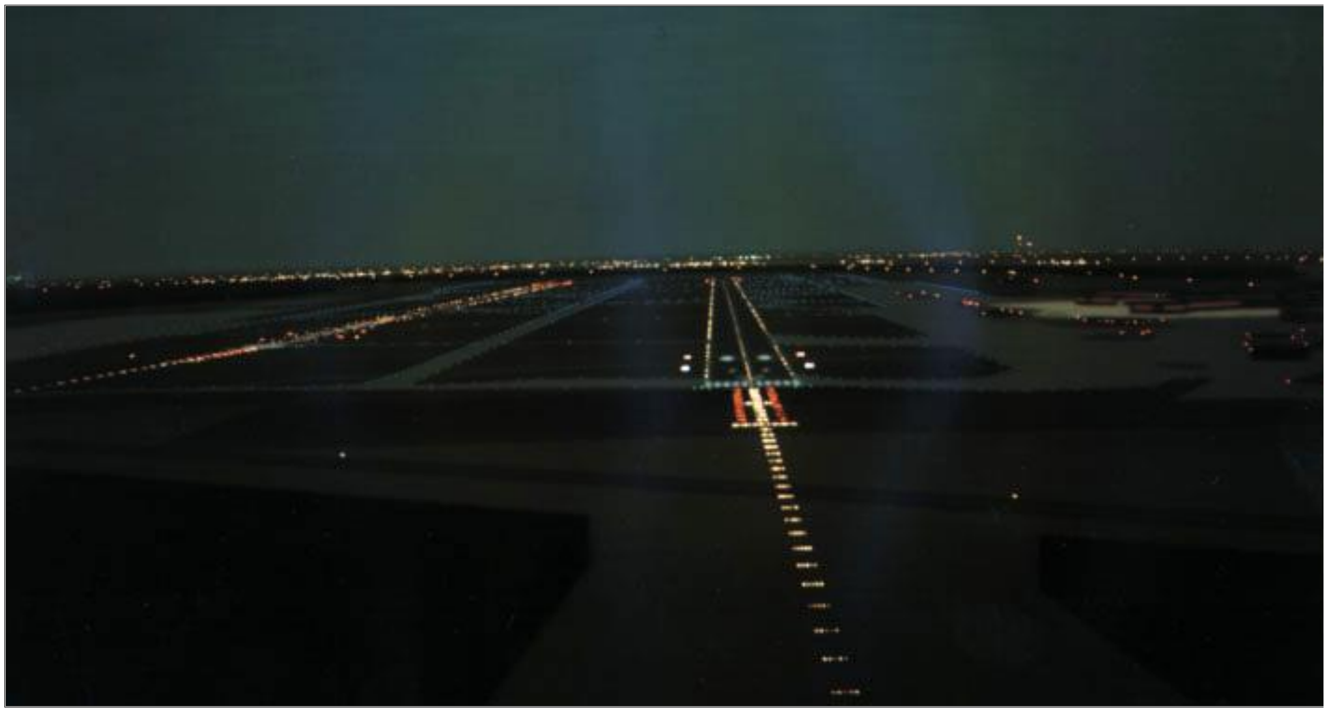


fig. 5-11a - Calligraphic light points modeled with random intensity

The string data will also include an intensity factor. This will be the basic brightness of the light before it is affected by distance, fog or any of the other special features described below. The range of intensity selectable in the string may be larger than that available in the display. This means that many lights seen in clear visibility and/or up close may appear at the same (maximum) brightness. This is done to allow a wider range of response to fog and relative distance between lights. Typically all lights in a string will have the same basic intensity. Some systems will provide a random intensity function (fig 5-11a) which varies the brightness of the lights in a string which adds to their effectiveness and realism in some applications.

The light string will also normally include some form of switch number which provides a method for the instructor to turn the string, or strings, on and off or to adjust their intensity. An airport model will be constructed such that all of one runway's approach lights, for example, are assigned to the same switch number. The instructor may then adjust their intensity to match the weather and illumination conditions. There may be as many as 64 individual switches available for this purpose. The values of these switches must be sent from the host.

Although calligraphic lights all tend to be displayed at the same size, those in the distance may appear smaller. This is primarily because they are dimmer. Some systems will provide perspective growth for lights which are very close. The effectiveness of this feature will greatly depend on the dynamic focus capability of the display device. Some systems may display raster lights at different sizes in order to create special effects and/or limited perspective growth. As the lights are drawn larger it is important that they be circular in shape.

Two related considerations of brightness should be mentioned. The first is the same point test. This is a selectable feature which is performed in the GP. Its purpose is to avoid a build up of brightness when two or more lights fall at or near the same location on the display. The second is a separate intensity function which may be applied to light strings which tend to be viewed primarily from one end (i.e. runway edge lights). Because the size of the lights is constant, strings of closely spaced lights may not exhibit the proper perspective characteristics from some viewing positions. To compensate for this effect, these lights are drawn slightly dimmer than is required by their distance from the eyepoint. This provides a more acceptable perspective appearance and avoids an unnatural build up of light points.

5.5.2 Directionality

Some form of directionality is required if complex arrangements of lights such as those around airports are to be represented. Directionality requires the system, again in the GP, to modify the intensity of each light point based on its position relative to the eyepoint. Some systems provide only horizontal directionality and treat any vertical requirements as a special case while other, normally larger, systems will allow both vertical and horizontal controls on all lights. Depending on how these are implemented, they may have an impact on the total number of lights available.

Directionality of the light string will be specified in the string data. The heading (relative to the model axes) will also be specified along with some indication of the type or shape of the simulated light beam. In some systems the shape of the beam will be selected from a predefined set of definitions stored in the hardware. In others the actual width and shape of the beam will be entered as part of the string data. A residual value may also be provided which would cause the light to be visible from all angles but become brighter when the beam is entered. The available lobe definitions may also include a bi-directional indicator which would cause the beam to extend in both directions.

5.5.3 Flashing and Rotating

The complex environments typically represented by CIG devices often contain special purpose lights which must be simulated if useful training is to be accomplished. These are sometimes coordinated by the visual computer and implemented in the GP. In other systems the entire process is passed into the hardware.

In order for a light to flash, the string data must include some method of indicating how fast and when. An on time and an off time are typical controls. These may be pre-defined and the string data indicates which one is required. The string may include the actual period, normally expressed in system cycles or fields, that the light is to be on or off. Note that in either case the light cannot flash faster than the system cycles. There may also be a phase control which will allow one light to flash relative to another. This is necessary in order to simulate a light which alternates between two different colors, for example.

Rotating lights or beacons in the environment differ slightly from flashing lights in that the intensity of the beam changes gradually as it swings toward or away from the viewer rather than simply flashing on and off. One method of simulating this effect is to change the heading of a directional light. A definition of the rate at which the beam heading is to rotate is required. The phase between two or more lights may be determined by adjusting their original headings and rotating them at equal rates. Alternately, controls may be included in the flashing light mechanism to control the manner in which the light is turned on and off.

5.5.4 Strobes

Strobes, or sequenced flashing lights, are capacitor discharge type lights which are used in several standard runway approach light configurations as well as numerous other applications in aviation related environments. A strobe light string is entered in much the same way as normal light strings but rather than draw all of the lights every field the image generator will draw only one. During the next field it will draw the next light in the string and so on until the end at which point it starts over again. The effect is of a bright light "running" down the string repeatedly.

In calligraphic systems the high brightness of these lights is achieved by allowing the electron beam in the display to hit the same spot several times. This effect, coupled with a high modeled brightness so that the light penetrates the selected fog setting, provides a very realistic effect in low visibility. In a raster system the effect is similar but the additional brightness of the multi-drawn light points is not available. In some systems the strobe strings can be programmed to wait more than one field before drawing the next light and to wait before starting the string over again. This allows other applications of strobe lights as well as systems of lights where several strobes may be connected together (as in the famous Canarsie approach to JFK). A special application of strobe effects can provide a limited simulation of traffic moving along highways in dusk and night scenes.

5.5.5 VASIs, et al

Visual Approach Slope Indicators (VASIs), and other similar devices, are an important part of an airport scene. These lighting systems tell the pilot when he is high, low or on glide slope as he approaches the runway (lights on each side of runway in fig. 5-11a). There are several different types in use around the world as well as numerous similar devices in use by the military. The implication of these devices to a CIG system is that the color and/or intensity of several light strings must be manipulated depending on the eyepoint's position relative to them. If the system has vertical directionality the intensity portion of the problem is taken care of but color changes must be handled separately. If vertical directionality is not available then the entire feature must be treated as a special case.

This is normally done in the visual computer and once the color and intensity of the subject lights has been determined the results sent on to the GP as normal light strings. For a standard VASI the problem is to determine whether the lights should be red or white or something in between. The light string definitions will contain the glide slope angle against which the eyepoint is to be tested as well as a transition zone in which the lights will be a blend of the two colors. Raster systems may have a predetermined set of colors, which are a blend of the two colors required (red and white normally), and will apply the appropriate one to the light string. Calligraphic systems may draw a red light and a white light at the same point on the screen and adjust the intensity of each to achieve the required blend. In either case, this is an essential but costly feature to simulate. The number of these active in the database should be kept to a minimum.

5.6 Dynamic Coordinate Systems

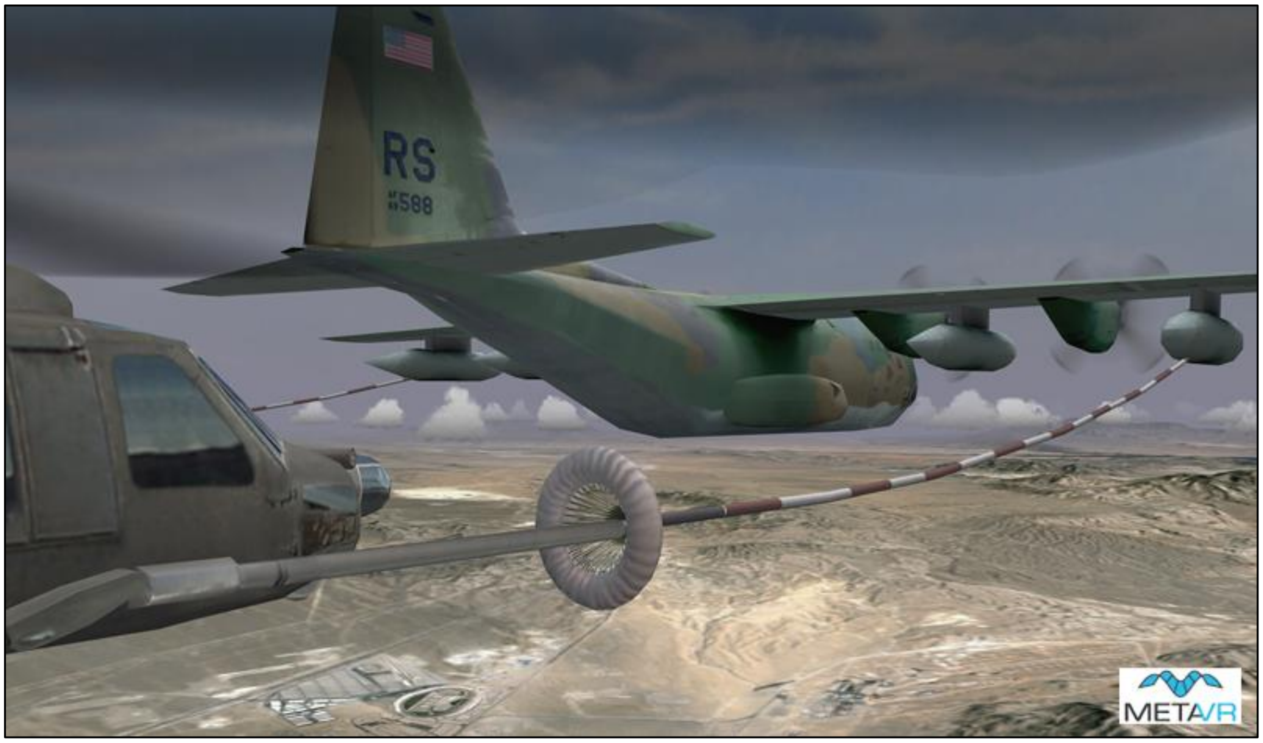
Dynamic Coordinate Systems serve many purposes in CIG databases. In some applications they are

E&S

necessary in order to satisfy the training requirements, in others they are useful and enhance the maneuvers which may be accomplished, and in some cases they only provide variety and realism to the imagery. In order to implement DCSs, the database and system must have some way for the modeler to direct how various groups of polygons and lights are to be processed. Most of them will be the normal database referenced to the model origin, but some scene elements will be passed through a different perspective calculation which will allow them to appear correctly where they are positioned. This needs to be a flexible mechanism, as there are several forms of DCS which the modeler may wish to employ.

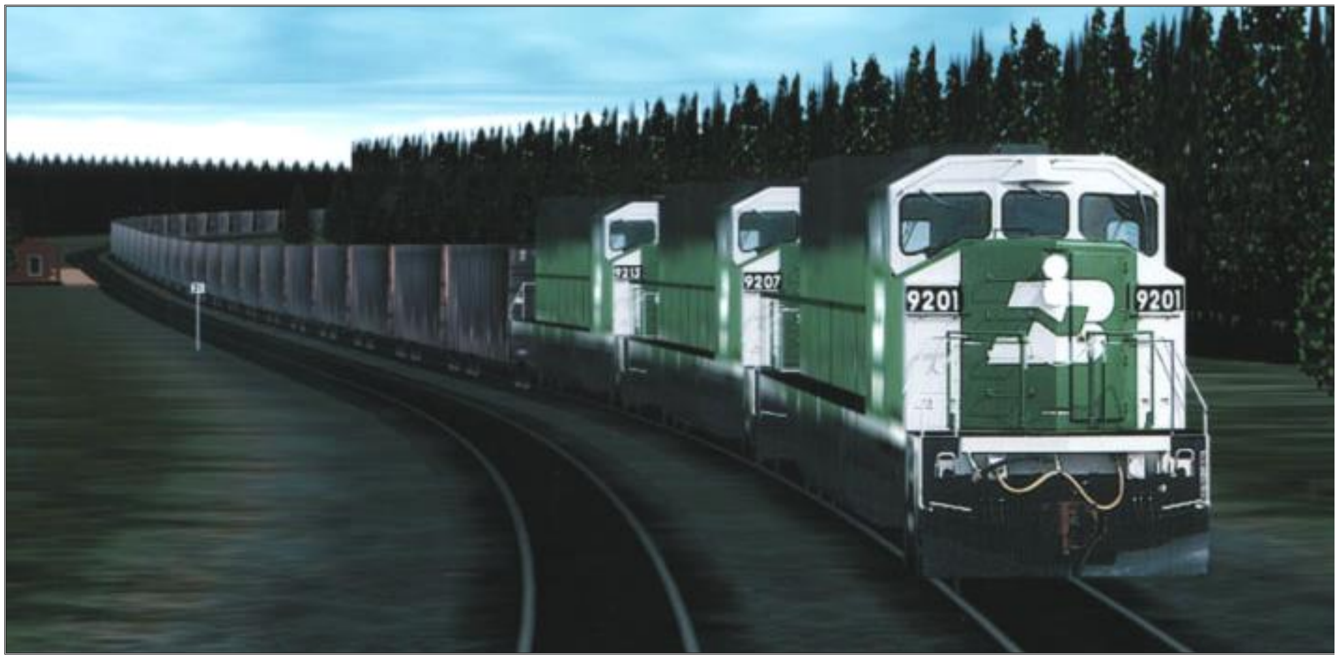
The independent DCS is one which is related directly to the database origin. This is the typical implementation for an aircraft in the database. The aircraft would be a six degree of freedom DCS and would be positioned in X, Y, Z, heading, pitch and roll. In some cases the application may not require all of these. A low detail ship model, for example, may only require X, Y and heading.

A chained DCS is one which is related to an independent DCS or to another chained DCS. A common example of a chained DCS is the wake of a ship model. The ship would be an independent DCS which, if it is a complex model, will require all six degrees of freedom. The wake attached to the ship should not be seen to roll or pitch along with the ship. It must therefore be processed separately in such a way that its position (X and Y) and heading are the same as the ship but it will receive no pitch or roll inputs.



SERVER

Another example is the refueling boom on a tanker aircraft model. The boom is required to pitch and yaw about the point where it is attached to the aircraft. It is therefore a DCS chained to the tankers independent DCS. A further example of a chained DCS may be the refueling pipe inside the boom. This has the same pitch and yaw as the boom itself but must change its length relative to the boom. An extreme case of chained DCSs is the probe and drogue refueling used by the U.S. Navy and most foreign services. In this case several chained DCSs must be concatenated in order to simulate the flexible hose attached to the tanker aircraft (fig. 5-12). A more extreme example is the train in fig. 5-12a. In order for the train model



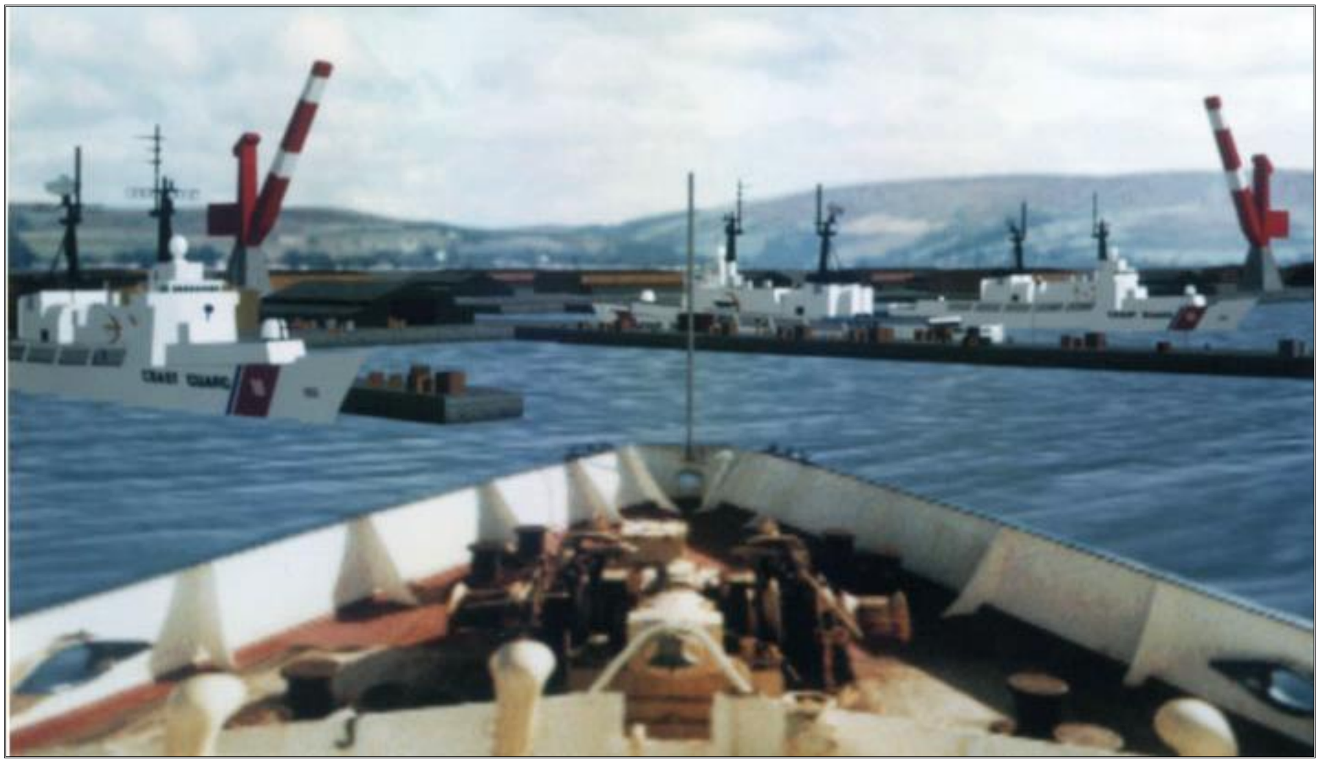
SGI/Hughes

fig. 5-12a - Each car of this train would require a separate coordinate system

to move along curved track each engine and car would require a separate DCS which would have to be positioned correctly relative to the track.

A special case of the chained DCS is when the coordinate system is related to the eyepoint. In a ship simulator, for example, the bow of the own ship must be simulated (fig. 5-12b). This may be accomplished by positioning a model of the bow such that it moves with the eyepoint. When gun sights or HUD symbology is provided from the visual system, it must be positioned in a similar way.

The number of DCSs which can be employed simultaneously and the impact they have on the system capacity can be an important performance parameter. The processing of the DCSs requires additional matrix math to calculate the separate coordinate systems and this will burden the system. Some system architectures are more forgiving of this additional load than others. Note that the number of polygons and lights attached to the DCSs is not a factor. Once the math is done for each DCS there should be no additional load for the polygons themselves.



E&S

fig. 5-12b - Bow polygon(s) chained to eyepoint coordinate system

5.6.1 Host Driven DCSs

The data to position the dynamic coordinate systems often comes from the host as part of the host data block which positions the eyepoint, sets the weather conditions, etc. There are several ways in which the host may derive the data.

One method is for the host to get the data from another simulator as in the case of an air-to-air combat training center or the large network team trainers (i.e. SIMNET and CCTT). Each aircraft or vehicle is considered a DCS in the other simulator(s) such that they can maneuver interactively (fig. 5-13). In the large team trainers this concept is carried to extremes with hundreds of vehicles involved in the same

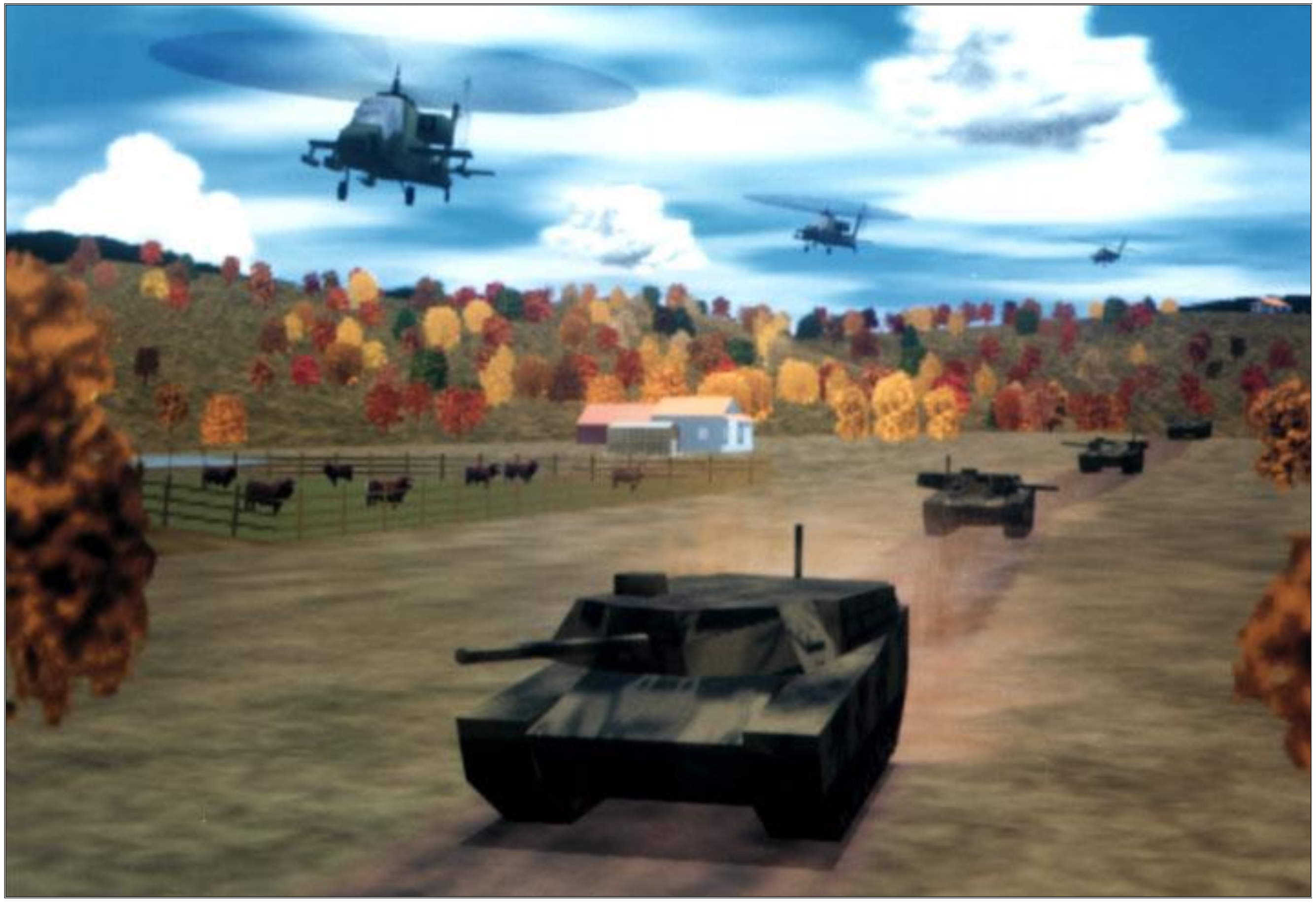


fig. 5-13 - Multiple host (SAFORS) or network driven dynamic coordinate systems

exercise. Some of these are other simulators on the network, but most of them will be controlled by Semi-Automated Force (SAFOR) systems. These complex systems will maneuver large numbers of DCSs in a programmed manner. The SAFORs vehicles will react differently depending on the actions of the manned simulators.

Another approach is for the instructor to maneuver the DCS via some type of controls at his station. This creates some difficulties, depending on the complexity of the simulation. The instructor already has a lot to do in just running the training mission without having to operate his own vehicle. Even if a second instructor is involved, he will probably not have a very good visual representation of where his vehicle is or what it is doing.

The more common approach to host driven DCSs is for the host to contain a recorded path for one or more objects. These are sometimes constructed by flying the simulator through a maneuver and recording the flight dynamics. These, perhaps with some modifications, are then played back into the system and the DCS attached to them. This is a particularly good way to create a flight path for formation flight training. If the path is intended for a ground vehicle, a target for example, it will be difficult to record the dynamics in an aircraft simulator and some other approach may be required. Also, the path of the vehicle on the ground will probably be limited to that database. A truck or aircraft which is intended to pull out onto a runway to create a hazard will not be effective if there is no road or taxiway in the area where the host positions it.

5.6.2 Software/Database Driven DCSs

Other methods of providing positions for the DCSs is to have the visual system software create them, or store them, or to have the database itself store the paths (fig. 5-14). The visual computer has many of the same problems creating or storing this data that the host does. Very little is gained by moving the task to the visual. The database, on the other hand, working with the visual program has a great deal of potential.

One approach is to include in the database several key points. A key point is a position, in database

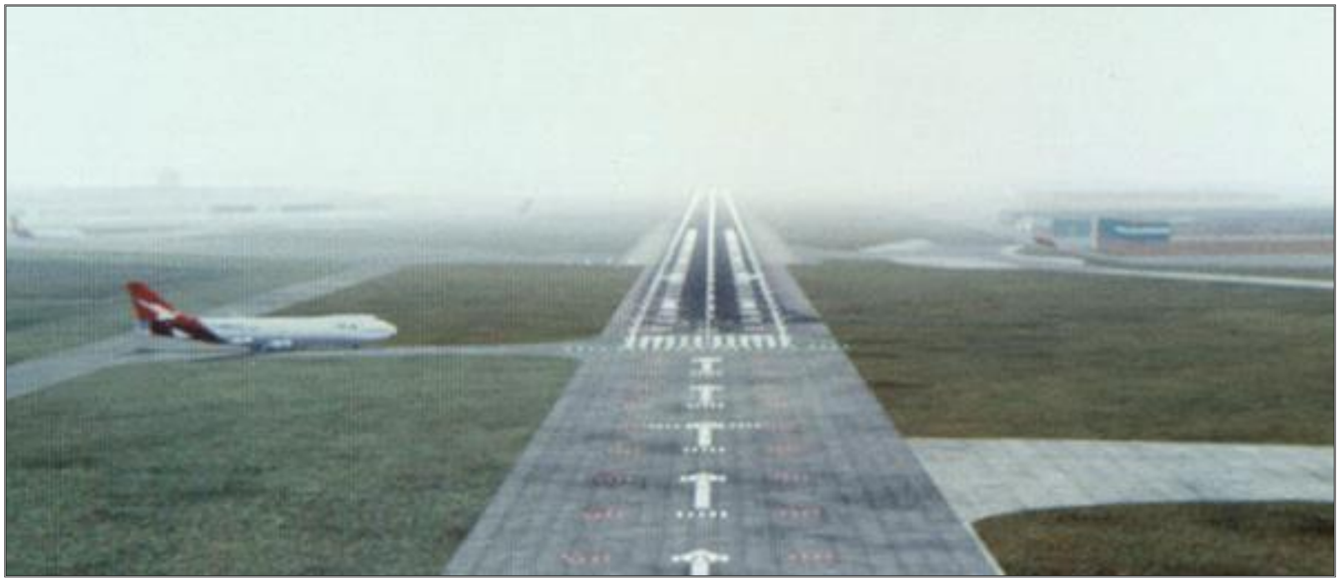


fig. 5-14 - Aircraft on taxiway can be moved onto the runway to create an emergency

coordinates, which the DCS will pass through. In addition to position it will also include attitude information and speed. A path within the database is defined by listing several of these points. A routine in the visual program then moves the DCS from one point to the next, the position, attitude and speed of the object being interpolated between two or more points. When the object reaches the end of the way point list it can either wait there or it may start over again. Since the points are part of the model they can be defined in appropriate database locations. The movement of the DCS may be initiated by the instructor in order to create a training problem or it may be on at all times to create a dynamic effect in the scene. Some systems will provide a high fidelity path through the key points with turns being interpolated so that they appear more natural rather than an abrupt change in heading to the next segment of the path.

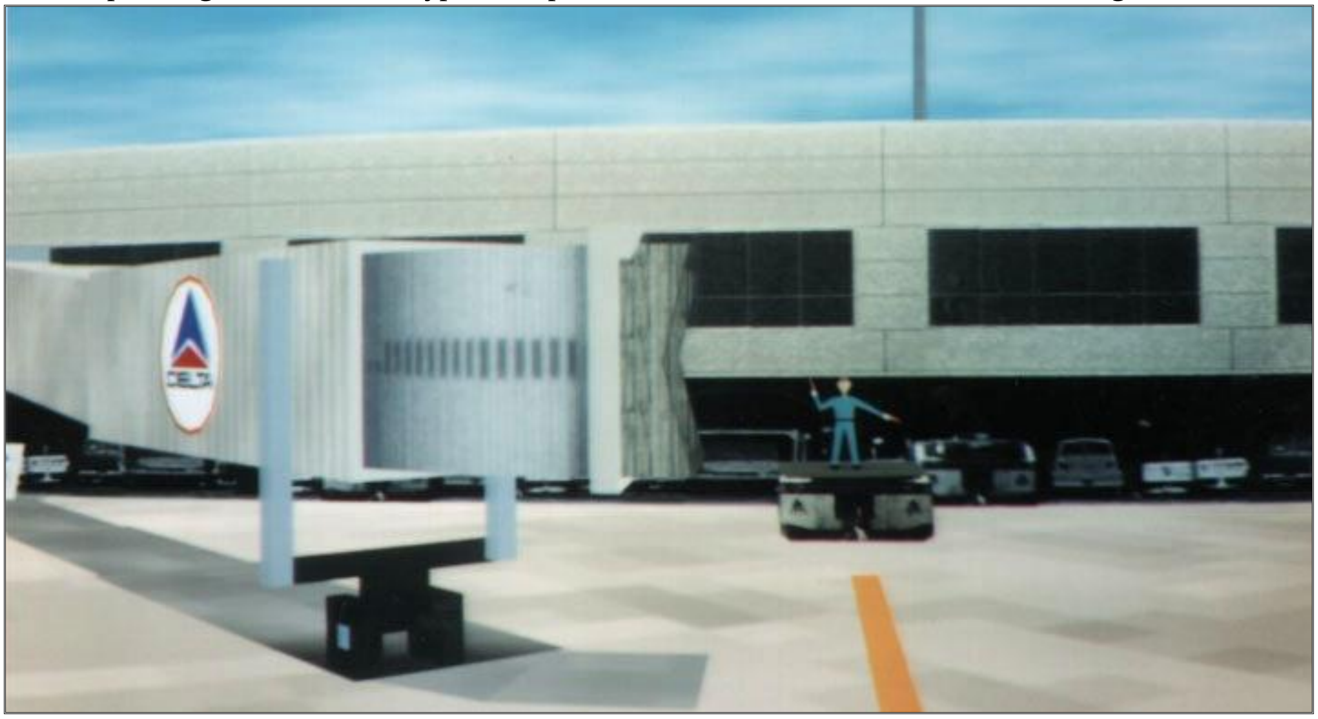
A similar implementation is for the visual program to cause the DCS to converge with the eyepoint on a collision course. The parameters are again stored in the database. They include a horizontal and vertical angle from which the object will approach, a speed at which it will approach and, since the idea is to avoid the collision, a time interval in which the student will be allowed to maneuver. The speed may be expressed as a ratio of the eyepoint speed. This feature would also be selected by the instructor in order to create a mid-air collision threat. Because the flight paths are referenced to the eyepoint, rather than a specific location in the database, the DCS may be initiated at any location.

5.6.3 Animation

Animation is not a DCS as described above, but it does use many of the same constructs in assigning polygons to specific applications and it does cause objects in the scene to move. Animation requires including several versions of an object in the database and then cycling through them in sequential order to create a dynamic effect. This is, of course, the same way that animated cartoons are done.

The advantages of animation over the normal DCS techniques are that in addition to movement, the size, shape, color and transparency of the object may be effected. Also, one animation sequence may replace a large number of complex DCS effects. Raising or lowering the landing gear on a formation aircraft, for example, might require one or more DCSs for each gear as well as one for each gear door. By modeling approximately 20 versions of the gear at states between up and down a smooth transition may be achieved. The modeling task is not overly complex since each state is similar to the next. The polygons for the gear and doors are rotated in the modeling tools to create the correct positions.

Another advantage to animation is that several alternative sequences may be available. A weapons effect, for example, might activate one type of explosion for a hit and another for a miss to give the student



E&S/Hughes

fig. 5-15 - Level D database with animated taxi director

immediate feed back. Another example is an aircraft taxi director which includes a number of animated arm signals (see fig. 5-15). As the aircraft (i.e. the eyepoint) approaches, the animation sequence is selected which portrays the correct signal.

The disadvantages to animation are that the different states do have to be modeled. While all cases are not as straightforward as the landing gear example above, they do tend to be somewhat similar due to the types of effects being implemented. A more serious complication is that all of the states which may be activated must be maintained in the active memory of the image generator. This is normally not a serious limit because the objects tend to be small with few polygons in each state.

The mechanism which controls the animation routine may allow the rate to be selected so that each state is displayed for several fields if appropriate. It will also determine if the cycle repeats itself, is displayed only once or is chained to additional animations. These might be combined as in an explosion (single cycle) which would then initiate a flame and smoke effect (continuous cycles).

6. DATABASE ISSUES AND TOOLS

In addition to the features implemented in the database which were covered in the previous chapter, there are a number of issues, some long standing and some new, which relate directly to the various aspects of the database and its interaction with the image generator(s). Some of these involve methods of reducing the cost of the models which can be a significant factor in the overall system cost. Others entail ways to re-use a database once it has been built (and paid for). A related issue is the problem of insuring that all of the databases required to simulate today's complex systems contain the same information.

6.1 Database Correlation

With the advent of more and more complex simulators and training requirements it was inevitable that eventually more than one database would be involved. In addition to the out-the-window visual scene, we now have various electro-optical sensors and various types of radar to consider. In the near future aircraft will carry their own CIG devices in the form of digital map displays and graphical tactics displays. It is, and will be, important that all of these devices present a consistent version of the world if the simulator is to provide the maximum training value.

When simulating these devices some will receive their imagery from the visual system and can use a version of the visual database. Others, particularly the radar, have their own image generators and databases. The challenge is to achieve correlation between all of these devices without overly limiting any of them.

The advent of distributed simulation (DIS) networks has placed new and more extreme correlation requirements on simulators and visual systems. If a "fair fight" is to be guaranteed to all players then everyone in the problem must see the same things. This is difficult enough when all of the players are the same type of vehicle, but if some are tanks and some are helicopters, for example, the problem is much more demanding. In this case even using the same image generator and database may not be possible.

6.1.1 Radar Correlation

Radar correlation is the most difficult in current simulators. Radar simulations tend to have fewer capacity and resolution limitations than visual systems do. Their databases cover large areas and have a great deal of detail. In particular, they normally include a much finer mapping of the real world terrain than is possible from current CIG systems.

There may be several radars in current simulators which the visual is expected to correlate with. In commercial airline systems the weather radar is an important part of the training tasks. Typical weather radar systems include a collection of storm scenarios which are selected and positioned by the instructor. The visual database must include similar representations of the same scenarios. In the calligraphic systems most often utilized by the airlines the capacity may not allow for a thorough emulation of the radar model. A two dimensional silhouette of the clouds may be used rather than a solid model which would require more polygons. Also, the visual system may be limited to displaying the cloud model only when the eyepoint is above a cloud deck with none of the ground database in view. The weather radar device may treat the clouds as one or more DCSs to provide a changing weather scenario. The host will then be required to send the appropriate cloud position(s) data to the visual.

In military simulations the visual database must often be correlated with the ground mapping radar. The radar database is normally derived from the same NGA data as the visual database. The problems of creating visual databases from this source will be discussed in 5.4.2. One novel approach to this problem is to build the visual database from NGA data to its optimum capacity and then to use the data in the visual model to produce the radar database. This means that the two correlate and both will represent the real world area in question. It is true that this technique somewhat reduces the fidelity of the radar model but careful selection of the content of the visual model (and therefore the radar model) will insure that nothing critical is omitted from either one.

Current military aircraft incorporate Synthetic Aperture Radar (SAR) which creates imagery much closer to a visual representation than to traditional radar. The normal radar simulation devices must be improved considerably to provide the detail and perspective that SAR images contain. Even then the database storage requirements are such that only a few locations in the overall database can be depicted. It may evolve that some portion of the visual system is better suited for this application than traditional radar simulation techniques.

Another, slightly different radar correlation problem arises with terrain following radar. The requirement here is similar to height above terrain. Either the visual system must provide data for the terrain following (or terrain avoidance) radar or the simulator must store and process its own version of the terrain database. The line of sight ranging discussed in 4.7 does not satisfy this requirement because the radar beam is too wide. This is a difficult simulation task which will probably require some form of custom hardware to provide the appropriate data. For example, a "piece" of a channel could be employed to process the database through a very small field of view (the same as the radar beam). This field of view would be positioned in the same manner as the radar antenna. The distance (i.e. Z depth) to the nearest polygon it "sees" could then be provided to the host as a simulation of a radar return. This implementation would require special hardware or a very modular system.

6.1.2 Sensor Correlation

Although low light level television and a few other electro-optical sensors must sometimes be considered, the most common, and most difficult, sensor to be simulated is infrared. Many aircraft and other military vehicles incorporate at least one infrared device and some will have several different ones. Infrared requires much the same database detail as the visual image but with additional color, or gray shade, considerations. Things that appear as one large polygon in the visual image, like the side of a ship, may require several white areas representing hot spots in the IR image. Similarly, areas with a high degree of detail in the visual image may appear as all one gray shade in the infrared scene. There may also be a requirement for a black hot version of the image which could effect the manner in which the color selections are made.

In addition to these database issues, a choice of several fields of view, all of them small, may be required. This may mean that a higher level of detail will be required due to magnification which. In order to simulate infrared's ability to penetrate some forms of smoke and haze, the visibility in the IR channel must be different than that in the visual channel (fig. 6-1,2). Further, since many IR applications involve night scenarios, the overall brightness of the IR channel relative to the visual ones will have to be different.

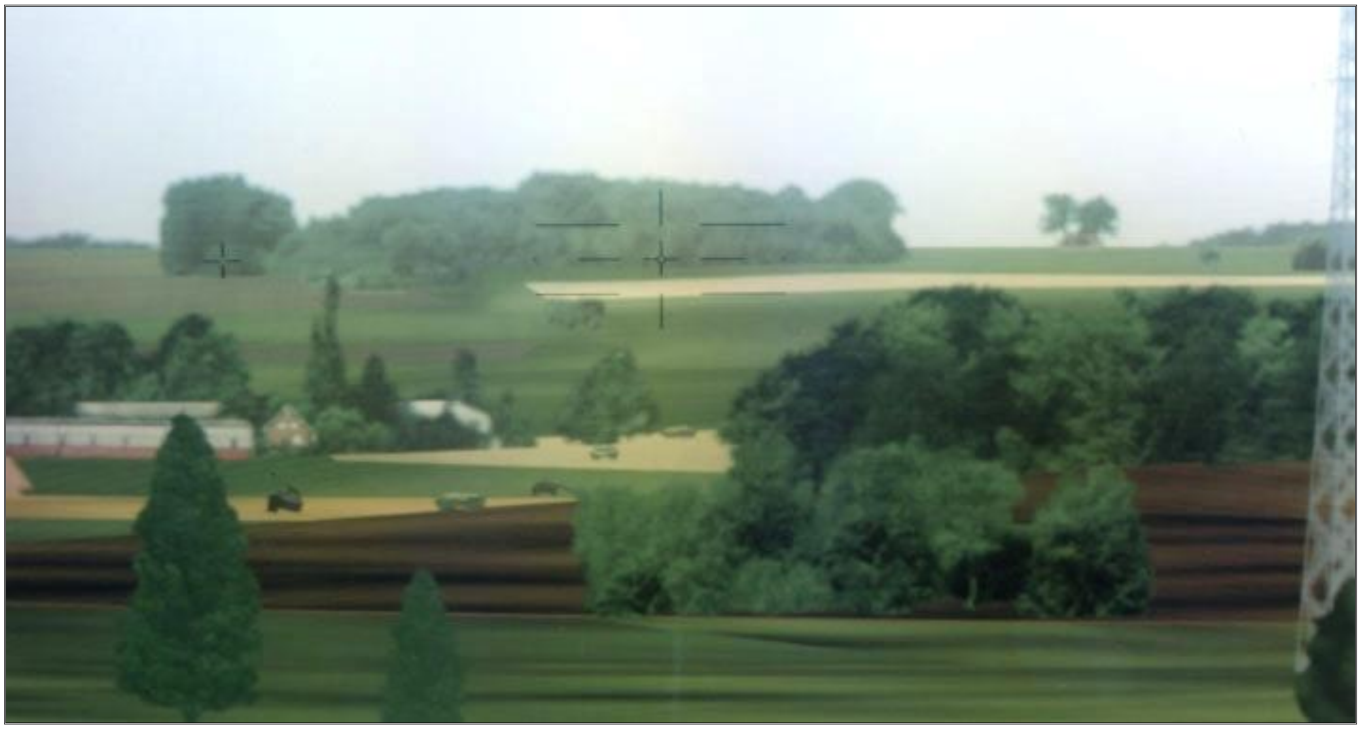


fig. 6-1 - Scene in visual channel

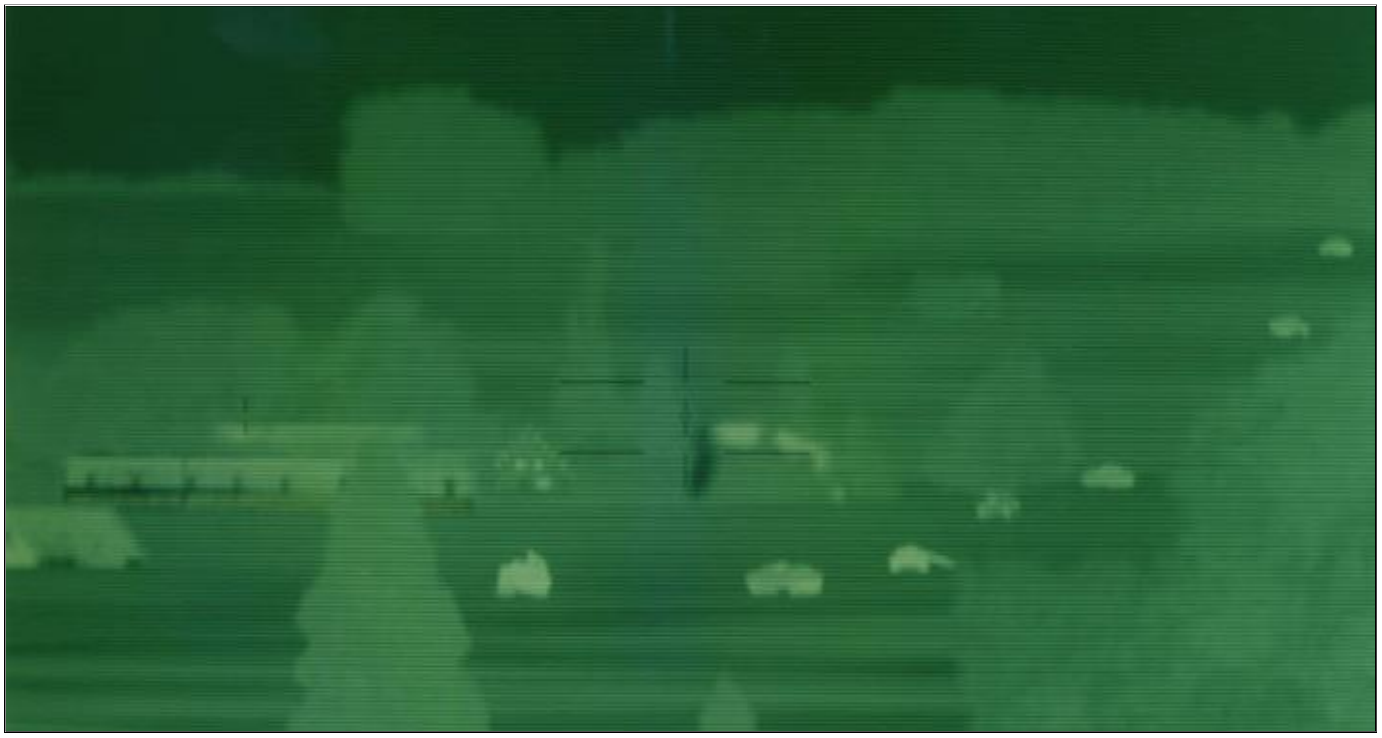


fig. 6-2 - Same scene (magnified) in infrared sensor channel

In addition to level of detail problems, the potentially high magnification of these systems creates other difficulties. On the bright side, a small field of view will imply relatively few polygons so the channel may not require high capacity. Since these sensors are normally slewable however, the possible field of regard is large. This means that a very large number of polygons must be kept on-line so that these magnified images are instantly available. This requirement may well overload the active database memory of the image generator.

One obvious approach is to use a separate image generator and database for the IR channels, although this does not solve all of the problems. Some CIG devices provide more than one color for each polygon in the database. These alternate colors (or gray shades) may be assigned to represent IR emissivity values and applied to the IR channel(s) of the image generator. This allows the same IG and database to provide both images which helps to insure correlation between the sensor and the visual. This approach provides a good representation of an infrared scenario if the variations due to weather changes are not to great. Applications which require a very flexible IR database where the simulated emissivity of objects in the database must change in response to humidity, temperature, time of day, etc. may require a more through simulation.

Although the imagery from IR sensors is constantly improving, it tends to be relatively low resolution and include various types of noise and clutter. After years of striving to create clear images without aliasing or other anomalies, CIG manufacturers find themselves looking for ways of putting some of these artifacts back in. Of course IR artifacts are not exactly the same as those found in typical CIG systems so more is involved than just removing the fixes. The artifacts imposed by the IR sensor which must be reproduced will also differ from one sensor to another. A device which can be programmed to provide the appropriate type of image clutter and noise is required. A post processor which is placed between the DP and the display is one solution.

6.2 Transportability

The problem of transportability is related to, but more far reaching, than correlation. This implies taking a database designed for one image generator an using it on another system. The purpose, of course, being to eliminate, or at least reduce, the cost to the user. The military, in particular, operates several different image generators in a number of applications. While many of these are very different, some are remarkably similar. Simulators of high performance aircraft, for example, could all use the same system. The procurement process and timing, however, tends to result in different manufacturers winning various contracts and certain system aspects must be duplicated. The database is one of these.
The idea of using a database on more than one system is not new. It has been discussed for many years. More recently, however, several factors have combined to increase the interest in database transportability. One of these is certainly the cost of current databases. Tactical and strategic aircraft training requirements have lead to very large and expensive databases. Literally millions of square miles may be required on-line for some applications. Another factor is the increasing commonality of image generator architecture throughout the industry. While it is true that no two image generators are identical by any means, current systems are much closer than in the past. For example, virtually all systems use the vertex and polygon definition for objects. Many of these moves toward a common solution result from the most efficient use of the available electronics technology and the trends are continuing. It is very likely, for example, that in the near future all new systems will utilize the Z buffer priority technique because current technology makes it possible and desirable.

6.2.1 Project 2851

To date, the most ambitious attempt to implement database transportability has been Project 2851 which was sponsored by the Air Force through Planning Research Corporation (PRC) of McLean, Virginia. PRC, with help from the simulation industry, attempted to develop a database process that would provide a single data source which all manufacturers could use to generate their databases. This was to include radar and sensor simulations as well as visual databases.

The program also supported a large number of Special Interest Groups, made up of industry representatives, concerned with the detailed implementation of various aspects of the process. These groups included Level-of-Detail, Photo Texture, Digital Image Exploitation, Mission Rehearsal, Terrain Polygonization and Networking. PRC's contract with the Air Force included the development of the basic formats, demonstrations to validate the process and the set up of a production facility at the NGA to produce the databases.

The long range plan was for the NGA to manage the production of source data in the P2851 format for large areas of the world. Each image generator manufacturer would then develop a "translator" to convert the NGA supplied data into the appropriate format for his specific system. It was expected that certain mission specific areas and objects will require custom efforts in order to satisfy the detailed training requirements.

There were a number of problems with putting these concepts into production. Not the least of which was that NGA did not have the production facilities (or budgets) to support the level of effort that was anticipated. It was eventually decided that the only practical implementation was for Project 2851 to develop interchange formats in which database information could be exchanged between IG manufacturers and other users. Databases will continue to be developed by visual system suppliers as they have always been, but there will be an added requirement that a version of the database be supplied in the Standard Interchange Format (SIF). SIF has been documented in Mil-Std 1821. These copies of the databases will be stored at an Air Force funded facility at Kirtland AFB. When gaming areas are required for future military simulation programs the vendor will/may make use of any previous versions of the area from the library (similar to current NGA data).

The Kirtland facility was opened in mid 1994 and is currently building up its library of databases. It is not yet clear how useful the SIF data will be in lowering the cost of future databases. SIF has been found to have some shortcomings in the forms and content of the data it can interchange effectively, but there are procedures in place to request improvements. It was never intended that the 1994 version of SIF would be the final word in database formats.

The International Air Transport Association (IATA) has also embraced the SIF standard. A working group of airlines and IG suppliers was established in 1990 with the goal of defining a similar format (called DBIF) which the world's airlines can use to limit the cost of future databases.

6.3 Database Generation Tools

Modeling tools have become a much more important aspect of CIG system in recent years. This is primarily due to the increased part that the database has to play in the visual system's contribution to the training objectives. Early databases were just that, data. Lists of geometry data which defined polygon and light characteristics. Today's databases are much more complex with many features which effect the way that the visual system performs (or does not perform). Furthermore, the recent emphasis on mission rehearsal and the associated requirements for very rapid database construction and update has tended to accelerate developments in this area.

The two aspects of modeling tools which the user should be concerned with are; 1) the ease with which the manufacturer can design and build a database which will satisfy his needs, and 2) will the user be able to modify those databases at a later date or even build new ones. The first of these issues will relate to the expense of the database. Automated tools to minimize construction cost will be discussed below.

Having recognized the need for the users to build or modify their own databases on a continuing basis, almost all manufacturers will include some or all of the database generation tools as part of the software utilities which are available with the system. In early systems, and many current ones, the various modeling tools would run on the visual computer in an off-line mode (i.e. when the visual was not required to process images). In recent years, as the databases have grown larger, they require relatively powerful computers with extensive operating systems to handle them efficiently. At the same time the front end computer of the image generator has tended to become smaller and more closely customized to the needs of the image generation task. Consequently, many systems now require a separate modeling work station which executes the database tools. These tools come in a wide range of capabilities but are generally comparable in performance to the image generators to which they apply (i.e. more powerful system have more powerful modeling tools).

6.3.1 Text Editors

The basic form of data entry for a visual database is the text editor. In its simplest form this is one of the editors which are provided with the visual computer or the computer the modeling is done on. For a large database this can be a tedious task involving long list of numbers in various formats. The resultant source file will then be compiled into the object form which the image generator will process.

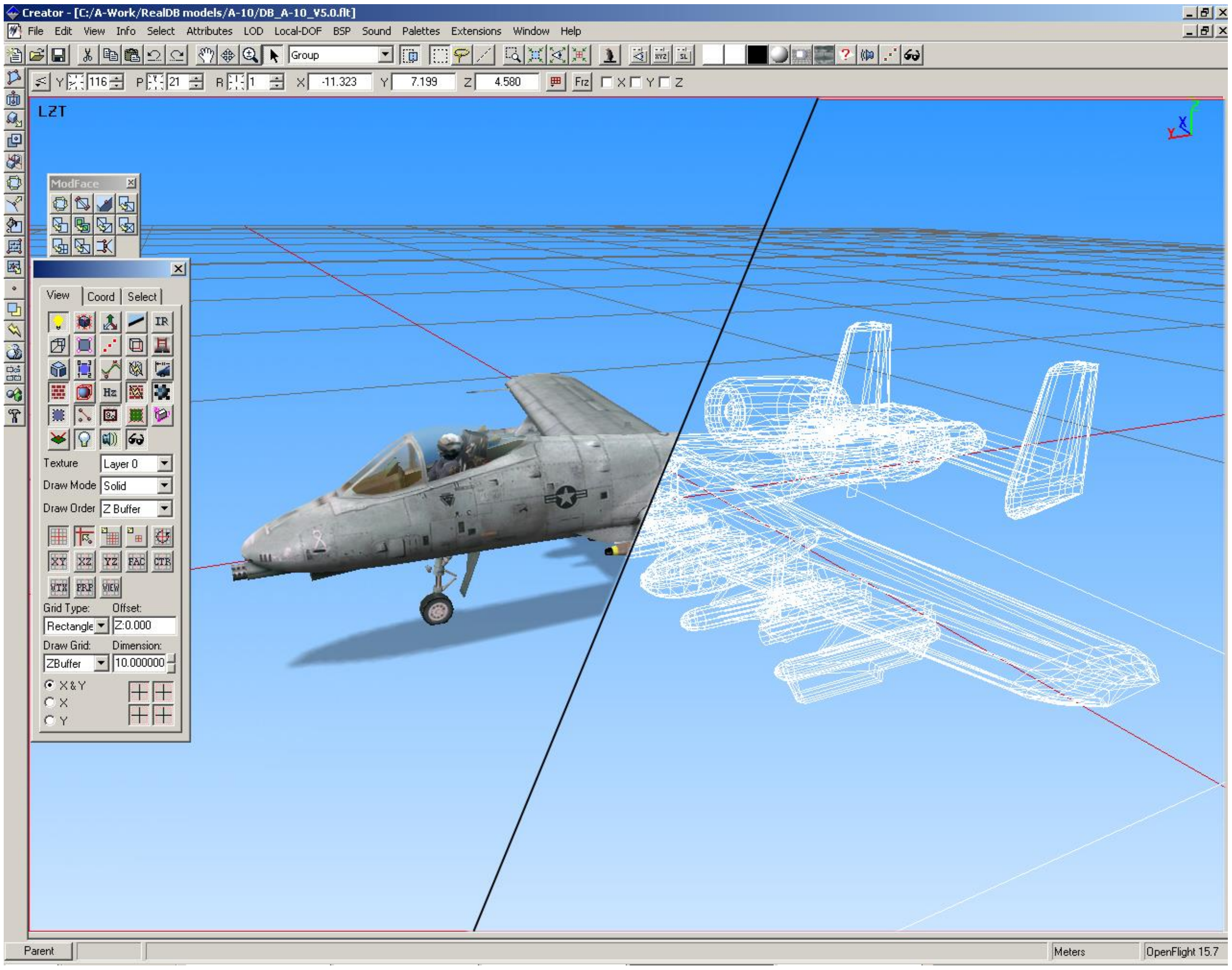
The typing task is aggravated by the amount of numerical data which is overly subject to errors. These may not be obvious until the model is displayed. Format errors are another problem. The image generator will expect certain features (vertex names in the polygon definition, for example) to be included in a specific order and arrangement. If this is not the case, the database may not compile or errors will appear. The error messages which are returned from the compiler in this case are critical to debugging the file.

After the compiler, the database may require processing through a linker of some sort. This program will join several small pieces of compiled database into one large database ready for the image generator. This serves two main purposes. It connects all of the hierarchical aspects of the database into one contiguous tree structure. It also allows the modeler to work on and compile small, easy to handle sections of the database rather than the entire model which may be quite large. The advantages of these types of text editors are that they will generally include powerful editing features which will allow rapid manipulation of files and data. It also provides procedure type modeling techniques with which the modeler can write software routines which will construct large numbers of polygons.

6.3.2 Interactive Editors

Some systems will provide a customized, interactive editor. These utility programs will lead the modeler through the data entry to some degree and will insure that the data is formatted correctly. This type of editor is particularly useful when first learning the database process because it will remind the modeler of what entries are required. If a light string is made directional, for example, the editor will next ask for the heading. Otherwise heading would not be requested. One disadvantage to this type of editor is that they are somewhat slow, particular for the experienced modeler who does not need to be reminded of what to type next. Nor do they provide a great deal of flexibility in manipulating files although some basic features will be present such as translate, rotate and scale. Even though interactive editors may not handle general procedural modeling they may allow macros. Much like a software macro, database macros will describe the basic shape of an object and take as arguments the position, attitude, color, etc. These may be designed to produce simple objects like buildings or trees or very complex sets of polygons such as a complete set of runway markings.

6.3.3 Graphical Editors


Creator - [C:/A-Work/RealDB models/A-10/DB_A-10_V5.0.flt]

Y 116 P 21 R 1 X -11.323 Y 7.199 Z 4.580

Texture: Layer 0
Draw Mode: Solid
Draw Order: Z Buffer
Grid Type: Rectangle
Offset: Z:0.000
Draw Grid: ZBuffer
Dimension: 10.000000

Parent | Meters | OpenFlight 15.7

MultiGen

fig. 6-3 - Graphical editor from MultiGen (now Presagis)

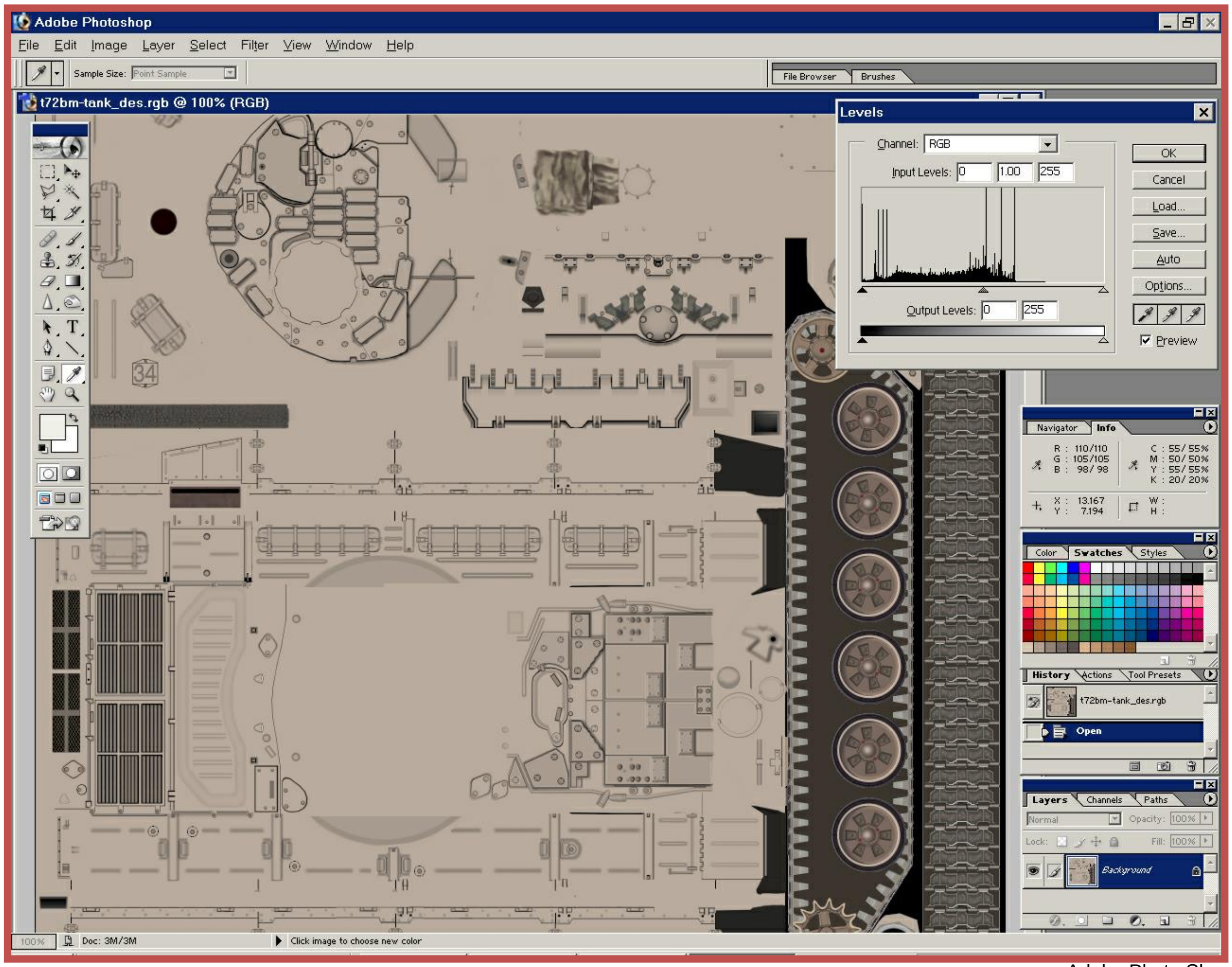
Another modeling tool which is frequently offered with current systems is referred to as a graphical editor. Many of the modeling tools discussed above will have a graphical mode, but this will normally only provide a rendering of database components after they have been entered in the standard manner.

Graphical editors (fig. 6-3) allow the entry of the vertices and polygons directly on the screen. They may use a digitizer and mouse or some form of touch sensitive screen. Since working in three dimensional space is difficult, numerous aids are normally provided. These include grids which can be displayed to form a plane to work from and mechanisms which will insure planarity. These tools will give a very good preview of the database so that less time is required on the image generator for final tuning. Although the tools will depict a large number of polygons, advance features such as texture and anti-aliasing are not generally available.

One important aspect of any collection of modeling tools is that it provide meaningful statistics. The number of polygons and lights in the database and the number displayed from specified eyepoints is invaluable to the modeler. The image generator itself may include statistics which monitor the system load at various points in the hardware, but it is important for the modeler to have some idea of the database content prior to moving it to the IG.

In addition to the modeling systems of this sort which many IG manufacturers supply with their systems, several independent companies have developed modeling tool packages of their own which may be available. In some cases this was done to support large in-house database efforts. Others have been developed as a product in their own right. An additional advantage of these systems is that they may be compatible with more than one image generator, and perhaps more than one manufacturer, with little modification.

6.3.4 Texture Editors



Sample Size: Point Sample

Channel: RGB
Input Levels: 0 1.00 255
Output Levels: 0 255

R : 110/110
G : 105/105
B : 98/ 98
C : 55/ 55%
M : 50/ 50%
Y : 55/ 55%
K : 20/ 20%
X : 13.167
Y : 7.194
W :
H :

Normal Opacity: 100%
Lock: Fill: 100%

100% Doc: 3M/3M Click image to choose new color

Adobe Photo Shop

fig. 6-5 – Photo Shop is now the "standard" texture editor

With texture imagery becoming an ever more important part of the visual scenes it is not surprising that tools to create and manipulate texture maps are also receiving a great deal of attention. The original texture editors were similar to "paint box" type programs. These primarily allowed the modeler to create a map by drawing the pattern by hand. Since the early texture maps were relatively small and the total numbers relatively few by today's standards, this approach was acceptable.

One important function of even the early editors was that they provide an easy mechanism for making the pattern tessellate with itself and/or blend smoothly with neighboring patterns. In current systems more and more of the texture maps are derived from photographs. The ability to manipulate the map so that it blends properly is more important than ever. The maps may be derived from very different sources with different resolutions, color and contrast. A typical problem might involve a collection of satellite photographs of a large area and several higher resolution aerial photographs of a small portion of that area which is the area of interest (i.e. target).

The satellite photographs must first be assembled into a mosaic which will be applied to DTED derived terrain polygons. This process is complicated by registration errors between the images and contrast problems because the pictures may have been taken on different days (cloud cover?) and different times of day (shadows?). It will then be necessary to insert the higher resolution images into this mosaic in such a way that the change in image quality and content is not overly apparent. It would be easy for the flight crew to find the target if they knew it was always in the center of an obviously different database area.

A de-perspective feature is also helpful. Very few source photographs will have been taken from an exactly perpendicular angle. A method of "squaring up" the photo image is required. It is also beneficial if the photo image can be referenced by the graphical editor functions. This will allow the modeler to build objects using the photo as a source. Three dimensional structures can be raised up from their two dimensional image in the photo, for example, and placed on the terrain model in the appropriate places.

6.4 Automatic Database Generation

Modeling, like software programming, tends to be a very labor intensive task. The cost of producing even small databases has historically been a significant part of the overall system price. With the advent of very large military databases, these cost have become a limitation in the application of the technology. Users cannot always afford all of the databases they would like in order to make the best use of their simulators. This is the basic problem that Project 2851 (see 6.2.1) is intended to solve.

One solution to this situation is to remove the modeler from the process by providing computer programs which will construct the databases. This is obviously more easily said than done. The modeler's task, in a few words, is to take source data (maps, photos, etc.) and create a polygonized version of the area in question which will satisfy the training needs and not overload the image generator. In order for a computer program to duplicate this task it must first be able to interpret the source data, and secondly, must be able to recognize and respond to the training needs.

6.4.1 National Geospatial-intelligence Agency (NGA)

Several approaches to achieving this automation have been considered. Most involve some form of digitization of the real world which the program will interpret and convert into polygons. Historically, the most promising of these approaches, and in fact the only one seriously pursued to any degree, is the use of

National Geospatial-intelligence Agency (NGA) digital data. This data results from an ambitious program at NGA to digitize the world for use in radar simulations. This program was not originally intended to provide data for training simulations, but for radar predictions (yes, mission rehearsal). These would support operational missions and, more recently, some missile guidance mechanisms. The original Level I NGA data came in two forms; terrain elevation data (DTED) and cultural feature data (DFAD). The application of this data to visual databases has two major problems. The first problem being that there is too much data. The second problem being that there is not enough data.

The terrain elevation data provides the height of the ground above sea level for points on a 1 arc sec. grid (about 300 ft.). A faithful rendering of this data by triangular polygons would require over 800 polygons per square nautical mile for the ground alone. Current CIG systems do not approach this capacity unless the visibility is severely restricted.

The feature data, because it was originally designed to support radar simulations, only includes features which are significant to the radar. This means that several things are missing, namely roads and highways, which are critical in a visual database.

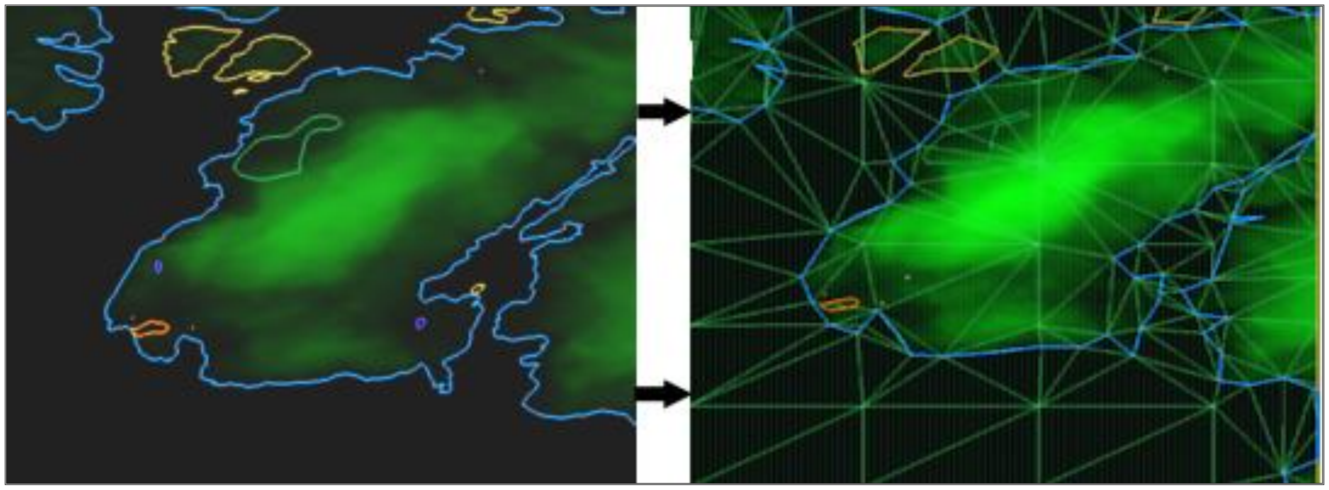


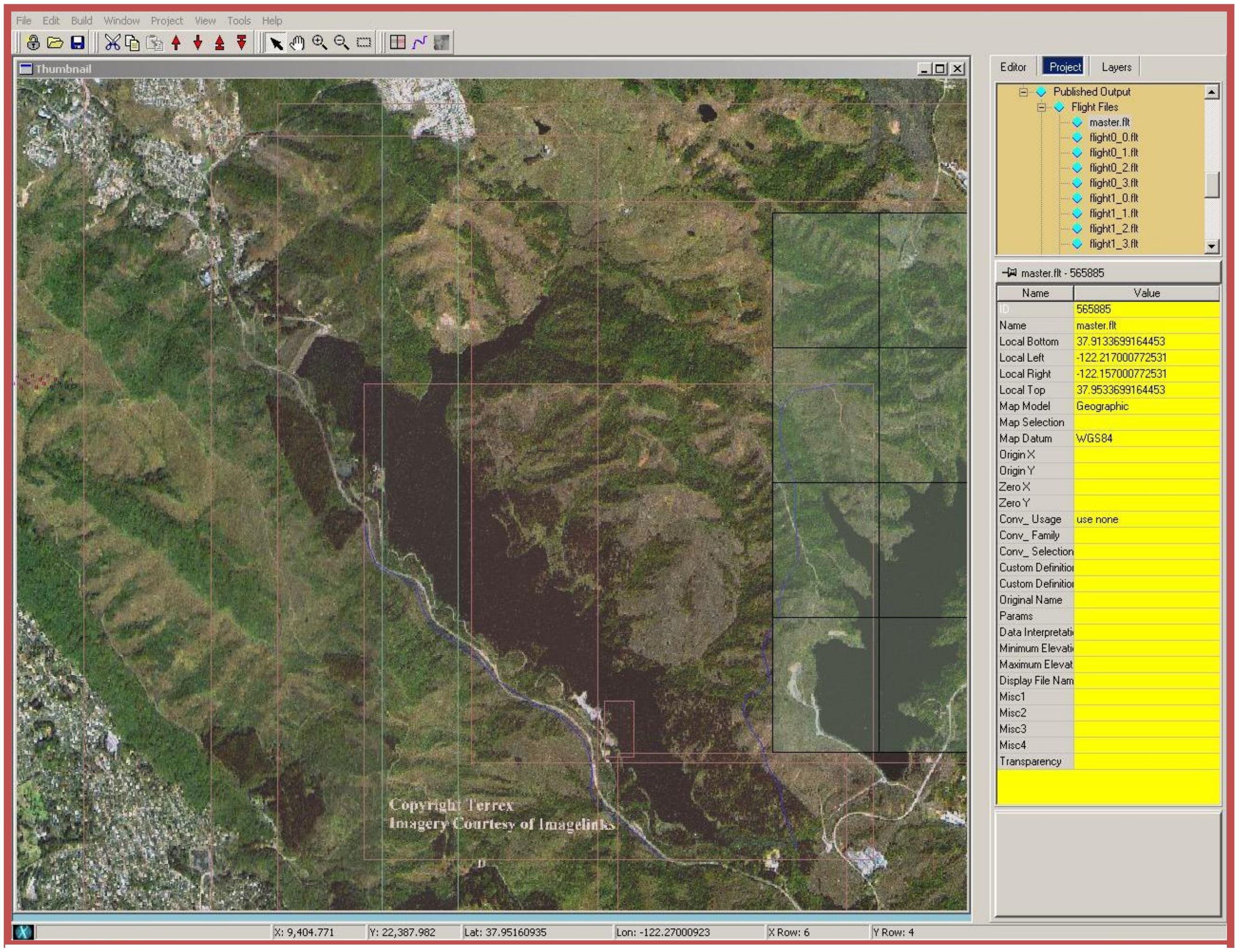
fig 6-6 - Terrain skinning tool creates terrain from DTED and places or creates DFAD objects.

NGA has recently begun correcting the second problem. After experimenting with Level V and Level X, both of which contained more than enough information for the visual system, but were too expensive to produce, NGA have begun issuing Level I edition 2 data. This is similar in resolution to the original Level I but incorporates the lines of communication and other features that must be included in a visual database. One of the results of Project 2851 is expanded feature data (XDFAD) which increases the resolution of the placement of features. It also increases the number of feature categories which are identified.

There are two basic approaches to filtering the terrain data to usable parameters. One solution is to only consider significant changes in slope and elevation. This means that areas which are primarily flat are represented by a few large polygons and areas which have drastic terrain changes are represented by many smaller ones. This works out well because flat areas tend to have additional requirements for polygons (farms, cities, roads, etc.) where rugged areas do not.

A second approach simply averages the DTED data into a lower resolution grid which requires fewer polygons. This results in lower fidelity terrain than the above approach, but the uniform size of the polygons is much easier to handle within the database structure.

The algorithms incorporated by these programs to optimize the terrain are very complex. Several different factors are taken into consideration in order to end up with as close a representation as possible. It is likely that the programs are applied several times before the modeler gets the ratio of terrain polygons to feature polygons that best satisfy his and the user's requirements.



master.flt - 565885

Name	Value
ID	565885
Name	master.flt
Local Bottom	37.9133699164453
Local Left	-122.217000772531
Local Right	-122.157000772531
Local Top	37.9533699164453
Map Model	Geographic
Map Selection	
Map Datum	WGS84
Origin X	
Origin Y	
Zero X	
Zero Y	
Conv_ Usage	use none
Conv_ Family	
Conv_ Selection	
Custom Definitio	
Custom Definitio	
Original Name	
Params	
Data Interpretati	
Minimum Elevati	
Maximum Elevat	
Display File Nam	
Misc1	
Misc2	
Misc3	
Misc4	
Transparency	

X: 9,404.771 Y: 22,387.982 Lat: 37.95160935 Lon: -122.27000923 X Row: 6 Y Row: 4

The features which must be added to the terrain skin to complete the model are taken from a library of typical objects. The NGA data defines objects in a similar way. There are different types of buildings, bridges, power line poles, etc. which are referenced in the data by code numbers. The visual database library will contain one or more versions of the same objects. Some may have to be scaled to fit into a specific application (a bridge, for example). Several versions of some objects may be required to provide variation in the scene even though the DFAD data may not differentiate between one application an another (a factory, for example).

When the modeler creates a database he will make numerous intuitive judgments about the placement of objects and how many polygons to apply to certain objects. This is particularly true when the database is complex and very close to system capacity. These judgments cannot be made by a computer left to assemble a database without human intervention and direction. The database would suffer from some degree of inefficiency which may be fatal to both the image generator operation and training requirements.

APPENDIX A: COMPUTER IMAGE GENERATION GLOSSARY

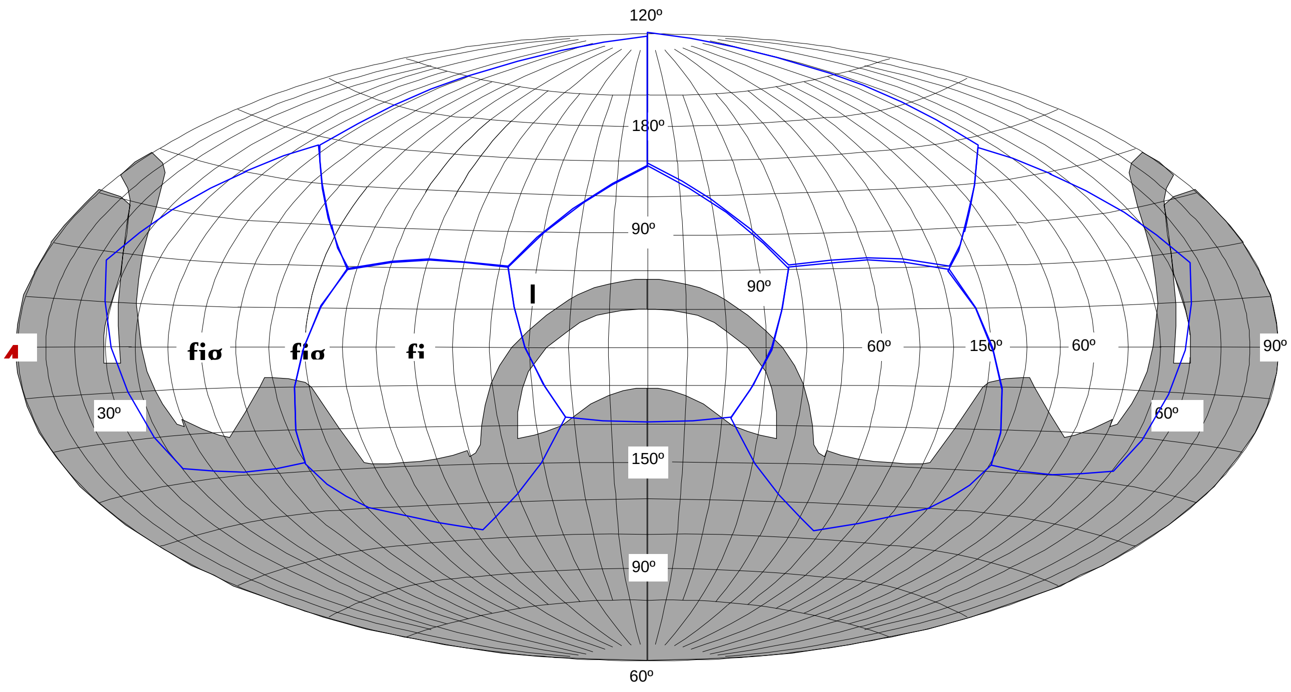
The definitions given below are specific to Computer Image Generation in general and simulation in particular. It is understood that some of these terms may have additional and different meanings in another context.

Active Database - The portion of the Available Data Base which has been retrieved from the system disk into the image generator's environment memory by the database management process or model selection. This portion of the model is ready for instantaneous display if it is in range and in the field of view.

Aliasing - The name given to a wide range of undesirable visual artifacts caused by the quantization of the image into pixels. Jagged and/or crawling edges, gaps in thin polygons and a tendency for small polygons to blink on and off are typical examples.

Alpha – A term usually referring to the transparency portion of texture maps, as in RGBA. This is particularly true in systems that use the OpenGL format.

Aitoff Plot – A cartographic projection developed by David Aitoff in 1889. It is often used in simulation to show the full field of view of the display system. One characteristic of an Aitoff plot is that the vertical axis is half the length of the horizontal axis. Cartographic purest will say that this is really a Hammer projection, but for some reason the simulation industry insists on referring to these diagrams (see below) as Aitoffs.



Animation - The process of displaying sequential versions of a dynamic event so that the entire event appears to occur smoothly. The system may display the event once (cued by the host, real-time system and/or instructor) or allow it to cycle until de-selected. Typical examples are an explosion (single cycle) followed by a fire animation (continuous).

Anti-aliasing - Any software and/or hardware implementations intended to reduce or eliminate aliasing. Normally involves super-sampling or multi-sampling the image and averaging the results to determine the content of the pixels. The typical result is a somewhat softer (i.e. blurry) image that is much less objectionable when the eyepoint is dynamic.

Area of Interest (AOI) - A display technology which places the bulk of the visual system's resolution/capacity where the viewer is looking. Other areas will have much lower resolution/capacity or no imagery at all. This approach requires that the system be able to determine where the viewer is looking through head and/or eye tracking techniques and that the image generator is able to change the viewing angle for each field.

Similarly, an area in a database that has higher resolution, fidelity and content than the surrounding areas. Examples are airfields and target areas that the eyepoint will come close to and/or spend a good deal of time around.

Areal Feature - A 2D feature in a database, usually derived from DFAD or VMAP data, which covers an area on the terrain. These are defined as a series of vertices that enclose the area. Typical examples are lakes, forest canopies, and city/urban areas. Database tools can use these vertices do define polygons that will represent the features, and the material codes are used to assign colors and/or textures. The tools will often filter the fidelity of these features, or delete them entirely, in order to produce a database that is within the polygon capacity of the image generator.

Atmospheric Effects - See Visibility.

Available Database - The total database which can be accessed without reloading or re-initializing the visual system. The system's efficiency in retrieving the available database will depend on the overall size of the database components, the database structure and the bandwidth of the system network over which the data must pass.

Backface Culling – The process of quickly determining that a polygon is facing away from the eyepoint, and therefore requires no additional calculations/processing.

Backfaced Polygons - Polygons which are facing away from the viewer/eyepoint and do not need to be processed. Normally deleted from the processing by a quick examination of a vector that is perpendicular to the polygon's visible side.

Beam Penetration Display - A display technology used primarily in night/dusk calligraphic systems. The inner side of the CRT is coated with layers of red and green phosphor. The voltage and duration of the electron beam on a given spot will excite one or both of the phosphors resulting in a light point or raster line which is some combination of red and green.

Bezier Curve – A non-uniform curve based on a function or algorithm. Complex curves are derived from only a few control points. Some modeling tools allow surfaces to be defined by Bezier curves, but the resulting shapes are converted to conventional polygons (normally lots of them) before they can be displayed.

CGI - Computer Generated Imagery. This refers to the actual imagery that is created by the CIG process.

CIG - Computer Image Generation. This refers to the process of creating CGI.

CIGI (Common Image Generator Interface) – A host to IG interface standard created and maintained by Boeing (St. Louis). The purpose is to make the integration of an IG, or multiple IGs, more efficient and less costly. For more information see cigi.sourceforge.net.

Calligraphic Display - A display or projector which can direct the electron beam(s) much more accurately and flexibly than normal raster displays. These additional capabilities imply more control of the final configuration of the image that, in turn, allows calligraphic projectors to work with curved screens. They are used in simulation primarily because the resultant light points are much sharper and brighter than those achieved within the raster structure.

Cartesian Coordinate System – A coordinate system where the two or three axis are perpendicular to each other. Points in the system are defined by measuring along each axis to determine the coordinates (e.g. x,y,z).

Chained Coordinate System - A moving model that uses another moving model as its reference. For example, the cannon on a tank model is chained to the turret (pitch only) which is in turn chained to the body of the tank (heading only).

Channel - A partitioning of IG equipment. Each channel will normally produce a different image. Each window in a simulation application will normally require a separate channel of IG hardware (see counter example under Window). Some systems will provide multiple images, or viewports, from a single channel. In these cases the IG hardware's capacity (polygons and pixels) is shared between the viewports. Alternately, several processing units (e.g. PC-IG nodes) may be combined to create the image for a single channel.

Clipping - The process of creating temporary edges for polygons that extend out of the field of view. A number of algorithms have been developed to accomplish this task as efficiently as possible. Light strings must also be clipped and new end points for the strings must be calculated. The need for clipping will depend on the raster/pixel fill technique used by the image generator.

Clipping Plane – One of the six planes that define the top, bottom, sides, front (near) and back (far) of the field of view (also called the viewing frustum). Scene polygons and lights may be clipped at these boundaries to form temporary edges that are required for the pixel fill process to work properly.

Cloud Layer - A feature that allows the top and bottom of a solid layer of clouds to be specified. The layer will be visible as a solid (normally textured) polygon when seen from above or below. When the eyepoint is inside the cloud layer the system goes to zero or near zero visibility (see Visibility).

Collision Detection - A process available in some image generators that will determine if a collection of defined test points or vectors (normally representing the ownship) have collided with objects in the visual database.

Color Blending - The process of smoothly blending two colors together. Normally a texture process with minimum texel values assigned one color and maximum values another. Also accomplished as a shading process on single polygons by assigning colors to the vertices of the polygon.

Color Palette - A table of color values, normally red, green and blue, which are assigned to polygons and light strings by their location in the table. This implementation allows all polygons or strings of a certain color to be altered by simply changing the values in a single table entry rather than editing each element.

Compiler – A software tool that converts the database source code into the primitives that the image generator will actually process in real-time. Among other things, this process will typically remove any construction elements that the database generation tool(s) may include to aid in the construction of the database and/or models.

Concave Polygon – A polygon which has at least one interior angle greater than 180°. In general, if a system allows polygons with more than three sides/vertices, the polygons cannot be concave (i.e. must be convex).

Coordinate System - A numerical method of describing space. In particular, a system of three axes which all intersect at right angles at a point called the origin (see Cartesian Coordinate System). Any unique point is then defined by its distance (x, y and z) from the origin along each axes. The y axis is normally assumed to point north, x to the east and z vertically although other conventions are also used.

Coplanar – Two or more polygons that are geometrically in the same plane. Painted stripes on a runway surface are typical simulation examples. In a system with the z-buffer priority implementation some mechanism must be employed to determine which polygon(s) should have visual priority (e.g. the stripes should be on top of the runway). If no special coding is available it may be necessary to model the stripes a small distance above the runway surface to avoid confusion in the z-buffer. The distance required will be determined by the resolution of the system's z-buffer.

Database - The term generally applied to the geometrical information that the image generator will process to produce an image. As a minimum this will include polygons that are defined by the position of their corners (see vertices) and some method of specifying color. In more advanced systems the database will be in a hierarchical format and may include (i.e. require) a number of other features such as texture, priority information, shading, etc. Light points, normally in a string format, are also part of the database. Sometime written Data Base, mostly because it makes better acronyms in this form (e.g. CTDB, DBWG, DBGS).

Database Management - The process by which the real-time system in the image generator can bring new portions of the database from the system disk as the eyepoint moves around the gaming area. The new data is taken from the Available Database into the Active Database. This is normally accomplished via some hierarchical structure which allows large portions of the database to be evaluated efficiently. See Level of Detail.

Defense Mapping Agency (DMA) – Previously a government agency headquartered in Fairfax, Va. that was in charge of all DoD mapping related activities. In particular, the DMA office in St. Louis, Mo. produced Digital Radar Land Mass Data that, although intended for radar simulations and predictions, has been used extensively to produce visual databases. Digital Terrain Elevation Data (DTED) and Digital Feature Analysis Data (DFAD), along with some manual enhancement, provide sufficient information to create a viable representation of a specific area. DMA was eventually incorporated into the National Imagery and Mapping Agency (NIMA), which provided the same digital products. The agency is currently a part of the National Geographic-Intelligence Agency (NGA).

Depth Complexity – See **Over-writes.**

Display - A device which takes the output of the image generator and creates an image or scene. Display normally refers to a Cathode Ray Tube (CRT) device, such as a high resolution television monitor, rather than a projector. More recently these are LCD or LCOS monitors or projectors.

Display Database - That portion of the Active Data Base which has passed all of the culling test in the image generator and is actually included in the displayed image. The maximum number of polygons and light points which can be included in the Display Database generally defines the system's simultaneous capacity.

Display System - That portion of a visual system which makes the picture created by the image generator visible to the user. In the simplest case this may only include the display device itself (e.g. a monitor) which the user will view directly. More complex Display Systems may include multiple display devices and optics devices to focus the imagery at the desired distance (e.g. near infinity). Some recent visual systems use head tracked and/or head mounted Display Systems to reduce the overall system complexity while still making a large field of regard available to the viewer.

Distributed Mission Operations (DMO) – Specifically, the US Air Force initiative to do combined training with multiple simulation sites, as well as real-world assets, in a single exercise. In order to facilitate this type of (literally) world wide training, the Air Force has created a DMO Standards group that is in the process of establishing standards for all aspects of simulator training. See www.dmodmt.com to request access.

More generally, DMO (or distributed mission training [DMT]) describes similar efforts by most military services to broaden the effectiveness and usefulness of simulator training.

Double Buffering – The process (and hardware) normally used for smoothly assembling and displaying images. The image is assembled in one side of the double image buffer while the other side is being sent to the display device. The functions are switched for the next frame, and so on.

Dynamic Coordinate System (DCS) - A coordinate system that is processed separately from the Database coordinate system (i.e. the database origin). The position and attitude of a DCS is normally defined relative to the Database coordinate system, but may be defined from the eyepoint or another DCS (i.e. chained coordinate systems).

Edges - A straight line defined by two vertices in the database. Three or four edges (typically) are used to define a polygon or surface which makes up an element of a database. Most current systems define polygons by defining the vertices at the corners (see polygon, vertice).

Emissivity - The relative power of a surface to emit heat by radiation. In particular, when simulating an infrared sensor, what must be modeled instead of color and intensity. The simulation is complex because the relative emissivity of surfaces (i.e. polygons) will change due to a number of factors such as air temperature, temperature history, humidity, sun angle and exposure, etc. In more recent systems it may be necessary to encode emissivity, among other things, into the texels that make up the geo-specific texture maps for the database terrain.

Exit Pupil - In a Display System that creates an infinity image the Exit Pupil is the area in which the viewer will see an undistorted scene. The Display System interface to the simulator should be designed such that the Exit Pupil coincides with and includes the typical location(s) of all viewers (e.g. the pilot's design eyepoint(s)). Sometimes referred to as the viewing volume.

Extrapolation - In image generators which operate asynchronously with the host computer for some or all of the control data an extrapolation must be performed to correct the latest eyepoint or moving model data (location and attitude) so that the position used for the visual calculations is as accurate as possible. See section 2.1.2.

Eyepoint - The point in space from which the image generator calculates its image(s). Some complex simulations will require more than one eyepoint. This may be very impactive on the IG depending on the distance between the eyepoints.

The simulator (and the vehicle being simulated) will have one or more design eyepoints, which is the physical location where the viewer's head is expected to be. The display system should be designed to provide the optimum image at this point.

The eyepoint of the simulator/vehicle will typically be some distance from the center of gravity of that vehicle. This is particularly true for aircraft where the center of gravity and rotation is back by the wings, and not in the cockpit. In order for the correct terms of movement and rotation to be provided to the image generator the simulator’s motion terms will have to be adjusted from the center of gravity to the design eyepoint.

FAA Level C and D (Phase II and III) - Simulator approval categories published by the Federal Aviation Administration. Phase II (now Level C) allows the airline to perform recurrent training, upgrade (co-pilot to Captain) and conversion (727 to DC-10) entirely in the simulator. Level D (Phase III) allows the airline to train a newly hired pilot in the simulator with no actual aircraft time prior to his first line flight. It should be understood that there is a great deal more involved in these approvals than the visual system. The simulator and the training syllabus receive the bulk of the scrutiny.

Visual systems are often required to meet Level C or D performance by organizations which have no intention (or need) to apply for the actual approval. These include foreign airlines (non-FAA) which may have similar requirements from their local authorities, and some military applications, particularly transport type aircraft. In these cases the FAA regulations are used as readily available (and accredited) performance measures.

Far Clipping Plane – Also called the back or yon clipping plane. See Clipping Planes

Field - Usually intended as a measure of time. The time required for the display device in an interlaced system to draw half of the raster lines. Two fields together (the odd and even rasters) is called a frame. Some image generators will calculate a new image for each field and are said to run at field rate. The field time is therefor the IG cycle time as well as the raster display time. Others IGs will display the same image for both raster fields while calculating the next image. These are said to run at frame rate.

Field Extend - Some image generators that run at field rate will extend the length of the field when an overload is encountered. This gives the IG some additional time to finish processing all of the polygons or fill all of the pixels (depending on the nature of the overload). During subsequent fields the overload management will attempt to reduce the load so that the normal field time can be resumed. The display device must be able to handle these variations in field time in order for this technique to be viable.

Field of Regard - In applications where the viewing direction of a channel can be rotated the field of regard is the horizontal and vertical limits to which it can be moved. This typically applies to sensor channels or the entire visual system in an area of interest display system (i.e. head and/or eye tracked). In some virtual image (i.e. infinity focus) display systems the viewer is able to see additional imagery by moving his eyes within the exit pupil. In these cases the field of regard is slightly larger than the field of view.

Field of View (FOV) - The area of the image produced by the visual system, normally expressed as horizontal and vertical angles (e.g. 180° x 50°). The actual FOV results from the display system, but must be carefully coordinated with the image size being calculated by the image generator.

Instantaneous - The instantaneous FOV is that which is available to the viewer without moving his head. This may assume that the eyepoint is in the center of the exit pupil.

Total - The total FOV is what can be achieved by the viewer moving his head to expose more image. This may involve moving the eyepoint to the extremes of the exit pupil. Any field of view achieved by moving the eyepoint outside of the exit pupil is normally disregarded because of the resultant distortions. Referred to as field of regard in some display configurations.

Field Rate - See Field

Field Tracking - The tendency of the eye to lock onto one raster field in an interlaced system in such a way that the two fields appear to overlay each other. See section 4.1.2.

Fixed Shading - A shading technique which allows a different intensity to be defined for each vertice of a polygon. The IG will then smoothly interpolate the intensity across the polygon. See section 5.3.2.

Flicker - A distracting variation in brightness caused by the refresh rate of the display system being too slow for the desired intensity level. Night/dusk system, for example, can refresh at 30 hz. with little or no flicker due to the low intensity expected in these modes. A full day system will require a minimum of 50 hz., with 60 hz. being preferred, in order to eliminate perceived flicker. One complication with determining or measuring flicker is that some people are more sensitive to it than others so that the refresh rate where it becomes noticeable or objectionable cannot be uniformly quantified. Flicker is also aggravated by the size of bright objects in the scene. This means that a system may be able to display a very bright object if it is small enough (i.e. highlight brightness) where a larger object or the whole screen filled with that brightness would definitely flicker. Flicker is also more detectable in a person's peripheral vision. This means that display systems with large fields of view will exhibit flicker at lower brightness levels than smaller systems. Flicker is a suspected cause, or contributor, to simulator sickness.

Fog - See Visibility.

Fractals – A class of mathematical functions that approximate (at least visually) the shapes and patterns found in nature. These are sometimes used to create higher levels of detail (i.e. more polygons) than the source data may support. For example, if terrain of higher fidelity than provided by DTED is required a fractal function can be used to generate additional natural looking polygons based on the DTED elevation data. The resulting terrain will provide higher density, and associated cues, but will not be accurate to the real-world. If additional detail (and levels of detail) is desired, the function can be iterated as many times as required. This process generates a very large number of polygons with just a few iterations.

Frame Buffer - A hardware storage area in an image generator where pixel color and intensity data for an image is assembled prior to being sent to the display device.

Frame Rate - See Field.

Frustum – See Viewing Frustum.

Gaming Area - The term used to describe the total geographical area over which a simulator can operate. This is often the same area as the database, but may be larger. In some applications there may not be dedicated database to cover the entire gaming area. Specific areas of interest such as airports or targets may have been modeled while the system's generic ground plane fills in the areas in between. Commercial airline simulators are the ultimate example of this in that the gaming area can cover much of the globe, but only the airports are normally modeled. In some complex full system applications, the visual database may cover a smaller geographic area than the radar gaming area.

Gamma Correction – The phosphor on a CRT does not respond linearly to the voltage from the electron gun. Gamma correction provides controls to correct the linearity. This is particularly important when the output of the image generator may be viewed on more than one type of display device (e.g. monitors and projectors).

Generic Database – A database (or gaming area) which represents an imaginary location that has been designed for some specific purpose. In training applications this may be done in order to compress a number of areas of interest (e.g. airfields, target areas) into a smaller geographic area than may be available in the real-world. This can make training more efficient, and can save money and effort in creating the database. These individual areas may be real-world features that have been repositioned for convenience, or they may also be artificial to enable a specific training maneuver (e.g. a non existent runway geometry).

This approach also provides some opportunities to maximize the throughput of the image generator in that the arrangement of the database elements can be adjusted to even out the processing load on the system. This, in turn, may allow a lower cost image generator to be utilized. This same approach is often used in entertainment applications (i.e. video games) for similar reasons.

Geocentric Coordinate System - A coordinate which uses the center of the earth as the reference datum.

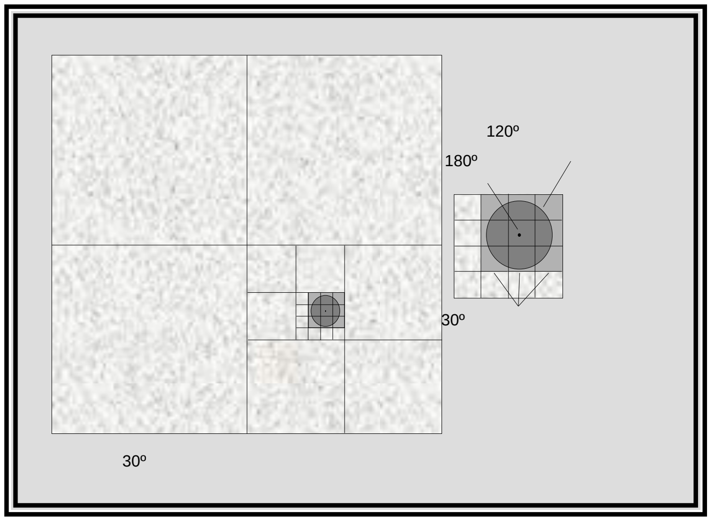
Geographic Information System (GIS) – A type of computer graphics (usually not real-time) that makes use of databases similar to simulation. In general the databases will include several types of information in addition to the geographic terrain. These may include demographic data and other infrastructure elements (e.g. power lines, sewer systems, etc.) organized in layers so that they may be viewed together or separately. GIS databases are sometimes considered as a starting point, or source data, for the generation of simulation databases of the same area. In general, these databases have much more data than a real-time system can handle, or require, but they may provide a good starting point.

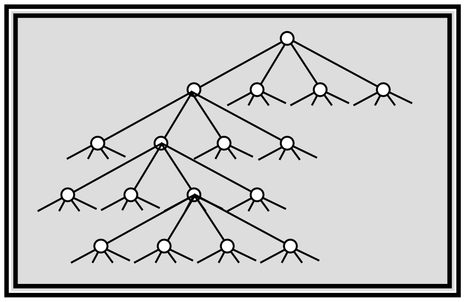
Geometric Processor (GP) - The portion of image generator hardware which converts the three dimensional database information (i.e. geometry) into perspective imagery. A number of culling operations are also performed in order to optimize the throughput of the GP. This component may be referred to by other names depending on the manufacturer.

Ground Fog - A meteorological condition in which a layer of fog a few hundred feet thick will cover the ground in otherwise good visibility conditions. This is an aviation problem during landings because the ground can be seen clearly from above the layer, but vision is dangerously obscured once the layer is entered. See Visibility.

Height Above Terrain (HAT) - The capability of some image generators to calculate the height of a test point above the polygon directly beneath it. See section 4.5.

Hierarchical - Normally refers to a database structure in which large portions of the database can be tested for management or priority purposes with a single operation. This is typically pictured as a tree structure in which a negative test of a branch will cause the processor to ignore the entire branch. A positive test, however, will cause the processor to continue down the limb to examine subsequent branches.





Host Computer - The term used to refer to the computer(s) which drives the simulator to which a visual system is interfaced. This computer is normally larger and sometimes slower than the visual system computer. In some applications the host functions are contained in the same CPU with the visual processes.

Host Interface - The hardware that connects the visual system computer to the host computer. Originally this was a one way interface with the host supplying position data which the IG used to calculate images. Most current systems use a two way Ethernet interface or something similar.

Image Generator (IG) - The generic term that refers to the collection of hardware and software used for creating computer generated imagery. At one time the term implied a significant amount of special purpose hardware and real-time operation. Currently the hardware in most IGs is made up of one or more personal computers with advanced graphics processing units. The are typically referred to as PC-IGs.

IMAGE Society – A group dedicated to furthering the art of visual simulation. Primarily specializing in military simulation when founded in 1977, the group has expanded to include special interest groups in virtual reality and medical imaging. The Society holds annual meeting in Scottsdale, AZ in June (yes, it's hot!). For details see www.image-society.org.

Interlace - The characteristic of some display devices to draw half of the raster lines during one field followed by the other half interleaved between the first set during the next field. This is referred to as the odd and even field. The eye and brain tend to "merge" these into a complete image. Originally all displays were of this type including home television sets. Most current displays can provide a full screen of raster lines for each refresh of the image (i.e. non-interlaced or progressive scan).

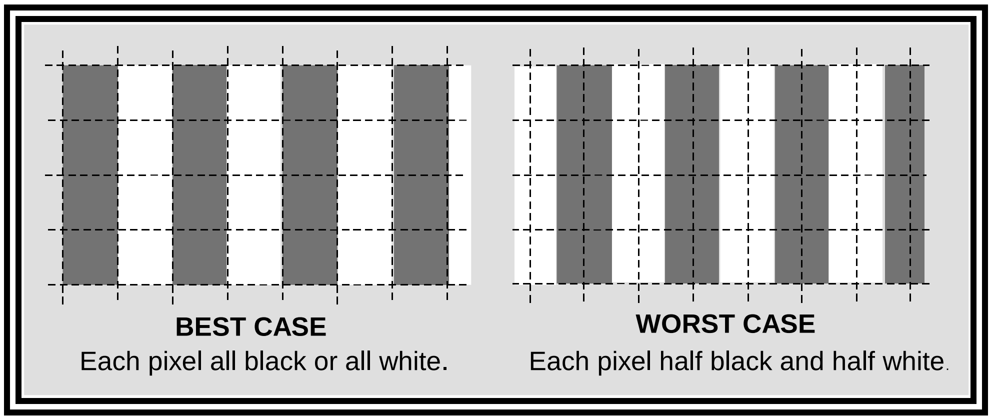
Instancing - The capability of some IGs to display multiple copies of an object which is stored on the disk only once. The database will include a reference to the object along with a location and perhaps rotation and scaling. The IG run-time software will distribute the feature within the scene based on the position data in the database.

IITSEC – Interservice/Industry Training, Simulation and Education Conference. This is the simulation industry's primary annual conference and exhibit. A large number of suppliers to the military simulation market will have equipment there to demonstrate. There are also technical papers in a number of disciplines, including technical and management areas. The conference is usually held in the week or two after Thanksgiving, in Orlando, FL. For details see www.iitsec.org.

ITEC – International Training and Education Conference. Similar to IITSEC, but held in Europe in the April time frame. For details see www.itec.co.uk.

Kell Factor - An attempt to statistically quantify the effect of pixel alignment variations when measuring display resolution. It should be possible for the display to draw black and white bars that are each the same size as a pixel. This can theoretically occur if the bars and the pixel samples are aligned correctly as shown below.

In the best case the pixel sample points fall in the center of each bar and the pixels will be fully black or white (i.e. 100% modulation). In the worst case the sample points fall on the edge of the bars and each pixel is half black and half white which will result in all of them being gray (i.e. 0% modulation). There are clearly a wide range of cases in between. If the bars are made wider the situation improves until there is always a measurable modulation (10% or more). The Kell factor (e.g. .7) is applied to the measured line size to approximate the actual pixel resolution.


BEST CASE
Each pixel all black or all white.
WORST CASE
Each pixel half black and half white.

Latency - Defined (by the FAA at least) as the portion of overall transport delay which is in excess of delays in the actual vehicle being simulated. Any attempt to measure or specify latency assumes that the actual vehicle dynamics are well understood and measured. See Transport Delay

Level of Detail - A database management process in which the number of polygons and/or light points used to represent a scene feature is increased as the range to the feature is reduced. For example, an airport terminal building can be represented with a few polygons when seen from several miles. As the eyepoint approaches additional polygons will be required to establish a more complex shape. These additional polygons may replace the lower level of detail or they may be added to the original shape. As the eyepoint gets still closer windows and other markings will be added. This all assumes that the image generator has the means to calculate the range to the feature and manipulate the database accordingly. Some systems will use their transparency feature to fade new levels of detail in and old ones out. This is referred to as Fade Level of Detail (FLOD).

One consideration that should be examined is the processing power required to determine the distance to an object and then to swap out the versions of the model. It may be more efficient for the IG to process the larger number of polygons rather than process the LOD calculations. This is particularly true of current graphics boards.

Light Point - The CGI representation of a light in the real world. The appearance of a light will depend on its brightness as modified by distance, visibility and directionality rather than perspective size (as with polygons). See section 5.4.

Light String - A data format for efficiently storing a large number of light points. Strings were originally straight with the first point, last point and number of lights being defined. Curved strings were then added to some systems with a third point (on the desired curve) being included in the definition. Some systems now include random strings (a contradiction of terms?) in which a horizontal area and a number of lights to scatter randomly within the area are defined. The string definition will also include color, brightness, directionality and other light point characteristics that apply to all of the lights in the string.

Linear Feature - A database feature, usually derived from DFAD or VMAP data, which is defined as a series of connected line segments. Examples include roads and railroads, powerlines, and coast lines.

The database tools will use this data to create polygons (e.g. roads) of the correct width, as well as assign color and/or texture.

List Priority - An approach to polygon priority in which the first polygon or light string listed has the highest visual priority followed in turn by all of the subsequent polygons and light stings. For anything other than a trivial example this approach will require some dynamic process to shuffle the polygons and strings into the proper order for the current eyepoint position.

Model - A term originally used interchangeably with database. More recently, the term is applied to individual 3D features in the database, either moving or static. Database Design Engineers are sometimes referred to as Modelers.

Mosaic – In database terms, the process of fitting a number of small polygons together to avoid layering one polygon over another. Polygons for the white stripes on a runway, for example, can be layered on top of a single gray polygon representing the runway, or the runway can be broken up into a large number of gray polygons that fit around the white stripe polygons. While this requires significantly more polygon processing, the benefit is in only filling the pixels once. This may be more efficient overall depending on the system architecture. The same process might be used to break a large polygon (like the runway) into smaller polygons for visibility and/or shading purposes. This is done in systems that use other than pixel fog implementations (e.g., vertice based fogging).

Moving Model – See **Dynamic Coordinate System (DCS)**

National Geospatial-Intelligence Agency (NGA) - The U.S. government agency in charge of maps and imagery used for database generation purposes. NGA is an amalgamation of a number of previous agencies, including DMA (Defense Mapping Agency), which itself had evolved into the National Imaging and Mapping Agency (NIMA). NGA is the provider of DTED and VMAP data as well as other products that can be used in the development of visual and sensor databases (e.g. satellite imagery).

NIMA (National Imaging and Mapping Agency) – See NGA above.

Non-Linear Image Mapping (NLIM) - The process of pre-distorting the image so that it will project correctly on a curved screen.

Normal Vector – A vector that is perpendicular to the plane of a polygon. This vector is normally included with the polygon data (as opposed to computing it on-line) and is used for backface culling and shading calculations among other things.

NURB – Non-Uniform Rational B-Spline - A method for defining complex shapes with math functions or algorithms. Similar to Bezier curves, NURBs are an efficient way of defining and storing a complex shape, but must be converted to conventional polygons before they can be displayed in real-time by a traditional image generator. Generally not used in simulation because the resulting run-time models are too polygon intensive.

Object - A typical division of database elements. Usually a specific number of polygons and/or light strings grouped together for priority, management or instancing purposes. A simple building may be included in a single object. A complex model such as an airplane could require several objects, one for each wing, one for the tail, etc.

Occultation - One polygon covering another. Some systems are limited in the number of layers or levels of occulting. These limits will be closely related to the priority mechanism in the system.

On-line Database - Another term for the Available Database. The database which can be accessed through the normal operation of the simulator (as opposed to re-loading software or disk packs).

Origin – The location in a Cartesian coordinate system where the axis intersect, and measurements begin. The (0,0,0) point in a typical simulation database.

OTW – Out the Window – The collective term used to describe all the imagery displayed outside the aircraft or vehicle windows, usually as differentiated from sensor imagery (e.g., IR, radar).

Overload - A condition that occurs when an image generator is forced to process more than it is designed for. This can be too many polygons and/or light points, too many pixels, too many dynamic coordinate systems, too much transparency, too much occulting, etc. Overload is somewhat peculiar to real-time CIG devices that are designed to operate at a specific rate. It is ultimately this restriction in processing time that makes overload a problem.

Overload Management - Any one of a number of mechanisms that are intended to overcome the effects of system overload. These will generally make use of the database management process to reduce the number of polygons and lights in the scene. Some implementations can also reduce and re-map the number of pixels being processed in order to compensate for pixel fill overload. See section 3.2.3.

Over-writes – A term that normally refers to the pixel fill process. In any normal 2D perspective image that is formed from a 3D database, it will be necessary for some foreground objects to occult, or partially occult, objects that are further away. The image generator must process all of these objects into pixels in order to determine their relative placement and contribution to the final image. Pixel over-write is a measure of the IG's capability to process, depth test and fill the required number of pixels for the scene being drawn. The number of pixel over-writes required will differ with the application. The number of pixel over-writes available will depend on the priority architecture of the image generator and the number/performance of the display processors utilized (in a modular system). Also referred to as Depth Complexity, Pixel Fill, or Pixel Hits.

Ownship - Referring to the vehicle being simulated. Also refers to the eyepoint from which the image is calculated.

Phong Shading – See shading.

Pilot Induced Oscillation (PIO) - A phenomenon that sometimes occurs when flying a simulator with too much transport delay. A correction is entered (e.g. bank angle) and left in until it appears (in the visual image) to be effective. Due to the long transport delay the correction is actually left in too long and a counter correction is required to alleviate the new problem. This is also applied late, and is also left in too long, and so on. The result is constant counter inputs to correct a possibly diverging condition.

Pixel - A shortened form of "picture element". This is the smallest unit along a raster line for which a display device can change color and/or intensity. Somewhat like the black dots in a magnified newspaper photograph.

Pixel Fill – See **Over-writes**.

Point Feature - An object, usually derived from VMAP, whose location is defined by a single point. Examples include radio masts, some buildings, and airfields. The database tools use this information to assign a 3D model, from a feature model library, and place it on the terrain.

Polygon - The basic element of most modern image generators. A two dimensional shape with straight sides, normally single sided, and defined by the location of its corners (vertices). Viable polygons are usually limited to convex shapes. System architecture may limit the number of polygon sides. Three or four is common. Some systems will allow many more vertices in the construction, but will break the resulting complex polygon into triangles internally during the compile process. Polygon capacity is usually calculated for three or four sided polygons.

Prediction Algorithm - A series of calculations intended to negate, or at least limit, the effects of transport delay. Various acceleration factors are used in an attempt to predict where the simulated vehicle will be when the image is actually drawn. The success of this process greatly depends on the nature of the vehicle's dynamics.

Priority - The process required in all CIG systems which determines the visual priority of features in the scene. Unlike the real world where occulting, and the resultant visual priority, occurs as a natural result of one thing being in front of another, an image generator must have some way of determining that a polygon should be drawn if front of or behind other polygons. One of two methods is normally used; a list priority approach or a Z-buffer implementation. See section 3.4.

Quad-tree - A database structure in which a geographic area is successively divided into quadrants. For database management purposes each quadrant can be tested against the current location of the eyepoint to determine if that portion of the database must be processed for viewing. The eyepoint is not in (or near) the quadrant, that entire portion of the database can be ignored. If the eyepoint is close enough to warrant further examination, the test is repeated for the four sub-quadrants of that quadrant. See diagram under **Hierarchical** and section 3.2.2.

Range Buffer - A priority mechanism similar to a Z buffer. Rather than use the Z depth of the pixel content to determine priority the actual range is calculated. The main advantage of the R buffer is for very large fields of view which approach 180^o per channel. At the edge of these channels a Z buffer may run out of resolution.

Raster - The media with which a video display or projector creates its image. The picture is drawn by the electron gun(s) being swept across the inside of the cathode ray tube (usually horizontally) where it excites the phosphor to create color and intensity. The speed with which the beam(s) can be modulated to change the color or intensity will determine the number of useable pixels available on each raster line (the IG may provide more). Many display devices use an interlace technique which will only draw half of the raster lines at a time (see interlace) depending on the persistence of the eye to "assemble" the image.

Ray Tracing - A simple, but computation intensive, technique for determining shading in a CIG scene. The light that illuminates each pixel is traced back to its source. This will be an illumination source (e.g. the sun), a reflection from another pixel, or the ambient illumination. This process results in very natural looking scenes with highlight reflections and shadows. To date no real-time image generator can manage the massive number of calculations true ray tracing requires, but there are a number of "emulations" that resemble ray tracing for very limited applications.

Real Image – A displayed image that is focused on the display surface, such as a rear projection screen, as opposed to a virtual image which is focused somewhere other than the screen (usually at a distance of approximately 50 feet which the eye equates to infinity).

Real-time - A much abused term intended to indicate that a CIG device cycles at a rapid and <u>stable</u> rate. A cursory look at transport delay issues will make it clear that no CIG system can possibly run in real-time in the pure sense. For the visual system the term has come to indicate an update rate that is fast enough to eliminate image stepping and display flicker. This is typically 30 updates per second for night/dusk systems and a minimum of 50 per second for day systems. It is also important that the update rate be stable. This is a function of system overload, but may also be related to the operating system under which the simulation is running. Some operating systems are designed to include periodic administration or utility tasks (e.g. keyboard interrupts, disk accessing) that will cause an objectionable pause in the normally fast update rate.

In more traditional software terms, a specific type of operating system that allows interrupts to maintain a consistent schedule of routines and/or events.

Refresh Rate - The rate at which the image is re-drawn on the display device. This is often the same as the update rate, but some systems will draw the same image more than once, giving the image generator more time to calculate the next scene.

Resolution - There are a number of resolution issues when dealing with visual systems. The three most common are:

Positional - This relates to how accurately the geometric calculations are performed including the position of the eyepoint and/or objects in the database. Some early systems were limited to 16 bits of resolution in this area which lead to either small database sizes or course movement depending on where the decimal point was placed. Most current systems use at least 32 bits and provide resolution down to 1/32 of a foot and up to 400 miles.

Display - This is basically a measure of pixel size as displayed to the viewer. It depends on the characteristics of the display device (particularly rear projection screens), the focus of any optics, the ability of the image generator to produce the appropriate number of pixels and the anti-aliasing techniques employed. There are also subjective and objective measures of display resolution as discussed in section 3.3.

Texture – Texture resolution relates to the size of the texels in database terms. A high resolution geo-specific texture derived from aerial photography may be one meter resolution. This means that each texel covers approximately one square meter on the ground. It also means that the source imagery was one meter resolution and that the smallest objects discernable in the image are approximately that size.

Scan Line - See Raster Line.

Scintillation - See Aliasing.

SEDRIS – Synthetic Environment Data Representation and Interchange Specification. For more details see www.sedris.org

Separating Plane - A mechanism used to resolve dynamic priority issues in a list priority system. The separating plane is included in the database but is invisible to the viewer. Objects, or groups of objects, on either side of the plane are referenced in such a way that the system can determine which is closer to the eyepoint by calculating where the eye is relative to the plane. See section 3.4.2.

Shaders – Vertex shaders and fragment or pixel shaders are small programs that are stored in the graphics processors of PC based systems. These programs are used to create specific rendering effects at the vertex or pixel level. These are somewhat new to the visual simulation industry and their impact and performance have yet to be fully comprehended.

Shading - Allowing illumination from a defined light source, usually the simulated sun, to effect the intensity of polygons by comparing the orientation of the polygon (i.e. a vector normal to the polygon) with the illumination angle. The less perpendicular the polygon to the "sun" the lower its intensity. There is normally an ambient illumination which serves as the minimum intensity rather than let the intensity go to black for polygons facing away from the source. Several different types of shading are given below. See section 5.3.2.

Fixed - A shading technique where an intensity value is defined for each of a polygon's vertices. This form of shading does not depend on the polygons orientation to the "sun".

Flat - The entire polygon receives the same intensity such that a flat shaded object appears to be constructed from facets (as it is).

Gouraud - Often called smooth shading. The normal vectors for two or more polygons are averaged so that each shared vertice has its own local normal. During processing of the polygons the intensity for each pixel is derived from the interpolation of the vertice normals. The surface of the object can appear curved even though it is made up of flat polygons.

Phong - Provides glint and highlights by processing the shading solution at the pixel level. Each pixel’s relationship to the illumination source(s) is calculated to determine the color/intensity of the pixel. For example, small areas of a polygon can show significant intensity changes which are perceived as glint from the surface. The polygons may require some definition of shininess in order for Phong shading to be effective. This capability is just now becoming available in real-time systems.

Smooth - See Gouraud.

Sun - A general term used to indicate a shading capability. Normally implies flat shading.

Shapefile – A digital vector storage format for storing geometric location and associated attribute information. Originally developed by ESRI for their geographic information systems (e.g. ArcView). It has been widely adopted into visual simulation as one of the basic formats for defining geographic and cultural features of all kinds.

SIF – Standard Simulator Database Interchange Format. This is the database format that resulted from the Air Force Project 2851, and defined by Mil-std 1821. SIF has pretty much disappeared as a database standard due to the difficulties and limitations associated with its use. SIF was replaced by SEDRIS, which attempted to maintain the data in more abstract terms that would allow more flexible interpretation by each system using the information.

SIGGRAPH – Officially the Association of Computing Machine's (ACM) Special Interest Group on Computer Graphics. SIGGRAPH, the group, has a great deal to offer in the way of education and services, including publications and events. When most people say SIGGRAPH, however, they mean the annual conference held somewhere in the U.S., normally in the August time frame. For more details see www.siggraph.org/home.html.

Special Effects - A generic term associated with any effects which require more than the normal polygon and light point processing. These often involve dynamic coordinate systems, animation and/or moving texture. Weapons effects (explosions) are a good example. More recently effects of this type are implemented with one or more shader effects. These provide much more interesting and correct visual simulations with much less processing power.

Sub-pixel - A term utilized in anti-aliasing techniques which refers to samples which are smaller than a pixel. Some number of sub-pixel values are typically averaged together to produce the actual pixel value to be displayed. See section 4.1.

Surface - While this can refer to the surface of an object, particularly in the context of shading or texture, it may also used to indicate a polygon. This is left over from the early edge based systems, some of which referred to surfaces or faces rather than polygons.

Synthetic Environment – A term that describes not only the visual and sensor databases, but all other aspects of the training environment. This includes the weather simulation (i.e. the Synthetic Natural Environment) and any automated forces or threat systems. The typical inference is that all simulations contributing to the synthetic environment must be correlated and coordinated.

Tessellate - Normally relates to texture maps, but also to some types of generic databases. For a texture map to fully tessellate its boundaries must not be obvious when the pattern is repeated many times in all directions. Some maps are designed to tessellate in one direction (e.g. horizontally), but not in the other. A tree-wall texture, for example.

Texel - Short for texture element. A single cell in a texture map array (similar to pixel).

Texture - Patterned modulation of a polygon's intensity, color or transparency.

Geo-specific - Texture maps derived from satellite and aerial photographs and used to simulate the area photographed. These maps do not tessellate, but are made to match adjoining maps such that the joins are not visible.

Geo-typical – Texture maps, possibly derived from satellite or aerial photographs, which are processed so that they will tessellate with themselves. The map is then used to represent any area in the database that is "typical" of that map. Examples include forest canopy, farm fields, city/urban areas, and desert. Texture of this type can be efficiently applied by database tools based on the culture contend of available digital data (e.g. DFAD or VMAP).

Photo - Any texture map derived from a photograph, normally via a scanning digitizer of some sort. Texture of this sort is often used on models (e.g. aircraft, buildings). The term may also be intended to imply a full color texture as opposed to a single modulation.

Bump Maps - Bump texture maps contain bend angles that are applied to the polygon's normal vector. These altered normals are then used in the illumination calculations to shade the polygon. The benefit is that as the viewing angle and/or the illumination angle/intensity changes, the appearance of the polygon will change as if the polygon surface contained 3D relief. The is particularly useful for water surfaces. Also adds depth to photo textured polygons. True bump mapping is usually associated with Phong shading, and makes use of the same pixel level intensity calculations. Like ray tracing, there are a number of emulations of bump mapping that provide effective results for very limited applications.

Transparency - The capability to display all or part (via texture) of a polygon as translucent (a truly transparent polygon is of little value). There are normally degrees of transparency which can be defined in the database. The color of the translucent polygon's pixels is mixed with those of polygons behind it in the scene. Also used to make level of detail changes less noticeable by fading new objects into the scene.

Transport Delay - The total time from a control input until the simulator/visual system responds (i.e. moves). This is a very complex issue with many contributing factors. The delay time is kept as short as possible, but not less than that of the actual vehicle. See Latency.

Update Rate - The rate at which a new image is created by the image generator. May or may not be the same as the display refresh rate.

Vertice - A zero volume point in a database defined by an X,Y,Z position relative to the coordinate system origin. Virtually everything in a database is positioned and/or defined by one or more vertices. Also called a vertex.

View Frustum – The six sided volume in which the elements of the active scene are contained. The sides of the view frustum are the clipping planes. Depending on the architecture the system my construct temporary edges/vertices for polygons that penetrate these planes. In addition to the top, bottom and sides, the near and far clipping planes are included in the six.

Viewport - The image calculated by an IG channel. Each viewport will be defined by a look angle and a field of view. Within the IG hardware the viewport will also be allocated a number of pixels. Some systems are modularized in such a way that a single hardware channel can provide more than one viewport. In these configurations the viewports will share the polygon and pixel capacity of that IG channel. While cost effective, these arrangements raise several database design issues.

Viewing Volume - See Exit Pupil.

Virtual Image – An image with a focal distance different than the display surface. Typical of collimated display devices that simulate an infinity focus. This is accomplished by the use of curved mirrors or lenses which transmit and/or reflect the image as parallel light rays. The eye interprets these rays as coming from a great distance (typically around 50 feet which the eye equates to infinity). Care must be taken that the rays do not diverge as this causes significant distress to the viewer (i.e. headaches, sim-sickness). A virtual image is often used for aircraft simulations where the typical scene components are 50 feet or more from the viewer. If a simulated object comes closer to the eye than the focal distance, as during some in-flight refueling training, the display will exhibit some interesting optical illusions. The alternative is a real image (see **Real Image**) where the scene is focused at the screen surface, such as a rear projection display or a normal monitor.

Visibility - The process of simulating visibility restrictions in a CIG system. As with priority, there is no natural attenuation of polygons and lights simply because they are further away. The system must calculate the amount of fog color to add to each polygon and/or pixel. Several variations are given below.

Fog - The generic term for visibility effects in a CIG system. The actual simulation may be of dust or smoke depending on the application.

Ground Fog - Defined in the image generator as a second visibility or RVR term. When the ownship is above the ground fog layer the scene is displayed at the normal visibility. As the ground is approached (during landing) the system smoothly reduces the effective visibility to that defined for the ground fog layer. There may be controls for the height and abruptness of the ground fog onset.

Haze - Another term for visibility. Sometimes is offered in place of fog, sometimes in addition. May refer to the effect near the horizon as seen from altitude on a relatively clear day.

RVR - Runway Visual Range. A measure of visibility used in aviation to indicate the range at which the runway lights can be seen. It is normally used only when the observed visibility is a mile or less and depends on the intensity setting of the runway and approach lights.

Scud - A variation in visibility when entering or leaving a cloud layer or ground fog layer. This is intended to simulate a ragged top or bottom to such a layer. There may be controls for the extent and degree of the scud effect.

Visual System - Often used to indicate an image generator, but actually refers to all of the components that work together to provide effective visual imagery to a simulator operator. These include the display devices and optics, databases, host integration and a number of support issues in addition to the image generator hardware itself.

Voxel – A volume element. This is a construct used in some systems (none of them real-time as yet) which build up the image content from solid or volumetric "pixels". This technique is appropriate when the inside of an object is important, and will be viewed. One application for voxels is medical imaging where objects might be cut open so that the interior is exposed.

Window - Usually refers to an actual window in the simulated vehicle for which imagery is being supplied. Is not, however, the same as an IG channel. Some applications will repeat the imagery from an IG channel in more than one window. A typical airline system with mirror and beam splitter optics (as opposed to a wide angle display system) will use a single channel to provide both the pilot and co-pilot view. It is felt that the small difference between the two eyepoint locations is not significant relative to the typical viewing distance. This configuration is normally referred to as a three channel/four window system (the pilot and co-pilots respective side windows being the other two).

Wireframe – A way of viewing a scene or model so that only the edges of the polygons are displayed. This is typically a debug technique used in the database tools to evaluate the shape of polygons without any color or texture content. It may also be used in the image generator to evaluate suspected fill rate problems. Since the polygons are not "filled" in wireframe mode there is little display processor effort required. Most or all of the geometry must still be performed, however, so overload that still persist in wireframe is probably related to the number of polygons rather than the number of pixel over-writes.

Z Buffer - Normally refers to a priority technique which compares the Z depth of a polygon (at the pixel level) to those which have already been processed. Sometimes called a depth buffer. In hardware terms, the actual memory buffer where these values are stored. Z-buffer resolution is also a factor. Depending on the hardware applied this may be 16, 24 or 32 bits. In lower resolution systems the ability to differentiate one level from another may not be sufficient to prevent priority flip flops (at the pixel level) when the perspective distance between two polygons is small. This can be "forced" by placing the polygons further apart than they actually are, particular in the lower levels of detail seen from a distance (see Z-Offset). See section 3.4.1.

Z Depth - The Z depth is a measure of the distance from the eyepoint to a position (usually a pixel) in the displayed scene. It is not the range to the polygon or pixel, but the Z component of the range.

Z-Offset – A technique of ensuring that a system's Z-buffer will provide the correct priority solution. Depending on the resolution of the Z-buffer, it may be necessary to offset two polygons that are really coplanar (as in stripes on a runway, or windows on the side of a building). If the system does not have some method of defining relative priority it will be necessary to create an artificial offset so that the correct image will result. In some advanced systems this is done automatically during run time if the polygons are identified in the database as coplanar. In less complex systems it may be necessary to build the offset into the database (i.e., floating the stripes above the runway). In these cases is may be further necessary to increase the offset in the lower levels of detail of the object or feature to account for the perspective distance.

SOURCES CITED:

Jenn, Adriav, "Faster Polygons!" 3Dgate March 2000: 16-24.

Ferguson, R. Stuart. Practical Algorithms for 3D Computer Graphics Boston: AK Peters LTD, 2001

MultiGen-Pradigm. Creating Models for Simulation San Jose: Paradigm Associates, 2000