

# AdamWilliams\_GGP\_CW2

*by Adam Williams*

---

**Submission date:** 16-Jan-2023 10:29PM (UTC+0000)

**Submission ID:** 195275592

**File name:** 25476\_Adam\_Williams\_AdamWilliams\_GGP\_CW2\_473926\_1131033514.docx (312.68K)

**Word count:** 1990

**Character count:** 12465

## Title Page

Module Name:

Graphics and Games Programming (ACCA6021)

Assignment Title:

3D OGL Games Programming

Student Name & Number:

Adam Williams (2008159)

## Table of Contents

### Contents

|                                         |    |
|-----------------------------------------|----|
| Title Page.....                         | 1  |
| Table of Contents .....                 | 2  |
| User Manual .....                       | 3  |
| Controls.....                           | 3  |
| Report .....                            | 4  |
| Including "OpenGL" Directories .....    | 4  |
| Camera Position & Controls .....        | 4  |
| Setting up Textures in the Array .....  | 6  |
| Setting up the Cards.....               | 7  |
| Drawing the Cards onto the Screen ..... | 8  |
| Video Demonstration of Code .....       | 9  |
| References .....                        | 10 |
| Appendix .....                          | 11 |
| Souce.cpp .....                         | 11 |
| Utils.h .....                           | 25 |

## User Manual

### Controls

"W" – Camera up

"S" – Camera down

"A" – Tilt camera left

"D" – Tilt camera right

## Report

### Including "OpenGL" Directories

```
1  #include "GL/glew.h"
2  #include "GLFW/glfw3.h"
3  #include "glm/glm.hpp"
4  #include "../glm/ext.hpp" // perspective, translate, rotate
5  #include <iostream>
6  #include "utils.h"
7  #include "stb_image_loader.h"
8  using namespace std;
9  float cameraX, cameraY, cameraZ;
10 float cubeLocX, cubeLocY, cubeLocZ;
11 int direction = 0;
12 GLuint tex[4];
13
14 #define numVAOs 2
15 #define numVBos 4
16 // allocate variables used in display() function, so that they won't need to be allocated during rendering
17 GLuint mvLoc, projLoc;
18 int width, height;
19 float aspect;
20 glm::mat4 pMat, vMat, mMat, mvMat;
21 void setupVertices(void);
22 GLuint renderingProgram;
23 GLuint vao[numVAOs];
24 GLuint vbo[numVBos];
25
```

Here I start off the code by making sure to include all the necessary "OpenGL" Directories I will be using through out the code. I also define variables related to the camera coordinates, and an texture array. I also define my VAOs, and VBos.

### Camera Position & Controls

```
void moveCamera(float Px, float Pz, float Vtorx, float Vtorz)
{
    cout << "Px" << Px << " Pz" << Pz << " Vtorx=" << Vtorx << " Vtorz=" << Vtorz << endl;
    vMat = glm::lookAt(glm::vec3(Px, 2, Pz), glm::vec3(Px + Vtorx, -2.0, Pz + Vtorz), glm::vec3(0.0, 1.0, 0.0));
}
```

This sets up the variables related to the camera controls such as px, and pz which are used for the camera position along the axis.

```

32 static void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
33 {
34     static int last = 0;
35     static float Px = 1.5, Py = 2, Pz = 3.2, Vtorx = 0, Vtorz = -1.0, angle = 0;
36     switch (key) {
37     case GLFW_KEY_A:
38         //look left
39         angle = angle - 1;
40         Vtorx = sin(angle * 0.01745); //
41         Vtorz = -cos(angle * 0.01745); //assume camera default is pointing down negative z axis.
42         break;
43     case GLFW_KEY_D:
44         //look right
45         angle = angle + 1;
46         Vtorx = sin(angle * 0.01745); //
47         Vtorz = -cos(angle * 0.01745); //assume camera default is pointing down negative z axis.
48         break;
49     case GLFW_KEY_W:
50         Px = Px + 0.1 * Vtorx;
51         Pz = Pz + 0.1 * Vtorz;
52         break;
53     case GLFW_KEY_S:
54         Px = Px - 0.1 * Vtorx;
55         Pz = Pz - 0.1 * Vtorz;
56         break;
57     case GLFW_KEY_Z:
58         //strafe left
59         cout << "A pressed" << last << endl;
60         last++;
61         direction++;
62         break;
63     case GLFW_KEY_X:
64         //strafe right
65         cout << "D pressed" << last << endl;
66         direction--;
67         break;
68     case 27:
69         exit(0);
70         break;
71     default:
72         break;
73     }
74     moveCamera(Px, Pz, Vtorx, Vtorz); //best placed where you'd normally use glm::lookat
75 }

```

This code updates the modifies and updates the camera position when one of the inputs is used. The static float at the top is the initial position of the camera. The first two "cases" are responsible for the camera angle rotation. When pressing "A" the camera rotates left, and when pressing "D" the camera rotates right. The third and forth case move the camera along the axis. When pressing "W" the camera moves forwards, when pressing "S" the camera moves backwards.

### Setting up Textures in the Array

```
80 void init(GLFWwindow* window) {
81     int x, y, n;
82     int force_channels = 4; //RGB and A
83     char filename[] = "checkers.png";
84     char filename1[] = "Stripes1.png";
85     char filename2[] = "Stripes2.png";
86     unsigned char* image_data = stbi_load(filename, &x, &y, &n, force_channels); //x = x dimensions, y= ydims,
87
88
89     if (!image_data)
90     {
91         cout << "error: could not load " << filename << endl;
92     }
93
94     // copy image into ogl buffer - stick to images power of 2 high and wide
95     //binds texture to a texture in the array
96     glGenTextures(3, tex);
97     glActiveTexture(GL_TEXTURE0);
98     glBindTexture(GL_TEXTURE_2D, tex[0]);
99     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, x, y, 0, GL_RGBA, GL_UNSIGNED_BYTE, image_data);
100     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
101     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
102     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
103     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
104 }
```

This code here sets up the textures and defines the file names. The files used for the textures are then called upon and bound to a slot of the "tex" array so they can be called upon. This process is repeated 3 times for the 3 textures used.

### Setting up the Cards

```
146 void display(GLFWwindow* window, double currentTime)
147 //runs 60 times a second by default in glfw - see main loop
148 {
149     glClear(GL_DEPTH_BUFFER_BIT);
150     glClearColor(0.8, 0.8, 0.9, 1.0);
151     glClear(GL_COLOR_BUFFER_BIT);
152     glUseProgram(renderingProgram); //specify linked program to use
153
154     //get a reference to the shader variable Vcol
155     //GLuint VcolLoc = glGetUniformLocation(renderingProgram, "Vcol");
156     mvLoc = glGetUniformLocation(renderingProgram, "mv_matrix");
157     projLoc = glGetUniformLocation(renderingProgram, "proj_matrix");
158
159     // copy perspective and MV matrices to corresponding uniform variables
160     glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvMat));
161     glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
162     // adjust OpenGL settings
163     glBindVertexArray(vao[0]);
164     glBindBuffer(GL_ARRAY_BUFFER, vbo[0]); //card pair mesh bound to VBO
165     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
166     glEnableVertexAttribArray(0);
167     glBindBuffer(GL_ARRAY_BUFFER, vbo[1]); //card pair mesh bound to VBO
168     glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 0, 0);
169     glEnableVertexAttribArray(1);
170     glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
```

Firstly, we setup the view matrix and the necessary render settings.

```
171 int xOffset = 0, yOffset = 0;
172 int numCards = 2;
173 int numPairs = 4;
174 //draw 8 pairs in 4 rows of 2
```

We then define the variables that are going to be used when setting up the cards. "xOffset" and "yOffset" are responsible for the position offset between all the cards giving space between them. Its value will be changed later on. "numCards" is the variable responsible for how many cards will be in a pair, so it is set at 2. "numPairs" is the variable responsible for how many pairs will be rendered onto the screen.

### Drawing the Cards onto the Screen

```

184 for (int i = 0; i < numPairs; i++)
185 {
186     for (int j = 0; j < numCards; j++)
187     {
188         xOffset = j * 2;
189         yOffset = i * 2;
190
191         if (j == 0) //every 2 cards a new texture is applied
192         {
193             glBindTexture(GL_TEXTURE_2D, tex[0]);
194         }
195         else if (j == 1)
196         {
197             glBindTexture(GL_TEXTURE_2D, tex[1]);
198         }
199         else
200         {
201             glBindTexture(GL_TEXTURE_2D, tex[2]);
202         }
203
204         //Create new model matrix and then mv matrix (no change to camera position/orientation so view matrix not touched)
205         mMat = glm::translate(glm::mat4(1.0f), glm::vec3(cubeLocX + xOffset, cubeLocY, cubeLocZ + yOffset));
206         mMat = glm::rotate(mMat, glm::radians(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
207         mvMat = vMat * mMat;
208         //copy new mv matrix to shader
209         glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvMat));
210         glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
211         k = k + 3; //next colour
212         glDrawArrays(GL_TRIANGLES, 0, 48); //draw a card
213     }
214 }
215
216

```

This main loop is responsible for creating the correct number of cards, in the correct positions. Using the variables "numCards" and "numPairs" the loops will go around the correct number of times making 2 cards a pair for 4 pairs. This code also changes the texture used by changes the texture within the "tex" array being used each time variable "j" is increased.

```

238 void setupVertices(void) {
239
240     float cardpair1VertexPos[144] = {
241
242
243         0.0, -2.0, 0.0, //bottom
244         0.0, -2.0, -1.5,
245         1.5, -2.0, -1.5,
246         0.0, -2.0, 0.0,
247         1.5, -2.0, -1.5,
248         1.5, -2.0, 0.0,
249
250     };
251
252
253

```

Here the coordinates for drawing the size of the card are defined here.

```

250     };
251     float cardpair1Tex[96] = {
252
253         //front
254
255         //bottom
256         1.0, 0.0,
257         1.0, 1.0,
258         0.0, 1.0,
259         1.0, 0.0,
260         0.0, 1.0,
261         0.0, 0.0,
262

```

The coordinates for drawing the textures that go with the card is defined here.

Video Demonstration of Code

[https://youtu.be/AktsT8\\_Z0r0](https://youtu.be/AktsT8_Z0r0)

## References

1. OpenGL - The Industry's Foundation for High Performance Graphics. OpenGL - The Industry's Foundation for High Performance Graphics [Internet]. The Khronos Group. 2011 [cited 2023 Jan 16]. Available from: <https://www.khronos.org/opengl/>

## Appendix

Source.cpp

```
#include "GL/glew.h"

#include "GLFW/glfw3.h"

#include "glm/glm.hpp"

#include "./glm/ext.hpp" // perspective, translate, rotate

#include <iostream>

#include "utils.h"

#include "stb_image_loader.h"

using namespace std;

float cameraX, cameraY, cameraZ;

float cubeLocX, cubeLocY, cubeLocZ;

int direction = 0;

GLuint tex[4];

#define numVAOs 2

#define numVBOs 4

// allocate variables used in display() function, so that they won't need to be allocated during
rendering

GLuint mvLoc, projLoc;

int width, height;

float aspect;

glm::mat4 pMat, vMat, mMat, mvMat;

void setupVertices(void);

GLuint renderingProgram;

GLuint vao[numVAOs];

GLuint vbo[numVBOs];

void moveCamera(float Px, float Pz, float Vtorx, float Vtorz)

{

    cout << "Px" << Px << " Pz=" << Pz << " Vtorx=" << Vtorx << " Vtorz=" << Vtorz << endl;
```

```

        vMat = glm::lookAt(glm::vec3(Px, 2, Pz), glm::vec3(Px + Vtorx, -2.0, Pz + Vtorz),
glm::vec3(0.0, 1.0, 0.0));
    }

static void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    static int last = 0;

    static float Px = 1.5, Py = 2, Pz = 3.2, Vtorx = 0, Vtorz = -1.0, angle = 0;

    switch (key) {
    case GLFW_KEY_A:
        //look left
        angle = angle - 1;
        Vtorx = sin(angle * 0.01745); //
        Vtorz = -cos(angle * 0.01745); //assume camera default is pointing down negative z
axis.

        break;
    case GLFW_KEY_D:
        //look right
        angle = angle + 1;
        Vtorx = sin(angle * 0.01745); //
        Vtorz = -cos(angle * 0.01745); //assume camera default is pointing down negative z
axis.

        break;
    case GLFW_KEY_W:
        Px = Px + 0.1 * Vtorx;
        Pz = Pz + 0.1 * Vtorz;
        break;
    case GLFW_KEY_S:
        Px = Px - 0.1 * Vtorx;
        Pz = Pz - 0.1 * Vtorz;
        break;
    case GLFW_KEY_Z:

```

```

        //strafe left
        cout << "A pressed" << last << endl;
        last++;
        direction++;
        break;
    case GLFW_KEY_X:
        //strafe right
        cout << "D pressed" << last << endl;
        direction--;
        break;
    case 27:
        exit(0);
        break;
    default:
        break;
}
moveCamera(Px, Pz, Vtorx, Vtorz); //best placed where you'd normally use glm::lookat
}

```

```

void init(GLFWwindow* window) {
    int x, y, n;
    int force_channels = 4; //RGB and A
    char filename[] = "checkers.png";
    char filename1[] = "Stripes1.png";
    char filename2[] = "Stripes2.png";

    unsigned char* image_data = stbi_load(filename, &x, &y, &n, force_channels); //x = x
    dimemsions, y= ydims,

```

```

        if (!image_data)
        {
            cout << "error: could not load " << filename << endl;

        }

        // copy image into ogl buffer - stick to images power of 2 high and wide
        //binds texture to a texture in the array
        glGenTextures(3, tex);
        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, tex[0]);

        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, x, y, 0, GL_RGBA, GL_UNSIGNED_BYTE,
image_data);

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

        image_data = stbi_load(filename1, &x, &y, &n, force_channels); //x = x dimemsions, y=
ydims,
        if (!image_data)
        {
            cout << "error: could not load " << filename1 << endl;

        }

        // copy image into ogl buffer - stick to images power of 2 high and wide
        glBindTexture(GL_TEXTURE_2D, tex[1]);

        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, x, y, 0, GL_RGBA, GL_UNSIGNED_BYTE,
image_data);

        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

```

```

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

image_data = stbi_load(filename2, &x, &y, &n, force_channels); //x = x dimensions, y=
ydims,
if (!image_data)
{
    cout << "error: could not load " << filename2 << endl;

}

// copy image into ogl buffer - stick to images power of 2 high and wide
glBindTexture(GL_TEXTURE_2D, tex[2]);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, x, y, 0, GL_RGBA, GL_UNSIGNED_BYTE,
image_data);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

renderingProgram = createShaderProgram(); //load, compile, create, attach and link shader
programmes done here

cameraX = 0.0f; cameraY = 0.0f; cameraZ = 18.0f; //camera position
cubeLocX = 0.0f; cubeLocY = -2.0f; cubeLocZ = 0.0f; // shift cube down Y to reveal
perspective

vMat = glm::lookAt(glm::vec3(cameraX, cameraY, cameraZ), glm::vec3(cubeLocX, cubeLocY,
cubeLocZ), glm::vec3(0.0f, 1.01f, 0.0f));

// build perspective matrix
glfwGetFramebufferSize(window, &width, &height); //need these to fit clip space to window
aspect = (float)width / (float)height;
pMat = glm::perspective(1.0472f, aspect, 0.05f, 100.0f); // 1.0472 radians = 60 degrees
setupVertices(); //initial vertices x,y,z values
glEnable(GL_DEPTH_TEST);

```

```

}

void display(GLFWwindow* window, double currentTime)
//runs 60 times a second by default in glfw - see main loop
{
    glClear(GL_DEPTH_BUFFER_BIT);
    glClearColor(0.8, 0.8, 0.9, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glUseProgram(renderingProgram); //specify linked program to use

    //get a reference to the shader variable Vcol
    //GLuint VcolLoc = glGetUniformLocation(renderingProgram, "Vcol");
    mvLoc = glGetUniformLocation(renderingProgram, "mv_matrix");
    projLoc = glGetUniformLocation(renderingProgram, "proj_matrix");

    // copy perspective and MV matrices to corresponding uniform variables
    glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvMat));
    glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
    // adjust OpenGL settings
    glBindVertexArray(vao[0]);
    glBindBuffer(GL_ARRAY_BUFFER, vbo[0]); //card pair mesh bound to VBO
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, vbo[1]); //card pair mesh bound to VBO
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(1);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    int xOffset = 0, yOffset = 0;
    int numCards = 2;
    int numPairs = 4;
    //draw 8 pairs in 4 rows of 2
    float cardCols[18] = {

```

```

1.0,0.0,0.0,
1.0,1.0,0.0,
174.0 / 255.0,85.0 / 255.0,186.0 / 255.0,
0.0,1.0,0.0,
0.0,0.0,1.0,
233.0 / 255.0,66.0 / 255.0,17.0 / 255.0
};
int k = 0;
for (int i = 0; i < numPairs; i++)
{
    for (int j = 0; j < numCards; j++)
    {
        xOffset = j * 2;
        yOffset = i * 2;

        if (j == 0) //every 2 cards a new texture is applied
        {
            glBindTexture(GL_TEXTURE_2D, tex[0]);
        }
        else if (j == 1)
        {
            glBindTexture(GL_TEXTURE_2D, tex[1]);
        }
        else
        {
            glBindTexture(GL_TEXTURE_2D, tex[2]);
        }

        //Create new model matrix and then mv matrix (no change to camera
        position/orientation so view matrix not touched)

        mMat = glm::translate(glm::mat4(1.0f), glm::vec3(cubeLocX + xOffset,
        cubeLocY, cubeLocZ + yOffset));

        mMat = glm::rotate(mMat, glm::radians(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));

```

```

        mvMat = vMat * mMat;

        //copy new mv matrix to shader
        glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvMat));
        glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
        k = k + 3; //next colour
        glDrawArrays(GL_TRIANGLES, 0, 48); //draw a card

    }

}

int main(void) {
    if (!glfwInit()) { exit(EXIT_FAILURE); }

    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    GLFWwindow* window = glfwCreateWindow(600, 600, "glfw window", NULL, NULL);
    glfwSetKeyCallback(window, key_callback);
    glfwMakeContextCurrent(window);
    if (glewInit() != GLEW_OK) { exit(EXIT_FAILURE); }
    glfwSwapInterval(1);
    init(window);
    while (!glfwWindowShouldClose(window)) {
        display(window, glfwGetTime());
        glfwSwapBuffers(window);
        glfwPollEvents();
    }
    glfwDestroyWindow(window);
    glfwTerminate();
    exit(EXIT_SUCCESS);
}

```

```

void setupVertices(void) {

    float cardpair1VertexPos[144] = {

        0.0, -2.0, 0.0, //bottom
        0.0, -2.0, -1.5,
        1.5, -2.0, -1.5,
        0.0, -2.0, 0.0,
        1.5, -2.0, -1.5,
        1.5, -2.0, 0.0,

    };

    float cardpair1Tex[96] = {

        //front
        1.0, 0.0,
        1.0, 1.0,
        0.0, 1.0,
        1.0, 0.0,
        0.0, 1.0,
        0.0, 0.0,
        //right
        1.0, 0.0,
        1.0, 1.0,
        0.0, 1.0,
        1.0, 0.0,
        0.0, 1.0,
        0.0, 0.0,
        //left

```

1.0, 0.0,  
1.0, 1.0,  
0.0, 1.0,  
1.0, 0.0,  
0.0, 1.0,  
0.0, 0.0,  
//bottom  
1.0, 0.0,  
1.0, 1.0,  
0.0, 1.0,  
1.0, 0.0,  
0.0, 1.0,  
0.0, 0.0,  
//back  
1.0, 0.0,  
1.0, 1.0,  
0.0, 1.0,  
1.0, 0.0,  
0.0, 1.0,  
0.0, 0.0,  
//front gable  
0.0, 1.0,  
0.0, 0.0,  
1.0, 0.0,  
//back gable  
0.0, 1.0,  
0.0, 0.0,  
1.0, 0.0,  
//left roof  
1.0, 0.0,  
1.0, 1.0,

```

        0.0, 1.0,
        1.0, 0.0,
        0.0, 1.0,
        0.0, 0.0,
        //right roof
        1.0, 0.0,
        1.0, 1.0,
        0.0, 1.0,
        1.0, 0.0,
        0.0, 1.0,
        0.0, 0.0
    };

    float cardpair2VertexPos[144] = {

        0.0, -2.0, 0.0, //bottom
        0.0, -2.0, -1.0,
        1.0, -2.0, -1.0,
        0.0, -2.0, 0.0,
        1.0, -2.0, -1.0,
        1.0, -2.0, 0.0,

    };

    float cardpair2Colour[144] = {
    //front
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
    }

```

//right

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

//left

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

//bottom

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

//back

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

//front gable

1.0, 1.0, 0.0,

1.0, 1.0, 0.0,

```

        1.0, 1.0, 0.0,
        //back gable
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
        //left roof
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
        //right roof
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
        1.0, 1.0, 0.0,
    };

    //initialise VAO and VBOs
    glGenVertexArrays(numVAOs, vao); //we will use one VAO
    glBindVertexArray(vao[0]);
    glGenBuffers(numVBOs, vbo); //we have one VBO
    glBindBuffer(GL_ARRAY_BUFFER, vbo[0]); //first VBO
    glBufferData(GL_ARRAY_BUFFER, sizeof(cardpair1VertexPos), cardpair1VertexPos,
GL_STATIC_DRAW);
    glBindBuffer(GL_ARRAY_BUFFER, vbo[1]); //first VBO
    glBufferData(GL_ARRAY_BUFFER, sizeof(cardpair1Tex), cardpair1Tex, GL_STATIC_DRAW);

    glBindBuffer(GL_ARRAY_BUFFER, vbo[2]); //3rd VBO

```

```
        glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat),  
                                (void*) (2 * sizeof(GLfloat)));  
        glBindBuffer(GL_ARRAY_BUFFER, vbo[3]); // 4th VBO  
        glBufferData(GL_ARRAY_BUFFER, sizeof(cardpair2Colour), cardpair2Colour,  
                     GL_STATIC_DRAW);  
  
    }
```

```

Utils.h
#pragma once
#include <string>
#include <iostream>
#include <fstream>
using namespace std;
string readShaderSource(const char* filePath);
GLuint createShaderProgram();

string readShaderSource(const char* filePath) {
    string content;
    ifstream fileStream(filePath, ios::in);
    if (!fileStream.is_open()) {
        cout << "file not found\n";
        exit(1);
    }
    string line = "";
    while (!fileStream.eof()) {
        getline(fileStream, line);
        content.append(line + "\n");
    }
    fileStream.close();
    return content;
}

GLuint createShaderProgram() {

    GLuint vShader = glCreateShader(GL_VERTEX_SHADER);
    GLuint fShader = glCreateShader(GL_FRAGMENT_SHADER);
    string vertShaderStr = readShaderSource("vertShader.glsl");
    string fragShaderStr = readShaderSource("fragShader.glsl");
    const char* vertShaderSrc = vertShaderStr.c_str();
    const char* fragShaderSrc = fragShaderStr.c_str();
    glShaderSource(vShader, 1, &vertShaderSrc, NULL);
    glShaderSource(fShader, 1, &fragShaderSrc, NULL);

    glCompileShader(vShader);
    glCompileShader(fShader);
    GLuint vfProgram = glCreateProgram();
    glAttachShader(vfProgram, vShader);
    glAttachShader(vfProgram, fShader);
    glLinkProgram(vfProgram);
    return vfProgram;
}

```



FINAL GRADE

54/100

GENERAL COMMENTS

Instructor

Late - mark limited to 40% unless EC or DD applied.

No real gameplay in evidence however you have applied textures to a surface of an object and the camera moves. Objects placed in world space ad camera view calculated and applied.

Use of additional textures would help improve this as would some further use of keyboard/mouse interaction to play the game.

Use of third party libraries. Good.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

GRADING FORM: SMU GENERIC\_GJ

ADAM WILLIAMS

0

**RESEARCH-INFORMED LITERATURE**

**KNOWLEDGE AND UNDERSTANDING**

**ANALYSIS**

---

## **PRACTICAL APPLICATION AND DEP.**

## **SKILLS FOR PROFESSIONAL PRAC.**