Architectures of Image Generators for Flight Simulators

Carl Mueller

Department of Computer Science

UNC Chapel Hill

February, 1995

Architectures of Image Generators for Flight Simulators

Carl Mueller
Department of Computer Science
UNC Chapel Hill

February, 1995

1. Introduction

This paper represents the results of a general investigation into the architectures of image generators for flight simulators. The focus of this research was techniques for real-time image generation, and thus the elements of the image generators most directly related to this topic will be concentrated on, while other issues (such as display systems) will not receive as much attention.

The term "flight simulator" is actually too specific for the range of systems studied here. While initially developed to support flight training, computer image generation systems have been used to support many kinds of training and simulation tasks. Military tactical training for ground-based vehicles (i.e., tanks) is a major example. Unless stated otherwise, the term "IG" will be used to refer generically to image generators developed for simulation and training purposes. While references to flight simulation will still be made, it should be understood that such remarks usually apply as well to other kinds of simulation training.

As the author came into this study with a background in general purpose graphics computers and real-time rendering, the paper was written for a reader who is also somewhat familiar with these areas. Given this, we further focus the topic of this paper on the differences between IGs and general purpose graphics systems, zooming in on the techniques which are more unique to the IG industry.

1.1. Investigating the Topic

The business of commercial flight simulator products is highly competitive, and most companies are quite secretive about how their systems work. This unfortunately leads to a drought of solid information about the various systems. However, there are still a few good sources of information to be found. The companies themselves typically do have some "technical overviews" about their systems, which may sometimes reveal a fair amount of information if one reads between the lines. Also, there are a few regular conferences where sharing of technical information takes place. These include the Image Society conference, I/ITSEC, ITEC, and AIAA. Please refer to the appendix for information on these and other sources.

1.2. The Nature of FS IGs

The application of flight simulation is fairly focused. The goal is to train a pilot for performing various flying tasks. These may include flying at night or during weather conditions offering poor visibility. Military trainers might emphasize special maneuvers such as tactical combat, nap-of-the-earth flying, or flying in formation.

The job of generating the images that the trainee sees is of course the main function of the IG. It must draw the images representing the "out-the-window" view of the vehicle being simulated. To provide the illusion of continuous motion, the IG must render a new image (or "frame") several times each second.

The frames generated by the IG are conveyed to the trainee by the display system. Display systems vary greatly depending upon the particular application. Typically, a display system will consist of one or more CRTs (or other projection devices), each scanning out a raster image composed of 1/4 million to one million pixels.

Ideally, the views presented by the IG would be realistic images such as the trainee may expect to see in a real environment. However, the computational requirements for presenting such realism in real time are beyond the capabilities of any graphics machine to date (except for a limited class of environments). Thus the compromise is that IGs present views containing only the necessary information for the training task at hand. For night-time landing, all that may be necessary is a set of runway lights, while the scenery required for tactical training may be a large, detailed combat area containing several moving combatants.

We call the stored representation of the scenery to be drawn the "database". It typically consists of terrain, light points, and some fixed and moving objects. The terrain and objects are typically modeled from polygons with various texture patterns applied.

(See [SCHA83], [TUCK84], [YAN85], and [ROLF86] for some general FS information. Much of the unreferenced information below is derived from these sources.)

2. High-level IG Considerations

We now enumerate some of the requirements and issues that set FS IGs apart from the context of general purpose graphics systems. These include:

- the rendering speed requirements
- the rendering quality requirements
- the nature of the database
- the nature of the display devices
- the additional functions assigned to the IG

These items are the focus of this section.

2.1. Rendering Speed

2.1.1. Real-time

FS IGs must be truly real-time graphics computers. The typical performance requirement is 30 frames drawn per second, with up to 60 frames per second also common. These requirements vary somewhat depending upon the exact application: the faster the vehicle can maneuver, the faster the update rate should be, and vice-versa.

It is essential that the IG synchronize with the rest of the system. The visual information provided by the IG must keep pace with motion or audio cues from other systems, and also with simulation's dynamics as it responds to the pilot's inputs. Therefore the real-time requirement of IGs is often strict: a frame must never take more than the allotted time to generate. Some systems may relax this requirement slightly by allowing the image generator to fall slightly out of sync when it has to (due to some overload condition), provided that it can catch up eventually.

2.1.2. Low Latency

Another aspect of rendering speed is latency, or the time it takes from when all viewing parameters are supplied and an image has begun rendering until it appears on the display. This latency is also referred to as transport delay. Having low latency is important since the pilot should not feel any unnatural delays in the time between his control input and the visual feedback he receives. Again, the amount of latency tolerable depends upon the exact application.

Even if the simulation itself might tolerate long latency, there are display devices which will not. For instance, some simulation systems use head-mounted displays (HMDs), which provide the pilot with images based upon his line of sight. Here especially latency is a critical issue, since too much will not only be unnatural and distracting, but may also cause "simulator sickness".

The HMD systems used in FS are usually of the "see-through" variety (as opposed to the enclosed, "immersive" type) since it is necessary for the pilot to see his physical cockpit environment (including his hands). In these setups, zero latency is desirable to avoid the distracting "swimming" effect of the virtual scenery in relation to the real physical scenery. ([LACR94] discuses some HMD issues as related to FS.)

2.1.3. Budgets

Given the limited time available to generate a frame, and given the amount time it takes to render an average polygon, what results is the average number of polygons that can be rendered per frame: the "polygon budget." A major problem is that the number of polygons present in a realistic scene may be several thousand or hundreds of thousands. While relatively small polygon budgets might be adequate for certain training applications, other applications continually push the limits of what modern technology can offer. This is feature shared with general purpose graphics computers: more raw polygon processing power is always in demand.

Related to the polygon budget is another factor known as the pixel budget. Each pixel from each polygon takes some amount of time to process. A scene that might meet an IGs polygon budget may still take too long to render if, for example, the polygons all have more pixels than "average", and thus exceed the pixel budget.

2.2. Rendering Quality

2.2.1. Antialiasing

Several elements are involved in producing an image of the necessary quality for simulation purposes. Perhaps of prime importance is adequate antialiasing. Antialiasing involves implementation of polygon rendering algorithms with proper filtering to prevent the visual anomalies such as "stair steps" that result from simplistic rendering algorithms. These visual anomalies not only take away from the realism of the scene but can also distract the pilot and make him miss the subtle cues that are important to his training.

There is no serious FS IG system that does not address the issue of antialiasing. While there is a fair amount of variability among systems, the FS IG minimum standard for antialiased polygon edges is 16-sample sub-pixel accuracy. For the rendering of textures, the standard is to use trilinearly interpolated MIP maps. Very few systems advertise anything less.

2.2.2. Display of Light Points

A feature particular to FS systems is the display of accurate light points. Early FS systems were designed specifically for the purpose of night training. In these systems, the principal (or only) visual features were airport lights, of which airports are full of, with many specialized varieties (directional, multi-color, strobing, etc.) designed to aid the task of night landings.

These night systems typically used calligraphic CRTs, where the electron beam could be controlled precisely in order to render the light points correctly. These display systems were very limited in their capability of drawing shaded surfaces, and thus were not so useful for daylight simulations. While raster displays solved the surface problem, they generally cannot display light points as accurately. Proper antialiasing of raster light points may be adequate for many purposes, but higher-end systems use special displays that can operate in both raster and calligraphic modes.

2.2.3. Textures

While light points provide the main visual cues for night flying, daylight flying depends upon other kinds of cues. One such cue is "optical flow." The term is used to describe how the complex patterns seen in a real environment appear to "flow" as one moves through the environment along a given path.

Optical flow is achieved by two main methods: adding detailed scenery to the database and adding texture patterns to the scenery. Because of polygon budget limitations, the second method has received attention starting from some of the very first raster-scan simulators. Fields, forests, bodies of water, sky, and runway asphalt all can be given a much more realistic appearance by "simply" adding texture.

With the availability of highly detailed pictures of the earth (through the Defense Mapping Agency and other efforts), texture is being used increasingly as a modeling tool to give true realism to terrain models. This requires that the IG can access a large volumes of photo-specific texture information, and thus be able to "page in and out" texture data dynamically during simulation. This is often referred to as a "global texturing" feature of an IG.

2.2.4. Transparency

Support of transparency is another standard feature of IGs that provides a large amount of flexibility and rendering power. When combined with texture mapping, transparency can simplify many modeling problems, such as the representation of trees. Certain special effects (mentioned below) are also made easier with transparency. Perhaps the most important use of transparency is for fading objects in and out of the scene and in between different representations. This is necessary for proper scene management (see below) while avoiding sudden changes.

The number of transparency levels required depends upon the specific task. From sixteen to 4096 levels may be supported. Most of the effects mentioned do not require a large number of levels.

2.2.5. Other Effects

Another set of visual features that is standard on most FS IG systems is the simulation of various visual effects. These may include atmosphere effects, such as fog, haze, precipitation, and cloud layers. Alternately, combat-oriented simulators may require the generation of effects such as explosions, flashes, or the display of artificial obscurants or debris.

2.3. Database Issues

There are two facets associated with database formats. One is the external representation of the database for archival purposes. The other is the internal or run-time representation of the database. This is our real concern, since the format affects the processing algorithms and even the basic rendering architecture of the IG.

The size of the database is a factor. Databases may span hundreds of square miles, and thus be much too large to even fit in memory. Therefore, database paging is an issue that most systems have to contend with.

Unlike general purpose graphics computers, FS IGs enjoy the benefit of having a limited, well-structured database organization. As mentioned earlier, the scenery for FS typically includes terrain, fixed ground objects, light points, and a few moving objects. The IG can and must take advantage of the inherent structure in such scenery, both in the manner in which the scenery is represented in the database, and how the resulting database is processed. It is common for the database to be subdivided into sections containing the parts mentioned, with each part stored in a specialized format.

2.4. Display Systems

Flight simulation is somewhat unique in terms of the large variety of display devices that have been employed. In the quest for a display system with all of the desired parameters, many different setups have been tried. Different display devices, including various types of CRTs, light valves, projection systems, laser-based devices, and others have been used in many kinds of display systems. A display system might be a single CRTs, or it might include multiple projectors aimed at spherical or cylindrical screens, perhaps employing an elaborate setup of mirrors and other optics. It might be an area-of-interest (AOI) display system, with servos rapidly positioning a projector image to follow the pilot's gaze as his head or eyes are tracked. It might also be a head-mounted display system, again coupled with a high-speed tracking system.

Common to many display systems is the requirement of generating multiple channels of video. This might be for multiple displays sharing the same viewport (a large front window, for example), or for multiple viewports with their own displays (front/side/rear windows), or for stereo displays in an HMD, or for a variety of other reasons. As with any multiple-display setup, all displays must be synchronized as the image is updated, or else artifacts might occur.

2.5. Other IG Functions

Aside from simply rendering the out-the-window scenery, an IG is often given other tasks, usually because of the intimate relation is has with the scenery database. Two such tasks include "height above terrain" (HAT) and line-of-site ranging (also known as laser ranging). These are basically queries of the nearest object either directly below the aircraft or along some specified direction. Sometimes more general collision detection functions are provided, such as checking whether a given volume or set of vectors intersects any database object. These tasks are often referred to as "mission functions".

A different kind of task that an IG may support is that of providing sensor image data. Rather than generate the out-the-window scenery, the IG renders the database as it may be seen be a forward-looking infrared (FLIR) sensor, or perhaps by radar or other type of sensor. These functions usually call for storing additional kinds of surface information in the database.

3. IG System Overview

To provide some common context, we now briefly describe a conceptual IG pipeline. The figure below shows such a pipeline.

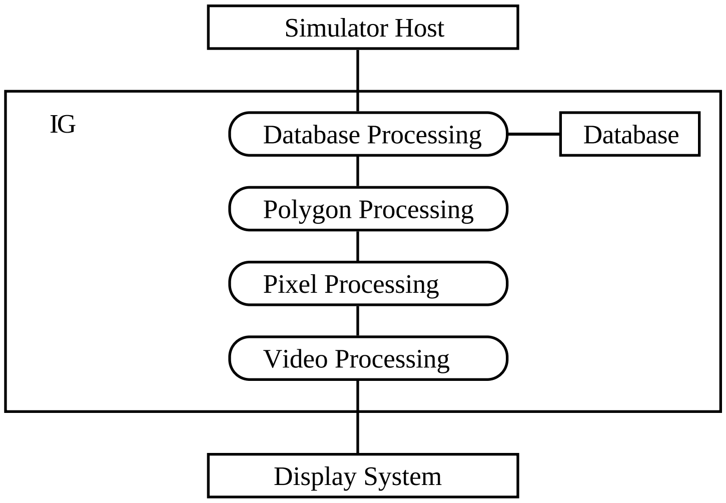

Figure 1. IG Pipeline.

The simulator host is the machine that is performing the non-visual computations for the simulation task. It communicates with the IG to specify information which affects the visual display, such as any database changes, or, more importantly, the current viewpoint information. It may also make queries of the IG for mission-related functions.

The purpose of the database processing stage is to examine the whole scenery database and determine, for the given viewpoint, which parts should be passed for consideration down the rest of the pipeline. This stage is also referred to as "scene management" or "database traversal". Since the entire database may be too large to fit in memory at once, this stage may also need to perform database paging from disk.

The polygon processing stage is where geometric transformation takes the polygons from object-space coordinates to screen-space coordinates. This involves applying the transformation matrix and perspective division operation to each vertex of the polygon. This is also where setup for shading and texture calculations occurs.

The degree of separation between database processing and polygon processing varies from system to system. These tasks may or may not be performed by the same processor or set of processors.

Pixel processing converts the resulting polygon descriptions into raster pixels. This is the stage where many pixel-specific actions happen: transparency and texture processing, shading, atmospheric effects, and antialiasing. The processor at this stage is often referred to as the rasterizer.

In one or both of the above stages (depending upon the specific system), the process of occulting takes place. This is process of deciding which parts of each object are visible, and which are covered by nearer objects.

In the video generation stage, some additional post-processing of the pixels may occur before the resulting image is transformed into an analog video stream. This might include color remapping or gamma correction, adding in digital overlays (perhaps from a HUD, heads-up display), or perhaps additional filtering for antialiasing. For the purposes of sensor-image generation, some systems apply additional post-processing at this stage.

The final streams of video are sent to the various display devices. In these, some additional processing may occur. This may include image distortion, edge blending, or contrast mapping. The first is used to help compensate for distorted screen geometry or display optics (some distortion correction may also occur previously in the IG). The second two are used to help blend multiple projected images into a single, matched, seamless image (these also may occur in earlier stages in the IG).

4. Performance Issues

Despite technology which has increased tremendously the computational power available to IG systems over the years, straightforward, using textbook rendering techniques to draw realistic images still require more processing time than that available to real-time systems. The art of IG design involves two tasks: finding new ways to harness the new technologies as they become available, and coming up with schemes for drawing as realistic a scene as possible while doing as little work as possible. This is what we shall examine now.

We'll try to examine systems on a feature-by-feature basis, following to some extent the order of features as they are encountered along the IG pipeline.

4.0. Motion Prediction

We have already discussed the fact that achieving low latency is an important IG concern. Given that the most IGs operate in fixed frame times, many include a prediction feature to allow the IG to extrapolate the motion of the simulated vehicle. Thus the IG will adjust the viewing parameters to account for the time taken to generate the image. This is not a very common feature; presumably, the simulation computer itself can perform such this task with greater accuracy, since it has more information about the simulation vehicle.

4.1. Database Processing

As mentioned before, database processing is the selection of potentially visible elements from the database to be processed by the rest of the system. Since this step controls the load for the rest of the system, some of the most important gains in IG performance can be achieved through the use of sophisticated selection algorithms.

We will refer to the entity that performs database processing as the scene manager. Besides simply choosing which elements are potentially visible (and "culling" those which are not), the scene manager can greatly reduce the IG load by choosing the proper representation for each element. This is known as level-of-detail (LOD) selection. The proper level of detail to display is mostly a function of range: as an element moves further away from the viewer, it can be drawn using simpler representations and still retain visual fidelity.

LOD is a powerful technique which will be discussed in more detail shortly. For now, we note that the creation of different LOD models can be a difficult task. For objects, this is typically done by hand. For terrain, automatic procedures can be used.

4.1.1. Database Structure

First, we consider database structure. Database processing is intimately tied to the choice of database structure. Because the entire database can be very large, it is not possible for the scene manager to check (cull-test) every element separately to determine if it is potentially visible. To reduce the number of checks necessary, one must take advantage of any structure which is present in the data.

Fortunately, FS databases are full of structure. As mentioned earlier, the typical database consists of terrain, fixed objects, light points, and a small number of moving objects. This categorization is one type of structure in itself; IG databases are universally partitioned by these categories.

The fact that the majority of the database tends to be static helps greatly. This allows one to take advantage of spatial coherence. The overall terrain can be divided into a number of patches; rather than cull-testing each terrain polygon, the scene manager now needs only to cull-test each patch.

4.1.2. Terrain

How the terrain should be represented is another question. It can be represented very efficiently as an array of points forming a regularly-grided height field. Evans and Sutherland is a major proponent of this storage method [COSM90], which offers several benefits. One is the compactness of this representation: for a given area, only Z (height) values need to be stored; the X and Y positions are given intrinsically by the array position. Compact storage is important since it not only conserves IG memory, but it also means that the database can be paged quickly from disk.

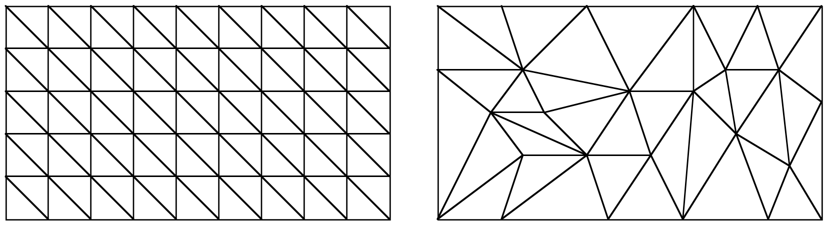

Figure 2. Regularly gridded terrain (left) vs. irregular triangular mesh (right)

With an opposing view, Martin Marietta (among others) favors an irregular triangular mesh for terrain [DONO]. The argument is that with such an arrangement, the triangles can be more custom-tailored to the environment, and thus many fewer are required. While the space savings may not be great, there are potential savings in the number of triangles that must be drawn.

Because of the tradeoffs involved, some companies (such as Sogitec) support both approaches. Sogitec claims that the former method is more suited for the low-detail level of representation, while the latter is more efficient for the high-detail representation [CHAU94a].

4.1.3. Hierarchy

Universally, IG databases are arranged in some type of tree-structured hierarchy. Each node in the hierarchy includes bounds information. Thus when the scene manager finds that a particular node is not in view, it can skip the processing of that node and all of its children nodes.

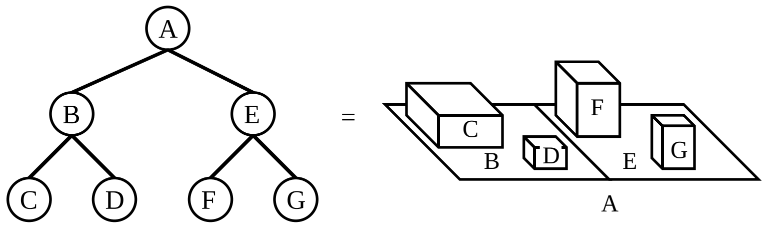

Figure 3. Hierarchy example.

To further help out the scene manager, there are some special types of hierarchy nodes included. One type of node subdivides space into regular regions. Each region forms a subhierarchy which could correspond to, for instance, a patch of terrain and all the objects found on that patch. With a regular region subdivision, visibility culling becomes a simple matter of traversing only the regions which overlap the field of view. Regions which do not overlap do not even need to be considered.

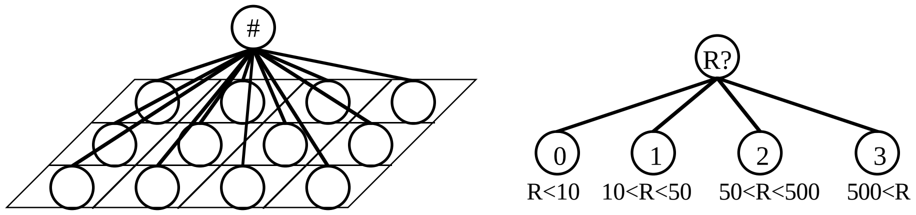

Figure 4. Regular region subdivision node (left); LOD-decision node (right)

Another special type of node is a LOD-decision node. When traversing such a node, the scene manager performs a range test based upon the data provided to make a decision about which branch to follow next. Thus the subhierarchies describing the more complex versions of terrain or models are not traversed when they are too far away.

4.1.4. LOD

We now consider LOD again. While we have already said that an object that is far away can be drawn with a simpler representation than when the same object is nearer, there is still the question of how one can switch from one representation to the other as the object moves nearer or farther. Just toggling between one representation and the next is not practical, since this will typically cause a noticeable "glitch" or distraction. There are two ways around this.

The common approach is to take advantage of the IG's transparency feature to fade out the "old" representation while simultaneously fading in the "new" representation. Since during the time of the fade, both representations are drawn, this time should be minimized. Thus once a range threshold has been crossed to trigger a change, the fade will proceed at a given pace regardless of range.

Another approach to LOD transition requires the IG to mutate the polygons of one representation until they match those of the next. This is the approach Evans and Sutherland has taken with terrain LOD processing. Since they use terrain based upon regular grids, this approach is quite practical [CLAR90, COSM90]. Because the number of polygons drawn during such LOD transitions is not greater than the number required by the more detailed representation, such transitions can be strictly range-based (rather than time-based).

Given the ability to perform LOD transitions, there are some points to be careful about. With terrain, for instance, one must be careful that the LOD chosen for one patch will still match up with its neighbors, even if they have a different LOD. Ignoring this consideration can lead to "cracks" in the landscape.

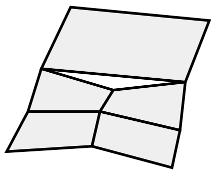

Figure 5. Terrain "cracks" caused by neighboring patches with differing LODs.

Another consideration is how LOD affects the relationship between terrain and the objects which sit upon it. When terrain changes LOD, it typically moves up or down slightly. As it does so, the objects that rest upon it must also move correspondingly. In addition, the terrain is changing shape. Any features such as long roads or pipelines that run along the terrain must change shape as well. This issue is addressed well in [CLAR90].

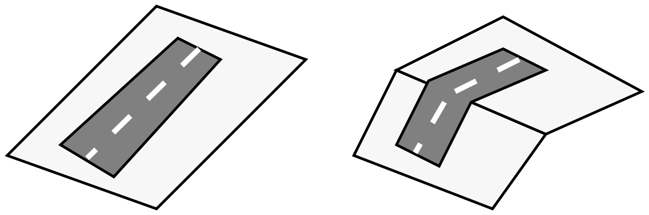

Figure 6. Object/terrain conformance vs. LOD.

A further consideration involves systems with multiple image channels. If an object crosses the boundary between two channels, it must be drawn with the same LOD on both channels [JARV87]. If this is not done, unnatural appearance may result. Since LOD is based upon range, this is not usually a problem. However, we will see in a moment why LOD is usually not based on range alone.

Finally, one might want to vary the range at which objects transition on an object-by-object basis, depending upon the "importance" of each object for training purposes. An approaching generic tree might retain low detail until it gets quite near, while an approaching tactical target should be provided more detail even when still quite far.

LOD seems to require a fair amount of work to implement. What does one gain from using it? Latham reports in [LATH85] that by using LOD, one can render scenes with one hundred to one thousand times greater complexity while using the same polygon budget. In addition, LOD allows the database paging rate to be reduced considerably, since it allows detail to be paged in gradually on an "as-needed" basis. LOD is undeniably an extremely valuable performance technique.

4.1.5. Small-Object Culling

A feature related to LOD is the culling of distant objects. This is useful when the objects are so distant that they occupy an insignificant portion of the screen, and may be viewed as a transition to the null LOD. Again, to avoid the problem of objects "popping" in and out of the scene, a transition strategy such as fading must be used.

4.1.6. Overload

As mentioned, IGs have strict polygon and pixel budgets. When a database is created, one cannot readily tell if all potential views of that database will fit within the IGs budgets. It is therefore inevitable that some scenes will produce an overload condition: there will be too many polygons or pixels to process in one frame time. Overload is extremely undesirable given the real-time requirements of the FS application. Fortunately, there are many ways to deal with it.

A simple strategy involves processing the scene in near-to-far (front-to-back) order. When time constraints dictate, the IG simply stops processing scene elements, shows what has been drawn so far, and begins to work on the next frame. The effect is that distant objects may be dropped from the scene. If these were small, insignificant objects, this may not be a problem. If the object were a large mountain or an approaching enemy vehicle, then there would be a problem. This strategy alone is usually not an adequate solution.

Another strategy, quite commonly used, is to adjust the transition ranges for LOD and small-object culling. This method requires more careful consideration. The technique involves keeping the IG running at about 95% capacity. When this threshold is crossed (because a frame takes too long to generate), the transition ranges are brought in slightly until the load decreases. When the load is reduced enough (down to 90% capacity, for example), the transition ranges can be increased back to normal again. The difference in capacity thresholds is necessary to prevent oscillation effects.

Given that overload compensation will take place only for the next frame, and that the next frame is different than this frame, overload may result anyway despite the attempted corrective action(s). If the frame requires more than 100% of the allotted time, the frame time must be extended to prevent dropping of scenery. This can be done either by altering the display's refresh cycle or by using additional display cycles. The former requires specialized IG-controllable display devices.

4.1.7. Run-Time Feature Conforming

It has already been mentioned how changing terrain LOD requires adjusting the features planted on the terrain. The idea can be generalized further: features and terrain can be modeled completely independently and attached together at runtime [CLAR90]. The advantage of this approach is that feature instancing can be used more widely, as features can be more generic since they do not have to be custom fitted at modeling time to any particular landscape. Thus the size of the database is reduced, and database bandwidth requirements are reduced as well. As pointed out by Rich [RICH92], the conforming of the features at run-time need not necessarily happen at frame rates; the terrain isn't necessarily changing every frame.

4.1.8. Generic Fill

One of the more interesting database techniques to appear is the use of generic fill methods. An early example is the use of "environmental universal features" in General Electric's AVTS system [FERG84]. The EUF's are simply a set of ground clutter objects (such as trees, rocks, or shrubs) which are placed on a surface using a type of texture map. A more sophisticated example is the generic fill algorithm described by Rich [RICH92], where generic fill objects are placed on surfaces procedurally.

While one of the advantages of using generic fill is ease of modeling, such techniques also help to improve IG performance. The techniques allow much more compact database specification of fill-type objects. Again, this results in reduced database storage and bandwidth requirements. The front of the IG pipeline can run faster since it can spend less time moving database information around.

4.2. Occlusion

4.2.1. List-priority or Z-buffer Rendering?

One of the biggest issues in IG design used to be the choice of occlusion algorithm used. The choice of occlusion algorithms is intimately related to an IG's polygon and pixel budgets. The basic choices are Z-buffer and list-priority algorithms, and the issues involved are fairly universal: performance, cost, and ease of use. Let's review the algorithms first (in simplified form, for ease of explanation).

With the Z-buffer algorithm, polygons may be processed in an arbitrary order. Each polygon is rasterized into pixels; computed with each pixel is a function (based on Z) of its distance to the viewer. The pixel is checked against any other pixels previously written to the same location; if it is nearer, it is written there, else it is discarded.

List-priority algorithms require that the database be rendered in front-to-back or back-to-front order. For front-to-back rendering, each polygon pixel is written only to empty screen locations. For back-to-front, every polygon pixel is written regardless of what was written before. Front-to-back priority is the favored approach, since each pixel must be written to only once.

There are many papers covering the advantages and disadvantages of the different algorithm choices. We will cover them only briefly here.

One problem with the straightforward Z-buffer implementation is its high pixel budget: every pixel of every on-screen polygon must be computed, whether it is visible or not. Another major problem is that the unordered polygon processing makes antialiasing and transparency difficult or expensive to perform properly. However, this is also the main advantage of Z-buffer rendering: it makes no assumptions about the polygon order or placement, and thus can render any polygon arrangement correctly without special treatment.

List-priority algorithms have a lower pixel-budget, and the polygon ordering makes antialiasing easier. However, these algorithms have two main disadvantages. One is that they are well-suited only for static scenery; moving objects require special treatment in order to be sorted properly. The other problem is that they cannot render interpenetrating or cyclically-overlapping polygons properly. Such polygons must be split, and this leads to many modeling and database difficulties.

List-priority used to be the algorithm of choice for IG designers. It offered performance and image quality that could not be achieved by Z-buffer systems of similar cost. However, three factors have been pushing more recent IGs to favor Z-buffer approaches. These are the difficulty of creating databases tailored for list-priority systems, the increasing number of moving objects in simulations, and the fact that VLSI technology has been driving down the cost of Z-buffer systems.

As a compromise, some systems adopt a hybrid strategy. For instance, they may use the list-priority algorithm for the static scenery and then merge in moving objects using a Z-buffer. Such architectures are probably just stepping stones on the way to a full Z-buffer approach.

The dominance of the Z-buffer does not mean the end to development of occlusion strategies, however. Advanced occlusion strategies have been developed in order to reduce not only the pixel budget of the system, but also the polygon budget as well. Front-to-back rendering is still a big issue with such strategies.

4.2.2. Advanced Occlusion Strategies

Before we abandon list priority, let's examine an advanced occlusion scheme used by a list-priority architecture, the Singer Company's MOD DIG (Modular Digital Image Generator) [LATH85, YAN86].

At the lowest level, this system (and many others) operates on rectangular portions of the image called "spans" (among other names). A span may be an array of, say, 16x16 sample points contributing to 2x2 pixels.

Since MOD DIG uses the front-to-back list-priority algorithm, it can skip over already-filled-in spans. The system goes two steps further and skips over already-filled-in row-segments of spans and already-filled-in complete rows of spans. It does this by utilizing a hierarchy of mask buffers that indicate when a particular span, segment, or row has been fully covered. As a polygon is converted into spans, the mask buffers are read to indicate when sections can be skipped over (see figure 7 below). The mask buffers are then updated when new spans are completely filled in.

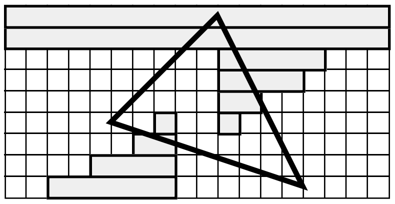

Figure 7. MOD DIG masks.

The idea that front-to-back ordering can speed up rasterization has been applied to the Z-buffer as well. We describe several approaches here.

General Electric's COMPU-SCENE PT 2000 [BUNK89] (among others) first dices up the polygons (which are processed in arbitrary order) according to the span layout. The collection of polygon fragments for a given span can then be easily sorted into range-separable groups. The groups are processed in front-to-back order until the span is fully covered. In the example shown below, processing can stop after group 2.

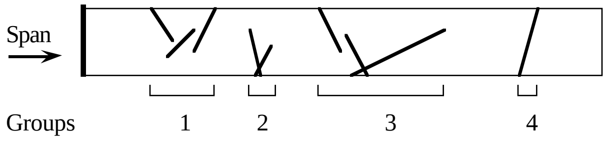

Figure 8. Looking at a span sideways.

The Loral GT200 system [LORA] is a Z-buffer based system that uses what Loral terms a "mask buffer" to help accelerate the rendering process. The technique requires a certain amount of front-to-back polygon ordering. First, all polygons up to a given depth are rendered normally. For each pixel that is filled, a bit is set in the mask buffer. Before the remaining scenery (which is all at a greater depth) is rendered, it is tested against the mask buffer. Any elements found to be obscured are not rendered. The technique is quite powerful, since the mask buffer test is performed at multiple levels: for terrain regions, for objects, for polygons, and for individual pixels. For the mask buffer tests to be efficient, it is likely that the mask buffer is structured in a hierarchical fashion similar to the masks in the MOD DIG above. Thus a single bit can be checked to determine if a given area has already been covered.

The mask-buffer technique can be extended further to provide even greater culling power. Instead of having a hierarchical buffer of mask bits, one stores depth values instead. At the finest level, one simply has the traditional Z-buffer. For blocks of pixels, one stores the farthest Z value in the block. This is repeated several levels for blocks of blocks. To perform the cull-test, one checks the nearest Z value of the given object against the Z value from the smallest block that encloses the object. If the object Z value is further, then it is completely obscured. If the object Z value is nearer, then the test may be performed recursively over the sub-blocks, rendering the object only where necessary. It appears that Sogitec's APOGEE system uses this technique to some extent with their "Meta-Zbuffer" system [CHAU94a].

4.3. Pixel Processing

4.3.1. Antialiasing Strategies

Antialiasing is a critically important feature for FS IGs. As such, it is not surprising that some very sophisticated techniques have been incorporated into hardware. We discuss a few of the approaches taken.

The techniques used by the E&S CT5 were described in several conference papers [SCHU80, COSM81, WALE83, HOWI84] and subsequently copied and used by many IGs. The CT5 is a list-priority machine, and this allows fairly sophisticated antialiasing to be implemented without too great expense.

The basic elements involve a high-resolution mask memory, a filtering device, and an accumulating framebuffer. The system scan-converts and processes the polygons in small area units called spans. The size of the span is the size of the antialiasing filter kernel and contains a large number of sampling points (perhaps 64).

For each span output by the scan-converter, the polygon is evaluated for coverage at each sampling point. The binary coverage of each sampling point for all previous polygons for the same span is stored in the mask buffer. Two binary computations are performed:

current coverage - mask coverage

current coverage + mask coverage

The difference result is passed onto the filtering device, while the sum result is stored back in the mask buffer.

The filtering device takes the coverage information for the visible part of the current polygon and weights and sums each sample point appropriately according the filter function. The result is a fraction which is then multiplied by the computed polygon color for the overall sample. Finally, the filtered color result is accumulated in the frame buffer.

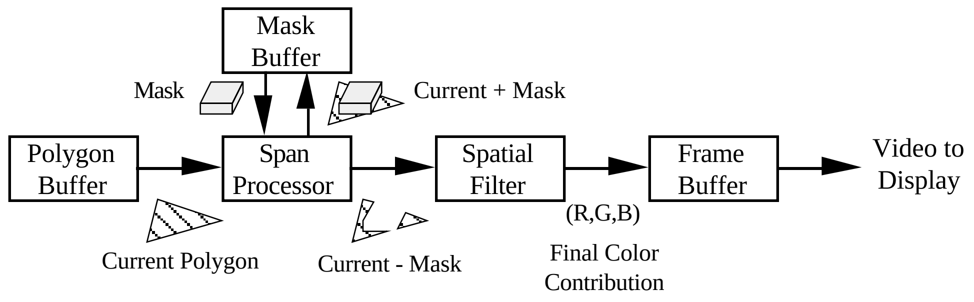

Figure 9. E&S CT5 Antialiasing.

By performing each sample point computation in parallel, accurate antialiasing can be done efficiently. Because occlusion is done using front-to-back ordering, only the one-bit-deep mask buffer needs to have subpixel accuracy.

As mentioned earlier, however, current IG systems are favoring the Z-buffer approach. How do they solve the antialiasing problem? In a system patented by Rediffusion Simulation [BAKE94], a full sub-pixel Z-buffer is proposed. This involves great expense, since one must now store and evaluate full depth values for every sample point. Various approaches have been taken to find ways around this expense.

In GE's PT 2000, a full sub-pixel Z-buffer is used, but only for a portion of the screen. Thus the polygons must be sorted according to screen regions, and each region is processed sequentially.

Another approach is Sogitec's AZtec system [CHAU94b], which uses notions from both A-buffer and Z-buffer algorithms. It boils down to an 8-entry per pixel Z-buffer, where each entry contains Z, mask, and tag information. This is in addition to the color frame buffer, which also participates in the algorithm. As polygons are scan-converted, their fragments are stored in depth order in the Z-buffer. Where possible, fragments are combined or deleted to conserve buffer entries. If a particular pixel overflows, a forced combination is performed. Finally, all the entries are collapsed to form a final pixel color. Special care is taken to avoid typical A-buffer artifacts. As a result, the chances that pixel will have a distorted color are extremely small.

4.3.2. Transparency

As discussed earlier, transparency is important both for modeling and for performing smooth transitions. Unfortunately, transparency is difficult to implement accurately in real-time. Accurate transparency processing requires that polygons or samples be processed in back-to-front order. This is contrary to other performance requirements which dictate front-to-back order.

Since transparency high precision is usually not required, many IGs use the "screen-door" transparency technique [FOLE90]. This involves "poking holes" in the polygon subsample mask, allowing underlying samples to show through. This technique has some limitations: the number of transparency levels is limited by the number of subsamples, and undesirable interactions are possible when multiple transparent objects are overlaid. An advantage of using this technique for object transitions is that occulting is not affected: even though an object is transparent, the back parts still will not be visible through the front parts.

For more accurate transparency, two approaches are possible. One requires keeping multiple samples around in a stack or buffer. Both the Rediffusion system and the Sogitec system discussed above do this. The other strategy is to process the transparent polygons only after all the opaque polygons have been processed. If the transparent polygons are sorted back-to-front, they can then be easily blended into the framebuffer. If they are not sorted, they can still be rendered correctly using a multipass algorithm as discussed in [FOLE90].

Since screen-door transparency and precision transparency offer different tradeoffs and appearance, many high-end simulators support the use of both techniques.

4.3.3. Texturing Strategies

The value of adding texture to simulation scenery was realized as early as 1963 in a system built by GE for the NASA Apollo program [BUNK89]. Since then, texture has become a standard feature for IGs, and many unique strategies of applying texture have evolved.

Early texture implementations were somewhat interesting. The GE system mentioned was able to draw polygons with perspectively correct textures formed from overlapping repetitive strip patterns. The technique, which was implemented with a relatively small number of diodes, logic gates, and op amps, is described in [SCHA83]. The technique was limited to planes parallel the ground plane and it required that the horizon be parallel to the scan lines. Thus to simulate vehicle roll, the raster image on the screen was rolled by turning the CRT deflection yoke.

Current techniques allow the application of perspectively correct, filtered texture patterns to arbitrary surfaces. The universal standard is the application of multilevel prefiltered texture patterns (often described as MIP MAPs [FOLE90], with varying interpretations), postfiltered using bilinear or trilinear interpolation.

4.3.3.1. Texture Application

The most obvious application of texture is to modulate the surface color of a polygon. The straightforward approach is to store RGB color in the texture map, and use the looked-up values to provide the polygon base color. The storage and processing required for 24-bit color texture is quite large, however, and thus many other strategies have been used.

The texturing strategies include: (TM = Texture Map)

RGB texture: three TM values modulate polygon color.

intensity modulation: a single TM value modulates the polygon intensity.

2-color blend: a single TM value specifies a blend factor between two colors:
$C = a\ C1 + (1\text{-}a)\ C2$

3-color blend: two TM values specifies a blend factor between three colors:
$C = a\ C1 + b\ C2 + (1\text{-}a\text{-}b)\ C3$

transparency texture: a single TM value modulates polygon transparency.
Transparent edges may be smooth or sharp.

illumination texture: one or more TM values modulate the lighting parameters used to
illuminate a polygon

With any of the above applications, the information from the TM may be used to either modulate the value intrinsic to the polygon, or replace it altogether.

4.3.3.2. Multiple Maps Per Polygon

Many IGs have the ability to combine the effects of multiple TMs on a single polygon. Thus a polygon may have both color and transparency values specified by two different TMs. Another use of this feature is for the application of detail texture.

Because a single texture map only has limited resolution, one will run into problems when a texture-mapped area is viewed from too near or too far. Too near results in a blurred effect from widely-space texels, while too far results in a repetitive pattern. To alleviate the problem, two texture maps are applied to the area at different levels of detail. Thus, when one is near, one can see the fine texture pattern, and when far away, one will see the course texture pattern. In fact, the fine texture pattern can interact with the course pattern to increase the apparent pattern size and thus help avoid the repetitive look.

High-end image generators often allow up to four different TMs to be applied to a polygon. Often, however, red, green, and blue are counted as individual TMs.

4.3.3.3. Texture Coordinate Specification

To apply texture, there must be a mapping from the polygon surface to texture space. Because texture was first applied to increase the realism of terrain, a simple mapping scheme was often used, known as ground-mapped texture. This involves simply projecting the terrain polygons down to a flat ground surface which is mapped to texture space. Thus the texture coordinates at the vertices of any given polygon are determined intrinsically by the polygon's position in world space.

To apply texture to moving models, however, different techniques must be used. A common technique appears to be the specification of mapping matrix from polygon space into texture space. Another alternative is the specification of explicit TM coordinates at each polygon vertex. The different techniques each have different behavior as textured objects are mutated. For instance, the former technique would allow one to make "brick" buildings of various size, yet still have all the bricks be the same size; the latter would scale the brick size with the building size. This becomes a concern when one wants to use different instances of a single model to conserve database storage and bandwidth.

4.3.3.4. Animated Texture Possibilities

Using a transformation matrix to specify or alter texture coordinates has another common application. By dynamically changing the matrix from frame to frame, animated texture patterns will result. This can be used to create "wave" effects in simulated water or crop fields. It can also be used to simulate explosions and similar effects.

4.3.3.5. Texture Storage & Paging

As mentioned previously, textures are commonly used as a modeling tool to create replicas of real-world landscapes by applying photographic textures to digitized terrain. Modeling photo-textured landscape for a large land area requires a large amount of texture storage. Thus high-end IGs, aside from having very large amounts of texture memory, also have one or more disk systems dedicated to storage of texture information.

In addition, moving throughout such photo-textured scenery requires real-time paging of the texture maps. To avoid having this information travel through the higher levels of the IG pipeline, systems such as Thomson's SPACE IG attach the texture disks directly to the pixel processors [JARV94b].

4.4. Fog, Haze

The simulation of fog and haze effects are required by FS IGs not simply for training purposes, but also because such effects obscure the horizon, thereby cutting down the IG load by allowing it to avoid drawing distant objects and scenery.

The simplest fog effect is to blend object colors into the fog color depending upon the distance of the object from the eyepoint. This is accurate for a uniform fog that surrounds the viewer and the environment. However, IGs are often called upon to render more complex effects such as ground fog or elevated fog layers; these require more complex simulation.

FlightSafety's VITAL VIII system [NIGU94] supports the display of variable density atmospheric layers through the use of real-time integration of a density layer table. Since the system also offers gradual transition through variations in regional weather, this table must be updatable in real time as well.

4.5. Volumetric Smoke

With their GT200 system, Loral has added capabilities to render more realistically effects such as smoke, dust, and clouds [LORA]. This is done by allowing texture maps to include depth and density information in addition to color. This information is used to appropriately modify the final computed color for each pixel.

4.6. Display Systems

The IG requirements set forth by display system requirements have varied widely over the years. Some early systems employed calligraphic CRT displays, which are limited in their drawing capability but well-suited for rendering light points. The IG requirements posed by such display systems were thus limited as well. In order to render solid surfaces, designers turned to raster-scan systems. Their timing requirements, combined with technological limitations, led to the generation of many scan-line based IG systems. As IG technologies improved, scan-line based systems were abandoned in favor of today's polygon-order, image-buffer based systems. We note that calligraphic display systems are not abandoned; high-end FS systems utilize displays with combined raster and calligraphic capability, since the latter allows more precise display of light points.

The need to tailor display systems to a wide variety of applications has resulted in a wide variety of approaches. The choice of display system is set by a variety of factors, including the type of vehicle being simulated, the task training requirements, human visual capabilities and limitations, and, of course, expense. A commercial airliner simulation may have only a pair of CRTs, whereas a military helicopter simulation may have a very large spherical screen filled by an array of CRT projectors. The literature contains many descriptions of some of the setups that have been used.

We take a moment to look at some of the IG developments that have resulted as a result of display considerations.

4.6.1. Calligraphic Light Points

As mentioned, calligraphic light points (CLPs) are still used as they can be drawn with more precisely and with greater brightness than raster light points. However, putting CLPs into the IG pipeline requires some careful consideration. Light points can obviously be occluded by solid objects and attenuated by transparent ones, and therefore CLPs must be present in the IG pipeline until the point where they would be stored in the framebuffer. At this point, the CLPs are stored in a list, to be processed after the raster has been drawn. [BAKE94] provides a good discussion of the treatment of CLPs.

4.6.2. Distortion Correction

Because of physical restrictions, display systems cannot always be designed such that a projected raster will result in uniform, rectilinear pixels. Adjusting the projector to compensate for screen distortion is often not enough, since the resulting pixel density (resolution) can vary widely over the screen. As discussed in [DOEN85] and [BUNK89], real-time distortion correction is a function which must be included in IG systems that are to work with unusual display arrangements.

Distortion correction requires that the IG perform the opposite distortion to the image as that which is done by the optics. Some arrangements may requirement only linear correction. This is fairly straightforward, and only requires adjustments to the viewing transformation matrix. However, nonlinear correction is more commonly required, as many display arrangements have curved optics. This requires different strategies.

One common strategy, discussed in [BUNK89], is to predistort the polygon coordinates as the polygons are rendered. Since straight lines must be mapped to curves, long edges must be subdivided into smaller segments to provide a piece-wise approximation to a curve. There is a performance tradeoff here, since small segmentation is desirable for more accurate correction, but having more segments increases the load for the IG.

Another approach, discussed in [BAKE94], is, instead of remapping the polygons, to remap the raster. Rather than performing rasterization on a rectilinear raster, it is done on a distorted grid. Thus the logical corners of each pixel are set according to a predistortion grid. The penalty is that the rasterizer hardware is more complex. A compromise help to reduce the penalty: the predistortion grid can be a piecewise-linear approximation of the true curve that is needed. Thus the rasterizer can still use linear techniques over a small area to increase performance.

4.6.3. Multiple Image Blending

Another display issue that impacts IG architecture is the need for blending multiple channel outputs into a single composite image. The alignment issues in such display setups can typically be taken care of by various means prior to real-time operation. While running, however, the IG must "feather" each channel image to blend it with its neighbors to avoid any discontinuities in the composite image. In addition, the perceived intensity of the composite image needs to be uniform over the entire display area. This issue can usually be addressed more easily through the IG than by adjusting the display hardware.

There are various approaches to solve these problems. The edge feathering can be achieved by displaying polygons with a transparency gradient. The polygons would be fixed in screen space along the appropriate edges of the various images. However, this approach may not provide the degree of control necessary to achieve a seamless image.

Achieving further control requires defining an intensity mapping function. A table is commonly used to define such functions, and the intensity remapping can occur as a post process applied to the video streams.

4.6.4. AOI Displays

A display system that calls extensively upon both of the prior features is the area-of-interest (AOI) display. This system is designed to reduce both IG and display system requirements by taking advantage of the fact that human visual acuity is greatest in the area around the direct line of sight, called the area of interest. Thus the system needs to draw a high resolution image only in this area; the surrounding area can be drawn with less detail. An AOI display system requires either eye and head tracking to determine the area of interest. The displays themselves can be implemented in a number of different ways.

On the IG side of this issue, one approach for the IG to change the LOD factors based upon the AOI. This can provide a small savings to the IG, but not to the display system. More advanced systems actually use differing image resolution for the AOI and surrounding areas. The usual way of doing this is by projecting a low resolution background image with a fixed projector, then adding a detailed AOI inset with a servo-controlled high-resolution projector. Creating the "hole" for the inset and edge-blending the images together are then dynamic processes which the IG must handle.

Evans and Sutherland advertises that their non-linear image mapping capability is versatile enough to support "a variable-acuity projector for use with eye-tracked systems." It is not known exactly what kind of projection system is referred to for this use.

5. System Issues

We now address a variety of issues which involve the system as a whole rather than individual pieces. These include the interconnections between IG components, and issues related to parallel processing.

5.1. Interconnections

We take a moment to examine the various interfaces that are used to connect various parts of an IG system together. We start with the host interface.

5.1.1. Host Interface

The host interface is used to connect the FS host to the IG. The main information that flows here is the data that indicates the new viewing position. As this link is the first in the IG pipeline, it is important that it have little latency. This link also carries a variety of other information, such as FS host changes to the IG database, database inquires to IG, and inquiry results to be returned to the FS host. For such purposes, small to moderate bandwidth is required.

A wide variety of technologies are used for the host link. These range from RS-232 serial to DR11W parallel, ethernet, or DMA through a shared bus. A large number of custom interfaces have been used as well. RS-232 is considered much too slow except for low-end applications. The other interfaces named are more common. Many IGs use a VME bus and incorporate extra slots for the FS host to be embedded. Ethernet is a popular interface choice because it allows easy interconnection for networked simulation systems.

5.1.2. Other Interfaces

The other interfaces within the IG are:

- database processor to database storage unit
- database processor to polygon processor
- polygon processor to pixel processor
- pixel processor to frame buffer

The first interface mentioned is the one used to load and page the simulation database. The SCSI interface is universally used here to attach disks, tape drives, and CD-ROM drives. Some systems also provide a SCSI interface for the pixel processor as well so that texture maps may be paged directly to this unit without having to travel down the pipeline.

The other interfaces associated with the IG pipeline require high bandwidth, and there have typically custom interfaces tailored to the particular needs of each system. However, more recently, there has been movement towards standardization for some of the interfaces. This movement is the result of efforts to lower costs and also of the emergence of interface standards with increasing performance. Thus Loral is using the "Skyburst" interface to connect polygon processors to pixel processors [LORA], while Kubota is suggesting that the PCI bus will eventually be used for the same purpose (from IMAGE VII conference presentation).

5.2. Parallelism

We now examine the topic of parallelism as it is used to increase the performance of IG architectures. Parallel processing at various stages of the IG pipeline is a concept that has been implemented since some of the very first IG systems [BUNK89]. We examine how various systems have incorporated parallel processing into their systems.

5.2.1. Parallel Pixel Processing

The pixel processing stage is typically the most computationally demanding part of an IG. A display that contains 1024 x 1024 pixels and which must be updated 30 times a second requires processing of more than 30 million pixels per second minimum. When one adds in the fact that each pixel is really the weighted sum of 16 or more samples, one can see that tremendous computational power is required.

Fortunately, the pixel processing task subdivides easily into multiple parallel tasks. Different areas of the screen may be assigned to different processors, each working independently. There are a large number of ways that the task be divided, however. We examine a few of the possibilities.

Sogitec APOGEE [CHAU94a, CHAU94b]

The Sogitec system subdivides the screen into 64x64 pixel regions referred to as "zones". The pixel-processing boards in this system each contain four processing "cells". Up to 8 pixel-processing boards are allowed (thus up to 32 cells). A single zone can be processed by multiple cells, though greatest efficiency would seem to be when cells are working on different zones.

The pixel processing algorithm of this system is rather interesting. An impact processor (one per pixel-processing board) examines a polygon an decides which boards and which cells in a board should deal with it. A cell processor examines the polygon and subdivides it recursively into 16 (4x4) pieces. Thus the polygon is processed in chunks of 16x16 pixels, 4x4 pixels, and finally 4x4 subpixels. At any given level, fully-covered chunks are passed to the rendering stage, while partially-covered ones are subdivided again. At the lowest level, the 4x4 subpixels are processed in parallel.

Loral GT200 [LORA]

The Loral GT200 IG allows up 12 pixel-processing boards in a system. Each board is responsible for filling the pixels in a 64x64 subregion of the overall display. The subregions are distributed in interlaced fashion over the display area. Within each pixel-processing board there are four pixel operators, each of which is responsible for a set of 2x2 pixel areas interlaced over the subregion. Presumably the 2x2 pixels are processed in parallel.

Thomson SPACE [JARV94a, JARV94b]

Thomson does not say much about their pixel processing architecture except that it uses a parallel array of pixel processing elements that performs computations for 64 "sub-rendering samples". However, they also claim that "an array of 16 custom-designed ASICs operate in parallel on each pixel area" and that "a depth comparison of 16 sub samples within each pixel is made before any anti-aliasing filter is applied."

Star Graphicon 2000 [STAR]

The Star G2000 pixel-processing boards each contain 5 logical pixel processing units which operate in parallel to each produce a stream of antialiased textured pixels. Up to four pixel-processing boards can be used for each output channel.

Rediffusion Simulation patented system [BAKE94]

The Rediffusion system is similar to the Sogitec system in certain ways. It first scan converts polygons into "supercells", each of which consists of a 4x4 array of "pseudocells", each of which is a 4x4 array of pixels (plus a half-pixel boundary area). The supercells are then processed by an array of "presorters", each of which produce a list of polygons that intersect a given pseudocell. The patent suggests that using four presorters is a good choice, though one to sixteen can be chosen, with presorters processing multiple pseudocells sequentially if fewer than sixteen are used.

The pseudocell-sized polygon fragments are passed to a set of modules that compute a polygon color for each pseudopixel within the pseudocell. These colors are combined in a "postsorter", which includes 200 sampling point processors and 16 weighting/accumulation processors (one for each pixel in the pseudocell). Presumably multiple color modules and postsorters can be used to increase performance.

Earlier systems: E&S CT5 [SCHU80], Singer MOD DIG [LATH85, YAN86]

Both the E&S CT5 and the Singer MOD DIG used similar pixel processing techniques as discussed earlier. We repeat the salient points briefly. The pixel processors operate on spans of 4x4 pixels (for MOD DIG; E&S avoids mentioning exact sizes). Computation for all of the samples in a given span occurs in parallel. Multiple pixel processors can be used; MOD DIG allows up to four pixel processors per video processor. Spans are interlaced over the available pixel processors.

5.2.2. Parallel Polygon Processors

The next most challenging task is polygon transformation and setup. This task requires a large amount of floating point computation (whereas most pixel-processing uses fixed-point calculations only). The methods for incorporating parallel polygon processing are not quite as varied as the methods for parallel pixel processing.

One way in which multiple polygon processors are incorporated into a system is by having one for each display channel. The architectures of multiple channel systems is an issue that will be addressed shortly. Meanwhile, we examine approaches where parallelism has been used to increase per-channel polygon performance.

Early systems incorporated parallelism into the polygon processing task by pipelining. Given that the number of steps to be performed for polygon rendering are finite, this technique has its limitations. Papers by Moon [MOON85] and Latham [LATH85] suggest that 1985 is when a general changeover to true parallel polygon processing occurred.

Many IG systems add parallel polygon processing by simply extending the traditional serial pipeline: the parallel polygon processor is just a plug-in replacement for the single polygon processor. Thus each processor simply takes a section of the database as doled out by the scene manager; this may be handled in either round-robin or first-free fashion. The results from these processors are then combined before proceeding to the next pipeline stage. In the upcoming section on channel management, we will examine some of the alternative strategies for incorporating parallel polygon processors into a system.

5.2.3. Parallel Scene Management

The work of the scene manager may be parallelized for a couple of reasons: either to increase the database traversal speed, or to support a number of different display channels. Because the amount of work at this stage of the pipeline might not be very large, many systems do not provide for parallel scene managers.

5.2.4. Additional Processing

As mentioned earlier, the IG is often given the duty of computing mission functions or database queries. This task is usually handled by the scene manager in its spare time. However, if the number of queries is large, the scene manager may not have enough time. Thus some systems allow additional processors to be added to handle these additional functions.

5.3. Channel Strategies

We now examine IG system architecture as a whole, and in particular we examine the various approaches that have been taken towards facilitating multiple channel displays. As mentioned earlier, multiple channel display systems are not at all uncommon for FS IGs. As far back as the early 70's, GE built a 14-channel system for use by the Air Force [BUNK89]. Over the years, many different multi-channel strategies have been pursued. Since multiple channel capability can add a great deal of cost to the system, this is not surprising.

5.3.1. Single Fork

There are several obvious approaches. The simplest is to enhance the video section of the IG to generate multiple outputs. The earlier pipeline resources are then time-shared to generate each channel image. While this may be the cheapest approach, it offers poor scaling characteristics: the system performance is inversely proportional to the number of channels.

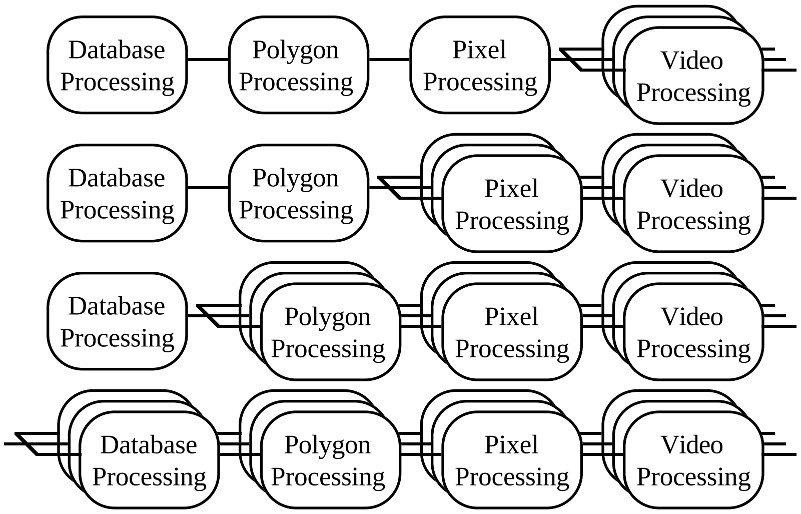

Figure 10. Single Fork Strategies

To keep performance more independent of the number of channels without greatly changing the system architecture, the "fork" in the IG pipeline can be moved to an earlier point. When the split is placed very near the beginning of the IG pipeline, the result is referred to as "fully channelized architecture." In this configuration, each channel has a complete independent IG pipeline, and thus performance is largely unrelated to the number of display channels. However, the system cost is now proportional to the number of channels.

To offer application and cost flexibility, some IGs provide both of the previous methods as channel expansion options. Additionally, the pipeline fork may be placed at other points in the pipeline. This results in different performance compromises depending upon which pipeline parts are shared and which are dedicated per channel.

5.3.2. Multiple Fork

A fully channelized architecture is not necessarily the best way to acquire the most performance from a given amount of hardware. Because there may be a large difference in the scenery displayed by each channel, it is possible that some channels will finish their work while others are still processing. Since the channels are independent, the resources of the idle channels cannot be used to help the more burdened channels. And since the channels must be synchronized, this idle time is wasted.

As a result, other channel strategies are taken. The common strategy is to provide not just a single fork in the IG pipeline, but several. Thus adding a channel may involve adding additional processors to all the pipeline stages while still maintaining a single pipeline that is shared by the various channels. A highly-touted advantage of the "multiple fork" approach is system modularity. For a given channel configuration, the various performance areas can be tailored by adding boards of the appropriate type. However, while the marketing literature would have you believe you can add as much performance as you want with this approach, that is seldom the case.

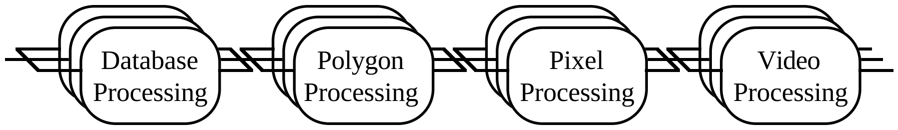

Figure 11. Multiple Fork Strategy

The performance ceilings for these systems are generally imposed by the communications networks connecting the pipeline stages together. Typically, various types of busses are used to join the various stages. Each bus typically has a fixed bandwidth ceiling, and these ceilings cannot be raised without redesigning the system. Thus the number of boards that can be added at any given pipeline stage is limited by the bus bandwidths into and out of that stage.

There are many questions left open, however, concerning exactly how the work of generating the scenery is split and recombined at each stage. Very few companies discuss explicitly how their systems resolve these issues, however. We did find one interesting exception, which we now describe.

Loral GT200 [LORA, SODE93]

We begin by describing how the portions of the database are assigned to the polygon processors. Pictured from above, the active database area is tiled into a many regions. Each region is assigned in interlaced fashion to the available polygon processors. The only regions which must be processed for a given frame are those which fall into the projection of the field of view onto the ground plane. Thus each polygon processor works on a set of database regions and outputs transformed polygons to the pixel processors by way of a polygon distribution bus. The polygon processors are also able to communicate with each other over an unspecified crossbar interconnect. This facility is used to allow lightly-loaded polygon processors to share the load from heavily-loaded ones.

As mentioned in an earlier section, the pixel processors are assigned to 64x64 "sub-regions". Each pixel processor handles multiple regions distributed in interlaced fashion across the entire multi-channel image. From the polygon descriptions sent by the polygon processors, the pixel processors compute shaded pixels. From the pixel processors, the shaded pixels are sent over a custom pixel distribution bus to a set of video generators. A video generator then produces the video signals which are sent to the display system.

This system makes heavy use of interlacing to solve potential load-balancing problems. If a particular pixel processor becomes overloaded, however, there appears to be no recourse. Load redistribution is possible among the polygon processors, but the costs of performing this redistribution are not clear. The issues involved are the overhead necessary to keep track of and manage load redistribution, plus the cost of actually moving the load from one processor to another.

So while the "multiple fork" approach seems to address channel load balance issues, in truth it only pushes the issues down to the next level. Multiple pipeline issues simply become multiple processor issues.

6.0 Conclusion

The various requirements placed upon IGs by the simulation task has led to the development of many interesting techniques and architectures. In this paper, we have attempted to cover some of these techniques, concentrating on those which are related to real-time performance.

These requirements have often led IG systems down different development paths than those followed by general purpose graphics computers. The paths have been interwoven in various ways, and, more recently, they have been have been coming closer together and even merging at many places. Technology has had a big impact here.

It has increased the performance of the general purpose machines and decreased the cost of the IG systems, to the point where the gap between them might be considered almost closed. (Witness Silicon Graphics Reality Engines and Evans and Sutherland's Freedom series.)

However, many of the advanced techniques for in IG systems have yet to find their way into the general purpose systems. As people develop more and more real-time applications for the general purpose systems, they find themselves facing many of the same problems that IG developers have faced down already. When the problems are similar enough, the IG developed solutions can be readily adopted. Still, sometimes the problems are characterized somewhat differently, and this may result in new solutions from the general purpose side. As the simulation field itself evolves, these solutions will often find homes in new generations of IGs.

Of course, such sharing of solutions requires communication. Let us hope that the silence, secrecy, and obscurity created by heavy competition do not overshadow such interaction.