# A1: Ray Casting（光线投射）

一、概述

**1.目标：**

基于正交摄像机和场景中的球体，实现一个基本的光线投射器。球体采用的着色模型十分简单，只有单一颜色。同时，我们还需要把光线和模型交点到摄像机的距离(t)用一种可视化的方式呈现（灰度图）。

**2. Ray Casting 介绍：**

将三维模型渲染到一张 2D 图片上需要通过一个摄像机来观察。摄像机分为正交(orthographic)和透视(perspective)两种。要确定 2D 图片上每个像素的颜色，摄像机会从每个像素处发射一条射线，检测场景中的所有物体，确定该射线和物体的最近交点，交点处的颜色就是像素的颜色。

**3.算法介绍：**

算法描述如下：

```
For every pixel
    Construct a ray from the eye
    For every object in the scene
        Find intersection with the ray
        Keep if closest
    Shade depending on light and normal vector
```

（1）遍历每个像素，从摄像机中构造一条射线：

图片的分辨率和摄像机的分辨率不同，要先将像素的坐标映射到 0-1 之间，再确定摄像机中射线的起点坐标。

```
Origin = center + (x-0.5)*size*horizontal + (y-0.5)*size*up
```

（2）找到射线和球体的交点：

**方法一：代数法**

射线的参数方程是：ray : P(t) = O + D·t ( t >= 0 )

球的方程：sphere : ( P-C )^2 = R^2

对于球来说，空间中的任意一点 P，如果它到球心 C 的距离和半径 R 相等，就说明它在球上。

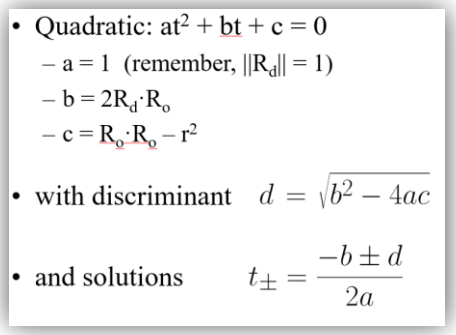
把 P(t)代入球的方程中，可以得到一个二元一次方程，如果方程无根，则射线和球没有交点。

二元一次方程：t·t + 2·(OC·D)·t + OC·OC - R·R = 0;

/*OC 为 OC 之间的距离，同右图中的 Ro（先把 Origin 移动到 center 处）*/

a = D·D = dot(D, D) = 1;　　b = 2·OC·D = 2·dot(OC, D);

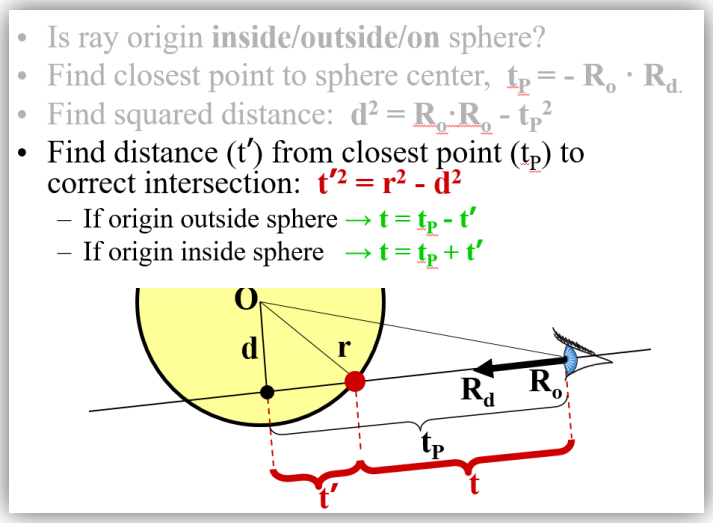
c = OC·OC - R·R = dot(OC, OC) - R·R;




若求解出根 t+和 t-。因为要选择距离摄像机最近的正向距离，所以可以先验证 t-，再验证 t+。

**方法二：几何法**

当射线和球相交的时候，球心 center 到射线 ray 的距离 d <= R，这个即为相交的条件。如果相交，求出 d 之后，可以根据勾股定理求出射线和球的交点。




Ro 表示向量 Ro->O

## 二、实现细节

**1.接收输入：**

输入格式:

```
-input scene1_01.txt -size 200 200 -output output1_01.tga -depth 9 10 depth1_01.tga
```

**2. 摄像机：**

**（1）OrthographicCamera 类：**

```
class OrthographicCamera : public Camera
{
public:
    OrthographicCamera (Vec3f center, Vec3f direction, Vec3f up, float size);
    ~OrthographicCamera ();
    virtual Ray generateRay (Vec2f point);
    virtual float getTMin () const;
private:
    Vec3f center;
    Vec3f direction, up, horizontal;
    float imgSize;
};
```

**（2）发射射线：**

```
Ray OrthographicCamera::generateRay (Vec2f point){
    Vec3f origin=center+ (-0.5f+point.x ())*imgSize * horizontal+ (-0.5f+point.y
())*imgSize*up;
    Ray ray (origin,direction);
    return  ray;
}
```

**3.检测球体和射线的交点：**

**（1）代数法：**

```
bool Sphere::intersect (const Ray &r, Hit &h, float tmin){
    Vec3f OC = r.getOrigin () - this->center;//OC距离
    Vec3f dir = r.getDirection ();

    float a = dir.absLength ();
    float b = 2.0*Vec3f::Dot (dir, OC);
    float c = OC.absLength () - pow (this->radius, 2);
    float discrim = pow (b, 2) - (4 * a*c);
    float t0=0, t1=0;
    if (discrim < 0) return false;//无根
    else if (discrim < FLT_EPSILON) { //一个根
        t0=t1 = (-1.0*b - sqrt (discrim)) / (2.0*a);
    }else {//两个根
        t0 = (-1.0*b - sqrt (discrim)) / (2.0*a);
        t1 = (-1.0*b + sqrt (discrim)) / (2.0*a);
    }
    //判断根是否有效
    if (t0 >= tmin && t0 <= h.getT ()) {
        h.set (t0, this->material, r);
        return true;
    }else if (t1 >= tmin && t1 <= h.getT ()) {
        h.set (t1, this->material, r);
        return true;
    }
}
```

**(2) 几何法：**

```
    Vec3f oc = center -r.getOrigin();
    float tp = dir.Dot3(oc);//oc在dir上的投影
    if (tp < 0)  return false; //无交点，投影小于0
    float oc2 = oc.Dot3(oc);
    float distance2 = oc2 - tp * tp;//求出d平方

    if (distance2 > pow(radius,2)) return false;//无交点

    float discriminant = pow (radius, 2) - distance2;//t'的平方
    float t0=0, t1=0;
    if (discriminant < FLT_EPSILON) t0 = t1 = tp;
    else {
        discriminant = sqrt (discriminant);
        t0 = tp - discriminant;
        t1 = tp + discriminant;
    }
//判断根是否有效的代码同上
```

**4.渲染循环：**

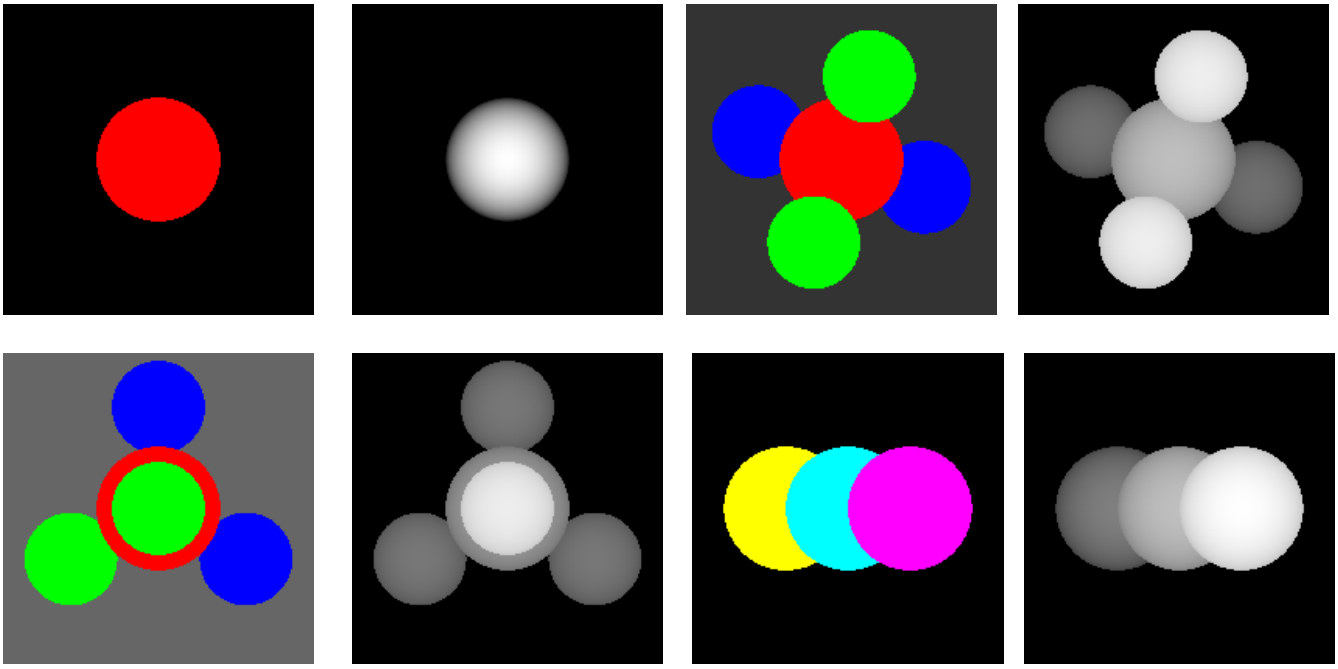
**（1）Render Colorful Image**

```
    SceneParser* scene =new SceneParser(input_file);
    Image* image = new Image (width,height);
    image->SetAllPixels (scene->getBackgroundColor());//背景色
  //************************RenderColorImage*****************************//
for (int x = 0; x < width; x++){
    for (int y = 0; y < height; y++){
        Vec2f coord((x+0.0)/(width-1),(y+0.0)/(height-1) ); //逐像素：范围0-1
        Ray ray = scene->getCamera ()->generateRay (coord);
        Hit hit (FLT_MAX, NULL); //初始化材质为空，距离为无穷大
        if (scene->getGroup()->intersect(ray,hit,scene->getCamera()->getTMin()))
       { //相交后，根据hit存储的值着色
          image->SetPixel (x,y,hit.getMaterial()->getDiffuseColor());
        }
    }
}
    image->SaveTGA (output_file);
```

**（2）Render Depth Image**

```
image->SetAllPixels (Vec3f(0,0,0));//背景色
for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
        Vec2f coord ((x + 0.0)/(width - 1),(y + 0.0)/(height - 1));//逐像素：范围0-1
        Ray ray = scene->getCamera ()->generateRay (coord);
        Hit hit (FLT_MAX, NULL); //初始化材质为空，距离为无穷大
        if (scene->getGroup()->intersect(ray,hit,scene->getCamera()->getTMin ()))
        {
            if(depth_file != NULL){
                if(hit.getT () < depth_min){
                    image->SetPixel (x, y, Vec3f (1.0f, 1.0f, 1.0f));
                }
                else if (hit.getT () > depth_max){
                    image->SetPixel (x, y, Vec3f (0.0f, 0.0f, 0.0f));
                }
                else{
             float grayScale = (depth_max - hit.getT ()) / (depth_max - depth_min);
                    image->SetPixel (x, y, Vec3f (grayScale, grayScale, grayScale));
                }
            }
          }
      }
}
image->SaveTGA (depth_file);
```

## 三、结果展示



## 四、心得体会

在查看射线和球体求交点的算法中发现以前的有些数学知识已经忘记了，没太看懂，在查阅资料和演算之后才搞明白，草稿纸是个好朋友。还有就是代码量比较多的时候 debug 比较麻烦，任何一步的出错都不能得到正确的结果，用单步调试和分段调试可以比较快的解决问题。