

一、实验步骤:

1、对 sphere 类 intersect 方法里增加法线的返回（使用新的 hit 类）:

主要代码:

```
Vec3f normal = r.pointAtParameter(t);
normal = normal - center;
normal.Normalize();
h.set(t, mat, normal, r);
```

2、在主函数中进行修改，添加 normal 和 shade_back 功能（添加 light.h）:

主要代码:

```
position.Set(1.0*i/width, 1.0*j/width);
r = cam->generateRay(position);
h.set(MAXnum, scene.getMaterial(0), n0, r);

if (gro->intersect(r, h, tmin)) {
    pinter = h.getIntersectionPoint();
    normal = h.getNormal();
    if (normal.Dot3(r.getDirection()) > 0
        && shade_back == 1) normal *= -1;
    cpro = h.getMaterial()->getDiffuseColor();
    color = cpro;
    color = scene.getAmbientLight() * color;
    int num_lights = scene.getNumLights();
    for (k = 0; k < num_lights; k++) {
        li = scene.getLight(k);
        li->getIllumination(pinter, ldir, clit);
        d_clamped = ldir.Dot3(normal);
        if (d_clamped <= 0) d_clamped = 0;
        color += d_clamped * clit * cpro;
    }

    outimg.SetPixel(i, j, color);

    if (depth_file != NULL) {
        depth = (depth_max - h.getT()) / (depth_max - depth_min);
        depcolor.Set(depth, depth, depth);
        depimg.SetPixel(i, j, depcolor);
    }
    if (normals_file != NULL) {
        Vec3f normal_d = makepos(normal);
        norimg.SetPixel(i, j, normal_d);
    }
}
```

注意不要忘记 ambient 影响的颜色，将光线与物体法线角度大于 90 度的情况删去（因为本算法中法线指向物体内部，所以角度大于 90 度表示此时在物体背面相交），生成法线图片时要将向量的三个分量置为正数。

3、构造透视相机的类:

```
class PerspectiveCamera : public Camera
{
public:
    PerspectiveCamera(Vec3f cer, Vec3f &dir, Vec3f &upp, float ang) {
```

```

        center = cer;
        direction = dir;
        direction.Normalize();

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

        tmp.Cross3(horizontal, direction, up);
        angle = ang;
    }
    ~PerspectiveCamera() {}
    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;
        yray = up * y;

        float a, c;
        c = 0.5 / tanf(angle / 2);
        a = sqrtf(c*c - 0.5*0.5);
        position = center + direction * a;
        position = position - 0.5 * up - 0.5 * horizontal;
        position += xray + yray;
        Vec3f raydir = position - center;
        raydir.Normalize();
        Ray tmp(center, raydir);
        return tmp;
    }
    float getTMin() const {
        return 0;
    }
private:
    Vec3f center;
    Vec3f direction;
    Vec3f up;
    Vec3f horizontal;
    float angle;
};

```

因为不管屏幕大小如何变化，只要 **angle** 确定，每个生成的光线的方向是不会改变的，所以在实现时我选择了屏幕大小为边长是 1 的正方形。注意 **tmin** 要变为 0，因为是透视投影。

4、构造 Plane 类，表示 Plane 的方法是 $P \cdot n = d$ （intersect 算法见课程 PPT）：

```

class Plane :public Object3D
{
public:
    Plane() {}
    ~Plane() {}
    Plane(Vec3f &normal, float offset, Material *m) {
        n = normal;
    }

```

```

        d = offset;
        mat = m;
    }
    virtual bool intersect(const Ray &r, Hit &h, float tmin) {
        bool state = 1;
        Vec3f Ro, Rd;
        Ro = r.getOrigin();
        Rd = r.getDirection();
        if (n.Dot3(Rd) == 0) return 0;
        float t = d - n.Dot3(Ro);
        t = t / n.Dot3(Rd);
        if (t < tmin) state = 0;
        else {
            if (t < h.getT()) {
                Vec3f normal = n;
                normal.Normalize();
                h.set(t, mat, normal, r);
            }
            state = 1;
        }
        return state;
    }
private:
    Vec3f n;
    float d;
    Material *mat;
};

```

与课程 ppt 不同的一点是，课程里平面的表示方法是 $P \cdot n + d = 0$ ，所以在具体实现算法时要注意 d 的符号和 ppt 里的是反的，另外，在标准化法线向量时，不要把原来的变量改变了。

5、加入三角形类，表示方法是三个点的坐标（intersect 算法见课程 PPT）：

```

class Triangle : public Object3D
{
public:
    Triangle() {}
    ~Triangle() {}
    Triangle(Vec3f &a, Vec3f &b, Vec3f &c, Material *m) {
        A = a;
        B = b;
        C = c;
        Vec3f AB = B - A;
        Vec3f BC = C - B;
        A.Cross3(normal, AB, BC);
        normal.Normalize();
        mat = m;
    }
    virtual bool intersect(const Ray &r, Hit &h, float tmin) {
        bool state = 1;
        Vec3f Ro, Rd;
        Ro = r.getOrigin();
        Rd = r.getDirection();
        float d0 = det3x3(A.x() - B.x(), A.x() - C.x(), Rd.x(),
                        A.y() - B.y(), A.y() - C.y(), Rd.y(),
                        A.z() - B.z(), A.z() - C.z(), Rd.z());
    }
};

```

```

        float d1 = det3x3(A.x() - Ro.x(), A.x() - C.x(), Rd.x(),
                          A.y() - Ro.y(), A.y() - C.y(), Rd.y(),
                          A.z() - Ro.z(), A.z() - C.z(), Rd.z());
        float d2 = det3x3(A.x() - B.x(), A.x() - Ro.x(), Rd.x(),
                          A.y() - B.y(), A.y() - Ro.y(), Rd.y(),
                          A.z() - B.z(), A.z() - Ro.z(), Rd.z());
        float d3 = det3x3(A.x() - B.x(), A.x() - C.x(), A.x() - Ro.x(),
                          A.y() - B.y(), A.y() - C.y(), A.y() - Ro.y(),
                          A.z() - B.z(), A.z() - C.z(), A.z() - Ro.z());

        float p1, p2, p3, t;
        p2 = d1 / d0;
        p3 = d2 / d0;
        t = d3 / d0;
        p1 = 1 - p2 - p3;
        if (p2 + p3 < 1 && p2 > 0 && p3 > 0 && t >= tmin){
            if (t < h.getT()) {
                h.set(t, mat, normal, r);
            }
            state = 1;
        }
        else state = 0;
        return state;
    }
private:
    Vec3f A;
    Vec3f B;
    Vec3f C;
    Vec3f normal;
    Material *mat;
};

```

三角形的法线方向由右手定则决定（顺序是 **ABC**），做一个叉乘即可。在取交点时，使用了 **Cramer** 法则求线性方程的解，行列式的计算函数在 **matrix.C** 中。

6、加入变换的类（是 **object3d** 的子类）：

```

class Transform :public Object3D
{
public:
    Transform() {}
    ~Transform() {}
    Transform(Matrix &m, Object3D *o) {
        ma = m;
        ob = o;
    }
    virtual bool intersect(const Ray &r, Hit &h, float tmin) {
        Matrix ma_d = ma;
        ma_d.Inverse();
        Vec3f Ro = r.getOrigin();
        Vec3f Rd = r.getDirection();
        ma_d.Transform(Ro);
        ma_d.TransformDirection(Rd);

        float Rdlen = Rd.Length();
        Rd.Normalize();
        Ray rd(Ro, Rd);

        h.set(h.getT()*Rdlen, h.getMaterial(), h.getNormal(), rd);
    }
};

```

```

bool state = ob->intersect(rd, h, tmin*Rdlen);

if (state) {
    float t_w = 1.0 * h.getT() / Rdlen;
    ma_d.Transpose();
    Vec3f n_w = h.getNormal();
    ma_d.Transform(n_w);
    n_w.Normalize();
    h.set(t_w, h.getMaterial(), n_w, r);
}
else {
    h.set(h.getT()/Rdlen, h.getMaterial(), h.getNormal(), r);
}
return state;
}
private:
    Object3D *ob;
    Matrix ma;
};

```

PWS = M*POS, 所以将 direction 改变为物体世界的情况时, 要乘上当前矩阵的逆 (注意不要改变原来的矩阵), 同时 origin 也要作修改。另外 direction 还要标准化, 不然还要修改 intersect 函数的代码, 不过标准化后要注意修改 tmin 和 normal。

v is perpendicular to normal n :

Dot product $n_{OS}^T v_{OS} = 0$



$$n_{OS}^T (M^{-1} M) v_{OS} = 0$$

$$(n_{OS}^T M^{-1}) (M v_{OS}) = 0$$

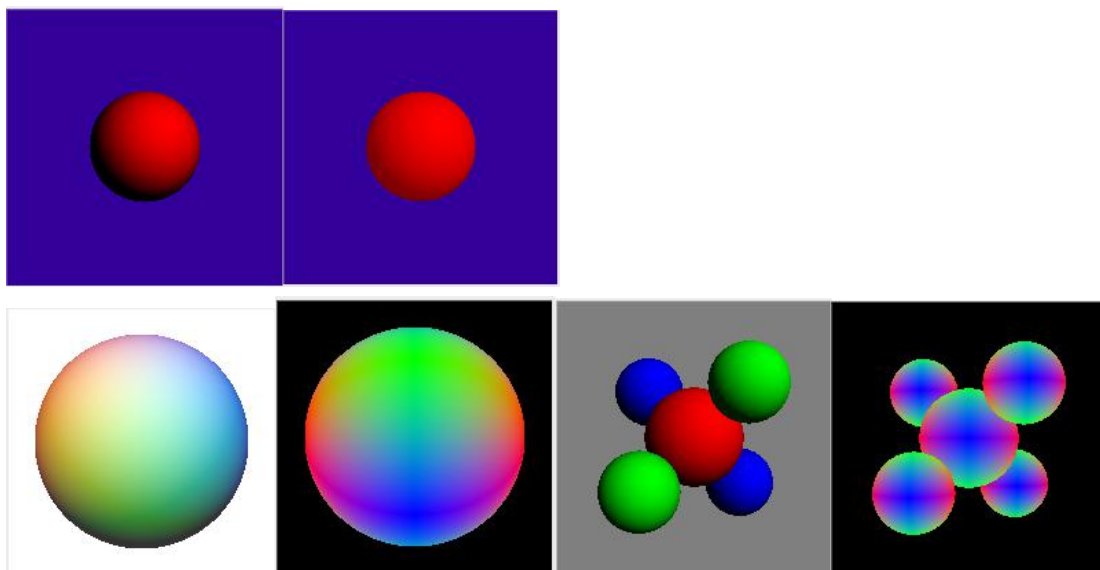
$$(n_{OS}^T M^{-1}) v_{WS} = 0$$

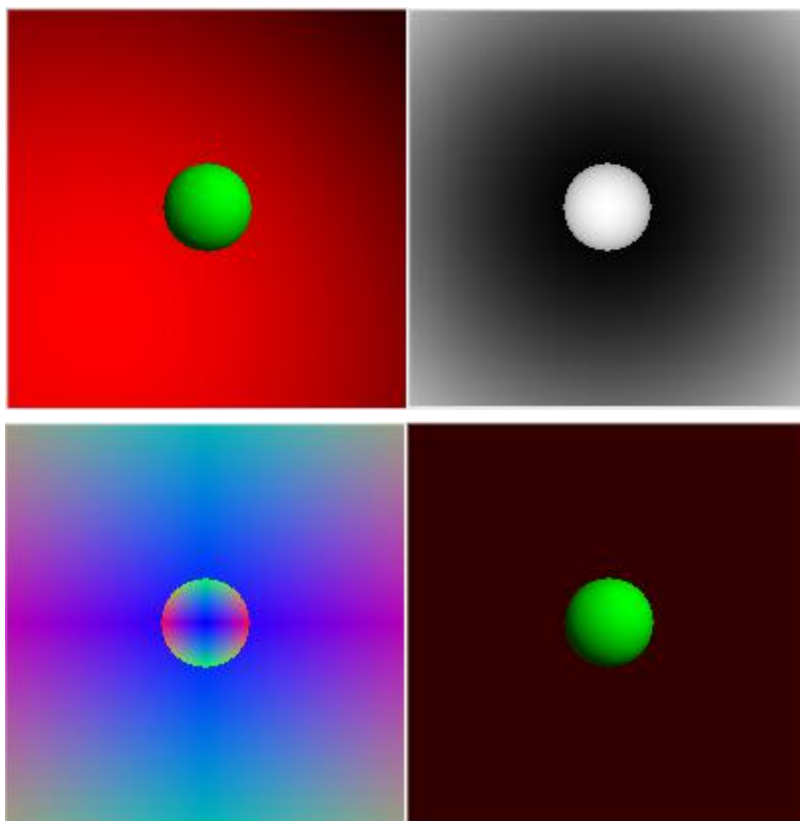
v_{WS} is perpendicular to normal n_{WS} :

$$n_{WS}^T = n_{OS}^T (M^{-1})$$

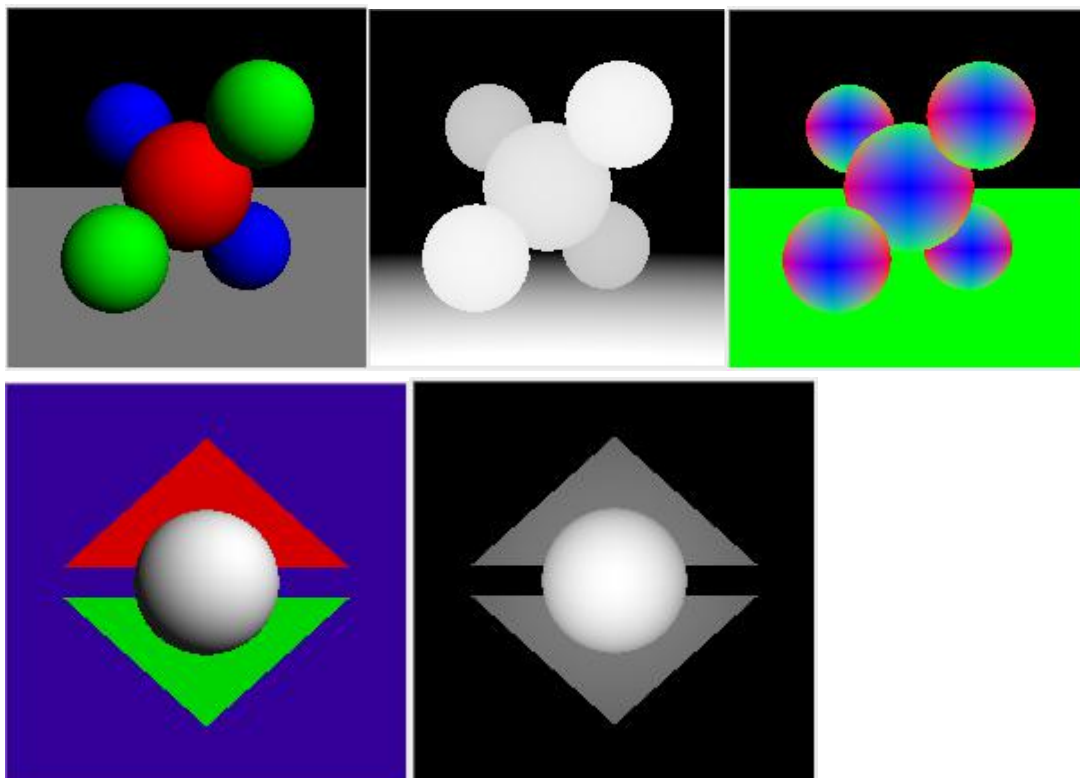
$$n_{WS} = (M^{-1})^T n_{OS}$$



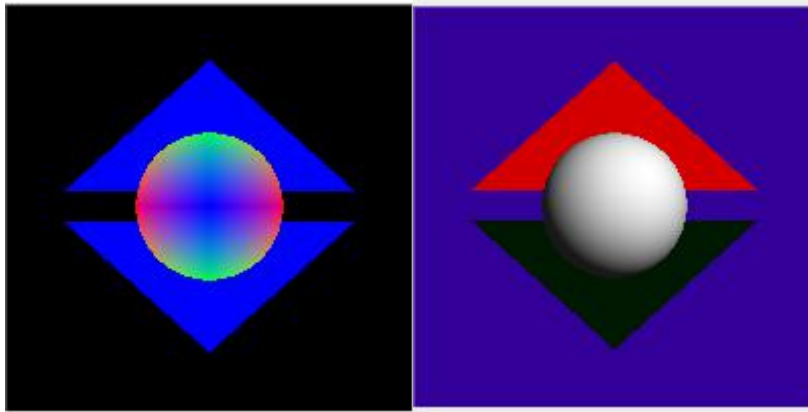
二、实验结果



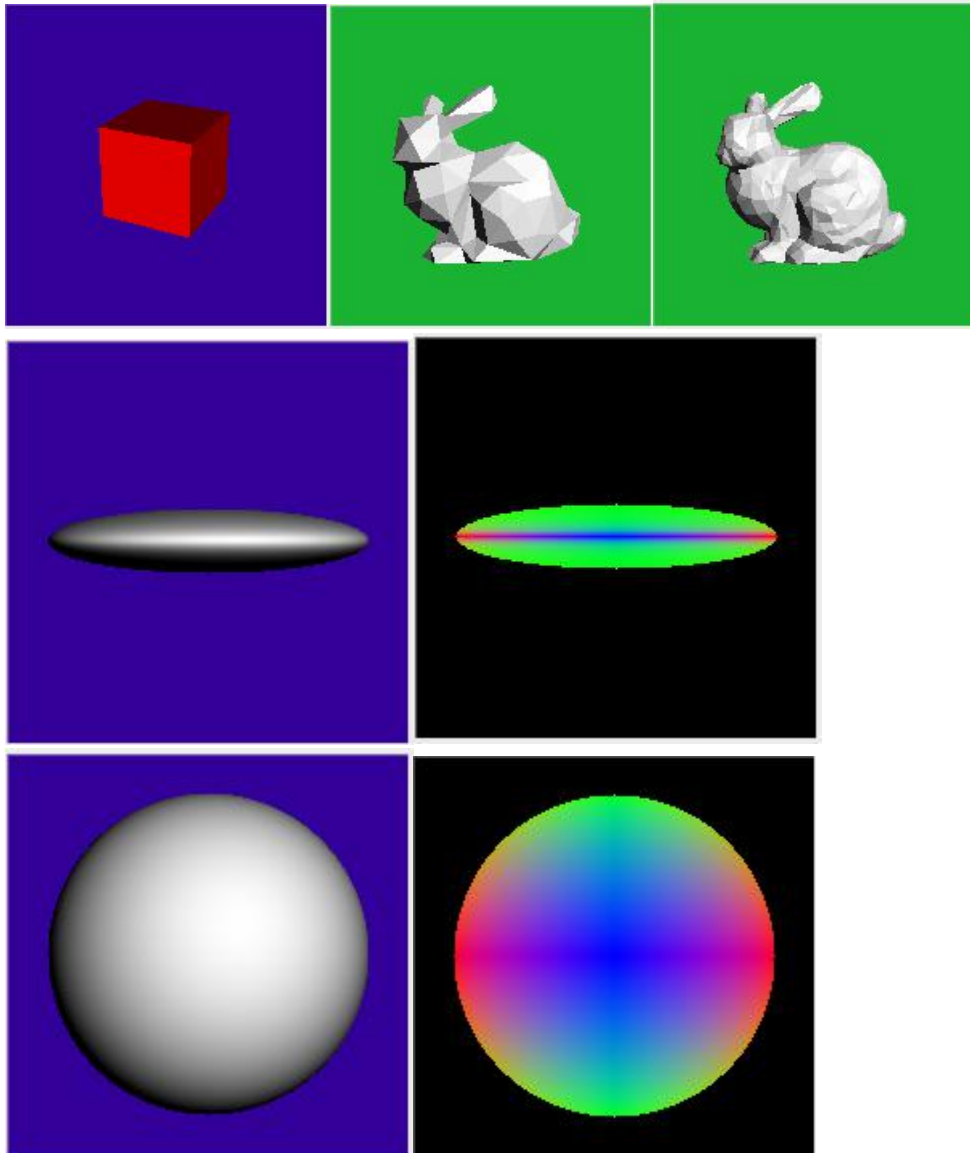


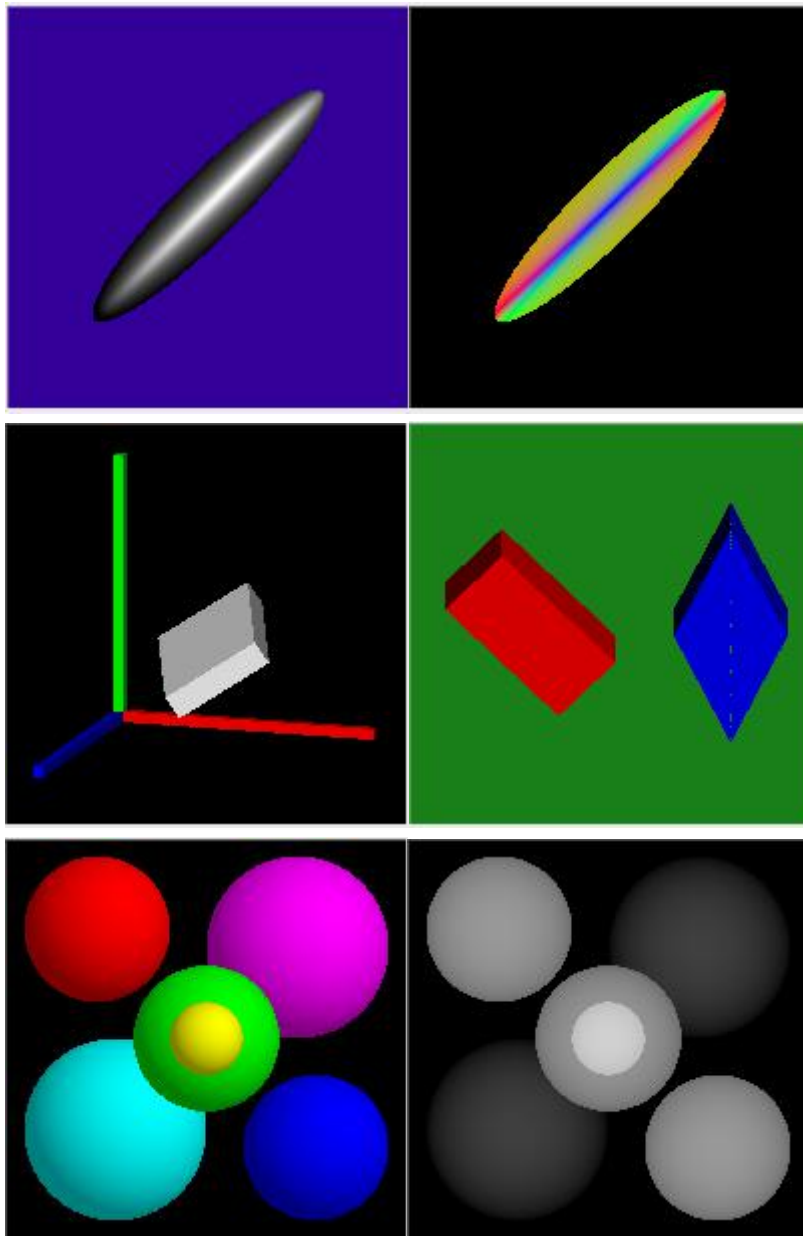
注意有无 `shade_back` 的区别，no back 图里的深红色背景是因为使用了 `ambient light`。





三角形也会受到 `shade_back` 影响。





三、实验心得

本次实验的实验内容与第一次相比明显增多，而且涉及的方向也较多（新物体类型、相机类型、光照的初步添加等等）。不过大部分内容是以课程 ppt 为基础进行的算法实现工作，具体执行起来难点在于细节的斟酌和把控。

一个较好的技巧是，因为实验的内容是分步进行的，所以在结果测试时，前面的输出不会被后面新增的代码影响。因此可以先做一部分内容，然后用示例结果进行验证，这样可以保证每一个小步的正确性。