# Ray Tracing

By Ian Graham

Humanity has always been fascinated by the interplay of two fundamental aspects of existence as we know it: the depth of the world around us, and the flatness of our vision. While our minds have learned to extract a third dimension from the images our eyes give them, there is nothing inherently three dimensional about them. That is to say, the depth we construct from these images is nothing more than an educated guess: our brains, armed with billions of years of evolution, a lifetime of experience, and two perspectives (one from each eye), can estimate what the scene before our eyes truly looks like with incredible accuracy. But it's not perfect. The act of tricking this intuition is millenia old and embedded into our very beings. Examples of this include the Parthenon's floor and columns, which are curved to create a false perspective, M.C. Escher's optical illusions, and, relatively recently, 3-D renderers.

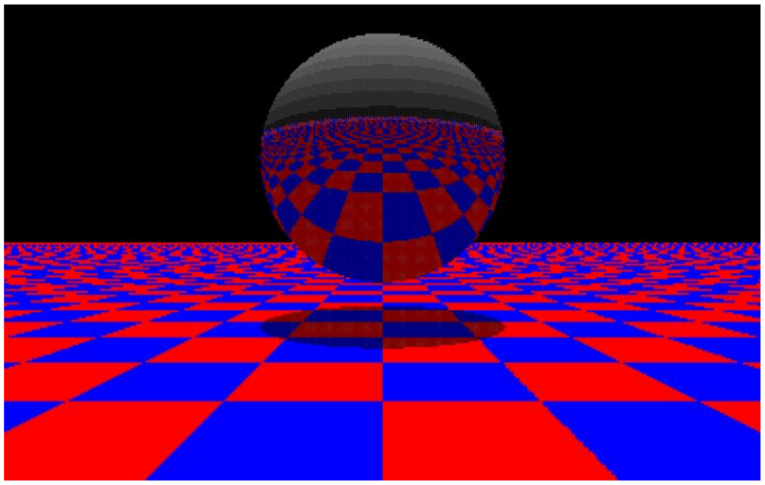

Figure 1: A ray-traced image rendered on Ti 84

That last example is especially interesting, because its aim is purely to immerse the mind into an existence with depth which does not exist. It creates perspective, which, unlike that seen in paintings and other man-made artwork, is not just an imprint of a scene found in someone's mind, is not the result of intuition, it's the result of strict mathematics and intertwined formulas. 3-D renderers photograph scenes which don't exist and worlds we can only imagine. While we take them for granted today, they undeniably give a truly and uniquely human interface to computing which is relatively new. Computer-aided design, video games, 3-D artwork, and animation simply would not be what they are today without 3-D renderers. All of this, while intriguing, does not get at the heart of the question I hope to answer: how do 3-D renderers work?

There are several potential answers to this question, but in this article, I'll only cover ray tracers. There are a few reasons for this. First, ray tracers are by far the most intuitive type of 3-D renderer, since they're based on simulating how light acts in the real world. Second, they produce the highest resolution images. While other, more common types of 3-D renderers, such as rasterizers, use lots of shortcuts to avoid unnecessary computations and make fast, real-time rendering possible, ray tracers do not, which means that they preserve details which other 3-D renderers might ignore. Lastly, ray tracers utilize all the same concepts as more common forms of 3-D renderers. All 3-D renderers work on the same foundations dictated by optics, which ray tracers exhibit more completely than any other form of 3-D renderer, so by learning how ray tracers work, you also learn how other 3-D renderers work, save for a few details and optimizations.

Bearing in mind that ray tracers simulate how light acts in the real world, we can begin to devise a basic algorithm based on the world around us. The first thing to consider is what light is. Physicists have argued about this for centuries, but, thankfully, we are not bound to the intricacies of the real world. Instead, we just have to create a way of mathematically describing how light behaves at a macroscopic level. The immediately obvious tool to use are 3-D vectors. A 3-D vector is a collection of three numbers which can be used to describe a point in space, and, consequently, the direction towards it. Not only that, but 3-D vectors in computer graphics can also be thought of as red, green, and blue components of a pixel, meaning that with 3-D vectors we can describe both how light moves and what the color it is. The next thing we must consider is what it means to see something. In the world, we see things because light travels from a light source, passes through surfaces or reflects off of them, and occasionally reaches our eyes. This immediately presents a problem to us: while light sources in real life emit unfathomable amounts of photons in all directions, our computers have limited resources and can't efficiently track all of them. But our intuition is not useless! We know that we only see light that enters our eyes, and therefore we can safely ignore any light which goes in other directions! To eliminate light which doesn't intersect our eyes by having all of our vectors originate from the same point, which will represent our eyes. This means that we can define a point in 3-D space as the viewer, and 3-D vectors originating from it as light we see. If we can compute the color of each of these rays we will know which rays of light reach our eyes and what color they are.

We must also decide how we're going to extract an image from this model. We can't look at only the point we're viewing from, since a single point can't be a 2-D image. Instead, we will look at the cross section of the rays before they hit our eyes, so that our image can distinguish between rays which intersect at the eye. With the model we have just created, we have a very basic, but correct, model of space and vision that we can begin to elaborate on.

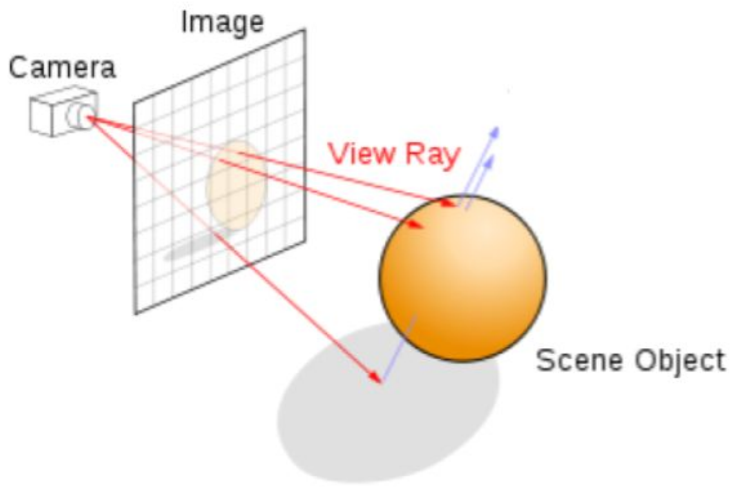

Figure 2: A visualization of the Ray Tracing Model (Source: Wikimedia Commons)

Now, we just have to find a way to describe the scene we want to draw. We might initially be tempted to try to 'pixelate' this 3-D scene as we sometimes do 2-D ones, dividing the space into small regions and describing each one individually (whether it be by color or some other property), but the math looks a lot simpler if we instead use equations to describe our scene. For instance, the points which satisfy the equation $(x-x_1)^2+(y-y_1)^2+(z-z_1)^2=r^2$ form a sphere of radius r centered at $(x_1, y_1, z_1)$. By using this equation, we can check whether or not and where a given ray intersects that sphere with a simple quadratic equation and some vector math. Describing a plane, a triangle, or a disc through equations allows us to describe complex scenes easily.

With the model we've created, we can go through each pixel, cast a ray through it, and see what kind of object, if any, that ray intersects, and where it intersects it. With that information, we can construct a new ray, originating at the point of intersection and bouncing off in a new direction. By creating a new ray, we can create reflections, and by combining the colors of all the objects our ray and its reflections intersect, we can discover the color of a specific pixel. Additionally, if we change how we create new rays after intersections, we can create the illusion of having different materials, for example, if we create imperfect reflections, which bounce off of angles that are slightly off, we can create blurred reflections or a matte surface. Similarly, we can not reflect the ray at all and instead refract it, creating a dielectric surface.

The math behind 3-D rendering and the tricks it uses to generate complex, accurate images goes far beyond what I've described here, and I encourage you to delve into the fascinating depths of how ray tracers behave, but I hope this brief introduction has given you some appreciation and understanding for what goes on behind the curtains of the software we use daily.

## Works Cited

1. Shirley, Peter. Ray Tracing in One Weekend. 2018, Ray Tracing in a Weekend, www.realtimerendering.com/raytracing/Ray%20Tracing%20in%20a%20Weekend.pdf.

2. (n.d.). Retrieved June 10, 2020, from https://www.cs.unc.edu/ rademach/xroads-RT/RTarticle.html

3. Blankenbehler, B. (2016, October 18). How Greek Temples Correct Visual Distortion. Retrieved June 16, 2020, from http://www.architecturerevived.com/how-greek-temples-correct-visual-distortion/