一、实验步骤：

1、在主函数里加入 opengl 初始化代码：

主要代码：

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

2、修改 camera.h 的代码，利用 opengl 实现镜头的旋转等功能：

主要代码：

```
void OrthographicCamera::dollyCamera(float dist)
{
    center += direction * dist;
}
void OrthographicCamera::truckCamera(float dx, float dy)
{
    Vec3f horizontal;
    Vec3f::Cross3(horizontal, direction, up);
    horizontal.Normalize();

    Vec3f screenUp;
    Vec3f::Cross3(screenUp, horizontal, direction);

    center += horizontal * dx + screenUp * dy;
    screenup = screenUp;
}
void OrthographicCamera::rotateCamera(float rx, float ry)
{
    Vec3f horizontal;
    Vec3f::Cross3(horizontal, direction, up);
    horizontal.Normalize();

    // Don't let the model flip upside-down (There is a singularity
    // at the poles when 'up' and 'direction' are aligned)
    float tiltAngle = acos(up.Dot3(direction));
    if (tiltAngle - ry > 3.13)
        ry = tiltAngle - 3.13;
    else if (tiltAngle - ry < 0.01)
        ry = tiltAngle - 0.01;

    Matrix rotMat = Matrix::MakeAxisRotation(up, rx);
    rotMat *= Matrix::MakeAxisRotation(horizontal, ry);

    rotMat.Transform(center);
    rotMat.TransformDirection(direction);
    direction.Normalize();
}
```

透视相机的代码是类似的

在使用提供的代码时要保证调用的变量名与原来类里声明的变量名一致。

3、构造 material 的子类 phongmaterial 并加入 opengl 操作（算法使用了课程内的 Blinn-Torrance 的版本）：

```
class PhongMaterial :public Material
{
public:
    PhongMaterial(){}
    PhongMaterial(const Vec3f &dColor, const Vec3f &sColor, float e) {
        diffuseColor = dColor;
        specularColor = sColor;
        exponent = e;
    }
    ~PhongMaterial(){}
    Vec3f getDiffuseColor() const { return diffuseColor; }
    Vec3f Shade(const Ray &ray, const Hit &hit, const Vec3f &dirToLight,
        const Vec3f &lightColor) const {
        Vec3f v = (-1) * ray.getDirection();
        Vec3f n = hit.getNormal();
        Vec3f h = v + dirToLight;
        h.Normalize();
        float an = n.Dot3(h);
        if (dirToLight.Dot3(n) < 0) an = 0;
        an = pow(an, exponent);
        float ad = dirToLight.Dot3(n);
        if (ad < 0) ad = 0;
        return diffuseColor * lightColor * ad + specularColor * lightColor * an;
    }
    void glSetMaterial(void) const;
    Vec3f getSpecularColor()const { return specularColor; }
protected:
    Vec3f diffuseColor;
    Vec3f specularColor;
    float exponent;
};
```

注意视线的方向应该取反，也要考虑 n 点乘 h 小于 0 的情况。在加入提供的代码后要调整一些变量名。

4、对所有物体类加入 paint 方法：

对于 Group：

```
    void paint(void) {
        int i;
        for (i = 0; i < number; i++) {
            if (p[i] == NULL) continue;
            else {
                p[i]->paint();
            }
        }
        return ;
    }
```

对于 Triangle：

```
    void paint(void) {
        mat->glSetMaterial();
        glBegin(GL_TRIANGLES);
        glNormal3f(normal.x(), normal.y(), normal.z());
        glVertex3f(A.x(), A.y(), A.z());
        glVertex3f(B.x(), B.y(), B.z());
```

```
        glVertex3f(C.x(), C.y(), C.z());
        glEnd();
    }
```

对于 Plane（画一个大的正方形，在取中心点时只要该点在平面上即可）：

```
    void paint(void) {
        float proj = d / (n.Length()*n.Length());
        Vec3f ow = proj * n;
        Vec3f v(1,0,0);
        if (n == v) v.Set(0, 1, 0);
        Vec3f b1, b2;
        v.Cross3(b1, v, n);
        v.Cross3(b2, n, b1);
        b1.Normalize();
        b2.Normalize();
        Vec3f p0 = ow + BIG * b1 + BIG * b2;
        Vec3f p1 = ow - BIG * b1 + BIG * b2;
        Vec3f p2 = ow - BIG * b1 - BIG * b2;
        Vec3f p3 = ow + BIG * b1 - BIG * b2;//注意顺序
        mat->glSetMaterial();
        glBegin(GL_QUADS);
        glNormal3f(n.x(), n.y(), n.z());
        glVertex3f(p0.x(), p0.y(), p0.z());
        glVertex3f(p1.x(), p1.y(), p1.z());
        glVertex3f(p2.x(), p2.y(), p2.z());
        glVertex3f(p3.x(), p3.y(), p3.z());
        glEnd();
    }
```

对于 Sphere（由很多四边形拼接而成 tessellation，在法线设置时，可以每个点设置一个法线，也可以设置为平面的法线）：

```
    void paint(void) {
        float step1 = 180 / tesh;
        float step2 = 360 / tesw;
        Vec3f v(0, -1, 0);
        Matrix transy, transx;
        transy = transy.MakeYRotation(DegreesToRadians(step2));
        transx = transx.MakeXRotation(DegreesToRadians((-step1)));
        Matrix transys, transxs;
        transys = transys.MakeYRotation(DegreesToRadians(step2 / 2));
        transxs = transxs.MakeXRotation(DegreesToRadians((-step1 / 2)));
        int iPhi, iTheta;

        mat->glSetMaterial();
        glBegin(GL_QUADS);
        for (iPhi = -90; iPhi <= 90; iPhi += step1){
            for (iTheta = 0; iTheta < 360; iTheta += step2) {
                Vec3f n = v;
                transys.TransformDirection(n);
                transxs.TransformDirection(n);
                n.Normalize();

                // compute appropriate coordinates & normals
                float y = radius * sinf(DegreesToRadians(iPhi));
                float r = radius * cosf(DegreesToRadians(iPhi));
                float z = r * cosf(DegreesToRadians(iTheta));
                float x = r * sinf(DegreesToRadians(iTheta));
                Vec3f p0(x, y, z);
                y = radius * sinf(DegreesToRadians((iPhi + step1)));
                r = radius * cosf(DegreesToRadians((iPhi + step1)));
```

```
                z = r * cosf(DegreesToRadians(iTheta));
                x = r * sinf(DegreesToRadians(iTheta));
                Vec3f p1(x, y, z);
                y = radius * sinf(DegreesToRadians((iPhi + step1)));
                r = radius * cosf(DegreesToRadians((iPhi + step1)));
                z = r * cosf(DegreesToRadians((iTheta + step2)));
                x = r * sinf(DegreesToRadians((iTheta + step2)));
                Vec3f p2(x, y, z);
                y = radius * sinf(DegreesToRadians(iPhi));
                r = radius * cosf(DegreesToRadians(iPhi));
                z = r * cosf(DegreesToRadians((iTheta + step2)));
                x = r * sinf(DegreesToRadians((iTheta + step2)));
                Vec3f p3(x, y, z);

                p0 = p0 + center;
                p1 = p1 + center;
                p2 = p2 + center;
                p3 = p3 + center;
                // send gl vertex commands
                if (isgouraud) {
                    n = p0 - center;
                    n.Normalize();
                    glNormal3f(n.x(), n.y(), n.z());
                    glVertex3f(p0.x(), p0.y(), p0.z());
                    n = p1 - center;
                    n.Normalize();
                    glNormal3f(n.x(), n.y(), n.z());
                    glVertex3f(p1.x(), p1.y(), p1.z());
                    n = p2 - center;
                    n.Normalize();
                    glNormal3f(n.x(), n.y(), n.z());
                    glVertex3f(p2.x(), p2.y(), p2.z());
                    n = p3 - center;
                    n.Normalize();
                    glNormal3f(n.x(), n.y(), n.z());
                    glVertex3f(p3.x(), p3.y(), p3.z());
                }
                else {
                    glNormal3f(n.x(), n.y(), n.z());
                    glVertex3f(p0.x(), p0.y(), p0.z());
                    glVertex3f(p1.x(), p1.y(), p1.z());
                    glVertex3f(p2.x(), p2.y(), p2.z());
                    glVertex3f(p3.x(), p3.y(), p3.z());
                }

                transy.TransformDirection(v);
            }
            transx.TransformDirection(v);
        }
        glEnd();
    }
```

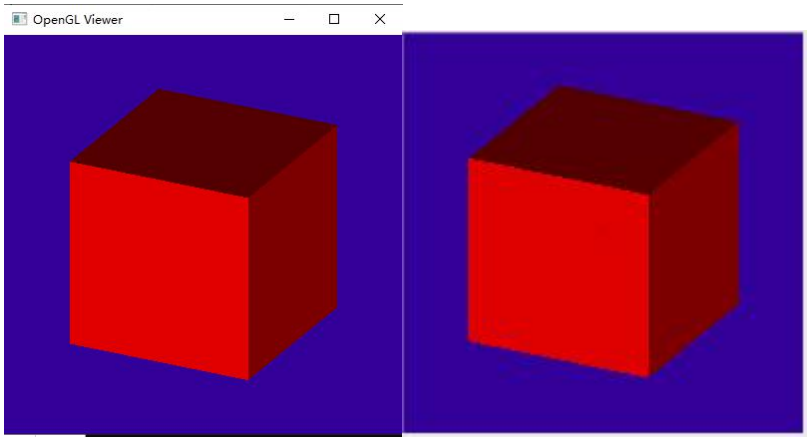
对于 Transform：

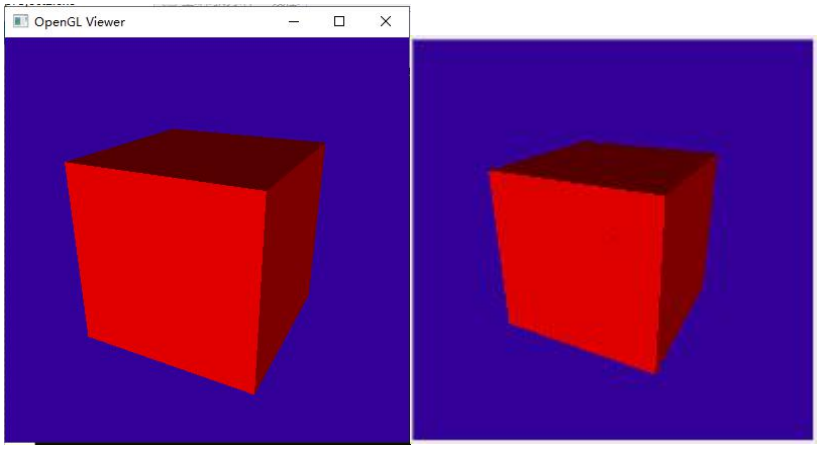
```
    void paint(void) {
        glPushMatrix();
        GLfloat *glMatrix = ma.glGet();
        glMultMatrixf(glMatrix);
        delete[] glMatrix;
        ob->paint();
```

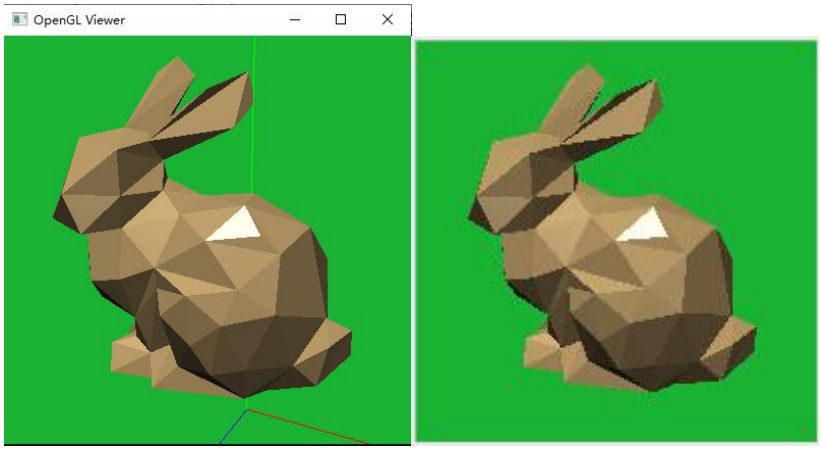
```
        glPopMatrix();
    }
```

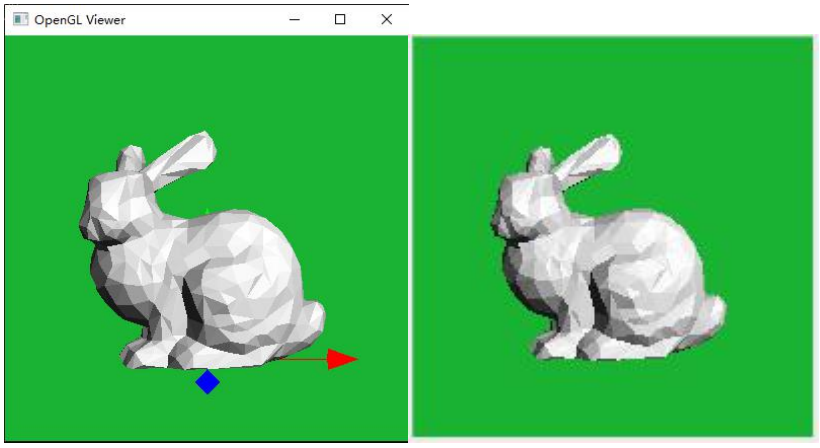
这里面最难的地方是 sphere 的 paint，需要考虑清楚参数的正负还有法线是否符合右手定则，可以在初步写完后用例子进行验证并与参考进行比对，如果出现问题，则需要分析是哪个变量的符合取反了，进行一定程度的微调后就可以了。

二、实验结果

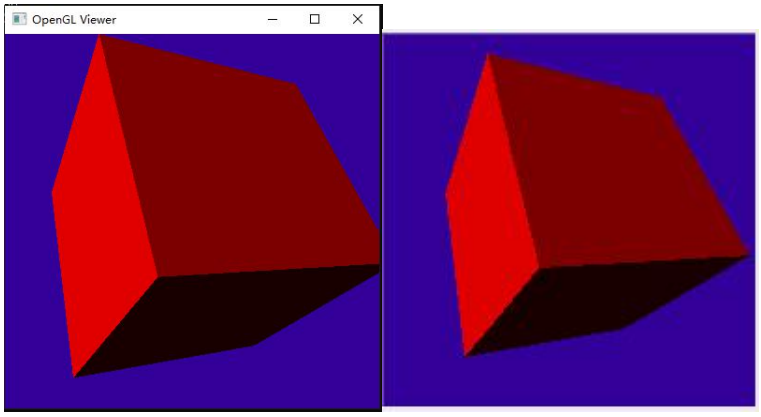


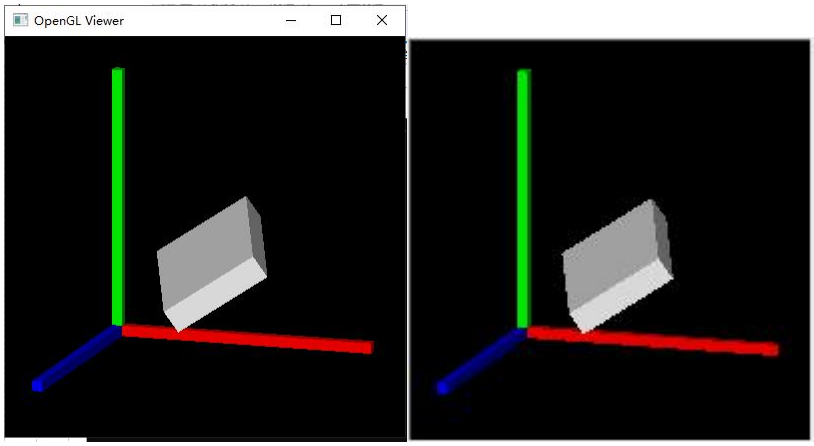


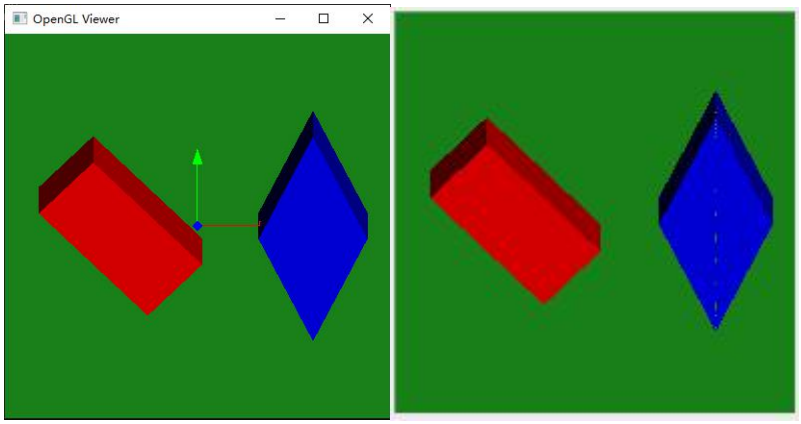


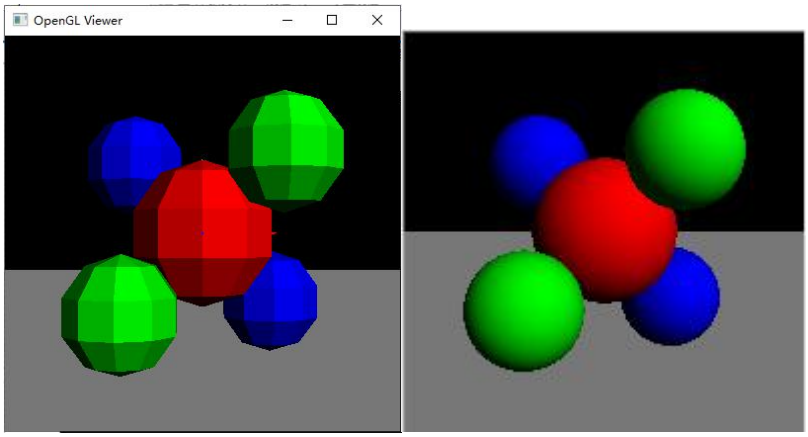


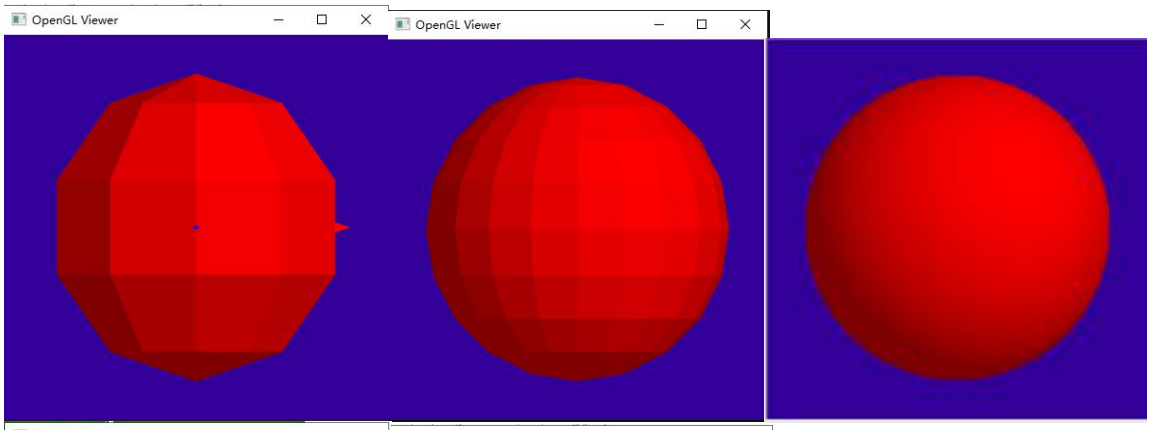
经测试鼠标左键改变照相机视角正常。右键改变观察远近：在正投影下没有变化，在透视投影下会变化。

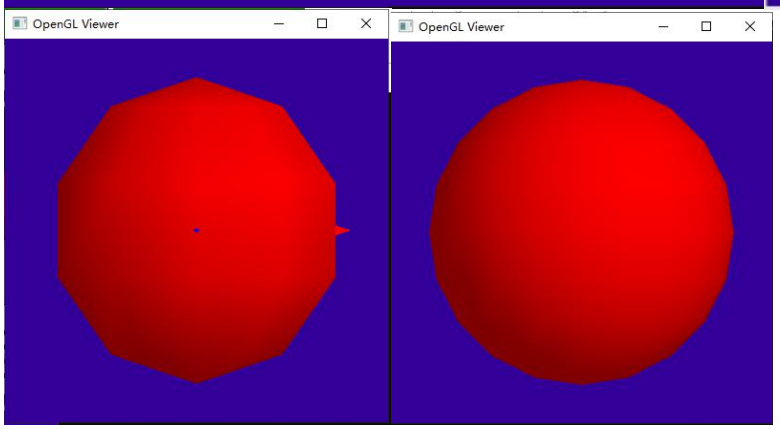


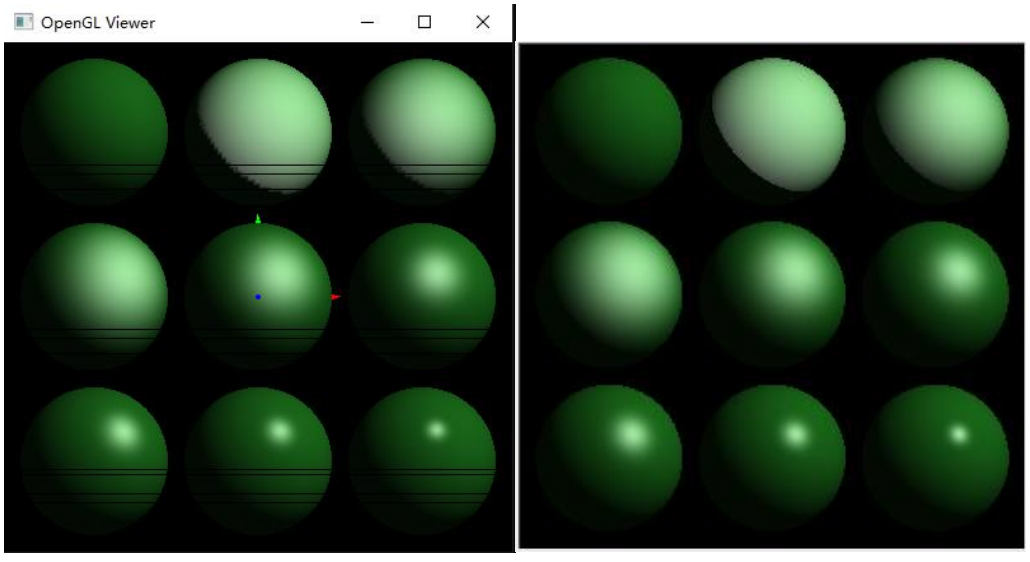


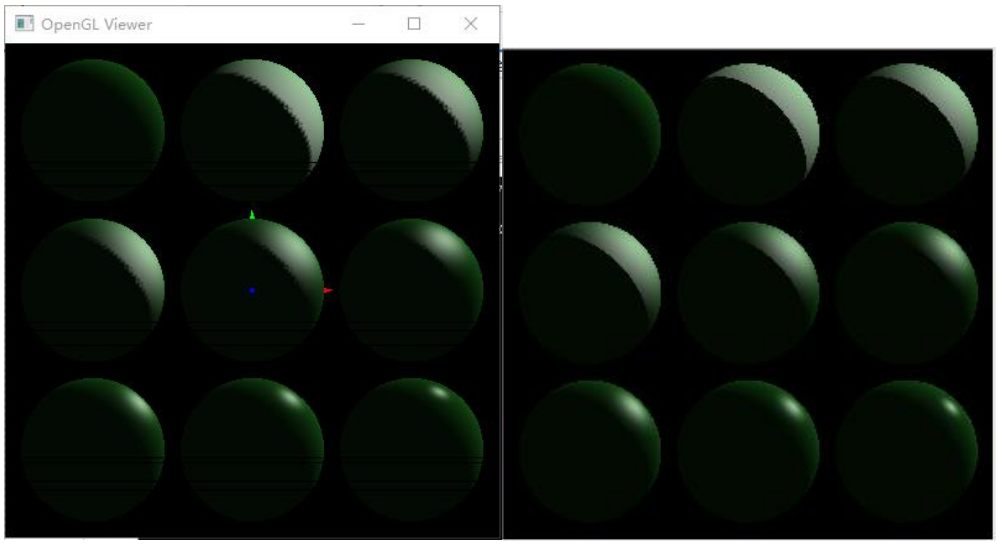


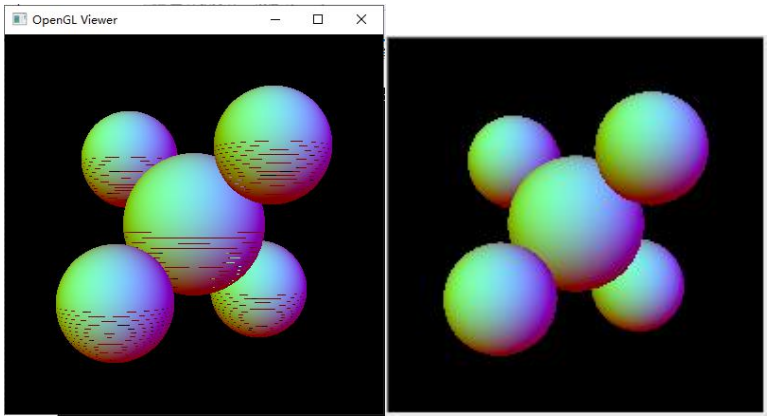

OpenGL Viewer

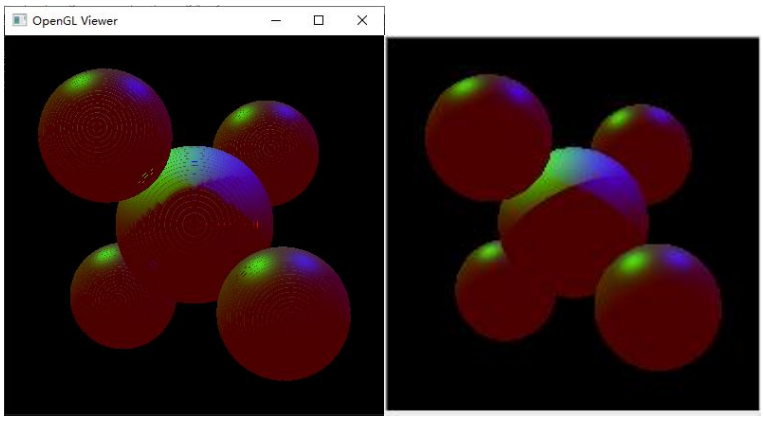

OpenGL Viewer
OpenGL Viewer

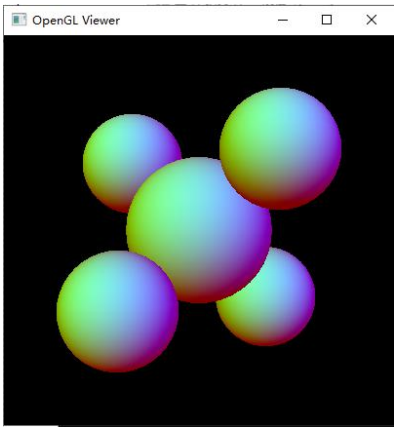

OpenGL Viewer
OpenGL Viewer


OpenGL Viewer









在上面几个 opengl 视角下出现了线状残缺，应该是分割过小导致计算精度不达标导致的结果，经过尝试，发现 tessellation 设置为 40 20 可以呈现比较好的效果（见右上图。）

三、实验心得

实验里较困难的是第一次接触 opengl 以及球体的呈现，在开始时我误认为 opengl 是 z 轴向上的，导致呈现效果很不理想，后来了解是 y 轴向上后，进行代码的编写就顺畅了许多。