# 作业5



本次作业框架理解起来有些偏难，所以笔者打算先进行代码框架分析，再进行代码编写，争取尽可能地理解代码框架，熟悉光线追踪的过程

# 代码框架分析

## global.hpp

### clamp()

限制值v在[lo, hi]范围内

### solveQuadratic()

根据传入的一元二次方程未知数参数a，b，c 求出该方程的解，并返回bool值判断方程是否有解

### enum MaterialType

枚举材料的种类

### get_random_float()

提供[0,1]的平均分布的随机数

### UpdateProgress()

显示当前进程完成率

## Vector.hpp

没有使用Eigen库的Vector3f和Vector2f，而是自定义了其相关实现和操作运算符

## Light.hpp

定义了Light类，其成员变量为Vector3f类型的position和Vector3f类型的intensity

## Object.hpp

定义了Object类，其成员变量包括材质、折射率、漫反射系数、镜面反射系数、漫反射颜色、镜面反射衰减指数

## Triangle.hpp

定义了继承于Object类的MeshTriangle类，有初始化MeshTriangle()函数、判断光线与三角形相交rayTriangleIntersect()函数、intersect()函数、得到三角形表面属性函数getSurfaceProperties()函数、估计漫反射颜色evalDiffuseColor()函数

## Sphere.hpp

定义了继承于Object类的Sphere类

## Renderer.cpp

### reflect()

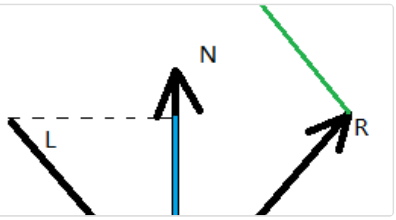
镜面反射的出射光线计算

### refract()

光在不同折射率介质间传播的折射光线计算

光的反射与折射的向量计算

在raytrace的计算中，光反射与折射的计算是两个非常重要的过程，而由于我们在图形学中通常都使用向量来计算，所以也有必要找到一套计算光反射和折射模型的向量计算方法。(该方法来自Bram de Greve的文章"Reflections and Refractions in

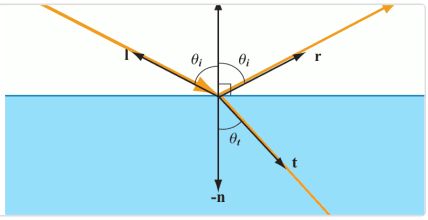



https://www.cnblogs.com/starfallen/archive/2012/11/05/2754992.html#:~:text=2.%E6%8A%98%E5%B0%84%EF%BC%8C%E6%8A%98%E5%B0%84%E7%9A%84%E8%AE%A1%E7%AE%97%E8%A6%81%E6%AF%94%E5%8F%8D%E5%B0%84%E5%A4%8D%E6%9D%82%E5%A4%9A%E4%BA%86%EF%BC%8C%E9%A6%96%E5%85%88%E8%A6%81%E7%9F%A5%E9%81%93%E6%8A%98%E5%B0%84%E5%AE%9A%E5%BE%8B%EF%BC%8C%E5%85%A5%E5%B0%84%E8%A7%92%CE%B81%E4%B8%8E%E6%8A%98%E5%B0%84%E8%A7%92%CE%B82%E6%BB%A1%E8%B6%B3sin%CE%B81%2Fsin%CE%B82%3D%CE%B72%2F%CE%B71%2C%E5%85%B6%E4%B8%AD%CE%B71%EF%BC%8C%CE%B72%E5%88%86%E5%88%AB%E4%B8%BA%E5%85%A5%E5%B0%84%E5%85%89%E4%B8%8E%E6%8A%98%E5%B0%84%E5%85%89%E6%89%80%E5%9C%A8%E7%A9%BA%E9%97%B4%E7%9A%84%E4%BB%8B%E8%B4%A8%E7%9A%84%E6%8A%98%E5%B0%84%E7%8E%87%EF%BC%8C%E5%8F%AF%E4%BB%A5%E5%8F%96%CE%B7%3D%CE%B72%2F%CE%B71%E4%B8%BA%E7%9B%B8%E5%AF%B9%E6%8A%98%E5%B0%84%E7%8E%87%EF%BC%8C%E8%80%8C%E6%88%91%E4%BB%AC%E7%9F%A5%E9%81%93%E7%9A%84%E5%B7%B2%E7%9F%A5%E9%87%8F%E6%98%AFN%20%28%E7%89%A9%E4%BD%93%E8%A1%A8%E9%9D%A2%E5%8D%95%E4%BD%8D%E6%B3%95%E5%90%91%E9%87%8F%29%EF%BC%8CL,%28%E5%85%89%E7%9A%84%E5%85%A5%E5%B0%84%E5%8D%95%E4%B

D%8D%E5%90%91%E9%87%8F%29%EF%BC%8C%CE%B7%20%28%E7%9B%B8%E5%AF%B9%E6%8A%98%E5%B0%84%E7%8E%87%29%EF%BC%8C%E4%B8%8B%E9%9D%A2%E5%B0%B1%E6%9D%A5%E6%B1%82%E8%A7%A3%E5%8D%95%E4%BD%8D%E6%8A%98%E5%B0%84%E5%90%91%E9%87%8FT%E3%80%82

### fresnel()

计算菲涅尔项




光的反射与折射--从Snell、Fresnel到Schlick

站在岸边，朝平静的湖面望去，近处的水面能看清水底，远处的水面则铺上倒影。在清澈的湖中划小船也是一样的效果。近处多参与折射，远处多参与反射，这种效果叫 菲涅尔效应 （Fresnel Effect）。这种现象与湖面的深浅没什么关系，与视角有

知 https://zhuanlan.zhihu.com/p/303168568

### trace()

如果光线遇到了最近的物体，则返回相交的相关信息

### castRay()

迭代光线追踪

# 代码编写

代码框架只有两个部分需要补全

- 第一部分，将屏幕空间的中[i, j]索引对应的像素点映射到Canonical Cube，需要注意[i, j]对应的像素需要加上0.5f，并且屏幕空间的y轴方向是从上往下的，所以编写代码如下：

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

    float scale = std::tan(deg2rad(scene.fov * 0.5f));
    float imageAspectRatio = scene.width / (float)scene.height;

    // Use this variable as the eye position to start your rays.
    Vector3f eye_pos(0);
    int m = 0;
    for (int j = 0; j < scene.height; ++j)
    {
        for (int i = 0; i < scene.width; ++i)
        {
            // generate primary ray direction
            float x;
            float y;
            // 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*

            x = (((i+0.5f)/((float)scene.width) * 2.f)-1.f) * imageAspectRatio * scale;
            y = (1-(j+0.5f)/(float)scene.height*2)*scale;

            Vector3f dir = normalize(Vector3f(x, y, -1)); // Don't forget to normalize this direction!
            framebuffer[m++] = castRay(eye_pos, dir, scene, 0);
```

```
        }
        UpdateProgress((j+1) / (float)scene.height);
    }
```

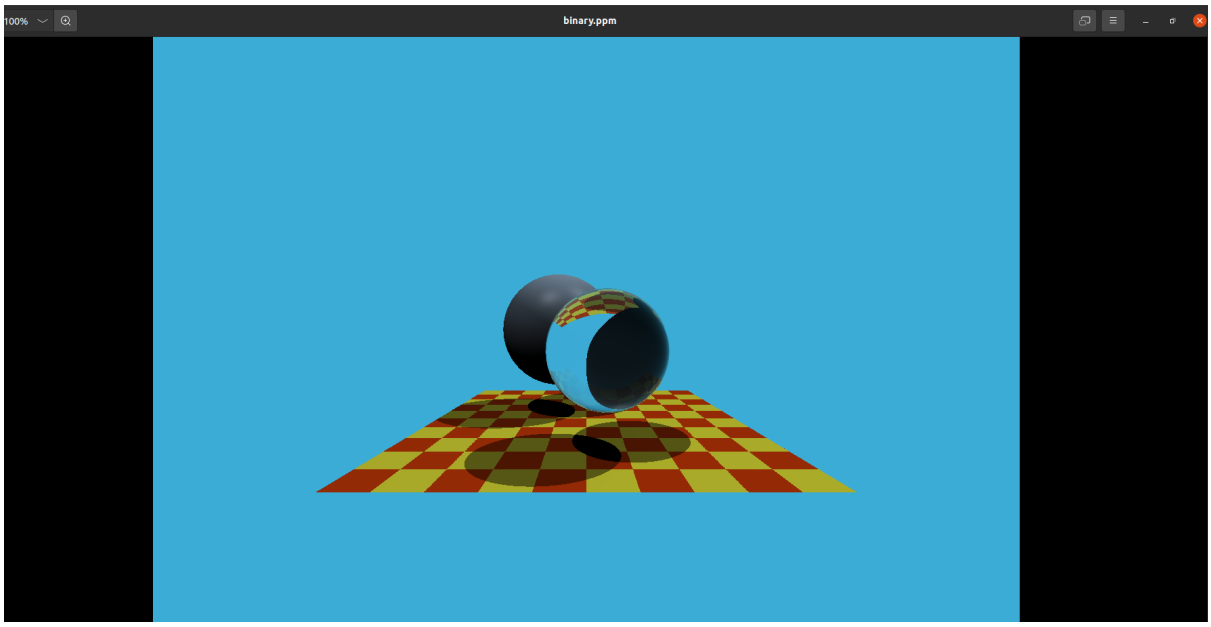
- 第二部分，需要完成的功能为判断光线与三角形相交，并得到交点关于该三角形的重心坐标，注意最近的交点用直线方程来表示，需要满足条件 `t > 0 && u > 0 && v > 0 && u - v ≥ 0` ，但是由于计算机在判断float类型的小数与int类型的整数0是否相等时，是存在问题的，所以笔者将条件修改为 `1.f-u-v>-__FLT_EPSILON__` ， `__FLT_EPSILON__` 表示float类型所能表示的最小精度，带上一个负号即可表示float类型所能表示的最大负数，这样渲染出来的图片就不会出现蓝点 (也就是背景颜色) 的情况，代码如下：

```
bool rayTriangleIntersect(const Vector3f& v0, const Vector3f& v1, const Vector3f& v2, const Vector3f& orig,
                          const Vector3f& dir, float& tnear, float& u, float& v)
{
    // TODO: Implement this function that tests whether the triangle
    // that's specified bt v0, v1 and v2 intersects with the ray (whose
    // origin is *orig* and direction is *dir*)
    // Also don't forget to update tnear, u and v.

    Vector3f E1 = v1 - v0;
    Vector3f E2 = v2 - v0;
    Vector3f S = orig - v0;
    Vector3f S1 = crossProduct(dir,E2);
    Vector3f S2 = crossProduct(S,E1);

    float SE = dotProduct(S1,E1);
    if(SE<=0)
        return false;
    tnear = dotProduct(S2,E2)/SE;
    u = dotProduct(S1,S)/SE;
    v = dotProduct(S2,dir)/SE;

    if(tnear > 0 && u > 0 && v > 0 && (1.f-u-v>-__FLT_EPSILON__))
        return true;
    return false;
}
```



至此，作业5完成！