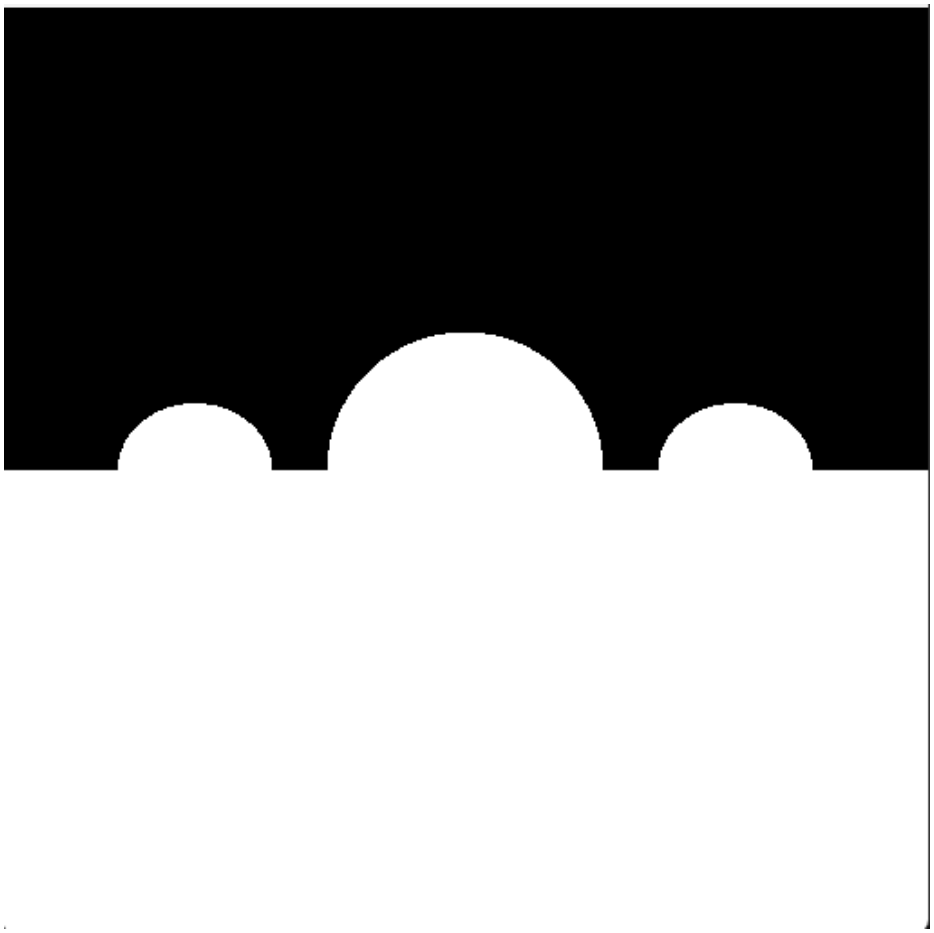
202112345 전건호

과제 보고서

결과 화면



Ray 클래스

- 광선의 시작점과 방향을 나타내는 클래스입니다.
- origin: 광선의 시작점을 저장하는 vec3 타입의 변수입니다
- direction: 광선의 방향을 저장하는 vec3 타입의 변수입니다.

```
// Ray 클래스: 광선을 표현합니다.
class Ray {
public:
    vec3 origin;    // 광선의 시작점
    vec3 direction; // 광선의 방향 벡터

    Ray(const vec3& origin, const vec3& direction) : origin(origin), direction(direction) {}
};
```

Camera 클래스

- 카메라의 위치, 방향, 뷰 영역을 정의하는 클래스입니다.
- eye: 카메라의 위치를 저장하는 vec3 타입의 변수입니다.
- u, v, w: 카메라의 방향을 나타내는 벡터입니다. 이 벡터들은 카메라의 로컬 좌표계를 형성합니다. u는 카메라의 오른쪽 방향, v는 카메라의 위쪽 방향, w는 카메라의 뒤쪽 방향을 나타냅니다.
- l, r, b, t, d: 뷰 영역의 left, right, bottom, top, distance 값을 저장하는 변수입니다.
- getRay(): 픽셀 좌표를 사용하여 카메라에서 광선을 생성하는 함수입니다.

```
// Camera 클래스: 카메라를 표현합니다.
class Camera {
public:
    vec3 eye;      // 카메라의 위치
    vec3 u, v, w; // 카메라의 방향 (u, v, -w)

    float l, r, b, t, d; // 뷰 영역 (left, right, bottom, top, distance)

    Camera(const vec3& eye, const vec3& u, const vec3& v, const vec3& w,
        float l, float r, float b, float t, float d)
        : eye(eye), u(u), v(v), w(w), l(l), r(r), b(b), t(t), d(d) {
    }

    // 픽셀 좌표를 통해 광선을 생성하는 함수
    Ray getRay(float ix, float iy) const {
        float ndc_x = (ix + 0.5f) / Width;
        float ndc_y = (iy + 0.5f) / Height;
        float screen_x = l + (r - l) * ndc_x;
        float screen_y = b + (t - b) * ndc_y;

        vec3 ray_direction = normalize(-d * w + screen_x * u + screen_y * v);
        return Ray(eye, ray_direction);
    }
};
```

Surface 클래스

- intersect(): 광선과 표면의 교차점을 계산하는 함수입니다. 이 함수는 광선이 표면과 만나는지 여부를 확인하고, 교차점의 매개변수 값 t를 계산합니다.
- getNormal(): 표면의 법선 벡터를 반환하는 함수입니다. 법선 벡터는 표면의 방향을 나타내며, 음영 계산에 사용됩니다.

```
// Surface 클래스: 모든 표면의 클래스입니다.
class Surface {
public:
    virtual bool intersect(const Ray& ray, float& t) const = 0;
    virtual vec3 getNormal(const vec3& point) const = 0;
};
```

Plane 클래스

- 평면을 나타내는 클래스입니다
- y: 평면의 y 좌표를 저장하는 변수입니다. 평면은 y축에 수직이며, y 좌표로 정의

됩니다.

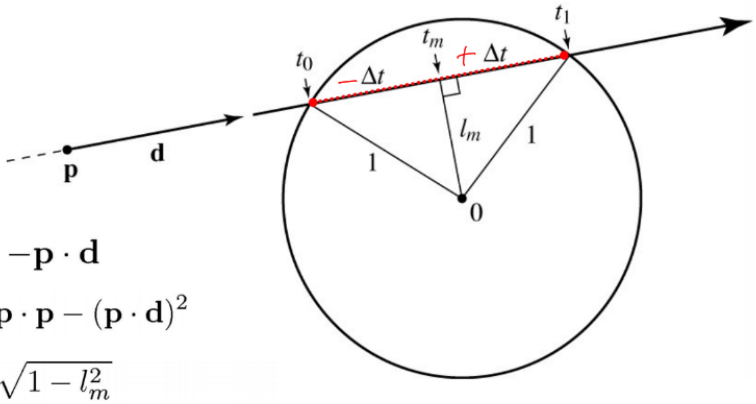
- intersect(): 광선과 평면의 교차점을 계산합니다. 평면의 방정식과 광선의 방정식을 연립하여 교차점을 구합니다.
- getNormal(): 평면의 법선 벡터를 반환합니다. 평면의 법선 벡터는 항상 일정하며, (0, 1, 0)입니다.

```
// Plane 클래스: 평면을 표현합니다.
class Plane : public Surface {
public:
    float y; // 평면의 y 좌표

    Plane(float y) : y(y) {}

    bool intersect(const Ray& ray, float& t) const override {
        if (abs(ray.direction.y) < 1e-6) { // 광선이 평면과 평행한 경우
            return false;
        }
        t = (this->y - ray.origin.y) / ray.direction.y;
        return t > 0; // 교차점이 광선 방향에 있어야 함
    }

    vec3 getNormal(const vec3& point) const override {
        return vec3(0, 1, 0); // 평면의 법선 벡터는 (0, 1, 0)
    }
};
```

Sphere 클래스

- 구를 나타내는 클래스입니다.
- center: 구의 중심을 저장하는 vec3 타입의 변수입니다.
- radius: 구의 반지름을 저장하는 변수입니다.
- intersect(): 광선과 구의 교차점을 계산합니다. 구의 방정식과 광선의 방정식을 연립하여 교차점을 구합니다.
- getNormal(): 구의 법선 벡터를 반환합니다. 구의 중심에서 교차점까지의 벡터를 정규화하여 법선 벡터를 구합니다.

Ray-sphere intersection: geometric



$$t_m = -\mathbf{p}\cdot\mathbf{d}$$

$$l_m^2 = \mathbf{p}\cdot\mathbf{p} - (\mathbf{p}\cdot\mathbf{d})^2$$

$$\Delta t = \sqrt{1 - l_m^2} = \sqrt{(\mathbf{p}\cdot\mathbf{d})^2 - \mathbf{p}\cdot\mathbf{p} + 1}$$

- $t_{0,1} = t_m \pm \Delta t = -\mathbf{p}\cdot\mathbf{d} \pm \sqrt{(\mathbf{p}\cdot\mathbf{d})^2 - \mathbf{p}\cdot\mathbf{p} + 1}$

```cpp
// Sphere 클래스: 구를 표현합니다.
class Sphere : public Surface {
public:
    vec3 center; // 구의 중심
    float radius; // 구의 반지름

    Sphere(const vec3& center, float radius) : center(center), radius(radius) {} // 구의 중심 좌표(center)와 반지름(radius)을 인자로 받아 초기화

    bool intersect(const Ray& ray, 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;// 판별식

        if (discriminant < 0) {
            return false; //교차점이 없는 경우 false 반환
        }
        // 교차점이 두 개인 경우 더 작은 값 선택
        t = (-b - ::sqrt(discriminant)) / (2 * a);
        if (t < 0) {
            t = (-b + ::sqrt(discriminant)) / (2 * a);
        }// 교차점이 광선 방향에 있으면 true 반환
        return t > 0;
    }

    vec3 getNormal(const vec3& point) const override {
        return normalize(point - center);
    }
};
```

Scene 클래스

- 장면을 관리하며, 광선 추적을 수행하는 클래스입니다.
- objects: 장면 내 객체들을 저장하는 Surface 포인터 벡터입니다.
- camera: 장면의 카메라를 저장하는 Camera 객체입니다.
- addObject(): 장면에 객체를 추가하는 함수입니다.
- trace(): 광선 추적 알고리즘을 구현하여 광선과 장면 내 객체 간의 교차점을 계산하고, 가장 가까운 교차점의 색상을 결정합니다. 이 함수는 장면 내 모든 객체에 대해 광선 교차 테스트를 수행하고, 교차점이 있으면 흰색, 없으면 검은색을 반환합니다.

```cpp
// Scene 클래스: 장면을 관리합니다.
class Scene {
public:
    std::vector<Surface*> objects;
    Camera camera;

    Scene(const Camera& camera) : camera(camera) {}

    void addObject(Surface* object) {
        objects.push_back(object);
    }

    // 광선 추적 함수
    vec3 trace(const Ray& ray) const {
        float closest_t = INFINITY;
        Surface* closest_surface = nullptr;
        // 모든 객체에 대해 가장 가까운 교차점을 가진 객체를 찾음
        for (Surface* object : objects) {
            float t;
            if (object->intersect(ray, t) && t < closest_t) {
                closest_t = t;
                closest_surface = object;
            }
        }
        //가장 가까운 교차점을 가진 객체가 있는가
        if (closest_surface) {
            return vec3(1.0f, 1.0f, 1.0f); // 흰색 반환
        }
        else {
            return vec3(0.0f, 0.0f, 0.0f); // 검은색 반환
        }
    }
};

// ----------------------------------------------------
```

Render 함수

- 각 픽셀에 대해 광선을 생성하고, 생성된 광선을 사용하여 장면을 추적하여 픽셀의 색상을 결정하고, 결정된 색상을 이미지 버퍼에 저장하는 과정을 반복하여 최종 이미지를 생성합니다.

```cpp
void render(Scene& scene) {
    OutputImage.clear();
    for (int j = 0; j < Height; ++j) {
        for (int i = 0; i < Width; ++i) {
            Ray ray = scene.camera.getRay(i, j);
            vec3 color = scene.trace(ray);
            OutputImage.push_back(color);
        }
    }
}
```