1 Offline Graphics

So far we have focused on interactive (real-time) graphics using the GPU on the graphics card. If we have a mirrored object, it cannot reflect other objects in the scene. Instead, we "fake it" by having it reflect a *picture* of the other objects, represented as a texture. The GPU usually can handle only triangles, not curved objects, but we "fake it" by using smooth shading.

Interactive graphics is necessary for video games. However, if you are making a movie, then you can spend as much time as you want to make each frame. You don't need to make each frame in 1/30 second. In fact, it is pretty standard to use an hour or more per frame!

In class today, I will demonstrate a scene with mirrored curved objects, a cylinder and a sphere, in *Blender*, on open-source modeling and animation package.

The modeling and setup in Blender is done using the GPU so we can move objects around and changes views in real time. Once we have everything set up like we want it, we click render and seconds to minutes later we have a more realistic image generated using ray tracing.

By the way, 2020 was supposed to be the "year of GPU ray tracing". We purchased computers that will be able to do ray tracing on the GPU. Alex and Jose are working on a presentation.

1.1 Ray Tracing

Off-line rendering often uses *ray tracing*. Each pixel in the image corresponds to a ray in the world. When that ray hits a mirror, it bounces according to the usual rule, to make another ray. Ditto until the ray leaves the scene, hits the light, or hits a non-mirrored object. Transparent objects use Snell's law. A piece of glass might send a reflection and refraction ray. For a non-mirrored object, send a ray towards each light. If it hits something else on the way, it is shadowed from the light. If it reaches the light, apply the texture mapping, and the usual shading at the intersection point. That color determines the color of the original pixel.

Ray tracing is time-consuming because it has to be done for each *pixel*. Each time the ray bounces, we have to do essentially the calculation you are doing to determine the location of your mouse click on your robot. That involves visiting all the objects and faces. Algorithms of Computational Geometry can be used to speed this up.

1.2 Details of Ray Tracing

As you know, a pixel (x, y) corresponds to a front point f and a back point b on the clipping cube. For each object, We transform these into model coordinates and set $q = f$ and $u = b - f$. The ray is $q + su$ as before.

We know how to figure out if it hits a face of a polyhedron. What about a sphere or cylinder?

In model coordinates, the sphere is the unit sphere: center $o = (0, 0, 0)$ and radius $r = 1$. A ray intersects a sphere if closest point to o is closer than 1. The closest point $p' = q + s'u$ satisfies $(p' - o) \cdot u = 0$,

$$(q + s'u - o) \cdot u = 0 \qquad \rightarrow \qquad (q - o) \cdot u + s'u \cdot u = 0 \qquad \rightarrow \qquad s' = -\frac{(q - o) \cdot u}{u \cdot u}.$$

If $(p' - o)^2 > 1$, then it does not hit the sphere. Otherwise, let $t = \sqrt{1 - (p' - o)^2}$. If $s' - t < 0$, that means the first hit is behind us which means we are inside the sphere. We also could have checked if $(q - o)^2 < 1$. If we are outside the sphere, the first hit is at $q + (s' - t)u$. Otherwise it is at the second hit $q + (s' + t)u$. (If $s' + t > 1$, then we can't see that either, BUT it still is on the ray and may bounce back into the visible region.) Set s for that object to $s' - t$ or $s' + t$ accordingly.

If the model is a cylinder, it is $x^2 + y^2 \leq 1$ and $-1 \leq z \leq 1$. To calculate the first hit with the cylinder, set $q_z, u_z = 0$ and use the math above. The point $p = q + sv$ lies on the side of the cylinder if $-1 \leq p_z \leq 1$. To calculate if it hits the top, solve $q_z + su_z = 1$, $p = q + su$, set $p_z = 0$, and check if $(p - o)^2 < 1$. For bottom, solve $q_z + su_z = -1$, etc. Use the hit with the smallest scalar.

Finally, the first object hit is the one with the smallest s. We have done this for dragging, but now this point (in world coordinates) becomes the new q. For a mirror, the new u is the reflection of the old u. To reflect, we need the normal. We know the normal to a sphere at point p is $p - o$. For the side of a cylinder, we set the z-component to zero. For the top or bottom of a cylinder, it is $(0, 0, 1)$ or $(0, 0, -1)$. These are in model coordinates. We transform the normal to world coordinates (how???).

If the object is glass, we will also use Snell's law to create a refraction u. Recurse for each u and add the results with a weighting of R times reflected plus $T = 1 - R$ times refracted. The coefficients of reflectance R and transmission T are about $R = 0.08$ and $T = 0.92$.

1.3 Global Illumination

Just ray tracing from the eye doesn't handle a diffuse surface lit by the reflection of a light off a mirror. For that you would have to ray trace from each light. So we could ray trace in each direction, reflecting or refracting until it hits a diffuse surface. This still would not handle ambient shading: diffuse surfaces lighting each other. If a ray of light hits a diffuse surface, it is reflected equally in all directions, so ray tracing is not feasible except as a *Monte Carlo* method that makes a repeated random choice.

Instead, suppose we break each surface up into patches of unit area. Patch i is at position p_i, has normal n_i, emits light e_i (zero if it is not a light source), and has reflectance r_i (its k_d) and has outgoing light o_i. Visibility factor v_{ij} is 1 if p_i sees point p_j and is 0 if there is something between them. Let $u_{ij} = (p_j - p_i)/|p_j - p_i|$. The outgoing light satisfies,

$$o_i = e_i + r_i \sum o_j (n_i \cdot u_{ij})(n_j \cdot u_{ji}) v_{ij}.$$

We can simplify this to

$$o_i = e_i + r_i \sum f_{ij} o_j \qquad \text{for} \qquad f_{ij} = (n_i \cdot u_{ij})(n_j \cdot u_{ji}) v_{ij}.$$

This is actually just a big matrix equation,

$$o = e + rFo \qquad \text{with solution} \qquad o = (I - rF)^{-1}e.$$

It is usually much too big to solve this way. However, since r_i is less than one, one can usually just set $o = e$ and iterate $o = e + Fo$ a few times. Various techniques iterate more for o_i that need it.