BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC SƯ PHẠM THÀNH PHỐ HỒ CHÍ MINH

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CÁ NHÂN

Nhóm: Game và những con nghiện

ĐỀ TÀI:

CỜ VUA 3D

MÔN ĐỒ HỌA MÁY TÍNH

Giảng viên hướng dẫn: Nguyễn Đỗ Thái Nguyên

Sinh viên thực hiện: Huỳnh Thanh Phong - 4501104172

Thành phố Hồ Chí Minh, 12/2021

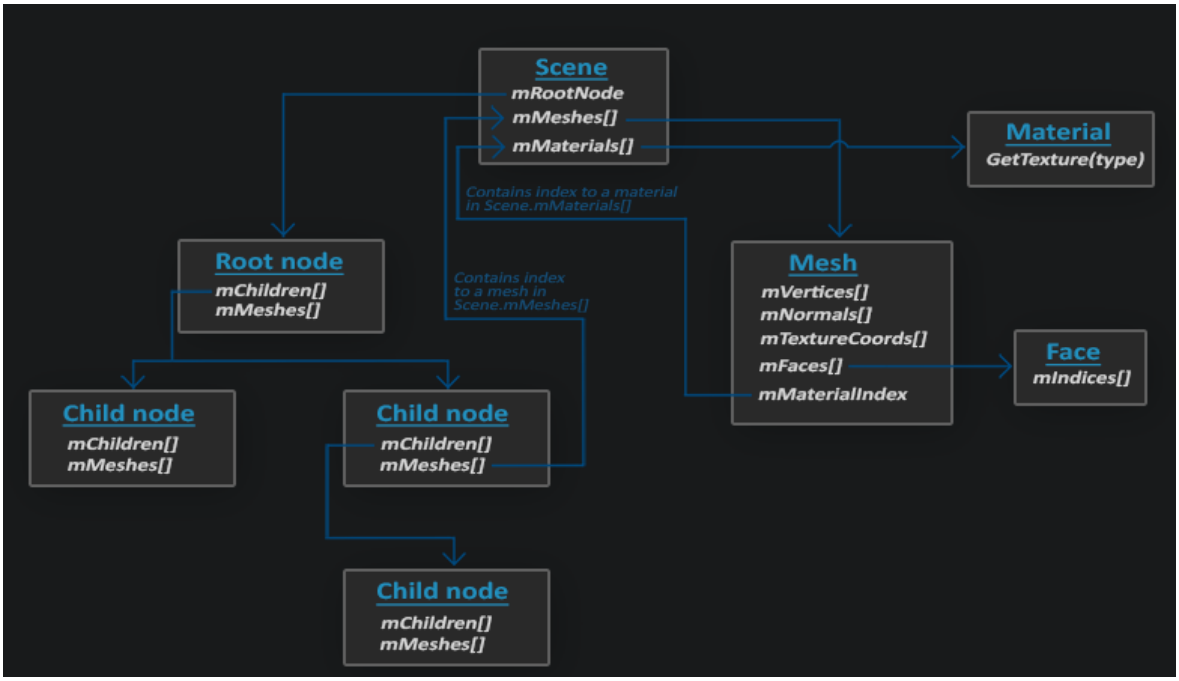
Mục lục

CHƯƠNG 1 :Assimp

1.1. Giới thiệu

Open Asset Import Library (Assimp) là một thư viện nổi tiếng cho việc load model 3D được viết bằng C++. Khi mà các model hay object chúng ta tạo ra ngày càng phức tạp hơn thông qua các tool vẽ 3D như Blender, 3DS Max và Maya thì việc load các model 3D là việc cần thiết. Và với thư viện assimp chúng ta có thể load dễ dàng nhiều model 3D phức tạp cùng một lúc với nhiều dạng file khác nhau. Assimp sẽ đọc các dữ liệu của model và đẩy nó vào cấu trúc dữ liệu của assimp. Và vì cấu trúc dữ liệu assimp không thay đổi ta có thể lấy dữ liệu của các model từ đó và sử dụng.

1.2. Cấu trúc dữ liệu của Assimp



Bao gồm:

- Các dữ liệu của model sẽ được lưu ở ô *Scene* như materials và mesh. Cũng như lưu đường liên kết tới ô root node

- Ô *Root node* lưu giữ đường liên kết tới các *Child node* của nó và bản thân lưu giữ thứ tự vẽ các dữ liệu ở bên trong mMeshes của ô *Scene* và các *Child node* cũng như vậy
- Ô *Mesh* dùng để chứa các dữ liệu cần thiết để có về vẽ lên cửa sổ như vị trí các điểm, vector, texture, faces và material
- Và ô *Mesh* có chứa nhiều faces nên cần một ô *Face* để có thể lưu giữ các vị trí của từng face
- Cuối cùng là ô *Material* dùng để giữ các hàm để có thể lấy texture và material từ object

Thông qua cấu trúc này, assimp sẽ tìm cách load model 3D vào ô *Scence* sau đó phân nhánh các mesh object cho từng node và mỗi mesh của từng ô sẽ chứa dữ liệu điểm, vị trí và material. Và sau cùng khi chúng ta tổng hợp lại tất cả mesh data chúng ta sẽ có một model 3D hoàn chỉnh

CHƯƠNG 2 :Mesh

2.1. Định nghĩa

Khi mà vẽ hoặc tạo model trên các tool như blender, developer thường tạo từng dạng của model rồi sau đó ghép tất cả lại thành một model hoàn chỉnh. Thường chúng ta sẽ gọi từng dạng model đó là mesh. Mỗi mesh sẽ chứa một phần nhỏ dữ liệu của để tạo nên một model hoàn chỉnh

2.2. Tạo mesh class cho OpenGL

Khi chúng ta lấy dữ liệu từ cấu trúc dữ liệu của assimp thì chúng ta cần phải có nơi lưu trữ nên chúng ta cần tạo mesh class của riêng mình

Chúng ta sẽ cần tạo struct cho *Vertex* và *Texture* trước dùng để chứa điểm và texture:

```
struct Vertex {
    // position
    glm::vec3 Position;
    // normal
    glm::vec3 Normal;
    // texCoords
    glm::vec2 TexCoords;
    // tangent
    glm::vec3 Tangent;
    // bitangent
    glm::vec3 Bitangent;
    // bone indexes which will influence this vertex
    int m_BoneIDs[MAX_BONE_INFLUENCE];
    // weights from each bone
    float m_Weights[MAX_BONE_INFLUENCE];
};

struct Texture {
    unsigned int id;
    string type;
    string path;
};
```

Sau đó chúng ta sẽ tạo class *Mesh* dùng để chứa vertex và texure ở trên:

```
Mesh(vector<Vertex> vertices, vector<unsigned int> indices, vector<Texture>
textures) {
	this->vertices = vertices;
	this->indices = indices;
	this->textures = textures;

	// now that we have all the required data, set the vertex buffers and its attribute
	// pointers.
	setupMesh();
}
```

Tiếp theo là tạo hàm *setupMesh()* dùng để khởi tạo các buffer và đưa các dữ liệu vào buffer như VBO,EBO và VAO:

```
void setupMesh() {
	// create buffers/arrays
	glGenVertexArrays(1, &VAO);
	glGenBuffers(1, &VBO);
	glGenBuffers(1, &EBO);

	glBindVertexArray(VAO);
	// load data into vertex buffers
	glBindBuffer(GL_ARRAY_BUFFER, VBO);
	// A great thing about structs is that their memory layout is sequential for all its
items.
	// The effect is that we can simply pass a pointer to the struct and it translates
perfectly
	// to a glm::vec3/2 array which again translates to 3/2 floats which translates to a
byte
	// array.
	glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(Vertex), &vertices[0],
		GL_STATIC_DRAW);

	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
	glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(unsigned int),
&indices[0],
		GL_STATIC_DRAW);

	// set the vertex attribute pointers
	// vertex Positions
	glEnableVertexAttribArray(0);
	glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void *)0);
	// vertex normals
	glEnableVertexAttribArray(1);
```

```
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex),
                              (void *)offsetof(Vertex, Normal));
        // vertex texture coords
        glEnableVertexAttribArray(2);
        glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex),
                              (void *)offsetof(Vertex, TexCoords));
        // vertex tangent
        glEnableVertexAttribArray(3);
        glVertexAttribPointer(3, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex),
                              (void *)offsetof(Vertex, Tangent));
        // vertex bitangent
        glEnableVertexAttribArray(4);
        glVertexAttribPointer(4, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex),
                              (void *)offsetof(Vertex, Bitangent));
        // ids
        glEnableVertexAttribArray(5);
        glVertexAttribIPointer(5, 4, GL_INT, sizeof(Vertex), (void *)offsetof(Vertex,
m_BoneIDs));

        // weights
        glEnableVertexAttribArray(6);
        glVertexAttribPointer(6, 4, GL_FLOAT, GL_FALSE, sizeof(Vertex),
                              (void *)offsetof(Vertex, m_Weights));
        glBindVertexArray(0);
}
```

Sau khi khởi tạo xong các buffer thì chúng ta cần tạo hàm *Draw()* để có thể vẽ các model lên màn hình:

```
void Draw(Shader *shader) {
    // bind appropriate textures
    unsigned int diffuseNr = 1;
    unsigned int specularNr = 1;
    unsigned int normalNr = 1;
    unsigned int heightNr = 1;
    for (unsigned int i = 0; i < textures.size(); i++) {
        glActiveTexture(GL_TEXTURE0 + i);  // active proper texture unit before binding
        // retrieve texture number (the N in diffuse_textureN)
        string number;
        string name = textures[i].type;
        if (name == "texture_diffuse")
            number = std::to_string(diffuseNr++);
        else if (name == "texture_specular")
```

```
            number = std::to_string(specularNr++);  // transfer unsigned int to string
        else if (name == "texture_normal")
            number = std::to_string(normalNr++);  // transfer unsigned int to string
        else if (name == "texture_height")
            number = std::to_string(heightNr++);  // transfer unsigned int to string

        // now set the sampler to the correct texture unit
        glUniform1i(glGetUniformLocation(shader->ID, (name + number).c_str()), i);
        // and finally bind the texture
        glBindTexture(GL_TEXTURE_2D, textures[i].id);
    }

    // draw mesh
    glBindVertexArray(VAO);
    glDrawElements(GL_TRIANGLES, indices.size(), GL_UNSIGNED_INT, 0);
    glBindVertexArray(0);

    // always good practice to set everything back to defaults once configured.
    glActiveTexture(GL_TEXTURE0);
}
```

CHƯƠNG 3 :Model

Sau khi đã tạo xong class *Mesh* ở trên ta phải tiếp tục tạo class *Model* vì như đã nói ở chương 1 mỗi mesh tượng trưng 1 phần của model:

```
class Model {
public:
    // model data
    vector<Texture> textures_loaded;  // stores all the textures loaded so far, optimization to make
                                      // sure textures aren't loaded more than once.
    vector<Mesh> meshes;
    string directory;
    bool gammaCorrection;
    glm::vec3 position;

    // constructor, expects a filepath to a 3D model.
    Model(){};
    Model(string const &path, bool gamma = false) : gammaCorrection(gamma) { loadModel(path); }

    // draws the model, and thus all its meshes
    void Draw(Shader *shader) {
        for (unsigned int i = 0; i < meshes.size(); i++) meshes[i].Draw(shader);
    }
};
```

Sau khi đã tạo xong class *Model* chúng ta đã có thể bắt đầu sử dụng assimp để load model 3D và đưa được nó lên cửa sổ và sáng tạo theo cách mình muốn. Như thế này:

CHƯƠNG 4 :Tài liệu tham khảo

- Joey de Vries(2014), Learn OpenGL, https://learnopengl.com
- OGLDEV, Loading models in OpenGL using Assimp, https://www.youtube.com/watch?v=sP_kiODC25Q&t=844s&ab_channel=OGLDEV