CSE 190 Discussion 3

PA2: Level of Immersion

Agenda

- Project 2 Overview
- Initial parts of PA2:
  - Rendering a 3D skybox
  - Varying cube sizes
  - Viewing Modes
  - Tracking Modes

Project 2 Overview

- Project 2 Due: May 3rd 2pm
  - If you have scheduling conflicts, let us know
- Given features in Project 2 Starter Code
  - 2 textured cubes
  - Monoscopic skybox
- Features you need to implement:
  - Render a skybox in stereo
  - Change the rendering settings
  - More specifications in the assignment page.

Getting Started

- Added files:
  - TexturedCube.h/cpp:
    - Loads in the ppm files and textures a cube object
  - SkyBox.h/cpp:
    - Derived class of textured cube
  - skybox.vert/frag
    - Shader to render the skybox and textured cubes
- Added/changed class:
  - Scene Class (was Color Cube Scene)

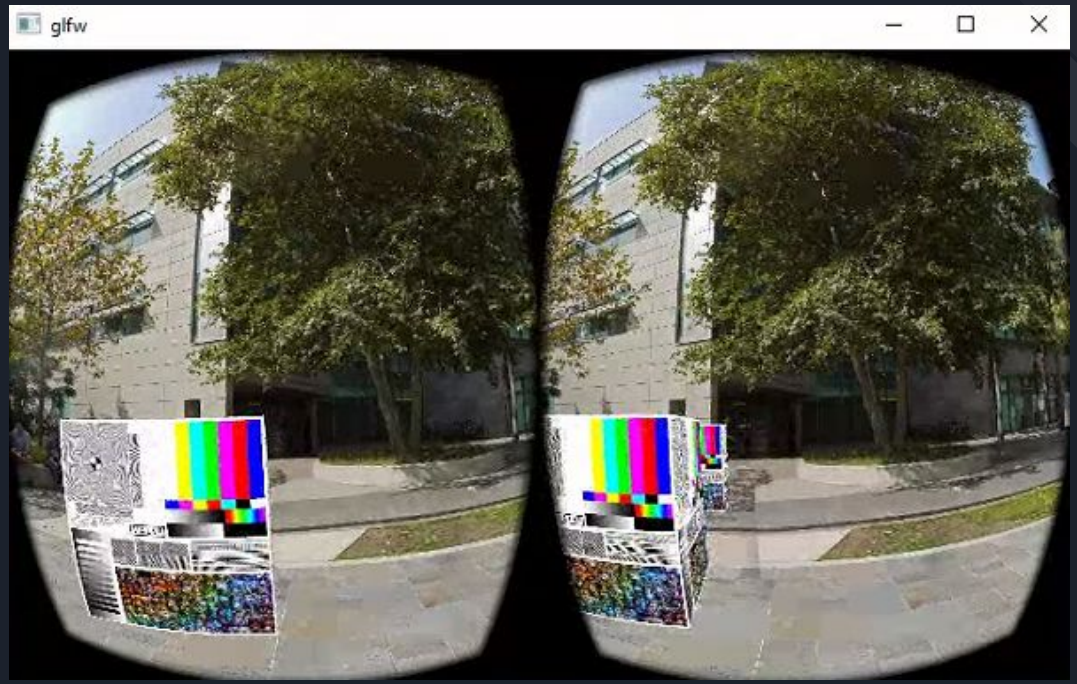
Getting Started

Option 1:

- Clone again and re-factor

Option 2:

- Copy over the additional files (h/cpp and cube/skybox folders)
- Make new Scene class and copy over the scene class in the main.cpp
- Make new h/cpp file for your Proj2 (a new ExampleApp file)


glfw

Render Skybox in 3D

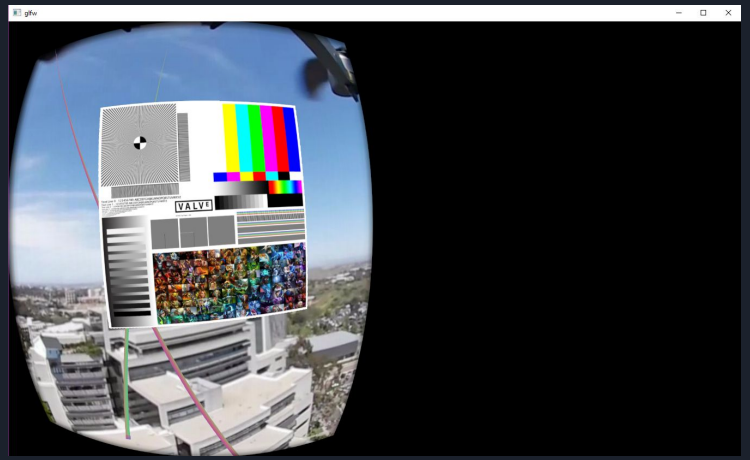
- Left eye skybox in repository
- Right eye skybox on project page
- You will need to differentiate which eye you are rendering to
  - Check the eye type
    - Based on that eye type (ovrEye_left vs ovrEye_right) render the appropriate skybox
  - Starter code renders as a 10m cube surrounding the user, so the physical placement may need to have a horizontal offset.

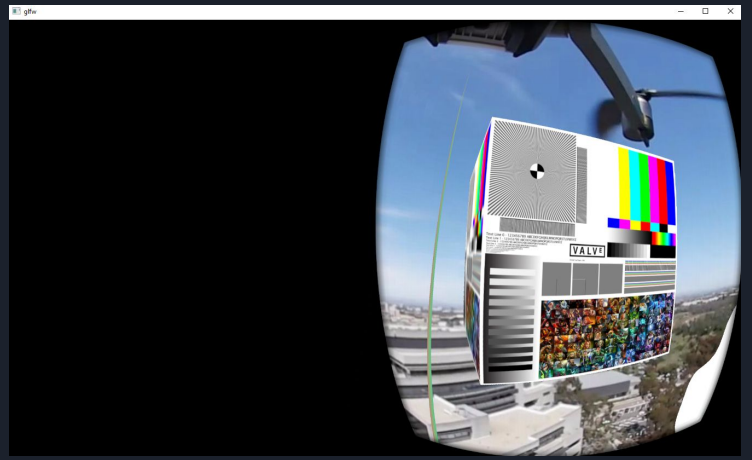
Viewing Modes

- Want to see the effects of monoscopic vs stereo vision so need to implement different viewing modes:
  - Stereo Vision
  - Mono
  - Left/Right eye only
  - Inverted Stereo
- Look through the render/draw functions to find where we are drawing each eye

Left eye Only and Right eye Only

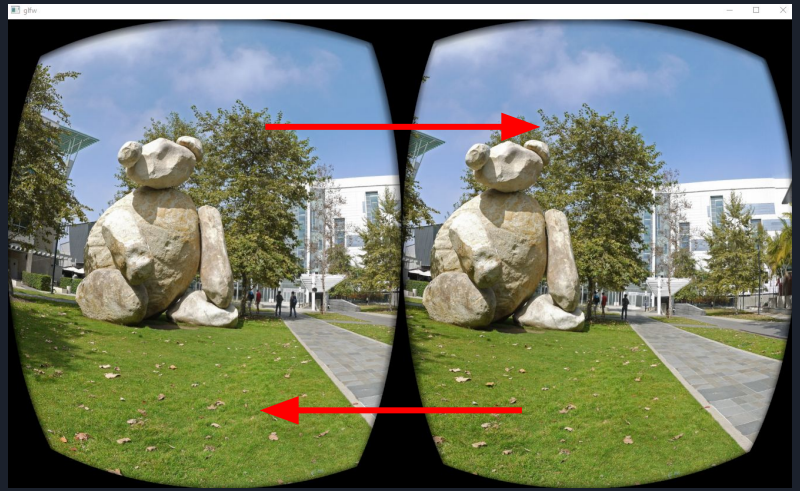
- Only render if it is the correct eye, otherwise just skip the rendering step





Inverted Stereo

- Simply swap which eye pose each eye uses to render
  - Still check ovrEye_left and ovrEye_right



Button Interaction

```
void update() final override {
      ovrInputState inputState;
      if (OVR_SUCCESS(ovr_GetInputState(_session, ovrControllerType_Touch, &inputState))){

            if (inputState.HandTrigger[ovrHand_Right] > 0.5f)
                  std::cerr << "right middle trigger pressed" << std::endl;
            if (inputState.IndexTrigger[ovrHand_Right] > 0.5f)
                  std::cerr << "right index trigger pressed" << std::endl;
            if (inputState.HandTrigger[ovrHand_Left] > 0.5f)
                  std::cerr << "left middle trigger pressed" << std::endl;
            if (inputState.IndexTrigger[ovrHand_Left] > 0.5f)
                  std::cerr << "left index trigger pressed" << std::endl;
            if (inputState.Buttons>0)
                  std::cerr << "Botton state:" << inputState.Buttons << std::endl;
      }
}
```

Vary Physical Size of Cubes

- Starter code has two 30cm cubes
- Need to be able to:
  - Change cube dimensions from 0.01m to 0.5m without changing the position of their centers
  - Reset the size to 30cm

Tracking Modes

- What to play with the position/orientation of the HMD to see what the effects are when loose either/both kinds of tracking
- Implement tracking modes
  - Regular
  - Orientation only
  - Position only
  - No tracking

QUESTIONS?