实验报告 6

姓名：蒋卓洋
学号：59119125

1、实现

1.1. getIntersection()
(1)位置：Triangle.cpp
(2)实现：

```
1.  inline Intersection Triangle::getIntersection(Ray ray)
2.  {
3.      Intersection inter;
4.
5.      if (dotProduct(ray.direction, normal) > 0)
6.          return inter;
7.      double u, v, t_tmp = 0;
8.      Vector3f pvec = crossProduct(ray.direction, e2);
9.      double det = dotProduct(e1, pvec);
10.     if (fabs(det) < EPSILON)
11.         return inter;
12.
13.     double det_inv = 1. / det;
14.     Vector3f tvec = ray.origin - v0;
15.     u = dotProduct(tvec, pvec) * det_inv;
16.     if (u < 0 || u > 1)
17.         return inter;
18.     Vector3f qvec = crossProduct(tvec, e1);
19.     v = dotProduct(ray.direction, qvec) * det_inv;
20.     if (v < 0 || u + v > 1)
21.         return inter;
22.     t_tmp = dotProduct(e2, qvec) * det_inv;
23.
24.     // TODO find ray triangle intersection
25.
26.     ////////////////Solution//////////////////
27.     ////Name:JiangZhuoyang
28.     ////StudentID:58119125
29.     ////FinishDate:21/11/19
30.
```

```
    //1.t<0
    if(t_tmp < 0){
        return inter;
    }

    //2.Renew Intersection Structure
    inter.distance = t_tmp;
    inter.happened = true;
    inter.m = m;
    inter.coords = ray(t_tmp);
    inter.normal = normal;
    inter.obj = this;
    //////////////////////////////////////////////
    
    return inter;
}
```

1.2. Render()

(1)位置：Renderer.cpp

(2)实现：

```
void Renderer::Render(const Scene& scene)
{
    std::vector<Vector3f> framebuffer(scene.width * scene.height);

    float scale = tan(deg2rad(scene.fov * 0.5));
    float imageAspectRatio = scene.width / (float)scene.height;
    Vector3f eye_pos(-1, 5, 10);
    int m = 0;
    for (uint32_t j = 0; j < scene.height; ++j) {
        for (uint32_t i = 0; i < scene.width; ++i) {
            // generate primary ray direction
            float x = (2 * (i + 0.5) / (float)scene.width - 1) *
                    imageAspectRatio * scale;
            float y = (1 - 2 * (j + 0.5) / (float)scene.height) * scale;
            // TODO: Find the x and y positions of the current pixel to get the
            // direction
            //  vector that passes through it.
            // Also, don't forget to multiply both of them with the variable
            // *scale*, and x (horizontal) variable with the *imageAspectRatio*

            // Don't forget to normalize this direction!
```

```
            Vector3f dir = Vector3f(x, y, -1); // Don't forget to normalize this direction!
            dir = normalize(dir);
            Ray ray(eye_pos,dir);
            framebuffer[m++] = scene.castRay(ray, 0);
        }
        UpdateProgress(j / (float)scene.height);
    }
    UpdateProgress(1.f);

    // save framebuffer to file
    FILE* fp = fopen("binary.ppm", "wb");
    (void)fprintf(fp, "P6\n%d %d\n255\n", scene.width, scene.height);
    for (auto i = 0; i < scene.height * scene.width; ++i) {
        static unsigned char color[3];
        color[0] = (unsigned char)(255 * clamp(0, 1, framebuffer[i].x));
        color[1] = (unsigned char)(255 * clamp(0, 1, framebuffer[i].y));
        color[2] = (unsigned char)(255 * clamp(0, 1, framebuffer[i].z));
        fwrite(color, 1, 3, fp);
    }
    fclose(fp);
}
```

1.3. IntersectP()

(1)位置： Bounds3.cpp

(2)实现：

```
inline bool Bounds3::IntersectP(const Ray& ray, const Vector3f& invDir,
                                const std::array<int, 3>& dirIsNeg) const
{
    // invDir: ray direction(x,y,z), invDir=(1.0/x,1.0/y,1.0/z), use this because Multiply is faster that Division

    // dirIsNeg: ray direction(x,y,z), dirIsNeg=[int(x>0),int(y>0),int(z>0)], use this to simplify your logic

    // TODO test if ray bound intersects

    /////////////////Solution//////////////////
    ////Name:JiangZhuoyang
    ////StudentID:58119125
    ////FinishDate:21/11/11
```

```
14.
15.     //1.Firstly, I deduce enter time and the exit time on paper which shows we just need t
        hese points: pMin,pMax,origin and invDir=(1.0/x,1.0/y,1.0/z) in fomular
16.
17.     //2.Calculate the basic time sictuation without consider the direction negtiveness.
18.     Vector3f t_min = (pMin - ray.origin) * invDir;
19.     Vector3f t_max = (pMax - ray.origin) * invDir;
20.
21.     //3.We need to consider if the direction is negtive with the array dirIsNeg.
22.     //Here, we image that direction x of ray is negtive and we know that the ray will just h
        as intersection with the objects whose pMin_x and pMax_x are negtive. (We do not cons
        ider the origin is inside an object)
23.     //So that, we will find the t_min and t_max will be opposite, we need to exchange the
        m.
24.     if(!dirIsNeg[0]){
25.         float temp = t_min.x;
26.         t_min.x = t_max.x;
27.         t_max.x = temp;
28.     }
29.     if(!dirIsNeg[1]){
30.         float temp = t_min.y;
31.         t_min.y = t_max.y;
32.         t_max.y = temp;
33.     }
34.     if(!dirIsNeg[2]){
35.         float temp = t_min.z;
36.         t_min.z = t_max.z;
37.         t_max.z = temp;
38.     }
39.
40.     //4.Deduce enter time and the exit time
41.     float t_enter = std::max(t_min.x, std::max(t_min.y, t_min.z));
42.     float t_exit = std::max(t_max.x, std::max(t_max.y, t_max.z));
43.
44.     //5.Do the judement:
45.     if(t_enter < t_exit && t_exit >= 0){
46.         return true;
47.     }
48.     else{
49.         return false;
50.     }
51.
52.     ////////////////////////////////////////////
53. }
```

1.4. getIntersection()

(1)位置：BVH.cpp

(2)实现：

```
Intersection BVHAccel::getIntersection(BVHBuildNode* node, const Ray& ray) const
{
    // TODO Traverse the BVH to find intersection
    /////////////////Solution//////////////////
    ////Name:JiangZhuoyang
    ////StudentID:58119125
    ////FinishDate:21/11/19
    //1.Define the returned structure.
    Intersection Intersection_result;

    //2.Prepare for the usage of the Function "Bounds3::IntersectP" on each recurrence st
    eps
    Vector3f invDir = (1.0/ray.direction.x, 1.0/ray.direction.y, 1.0/ray.direction.z);
    std::array<int, 3> dirIsNeg = {ray.direction.x>0,ray.direction.y>0,ray.direction.z>0};

    //3.Terminal Condition with the Function "Bounds3::IntersectP"
    //(1)Null node or Without intersection with the bounds at this recurrence step.
    if(node == nullptr || !node->bounds.IntersectP(ray,invDir,dirIsNeg)){
        return Intersection_result;//return the initial result
    }
    //(2)Leaf node
    if(node->left == nullptr && node->right == nullptr){
        Intersection_result = node->object->getIntersection(ray);
        return Intersection_result;//return the closet Intersection.
    }

    //4.Recurrence with the bound's renew
    Intersection hit1 = getIntersection(node->left,ray);
    Intersection hit2 = getIntersection(node->right,ray);

    //5.Get result of the closer Intersection and reture it.
    Intersection_result = hit1.distance <= hit2.distance ? hit1 : hit2;
    return Intersection_result;
    ////////////////////////////////////////////
}

```

2、结果

- 实验结果如下：

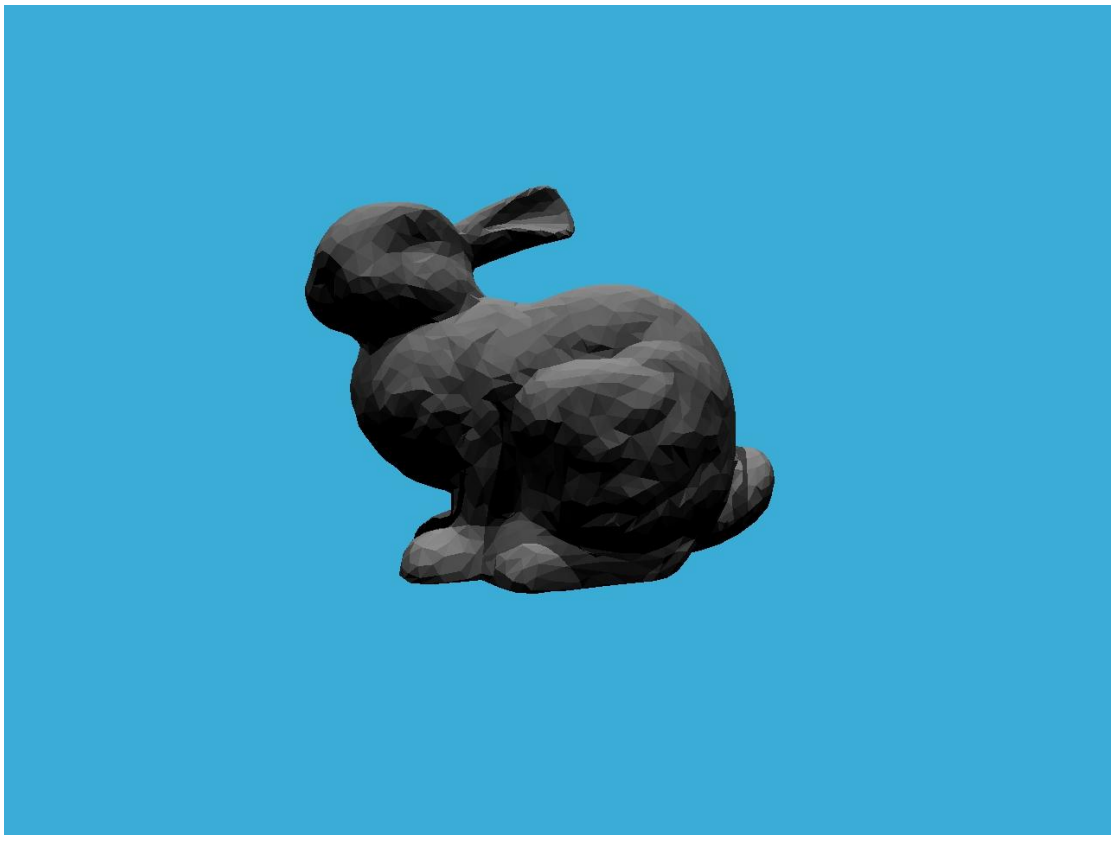

图 1. 实验结果