I) Giới Thiệu

1) Giới thiệu về OpenGL

a) OpenGL là gì?

OpenGL (Open Graphics Library) là một API đồ họa mạnh mẽ, được sử dụng để lập trình và hiển thị đồ họa 2D và 3D trên nhiều nền tảng khác nhau.

Được phát triển bởi Silicon Graphics vào những năm 1990 và hiện nay được quản lý bởi Khronos Group, OpenGL cung cấp một giao diện tiêu chuẩn để giao tiếp với phần cứng đồ họa, giúp các lập trình viên có thể tận dụng tối đa hiệu suất của GPU (Graphics Processing Unit).

Không giống như các engine đồ họa sẵn có như Unity hoặc Unreal Engine, OpenGL là một API cấp thấp, cung cấp quyền kiểm soát trực tiếp đối với quy trình render. Điều này giúp các lập trình viên có thể tối ưu hiệu suất và tạo ra các engine đồ họa tùy chỉnh, như một **engine voxel 3D** mà dự án này sẽ được triển khai.

b) Vai trò của OpenGL trong lập trình đồ họa

OpenGL đóng vai trò như một cầu nối giữa phần mềm và phần cứng đồ họa. Khi một chương trình cần vẽ hình ảnh lên màn hình, nó sẽ sử dụng các lệnh OpenGL để gửi dữ liệu đến GPU, nơi các phép toán xử lý đồ họa được thực hiện nhanh chóng và hiệu quả.

Một số nhiệm vụ chính của OpenGL trong lập trình đồ họa bao gồm:

- **Xử lý hình học (Geometry Processing):** Chuyển đội tọa độ, ánh xạ, texture, và các phép toán biến đổi đối tượng.

- **Render đối tượng 2D/3D:** Vẽ điểm, đường, tam giác, hoặc các đối tượng phức tạp hơn bằng cách sử dụng các buffer và shader.

- **Xử lý ánh sáng và hiệu ứng:** Mô phỏng ánh sáng, bóng đổ, phản xạ, và các hiệu ứng đặc biệt khác.

- **Tối ưu hóa hiệu suất:** Sử dụng **VBO (Vertex Buffer Object)**, **VAO (Vertex Array Object)**, **Instancing**, và các kỹ thuật khác để cải thiện tốc độ vẽ.

c) Các thành phần chính của OpenGL

Để hiểu cách OpenGL hoạt động, cần nắm rõ các thành phần quan trọng sau:

- **Pipeline đồ họa của OpenGL:** Là quá trình tuần tự từ khi nhận dữ liệu đầu vào (tọa độ, texture) đến khi hiển thị hình ảnh trên màn hình. **Pipeline** này bao gồm các giai đoạn như **Vertex Processing**, **Fragment Processing** và **Rasterization**.

- **Shader và GLSL:** OpenGL sử dụng **shader** – các chương trình nhỏ chạy trên GPU để xử lý từng điểm ảnh và đỉnh. Ngôn ngữ lập trình shader của OpenGL là **GLSL (OpenGL Shading Language)**.

- **Buffer và Texture:** OpenGL lưu trữ dữ liệu đồ họa trong bộ nhớ của GPU thông qua các buffer như **VBO**, **EBO**, và sử dụng **Texture Mapping** để hiể thị ảnh chi tiết hơn.

d) Vì sao OpenGL được sử dụng để tạo **engine voxel**.

OpenGL là một công cụ lý tưởng để phát triển engine voxel 3D, vì:

- **Hỗ trợ nền tảng đa dạng:** OpenGL có thể chạy trên **Windows**, **Linux**, **maxOS**, và nhiều thiết bị khác.

- **Cung cấp quyền kiểm soát thấp nhất:** Không giống **Unity** hay **Unreal**, OpenGL cho phép lập trình viên điều chỉnh từng bước trong quá trình render.

- **Tối ưu hóa hiệu suất:** Với các kỹ thuật như **Frustum Culling**, **Instancing**, **Compute Shader**, OpenGL có thể hiện thị hàng triệu voxel một cách hiệu quả.

2) Voxel là gì?

Voxel (Volumetric Pixel – Pixel thể tích) là đơn vị cơ bản để biểu diễn không gian 3D, tương tự như **pixel** trong đồ họa 2D.

Một voxel là một khối nhỏ trong không gian 3D, chứa thông tin về màu sắc, mật độ hoặc vật liệu của đối tượng.

Voxel không lưu trữ hình dạng bề mặt trực tiếp như **mesh (lưới tam giác)**, mà lưu trữ dữ liệu dưới dạng một thể tích lưới.

Một voxel – khối lập phương có thể được mô phỏng thành các khối chất lỏng, khói, chất rắn…vv.

Các game nổi tiếng sử dụng mô hình thế giới voxel như :

- Minecraft: Mô hình voxel đơn giản để tạo thế giới sandbox.
- Teardown: Game phá hủy môi trường bằng voxel.
- 7 Days to Die: Kết hợp voxel để tạo môi trường có thể thay đổi.

3) Tổng quan về Engine Voxel 3D

Engine Voxel 3D là một hệ thống phần mềm giúp tạo, quản lý và hiển thị thế giới voxel trong không gian 3D.

Không giống như các engine 3D truyền thống dựa trên lưới tam giác(mesh), engine voxel sử dụng các khối voxel để biểu diễn không gian và vật thể.

4) Mục tiêu và phạm vi của đề tài

Các chức năng chính của Engine Voxel 3D:

- Tạo và quản lý voxel:
    - Lưu trữ voxel trong cấu trúc dữ liệu phù hợp(mảng 3D, Octree, Sparse, Voxel Grid).
    - Hỗ trợ các kích thước voxel khác nhau để tối ưu hóa độ chi tiết.
- Render voxel bằng OpenGL:
    - Vẽ voxel bằng cách khối (cubes) hoặc Marchinh Cubes(biến voxel thành mesh mượt mà hơn).
    - Sử dụng VBO, VAO, EBO để tối ưu hiển thị.
    - Tạo shader để xử lý ánh sáng và màu sắc cho voxel.
- Quản lý Chunk (khối voxel lớn):
    - Chunking system giúp chia nhỏ thế giới thành các vùng để giảm tải bộ nhớ.
    - Load và unload chunk dựa trên vị trí của người chơi.

- Culling và LOD (Level of Detail):
    - Frustum culling: Chỉ vẽ những voxel nằm trong tầm nhìn camera.
    - Occlusion Culling: Ẩn các voxel bị che khuất để tối ưu hóa hiệu suất.
    - LOD (Level of Detail): Giảm số lượng voxel hiển thị ở xa để tăng hiệu suất.
- Xử lý ánh sáng và bóng đổ:
    - Áp dụng Global Illumination đơn giản bằng Light Propagation Volumes.
    - Dùng voxel Cone Tracing để mô phỏng ánh sáng mềm mại.
- Tương tác vật lý và mô phỏng:
    - Hỗ trợ phá hủy voxel (Destrucible Enviroments).
    - Sử dụng thuật toán AABB (Axis-Aligned Bounding Box) để phát hiện va chạm.
    - Áp dụng mô phỏng vật lý cho vật thể voxel rơi, nổ hoặc thay đổi hình dạng.
- Thế giới mở và tối ưu hóa lớn.
    - Perlin Noise / Simplex Noise để tạo địa hình tự động.
    - Sinh rừng, núi, hang động một cách tự nhiên.
- Hệ thống lưu trữ và tải dữ liệu:
    - Nén voxel để tối ưu lưu trữ.
    - Lưu và giải thế giới từ file hoặc database.

II) Kiến Thức Cơ Bản Về OpenGL

1 Nguyên tắc hoạt động của OpenGL

OpenGL (Open Graphics Library) là một API đồ họa đa nền tảng mã nguồn mở, được sử dụng để lập tình đồ họa 2D và 3D.

OpenGL cung cấp một tập hợp các hàm để tương tác với GPU, giúp lập trình viên vẽ hình ảnh bằng cách sử dụng các tọa độ, đỉnh (vertices), texture và shader.

OpenGL hoạt động theo mô hình pipeline đồ họa, nơi dữ liệu đầu vào được xử lý qua nhiều giai đoạn để tạo ra hình ảnh cuối cùng trên màn hình.

2 Pipeline đồ họa trong OpenGL

OpenGL hoạt động dựa trên nguyên lý pipeline đồ hoạt, bao gồm nhiều giai đoạn xử lý tuần tự:

a Nhập dữ liệu (Input Data)

Các tọa độ đỉnh (vertices), chỉ số (indices) được gửi vào GPU thông qua VBO (Vertex Buffer Object) và EBO (Element Buffer Object).

Các dữ liệu khác như màu sắc, texture cũng được tải vào bộ nhớ GPU.

b Xử lý đỉnh (Vertex Processing – Vertex Shader)

Mỗi đỉnh được xử lý thông qua vertex shader để xác định vị trí sau khi biến đổi (tranform).

Các phép biến đổi phổ biến:

- Mô hình (model Transformation) – thay đổi vị trí, góc quay, kích thước của vật thể.
- Nhìn (View Transformation) – chuyển từ tọa độ thế giới sang tọa độ camera.
- Chiếu (Projection Transformation) – chuyển từ không gian camera sang tọa độ màn hình (phối cảnh hoặc trực giao).

c Lập tam giác và Rasterzation

Các đỉnh được kết nối để tạo hình tam giác (primitives) (GL_TRANGLES, GL_LINES.. vv)

OpenGL chuyển đổi hình tam giác thành pixel thông qua quá trình rasterzation.

d Xử lý Fragment (Fragment Shader)

Fragment Shader xác định màu sắc của từng pixel bằng cách:

- Lấy màu từ texture (Sample Texture).
- Tính toán ánh sáng (Lighting Calculation)
- Xử lý hiệu ứng đặc biệt (Bump Mapping, Shadow Mapping..vv)

e Kiểm tra và hiển thị (Output Merging and Rendering)

OpenGL áp dụng các kỹ thuật Depth Test, Stencil Test, Alpha Blending để đảm bảo hình ảnh hiển thị đúng thứ tự.

Sau khi hoàn tất xử lý, hình ảnh được vẽ lên màn hình qua Framebuffer.

3 Shader và GLSL

a Shader là gì?

Shader là các chương trình chạy trên GPU, giúp thực hiện các phép toán đồ họa để xử lý hiển thị hình ảnh.

Shader trong OpenGL được viết bằng GLSL (OpenGL Shading Language), một ngôn ngữ lập trình chuyên dụng tối ưu cho GPU.

Shader hoạt động theo mô hình song song, nghĩa là GPU có thể xử lý hàng nghìn pixel cùng lúc để đạt hiệu suất cao.

b Các loại Shader trong OpenGL

OpenGL có nhiều loại Shader, mỗi loại phục vụ một mục đích riêng biệt trong pipeline đồ họa:

c Vertex Shader

- Xử lý tọa độ đỉnh và thực hiện các phép toán biến đổi không gian.
- Tính toán thuộc tính như màu sắc, texture coordinates, lighting cho từng đỉnh.
- Chạy một lần cho mỗi đỉnh trong mô hình 3D.

i Ví dụ Vertex Shader trong GLSL:

```
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec2 textCoords;
layout (location = 2) in int id;

out vec2 TexCoord;
out vec3 cube_color;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

vec3 hash31(float p) {
    vec3 p3 = fract(vec3(p * 21.2f) * vec3(0.1031f, 0.1030f, 0.0973f));
    p3 += dot(p3, p3.yzx + 33.33f);
    return fract((p3.xxy + p3.yzz) * p3.zyx) + 0.05f;
}

void main()
{
    gl_Position = projection * view * model * vec4(position, 1.0f);
    TexCoord = textCoords;
    cube_color = hash31(id);
}
```

ii Fragment Shader (Xử lý pixel):

Xác định màu sắc của từng pixel trong mô hình.

Lấy mẫu từ texture, tính toán ánh sáng, bóng đổ, hiệu ứng đổ bóng.

Chạy một lần cho mỗi pixel trong tam giác được vẽ lên màn hình.

d Ví dụ về Fragment Shader trong GLSL:

```
#version 330 core
out vec4 FragColor; // Màu đầu ra của pixel

in vec2 TexCoords; // Nhận từ Vertex Shader
uniform sampler2D texture1; // Texture

void main() {
    FragColor = texture(texture1, TexCoords); // Lấy màu từ texture
}
```

e Các loại Shader khác

Ngoài 2 loại Shader trên, GPU cho phép người lập trình can thiệp và tùy biến một vài giai đoạn khác trong quá trình xử lý tọa độ cũng như triết xuất pixel của các điểm ảnh. Các shader có thể kể đến như:

- Geometry Shader (Tùy chọn, tạo hình học mới)
- Tesselation Shader (Tùy chọn, tạo lưới mịn hơn)

4 Kết xuất hình học và ánh sáng

III) Tổng kết

Tìm hiểu sơ bộ về kiến thức cơ bản khi làm việc với OpenGL