

Cards_Richard Harris

by Rich Harris

FILE	6496_RICH_HARRIS_CARDS_RICHARD_HARRIS_311314_1996680599.DOCX (146.83K)		
TIME SUBMITTED	31-MAY-2021 08:42PM (UTC+0100)	WORD COUNT	3143
SUBMISSION ID	155489321	CHARACTER COUNT	21748

OpenGL Cards Game

Richard Harris

ACCA6021
(Advanced Graphics)

Introduction

The card game detailed in this report is unfinished but in depth description of the current build and its components are detailed below. Following Modern OpenGL development standards a visually pleasing scene has been created using Opengl shading language and common C++ standards.

Gameplay

Unfortunately no gameplay elements have not yet been implemented

Objects

The objects in the game have been created using Maya software and are loaded from a self - developed object loader, which converts all vertex positions, normals and texture coordinates into a structure array using the vector class library. All of the objects are loaded are a copy of the same vertex array drawn multiple times.

```
for (int i = 0; i < cardPositions.size(); i++)
{
    cardShader.SetInt("useTexture", random[i]);
    model = glm::mat4(1.0f);
    model = glm::translate(model, cardPositions[i]);
    cardShader.SetMat4("model", model * rotate_anim);
    glDrawArrays(GL_TRIANGLES, 0, vertex_size);
}
```

Textures

The textures are loaded from the stb_image library and drawn using the shader programs. Each texture is loaded and randomly drawn on each card for allowing multiple plays for when the card is completed.[1]

```
unsigned int LoadTexture(const char* path, int width, int height, int components, int desired_channels)
{
    unsigned int textureID;
    glGenTextures(1, &textureID);
    glBindTexture(GL_TEXTURE_2D, textureID);

    unsigned char* data = stbi_load(path, &width, &height, &components, desired_channels);

    if (data)
    {
        GLenum format = NULL;

        if (components == 1)
        {
            format = GL_RED;
        }
        else if (components == 3)
        {
            format = GL_RGB;
        }
        else if (components == 4)
        {
            format = GL_RGBA;
        }

        glTexImage2D(GL_TEXTURE_2D, 0, format, width, height, 0, format, GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);

        // Parameters for texture wrapping
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

        // Parameters for texture filtering
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

        // Clean up
        stbi_image_free(data);
    }
    else
    {
        std::cerr << "Texture failed to load at path: " << path << std::endl;

        // Clean up
        stbi_image_free(data);
    }

    return textureID;
}
```

Test Cases

CASE	Success/Fail	Result	Next
Loading and drawing a card	Success	Card loads onto screen without any bugs	Attach texture to card
Attach texture to card	Fail	Texture is not loading	Check shader program
Attach texture to card	Fail	Texture is not loading	Swap the normals of the object
Attach texture to card	Fail	Texture is not loading	Try a different texture slot
Attach texture to card	Success	Oddly, the compiler does not have a useable GLTEXTURE0 slot	Load multiple cards
Load multiple cards	Success	Cards are loading fine, need to translate their positions	translate card positions
Load multiple cards with new positions	Success	Card Positions are loaded and are evenly spaced	Load random textures on cards
Load random textures on cards	Fail	Texture are swapping rapidly	Make randomly generated numbers as a preset
Load random textures on cards	Success	Textures are applied correctly	Animate cards ready for gameplay mechanics
Animate cards	Success	Cards are currently animating constantly.	Generate mouse picking
Generate mouse picking	Fail	Used Unproject method not properly assigning the correct parameters	Debug code
Generate mouse picking	Fail	Unproject calculating hit in the wrong places	Debug code
Generate mouse picking	Fail	Unproject still calculating hit wrong	Debug code
Generate mouse picking	Fail	Unproject still calculating hit wrong	Check unproject parameters
Generate mouse picking	Fail	Unproject far coordinates seem to be unusualy	Use same parameters in house picking example
Generate mouse picking	Fail	Tested the same camera frustrum, position and angle on the house pick	Change parameters

		example, still far plane coordinates are different	
Generate mouse picking	Fail	Parameters edited still no change	Try using different mouse parameters
Generate mouse picking	Fail	Still hit in the wrong place	Mouse picking is to not be implemented.

Shader Program

The shader program is responsible for loading and colouring in the objects with the assigned textures, using logic tests to assign different textures to each object. The shader program also has an attempt at Blinn Phong shading. [2]

References

- [1] "stb/stb_image.h at master · nothings/stb · GitHub." [Online]. Available: https://github.com/nothings/stb/blob/master/stb_image.h. [Accessed: 31-May-2021].
- [2] "Learn OpenGL, extensive tutorial resource for learning Modern OpenGL." [Online]. Available: <https://learnopengl.com/>. [Accessed: 03-Dec-2020].

Appendix: Code Implemented

Application.cpp

```
#include <Windows.h>
```

```
#include <glad.h>
```

```
#include <glfw3.h>
```

```
#include <glm.hpp>
```

```
#include <gtc/matrix_transform.hpp>
```

```
#include <gtc/type_ptr.hpp>
```

```
#include <iostream>
```

```
#include <vector>
```

```
#include "Utilities/Shader.h"
```

```
#include "Utilities/Utilities.h"
```

```
#include "Utilities/ObjLoader.h"
```

```
#include "../vendor/stb_image/stb_image.h"
```

```
int main()
```

```
{
```

```
#pragma region Window/OpenGL Setup
```

```
    // Initialize GLFW
```

```
    glfwInit();
```

```
    // Set window hints
```

```

glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

glfwWindowHint(GLFW_SAMPLES, 8);

// Create window
GLFWwindow* window = glfwCreateWindow(SCREEN_WIDTH, SCREEN_HEIGHT, "OpenGL
Template", NULL, NULL);

// Check if window was created sucessfully
if (window == NULL)
{
    std::cout << "Failed to create GLFW window!" << std::endl;

    glfwTerminate();

    return -1;
}

// Make the current rendering context the window
glfwMakeContextCurrent(window);

#pragma region Callback Functions

// Register callback function for window resizing (callback functions should be registered
after window creation but before game loop)
glfwSetFramebufferSizeCallback(window, FramebufferSizeCallback);

// Register callback function for cursor position (callback functions should be registered after
window creation but before game loop)
glfwSetCursorPosCallback(window, MouseCallback);

```

```

// Tell GLFW to capture our mouse
glfwSetInputMode(window, GLFW_CURSOR_NORMAL, 0);

#pragma endregion

// Check all OpenGL function pointers have loaded
if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
{
    std::cout << "Failed to initialize GLAD" << std::endl;

    return -1;
}

#pragma endregion

// Arbitrary vertex data goes here
#pragma region Vertex Data

// Loads obj file and stores it into a vector of Vertex.
std::vector<Vertex> cardVerts;

ObjLoader Card("src/Resources/Models/Playing_Card_02.obj");
cardVerts = Card.StoreAll();
size_t vertex_size = cardVerts.size() * sizeof(Vertex);

/*for(int i = 0; i < cardVerts.size(); i++)
{
    std::cout << "UV X : " << cardVerts[i].TexCoords.x << " UV Y : " <<
cardVerts[i].TexCoords.y << std::endl;
}

```

```

    }

    */

    //TO DO: make this function return the full dimensions

    glm::vec3 objDMs = GetObjectDimensions(cardVerts);// reads through the vertex list and
returns the most -xyz value


#pragma endregion


    // Vertex Array Objects, Vertex Buffer Objects and specifying how our vertex data is
formatted all happens here

#pragma region Vertex Specification


    unsigned int cardVAO; // A value OpenGL can reference when we want to use this object
    unsigned int cardVBO;

    // Much like the triVAO, we can reference this VBO with this value


    // Using the triVAO and triVBO values as a reference to our objects,
    //we generate vertex arrays and buffers using said references
    glGenVertexArrays(1, &cardVAO);
    glGenBuffers(1, &cardVBO);


    // Bind the VAO first to say which VAO we want to bind subsequent VBOs to,
    //and then we follow up with binding the relevant VBOs
    glBindVertexArray(cardVAO);
    glBindBuffer(GL_ARRAY_BUFFER, cardVBO);

```

```

// boilerplate card positioning
//TO DO make this dynamically sorted/ such as testing against the screen space coordinates
glm::vec3 startPos(5.0f, -4.0f, 0.0f);
std::vector<glm::vec3> cardPositions{

    glm::vec3(startPos.x, startPos.y, -startPos.z),
    glm::vec3((startPos.x + objDMs.x - 0.7) , startPos.y, -startPos.z),
    glm::vec3(startPos.x + ((objDMs.x - 0.7) * 2), startPos.y,-startPos.z),
    glm::vec3(startPos.x + ((objDMs.x - 0.7) * 3), startPos.y,-startPos.z),
    glm::vec3(startPos.x + ((objDMs.x - 0.7) * 4), startPos.y,-startPos.z),
    glm::vec3(startPos.x + ((objDMs.x - 0.7) * 5), startPos.y,-startPos.z),

    glm::vec3(startPos.x, startPos.y - objDMs.y + 1, -startPos.z),
    glm::vec3((startPos.x + objDMs.x - 0.7) , startPos.y - objDMs.y + 1,-startPos.z),
    glm::vec3(startPos.x + ((objDMs.x - 0.7) * 4), startPos.y - objDMs.y + 1,-startPos.z),
    glm::vec3(startPos.x + ((objDMs.x - 0.7) * 5), startPos.y - objDMs.y + 1,-startPos.z),
    glm::vec3(startPos.x + ((objDMs.x - 0.7) * 2), startPos.y - objDMs.y + 1,-startPos.z),
    glm::vec3(startPos.x + ((objDMs.x - 0.7) * 3), startPos.y - objDMs.y + 1,-startPos.z),

    glm::vec3(startPos.x, (startPos.y - objDMs.y * 2) + 2, -startPos.z),
    glm::vec3((startPos.x + objDMs.x - 0.7) , (startPos.y - objDMs.y * 2) + 2, -startPos.z),
    glm::vec3(startPos.x + ((objDMs.x - 0.7) * 4), (startPos.y - objDMs.y * 2) + 2,-
startPos.z),
    glm::vec3(startPos.x + ((objDMs.x - 0.7) * 5), (startPos.y - objDMs.y * 2) + 2,-
startPos.z),
    glm::vec3(startPos.x + ((objDMs.x - 0.7) * 2), (startPos.y - objDMs.y * 2) + 2,-
startPos.z),
    glm::vec3(startPos.x + ((objDMs.x - 0.7) * 3), (startPos.y - objDMs.y * 2) + 2,-
startPos.z),

    glm::vec3(startPos.x, (startPos.y - objDMs.y * 3) + 3, -startPos.z),

```

```

        glm::vec3((startPos.x + objDMs.x - 0.7) , (startPos.y - objDMs.y * 3) + 3,-startPos.z),
        glm::vec3(startPos.x + ((objDMs.x - 0.7) * 4), (startPos.y - objDMs.y * 3) + 3,-
startPos.z),
        glm::vec3(startPos.x + ((objDMs.x - 0.7) * 5), (startPos.y - objDMs.y * 3) + 3,-
startPos.z),
        glm::vec3(startPos.x + ((objDMs.x - 0.7) * 2), (startPos.y - objDMs.y * 3) + 3,-
startPos.z),
        glm::vec3(startPos.x + ((objDMs.x - 0.7) * 3), (startPos.y - objDMs.y * 3) + 3,-
startPos.z),

};

// Assigning our cardVerts array data to our vertex buffer object

//using cardVerts.data() is the only way to fully allow a dynamic sized obj to placed and read
properly from the buffer array
glBufferData(GL_ARRAY_BUFFER, vertex_size, cardVerts.data(), GL_DYNAMIC_DRAW);

// Position data
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex,
Vertex::Position));

// Enable the vertex data we formatted above
glEnableVertexAttribArray(0);

// Texture coordinate data
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex,
Vertex::TexCoords));

// Enable the vertex data we formatted above
glEnableVertexAttribArray(1);

```

```
glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Vertex::Normal));
```

```
// Enable the vertex data we formatted above
```

```
glEnableVertexAttribArray(2);
```

```
#pragma endregion
```

```
// Load our shaders here to compile and link them for execution on the GPU
```

```
Shader cardShader("src/Resources/Shaders/Flat.vert", "src/Resources/Shaders/Flat.frag");
```

```
#pragma region Texture Loading
```

```
/// <summary>
```

```
/// loading multiple textures from encapped std_image function
```

```
/// </summary>
```

```
/// <returns></returns>
```

```
unsigned int textureID1 = LoadTexture("src/Resources/Textures/boltCard.png", 512, 512, 3, 0);
```

```
unsigned int textureID2 = LoadTexture("src/Resources/Textures/cheatCard.png", 512, 512, 3, 0);
```

```
unsigned int textureID3 = LoadTexture("src/Resources/Textures/heartCard.png", 512, 512, 3, 0);
```

```
unsigned int textureID4 = LoadTexture("src/Resources/Textures/moonCard.png", 512, 512, 3, 0);
```

```
unsigned int textureID5 = LoadTexture("src/Resources/Textures/smileyCard.png", 512, 512, 3, 0);
```

```
    unsigned int textureID6 = LoadTexture("src/Resources/Textures/starCard.png", 512, 512, 3, 0);
```

```
    // set up randomly generated numbers for texture assignment later
```

```
    srand(time(NULL));
```

```
    std::vector<int> random(cardPositions.size());
```

```
    for (int i = 0; i < random.size(); i++)
```

```
        random[i] = rand() % 6 + 1;
```

```
#pragma endregion
```

```
    // Render loop (this runs for as long as the window is open)
```

```
    while (!glfwWindowShouldClose(window))
```

```
    {
```

```
        // Timestep (if your interested there is a fantastic article here about timesteps  
https://gafferongames.com/post/fix\_your\_timestep/)
```

```
        // If we have enough time we may be able to discuss timesteps in class
```

```
        currentFrame = (float)glfwGetTime();
```

```
        deltaTime = currentFrame - lastFrame;
```

```
        lastFrame = currentFrame;

        // Process our mouse and keyboard input
        ProcessInput(window);

        // Enable the depth test
        glEnable(GL_DEPTH_TEST);

        // Set window background colour
        glClearColor(0.5f, 0.25f, 0.25f, 1.0f);

        // Clear the colour and depth buffer
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);


        // Binding our vertex arrays and shaders for our draw calls will happen here
#pragma region Render Here

        // Work with our cardVerts vertex array
        glBindVertexArray(cardVAO);

        // Returning our view matrix

        glm::mat4 view = camera.GetViewMatrix();

        glm::mat4 projection = glm::perspective(glm::radians(camera.zoom),
(float)SCREEN_WIDTH / (float)SCREEN_HEIGHT, 0.1f, 100.0f);
```

```
cardShader.SetMat4("projection", projection);  
cardShader.SetMat4("view", view);
```

```
        //set up rotations: rotation-anim is the animation and rotated is the back of the card  
facing the camera
```

```
        glm::mat4 model = glm::mat4(1.0f);  
        glm::mat4 rotate_anim = glm::rotate(model, 1.0f * (float)glfwGetTime(),  
glm::vec3(0.0f, 1.0f, 0.0f));  
        glm::mat4 rotated = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f,  
0.0f));
```

```
        //enable the shader  
cardShader.Use();
```

```
        //assign textures to the slot in the shader program
```

```
cardShader.SetInt("texture1", textureID1);  
cardShader.SetInt("texture2", textureID2);  
cardShader.SetInt("texture3", textureID3);  
cardShader.SetInt("texture4", textureID4);  
cardShader.SetInt("texture5", textureID5);  
cardShader.SetInt("texture6", textureID6);
```

```
        //activate and bind the textures
```

```
glActiveTexture(GL_TEXTURE1);  
glBindTexture(GL_TEXTURE_2D, textureID1);
```

```
glActiveTexture(GL_TEXTURE2);  
glBindTexture(GL_TEXTURE_2D, textureID2);
```

```
glActiveTexture(GL_TEXTURE3);
```

```
glBindTexture(GL_TEXTURE_2D, textureID3);
```

```
glActiveTexture(GL_TEXTURE4);
```

```
glBindTexture(GL_TEXTURE_2D, textureID4);
```

```
glActiveTexture(GL_TEXTURE5);
```

```
glBindTexture(GL_TEXTURE_2D, textureID5);
```

```
glActiveTexture(GL_TEXTURE6);
```

```
glBindTexture(GL_TEXTURE_2D, textureID6);
```

```
//place the cards and give them a texture each from the random numbers  
genertead earlier, and at the moment give them a spin.
```

```
for (int i = 0; i < cardPositions.size(); i++)
```

```
{
```

```
    cardShader.SetInt("useTexture", random[i]);
```

```
    model = glm::mat4(1.0f);
```

```
    model = glm::translate(model, cardPositions[i]);
```

```
    cardShader.SetMat4("model", model * rotate_anim);
```

```
    glDrawArrays(GL_TRIANGLES, 0, vertex_size);
```

```
}
```

```
/// <summary>
```

```
/// this is the latest attempt at the mouse picking, unfortunately something with the  
unprojected far plane is not working correctly
```

```
/// </summary>
```

```

    /// <returns></returns>

    glm::mat4 modelView(1.0f);

    glm::vec3 objNear = glm::unProject(glm::vec3(mouser.x * SCREEN_WIDTH, mouser.y *
    * SCREEN_HEIGHT, -1.0f), modelView,
        projection, glm::vec4(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT));

    glm::vec3 objFar = glm::unProject(glm::vec3(mouser.x * SCREEN_WIDTH, mouser.y *
    SCREEN_HEIGHT, 1.0f), modelView,
        projection, glm::vec4(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT));

    glm::vec3 ray(objNear);
    glm::vec3 direction = (objFar - objNear);

    glm::vec3 w(cardPositions[0] - ray);
    float vsq = glm::dot(direction, direction); //direction squared
    float wsq = glm::dot(w, w); //w squared
    float proj = glm::dot(w,direction); //w.v
    float result = wsq - proj * proj / vsq;
    float test = (objDMs.x * objDMs.x);
    //std::cout << "test: " << test << std::endl;

    std::cout << "near: " << objNear.x << ", "<< objNear.y << ", "<< objNear.z <<
std::endl;

    std::cout << "far: " << objFar.x << ", "<< objFar.y << ", "<< objFar.z << std::endl;

    //std::cout << "direction: " << direction.x << ", "<< direction.y << ", "<< direction.z <<
std::endl;

```

```
//std::cout << "result: " << result << std::endl;
```

```
    if (result > test)
    {
        std::cout << "hit" << std::endl;
    }
```

```
        //give the cards some blinn phone shading
        cardShader.SetVec3("dirLight.colour", 1.0f, 1.0f, 1.0f);
        cardShader.SetFloat("dirLight.AmbientIntensity", 0.5f);
        cardShader.SetVec3("dirLight.direction", glm::vec3(camera.front));
        cardShader.SetFloat("dirLight.diffuseIntensity", 0.3f);
        cardShader.SetFloat("material.specularIntensity", 1.0f);
        cardShader.SetFloat("material.shininess", 32.0f);
        cardShader.SetVec3("cameraPos", camera.position);
        cardShader.SetBool("blinnPhong", true);
```

```
#pragma endregion
```

```
    glfwSwapBuffers(window);
```

```
        // Get GLFW to poll any events (window and input events, this is where GLFW
callbacks occur)
```

```
        glfwPollEvents();
```

```

    }

    // Clean up
    glfwTerminate();

    return 0;
}

Flat.vert
#version 430 core

// Input the location of our vertex position in our vertex buffer
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec2 aTexCoords;
layout(location = 2) in vec3 norm;

out vec2 TexCoords;
out vec3 Normal;
out vec3 FragPos;

out vec3 oNorm;

// Specify our uniforms (this is our data from our application which is CPU bound)
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
    oNorm = norm;

    TexCoords = aTexCoords;

    gl_Position = projection * view * model *vec4(aPos, 1.0);

    Normal= mat3(transpose(inverse(model)))*oNorm;

    FragPos = (gl_Position * vec4(aPos, 1.0)).xyz;
}

```

Flat.frag

```
#version 430 core

out vec4 FragColour;

in vec2 TexCoords;
in vec3 Normal;
in vec3 FragPos;

in vec3 oNorm;

struct DirectionalLight
{
    vec3 colour;
    float AmbientIntensity;
    vec3 direction;
    float diffuseIntensity;
};

struct Material
{
    float specularIntensity;
    float shininess;
};

uniform sampler2D texture1;
uniform sampler2D texture2;
uniform sampler2D texture3;
uniform sampler2D texture4;
uniform sampler2D texture5;
uniform sampler2D texture6;

uniform int useTexture;
//

uniform DirectionalLight dirlight;
uniform Material material;
uniform vec3 cameraPos;
uniform bool blinnPhong;
vec3 fragToCam= normalize(cameraPos - FragPos);

vec3 CalcDirLight(DirectionalLight light, vec3 normal)
{
    // ambient
    vec3 ambient = light.AmbientIntensity * light.colour;
    // diffuse
    vec3 norm = normalize(Normal);
    vec3 lightDir = -normalize(cameraPos - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * light.colour;
    // specular
    float specularStrength = 0.5;
    vec3 viewDir = normalize(-FragPos); // the viewer is always at (0,0,0) in view-
    space, so viewDir is (0,0,0) - Position => -Position
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
    vec3 specular = material.specularIntensity * spec * light.colour;

    return vec3(ambient+diffuse+spec);
}
```

```

void main()
{
    vec4 finalColour = vec4(1.0f);
    if(useTexture!= 0)
    {
        if(useTexture==1)
        {
            finalColour = texture(texture1, TexCoords) *
vec4(CalcDirLight(dirLight, Normal),1.0f);
        }
        if(useTexture==2)
        {
            finalColour = texture(texture2, TexCoords) *
vec4(CalcDirLight(dirLight, Normal),1.0f);
        }
        if(useTexture==3)
        {
            finalColour = texture(texture3, TexCoords) *
vec4(CalcDirLight(dirLight, Normal),1.0f);
        }
        if(useTexture==4)
        {
            finalColour = texture(texture4, TexCoords) *
vec4(CalcDirLight(dirLight, Normal),1.0f);
        }
        if(useTexture==5)
        {
            finalColour = texture(texture5, TexCoords) *
vec4(CalcDirLight(dirLight, Normal),1.0f);
        }
        if(useTexture==6)
        {
            finalColour = texture(texture6, TexCoords) *
vec4(CalcDirLight(dirLight, Normal),1.0f);
        }
    }else
    {
        finalColour = vec4(0.0f,0.5f,0.1f,1.0f) * vec4(CalcDirLight(dirLight,
Normal),1.0f);
    }

    FragColour = finalColour;
}

```

ObjLoader.cpp

```

#include "ObjLoader.h"

ObjLoader::~ObjLoader()
{
}

void ObjLoader::CheckFile(const char* p)
{
    if(!p || p== NULL)
    {
        std::cout << "NO FILE FOUND" << std::endl;
    }
}

```

```

    }

}

// this reads the file and pushes all the information in an expanding vector array.
//parses the f values and pushes them on to a Vertex array with the corresponding f
number.
//then returns the Vertex array.
std::vector<Vertex> ObjLoader::StoreAll()
{
    Vertex vert_local(glm::vec3(0,0,0), glm::vec3(0, 0, 0), glm::vec2(0, 0));
    float x, y, z; // temp values
    std::string data;
    std::string line = " ";

    std::ifstream file(filePath);

    while (!file.eof())
    {
        /*getline(file, line);
        std::cout << line << std::endl;*/ // test file read

        getline(file, line);
        if (line.compare(0, 2, "v ") == 0)
        { // vertex position ("v" case)
            std::stringstream ss(line.erase(0, 1));
            ss >> x; ss >> y; ss >> z;
            v.push_back(glm::vec3(x/10, y/10, z/10));
            //vert_local = (Vertex(glm::vec3(x/10, y/10, z/10)));
        }

        if (line.compare(0, 3, "vn ") == 0)
        { // texture position ("vn" case)
            std::stringstream ss(line.erase(0, 2));
            ss >> x; ss >> y; ss >> z;
            vn.push_back(glm::vec3(x, y, z));
        }

        if (line.compare(0, 3, "vt ") == 0)
        { // normals position ("vt" case)
            std::stringstream ss(line.erase(0, 2));
            ss >> x; ss >> y; ss >> z;
            vt.push_back(glm::vec2(x, y*-1));
        }

        //if (line.compare(0, 2, "f ") == 0)
        //{ // faces ("f" case)
        //    std::stringstream ss(line.erase(0, 1));
        //    ss >> x; ss >> y; ss >> z;
        //    f.push_back(glm::vec3(x, y, z));
        //}

    }
}

```

```

        if (line.compare(0, 2, "f ") == 0) { // triangle faces ("f" case)
            std::string oneCorner, vert, tex, norm;
            std::stringstream ss(line.erase(0, 2));
            for (int i = 0; i < 3; i++) {
                getline(ss, oneCorner, ' '); // extract triangle face

                std::stringstream oneCornerSS(oneCorner);
                getline(oneCornerSS, vert, '/');
                getline(oneCornerSS, tex, '/');
                getline(oneCornerSS, norm, '/');
                int vertRef = (stoi(vert) - 1); // "stoi" converts string
                int tcRef = (stoi(tex) - 1);
                int normRef = (stoi(norm) - 1);
                verts.push_back(Vertex(v[vertRef], vn[normRef], vt[tcRef]));
            }
        }

        /* if (v.empty())
        {
            std::cout << "empty" << std::endl;
        }
        vert_local=(Vertex(v.back(), vn.back(), vt.back()) );
        verts.push_back(vert_local);
    } */

    file.close();
    return verts;
}

```

ObjLoader.h

```
#pragma once
#include <iostream>
#include <fstream>
#include <sstream>
#include<vector>
#include <glm.hpp>
#include "Utilities.h"

class ObjLoader
{
#pragma region Vertex Structs

public:
    // upon creating object ask user for name of file, check if it exists, then
    open AND READS the file
    ObjLoader(const char* path) { filePath = path; CheckFile(filePath); }
    ~ObjLoader();

    void CheckFile(const char* p);

    std::vector<Vertex> StoreAll();

private:
    const char* filePath;

    std::vector<glm::vec3> v;
    std::vector<glm::vec2> vt;
    std::vector<glm::vec3> vn;
    std::vector<glm::vec3> f;
    std::vector<Vertex> verts;

};
```

Utilities.h(The functions that I implemented)

```
extern glm::vec3 GetObjectDimensions(std::vector<Vertex> objectData);

extern unsigned int LoadTexture(const char* path, int width, int height, int
components, int desired_channels);

extern void OBB_Intersect(glm::vec3 ray_origin, glm::vec3 ray_direction, glm::vec3
AABB_min,
    glm::vec3 AABB_max);

extern void Pick_test(glm::vec3 mouse, glm::mat4 modelview, glm::mat4 proj, glm::vec4
viewport);
```

Utilities.cpp(The functions that I implemented)

```
glm::vec3 GetObjectDimensions(std::vector<Vertex> objectData)
{
    glm::vec3 data1 = objectData[0].Position;
    glm::vec3 data2 = glm::vec3(0.0f);
    glm::vec3 objDM = glm::vec3(0.0f);

    for (int i = 0; i < objectData.size(); i++)
    {
        if (objectData[i].Position.x < data1.x)
        {
            data1.x = objectData[i].Position.x;
            std::cout << "beginning vert moved " << data1.x << std::endl;
        }
        > data2.x)
        else if (objectData[i].Position.x > data1.x && objectData[i].Position.x
        {
            data2.x = objectData[i].Position.x;
            std::cout << "end vert moved " << data2.x << std::endl;
        }

        if (objectData[i].Position.y < data1.y)
        {
            data1.y = objectData[i].Position.y;
            std::cout << "beginning vert moved " << data1.y << std::endl;
        }
        > data2.y)
        else if (objectData[i].Position.y > data1.y && objectData[i].Position.y
        {
            data2.y = objectData[i].Position.y;
            std::cout << "end vert moved " << data2.y << std::endl;
        }

        if (objectData[i].Position.z < data1.z)
        {
            data1.z = objectData[i].Position.z;
            std::cout << "beginning vert moved " << data1.z << std::endl;
        }
        > data2.z)
        else if (objectData[i].Position.z > data1.z && objectData[i].Position.z
        {
            data2.z = objectData[i].Position.z;
            std::cout << "end vert moved " << data2.z << std::endl;
        }
    }
}
```

```

    }

    }
    objDM = glm::vec3(-abs(data1.x - data2.x) + 0.5, -abs(data1.y - data2.y) +
0.5, -abs(data1.z - data2.z) + 0.5);
    return objDM;
}

unsigned int LoadTexture(const char* path, int width, int height, int components, int
desired_channels)
{
    unsigned int textureID;
    glGenTextures(1, &textureID);
    glBindTexture(GL_TEXTURE_2D, textureID);

    unsigned char* data = stbi_load(path, &width, &height, &components,
desired_channels);

    if (data)
    {
        GLenum format = NULL;

        if (components == 1)
        {
            format = GL_RED;
        }
        else if (components == 3)
        {
            format = GL_RGB;
        }
        else if (components == 4)
        {
            format = GL_RGBA;
        }

        glTexImage2D(GL_TEXTURE_2D, 0, format, width, height, 0, format,
GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);

        // Parameters for texture wrapping
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

        // Parameters for texture filtering
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

        // Clean up
        stbi_image_free(data);
    }
    else
    {
        std::cerr << "Texture failed to load at path: " << path << std::endl;

        // Clean up
        stbi_image_free(data);
    }

    return textureID;
}
/// <summary>

```

```

/// more different attempts at the mouse picking
/// </summary>
/// <param name="ray_origin"></param>
/// <param name="ray_direction"></param>
/// <param name="AABB_min"></param>
/// <param name="AABB_max"></param>
extern void OBB_Intersect(glm::vec3 ray_origin, glm::vec3 ray_direction, glm::vec3
AABB_min,
    glm::vec3 AABB_max)
{

    glm::vec3 near; // nearest intersection right now
    glm::vec3 far; // "infinty" intersection
    //ray_origin = glm::vec3(0.0f, 0.0f, 0.0f);
    ray_direction = far - near;

    glm::vec3 w = (AABB_max - AABB_min) - ray_origin;
    float dirSqu = glm::dot(ray_direction, ray_direction);
    float wSqu = glm::dot(w, w);
    float proj = glm::dot(w, ray_direction);

    float lineSquared = wSqu - proj * proj / dirSqu;

    if (lineSquared > AABB_min.x && lineSquared < AABB_max.x)
    {
        if (lineSquared > AABB_min.y && lineSquared < AABB_max.y)
        {
            std::cout << "hit" << std::endl;
        }
        else
        {
            std::cout << "miss" << std::endl;
        }
    }

}

void Pick_test(glm::vec3 mouse, glm::mat4 modelview, glm::mat4 proj, glm::vec4
viewport)
{
    mouser = glm::unProject(glm::vec3(lastX, lastY, 0.0f), modelview, proj,
viewport);
}

```

Cards_Richard Harris

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

60/100

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

PAGE 27

GRADING FORM: SMU GENERIC_GJ

RICH HARRIS

59.50

RESEARCH-INFORMED LITERATURE



You included some references. Do use OGL official published texts - not just web references. Evidence of research into the shading and pgraphics programming methods and techniques required fro this assignment.

3

KNOWLEDGE AND UNDERSTANDING



Textures loaded and shader developed which demonstrates lighting effects. Blinn Phong attempted here? Specular lighting included. Good.

8

ANALYSIS

 A fair interpretation of the spec but no game play added. Problems with picking? Testing detailed. In general avoid documenting failed tests but here I see it helps in explaining your approach to development. Very detailed. You could have used the keyboard for input.

12.5

PRACTICAL APPLICATION AND DEP.

 DSome very good graphics work here but I see that the problems you have with ray casing has prevented you from going further with your assignment. Textures, Shading and some animation in evidentce. Object loading too. Good. Shame that there is no Game play in evidence.

32.5

SKILLS FOR PROFESSIONAL PRAC.

 Structured report with detailed testing. Refs provided.

3.5