Note 07 | 15 June 2010

Talking raytracing to dad ...

So you really want me to explain "raytracing" to you? No kidding? Well then, be prepared for a bit of mathematics, even though I will try to keep this down to a minimum. Programming, you must know, is algorithmics. And algorithmics is maths. And maths is easy once you start thinking very precisely about your subject matter. However, since they prevent us from rigorous and critical thinking, we are not accustomed to this sort of thinking. Therefore, we prefer being vague and wave our hands instead of think hard. But you blame that deplorable state of affairs on mathematics instead of the school system, the church, and the political system ...

Anyways. You have asked what raytracing is. So, what is it? Well, it's tracing rays. You trace rays in order to force the computer – or, better, your program – to calculate an image and display it.

For sure, you know what you do with your camera when you take a picture of your family. You hold up the camera, point it in direction where we stand, and when you think to have arranged everything beautifully, you press a button. That's all.

Whatever pressing that button is doing, raytracing is the same but with the computer instead of a camera. The only difference is, it is done the other way around, and not in physical life (your body) but in metaphorical life. What that is, you should look up in the dictionary.

You know what the camera is doing? Right, it's collecting light, and the light does something inside the box of the camera. In the old days, there was something like a strip of paper, only it was not paper, but a material you had to keep in total darkness, and you had to take it to the store for some chemical treatment. You know. And in the end, you got your pictures.

With digital cameras, this has become simplified, I believe. But I don't really know. Because mysterious it still seems to be. Raytracing is definitely different because of the metaphor. So try to think, just think.

First thing to think: what is a *ray*? Well, isn't it something that starts at some point in space, wherever you want it to start, and from there runs right into infinity, always straight, as straight as it could be. A sunray, for instance. Or radiation. Nuclear fallout. But that's way too real. A ray is just a straight line, but one that has a start and a direction and no end.

Now, raytracing is photocamera shooting the other way around. In the case of the camera, rays of *light* are collected as they are bouncing around at the speed of light wherever you have light, and that's almost everywhere. (With the exception only of James Turrell's works. But even there is light. If you don't know his installations, make an effort.) The rays in raytracing are different: they come from the camera and leave it into the scene you want to take a picture of. I told you, it is the opposite of what you expect. It's not physics, it's algorithmics.

Frieder Nake: OAS building Linzer Str. 9a room 3015 | (0421) 218-3525 | nake@informatik.uni-bremen.de
Office hours: Wednesday 12-14 | www.agis.informatik.uni-bremen.de

When you want to raytrace a scene into an image, first thing is you don't start from a real scene, one that you could walk through. Instead, you need to have a virtual scene, one that could exist but does not other than inside the computer. It is a *model*. It is data. Structures of data. Mathematics. It is in your head. But it's inside the computer also. It's unreal, but unreal in a real sense. It is, if you see what I mean, in a state of "as-if". Great, right? Everything on a computer is an as-if.

What is a family scene in real life is a *description* of a family scene in the world of as-if. Even more: it is an algorithmic description of a family scene. A description is algorithmic if the computer can read it and operate on it.

The scene in our case consists of objects that have shape and color and location and, perhaps, a few more features. The scene also contains one or more light-sources. Descriptions of light-sources, to be correct. Plus it has a description of a camera situation. That is the location of the camera (called the center of projection), and the size of the image plane, its distance from the center of projection, and its orientation in space. You will understand, I am sure, that all this is needed before raytracing can do anything. All these descriptions (models) are necessary in the world of as-if, before anthing can happen there.

What will happen needs one last step in preparation. Have you heard of the *pixel*? No? Well then, we need a bit of education in postmodern times. A pixel is a picture element. Since computers can only deal with finite amounts of things, the image plane must be divided up into so many picture elements. You know that because the monitor of your laptop has, perhaps, 1024 by 768 pixels. That's called the resolution. But what is important is the following statement that should appeal to you: *The image of the scene will be completely determined. when each picture element has been given a color*.

So the task of calculating an image from a scene is equivalent to determining, for each and every pixel, its color value – according to the description of the scene. Raytracing then is *one* method of doing just this. But how?

Your program takes three steps to do just that:

(1) it defines one *ray* from the center of projection through a pixel of the image plane and into the scene;

(2) it uses this ray to calculate the *visible* spot of the scene;

(3) it uses even more rays to collect from the scene the *color* of the visible spot.

Is this easy to understand? Clearly, you only see that that is visible. Therefore task no. 2. If a spot of the scene is visible, it must be of a certain color, otherwise it cannot be visible. Well, every spot of the scene is of some color. The visible spots are no difference. But their color must be determined. Because, if they are visible, we see them. So they appear in the image. Therefore task no. 3.

The first ray, from the center of projection through the pixel and into the scene, is called the *primary ray*. That's not important but I tell you anyhow, because we can then talk more intelligently about the situation. Primary rays solve the visibility problem. The other problem is the coloring problem. It is solved by using *secondary rays*. Great surprise.

I don't want to bother you with undue detail, but you will agree with me that secondary rays will be used, starting from a visible spot, to probe the scene as much as ever possible. Each of those probes should deliver a contribution from the rest of the scene to the final color value at the visible spot. In principle, you may admit, light may get to a visible spot from everywhere in the scene. The effect that all rays of light generate at a visible spot, is called its color. The secondary rays are used to approximate the set of all rays arriving at the visible spot.

So the last indication I want to make is on how these probes of secondary rays in raytracing are the complete reversal of the light rays in the case of photography. When, in the photography situation, the camera collects *light* rays, in the case of the computer image the computer is shooting *sight* rays (pun intended). From light rays collected to sight rays shot. That's

the difference. It is the essence. Once you accept it, everything else follows suit. Rays are bouncing off objects they hit in the scene. They also infiltrate objects when they hit them. A visible spot may also receive light from a light source. Metaphorically, we may say that spots may be seen from the location of the light. This then is the third case of secondary ray.

We call the cases of secondary rays, dear and poor dad: reflected ray, refracted ray, and shadow ray. Each one is dealt with in the same way as the original primary ray. A whole cascade of secondary rays is shot into the scene. We call this tremendous algorithmic effort *recursive* raytracing. If you program this, you will discover the beauty of programming, and the power of recursive thinking. But that's another story. And you have listened to me long enough.

Let me only add to this how you elegantly determine the visible spot. Remember: it needs the primary ray. So if you embark on the *gedanken-experiment* of following a primary ray, what happens? You follow your ray. Only two events are possible as you do this: either the ray disappears into the background, or it hits an object and, perhaps, even more. So the primary ray hits 0 or 1 ore more objects. Zero objects is clear: background color. Otherwise, one of the objects must be the first to be hit. That's where you have the visible spot.

Enough of this. I had warned you. There is a lot more of technical detail. I will not bother you with it. Do you see how simple and how great computer science and computer graphics is? I guess you deserve an example of a ray-traced image. And tell mom. She will love you.

A raytraced image. Found on the internet. So it's free.