READ ME

姓名：周宁
班级：111171
学号：20171004140

一、实习思路：

1. 如何编译和运行代码

①在 vs 中创建一个工作项目，将作业“Assignment2.files”文件夹下的对应的.h 和.c 文件复制中项目当中，然后将.c 文件改为.cpp 文件即可，如：将文件“scene_parser.C”复制到工作目录当中，然后改为“scene_parser.cpp”。

②从“31”（上次题目）中复制一些必要的文件，到工程项目中，如：image.h 等文件，这些是是运算所必须的文件，所以也需要添加进去。

2. 编写纯虚类 Object3D

因为是纯虚类,所以 Object3D 也比较简单,里面只有一个数据成员 **Material *m_material**，也只有一个比较重要的函数 **virtual bool intersect(const Ray& r, Hit& h, float tmin) = 0**。

①数据成员 **Material *m_material**。Material 代表材质，表示该物体的材质。

②函数 **virtual bool intersect(const Ray& r, Hit& h, float tmin) = 0**。该函数是一个纯虚函数，代表着物体的相交算法。

3. 编写 Sphere 类

类 Sphere 继承自 Object3D，所以需要实现虚函数 **intersect**。

①设计 Sphere 的数据成员。Sphere 由两个新的字段球心（**m_center**）、半径（**m_material**），由于 Sphere 是继承自 Object3D，所以还从父类里面继承了 **m_material**，体实现可以参考图 1。

```
protected:

    Vec3f m_center; //球心位置
    float m_radius; //半径
```

图 1

②实现虚函数 **virtual bool intersect(const Ray& r, Hit& h, float tmin)**,该函数的功能是实现光线和球的相交算法。可参考 ppt 中“08_光线投射-2 求交算法”。首先是创造出以球心为原点的光线表达式，这样才能使用到 ppt 上相对应的线球求交的算法当中，其实

方法比较简单，只需要将光线的“起始点”减去球的“中心点”，就可以将线转化到以球心为原点的表达式当中，具体的可以参考图 2。

```
//需要求光线和球的交点
//可以参考ppt光线投射求交算法
//Rd • Rdt2 + 2Rd • Rot + Ro • Ro - r2 = 0
//a=1,b = 2Rd • Ro,c = Ro • Ro  -  r2

//转化为以球为原点的 direction、origin
Vec3f direction = r.getDirection();
Vec3f origin = r.getOrigin()-this->m_center;
```

图 2

然后求出 ppt 中的 b 和 c，再根据判别式 b^2-4ac(a=1)判断是否存在根，具体可以参考图 3。

```
//求出b和c，然后使用判别式
double b = 2*direction.Dot3(origin);
double c = origin.Dot3(origin) - m_radius * m_radius;

//判断有没有根 b^2 -4ac ，其中a=1
double d = b * b - 4 * c;
if (d > 0.00001) {
```

图 3

如果存在根，可以求解出较小的根，再判断求解出来的最小的根 t，和碰撞 h 中的最小值进行比较，如果 t<h.getT()，则说明离碰撞点更近，需要修改 hit 的值，固体可以参考图 4。

```
    //说明存在根，求解较小的根
    double t = -(b + d) / 2;
    //说明离碰撞点更近，需要修改hit的值
    if (t >=tmin && t < h.getT()) {
        //cout << "这里被调用" << endl;
        h.set(t, m_material, r);
        //说明相交了
        return true;
    }
```

图 4

4. 编写纯虚类 Camera

纯虚类 Camera 比较简单只有两个纯虚函数 **virtual Ray generateRay(Vec2f point) = 0;** 和 **virtual float getTMin() const = 0**，其中 generateRay 的功能是根据输入的点生成相对应的光线，而 getTMin()是返回相机中认为的最小值 tmin，只有大于这个最小值 tmin 才认为在相机的前面。

5. 编写 OrthographicCamera 类

类 OrthographicCamera 继承于纯虚类 Camera，所以需要实现其中的两个虚函数。

①编写 OrthographicCamera 类的数据成员。我设计了 6 个数据成员，分别是相机中心

（m_center）、眼睛看向方向（m_direction）、垂直方向（m_horizontal）、竖直方向（m_up）、屏幕大小（m_size）、相机最小值（m_tmin），具体可以参考图 5。

```
Vec3f m_center;     //相机中心
Vec3f m_direction;  //方向，指眼睛看向的方向
Vec3f m_horizontal; //垂直方向
Vec3f m_up;         //竖直方向
float m_size;       //屏幕大小
float m_tmin;       //相机最小值
```

图 5

② 编写 OrthographicCamera 类的构造函数。根据 SceneParser 中函数 parseOrthographicCamera()，可以判断出 OrthographicCamera 的构造函数参数类型（如图 6），需要传入相机中心、方向、垂直方向、屏幕大小。

```
camera = new OrthographicCamera(center,direction,up,size);
```

图 6

根据文档中所讲述的，需要对 direction 进行单位化，然后对 direction 和 up 进行点乘，判断两者是否正交，如果两者不正交，则需要进行相对应的处理，使其正交。我采用的方法是先将 direction 和 up 叉乘得到 m_horizontal,然后再使用 m_horizontal 和 direction 进行叉乘得到 m_up，最后再将 m_horizontal 和 m_up 进行单位化即可（在这里注意如果得出的图反了，可以将叉乘的顺序换一下）。除此之外在这里需要设置 m_tmin,因为是正交相机，光线总是始于无限远处，因此 tmin 应为一个很大的负数，在这里我设置为“-FLT_MAX”，具体可以参考图 7。

```
OrthographicCamera::OrthographicCamera(Vec3f center, Vec3f direction, Vec3f up, float size)
{
    m_center = center;
    m_direction = direction;
    m_direction.Normalize();

    if (direction.Dot3(up) == 0) {
        m_up = up;
        Vec3f::Cross3(m_horizontal, direction, up);//得到m_horizontal向量
    }
    else
    {
        Vec3f::Cross3(m_horizontal, direction, up);//得到m_horizontal向量
        Vec3f::Cross3(m_up, m_horizontal, direction);
    }
    m_up.Normalize();
    m_horizontal.Normalize();
    m_size = size;
    m_tmin = -FLT_MAX;  //这个值可以自己进行一定的设置
}
```

图 7

6. 编写主程序 main

主函数我编写一个“main.cpp”当中，里面我模仿了作业 31 当中的“demo0.cpp”的内容，然后需要在里面实现对 width 和 height 进行插值，来实现光线的生成。

①对 width 和 height 进行插值。我将插值的步长分别设置成 1/width 和 2/height，然后设

置一对参数 x，y 作为步长相加后的结果，这样就保证了（x，y）两点一定在（0，0）到（1，1）之间。
②写两层 for 循环，在最第二层中利用 x、y 生成光线，并且设置一个碰撞，然后对 group 进行判断是否有碰撞产生，如果产生碰撞，就对取出颜色，并且对相对应的位置的像素颜色，然后判断深度，如果距离 t 在显示深度的范围内，就需要对深度的 image 设置深度的颜色，我设置深度的颜色是采用了网上的方法进行设置，具体内容参考图 8

```
for (int i = 0; i < width; ++i,x+=1.0/ width) {
    y = 0;
    for (int j = 0; j < height; ++j,y+=1.0/ height) {

        //生成光线和碰撞
        Ray ray = camera->generateRay(Vec2f(x,y));
        Hit hit(FLT_MAX,NULL);

        if (group->intersect(ray,hit,camera->getTMin())) {
            Material* m = hit.getMaterial();
            image->SetPixel(i, j,m->getDiffuseColor());
            float t = hit.getT();

            //进行深度图显示显示
            if (depth_min <=t && t<=depth_max) {
                Vec3f  color = (depth_max-t) / (depth_max - depth_min) * white;//采用网上方法进行深度设置

                depthImage->SetPixel(i, j, color);
            }
        }
    }
}
```

图 8

二、实习中遇到的问题以及解决方法：

1.开始计算球交算法时，ppt 上给的算法是以球心在原点而展开的，我没有注意到这一点直接使用公式导致错误。
解决方式：思考了一下，通过向量转化的方式，通过光线起点坐标减去球心中心坐标就将琪转化到了以球心为中心的坐标系上，然后就可以继续使用公式

2.利用灰度颜色代表距离
解决方式：上网查询资料，发现一篇博客当中提供了计算方法，然后进行尝试了一下，发现真的有效。

三、实习结果：

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

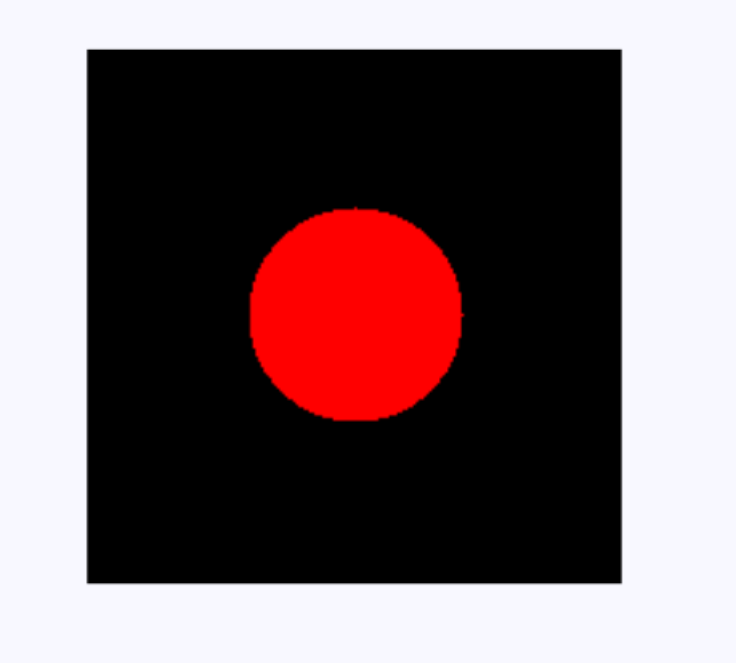

图 9

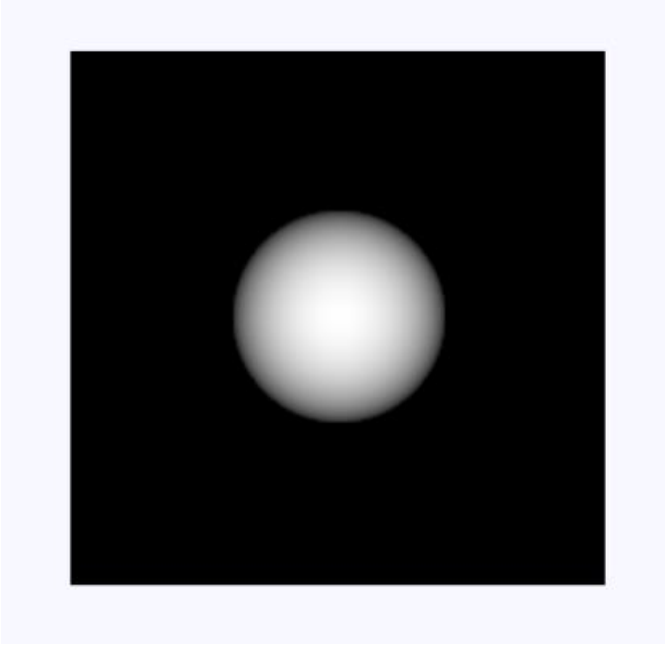

图 10

2.raytracer -input scene1_02.txt -size 200 200 -output output1_02.tga -depth 8 12 depth1_02.tga

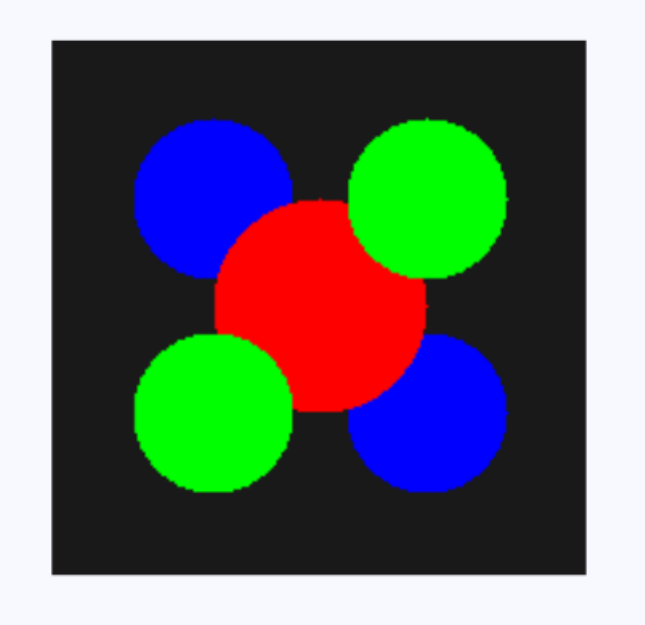

图 11

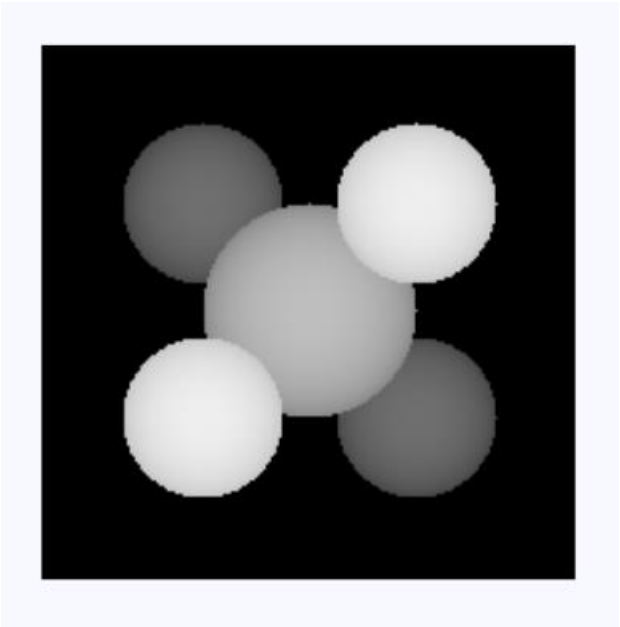

图 12

3. raytracer -input scene1_03.txt -size 200 200 -output output1_03.tga -depth 8 12 depth1_03.tga

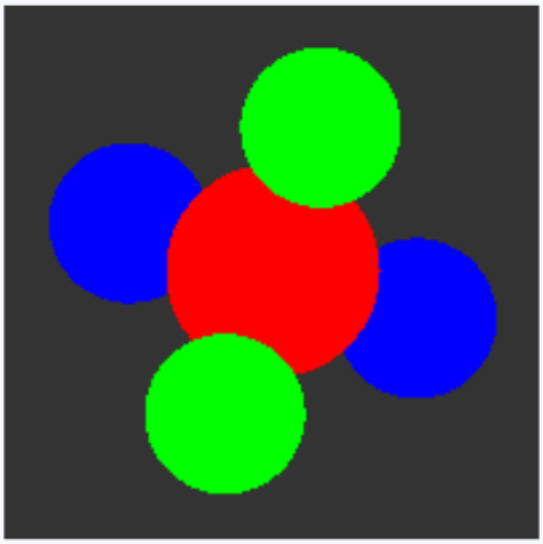

图 13

图 14

4.raytracer -input scene1_04.txt -size 200 200 -output output1_04.tga -depth 12 17 depth1_04.tga

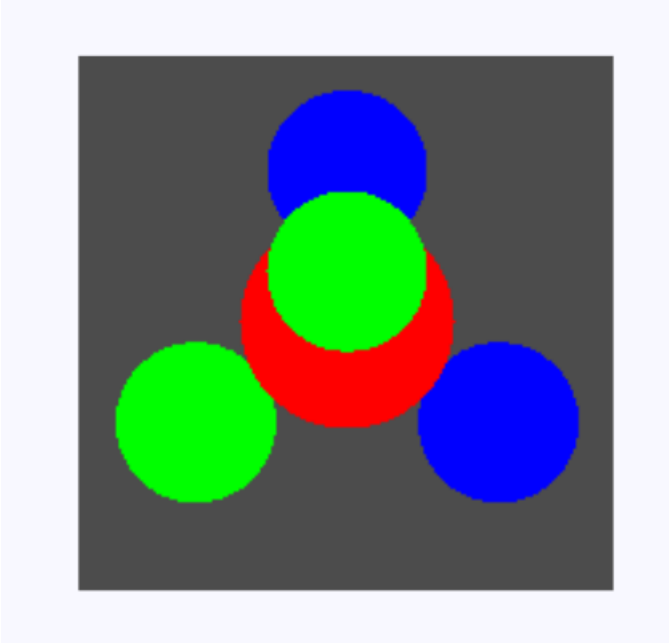

图 15

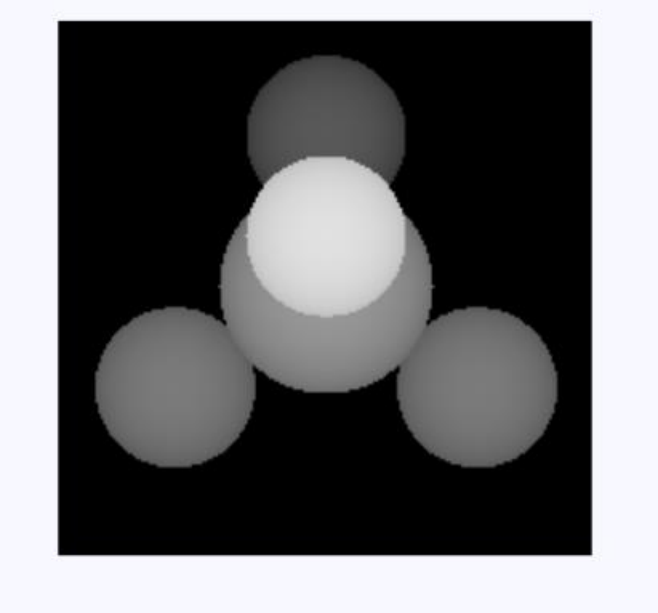

图 16

5.raytracer -input scene1_05.txt -size 200 200 -output output1_05.tga -depth 14.5 19.5 depth1_05.tga

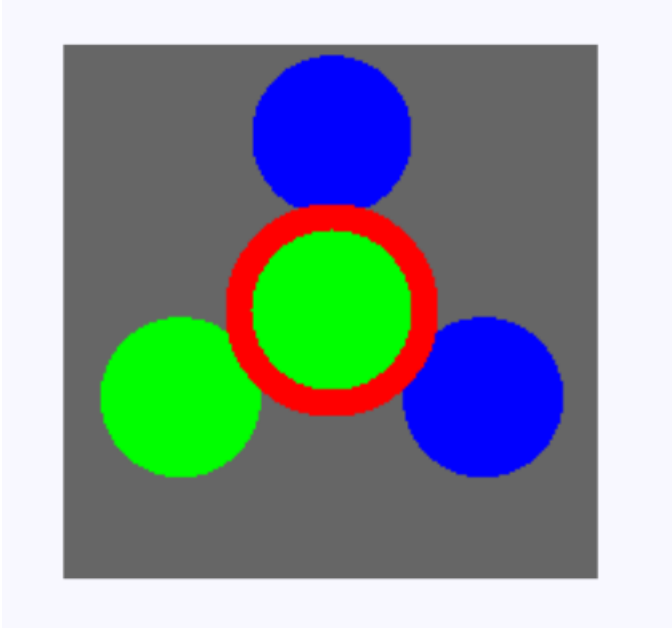

图 17

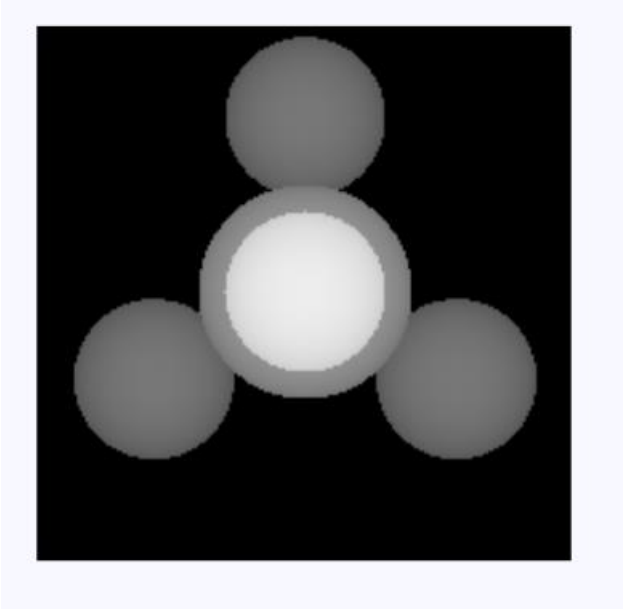

图 18

6. raytracer -input scene1_06.txt -size 200 200 -output output1_06.tga -depth 3 7 depth1_06.tga

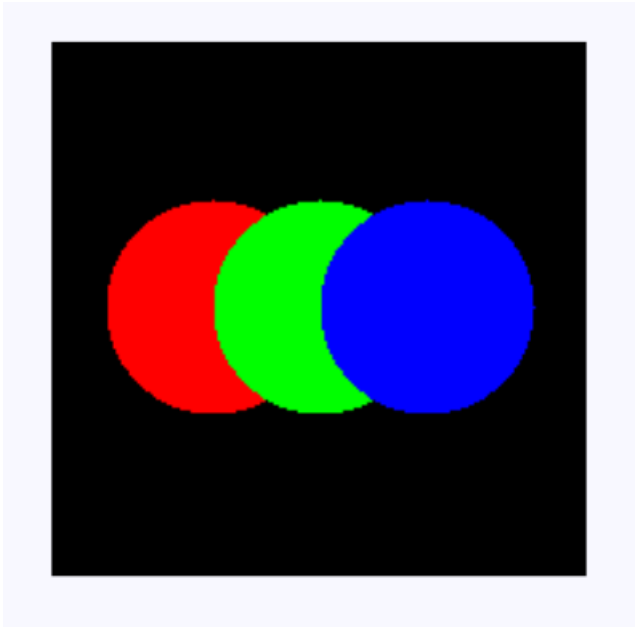

图 19

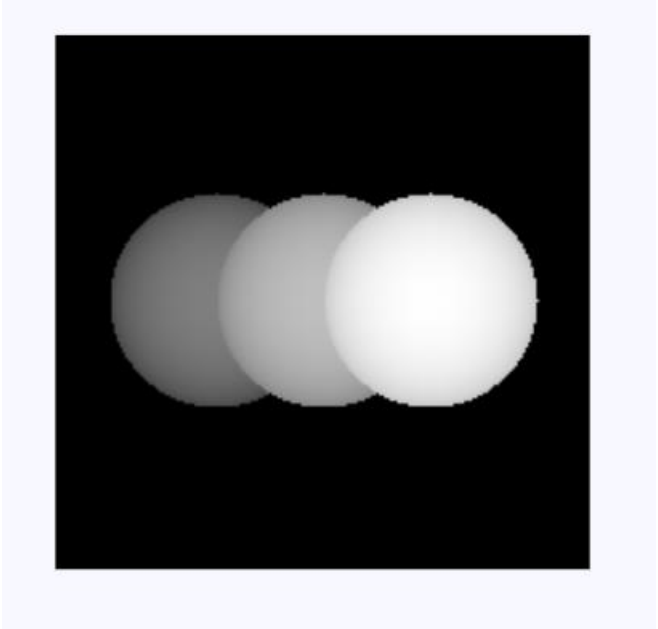

图 20

7.raytracer -input scene1_07.txt -size 200 200 -output output1_07.tga -depth -2 2 depth1_07.tga

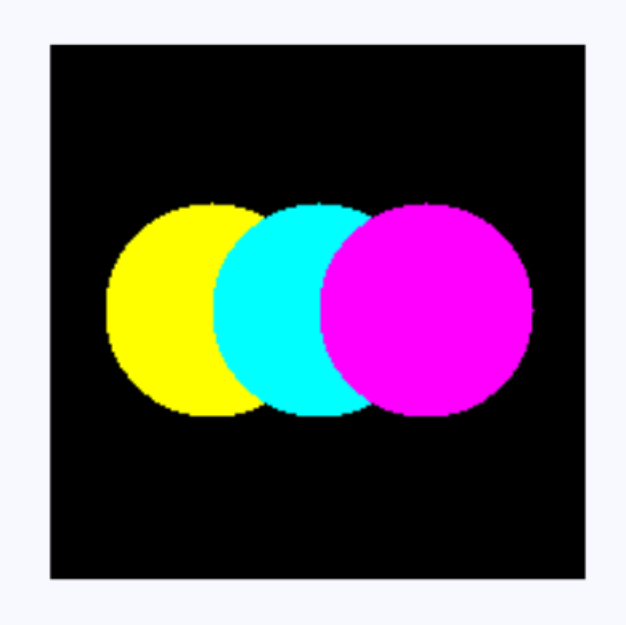

图 21

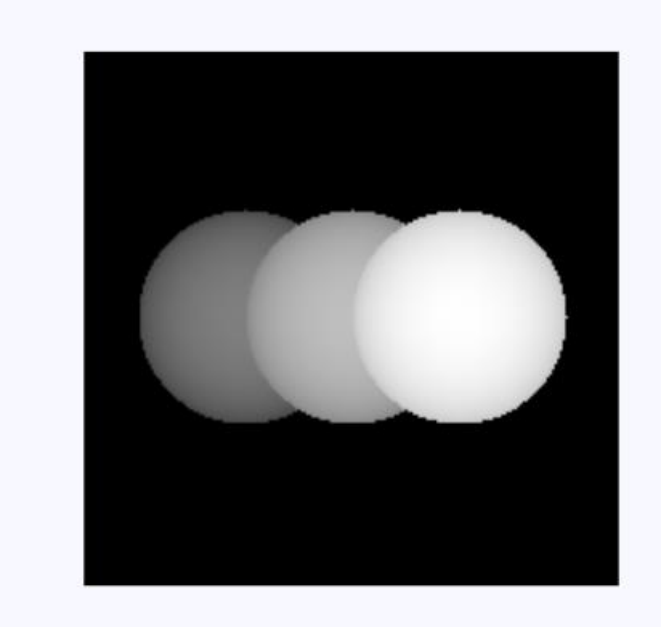

图 22

四、实习总结

本次实习主要是考察光线投射中的球交算法，比较基础难度不大，相比于下一个作业就更为简单了，但是这个是基础内容，完成了这个为下一个作业打好了基础，是一个不错的开端。