CMPM 163 Notes

Malcolm Riley

Winter 2019 — April 4, 2019

Class Information

- Class Website:
- Class Slack Channel:
- Slack channel is primary avenue of communication for the class.

3D Scene Overview

- In most graphics frameworks, scenes consist of the following components:
- The **Camera** is the object that characterizes the extent of the 3D space that will be projected onto the 2D image plane (of the screen). It defines “how much of the 3D scene is viewed”
- The **Lights** are the components used for simulating illumination. They typically incorporate a position, orientation, color and will typically have different types.
- **Geometry** refers to the objects, composed of triangles, that populate the scene and are lit by the different **lights**
- **Materials** are the characterizing properties of the objects in the scene that determine how individual pieces of **geometry** respond to **light**.
- **Textures**

Perspective Camera

- Cameras are defined by a number of variables, typically a matrix.
- Only objects within the **View Frustum** of the camera are rendered.

Rendering Overview

- The **Rendering Pipeline** describes the series of operations that transform programmatically defined 3D data into a 2D image for display on a screen.
- Historically, the rendering pipeline was **fixed-function**, meaning that developers didn't have much control over how the hardware operated. Modern rendering pipelines are **programmable** via programs known as **shaders**.
- The **Vertex Shader** is the responsible for turning 3D geometry into "normalized device coordinates", 2D values between -1 and $+1$, plus a depth value.
- The **Fragment** or **Pixel Shader** is responsible for coloring in individual pixels (more accurately, "fragments", and then drawing them to the screen.
- (Pipeline Overview Here)
- In Unity, each shader program occupies a single function. At the start of an application, these files are compiled into a "program" and dispatched to the GPU along with texture data for use. When the application is running (usually 60FPS), during each frame, the following occur:
  1. The shader program is **bound** (activated) on the GPU
  2. Texture data that will be used for the program will also be **bound**.
  3. 3D points describing the geometry are dispatched to the GPU and input to the vertex shader, one triangle at a time.
  4. The vertex shader projects the triangles into a simpler coordinate system and outputs it as **fragments** (basically, pixel data), which is then fed to the **fragment shader**.
  5. The fragment shader decides what color to make each pixel, and then draws it on the screen.
- The GPU capitalizes on massive parallelization. For instance, when the fragment shader is called, it operates agnostic of any other fragment being drawn.

Rendering in Unity

- The most basic type of vertex shader in Unity is called a **Passthrough Shader**. This vertex shader merely projects the 3D geometry onto the 2D plane, and then passes the 2D data to the fragment shader.
- The most basic fragment shader is a color shader, which simply colors the fragment a solid color irrespective of input data.
- Shaders are applied to geometry via the materials system. Objects may have a Material attached, and an individual Material may have one `.shader` file attached to it.
- (Shader Examples Here)
- `UnityObjectToClipPos()` - Project 3D to 2D
- `tex2D()` - UV color lookup for 2D texture