CIG SCENE REALISM: THE WORLD TOMORROW

Michael A. Cosman, Robert L. Grange

Evans & Sutherland Computer Corporation

Salt Lake City, Utah

ABSTRACT

In recent years there have been remarkable advances in the rendering of realistic imagery by sophisticated software running on ultra-high-performance compute engines. Technology now makes it possible to incorporate these exotic lighting, shading, and texturing processes into a true realtime computer image generation (CIG) system. This paper reports on work being done to combine advanced rendering algorithms and historical simulation capabilities into a new open system that provides the best of both worlds. Particular emphasis is given to improving edge quality, texture sharpness, lighting and shading flexibility, and the way both opaque and transparent surfaces interact in a dynamic visual scene. The system incorporates many new rendering capabilities that have never before been accelerated in hardware. Higher performance, lower development and recurring costs, and widely scaleable price and performance can be achieved through adoption of industry standards in processors, operating systems, and graphics application program interfaces (APIs). A modular approach allows system configurations for workstation or image generator applications.

ABOUT THE AUTHORS

Michael A. Cosman is a Research Scientist working with the Architecture group at Evans & Sutherland Computer Corporation. He has been involved in the development of CIG algorithms and architectures at E&S for over twenty-two years. He has authored or coauthored numerous technical papers for I/ITSEC and other simulation conferences. Mr. Cosman holds a bachelor's degree in physics from Brigham Young University.

Robert L. Grange is a Product Director working with the Graphics Systems group at Evans & Sutherland Computer Corporation. He has been involved in engineering management, business development, and technical marketing in the simulation industry for the past nine years. Mr. Grange holds a bachelor's degree in electrical engineering from Brigham Young University.

CIG SCENE REALISM: THE WORLD TOMORROW

Michael A. Cosman, Robert L. Grange

Evans & Sutherland Computer Corporation

Salt Lake City, Utah

INTRODUCTION

Historically, the market for computer-generated graphics has been characterized by two very different performance paradigms. In the deterministic realtime simulation world, the approach is to build systems that do a few very important things well, all “at speed,” in predictable mixes, and geared toward a fairly tight performance specification that doesn’t permit frame-time variability. In contrast, the workstation world wants the richest possible smorgasbord of graphics features and is willing to settle for significant performance hits that accrue as users pick and choose features and modes. Computational frame time is a “dependent variable,” and users live with highly variable update rates that seldom approach real time.

Most graphics system architectures have traditionally been driven by applications that trade off image quality and processing time. General-purpose graphics workstations achieve high image quality and special effects by using a variable update rate that allows as much processing time per frame as required. Consistent, deterministic update rates can only be approached by significantly derating specified capacities, thus decreasing the feature density and the simultaneous effects that can be processed. Deterministic realtime applications have typically required special-purpose image generators in order to provide simultaneous synthetic-environment-rendering capabilities, including a high level of realism, fully interactive, predictable, realtime performance, simulation of a wide variety of sensors, static and dynamic nonlinear image mapping, calligraphic lights, and correlation of visual, sensor, and nonvisual environments.

Over the last decade, general-purpose processors have increased in performance at an accelerating pace, and application-specific integrated-circuit (ASIC) and memory technologies have allowed continual miniaturization of graphics-processing circuitry. While general-purpose workstations aren’t powerful enough to perform all of the simultaneous requirements of a realtime image generator in addition to traditional nonrealtime rendering, a common modular architecture with different configurations can meet the needs of both markets.

This paper discusses an architectural approach that uses maximum commonality of hardware and software and builds on a foundation of industry standards to provide the best of both worlds. Particular emphasis is given to innovative improvements in image quality, texture sharpness, realistic lighting, pixel-rate shading, and transparency. The approach provides the traditional image generator features listed above, including terrain morphing, layered fog, global texture, and depth-complexity management along with workstation-class shading flexibility, reflection and environment mapping, OpenGL™ graphics compliance, and high-quality order-independent scene processing. This architecture also incorporates many new rendering capabilities that haven’t been accelerated in hardware before.

ARCHITECTURAL CHARACTERISTICS

This radically new modular architecture allows various system modules to be combined either as a workstation or as an image generator. All configurations will have the same feature set, user interface, 2D windowing, and 3D graphics acceleration and will run the same or different applications. The primary difference is the inclusion of a realtime module and an operating system in the image generator configurations that allow deterministic realtime performance as well as nondeterministic and mixed-mode operation as required.

Benefits of Standardization

A major shift is occurring in the computer industry that is forcing a change from a vertical integration model to a more horizontal one. Where vendors traditionally have developed their own proprietary processors, operating systems, I/O, graphics acceleration, etc., many are now using industry-standard processors, busses, operating systems, and graphics application program interfaces (APIs) to the largest extent possible. In graphics and other applications, development is instead focused on specific areas of competitive product differentiation.

The huge global growth of the computer industry has narrowed the pack of core technologies significantly. Those that have emerged, survived, and thrived to become standards enjoy the benefit of huge financial war chests with which to continue technological advancement. By comparison, the few remaining proprietary technologies continue to lose ground as shrinking markets deplete essential development capital even as the complexity of these technologies, both software and hardware, demands increasing resource commitment.

Those who ride the standards wave will reap a continuing technology windfall as the volume market propels development while reducing costs. This new architecture leverages several "best of breed" technologies. Early adoption of these technologies is allowing this new architecture to achieve higher performance, lower development and recurring costs, and widely scaleable price and performance. These technologies also allow increased focus on fundamental graphics performance issues.

Four Winning Choices

The Intel Pentium® Pro will provide the computational horsepower for all general-purpose computing, including the operating system, realtime system, and geometry processing functions. This strategy includes current and future industry-standard bus architectures.

Microsoft's Windows NT™ will be the man-machine interface and the operating system when deterministic realtime performance isn't required. Windows NT™ is rapidly overtaking Unix in all its variations and emerging as *the* 32-bit operating system.

Wind River Systems' VxWorks® will be the operating system for simulation applications that require deterministic realtime performance. VxWorks® has emerged as a proven, mature, high-performance system that has a long and successful track record. VxWorks® runs in the realtime module, while Windows NT™ simultaneously provides the human interface.

OpenGL™ will be the 3D graphics API. The use of industry-standard windowing, graphics, and operating system software means that a large and growing body of applications written with OpenGL™ will drop transparently onto the new architecture and run efficiently.

The balance of this paper reports on a number of advances in rendering technology that provide performance never previously available at realtime rates. First is a discussion of list-priority and depth-buffer rendering architectures in broad terms and an exploration of their relative cost and performance characteristics, including some particularly vexing depth-buffer image-quality problems. Then a new multisample depth-buffer architecture that addresses these issues is discussed. Lighting, illumination, and shading are discussed next, and the benefits of pixel-rate illumination are established. Bump-mapping is introduced, and some important extensions to layered-cloud processing are discussed. Finally, a number of texture-quality issues are explored and important new rendering capabilities are introduced.

TWO RENDERING PARADIGMS

In general, today's graphics systems incorporate two very different processes for rendering 3D imagery.

List Priority

If the application (i.e. the user) is willing to sort the visual environment so that all polygons can be submitted in visual-priority order, the rendering hardware can employ semianalytic processes to filter and antialias scene details. This requires that the environment be *sortable* (not always possible unless some constraints are applied) and that it contain information to assist the realtime sorting process—a significant complication of the modeling step. The runtime benefits of the list-priority algorithm are considerable, however, and such approaches are still used in many applications today.

The list-priority approach is particularly economical because scene buildup is largely an accumulation process. As each polygon is rendered, the system already knows how much of it is occulted by prior scene elements and can compute its contribution to the display video directly. As the polygon is processed, it contributes to an accumulating "occultation mask" that records which portions of the image plane are already covered. After it is processed, nothing more needs to be remembered about it. Furthermore, as portions of the screen are fully covered, they can be removed from consideration as subsequent polygons are submitted—a mechanism that can save significant processing time. List-priority mode can be implemented with an absolute minimum of frame-buffer memory and can produce unsurpassed image quality. Such systems were pioneered in the late 1970s (Schumacker 1980).

Depth Buffer

As visual environments grew in complexity, making them sortable became progressively more difficult and constraining. Meanwhile, advances in technology made some alternative order-independent approaches practical. One, the *z*- or depth buffer, has become somewhat of a standard, and nearly all current graphics systems support it to some extent (Catmull 1974). A depth buffer solves the hidden-surface problem by computing the distance, or depth, from the eye to each scene element at each pixel (or subpixel). As scene elements are rendered, the system saves only the winning (i.e. closest) scene element's contribution at each pixel or subpixel. After all scene elements have been rendered, the pixels or subpixels are converted into display video. One particular benefit of a depth buffer is its ability to render two polygons that pierce each other—a situation encountered regularly in many applications.

A depth buffer first converts the continuous polygonal representation of the scene into pixel- or subpixel-size quanta of color information, each with an associated depth or distance. Image quality is a function of the number of these quanta, or samples, associated with each pixel. Extremely cost-sensitive applications like home video games might use only one sample per pixel, while most realtime simulation applications require at least four subpixels, and significant image quality improvement can be obtained all the way up to 16 subpixels per pixel, where image quality compares favorably with the front-to-back analytic rendering process.

Cost and Performance Issues

The number of subpixels is an important cost driver, because each subpixel must remember, at a minimum, the depth and color of the winning polygon fragment it saves. Considerable additional information is also required if the system is to support a graphics API like OpenGL™, with immediate-mode graphics, multiple image planes, and double buffering. Large memories are expensive and difficult to access at current rendering speeds, so these systems are rarely configured to do more than four subpixels and can't deliver multisample image quality that's as good as sorted mode. Users are forced to choose between poor image quality and the tremendous effort required to sort a complex visual environment into front-to-back order—a task that isn't always possible.

Depth buffers use subpixel distance values to decide what to keep at each subpixel. Inadequate computational precision will result in erratic subpixel "pop-throughs" of scene details that are close together in the depth direction. But users and applications routinely (and necessarily) model surfaces that are *coplanar*—a problem that the depth buffer cannot solve at all. Special mechanisms must be brought into play to force the depth buffer to render the required result. Many of these mechanisms arise from obscure application-specific problems and must be carefully coordinated with the basic depth-buffer process.

Depth buffers have another fundamental problem that affects image quality. Only one scene detail can claim each subpixel, so the only way to represent transparent objects is to disable a portion of their subpixels before rendering them—in effect, shooting them full of holes where more distant scene details can show through. This approach is called *screen door* transparency. The number of levels of transparency is limited by the number of subpixels per pixel, so there are usually only a few discrete levels of transparency, with large steps between them. Where the user intended to see a smoothly varying transparent effect, either from texture or varying vertex values, he sees instead coarse contours of transparency. If the transparency is changing dynamically, he sees distracting "pops" between these few levels.

Transparent surfaces rendered this way interact strangely with other scene details. Whenever a transparent surface occults another transparent surface, the combined effect is usually incorrect and often bizarre. For example, if two polygons that have the same level of transparency are overlapped, the second one will be invisible where it is occulted by the first. This is because they both have the same hole pattern. Imagine a pilot looking through his wingman's canopy at a third jet. The third jet's canopy will be missing—eliminated by the matching transparency of the wingman's canopy.

Furthermore, all polygons that are rendered behind a transparent polygon will suffer a significant degradation of edge quality, because only a portion of the subpixels is available for edge filtering—a problem that gets very bad very quickly with decreasing numbers of subpixels. For example, if transparency is used to model a thin cloud of battlefield smoke, all scene details (including targets) behind the smoke will have poor edge quality, affecting both detection and identification. Small scene details that would otherwise behave adequately can be intermittently hidden and exposed as they move between the holes in a transparent polygon.

A NEW MULTISAMPLE ARCHITECTURE

A unique new architectural approach addresses both the economic and visual issues of multisample depth buffers. Subpixel information in the depth buffer is stored in a compressed form that requires far less total memory than current multisample approaches. The nature and organization of this information support full-speed resolution of complex subpixel occulting geometries, including interpenetrating polygons and overlaid transparencies. The total amount of required frame-buffer memory is greatly reduced and is no longer a significant function of the number of subpixels. Because the number of subpixels has little effect on the cost, size, or complexity of the system, this approach provides 16 subpixels for all configurations, providing excellent image quality.

Area Sampling

The number of subpixels and the algorithm used to choose them have a large bearing on the resultant image quality, particularly edge antialiasing. A typical approach is to distribute the subpixels in a uniform grid over the pixel and choose them with a point-sampling approach; i.e. a subpixel is assigned to a polygon if it lies on the polygon. Point sampling tends to capture subpixels in groups if the edge of the invading polygon is aligned with rows, columns, or diagonals of the subpixel pattern, so it behaves poorest on edges that are aligned with the horizontal or the vertical—critical orientations for most applications. Subpixels are assigned using a form of area sampling—an approach that matches the subpixels more closely with the actual polygon geometry. Area sampling allows a polygon to "invade and conquer" a pixel in a sequence of single-subpixel increments.

Analytic Occultation

As polygons are rendered into each pixel, the new architecture computes an analytic occultation solution, expressed as an intersection edge where the new polygon potentially pierces any previously stored polygon. Most of the time, of course, this process declares either the new or the prior polygon the sole winner of any contested subpixels. When they do occur, polygon interpenetration edges are rendered with the same quality as regular edges. This part of the system also implements a robust coplanarity solution and several other mechanisms that ensure proper handling of things like point lights. There is also a mechanism for skipping large portions of a polygon when there is no chance of its appearing in the related pixels. This recoups much of the pixel fill-rate efficiency of the list-priority approach.

Continuous Transparency

Transparency is now a continuous property defined by a number between 0 and 1, not the highly granular effect achieved by shooting holes in the subpixel mask. This continuous value is used to calculate an accurate cumulative effect during the resolution of pixel shading. For example, if two overlapped polygons each are 40% transmissive, the area of their overlap will be 16% transmissive (the product of 40% and 40%), and the color of this overlap will be the proper blend of the closer and the farther attenuated by the closer. Any of the polygons that occupy a pixel can be transparent. Also, the edge quality of polygons behind a transparency is unaffected by the transparency. The shade of the polygon is attenuated, but it still gets its full complement of subpixels for edge antialiasing. In the formation example, this means that the canopy of jet number three will be properly rendered and that all the polygons of jet number three will be rendered with full edge quality.

The older screen-door form of transparency is supported for compatibility with other systems and for use with fade level of detail, which requires pair-wise complementary screen-door masks.

A PRIMER ON ILLUMINATION

A lot of the visual impact of today's blockbuster movies depends on computer-generated creatures and effects. These in turn depend on advanced illumination and shading processes. As these capabilities migrate into realtime hardware, users will want to understand a little more about how they work.

Light

The appearance of scene details depends on the nature of the light that illuminates them and on how they redirect that light to the viewer. Surface composition plays an important role here because the redirected light is a function of several surface-material characteristics, including color, roughness, and shininess, in addition to the relative geometry of surface, light, and viewer.

Emissive light is created by a scene element itself. Night and dusk scenes are richly populated with point lights and luminous polygons that represent emissive light sources. They light themselves, but not other nearby scene surfaces, so they aren't treated as general illumination sources.

Ambient light comes from everywhere, hits everything, and is scattered equally in all directions. Because its effect doesn't depend on surface orientation, things viewed under ambient light are flat and featureless, with no sense of shape. Ambient illumination represents the cumulative effect of all illumination sources after their light has been bounced around by all the scene elements.

All other illumination sources have positions somewhere in the simulated world, so the light from each of them approaches a surface from a particular direction and travels a particular distance. The surface redirects this light in two important but fundamentally different ways.

Diffuse light comes from a particular light source, hits a surface, and is scattered equally in all directions. Its effect depends on the orientation of the surface to the light source, so things viewed under diffuse light show shape and orientation.

Specular light comes from a particular light source, hits the surface, and is reflected. Its effect depends on the orientation of the surface both to the light source and to the viewer. Specular behavior imparts a strong sense of shape to scene features. This effect can be amplified by a surface parameter called the specular exponent, which controls the "shininess" of the surface. This particular formulation was developed by Phong, and is called the Phong illumination model (Phong 1975).

A complete lighting environment might consist of a number of illumination sources, each contributing to the final shade of scene details. Each illumination source can have its own characteristics, including color, intensity, location, pointing direction, beam width, and range attenuation. The location of an illumination source has a strong effect on the visual behavior of a surface because of the way it affects the relative orientations of the light, surface, and viewer and the range used for attenuation.

Surface Characteristics

Scene elements are modeled with surface coefficients that define how they respond to ambient, diffuse, and specular light and if they emit their own light. In general, the color of a polygon is multiplied by the color of the incoming light and the corresponding coefficients to compute the combined effect, and this must include any polygon color variations due to texture. Specular highlights are reflections, so they retain the color of the light source.

The surface orientation of a polygon is specified by unit-length surface-normal vectors associated with each vertex. The degree of alignment of these vectors with the incoming light determines the amount of diffuse light that is scattered, the amount of specular light that is reflected, and the reflection direction.

Flat and Smooth Shading

First-generation graphics systems employed extremely simple shading and illumination algorithms. Typically, a single illumination source (the sun) was placed at infinity, and the brightness of scene surfaces depended on the angle between each surface and the sun. This brightness was applied uniformly across the surface, so changes in appearance only happened at the edges between surfaces. This approach required only a single illumination computation per polygon and was called *flat shading.*

In 1967 Wylie, Romney, Evans, and Erdahl pioneered the use of *interpolated shading* (Wylie et al. 1967). Shading values are computed for each vertex of a polygon and then interpolated across the interior of the polygon for each pixel. The orientation of the surface at each vertex is defined by a surface-normal vector, and if a continuous mesh of polygons (e.g. the fuselage of an aircraft) shares surface-normal vectors, the resulting surface appearance is “smooth;” that is, the interior polygon edges disappear. The first graphics system that used smooth shading, a capability that has since become universal in the industry, was introduced in the early 1970s.

Limitations of Interpolated Shading

Interpolated shading is typically used with a fairly simple illumination model, usually consisting of just ambient, diffuse, and emissive terms. Most of the time, shading changes due to these three terms vary slowly with surface geometry and are adequately captured by evaluation at the vertices. For years the baseline illumination model in the realtime simulation world has been a combination of emissive, ambient, and diffuse effects, usually with a single directional light source infinitely far away—the sun.

As the lighting environment gets more complex, interpolated shading begins to show its shortcomings. If a light source is close to a scene element, shading gradients can become very steep because some vertices are much farther away from the light than others, and their illumination is more sharply attenuated due to range. For the same reason, the relative direction from the light to each vertex (and hence to each surface normal) can change greatly, resulting in large variations in shading. Specular behavior, which changes rapidly with viewing geometry, will always create steep shading gradients. Since the illumination at any vertex affects the illumination across the entire polygon, the dynamic artifact is large swings in surface brightness as vertices move relative to light sources and specular highlights. Systems that employ interpolated shading with complex lighting environments must subdivide scene elements into nearly pixel-size polygons to suppress these artifacts. Another disadvantage is that if texture is applied to the surface, it will erroneously affect the specular highlight because the specular behavior will be encoded into the vertex shade before the individual textured pixels are computed.

Phong Shading

In 1975 Phong developed a process called *normal-vector interpolation shading*, or Phong shading. He found that much more accurate visual results can be achieved if the surface orientation is interpolated for each pixel prior to the application of the illumination model. With Phong shading, much more elaborate illumination models can be used without distracting visual artifacts. Pixel-rate surface-normal interpolation also makes bump-mapping possible (Blinn 1978).

The term “Phong shading” has unfortunately been used to describe systems that use the Phong illumination model but only do vertex-rate illumination processing. They employ sophisticated lighting models, but fall prey to all the distracting visual artifacts of interpolated-shade approaches.

Figure 1 illustrates the various components of the illumination equation. Each sphere is constructed from 240 triangles, of which about 120 are front faced and visible—a comparatively low budget. Each shows the combined effect of ambient light and the illumination from two nearby light sources—a dim one to the right and above each sphere, and a brighter one to the left and below. The spheres have a fairly high degree of shininess, which is evidenced by the small specular highlights. Both spheres use the same Phong illumination model, but the sphere on the right is rendered with pixel-rate illumination calculations—true Phong shading—while the sphere on the left is rendered with vertex-rate illumination calculations. The network of Mach bands on the left sphere reveals the underlying polygonal facets and errors in the location and shape of the specular highlights. The bright lower-left highlight is missed because no polygon vertex falls within it, and the dim upper-right highlight is exaggerated for the opposite reason. The sphere on the left can be made to look like the one on the right if it is subdivided into about 16,000 polygons.

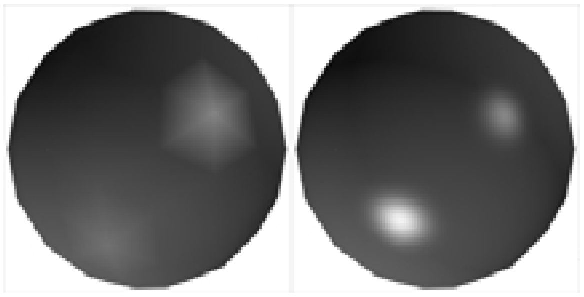

Figure 1: Phong Lighting and Shading

PIXEL-RATE ILLUMINATION CALCULATIONS

A lot of complex calculations are required to produce an image like the sphere on the right. Figure 2 shows some of the geometric relationships involved. As a polygon is rendered into pixels, the system computes the location (e.g. in “world” coordinates) where each pixel ray hits the polygon. It already knows where the viewer and the light sources are. It constructs vectors from the pixel/polygon intercept to each light source and from the intercept to the eye. Floating-point math is used at this point because the dynamic ranges are so large.

Light sources can be directional and are attenuated with distance, so the system first computes the range from the light to the polygon and uses a second-order polynomial to compute the attenuation for that range. The range is reciprocated and used to construct a unit-length polygon-to-light vector. This vector and the light-direction unit normal are used to apply off-axis attenuation if the light is directional (i.e. a lobe or beam of light.) The range from the eye to the polygon is also computed and used to construct a unit-length polygon-to-eye vector. The system has already interpolated the vertex surface normals to a unit-length surface normal at the pixel intercept, including the effects of smooth shading and bump-mapping. It now reflects the eye vector about the surface normal to construct a unit-length reflection vector.

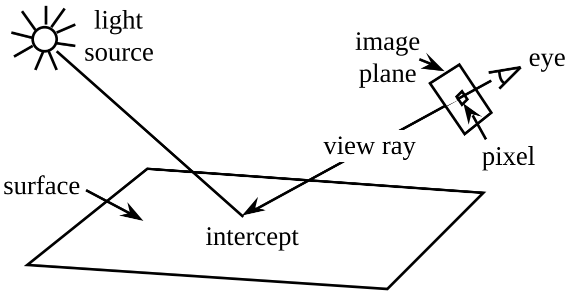

Figure 2: Light/Polygon/Eye Geometry

Figure 3 shows these four unit-length vectors in a “side” view. The diffuse illumination from the light is controlled by $\mathbf{L} \cdot \mathbf{N}$ ($\mathbf{L}$ “dot” $\mathbf{N}$, or the vector scalar product of $\mathbf{L}$ and $\mathbf{N}$), and the specular illumination is controlled by $(\mathbf{L} \cdot \mathbf{R})^s$, where s is the specular exponent. The light has its own color, so the color multiplication and mixing generally involves three components everywhere. The effects of additional light sources are computed and accumulated just as above.

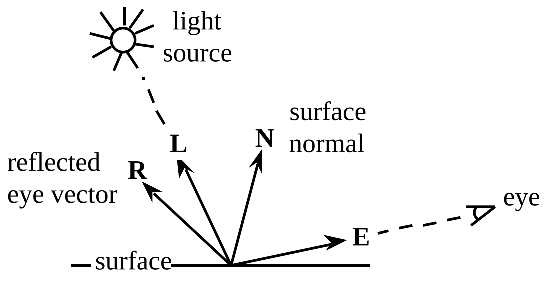

Figure 3: Lighting Unit Vectors

Biting the Bullet

These calculations must be done for every pixel for each light source that affects the polygon. Dedicated silicon that performs these computations at high-performance realtime rates has been developed, and the process has been integrated with other the technologies discussed in this paper. This approach extends the much richer lighting “recipe” historically reserved for the nonrealtime world to realtime applications. The recipe includes a very flexible strategy for combining ambient, diffuse, specular, bump, and texture effects on surfaces, and the lighting environment itself can contain many illumination sources with their own individual spatial and luminance characteristics. The realtime software can optimize performance and flexibility by forming "on-the-fly" associations of illumination sources and polygons in various combinations.

Some Practical Advantages

Two common applications illustrate the power and utility of this new capability. In the civil aviation training community, users demand accurate depiction of the illumination of the runway and airport environs by aircraft landing lights. In the past this illumination problem has been solved with 2D image-plane approaches requiring special hardware. Today this problem is solved more accurately by simply posing light sources on the wings and fuselage of the simulated aircraft with appropriate lobe characteristics and range-attenuation parameters. Since illumination sources can move, steerable lobes are automatic. The effects of multiple landing lights are mapped properly onto all scene details, and this approach accurately portrays the effects of range, surface orientation, and surface-illumination characteristics—things not easily accomplished with a screen-space 2D approach.

Battlefield illumination by flares is a particularly difficult problem for interpolated-shade approaches. Terrain and features underneath a descending flare must be carved into ever-smaller polygons in order for vertex-rate calculations to yield acceptable results. Typically this means that a few database areas where flares will be allowed must be predefined and given special modeling attention. But the pixel-rate illumination approach doesn't depend on polygonal subdivision to control shading artifacts, so it allows flares to be used anywhere in a database without prior special attention.

BUMP-MAPPING

Texture helps a polygonally simple model appear visually complex by supplying opacity and color variations. The illusion falters when texture is intended to represent *shape* variations, because the illumination effects are "frozen" into the texture map and don't respond to changes in the realtime lighting conditions. In 1978 Blinn demonstrated a solution to this problem. His method, called bump-mapping, uses texture to affect the appearance of a surface by modifying its surface orientation, pixel by pixel, prior to the application of an illumination model. By operating through the illumination process, bump texture supplies the same apparent surface-shape detail that would otherwise require thousands or millions of pixel-size polygons. Bump-mapping requires a lot of complex pixel-rate computations, so it has remained in the software domain—until now.

A bump-texture map contains values that define the local "tip" or "tilt" to be applied to the surface-normal vector. Typically, there are two signed values associated with each bump texel that define the forward or backward tilt in each of the two texture directions. A bump map is created by processing a height map into local tilt vectors, usually by computing local height differences in the two map directions. A bump map has levels of detail just like any other MIP map and is interpolated or blended just like any other texture.

When bump texture is applied to a polygon, the user must tell the system what directions to consider as the two texture axes at each vertex. These surface-tangent vectors are transformed along with the other vertex information, clipped if necessary, interpolated to the pixel level, and ultimately arrive in the bump hardware expressed in the same 3D coordinate system that the illumination will be computed in. Bump texture is looked up and interpolated to define the instantaneous surface-normal tip values at the pixel. These scalar values are multiplied by the interpolated surface-tangent vectors to transform them into the coordinate system of the polygon and are added to the instantaneous surface-normal vector, tilting it in the intended direction.

When this tilted surface-normal vector is used in the illumination calculations, the perturbations applied to it by the bump texture cause variations in shading. These variations affect the diffuse and specular illumination terms and create a compelling illusion of surface "bumpiness." The illusion hangs together remarkably well because the texture causes a well-behaved, spatially stable, coherent modification of the pixel-by-pixel surface orientation, much like the one that would be gotten from using actual polygons to model actual bumps. As the surface, the observer, or the lights move about, the resultant shading changes are accurate portrayals of a solid underlying "physical model." The illusion is so compelling that the viewer usually ignores the lack of silhouette roughness in bump-mapped objects.

Figure 4 shows how a bump map changes the appearance of the 240-polygon sphere discussed earlier. The bumps are clearly "innies," and they tessellate continuously over the surface of the sphere in spite of the relatively coarse underlying polygonalization. Because the vertices of the sphere properly share surface-normal and surface-tangent vectors, there is no hint of internal polygonal boundaries. The position and brightness of the two specular highlights are properly represented, and the bump motif itself is perspectively compressed at the periphery of the sphere just as real polygonal bumps would be.

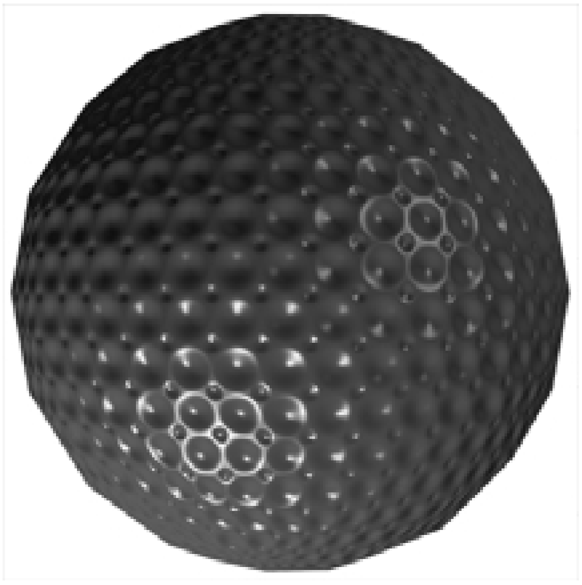

Figure 4: Bump-mapped Sphere

Since bump texture has its own MIP level of detail, the bump illusion functions over wide ranges. Several bump textures can be combined on a surface, just like any other texture. Bump texture can be given texture motion to create compelling dynamic effects like lakes or oceans. It can also be sharpened by any of the static and dynamic sharpening modes discussed in this paper.

LAYERED CLOUDS

In the flight-simulation world, one important requirement is the accurate depiction of the visual effects of ground fog, layered fog, and layered clouds. A number of approaches have been used over the years to simulate layered clouds. The use of a textured, transparent polygon is an obvious first hack and works tolerably for a cloud layer viewed from a distance. It does, however, pose a large pixel fill-rate burden and a potential hit on image quality, especially if the system can't support a large number of smoothly varying transparency levels. More important, it can't provide the visual effects of approaching, entering, and operating inside a cloud layer of finite thickness, since the visibility "attenuation" is a fixed property of the polygon and not an exponential function of actual view-ray/layer interaction. The visual effect doesn't respond properly to either range or view angle and never overcomes the strong impression that this is just a polygon, not a distributed, layered atmospheric effect.

Some advanced image generators have incorporated a more robust approach that uses dedicated hardware rather than transparent polygons. A cloud density profile specifies the visibility range at a number of altitude changepoints between the ground and the stratosphere and is used to compute the net obscuration due to clouds or fog along each pixel view ray. Cloud layers are implied in those altitude regions with restricted visibility ranges, and the eyepoint enters, traverses, and exits these regions in a well-behaved, continuous manner. The approach uses the same pixel-rate exponential formulation that regular fog uses, so it provides continuous and imperceptible gradations of visibility. Layered cloud effects are automatically added to all scene details as they are rendered, without any performance penalty and without the fill-rate burden of first processing a large, transparent polygon.

The approach described above provides extremely accurate visual results, but it doesn't provide for lateral variations in either cloud color or cloud density. A way to provide both of these has recently been developed. Cloud *color* texture provides important spatial cues by providing a sense of proximity, direction, and speed. Cloud *density* texture provides the visual effects of thin or patchy layers. Both these effects can be used simultaneously on a cloud layer, and the full cloud model can include several of these layers. Figure 5 shows a runway motif under a thin ground-fog layer that exhibits both color and density modulation.

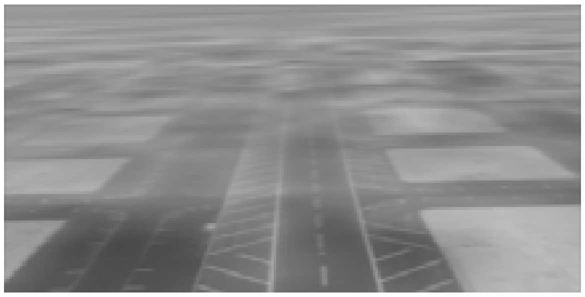

Figure 5: Patchy Fog Layer

Textural effects are applied to a cloud layer in a way that allows the pilot to gradually and progressively enter a layer while the textural cues smoothly transition from a "distant" to a "local" effect. There is no sense of a "texture plane" that is suddenly penetrated, and there are no sudden changes in the visual effects that would be associated with such a transition. While the observer is inside a layer, local visibility is still modulated by cloud texture, providing a natural scud effect and accurate obscuration of formation aircraft, for example. The effects of different cloud layers are computed and concatenated properly so that layers don't interact in unexpected or unnatural ways. For example, a pilot can look through several patchy cloud layers and see the proper parallax effects of the different layers and the ground moving by. As before, there is no fill-rate hit because there are no cloud polygons to render.

IMPROVING TEXTURE

Texture has become the main medium for conveying the visual message. A complex texture motif applied to a few polygons can represent a realistic, detailed structure or vehicle. Texture is used to vary intensity, color, and transparency across a polygon, to cut or crop a polygon to a sharp-edged but irregular shape, and even to control the selection of other textures across a polygon. Texture capabilities have been a major area of industry R&D.

For the first time, the somewhat diverse texture requirements of the simulation and workstation graphics worlds have been integrated. This new technology provides a full set of OpenGL™-compliant texture modes while retaining important simulation-specific capabilities like global terrain and edge-contour texture. Texture antialiasing is now handled by pixel-rate analytic computations of the instantaneous texture gradients—an approach that properly supports texture stretch and shear and makes possible some valuable new rendering modes. Many of these modes are aimed at giving texture the sharpness and perspective behavior formerly achievable only with polygon edges.

Texture Sharpness On Oblique Surfaces

Premature loss of texture detail on oblique surfaces is a fundamental computer-graphics problem that has existed since texture was discovered. A number of software solutions have been explored that were either inadequate or intractable.

The Problem

Aliasing is a general term applied to a wide variety of image quality problems. The term aliasing refers to high-frequency image content masquerading as low-frequency stuff. *Texture aliasing* occurs whenever adjacent screen pixels step over or miss adjacent surface texels—that is, when the projected pixel footprint gets larger than the texels that are on the surface of the polygon. When this happens, some texels are entirely missed by the pixel-computation process. Their intermittent presence or absence shows up as scintillation, breakup, or crawling of the texture motif.

Figure 6 illustrates this effect graphically. It shows a repeating runway-like texture motif that continues for some distance on a horizontal ground plane. The horizontal field of view is approximately 50 degrees—typical of an out-the-window display. Texels in the foreground are about the size of pixels, so the motif is accurately displayed. Farther away, texels get smaller than pixels and some of them get missed by the rendering process. Near the horizon, so many texels are skipped that the motif is unrecognizable. Under motion, all these artifacts scintillate wildly, making the scene unusable.

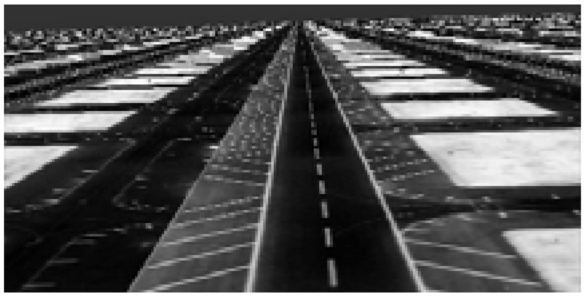

Figure 6: Texture Aliasing

To prevent texture aliasing, graphics systems employ a multiple-level-of-detail strategy called MIP texture (Williams 1983). Each successive texture level of detail consists of fewer, larger texels that cover the same area of the polygon. Typically, for each coarser (lower) level of detail there are one-fourth as many texels, each twice as long and twice as wide as those at the previous (higher) level of detail. The value of each coarser texel is derived by filtering the finer texels it replaces.

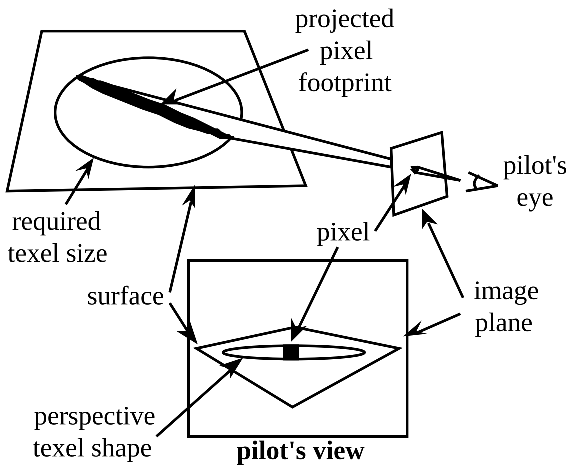

Figure 7: Projected Pixel Geometry

Figure 7 helps illustrate the geometric relationships involved. As the polygon is scanned into pixels, the footprint of each projected pixel is cast onto the polygon and measured. Unless the polygon is being looked at head on, the footprint will be stretched in some direction. If the long dimension of the stretched footprint is *smaller* than a texel, the underlying texels can be used directly to color the pixel. As the long dimension of the footprint gets *larger* than a texel, the system switches or blends to the next coarser texture level of detail (and its correspondingly larger texels) to find the underlying texels that are larger than the footprint. This process continues through all the available texture levels of detail as the projected pixel footprint gets larger. The width of the projected pixel footprint is a function of range, while the length of the footprint is a function of both range and angle of incidence.

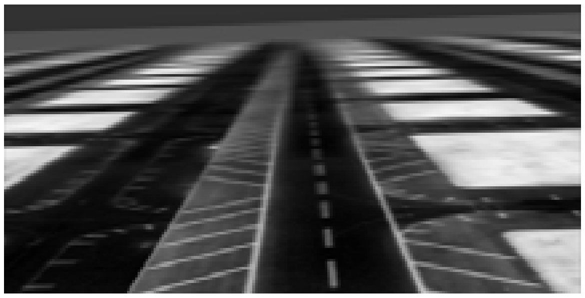

Figure 8: Runway With MIP Texture

Figure 8 shows the same runway motif rendered with a MIP texture map. The artifacts are gone, but a new problem is now evident. Textural detail is progressively lost in the background and is entirely missing near the horizon. It doesn't alias, but it still doesn't work.

The problem illustrated by Figure 8 is fundamental to the texture-rendering process. Since MIP maps generally contain square or nearly square texels, the level of detail that will be selected for display is the level where the texel size is approximately the length of the pixel footprint. This is so pixel steps in the footprint "length" direction won't skip over texels and cause scintillation or aliasing. As viewed on the display, these texels will appear to be approximately a pixel wide in their perspectively "squished" direction, but will be *more* than a pixel long in the other direction. They will, in fact, be *many* pixels long, in the same length-to-width ratio as the original projected pixel footprint. As a result, the texture motif will exhibit pixel-size detail in its perspectively compressed direction, but become extremely fuzzy in the orthogonal direction.

Texture detail *in the horizontal direction* is progressively lost for pixels nearer and nearer the horizon. The problem is dramatic, because surfaces don't need to be very oblique before pixel footprints become fairly elongated. The horizontal degradation shown in Figure 8 will *always* trash the top half of the image *no matter what the altitude*, because the effect depends only on the angle of incidence of each pixel view ray with the surface.

The projected pixel footprint aspect ratio is the reciprocal of the sine of the angle of incidence of the view ray and the surface. At the horizon this value is infinite; at 5 degrees below the horizon, the footprint aspect ratio is over 11; at 10 degrees below the horizon (about half-way to the bottom of the image) the ratio is still nearly 6. For this typical out-the-window situation, most of the runway texture visible to a pilot who is landing is unacceptably fuzzed out. This is one reason why civil airline users insist on a polygonal representation of the runway surface and markings—a textural representation would be unusable for critical scenarios.

The Kludge

In certain specialized situations, modeling strategies can be used to partly compensate for this fundamental problem. For example, if the driver of a car is constrained to remain on and drive down a road, the road can be decorated with a texture motif that has been stretched along the direction of travel. It will function better if the observer looks down the road and worse if the observer looks across the road—a compromise that may be acceptable in some cases.

The Solution

A robust process for correcting this problem in its most general form has been developed. The solution is applied by dedicated hardware within the pixel-processing portion of the image generator. This hardware determines, on a pixel-by-pixel basis, an optimal texture-sharpening strategy. Because the sharpening process uses pixel clock cycles, the strategy is only applied to polygons marked for sharpening, is used only on those pixels that can be sharpened, and is subject to global control by the realtime software. A typical strategy is to use excess fill-rate capacity to provide sharpening—a useful employment of the leftover frame time that otherwise goes unused in many applications.

The process can be applied to any polygon, at any orientation, with repeating or nonrepeating texture and is compatible with all types of texture, including global terrain and bump texture. The algorithm takes into account the size and shape (stretch or shear) of the applied texels and the required and available texture levels of detail. It won't waste time trying to sharpen texture where the projected pixel footprint is mostly square or if it is already using the highest-resolution texels available. When the sharpening mode is active, the system is able to relax the MIP level of detail, typically so that texels the *width* of the projected pixel footprint (rather than its length), can be used.

Figure 9 shows this process in action, and the improvement in texture sharpness is dramatic. The runway motif is now crisp and sharp clear to the horizon, with no staircasing, breakup, distortion, or scintillation. A pilot landing on this runway will see uniform image sharpness from his final turn to touchdown and taxi. When he gets to the terminal area all the textured polygons of the 3D structures will be similarly improved and hence more realistic. Modelers will use far fewer polygons modeling such scenes because now texture can do the job and provide true photorealism. They won't need arcane strategies like highly stretched texture, won't have to constrain where the viewer can look or travel, and won't need to compromise database accuracy to hide edge-on fuzziness.

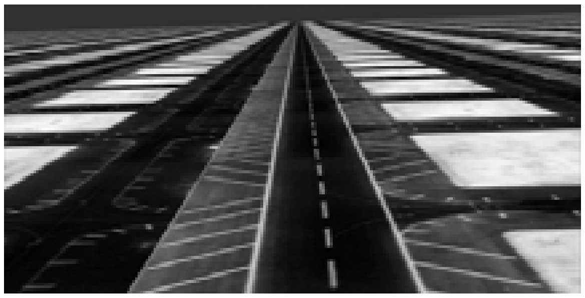

Figure 9: Texture Sharpening

Constructing Sharp Edges

Texture is often used to cookie-cutter polygons into complex shapes—a way of trading polygon load for fill-rate load. Another related use is the construction of sharp-edged color motifs like lettering on signs. With MIP texture maps, the behavior of such motifs in the distance is quite acceptable. However, when the viewer gets close to these motifs, texels get much larger than pixels, and the sharpness of the color or occultation boundary is lost. OpenGL™ includes a sharpening mode that partially addresses this need. An alternate mode maintains the sharpness of these boundaries even when texels become many pixels large. Unlike earlier "contour" texture strategies, this mode doesn't require a special map or intensive map preprocessing, but works with ordinary MIP textures. The process guarantees that the sharpness of the resulting edge is properly coordinated with pixel size to prevent crawling or scintillation. Figure 10 shows a map rendered as an intensity modulation and then in the sharpen mode. Even coarse motifs are captured and rendered with surprising sharpness using this feature.

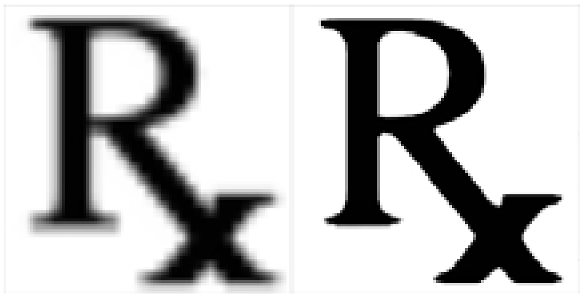

Figure 10: Sharp-Edged Texture

Improving Texture Stylization

Another texture improvement was originally born out of the need to provide better global terrain texture. Many applications require the decoration of large areas of terrain skin with a continuous photographic or photorealistic motif that correlates with real-world source material. Since few users can afford to acquire, process, store, and use small texels over large areas, these databases are typically built with fairly coarse texels of perhaps one to several meters. This means the user will often be looking at terrain texels that are much larger than his pixels.

When texels get lots bigger than pixels, two visual problems arise. The texels get fuzzy and don't provide adequate optical flow (a situation that can be improved by the addition of a generic fine-detail texture), and the bilinear blending used to render the texture motif causes distracting stylization errors. Figure 11 shows this effect on a typical photoderived motif where the texels are so large they are just barely adequate to capture the details of an urban area. Much of the detail in this area runs obliquely through the texture motif and is highly linear because it represents roads, sidewalks, houses, and other man-made things. Bilinear blending renders the diagonal detail as regularly jagged staircases—an effect that is spatially stable (i.e. it doesn't alias under motion) but aesthetically jarring.

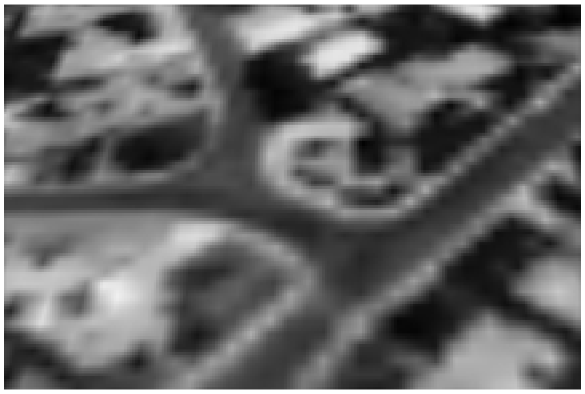

Figure 11: Bilinear Texture Blending

The stylization problem is fundamental to the bilinear-blending process. The texture lookup is done by casting a view ray from the eyepoint through the pixel and out onto the textured surface. The integer portions of the 2D texture coordinates are used to find the texel "quadrangle" bounding the view-ray intercept. The fractional portions of the coordinates establish the precise position within this quadrangle. The texture value used to shade the pixel is a bilinear blend of the four bounding texels, using the fractional parts of the coordinates as the weights. In general, all four texels contribute some effect to the pixel, except when the view ray hits one of the quadrangle boundaries.

The new approach includes a special *biplanar* blending mode that can be selected instead of bilinear blending. This mode considers the bounding quadrangle to be two triangles, where the direction of the diagonal cut is selected by a bit stored with each texel. The filter process looks at the direction of the diagonal cut and which triangle the view-ray lands in and performs a planar interpolation from the three texels of the bounding triangle. The *rule bit* for each texel is computed when the texture is acquired and processed into a MIP map. In general, the rule bit is set to select the diagonal cut that makes the center of the quadrangle match the source material most closely. Full-color texels contain a single rule bit that is used with all three color channels. All the texels at every MIP level of detail contain rule bits.

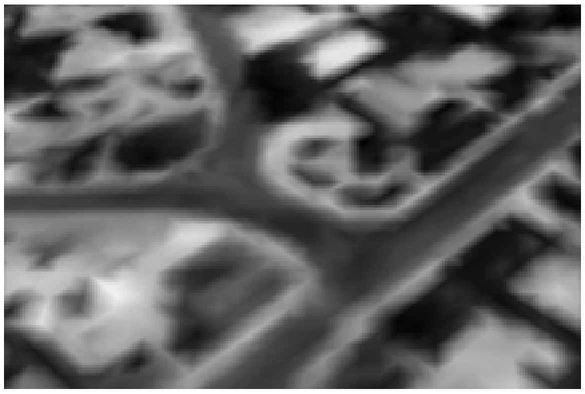

Figure 12: Biplanar Texture Blending

Figure 12 shows the scene in Figure 11, using the same texture map but with the rendering mode set to use the rule bit. Most of the diagonal features are rendered with smoother, more continuous edges and many small details that were broken up by the bilinear stylization seem to have congealed back together. The rendering as a whole seems "quieter" and more solid, even though numerical analysis of the image reveals that it has *more* contrast and detail. In addition, linear motif details that are aligned *with* the texture aren't degraded by the use of the biplanar mode. While originally developed for global terrain texture, biplanar blending can be used with any texture, repeating or not, on any polygon at any orientation. There is no performance hit, since this mode runs at full speed. The rule-bit generation process has been kept in software to optimize flexibility.

CONCLUSION

This paper has highlighted work being done to create a modular, scaleable architecture that can be used in both workstations and image generators. This architecture combines the user friendliness of a graphics workstation with the determinism of a realtime image generator and is built around industry-standard hardware and software. Emphasis has been on innovative new features that solve problems or minimize shortcomings in current architectures as well at those that enhance scene realism. These features include high-resolution area-based multisampling, continuous transparency, pixel-rate Phong shading, multiple illumination sources, bump texture mapping, textured layered clouds, texture sharpening on oblique surfaces, sharp-edged texture, and improved texture stylization.

REFERENCES

Blinn, J. F. 1978. Simulation of Wrinkled Surfaces. *SIGGRAPH 78*, 286–292.

Bui-Tuong, Phong. 1975. Illumination for Computer-Generated Pictures. *CACM,* 18(6):311–317, June.

Catmull, E. 1974. A Subdivision Algorithm for Computer Display of Curved Surfaces. *Ph.D. Thesis, Report UTEC-CSc-74-133,* Computer Science Department, University of Utah, Salt Lake City, Utah, December.

Schumacker, R. A. 1980. A New Visual System Architecture. *Proceedings of the 2nd I/ITEC Conference,* November.

Williams, L. 1983. Pyramidal Parametrics. *SIGGRAPH 83,* 1–11.

Wylie, C., G. W. Romney, D. C. Evans, and A. C. Erdahl. 1967. Halftone Perspective Drawings by Computer. *FJCC 67,* Thompson Books, Washington, D.C., 49–58.