Lecture 6: Non-Canonical Cameras and Ray-Sphere Intersection

Last time we played with a "canonical camera":

- Eye is at the origin (0, 0, 0)
- Looking down the negative z axis
- Viewport is aligned with the xy plane
- viewport height and width are 1
- distance from eye to viewport is 1
- image height = H = image width = W

We imposed a (u, v) coordinate system in the image plane and determined the uv position of pixel (i, j).

Warmup exercise:
1a. You have a canonical perspective camera (i.e., centered at the origin, looking at the -z axis, viewport height and width 1, distance to viewport 1). Your image is 400x400 pixels. The (u, v) coordinates of the pixel are $(0.2, -0.1)$. What is the viewing ray of this pixel? Write the ray in parametric form (i.e., $\mathbf{p} + t\mathbf{d}$).

1b. Out in the scene, there is planar object occupying the entire plane at z=-6. What is the value of t at the intersection point of the ray with the plane?

Then, we had the viewing ray for that pixel was:

$\mathbf{p} = (0, 0, 0)$
$\mathbf{d} = (u, v, -1)$

Let's see what might change:

- viewport height and width - just a scale factor on the u, v formulas.
- **Exercise:** what if distance from eye to viewport is d ?
  - the z coordinate of **d** becomes $-d$.
- eye position changes - ray origin = eye position
- eye **orientation** changes - this one requires some math

Camera Orientation

The idea here is to take advantage of all that work we did for the canonical camera. To so that, we're going to define a coordinate system (i.e., a **basis** plus an **origin**) in which the camera's position **is** canonical:

- The origin is at the eye
- The $\vec{u}$ axis points to the right in image space
- The $\vec{v}$ axis points towards the top of the image.
- The $\vec{w}$ axis points *backwards* directly away from the viewport.

This is just like what we did last time with the canonical camera. In that case:

- $\vec{u} = (1, 0, 0)$
- $\vec{v} = (0, 1, 0)$
- $\vec{w} = (0, 0, 1)$

And the view ray's direction is, formally:

$$\mathbf{d} = u\vec{u} + v\vec{v} + -d\vec{w}$$

The key thing to observe here is that if we change (u, v, w) to a different basis, this same thing applies.

Suppose "right" from the camera is in the xz plane midway between the +x and -z axes

- the $\vec{u}$ basis vector would be $(\frac{1}{\sqrt{2}}, 0, -\frac{1}{\sqrt{2}})$
- the $\vec{v}$ basis vector would be $(0, 1, 0)$
- the $\vec{w}$ basis vector would be $(\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}})$

To find the view ray, we can start by computing u and v values as we did before - in the $(\vec{u}, \vec{v}, \vec{w})$ coordinate system the camera **is** the canonical camera, so this calculation is the same.

To get the ray in the global x, y, z coordinate system we simply need to apply the same equation as above to change the basis:

$$\mathbf{d} = u\vec{u} + v\vec{v} + -d\vec{w}$$

Generating a camera basis

Setting the $(\vec{u}, \vec{v}, \vec{w})$ basis vectors manually works, but isn't the most intuititve. Often we want to specify a camera in terms of more intuitive things like the eye and view direction, or the eye and a point the camera is looking at.

Both of these have an ambiguity: the camera could roll along its viewing axis and satisfy both, so usually we also supply an "up" vector. This is not necessarily the $\vec{v}$ axis, but a vector that points "up" in the scene so the camera is "level".

So given three 3-vectors, specifying:

- **eye position**
- **view direction**
- **up direction**

How do we get a u, v, w basis?

(whiteboard picture for each of the following steps)

First, it's clear that the view direction should be simply the opposite of $\vec{w}$, so let's start with

$$\vec{w} \leftarrow normalize(\mathbf{view})$$

Both other vectors should be orthogonal to $\vec{w}$, and we'd like $\vec{u}$ to also be orthogonal to **up**, so let's pull out the cross product. $\vec{u}$ sould point right, so we want the cross product vectors to be in counter-clockwise order. So

$$\vec{u} \leftarrow normalize(\mathbf{up} \times \vec{w})$$

Now we have our "right" and "back" vectors, so we can use a cross product to get the up-in-image-space vector:

$$\vec{v} \leftarrow \vec{w} \times \vec{u}$$

Exercise: What if instead we were given

- **eye position**
- **at** - the 3D position of the point the camera is looking at (i.e., the center of the image)
- **up direction**

Intersecting Rays with Objects

Once we've calculated the ray for a pixel, we need to send it out into the scene and see what object it hits first. For now, this requires checking for each object whether the ray intersects it. In the earlier Exercise, you intersected a ray with a particular plane. We're going to start here with intersecting rays with **spheres**.

Parametric vs Implicit Equations

Consider a line in 2D. Formally, a line is a particular set of points in $\mathbb{R}^2$. There are multiple ways to write equations that describe the line; this one probably looks familiar:

$$ax + by + c = 0$$

This is **implicit** equation: the defining characteristic is that it gives us an equation that is *true of all points on the line*. Usually for maximum generality we put a zero on one side of the equals sign. However, this is just an algebraic rearrangement of the more familiar $y = mx + b$.

Here's another way to describe a 2D line:

$$r(t) = \mathbf{p} + t\mathbf{d}$$

This is actually vector notation for the following:

$$x = p_x + td_x$$
$$y = p_y + td_y$$

This is just the ray equation we saw last time - and if we don't insist that $t > 0$, this describes all possible points on the line. This is a **parametric** equation, because it uses an extra *parameter* variable, t, and by setting t to different values we are able to **generate** all points on the line.

Generally speaking, it's easiest to find the intersection between two objects when one is parametric and one is implicit. Consider intersecting two lines: if we have one in parametric form, we can just drop the r.h.s of each parametric equation into the implicit equation for the other line and solve a single equation for t.