一、实验步骤

1、构造 Object3D 类：

```
class Object3D {
public:
    Object3D(){}
    Object3D(Material *pointer) {
        mat = pointer;
    }
    ~Object3D(){
        if (mat != NULL) delete mat;
    }
    virtual bool intersect(const Ray &r, Hit &h, float tmin) = 0;

private:
    Material *mat;
   };
```

2、在此基础上构造其子类 Sphere（具体算法参考 RayCasting1）：

```
class Sphere : public Object3D
{
public:
    Sphere() {}
    Sphere(Vec3f &cen, float &r, Material *p) {
        center = cen;
        radius = r;
        mat = p;
    }
    ~Sphere() {
        if (mat != NULL) delete mat;
    }
    virtual bool intersect(const Ray &r, Hit &h, float tmin) {
        bool state = 1;
        Vec3f ori, Rd;
        ori = r.getOrigin();
        Rd = r.getDirection();
        Vec3f Ro;
        Ro = ori - center;
        int isin = 0;
        if (Ro.Length() * Ro.Length() > radius * radius) isin = -1;
        else if (Ro.Length() * Ro.Length() < radius * radius) isin = -1;
        else state = 0;

        float tp;
        if (state) {
            tp = - Ro.Dot3(Rd);
            if (isin == -1 && tp < 0) state = 0;
        }

        float d2;
        if (state) {
            d2 = Ro.Length() * Ro.Length() - tp * tp;
            if (d2 >= radius * radius) state = 0;
        }

        float t_;
```

```
        float t;
        if (state) {
            t_ = radius * radius - d2;
            t_ = sqrtf(t_);

            if (isin == -1) t = tp - t_;
            else if (isin == 1) t = tp + t_;

            if (t < tmin) state = 0;
            else {
                if (t < h.getT()) h.set(t, mat, r);
                state = 1;
            }
        }


        bool stated = 1;
        Rd = Rd * (-1);
        Ray rd(ori,Rd);
        if (tmin < 0) {
            tp = -Ro.Dot3(Rd);
            if (isin == -1 && tp < 0) stated = 0;

            if (stated) {
                d2 = Ro.Length() * Ro.Length() - tp * tp;
                if (d2 >= radius * radius) stated = 0;
            }
            if (stated) {
                t_ = radius * radius - d2;
                t_ = sqrtf(t_);

                t = tmin - 1;
                if (isin == -1) t = tp + t_;
                else if (isin == 1) t = tp + t_;

                if (t < tmin) stated = 0;
                else {
                    if (-t < h.getT()) h.set(-t, mat, r);
                    stated = 1;
                }
            }
        }
        if (state == 1 || stated == 1) state = 1;
        return state;
    }

private:
    Vec3f center;
    float radius;
    Material *mat;
};
```

需要注意的是，在给出的案例中，可以看出，不管视点朝向是正还是负，其两侧的球体都可以被投射显示出来，所以 intersect 函数中要注意对两个方向的球体都进行判断。

3、构造 Object3D 的子类 Group，用于 存放一系列物体：

```
class Group : public Object3D
{
public:
    Group(){}
    Group(int &num) {
        number = num;
        p = new Object3D *[number];
    }
    ~Group(){
        if (p != NULL) delete[] p;
    }
    void addObject(int index, Object3D *obj) {
        p[index] = obj;
    }
    virtual bool intersect(const Ray &r, Hit &h, float tmin) {
        int i, state = 0;
        for (i = 0; i < number; i++) {
            if (p[i] == NULL) continue;
            else {
                if (p[i]->intersect(r, h, tmin)) state = 1;
            }
        }
        return state;
    }
private:
    Object3D **p;
    int number;
};
```

其 intersect 函数只是递归地调用所有的物体。注意析构函数要 delete。

4、构造 camera 类，和其子类正投影相机：

```
class Camera {
public:
    Camera (){}
    ~Camera() {}
    virtual Ray generateRay(Vec2f point) = 0;
    virtual float getTMin() const = 0;
};

class OrthographicCamera : public Camera
{
public:
    OrthographicCamera(){}
    OrthographicCamera(const Vec3f &cen, const Vec3f &dir,
                       const Vec3f &upp, const float &siz) {
        center = cen;
        direction = dir;
        direction.Normalize();

        Vec3f tmp;
        tmp = direction * upp.Dot3(direction);
        up = upp - tmp;
        up.Normalize();

        tmp.Cross3(horizontal,direction,up);
```

```
        //horizontal.Normalize();
        size = siz;
    }
    ~OrthographicCamera(){}

    Ray generateRay(Vec2f point) {
        Vec3f position,xray,yray;
        float x, y;
        point.Get(x, y);
        assert(x >= 0 && x < 1);
        assert(y >= 0 && y < 1);
        xray = horizontal * (x * size);
        yray = up * (y * size);
        position = center - (size/2) * up - (size/2) * horizontal;
        position += xray + yray;
        Ray tmp(position,direction);
        return tmp;
    }

    float getTMin() const{
        return -MAXnum;
    }

private:
    Vec3f center;
    Vec3f direction;
    Vec3f up;
    Vec3f horizontal;
    float size;
};
```

需要注意的是，构造时，要将传入的参数标准化，叉乘的顺序不能颠倒，tmin的值设为一个很大的负数。

5、编写主函数生成颜色图和深度图：

```
SceneParser scene(input_file);
    Camera *cam = scene.getCamera();
    Group *gro = scene.getGroup();
    float tmin = cam->getTMin();
    Ray r;
    Hit h(MAXnum,scene.getMaterial(0));

    Vec2f position;
    Vec3f color,depcolor;
    depcolor.Set(0, 0, 0);

    Image outimg(width,height);
    outimg.SetAllPixels(scene.getBackgroundColor());
    Image depimg(width, height);
    depimg.SetAllPixels(depcolor);

    float depth;
    int j, k;
    for (i = 0; i < width; i++) {
        for (j = 0; j < height; j++) {
            position.Set(1.0*i/width, 1.0*j/width);
            r = cam->generateRay(position);
```

```
            h.set(MAXnum, scene.getMaterial(0), r);
            if (gro->intersect(r, h, tmin)) {
                color = h.getMaterial()->getDiffuseColor();
                outimg.SetPixel(i, j, color);

                depth = (depth_max - h.getT()) / (depth_max - depth_min);
                depcolor.Set(depth,depth,depth);
                depimg.SetPixel(i, j, depcolor);
            }
        }
    }
    outimg.SaveTGA(output_file);
    depimg.SaveTGA(depth_file);
```

注意在生成颜色图之前要将背景颜色设置好。

二、实验结果