Ray Tracing and Gaming - Quake 4: Ray Traced Project

Author: Daniel Pohl
Date: Dec 18, 2006
Subject: Graphics Card
Manufacturer: General

Introduction to Ray Tracing

Imagine playing a computer game in the quality seen in the movies of the "Lord of the Rings"-series; absolute realistic lighting, details like in real life, believable skins and much more.

("Lord of the Rings"-Movie: Rendered image with ray tracing - Copyright Newline Cinema)

One technique that was used creating the images for those movies is called "ray tracing". This is an alternative rendering technology compared to what actual graphic cards on modern PCs and consoles do. For many years ray tracing has only been used for offline-rendering and the generation of pictures for movies often took many days to calculate. Real-time ray tracing has been made possible with the OpenRT Ray tracing library. Through using many PCs over an Ethernet network interactive frame rates could be rendered in high resolution. Now, four years later CPUs have progressed a lot and ray tracing works in small resolutions on a single PC in real-time, but more on this later.
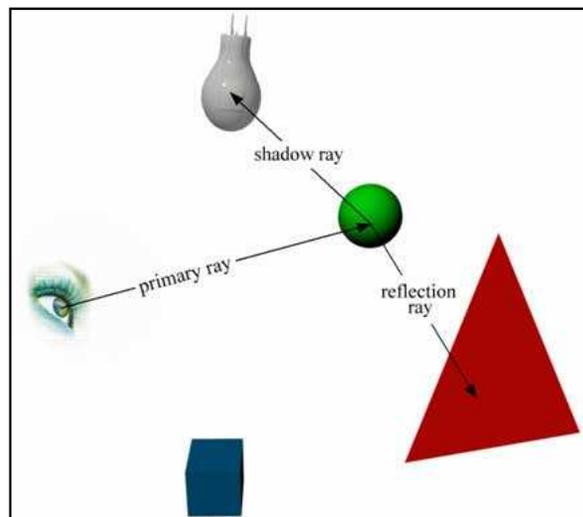
OpenRT
OpenRT is a ray tracing library developed by the Computer Graphics Group of Saarland University. The syntax of the API is nearly identical to OpenGL. A free Linux test version for programmers can be downloaded.
Webpage: http://www.openrt.de/

Ray tracing

So, how does ray tracing work? The algorithm is quite simple.

A basic ray tracing scenario.

1. From a virtual camera (denoted by the eye), primary rays are shot through every pixel on the screen.
2. For every ray the intersection hitpoint$_{primary}$ with the object closest to the camera is calculated. In the example this is the green sphere.
3. The shader program of the green sphere is called.
4. In this example the material of the green sphere should be totally reflective, so the shader shoots a reflection ray from hitpoint$_{primary}$ into the reflected direction.
5. The reflection ray hits an object and hitpoint$_{reflected}$ is calculated. In the example it is the red triangle.
6. The shader program of the red triangle is called.
7. A shadow ray is shot from hitpoint$_{reflected}$ into the direction of the light source (not shown in the figure).
8. There is an object between the point on the triangle and the light source.
9. Nothing is added to the triangle shader color, therefore the point represents shadow.
10. The triangle shader returns its color, which is added to the frame buffer.
11. A shadow ray is shot from hitpoint$_{primary}$ into the direction of the light source.
12. There is no object between hitpoint$_{primary}$ on the triangle and the light source.
13. Through the attributes of the light source a color values is calculated and added to the frame buffer.

For efficient ray tracing a space acceleration structure (e.g. a BSP http://en.wikipedia.org/wiki/Binary_space_partitioning) is used, which holds all geometry of the 3D scene. Using this technique the blue cube from the figure above was not touched for the calculation of the final color.

Quake 3: Ray traced

In 2004 I started working on one of the first raytracing computer games as my student research project. As a base for this project the content from "Quake 3" was used. It was amazing how easy it was to program special effects that would have taken much longer using the conventional graphic technology called "rasterization" used by common graphic cards. For example dynamic, per-pixel real-time shadows are calculated with about ten lines of code as described in the introduction figure about ray tracing.
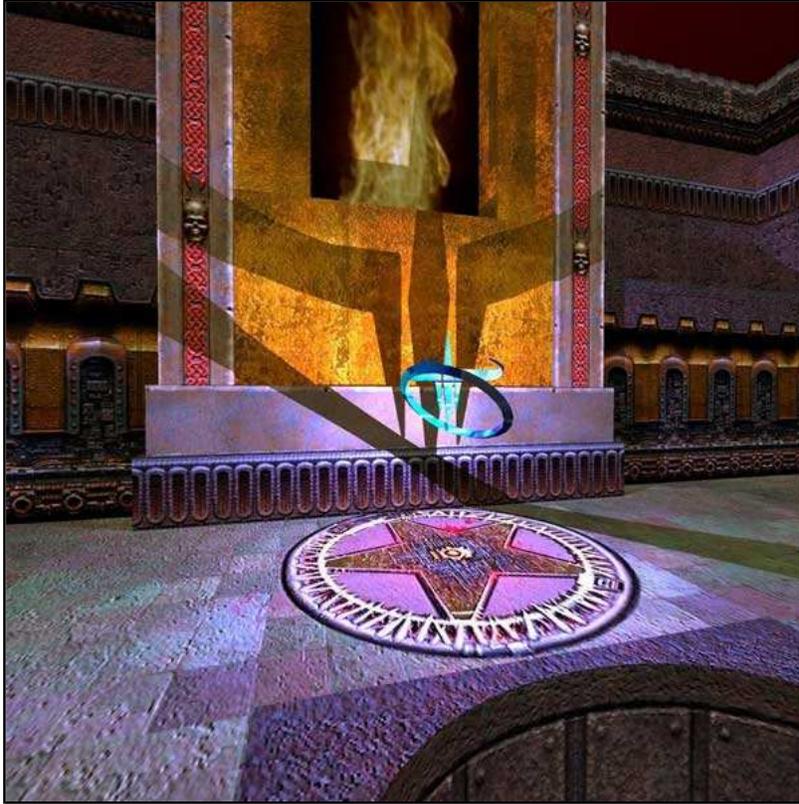
(Shadows in Quake 3: Ray traced (2004))

Generating shadows without disturbing artifacts is still a problem in many games, though some games have done it right. Here some of the bad examples from games, all released in 2006:


("Gothic 3" (2006) Shadows

*("The Lord of the Rings: The Battle for Middle-earth II" from 2006)*

("Call of Juarez" from 2006)

Another advantage of ray tracing is that you can use many more polygons to render a scene without a dramatic performance impact compared to the rasterization approach. So I "pimped up" some of the walls and floors in Quake 3 to see how well it performs. Instead of two triangles for a wall I used 5,000.
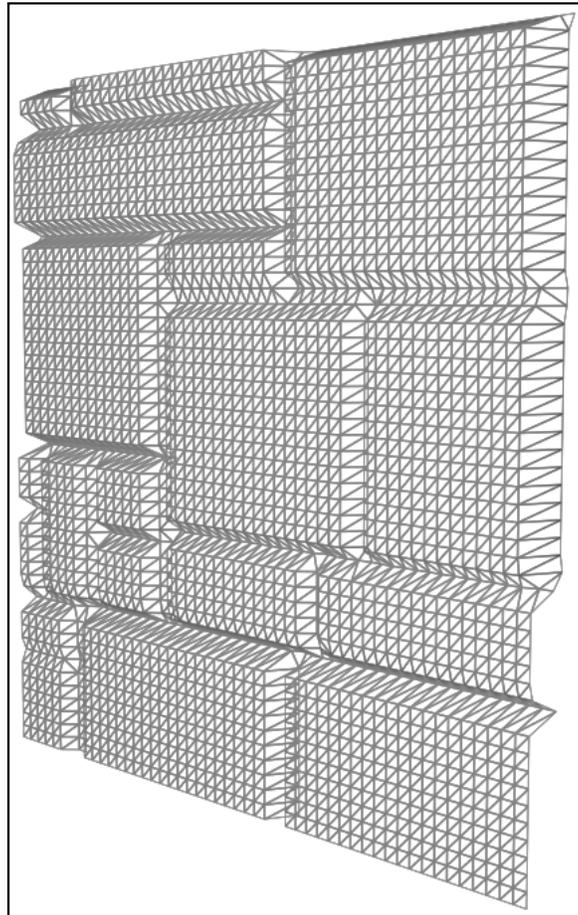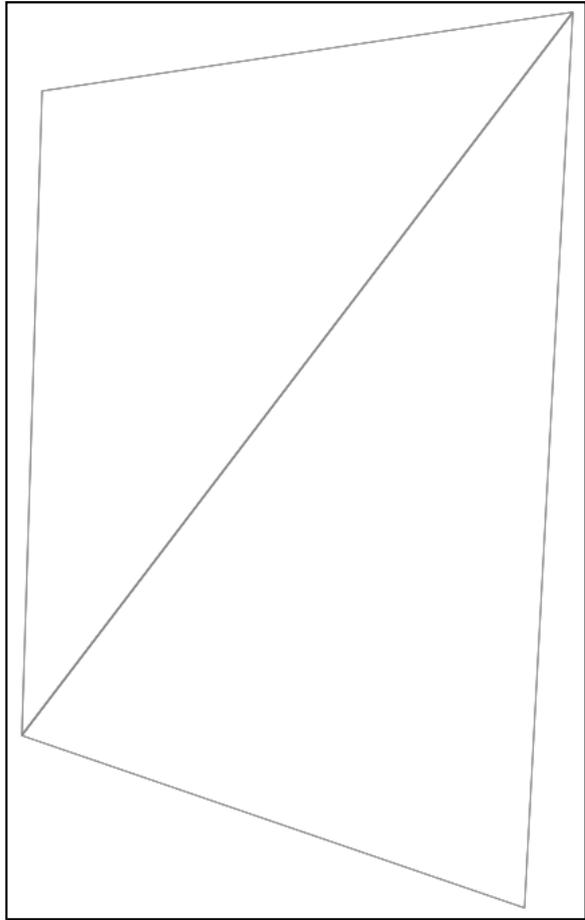
(Using 5,000 triangles instead of 2.)
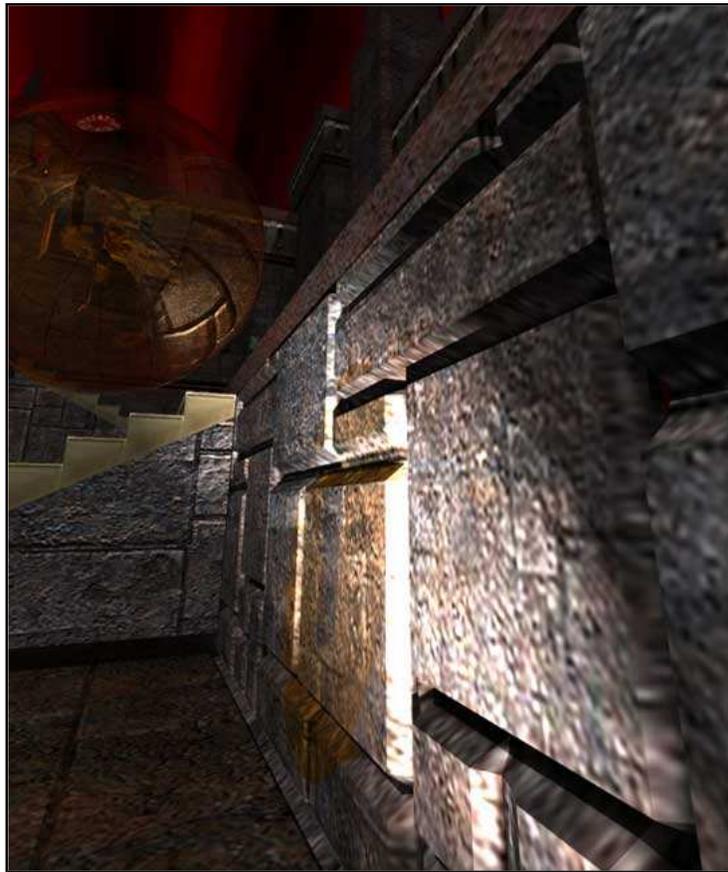
(The replaced walls in game. The real geometric displacement can be seen from any perspective.)

The result was: while the geometry complexity of the Quake 3 Level was six times higher, the frame rate dropped only to about ¾ of its original value.

Other special effects in Quake 3: Ray traced (http://www.q3rt.de/) are glass, mirrors in mirrors, camera portals and ground fog.

To see Quake 3: Ray traced in action you can look at these two videos:

Q3RT Video 1: Mirror Idfun.de | Mirror FileShack
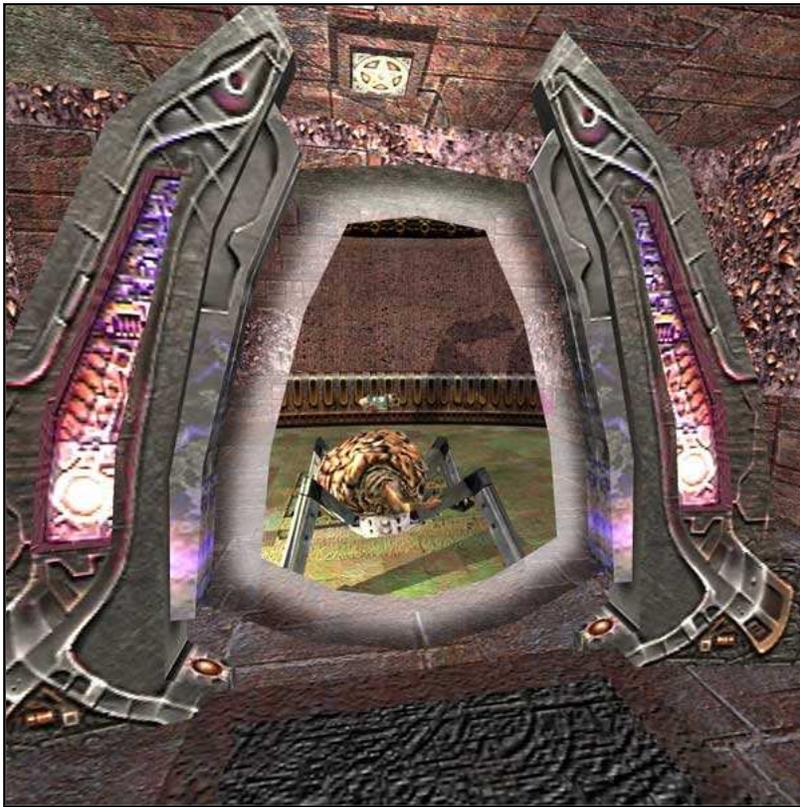
Q3RT - Video 2: Mirror Idfun.de | Mirror FileShack

Quake 4: Ray traced

For my master thesis I wanted to analyze how well a modern 3D Shooter from the year 2005 would perform using ray tracing. For this purpose Quake 4 by Raven Software and id Software has been chosen.

An algorithm was invented for this work that uses rays for collision detection (CD). A polygon-exact CD is quite easy for direct weapons. Only one ray is needed to determine the target. Shooting this ray works through the exact same mechanisms and data structures as shooting a ray to get a color value in rendering.

(Polygon exact collision detection using ray tracing)

For the player model a bounding sphere was approximated through many rays like in a radar system.
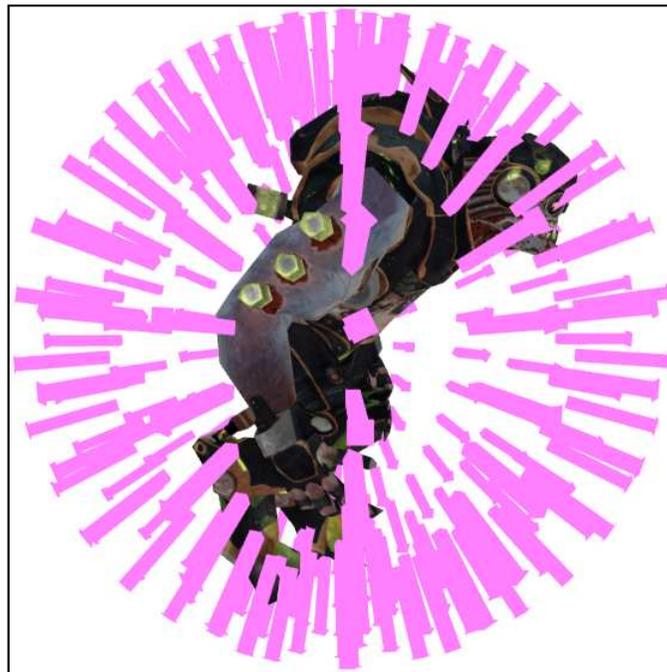
(Bounding sphere approximated through rays)

An interesting special effect which I want to present here in more detail is water. As we expect it from nature water should reflect the surrounding environment and it should be possible to look through the water with some refractions going on. Water in motion should not look flat; there should be some visible height differences in the waves. I want to present you some examples of water in games that we have seen the last years. Of course these "optimizations" had to be done to make the game render fast enough, so this should not be a critic of the game or company itself. It should just show how water looks today in some games, what it lacks and how it could look in ray tracing games.

("Far Cry" (2004): The reflection in the water shows only the mountains, not the trees.)

("Far Cry": Taking a close look at the reflections you see that the resolution is lower then the rest of the world.)

("Gothic 3" from 2006: Quite unspectacular water without any reflections)

The water in "Quake 4: Ray traced" (http://www.q4rt.de/) uses an animation set of many normal maps to simulate the height differences from the waves. One ray is used for reflection on that normal maps, one ray is used to get the refraction through the water. The result is nice looking water.

(*"Quake 4: Ray traced": The water reflects the environment and the player*)

Samples from Q4RT in action can be seen in this video:

Q4RT - Video 1: Mirror Q4rt.de | Mirror FileShack

Performance and Conclusions

So why don't we see raytracing right now in games? The problem still is performance. Rendering all these effects through the CPU is not as fast as using special purpose hardware like current graphic cards for the rasterization algorithm. But the evolution of CPUs is fast and comparing the CPU power available for development in 2004 with Q3RT with the CPU power available now is by more than a factor of 4. Intel's new quad core is out and the efficiency using the same CPU clock rate is about 30% higher then before.

One big advantage of raytracing is that it is perfect for parallelization. As explained in the introduction for every pixel of the image a ray is shot through the 3D scene. So if you render an image with 640x480 pixels, you have about 300,000 rays. Each of them can be calculated independently of the others. This means: The image could be split up into four parts and every core of a quad-core CPU can go ahead and calculate the color values of the image without waiting on any of the others cores for intermediate results. Therefore the scaling of performance with the number of cores in Quake 4: Ray traced with OpenRT on Intel's quad-core "Core 2 Extreme QX6700" is great. The following benchmarks were taken at a resolution of 256x256 pixels on the Quake 4 map "Over the Edge (Q4DM7)".
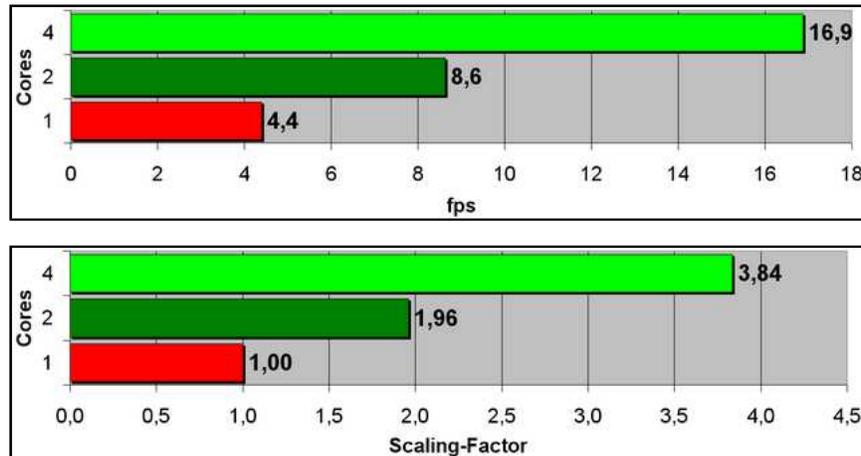
Problems

Unfortunately ray tracing does not only offer advantages. Right now there are some performance issues regarding to rendering dynamic, randomized changes of the 3D scene that cannot be precalculated. But research on this topic will continue and in the area of dynamics for skeletal animation (e.g. for player models) there are already promising solutions (http://www.mpi-sb.mpg.de/~guenther/modecomp/).

Future

Ray tracing has the potential to become the widely used rendering technology on desktop computers. The number of CPU cores is increasing and special purpose ray tracing-hardware-prototypes (http://www.saarcor.de/) show impressive results in speed improvement. It is still a long way to playing computer games in graphics like the "Lord of the Ring"-movies, but we are getting closer to it.

About the author

Daniel Pohl (http://www.idfun.de/temp/q4rt/Daniel%20Pohl/index.html), 26 years old, has recently completed his study of computer science at University Erlangen in Germany. Right now he is looking for a job in the computer games industry or a sponsor for continuation of his research in the area of ray tracing games.

Editor's note: our sincere thanks goes out to Daniel for sharing this article with our readers at PC Perspective. Please send him some comments and thoughts on the article, or leave them in our forums. Hopefully Daniel's research and investigations into ray tracing will produce even more fantastic results!