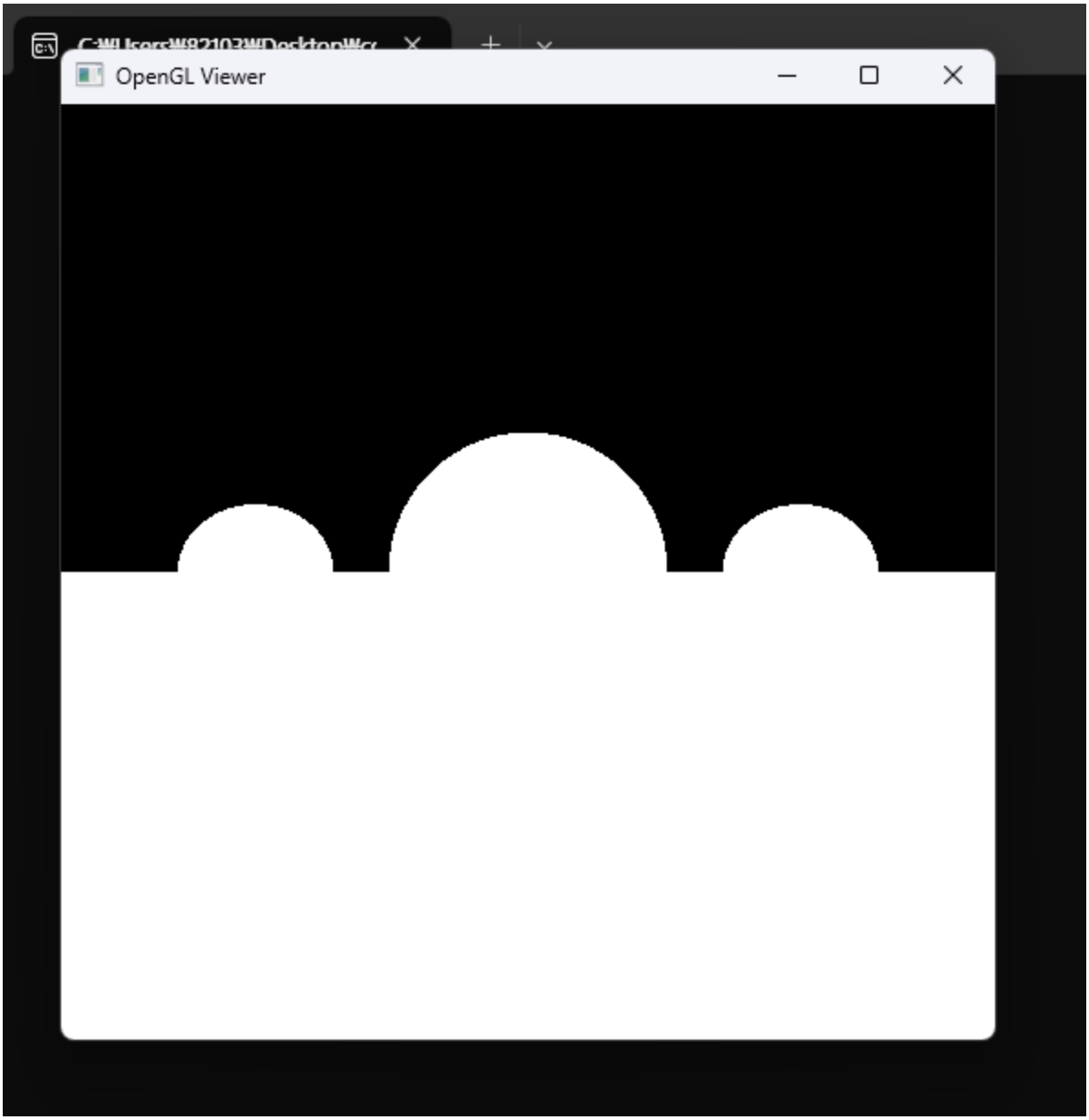
202112336 송수민 CG HW1 결과화면 + 레포트

결과화면



레포트

기존 코드에 github 코파일럿의 도움을 받아 코드 작성함.
변경 추가된 부분만 언급

- 결과화면 전역변수로 고정.

```
int Width = 512;
int Height = 512;
```

- Ray 클래스 작성. 시작점과 방향 선언

```
// class Ray
class Ray {
public:
        vec3 origin;
        vec3 direction;
};
```

- Surface 클래스 작성. 가상 부울 변수로 추후에 도형마다 정의해줌.

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

- Sphere 클래스 작성. 판별식 이용해 만나면 intersect가 참 반환.

```
// make Sphere using discriminant
class Sphere : public Surface {
public:
        vec3 center;
        float radius;

        Sphere(const vec3& c, float r) : center(c), radius(r) {}

        bool intersect(const Ray& ray, float tMin, float tMax, float& t) const override {
                vec3 oc = ray.origin - center;
                float a = dot(ray.direction, ray.direction);
                float b = 2.0f * dot(oc, ray.direction);
                float c = dot(oc, oc) - radius * radius;
                float discriminant = b * b - 4 * a * c;
                if (discriminant > 0) {
                        float temp = (-b - std::sqrt(discriminant)) / (2.0f * a);
                        if (temp < tMax && temp > tMin) {
                                t = temp;
                                return true;
                        }
                        temp = (-b + std::sqrt(discriminant)) / (2.0f * a);
                        if (temp < tMax && temp > tMin) {
```

```
                        t = temp;
                        return true;
                }
        }
        return false;
    }
};
```

- Plain 클래서 작성. 한 점과 법선벡터를 받음. 내적이 0이면 평행하므로 false반환.

```
// class Plane
class Plane : public Surface {
public:
    vec3 point;
    vec3 normal;

    Plane(const vec3& p, const vec3& n) : point(p), normal(normalize(n)) {}

    bool intersect(const Ray& ray, float tMin, float tMax, float& t) const override {
        float denom = dot(normal, ray.direction);
        if (abs(denom) > 1e-6) {
            vec3 p0l0 = point - ray.origin;
            t = dot(p0l0, normal) / denom;
            if (t >= tMin && t <= tMax) {
                return true;
            }
        }
        return false;
    }
};
```

- 과제 조건에 맞게 카메라 클래스 작성.

```
// set Camera and get Ray
class Camera {
public:
    vec3 eye;
    vec3 u, v, w;
    float l, r, b, t, d;

    Camera() {
        eye = vec3(0.0f, 0.0f, 0.0f);
        u = vec3(1.0f, 0.0f, 0.0f);
        v = vec3(0.0f, 1.0f, 0.0f);
        w = vec3(0.0f, 0.0f, 1.0f);
        l = -0.1f;
        r = 0.1f;
        b = -0.1f;
        t = 0.1f;
        d = 0.1f;
    }
```

```
        Ray getRay(float i, float j) const {
                float u_coord = l + (r - l) * (i + 0.5f) / Width;
                float v_coord = b + (t - b) * (j + 0.5f) / Height;
                vec3 direction = normalize(u_coord * u + v_coord * v - d * w);
                return Ray{ eye, direction };
        }
};
```

- Scene 클래스를 통해 ray와 물체가 만났다면 흰색, 아니면 검은색 반환.

```
// class Scene if hit object return white else return black
class Scene {
public:
        std::vector<Surface*> surfaces;

        vec3 trace(const Ray& ray, float tMin, float tMax) const {
                float closest_t = tMax;
                const Surface* hit_surface = nullptr;
                for (const auto& surface : surfaces) {
                        float t;
                        if (surface->intersect(ray, tMin, closest_t, t)) {
                                closest_t = t;
                                hit_surface = surface;
                        }
                }
                if (hit_surface) {
                        return vec3(1.0f, 1.0f, 1.0f); // white
                }
                return vec3(0.0f, 0.0f, 0.0f); // black
        }
};

void render()
{
```

- 카메라와 Scene생성. 표면, 구3개 생성.

```
        //Create camera and scene
        Camera camera;
        Scene scene;
        scene.surfaces.push_back(new Plane(vec3(0.0f, -2.0f, 0.0f), vec3(0.0f, 1.0f, 0.0f))); //
Plane P
        scene.surfaces.push_back(new Sphere(vec3(-4.0f, 0.0f, -7.0f), 1.0f)); // Sphere S1
        scene.surfaces.push_back(new Sphere(vec3(0.0f, 0.0f, -7.0f), 2.0f)); // Sphere S2
        scene.surfaces.push_back(new Sphere(vec3(4.0f, 0.0f, -7.0f), 1.0f)); // Sphere S3
```

- 이미지 초기화 후 반복문을 통해 ray와 물체가 만나는 지 확인.

```
        OutputImage.clear();
        for (int j = 0; j < Height; ++j)
        {
                for (int i = 0; i < Width; ++i)
                {
                        // Create a ray from the camera and check plane and spheres with
scene.trace
                        Ray ray = camera.getRay(i, j);
                        vec3 color = scene.trace(ray, 0.0f, std::numeric_limits<float>::max());

                        OutputImage.push_back(color.x); // R
                        OutputImage.push_back(color.y); // G
                        OutputImage.push_back(color.z); // B
                }
        }
}
```