Lecture 17: Viewing Transformations

Transforming Normals

One last thing to note about affine transformations is that they don't work right on normals. See whiteboard notes for intuition and derivation, but the upshot is that **to transform normal vectors, multiply by the inverse transpose** of the transformation matrix:

$$\vec{p}' = M\vec{p}$$
$$\vec{n}' = (M^{-1})^T\vec{n}$$

Wireframe Rendering using Transformations

Object-order rendering is based around the idea of using **transformations** to get the objects in a scene from where they live (object space) to where we see them through a camera (image space). In fact, since they're composable, we're ultimately going to accomplish this with little more than a single matrix multiplication.

We're going to start with just geometry, and here's the trick: under all the transformations we use, lines stay lines. Which means I can take two points with a line between them, transform just the two points, then draw the line between the transformed points. This is going to give us the ability to render meshes in what's called *wireframe*, where geometry is represented by vertices connected by lines.

In raytracing, we had to do a lot of work to find the closest object. In wireframe rendering, objects are not solid, and therefore they don't occlude each other, so we don't care what's closest. We'll talk later about how to fix this and render solid surfaces using object-order rendering.

Our goal is to get from object space (e.g., the coordinates in which we modeled our cube with side length 2) all the way to an image of the cube. The approach will be to take all the cube vertices, transform them, then draw lines between them in 2D to form the wireframe cube.

How the image appears depends on quite a number of things - the transformation we're going to build is pretty complicated. **brainstorm:**

- Position and orientation of the model in the scene
- Position and orientation of the camera in the scene
- The type of camera and its parameters
    - Orthographic vs Perspective
    - Viewport size and position
- Pixel dimensions of the image

To handle all these variables, we're going to break the viewing transformation up into four pieces:

- Model transformation (per object)
    - moves object to its world-space pose in the scene
    - OR converts object from its local frame to world frame
- View (camera) transformation
    - move the camera to its world-space pose in the scene
    - OR convert everything in the scene from world coordinates to canonical camera coordinates
- Projection transformation
    - squish and warp the scene into a conveniently shaped box (the *canonical view volume*)
    - OR change coordinates to *normalized device coordinates*
    - In both views:
        - the whole viewable scene lives in the box defined by $x \in [-1, 1]$, $y \in [-1, 1]$, and $z \in [-1, 1]$, and
        - the z axis is lined up with each pixel's viewing ray
- Viewport Transformation:
    - transform objects from the canonical view volume onto the image plane
    - convert from (x, y) normalized device coordinates to (i, j), and carry the z coordinate along unchanged (we'll need it later!)

The good news is that we already have the machinery for almost all of these! Next, we'll start filling in the few pieces we don't, then put it all together into a wireframe renderer.