CS 314: Finishing Raytracing

Robert Bridson

October 23, 2008

1 Raytracing Hierarchical Models

Let's take a quick break from shading to pick off another miscellaneous raytracing topic.

Earlier we saw the attraction of hierarchical modeling for a Z-buffer renderer: complex geometry can be stored immutably in object space, and transformations can place as many instances as we'd like in world space anywhere. All of these advantages can apply to a raytracer as well.

It was a simple matter of just multiplying in extra object-to-world space transformations in the Z-buffer context to render geometry stored with object space coordinates. We need to do something a little different for raytracing.

However, we've already seen how to transform rays from one coordinate system to another: we built rays in camera space and used the inverse of the modelview matrix to transform them into world space. To check a world space ray against geometry stored in object space, we simply need to use the inverse of the object-to-world transformation to transform the ray to object space. We can do the intersection test there; if there is an intersection and further processing is needed, it's probably best to transform back the intersection point to world space (e.g. to allow for simple shading with lights stored in world space).

Note also that this allows BVH's to be built once in object space and left unchanged, even if rendering multiple images with the object moving to different positions; this can be a pretty useful optimization.

2 Transparency and Refraction

Getting back to shaders, a less common but still often important class of materials are those that are **transparent** (at least partially): things like glass or acetate or water which let light rays pass through them.

Note that we don't include here examples such as leaves or milk or paper, which do allow light through but not directly: these are better classed as **translucent**, implying a clear image is not visible through them since light is scattered as it proceeds through.

This is another nice case for raytracing; if a ray hits a transparent surface, we trace a secondary ray on the other side of the surface to find what light shines through the surface. (If there's no refraction, this is just the first ray continued on, which allows for some possible optimizations.) Most transparent materials won't transmit all the light through, however, as they also reflect or absorb a fraction of the light: to model this we can include as a material parameter an RGB vector with fractional values between 0 and 1 which we scale the incoming light by, just like we did with mirror reflections.

Without refraction, transparency effects can be done with some effort in a rasterization algorithm—though a sort for the painter's algorithm is necessary, as the Z-buffer doesn't help in this case. Look up alpha blending modes in OpenGL for more on this if you're curious.

With refraction, things get much more interesting: nearly impossible for a rasterization renderer, but as simple as ever for raytracing.

The physics underlying refraction is the fact that light travels at different speeds in different materials. The **refractive index** η of a material is the ratio of the speed of light in a vacuum (approximately $3 \times 10^8 m/s$) divided by the speed in the material. You can easily look up this value for common materials (e.g. $\eta \approx 1.0008$ for air at normal conditions, $\eta \approx 1.3$ for water at normal conditions, $\eta \approx 1.5$ for common types of glass, $\eta \approx 2.4$ for diamond[1]). When a ray of light hits the surface between two materials of different refractive indices at an oblique angle (not perpendicularly), it has to bend. This is most easily seen in terms of a wave model of light: an advancing wavefront coming in at an angle will change speed where it hits the surface and necessarily develop a kink there. The change in angle is fairly simple to derive geometrically, leading to **Snell's Law**:

$$\eta_1 \sin(\theta_1) = \eta_2 \sin(\theta_2)$$

Here η_1 and η_2 are the refractive indices of the two materials, θ_1 is the angle of the ray relative to the

[1]This is a very high value compared to most other transparent materials, in particular just about any other gemstone, which is one reason why diamonds are so prized, and can be distinguished from fakes just by inspection.

surface normal on the material 1 side, and θ_2 is the angle of the ray relative to the surface normal of the material 2 side.

There is a caveat here though: if the ray is coming from a "slow" material to a "fast" material ($\eta_1 > \eta_2$) at an oblique enough angle, it may be that $\eta_1 \sin(\theta_1) > \eta_2$, and thus there isn't any possible angle θ_2 which will make Snell's law work. No light is transmitted through the surface in this case, but instead the incoming light has no choice but to reflect off the surface. This is termed **Total Internal Reflection**, and underlies fibre optics (which keep a beam of light traveling inside a possibly curved glass fibre).

With a little more effort, we can use Snell's law (and the check for total internal reflection) to derive the secondary ray corresponding to a primary ray hitting a refractive surface, building refraction into a raytracer almost the same way we did mirror reflection.

Refraction has some extra complications in the real world that can be modeled with rather more effort in a raytracer. For example, the refractive index usually has some dependence on the wavelength of light traveling through the material—meaning some colours of light will bend at different angles. This is the reason for rainbows appearing in the sky or out of prisms: white light (containing many different wavelengths) is refracted by water droplets or the glass of the prism, beding each component a slightly different amount so the separate wavelengths end up spreading out.

In addition, in some cases the refractive index can vary continuously throughout a volume. For example, cold air has a higher index of refraction than hot air, so if the temperature changes smoothly then light rays may curve (get smoothly refracted) as they travel through an expanse of air. This is visible as the shimmering above a hot car or pavement or aircraft exhaust, as mirages over very hot ground, or in reverse as higher-than-normal horizons over ice packs in the sea.

Incidentally, refraction applies to any wave phenomena where the speed of the waves can change. For example, the speed of waves in a shallow body of water is roughly proportional to the square root of the depth, meaning as the water gets shallower the waves slow down. You can see the effect of this at a beach: waves coming in off the ocean slow down as they come in, getting refracted as they go, until they bend around enough to be nearly perpendicular to the beach no matter what direction they were traveling in further away. (The other effect is that as they slow down, the water has no choice but to pile up—kind of like a traffice jam—and the waves get steeper until they possibly curl over and break.)

3 Glossy Materials

We've seen two opposite material types: Lambertian materials (which reflect an incoming light ray into every outgoing direction uniformly) and mirror reflectors (which reflect an incoming light ray into exactly one outgoing direction). Most real materials lie somewhere in between. An incoming light ray will be reflected into all directions, but most of the energy ends up leaving in directions close to the mirror reflection instead of being uniformly distributed. The degree to which the outgoing energy is concentrated near the mirror reflection defines the **glossiness** of the material, or how **specular** it is.

The simplest and most popular glossy material model is the **Phong** model: it's heuristic in nature, not being derived from physical principles, but often works well. For an incoming light direction $\vec{l}$ and an outoing ray direction $\vec{d}$, we figure out the fraction of light reflected along $\vec{d}$ based on the angle between $\vec{d}$ and the perfect mirror reflection direction

$$\vec{r} = \vec{l} - 2(\vec{l} \cdot \hat{n})\hat{n}$$

where $\hat{n}$ is the surface normal as usual. We take the cosine of the angle, i.e. the dot-product $\vec{r} \cdot \vec{d}$, which is highest when the vectors are parallel, and raise it to some exponent $p > 1$. The fraction of light reflected along $\vec{d}$ is proportional to:

$$(\vec{r} \cdot \vec{d})^p$$

Again, we can scale this with a material property for each RGB component. The exponent p controls how glossy the material appears. As $p \to \infty$, the limit becomes a perfect mirror, since then the term $(\vec{r} \cdot \vec{d})^p$ ends up zero unless $\vec{r} = \vec{d}$. As p gets smaller, the result looks softer, more matte. The classic demonstration of the effect of p is to render a sphere under a point light source: the light source shows up on the sphere as a **specular highlight**, whose width is larger the smaller p is.

We can use the Phong model in a renderer much the same way we handle diffuse materials: for each light source we compute the appropriate dot-product raised to the exponent p (possibly after a shadow ray check in a raytracer).

4 Emissive Materials

The simplest material model of all is **emission**. This handles materials which glow, giving off their own light. Usually we assume it's emitted uniformly in all directions. For the Z-buffer approach and the simple raytracer we've been developing, this simply means adding a constant colour (an RGB vector storing the amount of emission from the object) independent of surface normal, view direction, lighting, etc.

This is particularly useful in these sorts of renderers to model light sources that could show up in the final image. We've discussed distant lights and point light sources, for example, and their effect on rendered surfaces in the scene, but we haven't handled what happens when these light sources are in view of the camera. Since we only render geometry, not lights, they don't show up. This can be remedied by sticking in surrogate geometry with an emission colour, for example a brightly glowing light shade that is supposed to be the visible representation of a point light source in the scene. (In the context of a raytracer, however, you might want to add special checks to the code to make sure the surrogate light geometry doesn't end up casting inappropriate shadows from the underlying point light source—making it invisible to shadow rays pointed there, for example.)

5 More Exotic Materials

We've only scratched the surface of models for how materials reflect or transmit light.

For example, the class of retroreflective materials where incoming light is mostly reflected back opposite the incoming direction instead of the usual mirror reflection direction is widely used in road signs and paint for safety reasons, and also comes up naturally in animal eyes and the surface of the moon.

Many materials that generally appear to be quite matte actually have strong mirror reflectance when looked at a very oblique angle (almost parallel to the surface): this is **Fresnel reflectance**. Reflectance can also change depending on other angles leading to "anisotropic" materials, such as silk Christmall balls or hair, where the direction of fibres plays a huge role in how glossy it appears. Translucent materials such as skin and marble (and fluids such as milk or clouds) also allow light to enter but then scatter around internally before emerging at a *different* location at a different angle. Some materials also behave differently depending on the polarization of the light, or the wavelength—and in the case of fluorescent materials (which includes most white paper and laundry detergent as well as the more obvious neon highlighters etc.) can transform an incoming colour to a different outgoing colour.

6 Path tracing

We have hit several cases so far where we had to make approximations or give up in our standard recursive raytracer because infinitely many light rays get involved. For example, when a light ray hits a diffuse surface it scatters off in infinitely many directions uniformly; we ignored most of these, and modeled what was missing with the ambient light hack. We skipped over area lights because they emit light from infinitely many points. We skipped a treatment of real camera systems with lenses, apertures, and depth-of-field since it involved tracing infinitely many light rays that converge onto a pixel—and we only briefly talked about the fact that a pixel really represents an area which has infinitely many ray origins to trace out from.

There are more sophisticated raytracing techniques which can handle all of these cases and more. The most fundamental is called **path tracing**. The idea is to approximate the effect of infinitely many rays (wherever they appear) by randomly sampling an adequately large but finite subset of them and summing or averaging over this sample. The same statistical technique is of course used in many other areas: political polls don't question every single voter, but instead select at random a small subset of voters which hopefully will be representative, giving an accurate estimate of the true result. To make path tracing work requires being a lot more careful about physical principles such as conservation of energy (making sure that the amount of light reflected off a surface isn't greater than the light shining on it), along with a good understanding of statistics; we won't cover it in this course, but you should know it's doable.