Light It Up! *Quake Wars** Gets Ray Traced

BY DANIEL POHL

A SCENE UNFOLDS in the computer room of a major university. Those watching sense electricity in the air, the kind of tension that builds before a thunderstorm, as a cluster of 20 networked PCs, each equipped with spanking new dual-socket technology and dual processors, warm to the task assigned to them: distributed ray tracing of the game *Quake* 3* (www.q3rt.de). Though the modest display resolution (512x512) and a frame rate of 20 frames per second (fps) aren't overwhelming by the standards of the day, this doesn't diminish the accomplishment in the least. Special effects never before seen flimmer across the display screen. The viewers watch with rapt attention and a feeling of satisfaction as the intricately rendered images move about the screen. Amazingly, this happened in 2004, a time when most people rejected the concept of real-time ray tracing.

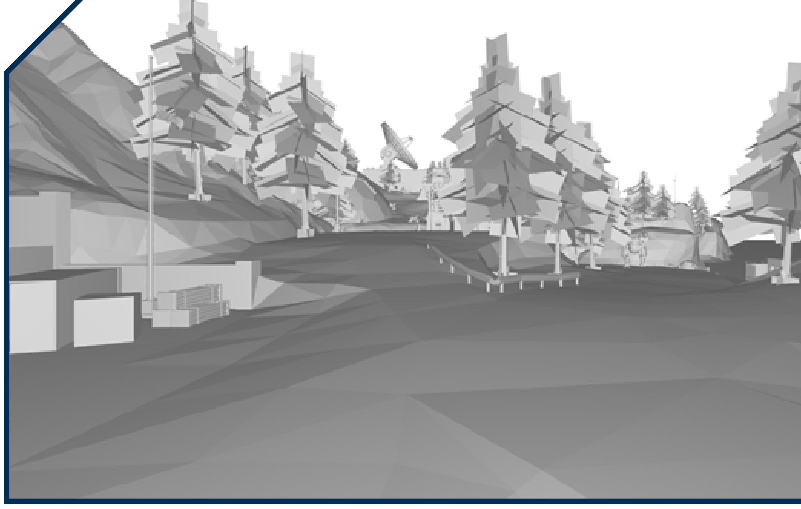
BACK TO THE FUTURE (2008, THAT IS)

A new research project from the ray-tracing team at Intel advances beyond the 2004 achievements, this time converting the game *Enemy Territory: Quake Wars**, which was created by id Software and Splash Damage, to use ray tracing. Read on to learn about the development process that followed, the challenges we had to overcome, and the benefits we ultimately achieved—all of which provide valuable insights into the future of ray tracing. To pump up your visual adrenaline level, we've also included numerous images to show the process in action. By the time you finish this article, you'll have a better idea of the ways in which ray tracing can quickly and easily render light and shadow.

STARTING FROM SCRATCH

For this project, we started rewriting the renderer from ground zero. Because of this, the very first images from the renderer were not of typical ray-tracing caliber, but displayed only the basic parts of the geometry, without any shaders or textures (Figure 1).

Figure 1. Quake Wars*: Ray traced without textures.



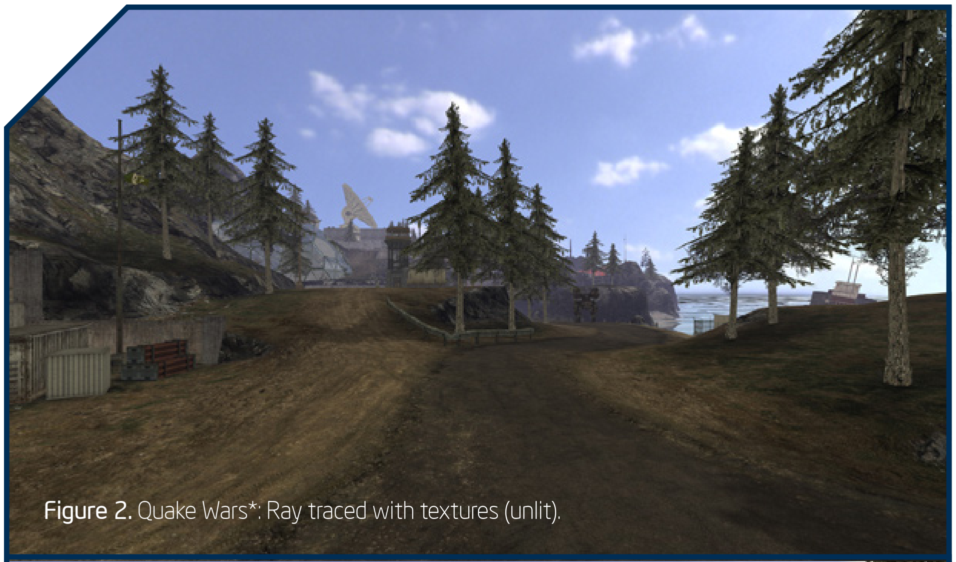


Figure 2. Quake Wars*: Ray traced with textures (unlit).

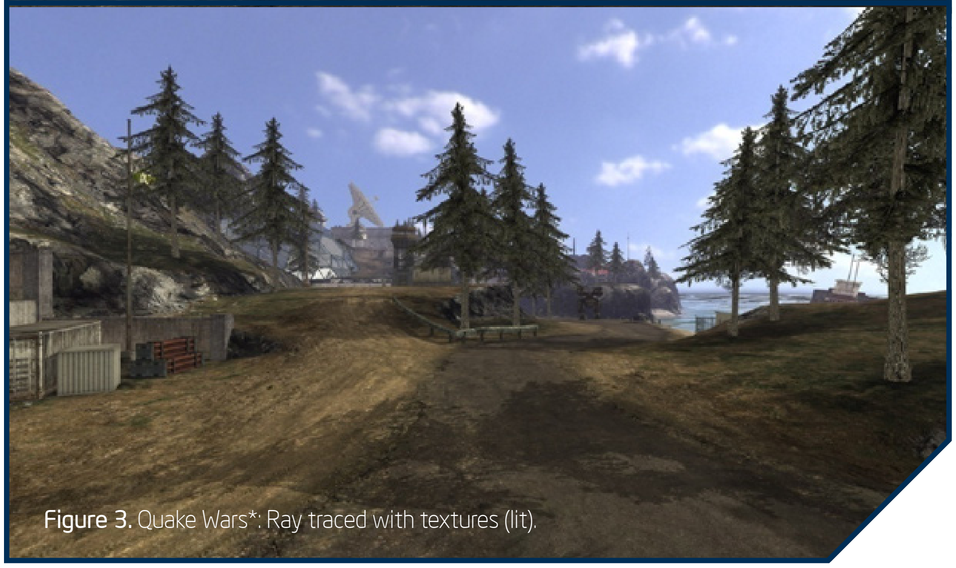


Figure 3. Quake Wars*: Ray traced with textures (lit).

Typically, games load their geometry from a variety of different model formats—either created over the in-game map editor or through external modeling tools. Once it is verified that there are no missing objects, the loading of textures can begin. Modern games have their own material description language that allows designers to easily modify texture parameters, blend textures, use bump and specular maps, and write small shader programs. For example, compare the untextured image in Figure 1 with the unlit (Figure 2) and lit (Figure 3) textured images of the same scene.

Today's games all use a rendering technique called rasterization. Rasterization requires difficult programming work and as many special effects (such as shadows or reflections) need to be calculated as approximations over multiple rendering passes and are often stored in resolution-limited textures in between. These approximations fail in certain cases. Let's look closer at shadows. With ray tracing, you only need to check if the path from the light to the surface is blocked or not. This can be easily determined with just a ray (the so-called "shadow ray"). If the ray from the light source can reach the surface, the point on the surface is lit. Otherwise, it is in shadow. The gameplay in *Quake Wars: Ray Traced* takes place primarily outdoors, where the most important light source is sunlight. We were able to apply this form of lighting to the scenes quite easily, and the appearance of the shadows is what one would expect.

TRANSPARENCIES

Instead of employing real 3D geometry, game developers sometimes approximate 3D properties with a 2D quad surface (or two triangles, as shown in Figure 4) and a texture on which transparency values have been applied.

Creating correct shadows from partially transparent quads is not an easy task for a rasterizer. The most commonly used algorithms for calculating shadows in rasterization (called "shadow mapping"—see http://en.wikipedia.org/wiki/Shadow_mapping) does not deliver additional information that might help in the case of shadows from

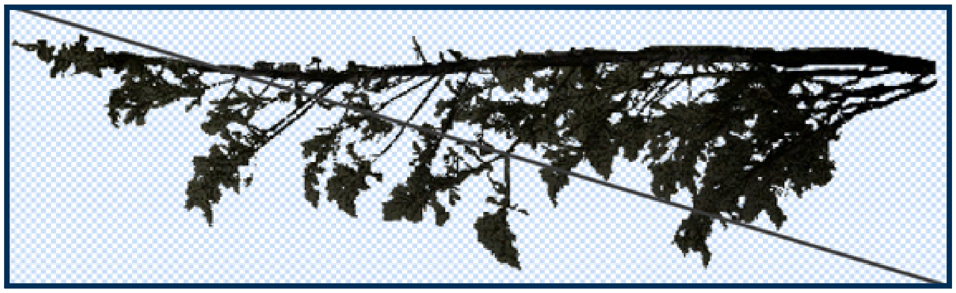


Figure 4. Example of a partially transparent leaf texture applied to a two-triangle surface.

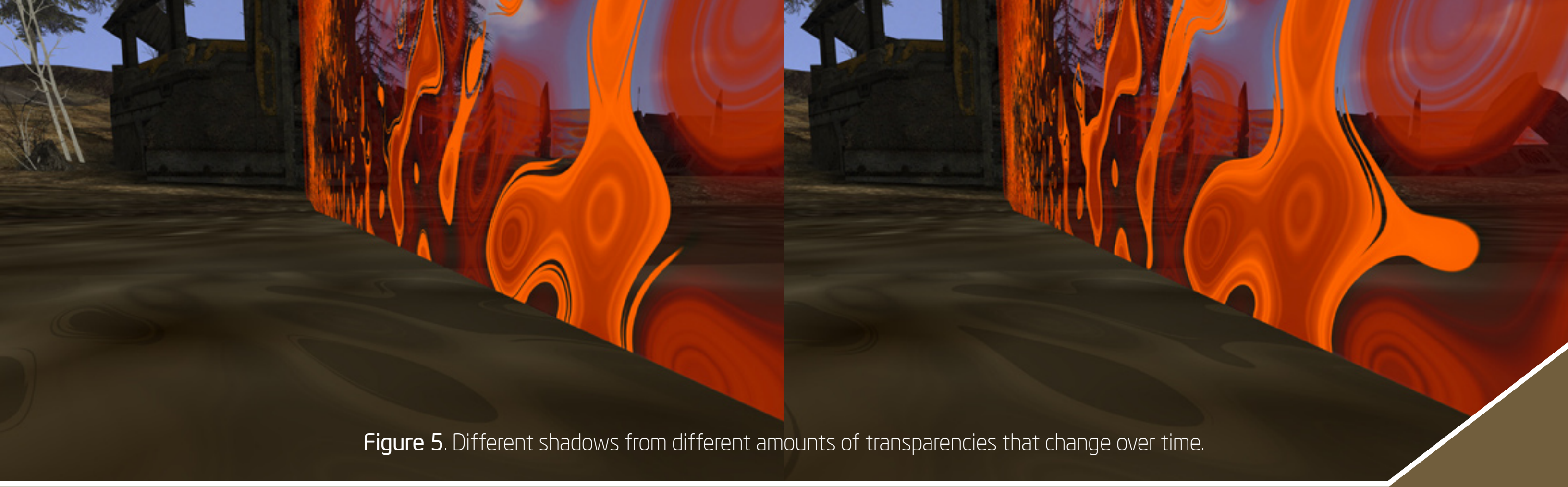


Figure 5. Different shadows from different amounts of transparencies that change over time.

transparencies. For that reason, shadows are sometimes baked into textures, and, because of this, they don't change when the light position changes (such as when a scene changes from sunrise to sundown).

When using ray tracing, however, the algorithmic solution is simple. If the shadow ray hits an object, the program can read the transparency value of the texture and continue tracing that ray, when the texture sample is transparent. This offers interesting special effects, but also creates challenges. The images in Figure 5 show an animated force-field shader effect that casts a different intense shadow depending on the transparency values of the orange force field.

Another advantage of using a ray tracer for partially transparent objects is that they don't need to be sorted by their depth. This makes it easier for the developer, but there is a downside: increased rendering costs. Whenever a ray hits such a surface, another ray needs to be shot from that point in the same direction. If this happens once, the impact is small. But what happens if you have ten or more of these surfaces in a row? This can happen if a tree, for example, consisting of a mix of partially transparent quads, is rendered [Figure 6(a) and (b)].

Rendering a large number of those trees in the outdoor world quickly became our biggest performance bottleneck. During several optimization cycles, we came up with many improvements. The following improvements had the greatest impact on performance.

- Avoid shooting a new ray each time after reading the transparency value; instead, we reused the same ray now originating from the hit position and continued in the same direction afterwards.
- Signify whether a texture uses transparencies with a single flag. If not, there is no need to shoot additional rays through potential transparencies. The bark inside a tree is an example of an opaque texture mixed in between many partially transparent textures.
- Decrease the number of rays that are bundled together. In many cases, bundling rays with almost the same path can lead to substantial speedups. However, if one part of the bundle hits another surface then the other one, it produces some reorganization

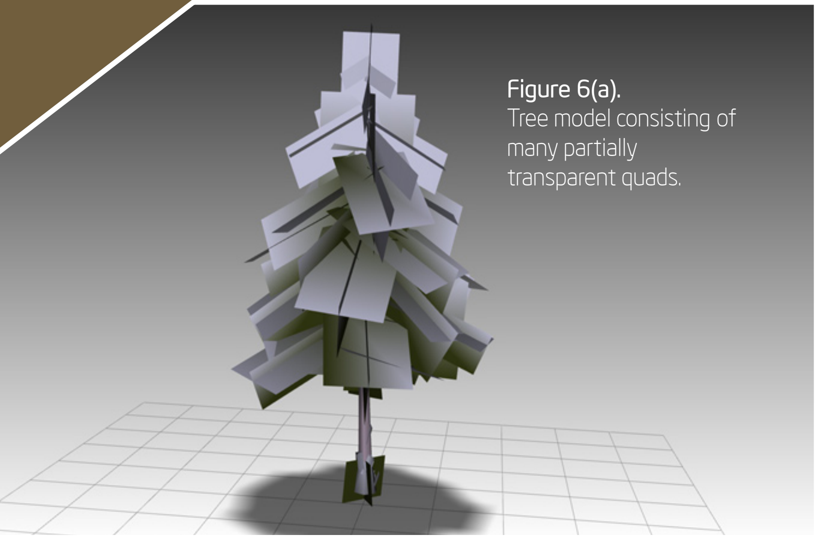


Figure 6(a). Tree model consisting of many partially transparent quads.

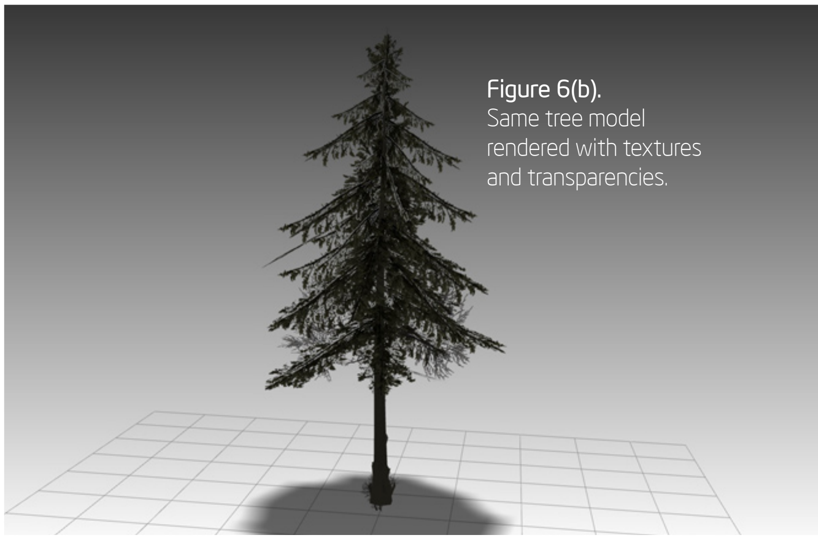


Figure 6(b). Same tree model rendered with textures and transparencies.

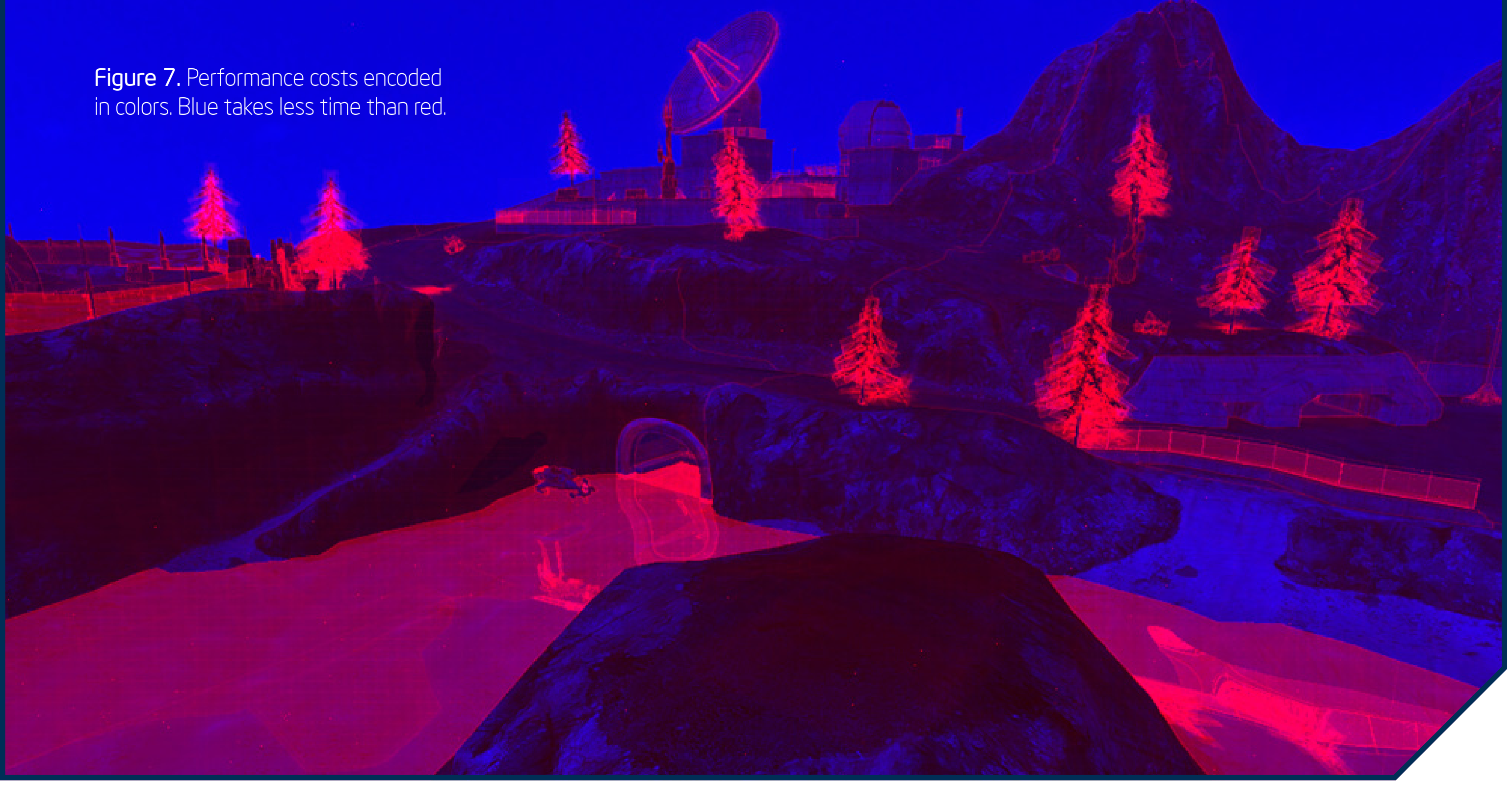


Figure 7. Performance costs encoded in colors. Blue takes less time than red.

In the case of rendering the trees, this overhead can become significant, slowing everything down.

Even after a great deal of tweaking, rendering the trees is still very time consuming. We visualized the costs of rendering a single pixel in a color scheme where a blue pixel represents a quickly calculated pixel and a red pixel an expensive one (Figure 7). An intense red is also more costly than a light red tone. As can be seen in the figure, rendering the trees is still more expensive than rendering a reflecting water surface or other parts of the scene. More research needs to be done on rendering these trees to discover if further improvements are possible.

ADDING MORE SPECIAL EFFECTS

In our ray-tracing conversion, once we reached the same quality as the original game, we began adding enhancements and more special effects. Ray tracing does a very good job with reflections and refractions. The most common everyday objects in the world that exhibit this behavior are glass and water.

GLASS

A large dome exists in the original game. We changed the surface properties of the dome so that it would appear to be made out of glass (Figure 8). Using the refraction index for glass (which you can find in your favorite physics books), we wrote a shader to accurately depict the reflections and refractions. The code is about 15 lines long in our HLSL-like ray-tracing shading language and generates very pleasing results.

WATER

Rendering water can be accomplished different ways. We investigated two approaches: water on a 2D surface and water with genuine 3D properties[1].

[1]Source: Implementation by Jacco Bikker

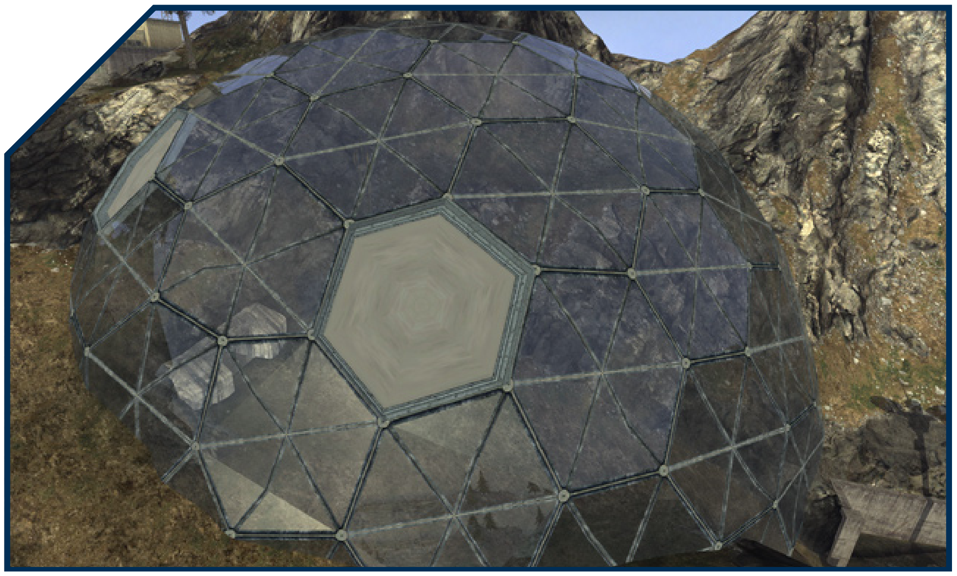


Figure 8. Dome appears to be made out of glass after applying a shader.

To render the water in 2D, we used a bump map to simulate waves [Figure 9(a)]. The 3D water image uses a mesh with around 100,000 triangles in several subgrids [Figure 9(b)]. Those subgrids are updated every frame, depending on their visibility. (During rendering, subgrids that are not visible are ignored.) The visibility test is performed over rays.

THE PERFORMANCE ISSUE

Performance is the main reason why ray tracing is not yet used in mainstream games. Compared to special-purpose rasterization graphics hardware—such as current-generation GPUs—ray tracing is fairly slow. Also, a lack of texture units for our CPU-based approach to ray tracing causes significant slowdowns when trilinear filtering is used for all texture samples. With Intel's latest quad-socket systems—equipped with a 2.66 GHz Dunnington processor in each socket—we can achieve approximately 20 to 35 fps at a resolution of 1280x720. Nonetheless, this represents a significant improvement over the experiments in 2004 that required 20 machines to render a simpler game more slowly and at a lower resolution. The greatest performance gains result from research efforts around the world that improve efficiency and the new, many-core hardware platforms that use parallelism to accelerate graphics operations.

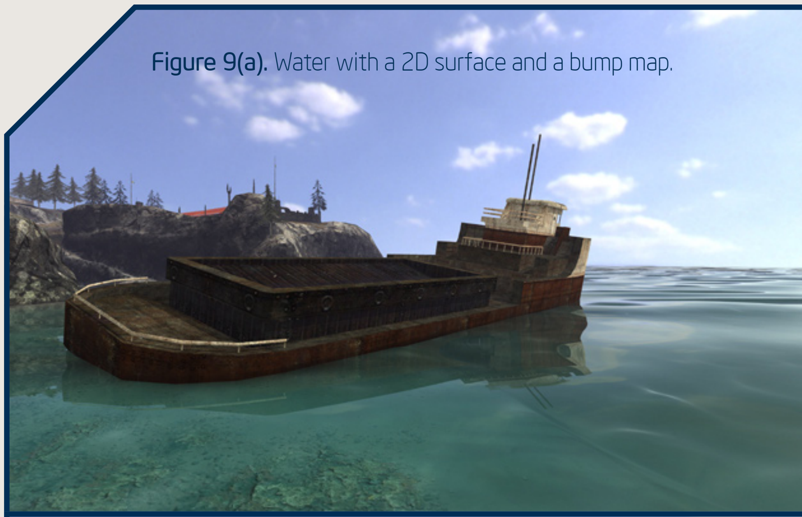


Figure 9(a). Water with a 2D surface and a bump map.

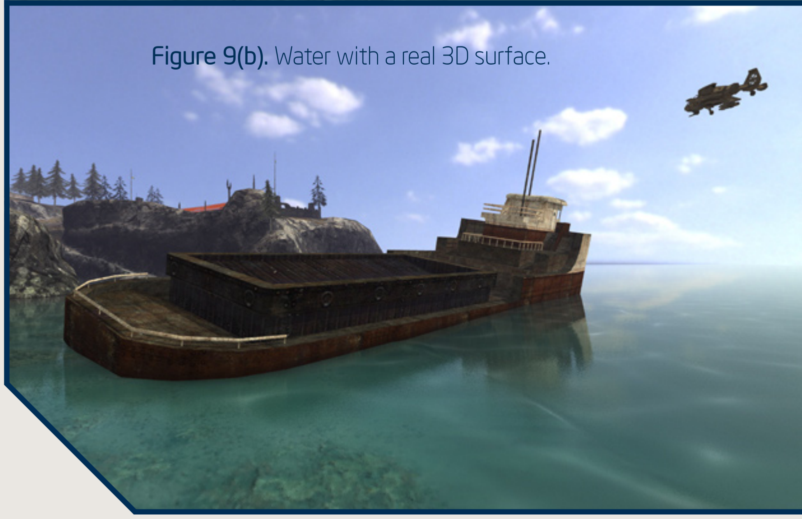


Figure 9(b). Water with a real 3D surface.

THE FUTURE OF RAY TRACING

As mentioned earlier, creating very realistic shadows in games is not an easy task. Given the current state of our demo work, only hard-edged shadows are produced. Modern games tend toward soft shadows, which usually require many more rays. This important topic deserves more study; smarter approaches to this task need to be developed. Also, to obtain higher quality images, better anti-aliasing methods are needed. Adaptive super-sampling is a smart way of refining the rendering of the scene at those exact places where it will deliver the greatest benefit. There are experimental implementations, but they need to be tested and tuned for the best results. With the industry moving from multi-core to many-core (that is, greater than ten cores), improving the algorithms so they can fully use the newly acquired power will be interesting.

Even though Intel's upcoming many-core graphics architecture, code named Larrabee, has been primarily developed as a rasterizer card, it will also be freely programmable. This opens up some extremely interesting opportunities to perform ray tracing with the Larrabee architecture.

Stay tuned for more information about our upcoming ray-tracing projects! ▪

ABOUT THE AUTHOR

Daniel Pohl started researching real-time ray tracing for games in 2004 during his study of computer science at the Erlangen-Nuernberg University in Germany. As his master's thesis, he developed a ray-traced version of Quake 4*. In 2007, he joined Intel's ray-tracing group. In 2008, he moved from Germany to sunny California where he continues to research game-related ray tracing.*

RESOURCES

Intel® Software

Intel's heightened focus on visual computing and graphics processing is complemented by software development products, graphics chipsets, professional services, technical expertise, and developer-oriented resources. Keep up with the activities of Intel's Visual Computing Software Division through www.intel.com/go/visualcomputing.

Explore topics from this issue of *Visual Adrenaline* further:

Alert: Latest Call of Duty Release Breaks New Ground*

For an insider's view of the Treyarch work on *Call of Duty*: World at War*, go to www.treyarch.com.

To learn more about the history of cooperative gameplay, view this Wikipedia entry: en.wikipedia.org/wiki/Cooperative_gameplay.

For the latest releases from Activision, visit www.activision.com/index.html.

An Evil Genius Test Drives the Intel® Core™ i7 Processor Extreme Edition

For the latest adventures of Team EG, visit www.myeg.net.

To learn more about the Intel® Extreme Masters events, go to www.intelextrememasters.com.

To follow the Electronic Sports League WC3L series, visit www.esl.eu/eu/wc3l/.

Damien Thaller: Passion and Talent Bring Stories to Life

To view more of Thaller's portfolio, posted in the CG Society gallery, go to thaller.cgsociety.org/gallery.

For information about Evolve Pictures, visit www.evolvepictures.com.au/.

To learn more about Animal Logic, go to www.animallogic.com/#Home.

DreamWorks Animation and Intel: Forging an Alliance to Advance S3D Entertainment

For a look behind the scenes of modern day animation, go to www.dreamworksanimation.com and click the Studio button.

For more background on InTru™ 3D and the history of animation, go to www.intel.com/consumer/learn/intru3d.htm.

Enabling 3-D Moviemaking: Autodesk® Retools Maya®

For details about advances in digital cinema, visit the *Digital Cinema Report** at www.digitalcinemareport.com.

For more on the history of Autodesk Maya® and its 10th anniversary celebration, go to area.autodesk.com/maya_anniversary.

For lists of 3D motion picture theaters in different parts of the country, go to marketsaw.blogspot.com/2007/12/wow-they-are-popping-up-everywhere.html.

IGOR: The Making of a Monster Hit

For a spirited introduction to the characters that populate the world of *Igor* (the game), visit www.igorgame.com.

For a trailer and the inside scope on *Igor* (the movie), visit www.igor-movie.com/.

To tap into the energy at Santa Cruz Games, visit www.santacruzgames.com.

Check out the Sparx Animation Studios activities at www.sparx.com/index.html.

Light It Up! Quake Wars Gets Ray Traced*

For more details about the original ray tracing of Quake* 3, visit www.q3rt.de.

To learn about recent ray-tracing developments, go to www.q4rt.de (*Quake 4: Ray Traced*) and www.qwrt.de (*Quake Wars: Ray Traced*).

For the latest Intel news and development on the ray-tracing front, visit the Visual Computing Developer Center: software.intel.com/en-us/visual-computing.

- Subscribe to the *Intel® Software Insight* magazine: www.intel.com/go/softwaredispatch
- Sign up for the Intel® Software Partner Program, available to software companies: www.intel.com/partner
- Tap into multi-core resources: www.intel.com/software/mcdeveloper
- Find out more about Intel® Software Network: www.intel.com/software
- Explore Intel® Software Development Products: www.intel.com/software/products
- Build your knowledge base with books from Intel® Press: www.intel.com/intelpress/
- Find online and classroom training courses from Intel® Software College: www.intel.com/software/college
- Interact with a lively community of individuals in the Intel® Graphics Developer Community: www.intel.com/software/graphics/

To sign up for an on-going subscription (gratis) of the *Intel® Visual Adrenaline* magazine, as well as the Intel® Software Dispatch for Visual Adrenaline e-mail program, go to: www.intelsoftwaregraphics.com.

To subscribe to Intel® *Visual Adrenaline,*
go to **www.intelsoftwaregraphics.com**