**Project 10: Advanced Shaders 2**

Benjamin Carter and Josh Canode

Grand Canyon University

CST-305: Computer Graphics

Professor Ricardo Citro

December 12th, 2023

**Video:**

<https://codingcando.com/fileShare/file?code=GKIUQ9U3ME>

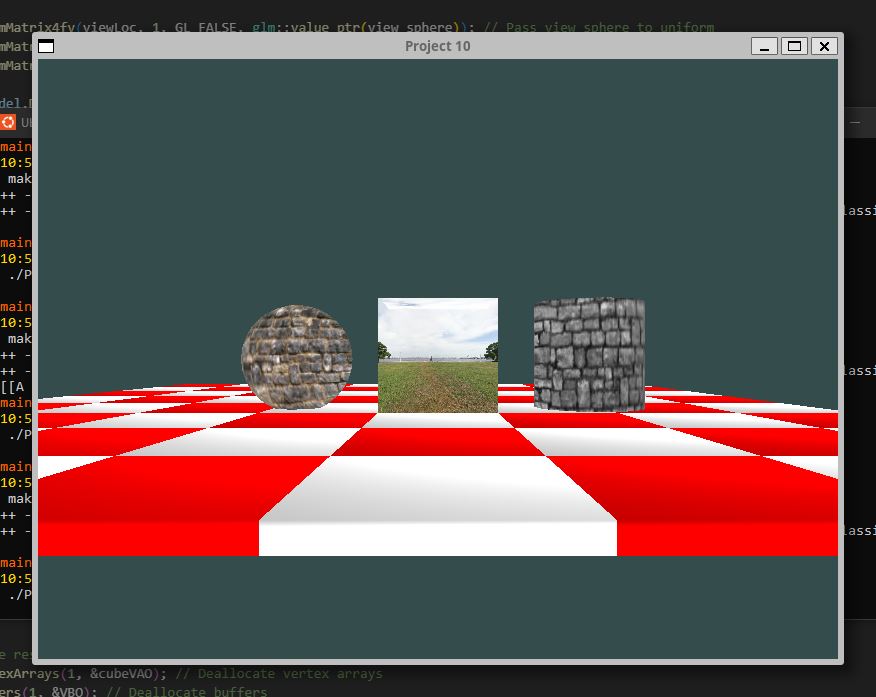
**Project Description and Main Idea**

This project consists of taking the basics from Project 9 and learning to apply advanced shaders to them. Project 9 contained a checkered board with different shapes resting on the checkered board. Then, the camera is adjustable using a set of keyboard controls. These keyboard controls allow for a user to navigate around the scene. Now in Project 10 we take these basic shapes and apply textures to the cylinder and sphere and implement a skybox on the cube.

The resulting image:

Project 9 Project 10

A screenshot of a video game

Description automatically generated****

**Theoretical background**

In computer graphics, shaders are programs that are run on a GPU, which is a processing unit on a computer that specializes in parallel arithmetic operations. A shader instructs a GPU to compute the colors for the pixel values that it receives. In this program, shaders are used to convey the positioning and coloring of each object, including the checkered board. This project adds onto project nine by including texture rendering. First, the vertex shader calculates the appropriate texture coordinates, and then the fragment shader calculates the actual color of the pixel.

**Mathematical concepts**

We are using Vertex and Fragment shaders. Vertex shaders are responsible for transforming the 3D coordinates of objects from their local space to clip space, applying operations like translation and rotation. Fragment shaders determine the color for each pixel after vertex processing. We use a combination of both to manage lighting and fine tune the rendering.

For 3D Vertex Shaders, they take in three different matrices. The Model Matrix, the View Matrix, and the Projection Matrix. The Model Matrix converts the local space vectors to world space vectors. The View Matrix represents the camera transformations. The Projection Matrix represents the camera space to flat screen space.

* checkerboard.vs/.frag
* image.vs/.frag -- for both the sphere and the cylinder
* skybox.vs/.frag

**Controls**

|  |  |
| --- | --- |
| Left Arrow Key | Move left respective to camera |
| Right Arrow Key | Move right respective to camera |
| Up Arrow Key | Move up respective to camera |
| Down Arrow Key | Move down respective to camera |
| Shift + Up Arrow Key | Move forward respective to camera |
| Shift + Down Arrow Key | Move backward respective to camera |
| Ctrl + Up Arrow Key | Tilt camera down, pitch down |
| Ctrl + Down Arrow Key | Tilt camera up, pitch up |
| Ctrl + Left Arrow Key | Rotate camera counterclockwise |
| Ctrl + Right Arrow Key | Rotate camera clockwise |

**Programming implementation**Top of Form

**Flowchart:**

A diagram of a program

Description automatically generated

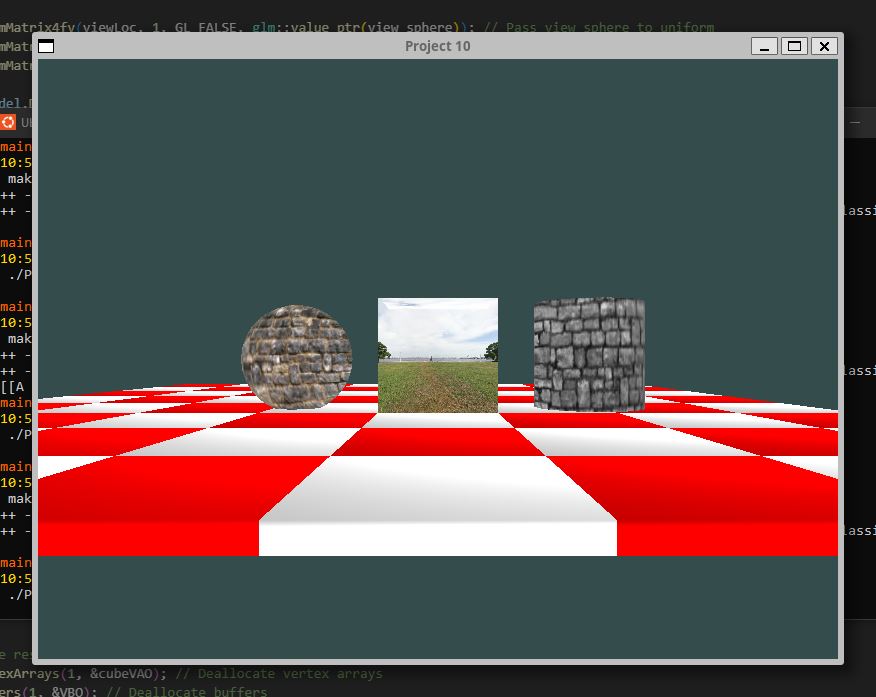
**Desired Effect:**

A screenshot of a computer game

Description automatically generated

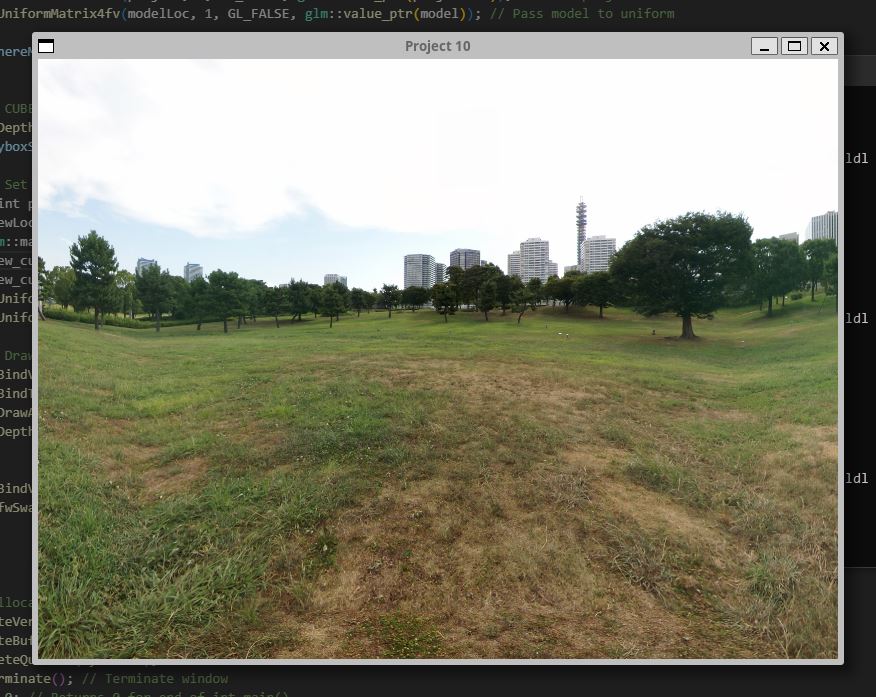
The goal of the image (but with changing the checkerboard color).

**Screenshots**

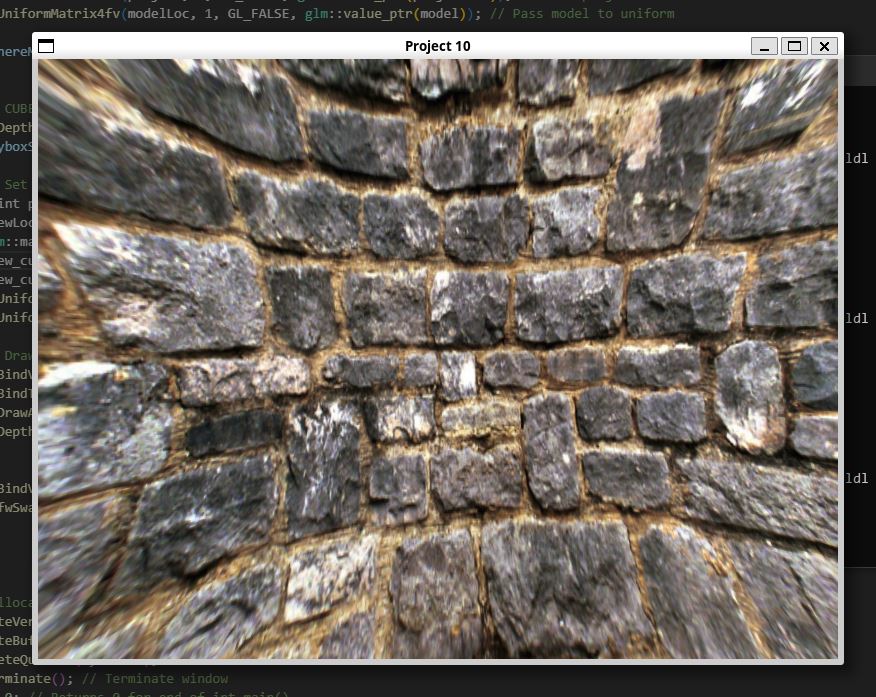
****

The resulting image with the three objects.

**Inside the skybox**

****

What the skybox looks like when you enter it.

**Inside the Sphere **

The sphere’s texture. The warping is observed as the texture is stretched across a sphere (a curved surface).

**Inside the Cylinder** 

The cylinder’s texture. The warping is observed as the texture is stretched across the cylinder (a curved surface).