

Assignment 2, Nathan Willis

by Nathan Willis

Submission date: 17-Jan-2025 02:25AM (UTC+0000)

Submission ID: 248773942

File name: 41436_Nathan_Willis_Assignment_2__Nathan_Willis_628113_1465673862.docx (128.73K)

Word count: 1665

Character count: 12115



Graphics assignment 2

Nathan Willis (2103532)



Contents

Design.....	2
Card Class.....	2
CardGame Class.....	2
User Guide.....	2
Starting the game.....	2
Controls.....	2
The game.....	2
Testing.....	2
Testing Game logic, rotation and shuffling with screenshots more testing provided in video in appendix.....	2
References.....	5
Video of "Game".....	5
Code.....	5
Card Class.....	5
Card.h.....	5
Card.cpp.....	5
CardGame Class.....	7
CardGame.h.....	7
CardGame.cpp.....	8

Design

Card Class

As the assignment included the rendering of several card objects the most obvious way to do this, for me, was to create a card class and call it multiple times. The class handled things like card rendering, textures and most of the transformations applied to the cards, things like card rotation etc.

CardGame Class

This class handles game logic and user input[1], the update function is called in main and contains most of the game logic. The CardGame class also contains member functions for handling the rotation and shuffling of cards it does the prior by calling a function from the card class and the later by manipulating the card objects as they are.

User Guide

Starting the game

Pressing the test game button initiates the game and disables the mouse cursor.

Controls

WASD control the movement of the camera, spacebar rotates the cards and reshuffles them, numbers 1-4 select cards and scale them up by 1.25 and pressing the enter key runs the check for if the cards selected match.

The game

When the game initialises, the cards rotate until they are at 180 degrees and facing the camera and then they swap randomly, the player must then use numbers 1-4 to select 2 matching cards. This is not very hard to do as the player can move the camera behind the cards to view them. The user can also press enter to restart the rotation and shuffling process.

Testing

Testing Game logic, rotation and shuffling with screenshots
more testing provided in video in appendix

Testing wrong combo

ACE

ACE

2

2

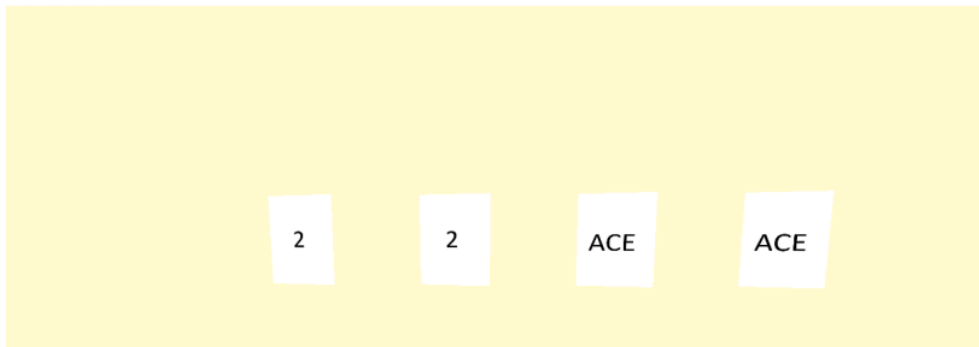
```
GLEW version: 3.3.0 NVIDIA 566.36  
Shader id:2  
Shader id:4  
Shader initialized successfully  
Card initialized successfully  
Shuffling cards  
YOU LOSE
```

```
GLEW version: 3.3.0 NVIDIA 566.36
Shader id:2
Shader id:4
Shader initialized successfully
Card initialized successfully
Shuffling cards
YOU LOSE
YOU LOSE
YOU LOSE
YOU LOSE
Starting rotation
Starting rotation
Starting rotation
Starting rotation
Starting rotation
Shuffling cards
```

All selected



None selected



References

[1]"LearnOpenGL - Camera," Learnopengl. <https://learnopengl.com/Getting-started/Camera> (accessed Jan. 17, 2025).

Video of "Game"

<https://youtu.be/6-JLInLtwQ>

Code

Card Class

Card.h

```
#pragma once
#include "VertexBuffer.h"
#include "VertexBufferLayout.h"
#include "IndexBuffer.h"
#include "VertexArray.h"
#include "Shader.h"
#include <memory>
#include "GL/glew.h"
#include <GLFW/glfw3.h>
#include "glm/gtc/matrix_transform.hpp"
#include "glm/glm.hpp"
#include "Texture.h"
class Card {
public:
    glm::vec3 position;
    glm::vec3 rotation;
    glm::vec3 scale;
    std::unique_ptr<VertexArray> m_VAO;
    std::unique_ptr<VertexBuffer> m_VBO;
    std::unique_ptr<IndexBuffer> m_IndexBuffer;
    std::unique_ptr<Texture> m_Texture;
    std::unique_ptr<Texture> backTexture;
    bool frontVisible;
    Card(glm::vec3 pos, const std::string& texturePath);
    void Draw(Shader& shader);
    void DrawBack(Shader& shader);
    void Rotation(float angle);
    int getPos();
    void select();
    bool selected;
};
```

Card.cpp

```
#include "Card.h"
#include "VertexBufferLayout.h"
#include "Renderer.h"
#include <glm/gtc/matrix_transform.hpp>

// Constructor to initialize a card with a position and textures
Card::Card(glm::vec3 pos, const std::string& frontTexturePath)
    : position(pos), rotation(0.0f), scale(1.0f), selected(false) {
```

```

// Vertex data for a card (2D rectangle) with texture coordinates for both
sides
const float cardVertices[] = {
    // Front positions      // Front Texture Coords
    -0.5f, -0.75f, 0.0f,    0.0f, 0.0f, // Bottom-left
    0.5f, -0.75f, 0.0f,    1.0f, 0.0f, // Bottom-right
    0.5f, 0.75f, 0.0f,     1.0f, 1.0f, // Top-right
    -0.5f, 0.75f, 0.0f,    0.0f, 1.0f, // Top-left

    // Back positions      // Back Texture Coords
    0.5f, -0.75f, -0.001f,  0.0f, 0.0f, // Bottom-left
    -0.5f, -0.75f, -0.001f, 1.0f, 0.0f, // Bottom-right
    -0.5f, 0.75f, -0.001f,  1.0f, 1.0f, // Top-right
    0.5f, 0.75f, -0.001f,   0.0f, 1.0f, // Top-left
};

const unsigned int cardIndices[] = {
    // Front indices
    0, 1, 2, 2, 3, 0,
    // Back indices
    4, 5, 6, 6, 7, 4
};

// Initialize the VAO, VBO, and IndexBuffer for the card
m_VAO = std::make_unique<VertexArray>();
m_VBO = std::make_unique<VertexBuffer>(cardVertices, sizeof(cardVertices));
m_IndexBuffer = std::make_unique<IndexBuffer>(cardIndices,
sizeof(cardIndices) / sizeof(unsigned int));

VertexBufferLayout layout;
layout.Push<float>(3); // Position
layout.Push<float>(2); // Texture coordinates
m_VAO->AddBuffer(*m_VBO, layout);

m_Texture = std::make_unique<Texture>(frontTexturePath);
backTexture = std::make_unique<Texture>("src/vendor/back.jpg");
}

// Method to draw the front of the card using the given shader
void Card::Draw(Shader& shader) {
    m_Texture->Bind(); // Bind the front texture
    m_VAO->Bind();
    m_IndexBuffer->Bind();

    // Create the model matrix for the card's transformations
    glm::mat4 model = glm::mat4(1.0f);
    model = glm::translate(model, position);
    model = glm::rotate(model, glm::radians(rotation.x), glm::vec3(1.0f, 0.0f,
0.0f));
    model = glm::rotate(model, glm::radians(rotation.y), glm::vec3(0.0f, 1.0f,
0.0f));
    model = glm::rotate(model, glm::radians(rotation.z), glm::vec3(0.0f, 0.0f,
1.0f));
    model = glm::scale(model, scale);

    // Set the model matrix in the shader
    shader.setUniformMat4f("model", model);

    // Draw the front of the card
    glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, nullptr);
}

// Method to draw the back of the card using the given shader
void Card::DrawBack(Shader& shader) {

```

```

        backTexture->Bind(); // Bind the back texture
        m_VAO->Bind();
        m_IndexBuffer->Bind();

        // Create the model matrix for the card's transformations
        glm::mat4 model = glm::mat4(1.0f);
        model = glm::translate(model, position);
        model = glm::rotate(model, glm::radians(rotation.x), glm::vec3(1.0f, 0.0f,
0.0f));
        model = glm::rotate(model, glm::radians(rotation.y), glm::vec3(0.0f, 1.0f,
0.0f));
        model = glm::rotate(model, glm::radians(rotation.z), glm::vec3(0.0f, 0.0f,
1.0f));
        model = glm::scale(model, scale);
        // Set the model matrix in the shader
        shader.setUniformMat4f("model", model);

        // Draw the back of the card
        glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, (void*)(6 *
sizeof(unsigned int)));
    }
    void Card::Rotation(float angle) {
        rotation.y = angle;
    }

    int Card::getPos() {
        return position.x;
    }

    void Card::select()
    {
        selected = !selected;
        if (selected) {
            scale = glm::vec3(1.25f, 1.25f, 1.25f);
        }
        else {
            scale = glm::vec3(1.0f, 1.0f, 1.0f);
        }
    }
}

```

CardGame Class

CardGame.h

```

#pragma once
#include "Tests.h"
#include "../VertexBuffer.h"
#include "../VertexBufferLayout.h"
#include "../IndexBuffer.h"
#include "../VertexArray.h"
#include "../Shader.h"
#include <memory>
#include "GL/glew.h"
#include <GLFW/glfw3.h>
#include "../Card.h"
#include "glm/glm.hpp"
namespace test {
    class CardGame : public Tests
    {
    public:
        CardGame(GLFWwindow* window);
        void Update(float deltaTime) override;
    };
}

```

```

        void Render() override;
        void RenderGUI() override;
        std::vector<Card> cards;
    private:
        GLFWwindow* m_window;

        glm::vec3 cameraPosition;
        glm::vec3 cameraFront;
        glm::vec3 cameraUp;
        float cameraSpeed;
        std::unique_ptr<VertexArray> m_VAO;
        std::unique_ptr<IndexBuffer> m_IndexBuffer;
        std::unique_ptr<Shader> m_Shader;
        std::unique_ptr<VertexBuffer> m_VBO;
        std::unique_ptr<VertexArray> f_VAO;
        std::unique_ptr<IndexBuffer> f_IndexBuffer;
        std::unique_ptr<Shader> f_Shader;
        std::unique_ptr<VertexBuffer> f_VBO;
        float angle;
        bool shuffCheck;
        bool rotationComplete;
        void prosInput();
        void mouseCallback(GLFWwindow* window);
        float lastX;
        float lastY;
        float yaw;
        float pitch;
        bool firstMouse;
        void startRotation();
        void shuffle();
        void SelectCard(int i);
        void winCheck();
        bool pressed1;
        bool pressed2;
        bool pressed3;
        bool pressed4;
        bool pressedEnter;
        bool pressedEsc;
    };
}

```

CardGame.cpp

```

#include "CardGame.h"
#include "../Renderer.h"
#include "imgui.h"
#include <GL/glew.h>
#include "GLFW/glfw3.h"
#include "glm/gtc/matrix_transform.hpp"
#include <glm/glm.hpp>
#include <glm/gtc/type_ptr.hpp>
#include "../Card.h"
#include <algorithm>
#include <random>
#include <ctime>
#include <cstdlib>

test::CardGame::CardGame(GLFWwindow* window) :
    m_window(window),
    cameraPosition(0.0f, 5.0f, -1.0f),
    cameraFront(1.5f, 0.0f, 0.0f),
    cameraUp(0.0f, 1.0f, 0.0f),
    cameraSpeed(0.1f),
    lastX(400.0f),
    lastY(300.0f),

```

```

        firstMouse(true),
        yaw(90.0f),
        pitch(0.0f),
        angle(0.0f),
        shuffCheck(false),
        rotationComplete(false),
        pressed1(false),
        pressed2(false),
        pressed3(false),
        pressed4(false),
        pressedEnter(false)

    {
        glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);

        // Initialize the shader with the combined shader file
        m_Shader = std::make_unique<Shader>("src/shaders/Card.shader");

        std::cout << "Shader initialized successfully" << std::endl;

        // Enable depth testing
        glEnable(GL_DEPTH_TEST);

        // Initialize one card with sample position and texture path
        cards.push_back(Card(glm::vec3( 2.f, 5.f, 2.f), "src/vendor/ACE.jpg"));
        cards.push_back(Card(glm::vec3( 0.f, 5.f, 2.f), "src/vendor/ACE.jpg"));
        cards.push_back(Card(glm::vec3(-2.f, 5.f, 2.f), "src/vendor/2jpg.jpg"));
        cards.push_back(Card(glm::vec3(-4.f, 5.f, 2.f), "src/vendor/2jpg.jpg"));

        std::cout << "Card initialized successfully" << std::endl;
    }

void test::CardGame::Update(float deltaTime) {
    mouseCallback(m_window);
    prosInput();

    static float rotationSpeed = 1.0f;
    //std::cout << "Delta Time: " << deltaTime << "\n";

    if (!shuffCheck && !rotationComplete) {
        angle += rotationSpeed;
        //std::cout << "Rotation Angle: " << angle << "\n";

        // Rotate each card
        for (auto& card : cards) {
            card.Rotation(angle);
        }

        // Check if rotation is complete
        if (angle >= 180.0f) {
            rotationComplete = true;
            shuffCheck = true; // Start shuffling
            shuffle();
        }
    }
    if ((shuffCheck && rotationComplete) && angle<=360) {
        angle += rotationSpeed;
        for (auto& card : cards) {
            card.Rotation(angle);
        }
    }
}

```

```

void test::CardGame::Render()
{
    Renderer renderer;
    renderer.Clear();
    renderer.ClearColourWhite();

    m_Shader->Bind();
    glm::mat4 view = glm::lookAt(cameraPosition, cameraPosition + cameraFront,
cameraUp);
    glm::mat4 projection = glm::perspective(45.f, 800.0f / 600.0f, 0.1f,
100.f);
    m_Shader->setUniformMat4f("view", view);
    m_Shader->setUniformMat4f("projection", projection);

    for (auto& card : cards) {
        card.Draw(*m_Shader);
        card.DrawBack(*m_Shader);
    }
}

void test::CardGame::RenderGUI()
{
    ImGui::SliderFloat("speed", &cameraSpeed, 0.1f, 5.f);
    ImGui::Text("position: %.1f,%.1f,%.1f", cameraPosition.x, cameraPosition.y,
cameraPosition.z);
    ImGui::Text("camera front %.1f,%.1f,%.1f", cameraFront.x, cameraFront.y,
cameraFront.z);
    float rate = ImGui::GetIO().Framerate;
    ImGui::Text("framerate, %.4f", 100 / rate);
}

void test::CardGame::prosInput()
{
    GLFWwindow* window = glfwGetCurrentContext();

    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS) {
        cameraPosition.x += cameraSpeed * cameraFront.x;
        cameraPosition.z += cameraSpeed * cameraFront.z;
    }
    if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS) {
        cameraPosition.x -= cameraSpeed * cameraFront.x;
        cameraPosition.z -= cameraSpeed * cameraFront.z;
    }
    if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS) {
        cameraPosition.x -= glm::normalize(glm::cross(cameraFront, cameraUp)).x
* cameraSpeed;
        cameraPosition.z -= glm::normalize(glm::cross(cameraFront, cameraUp)).z
* cameraSpeed;
    }
    if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS) {
        cameraPosition.x += glm::normalize(glm::cross(cameraFront, cameraUp)).x
* cameraSpeed;
        cameraPosition.z += glm::normalize(glm::cross(cameraFront, cameraUp)).z
* cameraSpeed;
    }
    if (glfwGetKey(window, GLFW_KEY_SPACE) == GLFW_PRESS) {
        startRotation();
    }
    if (glfwGetKey(window, GLFW_KEY_1) == GLFW_PRESS) {
        if (!pressed1) {

```

```

        SelectCard(1);

        pressed1 = true;
    }
}
else {
    pressed1 = false;
}
if (glfwGetKey(window, GLFW_KEY_2) == GLFW_PRESS) {
    if (!pressed2) {
        SelectCard(2);

        pressed2 = true;
    }
}
else {
    pressed2 = false;
}
if (glfwGetKey(window, GLFW_KEY_3) == GLFW_PRESS) {
    if (!pressed3) {
        SelectCard(3);

        pressed3 = true;
    }
}
else {
    pressed3 = false;
}
if (glfwGetKey(window, GLFW_KEY_4) == GLFW_PRESS) {
    if (!pressed4) {
        SelectCard(4);

        pressed4 = true;
    }
}
else {
    pressed4 = false;
}
if (glfwGetKey(window, GLFW_KEY_ENTER) == GLFW_PRESS) {
    if (!pressedEnter) {
        winCheck();
        pressedEnter = true;
    }
}
else {
    pressedEnter = false;
}
}

void test::CardGame::winCheck() {
    if ((cards[0].selected && cards[1].selected) && (!cards[2].selected &&
!cards[3].selected)) {
        std::cout << "YOU WIN" << std::endl;
    }
    else if ((!cards[0].selected && !cards[1].selected) && (cards[2].selected
&& cards[3].selected)){
        std::cout << "YOU WIN" << std::endl;
    }
    else {
        std::cout << "YOU LOSE" << std::endl;
    }
}
}

```

```

void test::CardGame::mouseCallback(GLFWwindow* window)
{
    double xpos, ypos;
    glfwGetCursorPos(window, &xpos, &ypos);
    if (firstMouse)
    {
        lastX = xpos;
        lastY = ypos;
        firstMouse = false;
    }

    float xoffset = xpos - lastX;
    float yoffset = lastY - ypos;
    lastX = xpos;
    lastY = ypos;

    float sensitivity = 0.1f;
    xoffset *= sensitivity;
    yoffset *= sensitivity;

    yaw += xoffset;
    pitch += yoffset;

    if (pitch > 89.0f)
        pitch = 89.0f;
    if (pitch < -89.0f)
        pitch = -89.0f;

    glm::vec3 direction;
    direction.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
    direction.y = sin(glm::radians(pitch));
    direction.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));
    cameraFront = glm::normalize(direction);
}

void test::CardGame::startRotation() {
    std::cout << "Starting rotation\n";
    angle = 0.0f;
    for (auto& card : cards) {
        card.Rotation(0); // Reset rotation
    }
    shuffCheck = false;
    rotationComplete = false;
}

void test::CardGame::shuffle() {
    std::cout << "Shuffling cards\n";
    std::mt19937 g(static_cast<unsigned int>(std::time(nullptr)));

    // Swap positions of cards to shuffle
    for (size_t i = 0; i < cards.size(); ++i) {
        std::uniform_int_distribution<> dis(0, cards.size() - 1);
        size_t j = dis(g);
        std::swap(cards[i].position, cards[j].position);
        cards[i].Rotation(180.f);
    }
}

void test::CardGame::SelectCard(int i)
{

```

```
int goal;
switch (i) {
case(1):
    goal = -4;
    break;
case(2):
    goal = -2;
    break;
case(3):
    goal = 0;
    break;
case(4):
    goal = 2;
    break;
}
for (int j = 0; j < 4; j++) {
    if (cards[j].getPos() == goal) {
        i = j;
    }
}
cards[i].select();
}
```

Assignment 2, Nathan Willis

GRADEMARK REPORT

FINAL GRADE

75/100

GENERAL COMMENTS

Comments:

Game Design (40 points)

Develop a game based on the matching pairs premise:
10/10 points

Use of OOP: 5/5 points

Include a mechanism to cover and uncover pairs of objects: 5/5 points

Implement object picking (keyboard, collision, or god solution, or another innovative approach): 5/5 points

Allow players to match objects; mismatched objects are covered again : 10/10 points

Maintain high scores: 0/5 point (local file storage or db solution)

Graphics and Rendering (40 points)

Use C++ with relevant APIs/SDKs:
5/5 points (OPENGGL/GLFW/GLEW/GLM/IMGUI)

Camera system: 5/5 points
(Orthographic/Projection)

Implement a lighting system: 0/5 points

Create and import appropriate textures: 5/5 points

Apply textures to primitives: 5/5 points

Import game objects using a consistent file format: 0/3 points

Implement depth testing: 5/5 points

Apply surface culling: 0/2 points

Batching: 0/5 points

Documentation/Report (20 points)

Write a report detailing the structure of the program using appropriate notation: 6/8 points

Explain the important algorithms and data structures used, justifying their use: 4/7 points

Evaluate the application by testing and validating it: 5/5 points

Alternatively a video discussing the same topics as the report while highlighting the key areas of

the code and operation of the system

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
