Sami Emre Erdogan
2019280513

Ray Tracing Project

One of the reasons why I have chosen raytracing is that it is purely C++ and it doesn't have any dependencies. For a Mac user who had difficulties setting up the dependencies especially in OpenGL projects is really frustrating. The second reason why I have chosen ray tracing is because it looked really fun and we had to look at the math part as well which I liked.

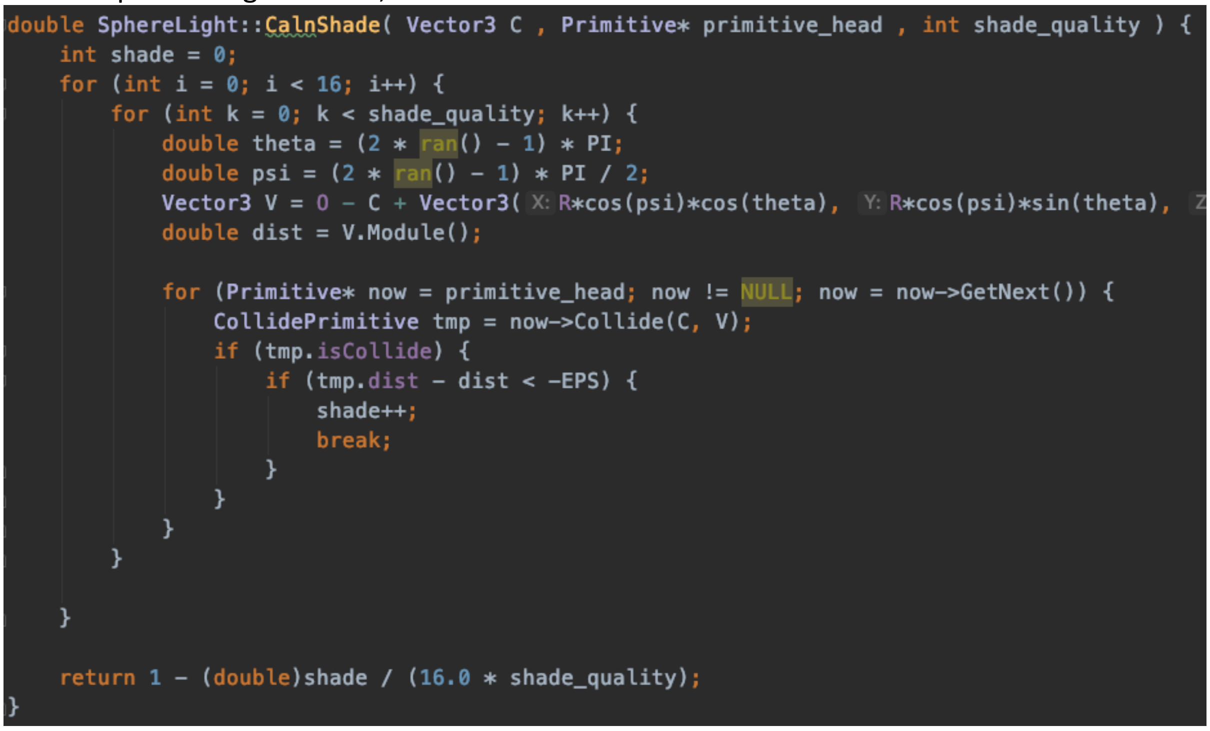
They wanted us to complete TO DO parts within the Raytracer.cpp, Primitive.cpp, Light.cpp. I have managed to complete everything apart from the Bezier primitives as I found it really difficult to implement. Also, I had limited time to complete it.

I want to explain briefly each of the functions that I have implemented:

LIGHT.CPP

For the surface light source and the spherical light source, each time the shadow test line is emitted, 16 points are randomly selected in the surface area / spherical surface to test whether there is an object blocking the connection line. This method can eventually achieve soft shadows, but the quality of shadows is greatly affected by the number of sampling points. The other TO DO is similar code so needs no explanation. We can get the idea.

For the spherical light source, the code is as follows:

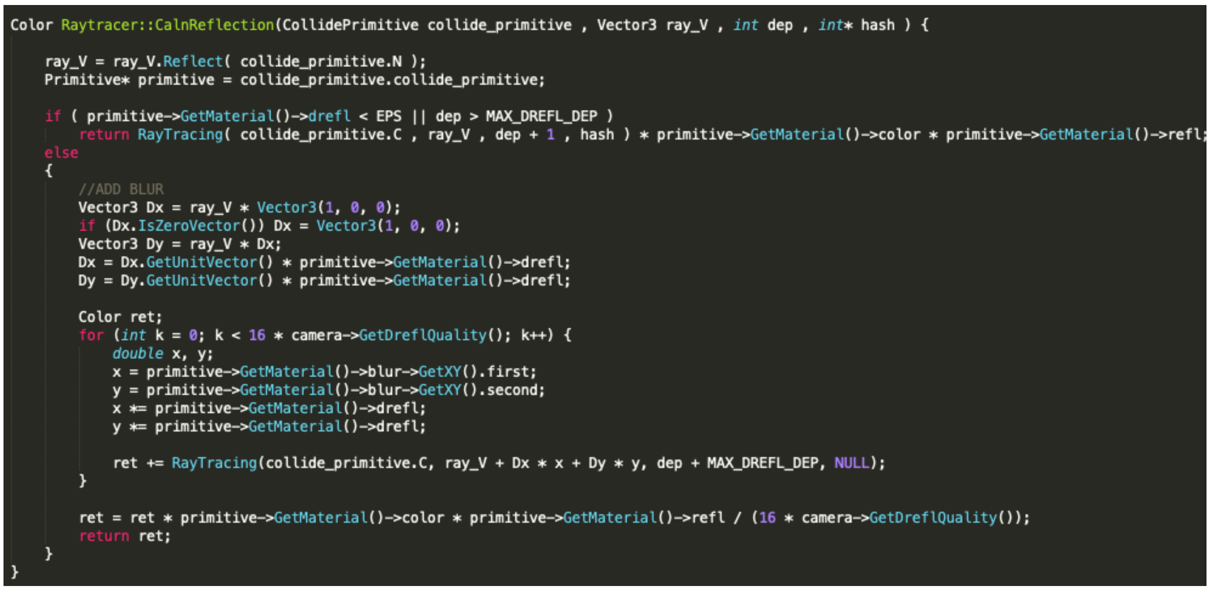


```
double SphereLight::CalnShade( Vector3 C , Primitive* primitive_head , int shade_quality ) {
    int shade = 0;
    for (int i = 0; i < 16; i++) {
        for (int k = 0; k < shade_quality; k++) {
            double theta = (2 * ran() - 1) * PI;
            double psi = (2 * ran() - 1) * PI / 2;
            Vector3 V = O - C + Vector3( X: R*cos(psi)*cos(theta),  Y: R*cos(psi)*sin(theta),  Z
            double dist = V.Module();

            for (Primitive* now = primitive_head; now != NULL; now = now->GetNext()) {
                CollidePrimitive tmp = now->Collide(C, V);
                if (tmp.isCollide) {
                    if (tmp.dist - dist < -EPS) {
                        shade++;
                        break;
                    }
                }
            }
        }

    }

    return 1 - (double)shade / (16.0 * shade_quality);
}
```

Sami Emre Erdogan
2019280513

RAYTRACER.CPP

For non-ideal specular reflection, a certain random reflection is added by adding a certain random offset to the light.
The specific method is, if the dreflect parameter in the material is not 0, the direction of the reflected light is randomly shifted. Sampling 16 times.



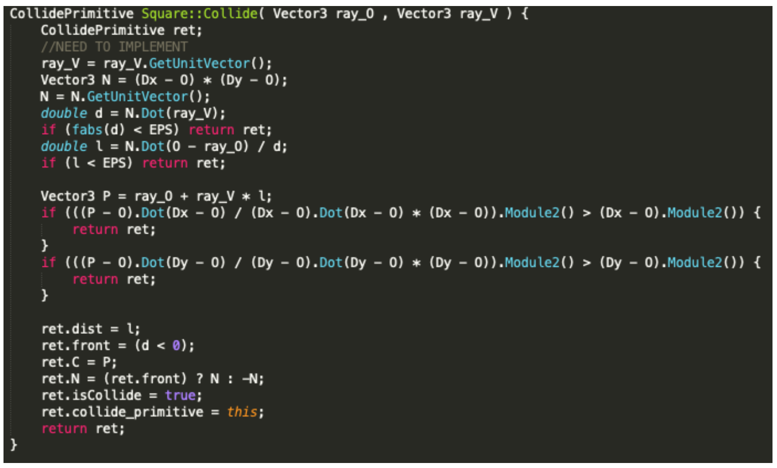
```cpp
Color Raytracer::CalnReflection(CollidePrimitive collide_primitive , Vector3 ray_V , int dep , int* hash ) {

    ray_V = ray_V.Reflect( collide_primitive.N );
    Primitive* primitive = collide_primitive.collide_primitive;

    if ( primitive->GetMaterial()->drefl < EPS || dep > MAX_DREFL_DEP )
        return RayTracing( collide_primitive.C , ray_V , dep + 1 , hash ) * primitive->GetMaterial()->color * primitive->GetMaterial()->refl;
    else
    {
        //ADD BLUR
        Vector3 Dx = ray_V * Vector3(1, 0, 0);
        if (Dx.IsZeroVector()) Dx = Vector3(1, 0, 0);
        Vector3 Dy = ray_V * Dx;
        Dx = Dx.GetUnitVector() * primitive->GetMaterial()->drefl;
        Dy = Dy.GetUnitVector() * primitive->GetMaterial()->drefl;

        Color ret;
        for (int k = 0; k < 16 * camera->GetDreflQuality(); k++) {
            double x, y;
            x = primitive->GetMaterial()->blur->GetXY().first;
            y = primitive->GetMaterial()->blur->GetXY().second;
            x *= primitive->GetMaterial()->drefl;
            y *= primitive->GetMaterial()->drefl;

            ret += RayTracing(collide_primitive.C, ray_V + Dx * x + Dy * y, dep + MAX_DREFL_DEP, NULL);
        }

        ret = ret * primitive->GetMaterial()->color * primitive->GetMaterial()->refl / (16 * camera->GetDreflQuality());
        return ret;
    }
}
```

PRIMITIVES.CPP

This part for me was the hardest part of the project because of trying to figure out the Bezier and at the end failed to do so, in this case I will just explain the square and the cylinder primitives.

SQUARE

1- Count the intersection with the plane
2- Normalization of the square
3- Do the calculations with the dot product
4- Check if the point we get is accurate.

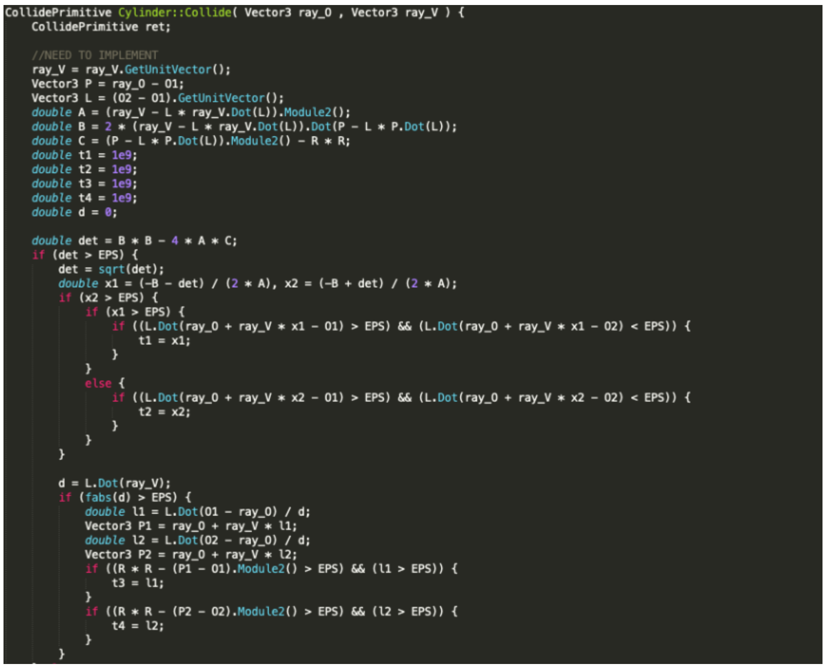


```cpp
CollidePrimitive Square::Collide( Vector3 ray_O , Vector3 ray_V ) {
    CollidePrimitive ret;
    //NEED TO IMPLEMENT
    ray_V = ray_V.GetUnitVector();
    Vector3 N = (Dx - O) * (Dy - O);
    N = N.GetUnitVector();
    double d = N.Dot(ray_V);
    if (fabs(d) < EPS) return ret;
    double l = N.Dot(O - ray_O) / d;
    if (l < EPS) return ret;

    Vector3 P = ray_O + ray_V * l;
    if (((P - O).Dot(Dx - O) / (Dx - O).Dot(Dx - O) * (Dx - O)).Module2() > (Dx - O).Module2()) {
        return ret;
    }
    if (((P - O).Dot(Dy - O) / (Dy - O).Dot(Dy - O) * (Dy - O)).Module2() > (Dy - O).Module2()) {
        return ret;
    }

    ret.dist = l;
    ret.front = (d < 0);
    ret.C = P;
    ret.N = (ret.front) ? N : -N;
    ret.isCollide = true;
    ret.collide_primitive = this;
    return ret;
}
```

Sami Emre Erdogan
2019280513

CYLINDER

1- Calculate the quadratic equation.
2- Check whether the intersection is in our defined boundary bases
3- Count the intersection of the cylinder bases which in our case is the top and the bottom part.
4- Closest collision to the ray is calculated at the end.



```
CollidePrimitive Cylinder::Collide( Vector3 ray_O , Vector3 ray_V ) {
    CollidePrimitive ret;

    //NEED TO IMPLEMENT
    ray_V = ray_V.GetUnitVector();
    Vector3 P = ray_O - O1;
    Vector3 L = (O2 - O1).GetUnitVector();
    double A = (ray_V - L * ray_V.Dot(L)).Module2();
    double B = 2 * (ray_V - L * ray_V.Dot(L)).Dot(P - L * P.Dot(L));
    double C = (P - L * P.Dot(L)).Module2() - R * R;
    double t1 = 1e9;
    double t2 = 1e9;
    double t3 = 1e9;
    double t4 = 1e9;
    double d = 0;

    double det = B * B - 4 * A * C;
    if (det > EPS) {
        det = sqrt(det);
        double x1 = (-B - det) / (2 * A), x2 = (-B + det) / (2 * A);
        if (x2 > EPS) {
            if (x1 > EPS) {
                if ((L.Dot(ray_O + ray_V * x1 - O1) > EPS) && (L.Dot(ray_O + ray_V * x1 - O2) < EPS)) {
                    t1 = x1;
                }
            }
            else {
                if ((L.Dot(ray_O + ray_V * x2 - O1) > EPS) && (L.Dot(ray_O + ray_V * x2 - O2) < EPS)) {
                    t2 = x2;
                }
            }
        }

        d = L.Dot(ray_V);
        if (fabs(d) > EPS) {
            double l1 = L.Dot(O1 - ray_O) / d;
            Vector3 P1 = ray_O + ray_V * l1;
            double l2 = L.Dot(O2 - ray_O) / d;
            Vector3 P2 = ray_O + ray_V * l2;
            if ((R * R - (P1 - O1).Module2() > EPS) && (l1 > EPS)) {
                t3 = l1;
            }
            if ((R * R - (P2 - O2).Module2() > EPS) && (l2 > EPS)) {
                t4 = l2;
            }
        }
```

BEZIER

I found some resources and articles about Beziers but I couldn't manage to implement it somehow. I was keep getting errors.☹ Once I have submitted my homework. I will try to implement it for learning purposes.

Sami Emre Erdogan
2019280513

RESULTS:
OUTPUT PICTURES FROM SCENE1 AND SCENE2:
The output pictures are stored in the “cmake-build-debug” folder.