2025 ComputerGraphics Hw1

학과	스마트ICT융합공학과
학번	202012336
이름	김동규

1. Objective

-Environment

IDE : VisualStudio2022

Language : C++

Library : GL , GLFW, GLM (etc)

-Objective

Render a scene using the ray tracing algorithm with 2D spheres.

Intersection points: Displayed as white spheres

Non-intersection areas: Displayed in black

2.Code Explanation

-Class Definition

Ray: Defines the center (starting point) and direction vector of the ray used in ray tracing. The direction vector is normalized.

```
class Ray {
public:
    vec3 origin, direction;
    Ray(const vec3& o, const vec3& d) : origin(o), direction(normalize(d)) {}
};
```

Surface: A parent class representing objects in space.

```
class Surface {
public:
    vec3 color;
    Surface(const vec3& c) : color(c) {}
    virtual bool intersect(const Ray& ray, float& t, float tMin, float tMax) const = 0;
};
```

Plane :

*A class representing Plane in space, inheriting from Surface.

*if normal and ray direction are orthogonal, it doesn't intersect

```
class Plane : public Surface {
public:
    vec3 normal;
    float d;

    Plane(const vec3& n, float d, const vec3& col)
        : Surface(col), normal(normalize(n)), d(d) {
    }

    bool intersect(const Ray& ray, float& t, float tMin, float tMax) const override {
        float denom = dot(normal, ray.direction);
        if (abs(denom) > 1e-6) { // Ensure the ray is not parallel to the plane
            t = (d - dot(normal, ray.origin)) / denom;
            return (t >= tMin && t <= tMax);
        }
        return false;
    }
};
```

Sphere:

*A class representing spheres in space, inheriting from Surface.

Defines the center, color, and radius of the sphere.

*Implements the intersect function to determine whether a ray intersects the sphere.

*The intersection condition is based on the discriminant: $b^2 - a \cdot c$ (where a is 1 and omitted).

*If the intersection value t is between tmin and tmax, it is returned as an output parameter.

```
class Sphere : public Surface {
public:
    vec3 center;
    float radius;
    Sphere(const vec3& c, float r, const vec3& col)
        : Surface(col), center(c), radius(r) {
    }

    bool intersect(const Ray& ray, float& t, float tMin, float tMax) const override {
        vec3 oc = ray.origin - center;
        float a = dot(ray.direction, ray.direction);
        float b = dot(oc, ray.direction);
        float c = dot(oc, oc) - radius * radius;
        float discriminant = b * b - c;//a is 1

        if (discriminant > 0) {
            float temp = (-b - std::sqrt(discriminant)) / a;
            if (temp < tMax && temp > tMin) {
                t = temp;
                return true;
            }
        }
        return false;
    }
};
```

Camera:

*Defines the camera's position and viewing direction.

*Implements the getRay function to return a ray in the camera's viewing direction.

*The Camera and Ray are kept separate for modularity.

```
class Camera {
public:
    vec3 position, direction;
    Camera(const vec3& pos, const vec3& dir)
        : position(pos), direction(normalize(dir)) {
    }

    Ray Sy(float x, float y) {
        return Ray(position, normalize(vec3(x, y, -0.1f))); // image plane is at -0.1
    }
};
```

Scene:

*A class that manages the Camera and Surface objects.

*Uses the trace function to find intersections between rays and objects in the scene.

*If an intersection occurs, it updates t only when a new intersection (tempT) is closer than the previous t.

*This ensures that, when multiple objects intersect the same ray, only the nearest object's color is rendered.

*If no intersections exist, the color defaults to black.

```
class Scene {
public:
    std::vector<Surface*> surfaces;
    Camera camera;

    Scene(const std::vector<Surface*>& sur, const Camera& c)
        : surfaces(sur), camera(c) {
    }

    vec3 trace(const Ray& ray, float tMin, float tMax) {
        float t = tMax;
        const Surface* hitSurface = nullptr;

        for (const auto& surface : surfaces) {
            float tempT;
            if (surface->intersect(ray, tempT, tMin, tMax) && tempT < t) {
                t = tempT;
                hitSurface = surface;
            }
        }
        return (hitSurface != nullptr) ? hitSurface->color : vec3(0.0f, 0.0f, 0.0f); //if null -> black color
    }
};
```

Image:

*A separate image class is used to simplify color output management when rendering.

```
class Image {
public:
    void set(int x, int y, vec3 color) {
        OutputImage[(y * Width + x) * 3] = color.x;
        OutputImage[(y * Width + x) * 3 + 1] = color.y;
        OutputImage[(y * Width + x) * 3 + 2] = color.z;
    }
};
```

-Function Definition

render()

*Initializes outputImage and defines Sphere, Camera, Surface, Scene, and Image objects.

*Iterates through each pixel and traces rays to determine color values.

position of ray apply this equation

$$u = l + (r - l)(i + 0.5)/n_x$$
$$v = b + (t - b)(j + 0.5)/n_y$$

```
void render() {
    OutputImage.resize(Width * Height * 3, 1.0f);
    vec3 wcolor = vec3(1.0f, 1.0f, 1.0f); // sphere color
    vec3 pcolor = vec3(0.5f, 0.5f, 0.5f); // plane color

    Sphere sphere1(vec3(-4.0f, 0.0f, -7.0f), 1.0f, wcolor); // sphere1 define
    Sphere sphere2(vec3(0.0f, 0.0f, -7.0f), 2.0f, wcolor); // sphere2 define
    Sphere sphere3(vec3(4.0f, 0.0f, -7.0f), 1.0f, wcolor); // sphere3 define
    Plane plane(vec3(0.0f, 1.0f, 0.0f), -2.0f, pcolor); // plane define

    Camera camera(vec3(0.0f, 0.0f, 0.0f), vec3(0.0f, 0.0f, -1.0f));
    std::vector<Surface*> surfaces = { &sphere1, &sphere2, &sphere3, &plane };
    Scene scene(surfaces, camera);
    Image image;

    for (int iy = 0; iy < Height; ++iy) {
        for (int ix = 0; ix < Width; ++ix) {
            float x = (0.2f * (ix + 0.5f)) / Width - 0.1f;
            float y = (0.2f * (iy + 0.5f)) / Height - 0.1f;
            Ray ray = scene.camera.getRay(x, y);
            vec3 color = scene.trace(ray, 0.0f, FLT_MAX);
            image.set(ix, iy, color);
        }
    }
```

-Global Variables

```
// ------------------------------------------------
// Global Variables
// ------------------------------------------------
int Width = 512;
int Height = 512;
std::vector<float> OutputImage;
// ------------------------------------------------
```

x: Defines the width of the screen (resolution).
y: Defines the height of the screen (resolution).
outputImage: Stores the rendered image.

Additional Library
Added the #include <limits> library to define tmax.

3. Result screenshots