

Games programming and graphics Assessment 2 - Ethan Walton-Deaves

by Ethan Walton-Deaves

Submission date: 16-Jan-2023 02:15PM (UTC+0000)

Submission ID: 195207706

File name: 24855_Ethan_Walton-Deaves_Games_programming_and_graphics_Assessment_2_-_Ethan_Walton-Deaves_473926_836782185.doc (85.5K)

Word count: 2057

Character count: 13317

Code and explaining code

```
#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 3
#define numVBOs 6

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];
```

This part initialises data. This includes GLM library that are in the external file to help the code along with calculations and functions such as the lookat function which helps the camera. The other important parts from above include introducing the rendering program and the VAO and VBOs which will help build items in the world. Notice I have 3 VAOs and 6 VBOs. Later they will be used to draw the table cards and blocker to hide the cards.

```
void moveCamera(float Px, float Pz, float Vtorx, float Vtorz) //test
{
    cout << "Px" << Px << " Pz=" << Pz << " Vtorx=" << Vtorx << " Vtorz=" << Vtorz << endl;
    vMat = glm::lookAt(glm::vec3(Px, 2.0, Pz)/*here is the angle*/, glm::vec3(Px + Vtorx, 0.0, Pz + Vtorz), glm::vec3(0.0, 1.0, 0.0)); //look up and down feature
}
```

This part sets the camera and tells it where it should be and then tells it where to look such as the angle and direction.

```

static void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    static int last = 0;
    static float Px = -1.0, Py = 6, Pz = -1.5, Vtorx = 0, Vtorz = 1.0, angle = 1; // starting location
    switch (key) {
        //case GLFW_KEY_A:
        //    //look left
        //    angle = angle - 2;
        //    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 + 5;
        //    Vtorx = sin(angle * 0.01745); //
        //    Vtorz = -cos(angle * 0.01745); //assume camera default is pointing down negative z
axis.
        //    break;
        case GLFW_KEY_W:
            //move forward along line of sight
            Px = Px + 0.1 * Vtorx;
            Pz = Pz + 0.1 * Vtorz;
            break;
        case GLFW_KEY_S:
            //move backwards along line of sight
            Px = Px - 0.1 * Vtorx;
            Pz = Pz - 0.1 * Vtorz;
            //vz= vz + 0.1;
            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
}

```

This part effects the camera and its moment around the world. Currently the camera will move

upwards with the letter W on the keyboard is pressed and backwards when S is pressed. It has units of movement which is set to 1 per click. Also to show is the letters A and D are disabled. This is because the world will only need to go up and down.

```
void init(GLFWwindow* window) {

    int x, y, n;
    int force_channels = 4; //RGB and A
    char filename[] = "checker.png";
    char filename2[] = "checker2.png";
    char filename3[] = "table.png";
    char filename4[] = "blocker.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
    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(filename2, &x, &y, &n, force_channels); //x = x dimemsions, y= ydims,
    if (!image_data)
    {
        cout << "error: could not load " << filename2 << endl;

    }

    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_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

    image_data = stbi_load(filename3, &x, &y, &n, force_channels); //x = x dimemsions, y=
ydims,
```

```

        if (!image_data)
        {
            cout << "error: could not load " << filename2 << endl;

        }
        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);
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

        image_data = stbi_load(filename4, &x, &y, &n, force_channels); //x = x dimemsions, y=
ydims,
        if (!image_data)
        {
            cout << "error: could not load " << filename4 << endl;

        }
        glBindTexture(GL_TEXTURE_2D, tex[3]);
        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_NEAREST);
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

```

This finds the filename in the folder and creates a link between the program and that file. In this case it is a PNG used. This will be used as textures for my objects in the world. 1 texture for the table another for the blocker and 2 more for the cards and binds them to the texture array assigning each PNG a number in that array helps when combing them later when drawing

```

        renderingProgram = createShaderProgram(); //load, compile, create, attach and link shader
programmes done here
        cameraX = 0.0f; cameraY = -10.0f; cameraZ = 0.0f; //camera position
        cubeLocX = 0.0f; cubeLocY = -1.0f; 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]);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(1);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
}
```

Creates the windows so that objects can be placed there. `glClear` helps with the colour in the situation.

Then we start binding items to our VAO and VBO arrays.

```
int xOffset = 0, yOffset = 0;
int numlines = 3;
int numCards = 2;
// We are adding extra objects here for the next part
//cards
int k = 0;
for (int i = 0; i < numCards; i++)
{
    for (int j = 0; j < numlines; j++)
    {
        xOffset = j * 3;
        yOffset = i * 3;
        if (j % 2 == 0) //even cards
        {
            glBindTexture(GL_TEXTURE_2D, tex[0]); //change textures
        }
        else
        {
            glBindTexture(GL_TEXTURE_2D, tex[1]);
        }
    }
}
```

```

        //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 object
    }
}

```

This part adds the first of the items and then duplicates it and places it 3 units part from each other. This also allows us to change the amount of lines and items on that line. This is indicated by the numLine and the numCards. Next It also allows us to set the textures to apply them to different duplicated items. In this case its every time the loop happens the textures change.

```

        //Adding another object to scene.
        //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 , cubeLocY, cubeLocZ +
yOffset + 5));
        mMat = glm::rotate(mMat, glm::radians(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
        mvMat = vMat * mMat;
        glBindVertexArray(vao[1]);
        glBindBuffer(GL_ARRAY_BUFFER, vbo[2]);
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
        glEnableVertexAttribArray(0);
        glBindBuffer(GL_ARRAY_BUFFER, vbo[3]);
        glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 0, 0);
        glEnableVertexAttribArray(1);

        glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvMat));
        glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
        glBindTexture(GL_TEXTURE_2D, tex[2]);
        glDrawArrays(GL_TRIANGLES, 0, 18); //draw a CARD
    }
}

```

This part assigns textures to the objects and draws the objects using “draw arrays” also assigns to different VAO and VBO numbers so later different effects can effect the object and will not interfere with other objects in the world unless it is linked to this one

```

        /// //////////////////////////////////////////
        //blockers
        mMat = glm::translate(glm::mat4(1.0f), glm::vec3(cubeLocX, cubeLocY, cubeLocZ +
yOffset + 5)); //here
    }
}

```

```

mMat = glm::rotate(mMat, glm::radians(0.0f), glm::vec3(0.0f, 1.0f, 0.0f));
mvMat = vMat * mMat;
glBindVertexArray(vao[2]);
glBindBuffer(GL_ARRAY_BUFFER, vbo[4]);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, vbo[5]);
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(1);

glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvMat));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
glBindTexture(GL_TEXTURE_2D, tex[3]);
glDrawArrays(GL_TRIANGLES, 0, 18);

```

Same as above but this time for the blocker object

```

}
```

```

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);
}

```

Termination if expectations are not met

```

void setupVertices(void) {

    float cardsPos[18] = {

        -3.0, -1.0, -0.0,
        -2.0, -1.0, 0.0,
        -3.0, -1.0, -2.0,

        -3.0, -1.0, -2.0,
        -2.0, -1.0, -2.0,
        -2.0, -1.0, -0.0,
    };
}

```

```
};  
float card1Tex[12] = {
```

```
    1.0,0.0,  
    1.0,1.0,  
    0.0,1.0,  
    1.0,0.0,  
    0.0,1.0,  
    0.0,0.0,
```

```
};
```

This part includes all the vertices needed to create the objects. This one however will be the duplicated object that is mentioned above. The second set of numbers are for the textures that are assigned to the object. Each number represents a point in world space and three points make a triangle and because of the “draw arrays” it will create a triangle between the points.

```
float table[18] = { // table
```

```
    -7.0, -2.0, -2.0, //table  
    -7.0, -2.0, -12.0,  
    8.0, -2.0, -12.0,  
    -7.0, -2.0, -2.0,  
    8.0, -2.0, -12.0,  
    8.0, -2.0, -2.0,
```

```
};
```

```
float card2Tex[36] = {
```

```
    0.0,1.0,  
    0.0,0.0,  
    1.0,0.0,
```

```
    0.0,1.0,  
    0.0,0.0,  
    1.0,0.0,
```

```
    1.0,0.0,  
    1.0,1.0,  
    0.0,1.0,  
    1.0,0.0,  
    0.0,1.0,  
    0.0,0.0,
```

```
    1.0,0.0,  
    1.0,1.0,
```

```

0.0,1.0,
1.0,0.0,
0.0,1.0,
0.0,0.0

```

This object is the table you will see the cards on and the textures to go with it

```

};
float block1Pos[18] = {
10.0, 0.0, 5.0,
10.0, 0.0, -5.0,
-5.0, 0.0, -5.0,

10.0, 0.0, 5.0,
-5.0, 0.0, 5.0,
-5.0, 0.0, -5.0,

```

```

};

```

```

float blockTex1[12] = {
1.0,0.0,
1.0,1.0,
0.0,1.0,
1.0,0.0,
0.0,1.0,
0.0,0.0,

```

```

};

```

This object is the blocker and is NOT in the position that is needed to actually block the cards In this situation this is because I would of like to make it move after a certain amount of time . Here is the code that will accomplish that goal in blue:

```

float block1Pos[18] = {
15.0, 0.0, 10.0,
15.0, 0.0, -10.0,
-6.0, 0.0, -10.0,

15.0, 0.0, 10.0,
-6.0, 0.0, 10.0,
-6.0, 0.0, -10.0,

};

```

end blue code here.

```

//initialse VAO and VBOs
glGenVertexArrays(numVAOs, vao); // VAO 1
glBindVertexArray(vao[0]);
glGenBuffers(numVBOs, vbo);//
glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);//first VBO
glBufferData(GL_ARRAY_BUFFER, sizeof(cardsPos), cardsPos, GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);//2nd VBO
glBufferData(GL_ARRAY_BUFFER, sizeof(card1Tex), card1Tex, GL_STATIC_DRAW);
glBindVertexArray(vao[1]); //VAO 2
glBindBuffer(GL_ARRAY_BUFFER, vbo[2]);//3nd VBO

```

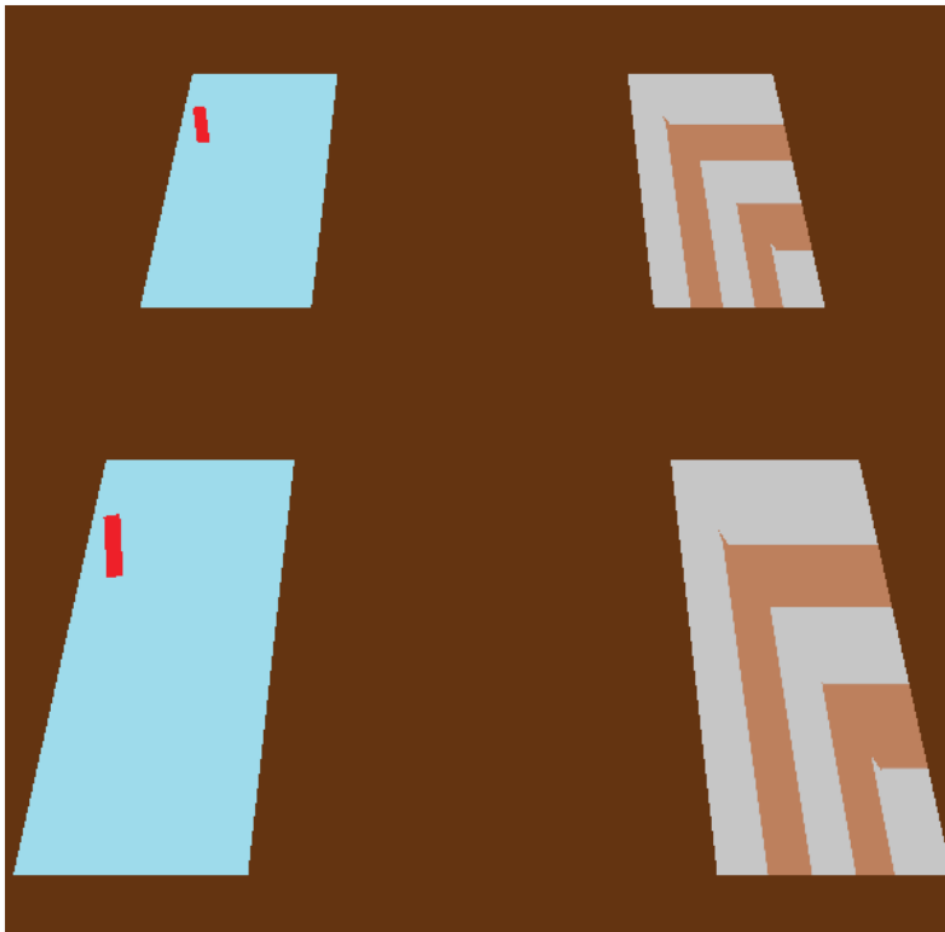
```

glBufferData(GL_ARRAY_BUFFER, sizeof(table), table, GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, vbo[3]); //4th VBO
glBufferData(GL_ARRAY_BUFFER, sizeof(card2Tex), card2Tex, GL_STATIC_DRAW);
glBindVertexArray(vao[2]); //VAO 3
glBindBuffer(GL_ARRAY_BUFFER, vbo[4]); //5th VBO
glBufferData(GL_ARRAY_BUFFER, sizeof(block1Pos), block1Pos,
GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, vbo[5]);
glBufferData(GL_ARRAY_BUFFER, sizeof(blockTex1), blockTex1,
GL_STATIC_DRAW);
}

```

This part binds the vertices that need drawing and the textures from above using the VAOs and VBOs that we pre-coded

Screenshots and explanations



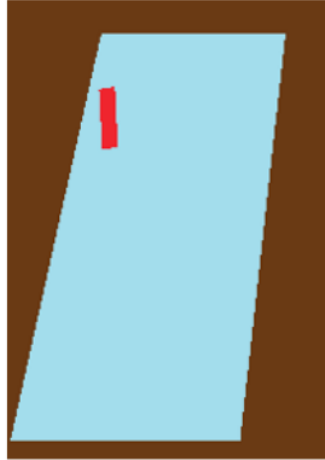
What the first sight will look like when the program is run



What the blocker is like after a few seconds of looking at the previous screen. Currently dose not do what is intended but this is showing what will happen once the timer is at 0. To see this in the code (since it is already there) deactivate lines 353-359 and reactivate 361 366.



Table that the cards are on uses a texture I made and implemented



Texture for the card that I made for the pair part of the game



Second card texture same as above

Other expectations and notes

- 1) The camera fixes itself once you press the "W" or "S" key

References

[1]	Scott gordon, john clevenger computer graphics programming in OpenGL with C++ Second Edition Kindle Edition
[2]	OpenGL “OpenGL Mathematics (GLM) [Online] https://www.opengl.org/sdk/libs/GLM/#:~:text=GLM%20is%20a%20C%2B%2B%20mathematics%20library%20for%20graphics,well%20which%20makes%20it%20really%20easy%20to%20use.

Games programming and graphics Assessment 2 - Ethan Walton-Deaves

GRADEMARK REPORT

FINAL GRADE

58/100

GENERAL COMMENTS

Instructor

Ref included. Test schedule provided.

You generated multiple textures and applied these. Well done. A time and block attempted. Fine but otherwise very little game play and user interaction. Your report included commentary of code but do also include more comments in the source code.

Objects placed in world space and camera placed - interactive movement - good, and camera transform applied correctly. Well done. Use of third party libraries. Good.

Mark may be capped if differentiated deadline and/or EC not in place.

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

GRADING FORM: SMU GENERIC_GJ

ETHAN WALTON-DEAVES

0

RESEARCH-INFORMED LITERATURE

KNOWLEDGE AND UNDERSTANDING

ANALYSIS

PRACTICAL APPLICATION AND DEP.

SKILLS FOR PROFESSIONAL PRAC.