

Lecture 01: Beginning

1. Overview of the class
 - a. A transition from 2D (x,y coordinates) to 3D (x,y,z coordinates)
 - b. Seems like an easy job, but it adds a whole host of complexities.
 - c. Some of the important aspects of making this transition:
 - i. Pixels vs. Fragments (Triangles)
 1. In 2D, the pixel is the basic unit
 - a. By setting the color of all the pixels in an area, we can create shapes, movies, whatever.
 - b. Recall, pixels are just areas of memory connected to the display device.
 - c. We'll still do this in 3D, but we'll use some **abstraction**.
 2. In 3D, we want to create the *illusion* of a 3D object on a 2D screen.
 - a. Just like a photograph or TV.
 - b. However, it is a synthetic 3D scene – we can't just use a camera.
 - c. We do it by constructing 3D *mathematical* models of an object (that approximate its shape and colors).
 - i. And a virtual camera...
 - d. Then, we must convert the 3D model to a 2D set of pixels.
 - e. That's the heart of computer graphics.
 3. Modern 3D GPU's primarily deal with polygonal **meshes**
 - a. A set of connected points
 - b. Often with additional color and texture data associated with some or all of the points.
 - c. These meshes are often (but not always, as we'll see) represented as a bunch of triangles or rectangles that connect the points of the mesh.
 - d. The GPU hardware is very efficient at transforming 3D triangles like this into 2D pixels.
 - i. DirectX and OpenGL are standard interfaces to this hardware.
 - ii. Pixel and Fragment shaders are another layer (that work with DX and GL).
 - e. Internally, there are a small number of mathematical operations the GPU performs over and over to do the 3D=>2D conversion.
 - ii. In this class, we'll be exploring some of the math behind this.
 1. Time permitting, we'll write two types of software renderers this semester
 - a. A renderer in this context is a 3D=>2D converter.
 - b. A software renderer doesn't use the GPU at all.
 2. A ray tracer
 - a. Uses slightly different math than GPU's
 - b. Uses more analytic objects (spheres, torii, planes, etc) than polygonal objects.
 - c. Produces a extremely polished look – but is very slow.
 - d. Usually what you're seeing in pre-rendered cut-scenes.
 3. A rasterizer
 - a. More like what the GPU does.

- b. Converts all 3D triangles to 2D and determines which are covering which.
- 2. Coordinate System
 - a. 1D Coordinates
 - i. Origin
 - ii. Axes
 - iii. Offsets (+5, -3)
 - 1. A number as an offset from the origin.
 - b. X and Y first.
 - i. Origin & Axes in 2D.
 - 1. X is usually the left-right direction
 - 2. Y is usually the up-down direction
 - 3. Unlike in 2D graphics, we won't *have to* have the inverted y-axis we have in pscreen and pygame.
 - c. Z is usually the in-out (of the screen) direction.
 - i. If we're talking about the +z direction, does it go in our out?
 - 1. It could be either! (OpenGL has +z coming out of the screen, DirectX has +z going into the screen)
 - 2. Right-handed coordinate systems vs. Left-handed
 - a. Put your 1st (thumb) in the direction of +x.
 - b. Put your 2nd (pointer) finger in the direction of +y.
 - c. Your 3rd (middle) finger will now point in the +z direction.
 - i. In LHS, it will point away from you (assuming thumb is going to the right, and pointer going up).
 - ii. In RHS, it will point towards you
 - d. You express a point in 3-space by a set of 3 numbers (x, y, z).
 - i. Let's say we have a cube at (0,0,0) and a sphere at (5,3,0).
 - ii. How will this look when rendered?
 - iii. It depends!
 - 1. Unlike 2D, there is no "standard" origin.
 - 2. Everything in 3D is relative.
 - 3. The missing piece in this case is a **camera**.
 - 4. If the camera is at (0,0,10) looking in the -z direction (Right-handed), the results will look very different than if the camera is at (0,10,0), looking in the -y direction.
- 3. Blender
 - a. In this class, we'll take a close look at the features of one 3D modeling program
 - i. There are many other modelers out there:
 - 1. Maya
 - 2. 3D studio Max
 - 3. Milkshape
 - 4. Lightwave
 - 5. Bryce
 - 6. many others
 - ii. Each has its own strengths and weaknesses.
 - iii. Blender is very powerful – its interface takes some getting used to; but once you do, it can stand with other commercial applications. And its free!
 - 1. And Blender is built with Python

2. So you can write your own plug-ins for Blender, or change the way the program works!
 - b. We'll use the 3D modeler to make 3D objects/scenes that can then be rendered
 - i. Using our own software renderer
 - ii. Using OpenGL/DirectX
 - iii. Using a GameEngine (Unreal, Ogre, Havok, etc)
 - iv. Using a commercial game (as in a mod)
 - c. If you want a copy of blender, go to
 - i. <http://www.blender.org>
4. VPython
- a. A *very* simple python module for displaying 3D graphics.
 - b. We'll use this to visualize things until we start seeing things in our raytracer.
 - c. Get it at <http://vpython.org>