A reprint from
American Scientist
the magazine of Sigma Xi, The Scientific Research Society

The Race for Real-time Photorealism

The coevolution of algorithms and hardware is bringing us closer to interactive computer graphics indistinguishable from reality

Henrik Wann Jensen and Tomas Akenine-Möller

Ever since the emergence of three-dimensional computer graphics in the early 1960s, graphics specialists have dreamed of creating photorealistic virtual worlds indistinguishable from the real world. Product designers, architects, lighting planners, gamers and scientific visualization pioneers have craved real-time reality on a chip; hardware designers and algorithm writers have made spectacular progress, as one can see by looking over the shoulder of teenagers playing the latest games *(facing page)*. But as we will see, the computational challenges that remain are immense. Implacable evolutionary progress has been made by software engineers in devising ingenious algorithms, and generations of hardware have been invented to traffic and execute the calculations, yet the sheer scale of the computational task keeps the goal of real-time photorealism at some distance over the horizon. Most office computers consume a small sliver above zero percent of their available computational cycles for routine work; the billions of calculations per second that are available on a modern multi-processor desktop computer are simply not required to process spreadsheets. Compare that to the overwhelming task of the most advanced 3D applications, churning through the calculations required to produce a scene using the latest algorithms, including the tracking of billions of simulated photons through a scene, and even the tracking of the simulated light penetrating the scene's surfaces to achieve the perfectly convincing photorealistic image.

Such renderings can take today's fast machines hours to produce a single frame.

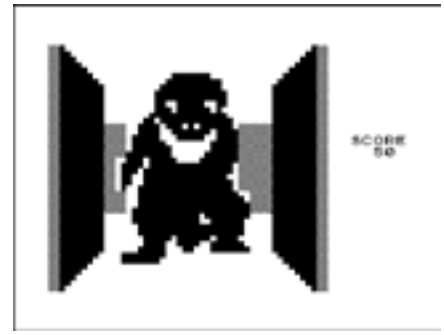
Elite gamers clamor for no less than 60 frames per second. Why so many? Because the pursuit of real-time graphics is driven by the desire for not just visual accuracy but also interactivity. Real-time scenes are created to be interacted with. The television standard of 29.97 frames per second is comfortably convincing for passive viewers; real-time applications, such as gaming and military cockpit simulations, must operate at the speed of human reflexes.

As participants in the enterprise of creating photorealistic graphics (one of us having a research emphasis on greater speed, the other on greater realism), we'll review the kinds of computations required, the schemes that have been invented to moderate the heavy computational chores, and the parallel world of hardware development to support the calculations. The hardware and software of photorealistic graphics have coevolved for several decades. The economics of hardware development, driven mainly by gamers' unquenchable lust for speed, has resulted in affordable graphics cards of awesome power. Computations have been moved from the central processing unit (CPU) of computers to the specialized graphics processing units (GPU) of consumer video cards. The leap in computational prowess then drives the development of greedier algorithms for more convincing realism. This cycle has gone on for decades, blossoming into multi-billion-dollar video card and game software industries.

Henrik Wann Jensen is an associate professor at the University of California, San Diego, where he specializes in realistic image synthesis and the rendering of natural phenomena. He received his Ph.D. in computer science at the Technical University of Denmark. Tomas Akenine-Möller is a professor of computer science at Lund University who works part-time with Intel, specializing in computer graphics and image processing. He received his Ph.D. in computer graphics at Chalmers University of Technology. Address for Jensen: Computer Science and Engineering, 4116, University of California, San Diego, CA 92093-0404. Email: henrik@graphics.ucsd.edu

Making the Scene

Current real-time graphics applications, such as games, represent the complexity of virtual environments by converting scene descriptions into millions of geometric primitives—points, lines, and polygons, usually triangles, connected to form polygonal surfaces. Early games represented 3D scenes using a few hundred triangles; in historical context, the experience of interacting with these 3D environments could be quite compelling, but the appeal had little to do with realism.



3D graphics circa 1981: *3D Monster Maze* on the Sinclair ZX81 with 16-kilobyte memory expansion.

More triangles made for more convincing scenes. An obvious first step in accelerating the rendering of a scene was to optimize the number of triangles required. Scene designers are obliged to make judicious decisions about the balance between polygon count and realism. How much detail is enough? Current GPU hardware can process scenes composed of several million triangles while still reaching

Figure 1. Consumer demand combined with algorithmic artistry and muscled-up hardware have driven computer graphics far toward the long-imagined goal of photorealistic animation. The state-of-the-art animated feature movie *Ratatouille,* released by Pixar Animation Studios in 2007, was produced by an arsenal of about 850 computers hosting nearly 3,200 processors. The average rendering time for each frame of animation was about 23,000 seconds per frame. Today's video gamers want the same visual quality—at 60 frames per second. And they are on the road to getting it, as can be seen in the Electronic Arts 2008 action and adventure game *Mirror's Edge,* which delivers dazzling interactive play at more than 60 frames per second on personal computers. (The image above from that game was rendered offline with additional resolution to achieve print quality.) The authors review the roadmap to a future in which advances in speed and photorealism finally achieve the goal of perfectly convincing interactive computer graphics in real time. (Image courtesy of EA Digital Illusions Creative Entertainment.)

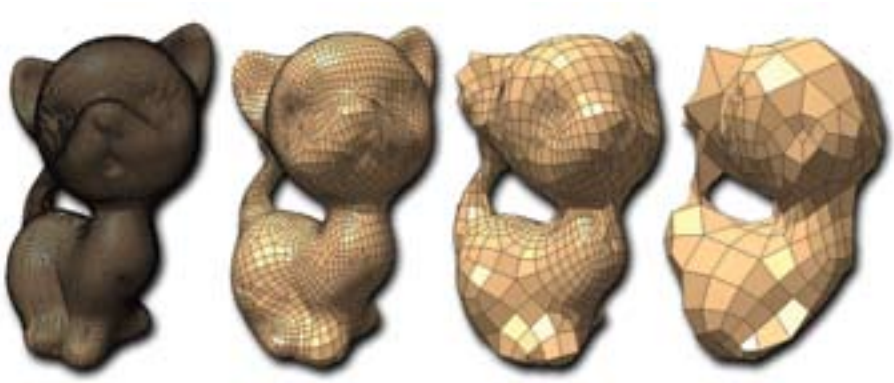


Figure 2. Optimization and approximation are keys to graphics rendering speed. Algorithms optimize how many calculations must be made, and scene elements such as shadows, reflections and even geometry may be approximated, with accuracy surrendered for speed. Above, the polygon count of a 3D model is progressively reduced. For a rendering scheme such as rasterization, which renders individual polygons, the middle models would render much faster than the one on the left. As the distance from the viewer to the cat increases, fewer and fewer polygons can be used with little loss in quality. (Adapted from Daniels, J., C. T. Silva, J. Shepherd and E. Cohen. 2008. "Quadrilateral mesh simplification." *Proceedings of SIGGRAPH Asia* 27(5):1–9.)

the gamer's benchmark of 60 frames per second. If current trends continue, we can expect hardware in the near future that handles hundreds of millions or even billions of triangles with sufficient speed. The question is: how many triangles are required to achieve a photorealistic rendering of a given scene? One of the founders of Pixar Animation Studios, maker of 3D blockbusters from *Toy Story* to *Ratatouille*, concluded that 80 million triangles would be required. It seems that the hardware will soon be up to the job of handling the geometry in real time. However, there is much more to photorealism than polygon count.

Faster with Rasterization

With contemporary hardware and software, the fastest way to *render* a scene (convert the 3D data to a visual image) is *rasterization,* the technique used by today's computer games. An algorithm processes the scene detecting what geometry is visible and what is screened from view (including the back faces of 3D objects facing the viewer). Nonvisible geometry is discarded to speed the calculation, and then the scanner determines which vertices are closest to the viewer. Triangles formed by vertices are painted onto a virtual screen, as shown in Figure 3. The color of each pixel on the screen is determined by the color and surface properties assigned to the triangle, as well as the lighting in the scene. The angles where triangles abut are made to vanish in the image by the neat trick of averaging the color values of adjacent triangles. Color and surface properties (ruggedness, sheen, and so on) are assigned by software instructions called *shaders*. Most commonly, the surface information is assigned using *texture maps,* which are digital images "glued" onto the 3D object. Texture mapping is an art in itself. In the production pipeline of 3D studios, artists specialize in the creation of texture maps to convey, for example, not just the color of an orange, but also the knobbly surface and the waxy shine. An early breakthrough on the road to photorealism was bump mapping, conceived by the computer graphics pioneer James F. Blinn. (It was said of him quite a few years ago, by the graphics hardware innovator Ivan Sutherland, that "there have been about a dozen great computer graphics people and Jim Blinn is six of them." Blinn has made many milestone contributions in the field of deriving convincing images from 3D data.) Bump maps convey details of microfine surface structure without adding to the overall geometry load by telling the renderer to handle local lighting *as if* the surface were bumpy, with the bumps defined by light and dark areas on the texture map.

Let There Be Lighting. And Shadows.

The critical element of lighting in a 3D environment comes from virtual light sources placed in the scene. In rasterization schemes, a few simple equations are used to compute how much light emanating from a light source arrives at a given point on each triangle, and how much of this light is reflected towards the observer.

The earliest 3D renderings had a signature, otherworldly look because they lacked shadows, a critical aspect of visual realism. Rendering shadows with rasterization is straightforward using a technique that employs multiple rendering passes. For example, one can use a shadow-mapping algorithm, where the scene is rendered from the light source into a shadow map in a first pass. The shadow map contains information about all the triangles visible from the point of view of a particular light. In a second pass, from the "camera" point of view, which is different from the light source, the color calculation for each triangle queries the shadow map to see if the triangle is visible from the light source or is in shadow. Adjustments are then made to the color of the triangle to account for the shadow.

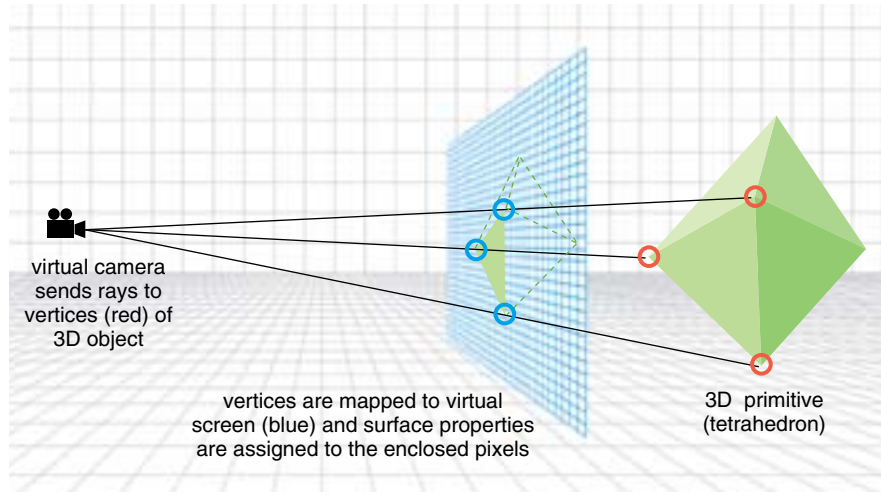


Figure 3. Rasterization algorithms render scenes by projecting rays to the vertices of geometry in the scene, thus defining polygons that are then mapped on pixels. Surface properties assigned to the polygons of the model are then computed and mapped to the pixel screen to create an image.

A decided weakness of rasterization is the rendering of reflections and refractions. Refractions in the real world can be seen as the bending of light when it passes through a transparent medium such as a glass of water. Like shadows, they contribute greatly to realism. For a variety of reasons, reflections and refractions cannot be computed using the triangle-painting technique that is the core strategy of rasterization. Workarounds have been devised to create illusions of reflection and refraction, but the basic problem these lighting effects present has proved intractable for rasterization schemes.

There are other lighting effects that rasterization fails to capture. In real scenes, color bleeding occurs when diffuse surfaces are illuminated by indirect lighting. For example, in a white room with a red carpet, the carpet casts a subtle red glow onto the white walls. Another elusive phenomenon is caustics; when real light is refracted or reflected through a transparent medium, focusing effects can produce blooms of intense brightness. An example of caustics is the shimmering waves of brightness seen on the bottom of a swimming pool. Subsurface scattering is a particularly notable recent development on the road to photorealism that is confounded by the limitations of rasterization. Real materials often have a degree of translucency on their surface. Think of how light penetrates jade. As light crosses the material's surface, it is scattered, some inward, some back out. The distinctive visual quality of subsurface scattering accounts for the appearance of, among many other things, human skin, and the difficulty of accurately reproducing it accounts for the notoriously unconvincing appearance of many 3D renderings of faces. At present, rasterization is the main player in real-time graphics, but in the opinion of many, for reasons that include its limitations at handling advanced lighting effects like those just mentioned, it will not be the road to real-time photorealism.

Racier Hardware

Hardware is part of the answer. Better graphics is the main reason why average consumers want faster computers, and one of the key technologies driving real-time graphics is the use of specialized graphics processing units that can process and display vast amounts of geometry rapidly. GPUs achieve their performance by using a high degree of parallel processing, in which the task of rendering a scene is divided

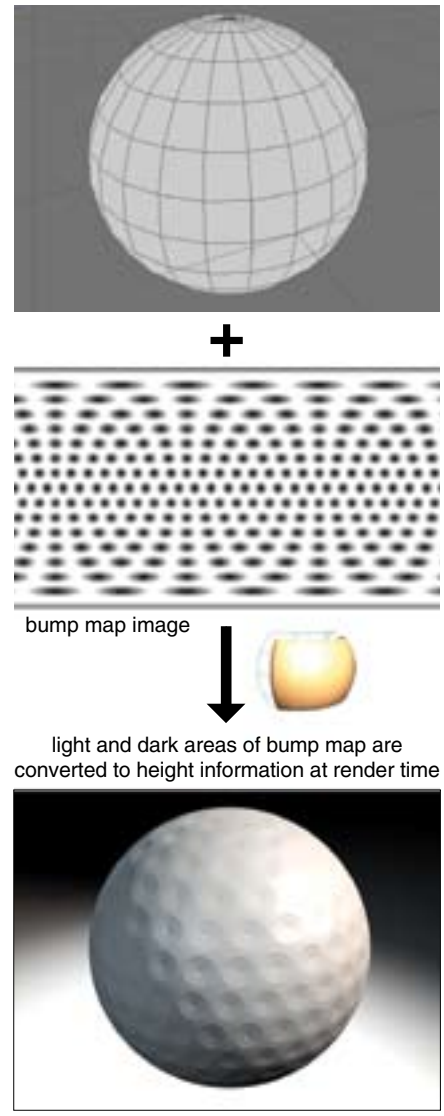
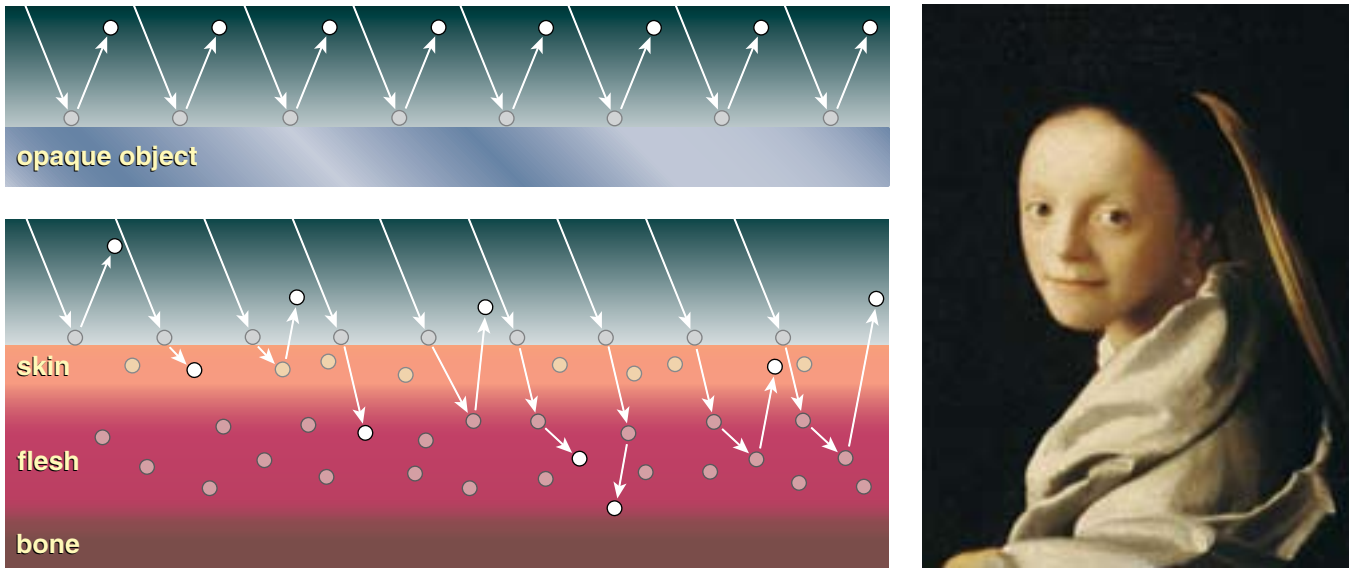


Figure 4. Bump mapping is an extremely efficient scheme for conjuring fine surface detail at render time. A texture map assigned to a 3D model gives shading instructions to the renderer in the form of light and dark regions, which indicate whether regions should cast shadows as if they were slightly raised or lowered from the actual surface of the geometry.



Metropolitan Museum of Art, New York/The Bridgeman Art Library Int.

Figure 5. Visual subtleties can be costly in terms of calculation yet necessary for realism. Light hitting a surface such as skin penetrates and scatters, illuminating the surface from within. Subsurface scattering is an algorithm that captures that effect by propagating light rays and tracking their effects, based on material properties assigned to the 3D object. Renaissance painters, such as Vermeer in his *Portrait of a Young Woman*, met the realism challenge with the analogous technique of glazing, applying layers of translucent pigments to capture and scatter light.

into smaller tasks that can be processed in parallel by specialized computing units within the GPU. Graphics can be seen as a black hole of computational power—the more power you throw at the problem, the more consumers and developers demand in order to render ever more complex images. Hardware architectures, both CPUs and GPUs, are being designed with these market forces in mind.

A recent development in GPU technology is programmability. Ten years ago, GPUs were essentially fixed-function units with some tweakable parameters to accommodate a few different types of calculations. The rigidity of GPUs greatly limited the types of graphic effects that could be rendered. With the newfound flexibility of programmable GPUs, a programmer can specify advanced lighting models chosen to maximize the potential of the hardware. Programmable hardware has also opened the door to conjuring tricks that overcome the inherent limitations of the rasterization approach. For example, researchers and game developers around the world, in pursuit of ever more realistic game scenarios, have developed approximative multi-pass algorithms that can imitate color bleeding, caustics, and subsurface scattering using rasterization on GPUs. However, this development is starting to hit a wall. The results may be attractive, even entrancing, but by the standards of photorealism they are not convincing. Achieving *true* photorealism will require a fundamental change in the way real-time graphics deals with geometry and lighting.

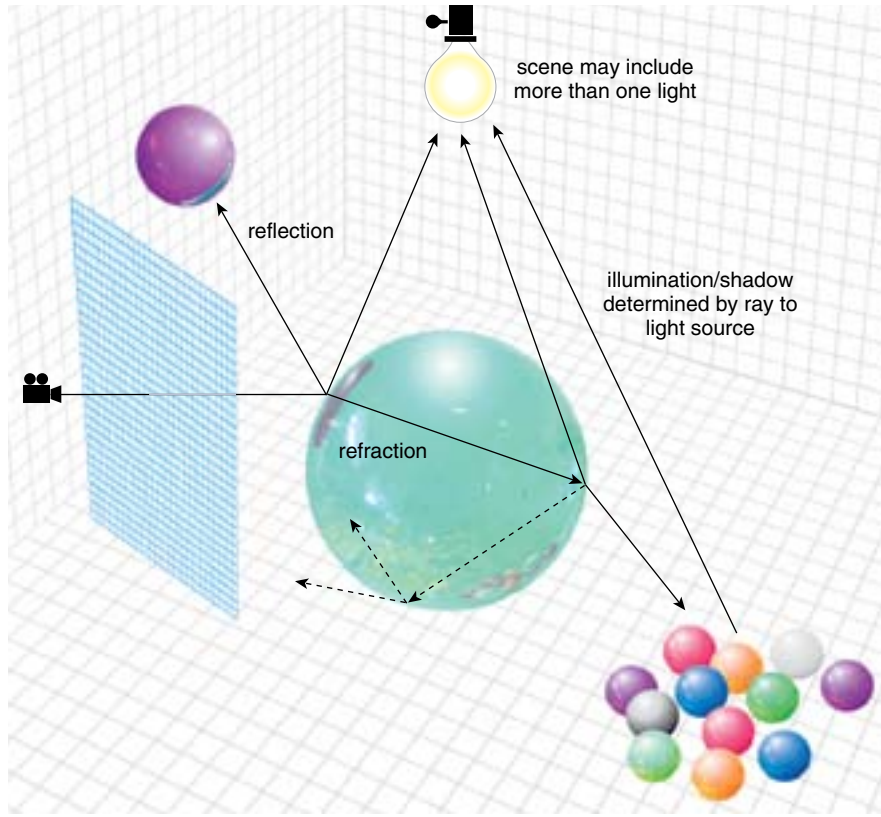


Figure 6. In ray tracing, a ray is shot through each pixel of a virtual screen. Intersection testing between the ray and the geometric primitives in the scene solves for whether the ray hits geometry. An important advantage of ray tracing over rasterization is the ability to represent reflection and refraction, which is done by propagating rays from the points of intersection and tracking their journey through the rest of the 3D scene. In a highly detailed scene, millions of individual rays may be required.

Realism with Ray Tracing

Whoever solves the riddle of moving beyond rasterization will likely hold the key to the future of real-time graphics. A race is on to develop new hardware capable of supporting new algorithms that can simulate the lighting effects that rasterization cannot handle. One of these algorithms is ray tracing.

Conceptually, ray tracing and rasterization are not that different: Both solve for visibility along a ray. Ray tracing differs in simulating individual light rays that shoot through a 3D environment, including the simulated propagation of new rays when light bounces off scene geometry—multiple new rays, in fact, if the light bounces off diffusely, reflectively, refractively, or in combination as real light generally does. By tracing individual light rays back to a light source, it is possible to account in a reasonably natural way for the actual physics of light, not just reflection and refraction but also specialized effects like color bleeding and caustics.

Ray tracing is an elegant algorithm, quite simple to specify in code—Paul Heckbert, now a 3D graphics architect at the video card vendor NVIDIA, coded instructions for a functional ray tracer that can be printed, just legibly, on a business card. (The feat was stimulated by a contest in which, it is gleefully reported, "repulsive C code tricks" were unveiled.) The natural way in which ray tracing deals with lighting makes it an obvious candidate to replace rasterization, but a simple algorithm does not necessarily correlate with rapid production of a finished image. The speed of rasterization derives from capturing the visible features of a triangle, then forgetting the triangle as it moves on to the next one. Ray tracing must take account of an entire scene in which light rays bounce around. In ray-tracing algorithms, it is necessary to process all of the triangles in the scene and then convert the data into an *acceleration structure,* a configuration of the data that optimizes the ability to determine if a given light ray hits a triangle. Different lighting effects may benefit from different acceleration structures. At every step in graphics rendering, researchers are exploring ways to optimize the calculations.

Ray tracing is unavoidably a highly computation-intensive algorithm. Because it tracks the path of every individual ray of light that illuminates a scene, it may be necessary to trace several million rays for a single image. If more advanced effects are incorporated, the number of rays can multiply substantially. The benefit that seduces researchers is the beauty of the images that result. For example, the imperfect lighting of lesser schemes can be replaced by the breathtaking realism of *global illumination,* in which environments are lit, as in reality, not by one or a few light sources, but by all the surfaces that reflect diffuse light back into the scene.

Given the speed advantages of nearly-good-enough rasterization and the computational challenges of better-than-good-enough ray tracing, the next generation GPUs are likely to support both algorithms. The current market for GPUs is thoroughly dominated by three vendors, Intel, NVIDIA and AMD/ATI, which in 2008 represented 97.8 percent of market share. These companies are known to be betting on an evolutionary approach to existing architectures, in which increased programmability will allow ray tracing to be implemented as a complement to rasterization. The world's largest chipmaker, Intel, embarked on the development of a completely new architecture codenamed Larrabee, a "many-core compute engine" based on Intel's highly successful x86 CPU architecture, the processor family used in both PC and Mac computers. The Larrabee architecture has been called a general-purpose GPU, indicative of the blurring boundary between GPUs and CPUs. While supporting traditional GPU functions like rasterized graphics, hybrid CPU features of the Larrabee can be used to carry out tasks such as ray tracing and advanced physics calculations. (A pleasing side effect of the thriving consumer market, in which competition for the millions of graphics cards purchased each year drives down prices, is the availability of inexpensive, high-performance computing power for other purposes, such as scientific computing.) In December 2009, Intel announced that the first graphics product based on the Larrabee architecture will not be a consumer product as originally planned. Instead, the hardware will be released as a software development platform that will be used by Intel and others to explore the potential of many-core applications. This is a familiar stage in the development of computer graphics over the years, as consumer desires drive the development of more muscular hardware, and hardware developments drive the advance of software applications like real-time ray tracing that come into reach on the new architectures.

The progression from fixed-function to highly programmable GPUs, and now to architectures with minimal fixed-function hardware, is a sign of the *wheel of reincarnation,* in which functionality is transferred from the CPU to special-purpose hardware for performance reasons, followed by power-craving expansion of the subsidiary unit. The process was first described and named by Todd Myer and Ivan Sutherland as early as 1968:

> We approached the task [of creating a graphics processor] by starting with a simple scheme and adding commands and features that we felt would enhance the power of the machine. Gradually the processor became more complex. We were not disturbed by this because computer graphics, after all, *are* complex. Finally the display processor came to resemble a full-fledged computer . . .

To escape the wheel of reincarnation, Myer and Sutherland suggested that if an architecture needs more computational power, it should be added to the core of the system, rather than spurring the creation of special-purpose

Figure 7. Many rendering effects depend on multipass rendering, with information from each pass combined in a final image. The top left image gives a striking view of depths in the scene using a specialized algorithm to capture shadow information. Upper right shows diffuse color without shadows. The two images are combined at bottom left, and at bottom right additional lighting information such as specularity (shininess) dramatically improves the realism of the image. (Images courtesy of Crytek GmbH.)

Figure 8. Computer graphics researchers probe reality for the delicate effects that make or break the realism of an image. Caustics appear when light that is reflected *(left)* or refracted *(right)* accumulates or cancels, generating exotic shapes and hues that the observer may not recognize, but expects. (Photograph courtesy of Tomas Akenine-Möller.)

hardware units. We may be seeing that in the emergence of multiple processors, multiple cores within processors, and enabling architectures that increasingly support parallel processing.

When?

Current graphics hardware is capable of processing several tens of millions of rays per second. Although this sounds impressive, it is still far from the required number of rays for a modern game setup. Modern games rendering at 60 frames per second in high-definition resolution, 1920 x 1080 pixels, with, let us say, 16 rays per pixel for all lighting effects, require 60 x 1920 x 1080 x 16 = 2 billion rays per second, which is approximately two orders of magnitude more than current hardware can deliver. One obvious strategy to overcome this challenge is to increase the capability of the hardware. A great advantage of ray tracing is that it is a highly parallel algorithm—it has been called "embarrassingly parallel." Each ray can be traced independently. This is significant since it allows ray tracing to exploit the parallel nature of GPUs; if 100 processors in parallel cannot complete the job quickly enough, perhaps 1,000 can.

NVIDIA, AMD/ATI and Intel are all betting on parallel computing. The latest GPUs contain hundreds of individual compute units, each capable of tracing individual rays. Intel's Larrabee architecture uses a hybrid strategy in which multiple x86-derived processors use specialized vector processing to trace batches of rays simultaneously. This approach is quite challenging to program and it is still unknown if ray tracing can utilize the hardware to its full potential, but promising work has been done on current CPUs. Yet the challenge is not to be underestimated. Moore's law, which has predicted the progress in computer power over the past 40 years, says that transistor density will double every two years. Due to performance increases in transistors, this can be translated to a doubling in computer performance every 18 months. If Moore's law holds, then, it will take roughly 10 years before consumer machines are capable of tracing the few billion rays required to render the game setups that are currently available. And in 10 years, the requirements for games and real-time graphics in general might be different, perhaps calling for higher resolutions or yet-to-be-thought-of algorithms.

Hybrid Future

Skeptics may claim, with some justification, that real-time ray tracing is a pipe dream that will never be realized; the hardware will always be too slow. Even if the hardware becomes fast enough to handle 16 rays per pixel in a full-resolution scene, that may not be enough to achieve all the lighting effects that photorealistic ray tracing may call for. With this in mind there is a growing train of thought that the future may be a hybrid approach that combines both rasterization and ray tracing.

Combining rasterization and ray tracing is an old idea in computer graphics. The basic approach uses rasterization to decide which triangles can be seen on the screen and then uses ray tracing to perform the shading calculations. This method can be used with current GPU hardware, employing ray tracing selectively to add reflections and refractions in strategic places. There is little doubt that future generations of real-time graphics for games will use this approach for as long as the pure ray-tracing approach is unattainable on available hardware.

Pixar uses a hybrid rendering technique to create its movies based on the Reyes algorithm, an advanced form of rasterization. (Reyes is an acronym for "renders everything you ever saw.") Reyes generates micropolygons—scene geometry is tessellated at render time into pixel-sized triangles or quadrilaterals. The use of micropolygons makes it possible to create complex geometric effects through the use of displacement mapping—similar to the bump mapping described earlier except that it actually displaces the geometry, on a tiny scale, rather than just giving the appearance of displacement. This is a powerful way of creating details such as the pores on human skin, although it can generate significantly more complex geometry than current ray-tracing algorithms can deal with. Micropolygon rendering can be practical on GPUs, and if future games were to use micropolygon rendering, the visual quality of a game could be similar to that of the movie *Toy Story.* However, micropolygon rendering fails at simulating the same lighting effects that limit rasterization. Pixar's response has been to use ray tracing coupled with micropolygon rendering in a hybrid setup. But when making its movies, Pixar doesn't have to worry about how long it takes to render a frame.

There is another alternative to ray tracing—trick the human observer. Perhaps it is not necessary to have fully accurate lighting and reflections in the next generation of games. This is the approach that current games use. The real-time graphics community has developed many tricks that deliver great-looking graphic images in real time. For example, NVIDIA has shown a demo of human skin rendered with

subsurface scattering running in real time on a GPU. Clever filtering techniques generated rendered images that looked very convincing; few people could see the difference between their result and a ray-traced image. However, an approach based on tricks has limitations. Each trick is usually highly specialized and often does not mix well with other tricks. For example, it would likely require acrobatic coding to simulate indirect lighting on a human face with simulated subsurface scattering. This ultimately is what makes ray tracing attractive. It scales very well with the addition of processing power, and it is trivial to account for advanced lighting effects by simply tracing more rays.

The annual SIGGRAPH conference (Special Interest Group, Graphics) is the premier venue for computer graphics research. At the August 2009 SIGGRAPH, the crowd-pleasing Computer Animation Festival component of the program presented the debut of a new session, Real-Time Rendering, in which developers demonstrated their most advanced real-time games and other applications alongside the ground-breaking prerendered works that are the staple of the conference. NVIDIA and Intel both demonstrated real-time ray tracing on their hardware. Intel, using their current-generation CPU architecture, code-named Nehalem and released in late 2008, demonstrated a ray-traced game scenario running at approximately 15 frames per second, featuring a sea bottom visible through the shimmering surface of a lagoon. Progress is being made.

Some years ago, veteran game developer Billy Zelsnack said, with hopeful irony, "Pretty soon, computers will be fast." Those words remain as true today as the day they were spoken. We add this, with less ambiguity: "Pretty soon, photorealism will be real-time."

References

Akenine-Möller, Tomas, Eric Haines and Naty Hoffman. 2008. Real-Time Rendering, 3d ed., A. K. Peters Ltd.

Jensen, Henrik Wann. 2001. Realistic Image Synthesis Using Photon Mapping, A. K. Peters.

Myer, T. H., and I. E. Sutherland. 1968. On the Design of Display Processors. *Communications of the ACM* 11:410–414.

Pharr, Matt, and Greg Humphreys. 2004. *Physically Based Rendering: From Theory to Implementation.* Morgan Kaufmann.

Seiler, Larry, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, and Pat Hanrahan. 2008. Larrabee: A Many-Core x86 Architecture for Visual Computing, *ACM Transactions on Graphics* 27:18.1–18.15.

Whitted, Turner. 1980. An Improved Illumination Model for Shaded Display. *Communications of the ACM* 23:343–349.

For relevant Web links, consult this issue of *American Scientist Online:*

http://www.americanscientist.org/issues/id.83/past.aspx