一、实验步骤：

1、为 phong material 类增加反射颜色和透明颜色的变量及其外部接口（代码略）。

2、构造 ray tracer 类，计算单条射线的交点颜色：

```
class RayTracer {
public:
    RayTracer(){}
    RayTracer(SceneParser *s, int m_bounces, float c_weight, bool shads, bool back) {
        scene = s;
        max_bounces = m_bounces;
        cutoff_weight = c_weight;
        shadows = shads;
        shade_back = back;
    }
    ~RayTracer(){}

    Vec3f traceRay (Ray &ray, float tmin, int bounces, float weight, float indexOfRefraction,
Hit &hit) const {
        if (bounces > max_bounces || weight < cutoff_weight) {
            Vec3f zero(0, 0, 0);
            return zero;
        }
        int k;
        Light *li;
        Vec3f ldir, clit, pin;
        Vec3f color(0,0,0);
        float Dis2Lit;
        Vec3f n0(0, 0, 0), normal;
        Group *gro = scene->getGroup();

        if (gro->intersect(ray, hit, tmin)) {

        //pass

        }
        else {
            color = scene->getBackgroundColor();
        }
        return color;
    }
private:
    SceneParser *scene;
    int max_bounces;
    float cutoff_weight;
    bool shadows;
    bool shade_back;
};
```

traceRay 函数中传入的 bounce 表示是经过几次折射或阴影或反射后的射线，weight 表示权重，indexOfRefraction 用于折射。

3、实现阴影颜色的计算，从表面上方一点朝光源发射射线：

```
            normal = hit.getNormal();
            pin = hit.getIntersectionPoint();
            if (normal.Dot3(ray.getDirection()) > 0 && shade_back == 1)
                hit.set(hit.getT(), hit.getMaterial(), -1 * normal, ray);

            if (bounces == 0) RayTree::SetMainSegment(ray, tmin, hit.getT());  //*

            color = scene->getAmbientLight() * hit.getMaterial()->getDiffuseColor();
            int num_lights = scene->getNumLights();
            for (k = 0; k < num_lights; k++) {
                li = scene->getLight(k);
                li->getIllumination(pin, ldir, clit, Dis2Lit);
                if (shade_back) pin = pin + ldir * epsilon;//这里是 epsilon
                Ray ray2(pin, ldir);
                Hit hit2(Dis2Lit, scene->getMaterial(0), n0);

                Group *gro = scene->getGroup();
                if (shade_back) gro->intersect(ray2, hit2, 0);
                else gro->intersect(ray2, hit2, epsilon);
                if (shadows == 0) {
                    color += (hit.getMaterial())->Shade(ray, hit, ldir, clit);
                }else if (hit2.getT()==INFINITY||posive(hit2.getT()-Dis2Lit)<0.0001) {
                    color += (hit.getMaterial())->Shade(ray, hit, ldir, clit);
                }
                RayTree::AddShadowSegment(ray2, 0, hit2.getT());           //*
            }
```

考虑无遮挡的条件：当声明 shadow 后，向光源发出去的射线可以到达光源。即在方向光的情况下 hit 的 t 值为无穷，在点光源的情况下，hit 的 t 值等于到点光源的距离（注意 float 类型变量作等于比较时不能用==）。注释中加*的行在后面会进行说明。

4、递归地调用函数，增加反光颜色的显示：

```
             Vec3f reflectColor = hit.getMaterial()->getReflectiveColor();
            if (reflectColor.Length() > 0) {
                Vec3f rdir = mirrorDirection(normal, ray.getDirection());
                Vec3f rori = hit.getIntersectionPoint();
                rori = rori + rdir * epsilon;//先移动 epsilon

                Ray reray(rori, rdir);
                Hit rhit;
                float rweight = weight * reflectColor.Length();
                rhit.set(MAXnum, scene->getMaterial(0), n0, reray);
                color += reflectColor *
                    traceRay(reray, 0, bounces + 1, rweight, indexOfRefraction, rhit);
                RayTree::AddReflectedSegment(reray, 0, rhit.getT());      //*
            }
```

关于 mirrorDirection 函数：

```
Vec3f mirrorDirection(const Vec3f &normal, const Vec3f &incoming) {
    float cosa = -1 * incoming.Dot3(normal);
    Vec3f reflect;
    reflect = incoming + 2 * cosa * normal;
    reflect.Normalize();
    return reflect;
}
```

注意在构造反射光线的时候，要将 origin 移动一小段距离。新的权重计算要乘上反射系数。

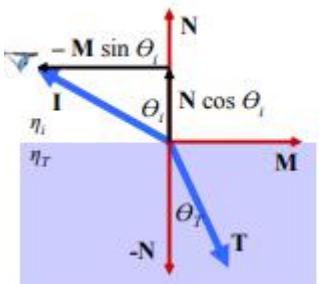
5、增加折射颜色的显示：

```
    Vec3f transColor = hit.getMaterial()->getTransparentColor();
    if (transColor.Length() > 0) {
        Vec3f tdir;
        Vec3f tori = hit.getIntersectionPoint();
        Vec3f income = -1 * ray.getDirection();
        float index_i, index_t;
        if (normal.Dot3(income) > 0) {
            index_i = 1;
            index_t = hit.getMaterial()->getIndexOfRefraction();
        }
        else {
            index_t = 1;
            index_i = indexOfRefraction;
            normal *= -1;
        }

        if (transmittedDirection(normal, income, index_i, index_t, tdir)) {
            tori = tori + tdir * epsilon;
            Ray tray(tori, tdir);
            Hit thit;
            float tweight = weight * transColor.Length();
            float ior = hit.getMaterial()->getIndexOfRefraction();
            thit.set(MAXnum, scene->getMaterial(0), n0, tray);
            color += transColor *
                traceRay(tray, 0, bounces + 1, tweight, ior, thit);
            RayTree::AddTransmittedSegment(tray, 0, thit.getT());  //*
        }

    }
```

要分清楚射线从物体内部向外发出还是从外部向里发出，对应的 index 选择和 normal 的方向要作出调整。关于 transmittedDirection 函数：

```
bool transmittedDirection(const Vec3f &normal, const Vec3f &incoming, float index_i, float index_t, Vec3f &transmitted) {
    float cosa = normal.Dot3(incoming);
    float nr = index_i / index_t;
    float delt = 1 - nr * nr * (1 - cosa * cosa);
    if (delt <= 0) return 0;
    float coN = nr * cosa - sqrtf(delt);
    transmitted = coN * normal - nr * incoming;
    return 1;
}
```

基本原理：



$$\mathbf{I} = \mathbf{N}\cos\theta_i - \mathbf{M}\sin\theta_i$$
$$\mathbf{M} = (\mathbf{N}\cos\theta_i - \mathbf{I})/\sin\theta_i$$

$$\begin{aligned}\mathbf{T} &= -\mathbf{N}\cos\theta_T + \mathbf{M}\sin\theta_T \\ &= -\mathbf{N}\cos\theta_T + (\mathbf{N}\cos\theta_i - \mathbf{I})\sin\theta_T/\sin\theta_i \\ &= -\mathbf{N}\cos\theta_T + (\mathbf{N}\cos\theta_i - \mathbf{I})\eta_r \\ &= [\eta_r\cos\theta_i - \cos\theta_T]\,\mathbf{N} - \eta_r\mathbf{I} \\ &= [\eta_r\cos\theta_i - \sqrt{1-\sin^2\theta_T}\,]\,\mathbf{N} - \eta_r\mathbf{I} \\ &= [\eta_r\cos\theta_i - \sqrt{1-\eta_r^2\sin^2\theta_i}\,]\,\mathbf{N} - \eta_r\mathbf{I} \\ &= [\eta_r\cos\theta_i - \sqrt{1-\eta_r^2(1-\cos^2\theta_i)}\,]\,\mathbf{N} - \eta_r\mathbf{I} \\ &= [\eta_r(\mathbf{N}\cdot\mathbf{I}) - \sqrt{1-\eta_r^2(1-(\mathbf{N}\cdot\mathbf{I})^2)}\,]\,\mathbf{N} - \eta_r\mathbf{I}\end{aligned}$$

Snell-Descartes Law:

$$\eta_i\sin\theta_i = \eta_T\sin\theta_T$$

$$\frac{\sin\theta_T}{\sin\theta_i} = \frac{\eta_i}{\eta_T} = \eta_r$$

- **Total internal reflection when the square root is imaginary**
- **Don't forget to normalize!**

6、加入 RayTree 相关函数方便检查光线追踪效果（即前面代码里注释为*的行），并为 opengl 的初始化加入新的参数：

```
    SceneParser s(input_file);
    scene = &s;
    if (dogl == 1) {
        GLCanvas glrender;
        glutInit(&argc, argv);
        glrender.initialize(scene, &Paint, &traceRay);
    }
    else {
        Paint();
    }
```

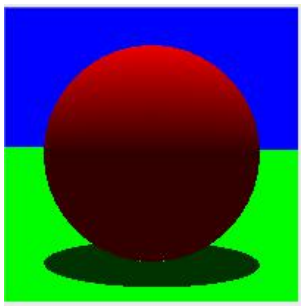
TraceRay 函数：

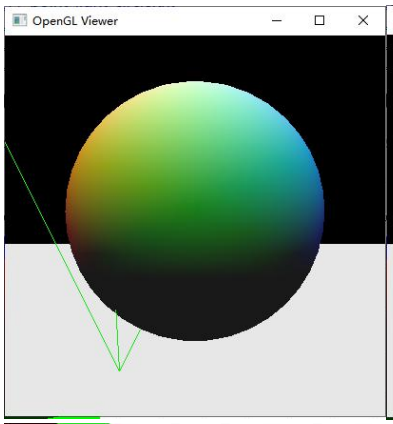
```
void traceRay(float x, float y) {
    RayTracer raytracer(scene, max_bounce, cutoff_weight, shadows, shade_back);
    Vec3f n0(0, 0, 0);
    Hit h(MAXnum, scene->getMaterial(0), n0);
    Ray r;
    Camera *cam = scene->getCamera();
    Vec2f position;
    position.Set(x, y);

    r = cam->generateRay(position);
    raytracer.traceRay(r, 0, 0, 1, 1, h);
}
```

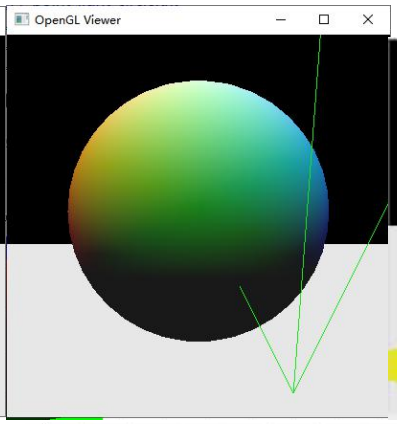
对 Paint 的修改：

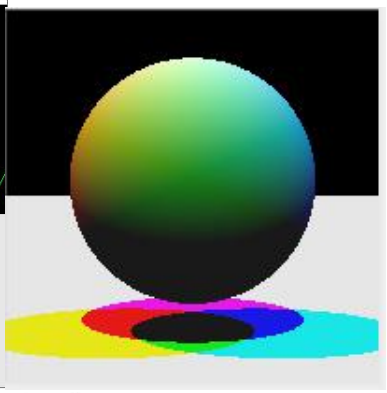
```
    Image outimg(width, height);
    outimg.SetAllPixels(scene->getBackgroundColor());
    RayTracer raytracer(scene, max_bounce, cutoff_weight, shadows, shade_back);

    float depth;
    int i, 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), n0, r);
            color = raytracer.traceRay(r, tmin, 0, 1, 1, h);
            outimg.SetPixel(i, j, color);

        }
    }
    outimg.SaveTGA(output_file);
```

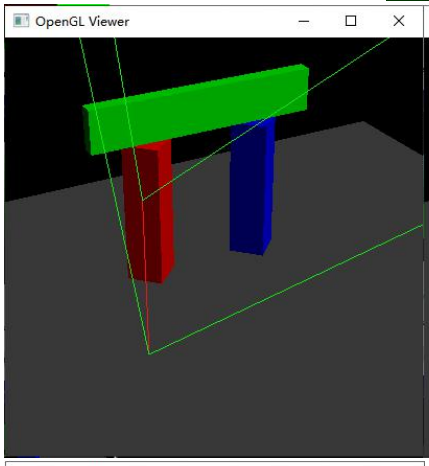
二、实验结果

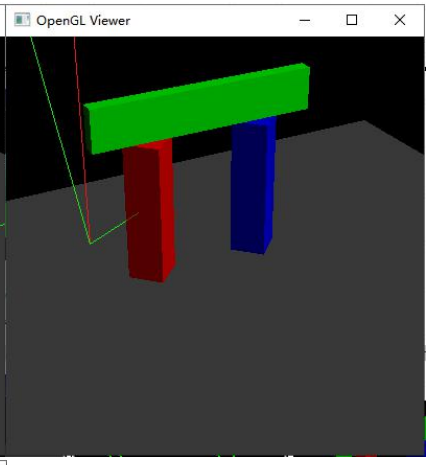


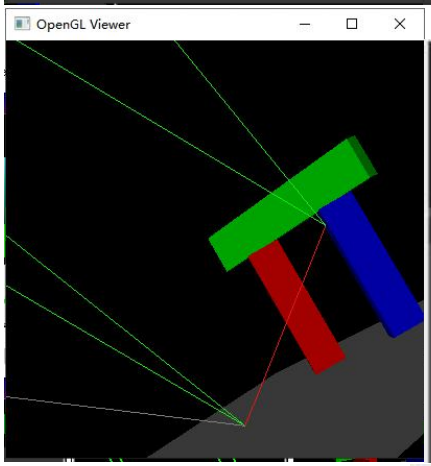

OpenGL Viewer

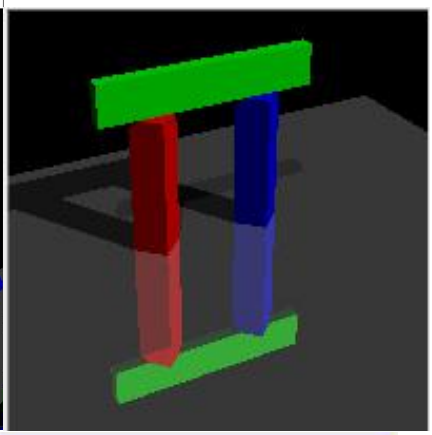

OpenGL Viewer

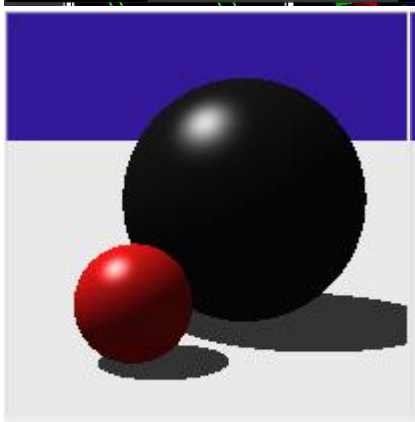


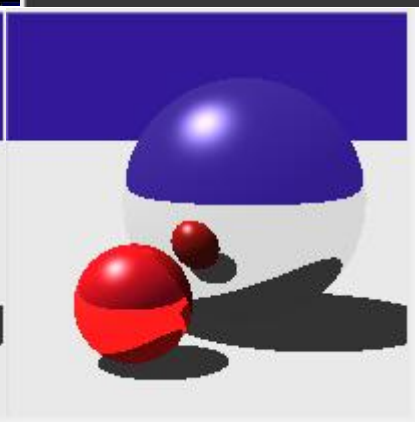

OpenGL Viewer

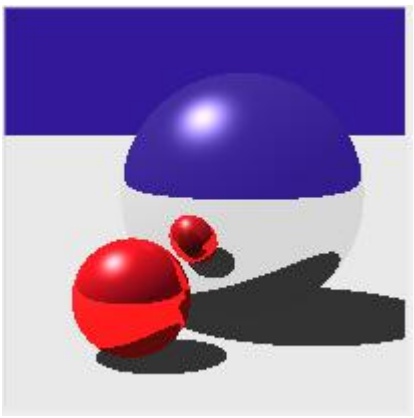

OpenGL Viewer

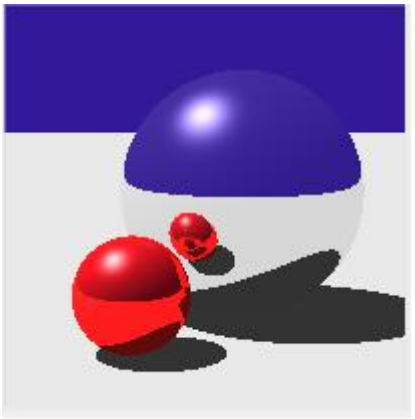

OpenGL Viewer

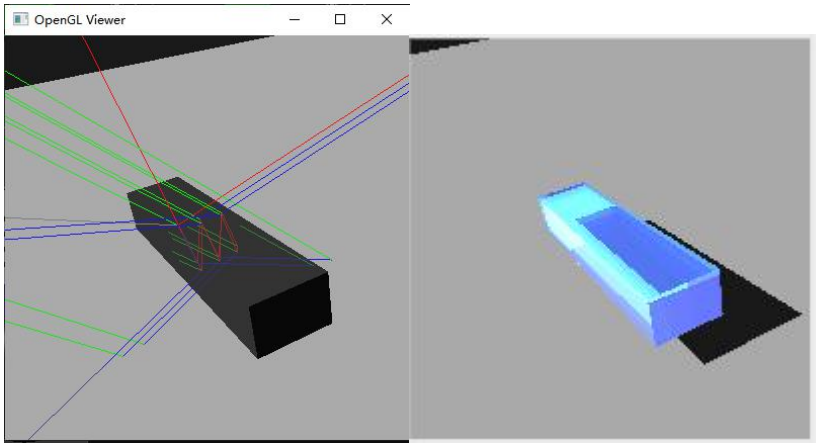


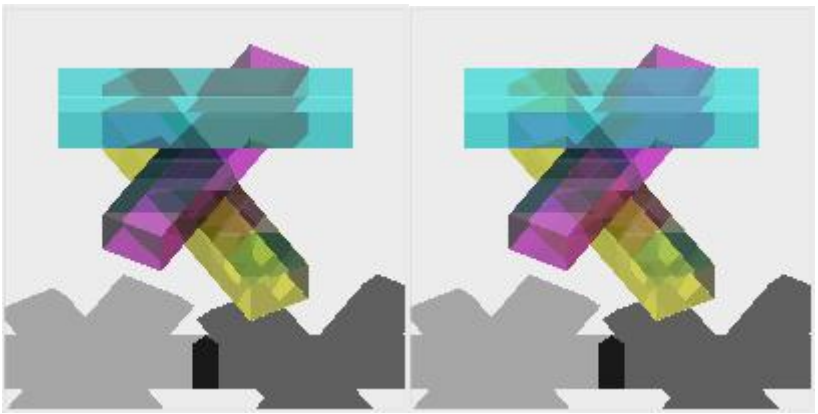


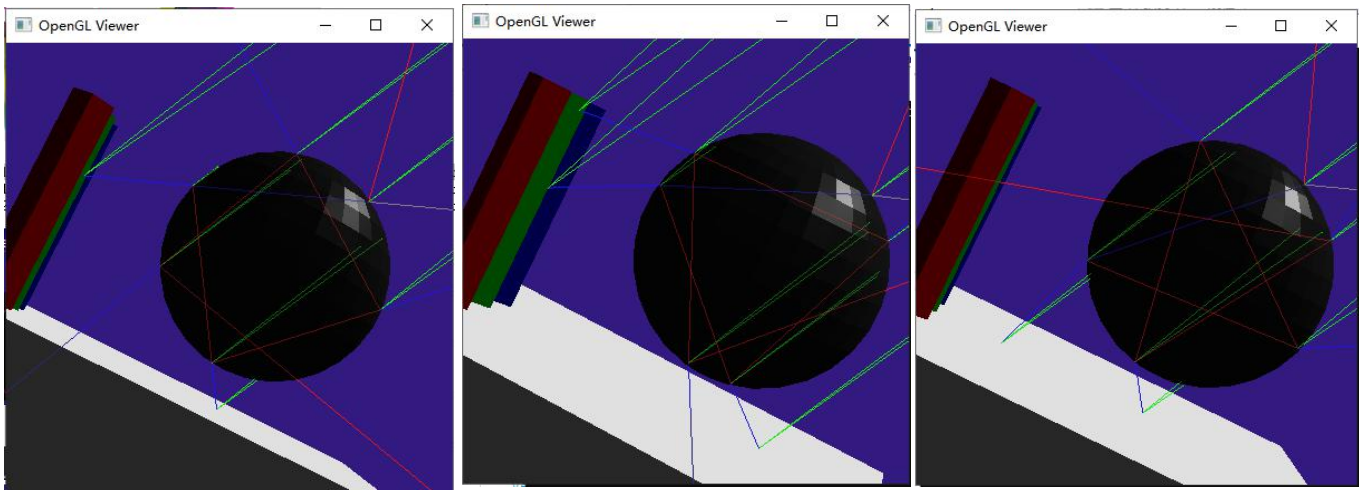


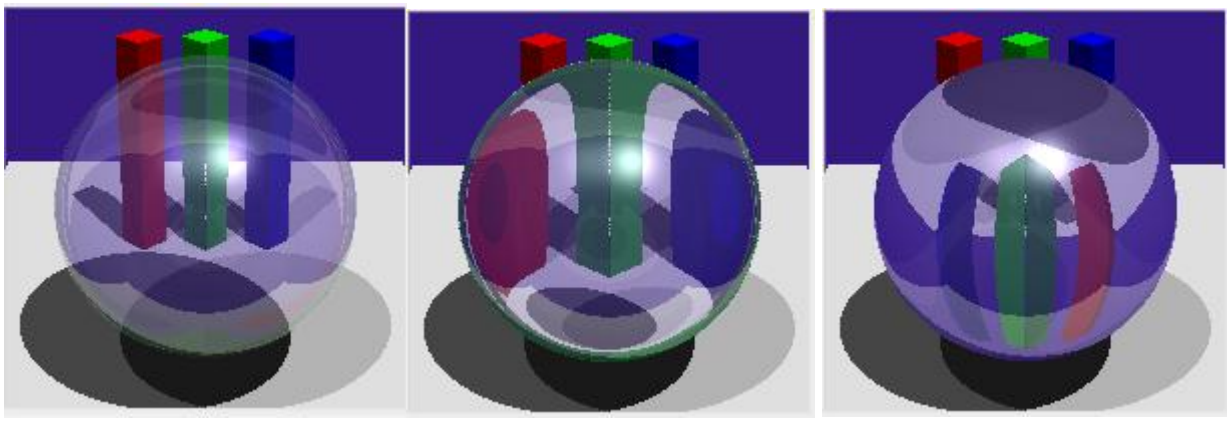


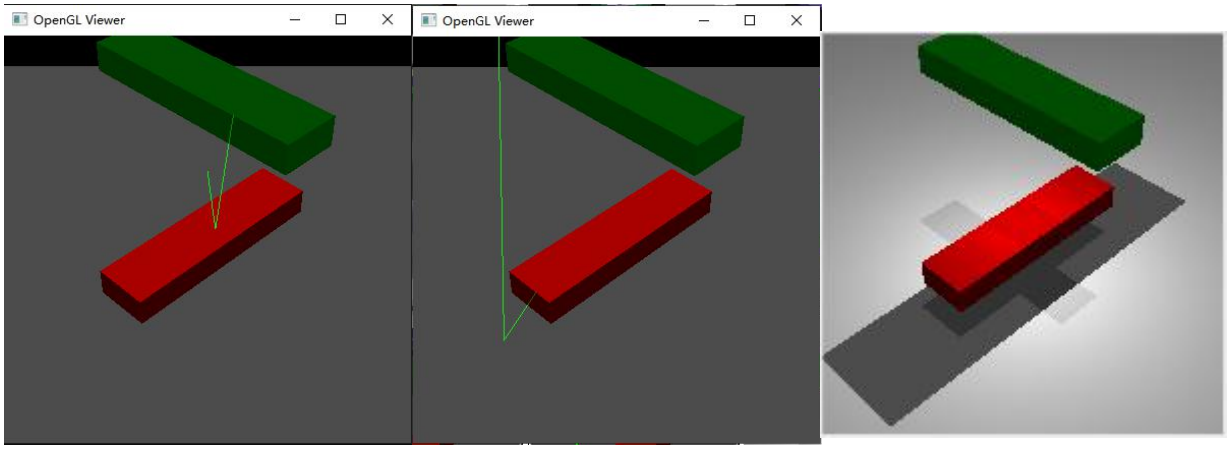


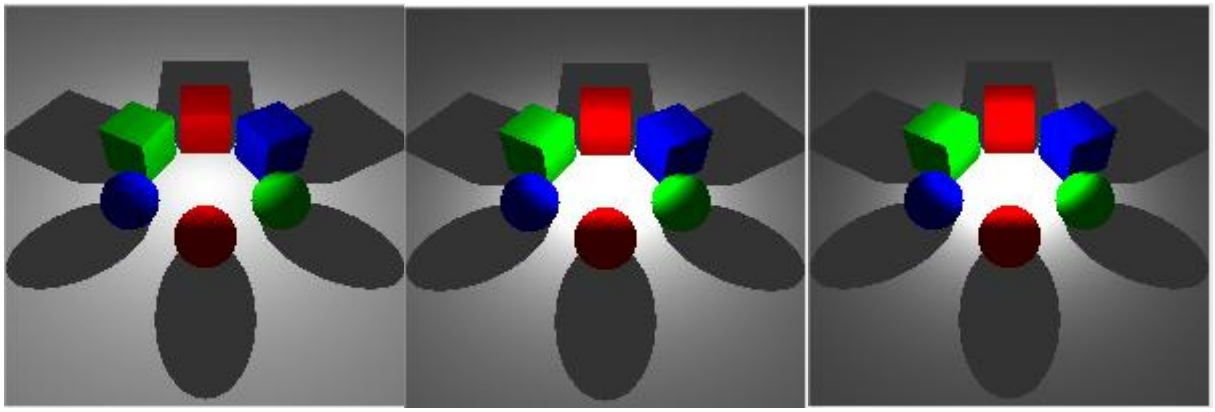

OpenGL Viewer

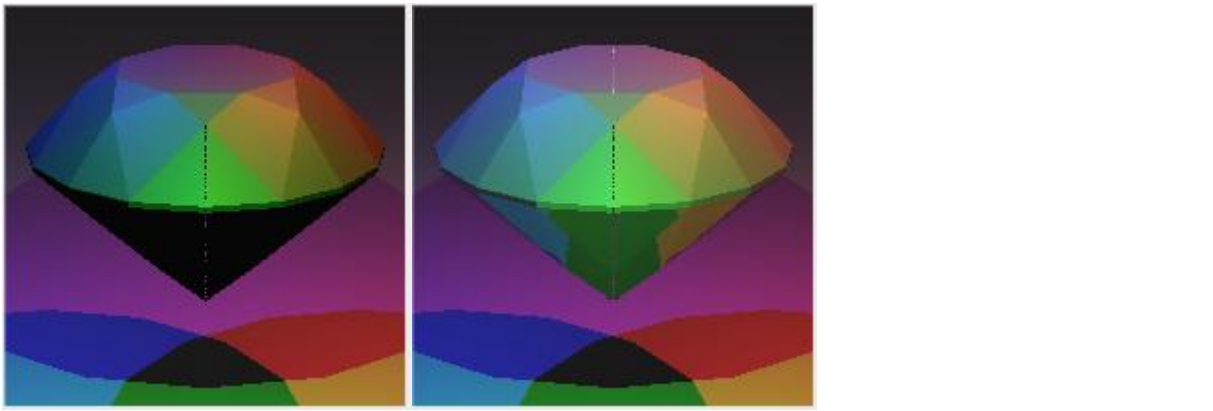


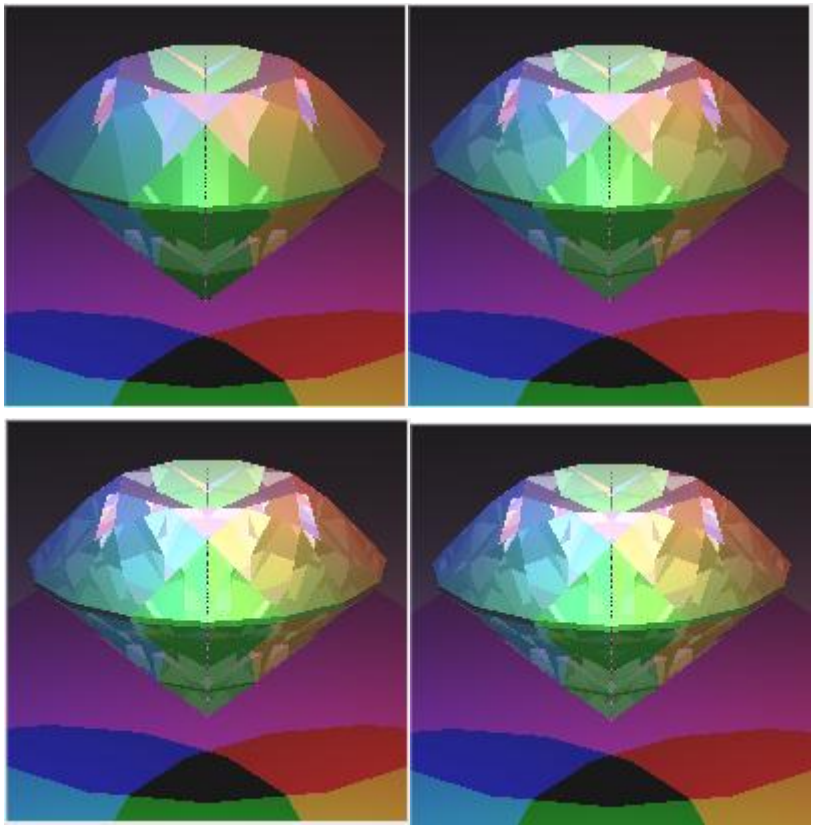

OpenGL Viewer
OpenGL Viewer
OpenGL Viewer

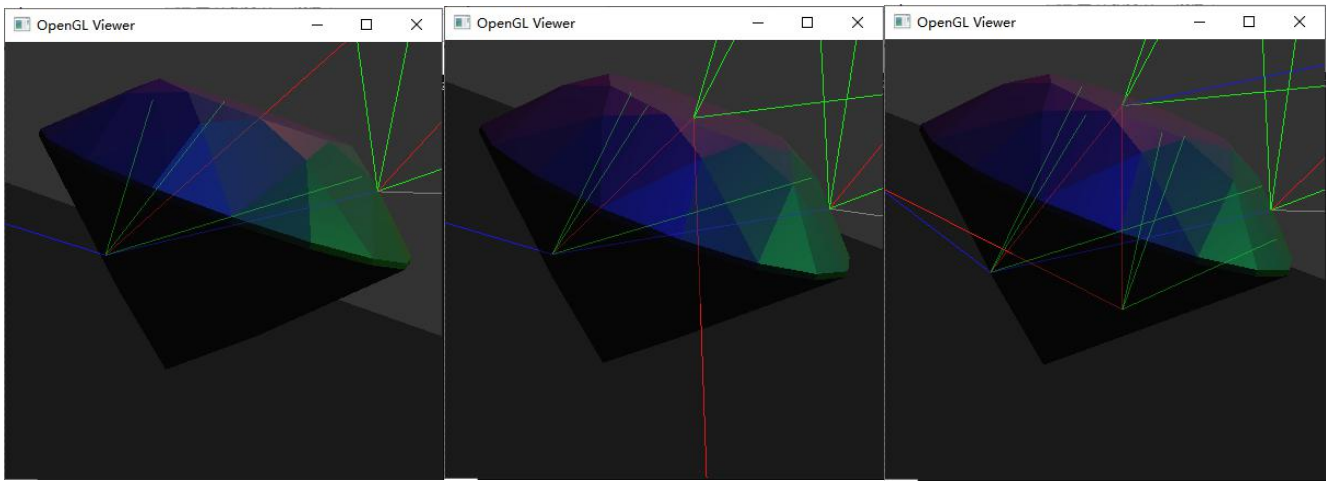



OpenGL Viewer
OpenGL Viewer









三、实验心得

关于 ray tracer 的实现步骤，每一步都给出了较为详细的指导，做起来也是较为顺利的，关键在于小量的把控：origin 要从交点处移一定的距离，太近的话没有效果，太远了又会产生更奇怪的结果（比如球体和平面相交的情况），所以要多尝试几次；另外 float 的判断我一开始用了==，发现没有效果，后来将小量设置和前面一样，发现产生的结果还是不理想，最后单独又设置了一个小量进行了调整就好了。

阴影部分的 shade_back 的调整需要多考虑考虑，可以结合 raytree 的功能来辅助修改代码。