Assignment 3 : Realistic rendering with ray tracing

NAME: NING BI
STUDENT NUMBER: 8574 5238
EMAIL: BINING@SHANGHAITECH.EDU.CN

1 INTRODUCTION

In this assignment, I created a scene with several objects including spheres and mesh objects made different material (perfect specular and perfect diffuse) then rendered them in a more realistic way by **ray tracing**.
The task I have completed are as follow:

(1) Load a mesh object from the OBJ file, and render the mesh with area light source in an enclosed environment box.
(2) Indirect lighting.
(3) Anti-aliasing with super-sampling and filtering.
(4) KD-tree speed up.

2 IMPLEMENTATION DETAILS

Since the assignment itself has already gave us a detail descriptions of whole process, I just briefly describe my method and give some image result.

2.1 Intersection with sphere

An arbitrary centered sphere can be represent as:

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - r^2 = 0$$

Then insert the ray equation in parametric form to obtain:

$$((o_x - x_0) + td_x)^2 + ((o_y - y_0) + td_y)^2 + ((o_z - z_0)^2 = r^2$$

Solve the equation:

$$t_0 = \frac{-B - \sqrt{B^2 - 4AC}}{2A}$$

$$t_1 = \frac{-B + \sqrt{B^2 - 4AC}}{2A}$$

where

$$A = d_x^2 + d_y^2 + d_z^2 \tag{1}$$

$$B = 2(d_x(o_x - x_0) + d_y(o_y - y_0) + d_z(o_z - z_0))$$

$$C = (o_x - x_0)^2 + (o_y - y_0)^2 + (o_z - z_0)^2 - r^2$$

Between t_0 and t_1 we should pick the one that bigger than zero and smaller that the other if it is also bigger than zero. Although something happened in my code that it could not really pick the right one even I applied those constrains.

```
ObjectIntersection Sphere::getIntersection(const Ray & ray)
{
    Vector3f dir = ray.direction();
    Vector3f pos = ray.origin() - _position;
    Vector3f ori = ray.origin();
    double A = dir.dot(dir);
    double B = 2 * dir.dot(pos);
    double C = pos.dot(pos) - pow(_radius,2);
    double delta = pow(B, 2) - 4 * A * C;
    if (delta >= 0)
    {
        float t0 = (-B - sqrt(delta)) / (2 * A);
        float t1 = (-B + sqrt(delta)) / (2 * A);
        if (t1 < 0 && t0 < 0)
        {
            return ObjectIntersection(false, 0, Vector3d(0.0), Material(DIFF));
        }
        float t = t0*t1 >= 0? std::min(t0,t1) : std::max(t0,t1);
        Vector3f intersectPoint = ori + t * dir;
        Vector3d norm = ( _position - intersectPoint).normalized();
        return ObjectIntersection(true, t, norm, _material);
    }
    return ObjectIntersection(false, 0, Vector3d(0.0), Material(DIFF));
}
```

Fig. 1. code of intersection with sphere

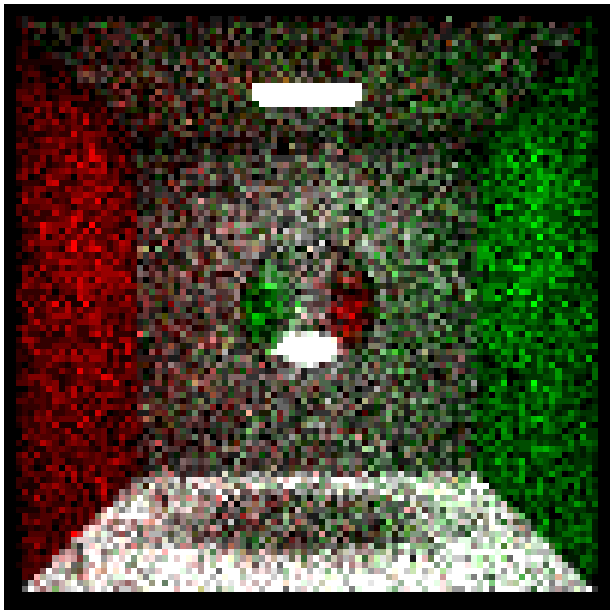

Fig. 2. An obvious inverse of the rendering side can be seen from here though I fixed the code.

2.2 Intersection with triangle

Here, we calculate intersection with triangles by using **barycentric coordinate**. In barycentric coordinate, any point inside the triangle can be written as:

$$p(b_1, b_2) = (1 - b_1 - b_2)p_0 + b_1 p_1 + b_2 p_2$$

```cpp
ObjectIntersection Triangle::getIntersection(const Ray & ray)
{
    const Vector3d e1 = _p1 - _p0;
    const Vector3d e2 = _p2 - _p0;
    const Vector3d s = ray.origin() - _p0;
    double head = 1 / (e1.dot(ray.direction().cross(e2)));

    double t = head * (e2.dot(s.cross(e1)));
    double b1 = head * (s.dot(ray.direction().cross(e2)));
    double b2 = head * (ray.direction().dot(s.cross(e1)));
    double b = b1 + b2;

    if (b <= 1 && b1 > 0 && b2 > 0)
    {
        Vector3d intersectPoint = ray.origin() + t * ray.direction();
        Vector3d norm = (_p0 - _p1).cross((_p1 - _p2));
        norm = norm.normalized();

        return ObjectIntersection(true, t, norm, _material);
    }
    return ObjectIntersection(false, 0.0, Vector3d(0), Material(DIFF,Vector3d(0.0)));
}
```

Fig. 3. code of intersection with triangle

```cpp
ObjectIntersection Quad::getIntersection(const Ray & ray)
{
    Triangle tri1 = Triangle(_a, _b, _c, _material);
    ObjectIntersection isct1 = tri1.getIntersection(ray);
    if (isct1._hit) {
        return isct1;
    }
    Triangle tri2 = Triangle(_a, _c, _d, _material);
    ObjectIntersection isct2 = tri2.getIntersection(ray);
    if (isct2._hit)
    {
        return isct2;
    }
    return ObjectIntersection(0, 0, Vector3d(0.0), Material(DIFF));
}
```

Fig. 4. code of intersection with quadratic

with a condition:

$$b_1 \geq 0, b_2 \geq 0, b_1 + b_2 \leq 1$$

We simplify the calculation in intersection with mesh and AABBox by divide quadratic into two triangles and apply intersection with triangles separately.

2.3 Speed up

There are two main strategies used to speed up ray tracing; one is **bounding volume hierarchies** achieved by KD-tree the other is sampling strategy used in get reflection of different materials.See reference for more kd-tree details.

2.4 Test result

Here, some results are posted.

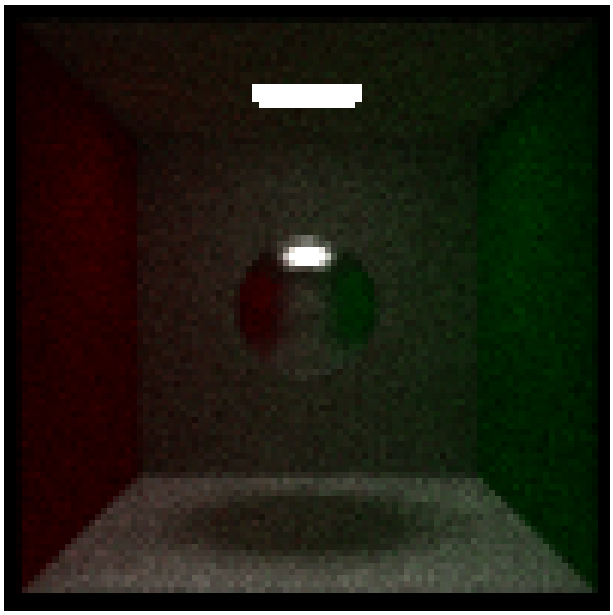

Fig. 5. 20000 samples specular sphere.

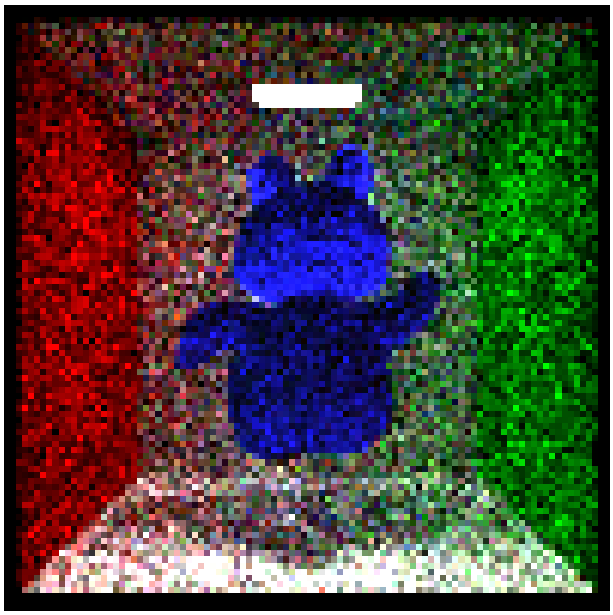

Fig. 6. 1000 samples diffuse teddy mesh.

3 REFERENCE

Kd-tree tutorial:
http://www.bowdoin.edu/~ltoma/teaching/cs3250-CompGeom/spring14/hw-kdtree.html

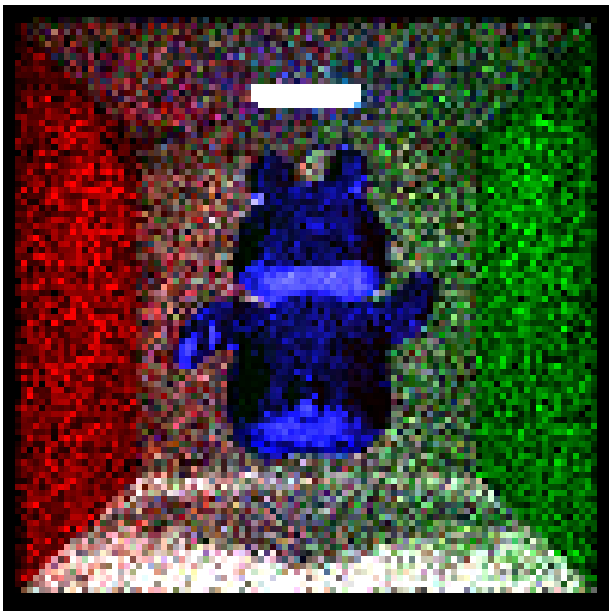

Fig. 7. 1000 samples specular teddy mesh.