Computação Gráfica - Tarefa 2

Lucas Toscano Pimentel Appolinário Cerqueira - 2110695

Classe Disk implementada:

A classe Disk foi implementada para criar discos 2D com um número configurável de fatias. O método estático Make cria e retorna uma instância, enquanto o construtor gera os vértices, cores e índices necessários para desenhar o disco. As coordenadas dos vértices são calculadas com base no número de fatias e geradas a partir de coordenadas polares, e os buffers correspondentes são configurados e vinculados a um VAO.

A implementação inclui a configuração dos buffers de vértices, cores e índices. O método Draw utiliza os dados gerados para renderizar o disco. O código segue a estrutura especificada, atendendo aos requisitos da interface fornecida.

```cpp
#include "disk.h"

// Factory method to create a DiskPtr
DiskPtr Disk::Make (int nslice) {
    return DiskPtr(new Disk(nslice));
}

// Constructor for Disk
Disk::Disk(int nslice) {
    m_nslice = nslice;

    // Generate vertices
    float* xy = generateDiskVertices(m_nslice);

    // Generate colors vertices
    unsigned char rgb[(m_nslice + 1) * 3];
    for (int i = 0; i <= m_nslice; ++i) {
        rgb[i * 3] = 255;   // Red
        rgb[i * 3 + 1] = 255; // Green
        rgb[i * 3 + 2] = 255; // Blue
    }

    // Generate indices
    unsigned int inc[m_nslice + 2];
    inc[0] = 0; // The center vertex
    for (int i = 1; i <= m_nslice; ++i) {
        inc[i] = i;
    }
    inc[m_nslice + 1] = 1; // Repeat first perimeter vertex to close the triangle fan

    // Create VAO
    glGenVertexArrays(1, &m_vao);
    glBindVertexArray(m_vao);

    // Create buffers
    GLuint id[3];
    glGenBuffers(3, id);

    // Coordinate buffer
    glBindBuffer(GL_ARRAY_BUFFER, id[0]);
    glBufferData(GL_ARRAY_BUFFER, (m_nslice + 2) * 2 * sizeof(float), (void*)xy, GL_STATIC_DRAW);
    glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(0);

    // Free memory
```

```
    // Free memory
    delete[] xy;

    // Color buffer
    glBindBuffer(GL_ARRAY_BUFFER, id[1]);
    glBufferData(GL_ARRAY_BUFFER, (m_nslice + 1) * sizeof(unsigned char) * 3, (void*)rgb, GL_STATIC_DRAW);
    glVertexAttribPointer(1, 3, GL_UNSIGNED_BYTE, GL_TRUE, 0, 0);  // Colors
    glEnableVertexAttribArray(1);

    // Index buffer
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, id[2]);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, (m_nslice + 2) * sizeof(unsigned int), (void*)inc, GL_STATIC_DRAW);

    std::cout << "Disk created with " << m_nslice << " slices."<< std::endl;

    // Unbind the VAO
    glBindVertexArray(0);
}

// Destructor for Disk
Disk::~Disk () {
    glDeleteVertexArrays(1, &m_vao);
}

// Draw method
void Disk::Draw() {
    // Drawing function: bind the VAO and draw the disk
    glBindVertexArray(m_vao);

    glDrawArrays(GL_TRIANGLE_FAN, 0, m_nslice + 2);

    // Unbind the VAO
    glBindVertexArray(0);
}

// Generate vertices for the disk
float* Disk::generateDiskVertices(int nslice) {
    // Allocate memory for the vertices: 1 center vertex + nslice vertices + 1 repeat of the first perimeter vertex
    float* vertices = new float[(nslice + 2) * 2]; // Each vertex has 2 components (x, y)

    // The center of the disk
    vertices[0] = 0.0f; // x
    vertices[1] = 0.0f; // y

    // The angle increment between each slice
    float angleIncrement = 2.0f * glm::pi<float>() / nslice;

    // Loop to create vertices for the perimeter of the disk
    for (int i = 0; i < nslice; ++i) {
        // Calculate the angle for the current slice
        float angle = i * angleIncrement;

        // Use polar coordinates to calculate the position of each vertex
        float x = 1 * cos(angle);
        float y = 1 * sin(angle);

        // Add the vertex to the array
        vertices[(i + 1) * 2] = x;     // x component
        vertices[(i + 1) * 2 + 1] = y; // y component
    }

    // Add the first perimeter vertex again at the end to close the fan
    vertices[(nslice + 1) * 2] = vertices[2];       // x of first perimeter vertex
    vertices[(nslice + 1) * 2 + 1] = vertices[3];   // y of first perimeter vertex

    // // Print the vertices
    // std::cout << "Disk Vertices (x, y):" << std::endl;
    // for (int i = 0; i <= nslice + 1; ++i) {
    //     std::cout << "Vertex " << i << ": ("
    //               << vertices[i * 2] << ", "
    //               << vertices[i * 2 + 1] << ")"
    //               << std::endl;
    // }

    return vertices; // Return the dynamically allocated array
}
```

Na main, as funcões initialize e display.

```
18  static void initialize()
19  {
20    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
21
22    shd = Shader::Make();
23    shd->AttachVertexShader("../shaders/vertex.glsl");
24    std::cout << "main.cpp: got here -  initialize()" << std::endl;
25    shd->AttachFragmentShader("../shaders/fragment.glsl");
26    shd->Link();
27
28    diskEarth = Disk::Make(64);
29    diskSun = Disk::Make(64);
30
31
32    Error::Check("initialize");
33  }
34
35
36  static void display(GLFWwindow * win)
37  {
38      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
39      shd->UseProgram();
40
41      float time = (float)glfwGetTime();
42      float angle = time * 0.5f;
43
44      unsigned int transformLoc = glGetUniformLocation(shd->GetProgram(), "M");
45      unsigned int colorLoc = glGetUniformLocation(shd->GetProgram(), "objectColor");
46      glm::mat4 trans = glm::mat4(1.0f);
47
48      // Earth
49      trans = glm::translate(trans, glm::vec3(0.6f, 0.0f, 0.0f));
50      trans = glm::rotate(glm::mat4(1.0f), angle, glm::vec3(0.0f, 0.0f, 1.0f)) * trans;
51      trans = glm::scale(trans, glm::vec3(0.1f, 0.1f, 0.0f));
52      glm::vec4 earthColor = glm::vec4(0.0f, 0.0f, 1.0f, 1.0f);
53      glUniform4fv(colorLoc, 1, glm::value_ptr(earthColor));
54      glUniformMatrix4fv(transformLoc, 1, GL_FALSE, glm::value_ptr(trans));
55      diskEarth->Draw();
56
57      // Sun
58      trans = glm::mat4(1.0f);
59      trans = glm::scale(trans, glm::vec3(0.3f, 0.3f, 0.0f));
60      glm::vec4 sunColor = glm::vec4(1.0f, 1.0f, 0.0f, 1.0f);
61      glUniform4fv(colorLoc, 1, glm::value_ptr(sunColor));
62      glUniformMatrix4fv(transformLoc, 1, GL_FALSE, glm::value_ptr(trans));
63      diskSun->Draw();
64
65      Error::Check("display");
66  }
67
```

Os dois discos, representando a Terra e o Sol, são inicializados na função initialize, onde os shaders de vértice e fragmento são carregados e vinculados, e ambos os discos são criados com 64 fatias.

Na função display, o disco da Terra é transladado para a direita e rotacionado com base no tempo obtido da função glfwGetTime, de modo que sua rotação varia a cada renderização. Além disso, o disco é escalado para reduzir seu tamanho e recebe uma cor azul. Já o disco do Sol é escalado para um tamanho maior e é colorido de amarelo. Ambas as transformações e cores são aplicadas antes de desenhar cada disco.