IMAGE GENERATOR ARCHITECTURES AND FEATURES

Roy Latham

Research and Development Department
The Singer Company, Link Flight Simulation Division
Sunnyvale, California 94088-3484

ABSTRACT

In order to meet user requirements, tradeoffs are made in the implementation of the four functions (scene management, prioritization, geometric processing, and video processing) that comprise a digital image generation system of the sort used for flight training. This paper discusses how different approaches to image generator architecture affect the features apparent to the user. Among the architectural variations discussed are programmable versus pipelined geometric processing, and four variations of video processor (scanline, reverse-priority-ordered frame buffering, priority-ordered frame buffering, and distance buffering). The architectures are compared with respect to system cost, overload sensitivity, and the implementation of anti-aliasing, texture, and translucency features, among others. Understanding the tradeoffs involved will help designers and users better meet the requirements of a training task.

SPECIFICATION AND DESIGN OF IMAGE GENERATORS

Visual image generators are used to synthesize scenes which respond in real time to, typically, control inputs of a pilot in a flight simulator. The image generators discussed in this paper use digital electronics to generate signals to drive a raster display, usually a cathode ray tube or a light valve projector. Because the displayed scene must respond interactively to the control inputs, new images must be generated 30 to 60 times per second. This requirement, together with the required resolution and scene detail of the images, leads to the development of image generators which are large, complex electronic systems.

The functions performed by the image generator in making a scene may be grouped into four categories: scene management, prioritization, geometric processing, and video processing. The scene management function is the process of collecting the mathematical descriptions of the objects to be used in the scene, a process which usually starts by data retrieval from a disc storage device. Prioritization is the determination of which objects may occlude other objects in the generated image; roughly speaking, higher priority objects are closer to the eyepoint than lower priority objects. Geometric processing is the conversion of the mathematical descriptions of the three-dimensional data base objects into two-dimensional descriptions associated with the coordinates of the display. Video processing comprises all of the remaining steps needed to define the image at each picture element of the display, including the geometric subdivision of faces into pixels, detailed occlusion, and the generation of video signals for the display.

There is no single conventional architecture for implementing the four categories of functions, although there are a limited number of approaches presently in use. The situation with respect to image generator architectures may be contrasted with that of computers, for example, in which nearly all of the systems produced are refinements of one basic architecture. Having no conventional architecture for image generation systems is a situation with implications for both the user and the designer.

Ideally, the user would not need to know how system functions are implemented. Instead, the user might prefer to just know the features of the system, i.e., what the system does, not how it does it. Unfortunately, performance limits in present image generators are too great to permit a simple specification like "images shall be indistinguishable from the real world", and every attempt to write a specific detailed specification runs the risk of using terms or concepts which are inapplicable to the architecture and set of features of a particular design. For example, specifying the capability of a system to produce edges will run into problems in systems designed with limits on a polygon basis, or which make use of texture or curved surfaces.

From the designer's viewpoint, the situation would be simplified if there were simple common measures of performance. In the real situation, there are innumerable tradeoffs among architectures and features. The "correct" tradeoffs depend upon the intended application. Every product designer must be aware of the requirements of the intended user of his product, but the complexity of the product and the subtlety of the tradeoffs make this task especially difficult in the case of image generators.

This paper discusses some of the fundamental tradeoffs involved in designing an image generator to provide certain features to meet the user's needs. To limit the scope of the discussion, only the top level of image generator design is discussed, and then for only a restricted number of the most popular variations. The features discussed are limited to a few of the most architecturally interesting, but include both image related features, such as capabilities for textured and translucent objects in the scenes, and application related features, such as system cost and overload conditions.

SCENE MANAGEMENT AND PRIORITIZATION

System Functions

Scene management and prioritization functions are often performed concurrently in a general purpose computer, but the functions are distinct. The image generation for simulation begins with an eyepoint position and direction of view provided by another computer in the simulator system. The first step in making the image is to select the objects that might be in view from that viewpoint, and this selection process is the basic scene management function. Scene management is performed continually as the viewpoint changes, and the differences between successive viewpoints are usually so slight that the computation requirements and data bandwidths are within the capacity of conventional minicomputers. Similarly, the priority relations among objects which determine occlusion also change slowly in many applications, so it is convenient to run the prioritization algorithms in the general purpose computer as well.

The prioritization function is distinct from occlusion. Two objects have a priority relationship if one may potentially occlude the other, i.e., if every line of sight from a given viewpoint will strike the high priority object before the lower priority object whenever the two happen to overlap in the view. Occlusion is the precise determination of which parts of each object are visible given the priority relationship. If the objects interpenetrate or mutually overlap no priority relationship of the sort described will exist. Consequently, image generators which depend upon priority algorithms to establish priority relationships must use data bases meeting special conditions. These conditions usually end up requiring that all objects be subdivided into convex pieces.

Systems using priority algorithms usually have an architecture which feeds the results of the priority algorithm into the video processing (Fig. 1). The video processor takes the two-dimensional descriptions of the objects, as produced by the geometric processor, and uses the priority order to determine on a picture element (or finer) basis which portions of the objects should be displayed.

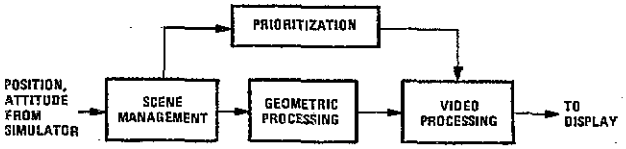

Figure 1. Conventional organization of image generator functions.

An alternative, however, is to have the prioritization performed in the video processor along with occlusion (Fig. 2). Conceptually, this approach is straightforward; for each picture element the video processor must decide which face is closer to the viewpoint and then perform occlusion accordingly. The distance sorting approach removes the restrictions on the types of objects in the data base that are imposed by priority algorithms, and it does away with the priority algorithm execution. Despite the conceptual simplicity and flexibility of distance sorting, the cost of the high speed hardware required to do operations on each picture element, and problems with picture quality and translucency effects, limit the application of the technique.

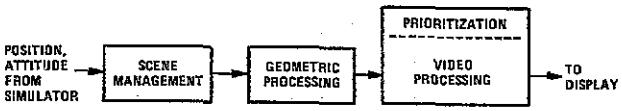

Figure 2. Prioritization performed with video processing in a distance buffered architecture.

If a priority algorithm approach is selected, there are circumstances when the processing speeds of conventional minicomputers may be inadequate. Ordinarily, objects change priority slowly as the viewpoint moves in the data base. Moving objects in the data base complicate the situation because they must be prioritized not only with respect to each other, but with respect to the fixed objects. In addition to running the priority algorithm more frequently, the real time software may have to build mathematical planes to separate moving objects from fixed objects. Separating planes, or other mathematical structures used by the priority algorithm, are ordinarily built off-line for the fixed data base. Altogether, increased processing requirements may require a more powerful computer or special purpose hardware to execute the algorithms.

Scene management functions may also tax the capacity of minicomputers. Ordinarily, the viewpoint changes relatively slowly, as determined by the motions of the vehicle being simulated. Consequently, lists of data potentially in view can be updated rather slowly. If the vehicle can turn rapidly or rotate, it is usually not a serious problem to expand the potentially viewed area to include a wider margin. Even expanding to the whole sphere of view may only require a few times as much data as the actual displayed portion. However, new developments in display technology make it possible to concentrate the scene detail in an "area-of-interest" where the person under training is looking. The ultimate in area-of-interest displays is an eye-tracked system, where scene detail is concentrated in the high resolution portion of human vision. This approach avoids wasting image generation capacity on portions of the scene only peripherally in view, but it also means that scene management must be performed at rates compatible with the very high slew rates of the eye. As with prioritization, this implies either a more powerful computer or specialized hardware, or both.

GEOMETRIC PROCESSING

In order to show objects which appear perspectively correct from any viewpoint, processing must start with a data base representation which gives a three-dimensional

model of the object to be depected. Most conventionally, the representation uses vertices in rectangular (x,y,z) coordinates to build polygonal faces and polyhedral objects, but the surfaces could be described by equations of curved surfaces or by more general recursive functions. However represented in the data base, geometric processing is performed to transform the objects into two-dimensional representations in the coordinate system of the display screen. The steps in this processing typically include translation and rotation, illumination, perspective division, clipping (to the screen boundaries), and computation of parameters needed for later shading and texturing of the surfaces.

There are two basic approaches to geometric processor design: special purpose pipelined hardware or general purpose programmable hardware. Until very recently, the only feasible method of achieving the required throughput was to build pipelined hardware specifically for the task, perhaps with some microprogrammable processes embedded in the design. The pipelined approach achieves very high throughput, but the design is complex and expensive to modify. A more general computer approach is now just becoming feasible with parallel processing and the use of high performance VLSI computing elements. Because the development of general purpose computing elements is supported by a broad market and the image generator market is comparatively small, the economics of the programmable approach will probably become more favorable with respect to custom pipeline designs as time goes on. In addition, there is the inevitable trend to more complex geometric processing algorithms to support a greater variety of image features. This means more programming rather than more custom hardware when the programmable approach is adopted.

The geometric processing problem is ideally suited to parallel processing because the image is built of hundreds or thousands of independent objects. Nonetheless, the computational requirements are formidable, requiring roughly 20 million floating point operations per second and perhaps 50 million bytes per second of input and output. Considering just the arithmetic operations, about 150 of the most advanced microprocessor arithmetic units would have to be kept busy to perform the calculations. Spreading the load evenly among so many processors poses many problems, but the approach is near the borderline of feasibility for devices as expensive as image generators.

Whether programmed or done in special hardware, there are distinct possibilities of merging scene management and prioritization functions into the geometric processing hardware structure, and one of the design tradeoffs in doing so is the flexibility of making changes versus the efficiency of processing. (This is evident within the software realm itself, as well. The trend is definitely away from writing scene management software in assembly language and towards coding in high level languages. The loss in execution efficiency is now more than compensated by increased flexibility and maintainability.)

Near the video processing end of the geometric processing, the principal design tradeoff is in regard to system bandwidth versus load management ability. Image generators often drive more than one display, so it is natural to dedicate hardware to each display to provide parallelism in the computations. However, whenever hardware is dedicated to a display, the system will have some susceptibility to overload in one channel while there is spare capacity in another, an obvious inefficiency. At some point in the system, the bandwidth of generated data becomes too great to permit interchange among processors, but the exact point is a design decision. One logical point to make the division is after geometric processing, but the split could be made earlier, in the middle of geometric processing. The point in the system after clipping to window boundaries is another logical point for the split. In general, systems dividing the data streams earlier will avoid design problems of high data bandwidths within the machine, but at the expense of less efficiency and greater overload susceptibility.

VIDEO PROCESSING

A general purpose computing approach may be considered for geometric processing, but video processing, with requirements hundreds of times greater, is safely in the domain of special purpose hardware for the foreseeable future. High processing and data bandwidth requirements are inherent in the generation of 30 million pixels per second per channel, the result of subdividing the displayed faces into picture elements. Each picture element is described by three color components which in turn are affected by smooth shading, atmospheric haze fading, translucency (which will show the colors of underlying pixels), anti-aliasing (which "smooths" edges), and a variety of other effects.

Scanline Architectures

One of the original approaches to dealing with so much processing was to perform the operations a scanline at a time in synchronism with the raster display of the scanlines. This approach is now well documented and has been used by most of the image generator manufacturing companies. [1]

In order to generate the data for a given line of the display, the objects (or the edges of polygonal faces forming the objects) must first be sorted to find those intersected by the line. The beginning and end of each face on the line is then determined, and occlusion performed using the previously computed prioritization. The visible pieces of polygons are then converted to picture elements and displayed.

It might seem that the sorting process would be a principal drawback to the technique, but it actually is not. The sort is implemented in hardware that supports many simultaneous comparisons, so that the N log N growth of sorting times associated with the sequential sorting of random arrays does not apply. In fact, the principal disadvantage is that the

entire processing operation is synchronous by line, and hence the processing overload condition will be driven by the single most complex scanline in the scene. Consequently, a scene which is sparse overall may nonetheless overload the system because time spent on a complex line cannot be made up later on empty lines.

Buffering two or more scanlines alleviates the problem by allowing the generator to run less strictly in synchronism with the display. Once it becomes economically feasible to buffer a whole screen, a family of "frame buffered" architectures becomes possible that offers other advantages in addition to the elimination of scanline overload.

Reverse-Priority-Ordered Writing

Rather than fill scanlines one at a time from the top of the screen, a frame buffer allows construction of the picture in a number of different orders, unrelated to scanline position. The most conceptually simple is the so-called "painter's algorithm" in which faces are converted to pixels and written into the buffer in reverse-priority-order, i.e. the lowest priority faces are written first and then overwritten by higher priority faces to accomplish occlusion. For example, the sky would be written first, followed by the ground plane, distant objects, and finally close objects.

Note that to accomplish occlusion by this method it is necessary to generate every pixel of every face, regardless of whether or not any portion of a particular face is visible on the screen. Thus if there are a million pixels in the display, several times that number of pixels may have to be generated, depending upon the amount of occlusion in the scene. Consequently, although the generation of the pixels is straightforward, a great deal of fast hardware will be required, and the system may be subject to overload under a non-intuitive condition of total pixel area written.

Priority-Ordered Writing

The overload due to total writing can be partially cured by changing to the less straightforward scheme of writing the highest priority objects first. In this method a mask must be built to keep track of the portions of the screen written during the course of processing. By reference to the mask, which could actually be built hierarchically with blocks of bits corresponding to the pixels of the display, succeeding faces can be checked for occlusion and portions skipped over as appropriate. System complexity is increased in that additional logic is required for the masking and skipping, but the efficiency is also increased. The processing of shading, fading, texturing, and the like is avoided on occluded pixels in favor of the simpler masking and skipping operations. If the skipping and mask updating can be accomplished much faster than the processing of displayed pixels, then the system will be very robust with regard to overloads.

Distance Buffering

Both priority-ordered and reverse-priority-ordered writing depend upon a separate prioritization process to support the video processing. Prioritization can be performed concurrently with video processing by expanding the dept of the frame buffer to hold the distance to the picture element in addition to its color components. The objects may then be computed in any order and the distances to new picture elements compared to those previously written. At the end of processing, the surfaces closest to the viewpoint at each pixel will be represented in the buffer.

For computational convenience, the existance to a plane parallel to the screen plane may be used instead of the true viewing distance, and the reciprocal of the distance can be used in memory rather than the direct function. Systems are called "Z-buffered" rather than distance buffered when the distance to a plane is used.

Z-buffering has the great advantage of solving the priority problem, but it has several other drawbacks. Like reverse-priority ordered writing, every picture element must be computed, even the ones ultimately occluded. The frame buffer must be enlarged to store the distance function, a disadvantage that diminishes as memory becomes cheaper. But the most serious disadvantages lie in the difficulties with anti-aliasing and translucency implementations, which are discussed in the next section.

IMAGE GENERATOR FEATURES

The features of a system are the elements of its specification. These elements include overload properties and other applications related characteristics, and the image quality and types of visual effects that can be produced. There are dozens of visual effects that can be itemized in an image generator specification, ranging from color properties to the generation of landing light illumination effects. For present purposes it is sufficient to discuss a few of the important features that have an architectural impact upon the system.

Image Content

Anti-aliasing. Aliasing problems originate from attempting to determine the display of an entire pixel based upon a single sample point within the pixel, or from too few sample points within the pixel. The symptoms of aliasing in an image are stairstepped edges, thin lines broken into dotted lines, and discrete rather than continuous motion as objects undergo transitions from scanline to scanline when the scene changes.

Solutions to the aliasing problem involve either the sampling of many points within the pixel and possibly adjacent pixels, and then combining the points in a weighted average to obtain the picture element color and intensity, or treating the image analytically with a similar combination of weighted areas in the picture element.

Translucency. There are translucent objects, particularly smoke and clouds, that enhance some generated scenes. However, the principal use of translucency is to facilitate the change in the level of detail of object models. To save capacity, simple models can be used for objects in the distance and then replaced with more detailed models as they become closer. A translucency capability allows a smooth transition among the models, thereby permitting later transitions without the distraction of a sudden replacement.

Texture. Texture is a modulation of the surface of a generated object using a hardware generated or stored pattern. The intensity of the surface is usually varied pseudorandomly to give a greater sense of depth and apparent motion in the image. Surface parameters such as color and translucency can be modulated in addition to intensity to provide a great range of effects. An interesting special case of texture is the reproduction of digitized photographs, perhaps including translucent regions, to add realism to the generated scene (Fig. 3.)

Figure 3. An image with translucency and texture effects.

Smooth Shading. Image generators designed to work with polygons can provide an illusion of smoothly curved surfaces by shading the objects as if they were curved. The smooth variation of intensity used for smooth shading can be extended to smooth variation of color or translucency in the same manner in which texture was extended for these parameters.

Curved Surfaces. Curved surfaces may be approximated by polygons, but special hardware can also be provided to generate the surfaces from a few parameters. Because polygon processing is so straightforward, especially in video processing, it is debatable whether less hardware is needed to directly generate complex curved surfaces rather than generating them by polygons, but curved surfaces clearly provide for a more compact data base representation of objects that are inherently curved.

System Features

The remarks regarding the variety of visual effects also apply to the profusion of other system features. Three of the most important are discussed here.

Overload Resistance. All systems have finite processing capacity, but all of the elements of the system should be well balanced so there are no critical bottlenecks. The capacity of the system should be as large as possible within the cost constraints, and there should be graceful overload and recovery characteristics.

Modular Construction. Presently, there is no choice but to make image generators large and to some degree complex, but many of the problems of large systems can be minimized if the architecture is highly regular. The "modularity" of a system may be quantified as the ratio of the number of circuit cards in the system to the number of types of cards in the system. High modularity is desirable because it helps production economy, simplifies training of maintenance personnel, and minimizes the spare parts requirements. The greater the modularity number, the greater the benefits.

Another reason for desiring a highly regular architecture is that it eases the transition to increased use of large and very large scale integration. The development of custom integrated circuits is more easily amortized if they are used repeatedly within the system.

Configurability. Designing an image generator from scratch is too lengthy and expensive a process to be undertaken for every new application. Consequently, it is desirable to design systems which are adaptable in terms of capacity and features so as to fit as wide a range of applications as possible. This also provides a growth path for the user should his requirements change.

Making a system reconfigurable by capacity is generally more consistent with high modularity than making it reconfigurable by feature. Nonetheless, features which involve substantial amounts of hardware, such as curved surface generation, are candidates for design as options.

ARCHITECTURE CONFLICTS WITH FEATURES

Obstacles

Some architectures lend themselves better to the implementation of certain features than do others. Rarely are the obstacles theoretically insurmountable, but practical system cost and complexity may be unacceptable unless a clever innovation is found to resolve the potential incompatibility. The resolution of the problem may take the form of a restriction of the system capability, the use of an approximate rather than exact method, or, ultimately, the exclusion of the feature in favor of others more compatible with the architecture and more important in the intended application.

Scanline Architectures

The major problem of scanline architectures is the intersection overload problem previously discussed. There are other, more minor, problems related to the maintenance of a stack of scanstripe segments and the resolution of overlaps and translucent objects in the stack. The algorithms for resolving all of the occlusion analytically on a subpixel basis lead to complex hardware.

Nonetheless, a very general technique for "modularizing" a video processor architecture applies. If work is to be shared among processors, it is better to share it by alternating among pixels or scanlines than to subdivide the screen into regions. The loading is much more likely to be even if the load is shared as suggested. In the scanline architecture, it is better to have, for example, one processor do the odd scanlines and the other the even scanlines (in a field) than to have one do the right half and the other the left half of the screen. If the screen is dedicated by halves, an overload may occur if the scene complexity, perhaps the airport in a flight simulation, happens to fall predominately in one half.

The reverse-priority-ordered writing problem of having to write every pixel, whether occluded or not, is partially offset by the relative simplicity of the approach. General translucency is implemented straightforwardly by mixing a portion of the underlying color and intensity with the new object. Anti-aliasing can be performed by accounting for the fraction of the area in the picture element being covered, and mixing according to the fraction; the anti-aliasing can be done over a larger convolution base with a weighted function if required.

The most difficult problem with this method arises from occlusion not being performed on a subpixel basis. Consider a mountain made of green poloygonal faces silhouetted against a bright sky. The sky would be written first into the buffer, then one green poloygon, and then a matching adjacent polygon. The two polygons should meet on an internal edge so as to occlude the sky completely below the top of the silhouette. However, when the first of the mountain polygons is written, the fractional pixels on all the edges will be blended with the background color, the sky. The adjacent green polygon will then be blended with edge pixels having a sky-colored component already included in those pixels. The net effect is that a thin line of sky will appear along the internal edges of the mountains.

One approach to curing the problem is to tag internal edges when they are first written into the buffer so that the first internal edges will completely overwrite the edge pixels rather than blend. However, there are cases of vertices and thin edges where there is more than one internal edge in the pixel. In addition, the internal edges of translucent objects must pick up the background color, so there are more special cases. Overall, the tagging schemes are at best complicated.

Priority-Ordered Writing

To perform priority-ordered writing with anti-aliasing, occlusion must be performed on a subpixel basis. One way of keeping track of the subpixels is to keep a "bed of nails", a bit map with each bit corresponding to a subarea of the pixel. If the bit is set it means that the corresponding nail has been covered; initially all the bits are set to zero. Only one set of color components need be maintained for the whole pixel, with additions made as the nails are filled in. This solves the internal edge problem for opaque faces.

Translucent faces now cause a potential problem, however, because we must decide what to do with the nails when a translucent face is written. If we set the nails for a translucent face, the nails will not be available for the correct occlusion of underlying opaque faces that will be written later. If we don't set the nails for translucent faces, then the internal edges of transparent faces (which might be almost 100% opaque) will not be anti-aliased correctly. In any case, a separate translucency factor will have to be stored, adding to the nails and color components already needed for each pixel.

The algorithm to treat the problem of top down translucency is somewhat more tractable than the problem of internal edges in reverse-priority-ordered writing. The bed of nails is a basic mechanism for detecting internal edges in any combination without having precomputed tags. Nonetheless, there seems to be no perfect solution under all combinations of multiple overlays of translucency and internal edges.

Note that the presence of translucent objects also diminishes the overload resistance of priority-ordered writing. If all objects were opaque, only pixels intersected by visible edges of surfaces would receive more than one contribution requiring a write cycle. A total of one plus the intersected fraction times that screen size would have to be generated to make a frame. The fraction of edge pixels is small (less than 0.3) so the total amount of processing (assuming skip over is negligible) is quite predictable. Translucency adds a potentially large and variable amount of writing to the processing, considering a cloud or smoke puff covering the whole screen. Some restriction must be placed upon the amount of translucency in the scene to protect the system from such overloads.

Distance Sorting

Z-buffered architectures have difficulties with anti-aliasing and translucency implementations. To correctly occlude and suppress aliasing in the image, operations must be performed on a subpixel basis. Since the distance sorting is performed on discrete sample points, the straightforward approach would be to adopt sorting on a subpixel basis. To accomplish this, the distance and color components of each subpixel must be stored. The expense of generating and storing the information for more than one or two subpixels seems prohibitive with present technology, so it seems that Z buffered systems must be relegated to applications where image

quality is not important, or where the required resolution is low. A possible approach to improving the image quality is to store the Z data analytically for small blocks of the screen, but this would require sophisticated processing algorithms.

Translucency presents the problem of storing a number of overlaying faces that cannot be resolved until the entire scene is built. Building the stack of data at each pixel is prohibitively expensive, so translucency cannot be handled within the normal structure of the system. Rather than abandon the feature entirely, it is possible to separately prioritize the translucent objects and add them after the rest of the scene has been built. It is easiest to add the translucent objects in reverse-priority-order, with the additional step of checking distance for occlusion of the objects by previously written opaque faces. The translucent faces will have the same internal edge problems as ordinary reverse-priority architectures.

CONFLICTS AMONG FEATURES

Textured Translucent Surfaces

Any operation which must be performed on each picture element will require additional pipelined hardware, and it will therefore tend to be expensive to implement. This is true for operations such as smooth shading, smooth coloring, and haze fading, but texture is especially expensive because the computations are more complex. To save texture hardware it would be desirable to only put texture on visible pixels. For example, a Z buffered system could save a pointer to the texture mapping plane equation for each sample point which could be used along with the stored Z data to compute the texture modulation just prior to display. There would be some defect in the anti-aliasing of edge pixels, but no worse than the regular Z buffered images. Similar schemes can be worked out with other architectures.

The addition of translucency complicates matters. With the possibility of overlapping textured translucent faces, texture must be generated prior to the frame buffer input and the expense must be borne. To allow use of texture generation upon output in a system with translucency, restrictions must be placed upon the use of texture. One possibility is to prohibit the use of texture on translucent surfaces and to store an attenuation factor for the texture modulation on the underlying surface. Another approach is to only allow translucency for model switching and to keep the outlines of the objects similar so that a texture may be applied using the Z value from either model; no texture is allowed on models that may be switched completely out.

Prioritization of Curved Surfaces

There are several problems in integrating curved surfaces into the cited architectures. General curved surfaces have concave features that are difficult to prioritize by methods other than Z buffering. A restriction to convex pieces solves this problem, at a significant loss in generality.

If a large curved surface, such as a terrain representation, must be subdivided for any reason, there will be a problem in matching the edges of the curved patches when the patches must change to different levels of detail. This can be avoided by restricting level of detail changes to small curved features added on top of underlying surfaces. However, this will limit switching effectiveness and also generally require the computation of the intersection of the two curved surfaces, a much more complicated requirement than bounding the surfaces by planes.

SUMMARY AND CONCLUSIONS

This paper has discussed two geometric processing architectures and four video processing architectures in the context of a half dozen or so image generator features. Each architecture has strong and weak points as summarized in Table 1.

Architecture	Strengths	Weaknesses
Pipelined Geometric Processor	Throughput	Inflexibility, complexity
Parallel Programmable Geometric Processor	Flexibility, modularity	Throughput
Scanline Video Processor	Requires little memory	Scanline overload, priority-overload
Reverse-Priority Video Processor	Easy translucency	Internal edge occlusion
Priority-Ordered Video Processor	Overload resistance	Translucency implementation, large memory required
Distance Sorting Video Processor	Easy prioritization	Aliasing, translucency implementation

Table 1. Image generator architecture characteristics.

For any particular application, the strengths and weaknesses must be weighed. For example, if data base flexibility is important but image quality is not important, the distance sorting architecture is suggested. On the other hand, every architecture has weaknesses. The designer does well to identify the weaknesses as early as possible, and to do the best possible to minimize them through innovative design or appropriate operational restrictions. Users should find out what the weaknesses are and assess those weaknesses in the light of the application requirements.

For high performance flight simulation, the author's opinion is that within the next two or three years, reduced costs of processors and memories will make a parallel programmable geometric processor with a priority ordered video processor the architecture of choice. This conclusion is based upon the belief that overload resistance and image quality are extremely important in this application. The disadvantages of translucency appear tractable with clever design, and in any case translucency is a feature with specialized application where some limitations can be accepted.

More generally, an important theme of this paper is that image generator design must deal with a set of features that must work together. Features new to the art such as photographic texture or curved surfaces must be combined with all the other features required to make a system that meets a given set of requirements. Users should therefore approach innovative features with some skepticism; the tradeoffs must be understood to find out if they meet the user's requirements.

REFERENCES

1. Schachter, Bruce J. (ed.), Computer Image Generation, John Wiley & Sons, New York, 1983, Chapters 3 and 4.

ABOUT THE AUTHOR

Roy Latham received a B.S.E.E. and a B.S. in Aeronautics and Astronautics from M.I.T. in 1970, an M.S. in Applied Mathematics from S.U.N.Y. at Stony Brook in 1974, and an M.S. in Computer Science from the University of Santa Clara in 1983. He joined Grumman Aerospace Corporation in 1970, where he worked on the development and testing of aircraft navigation systems. In 1978 he joined the Link Division of the Singer Company where he worked as a visual system project engineer and is presently assigned to the development of advanced features for image generation. The author is a licensed professional engineer and a U.S. Patent Agent; he has written five publications and holds a patent in the field of navigation.