中山大學
SUN YAT-SEN UNIVERSITY

《计算机图形学》

实验报告

（作业三）

学 院 名 称	：	数据科学与计算机学院
专业（班级）	：	17 计科 7 班
学 生 姓 名	：	张富瑞
学 号	：	17341197
时 间	：	2019 年 1 月 9 日

一.实验要求

参考作业文档，编写GLSL着色器程序，绘制沿固定轨道运动的小球。

3.1 使用GL_POINTS绘制沿固定轨道运动的小球

每个glVertex调用指明一个小球的球心位置。

小球大小根据离观察点距离变化（近大远小）。

使用Phong Shading。

参考Assignment3.pdf第9页的Vertex Shader和Fragment Shader。

3.2 （选做）使用VBO进行绘制

参考Assignment3.pdf第10页 的绘制函数。

二.实验过程

1.顶点着色器

```
const char *vsChar = R"(
    #version 120

    varying vec3 vertex_to_light_vector;
    varying vec4 vertex_in_modelview_space;
    void main()
    {
            //顶点变换到裁剪空间位置，作为输出
            gl_Position = gl_ModelViewProjectionMatrix * gl_Ve
rtex;

            //进行近大远小
            gl_PointSize = 100*(0.6+gl_Position.z);
            gl_FrontColor = gl_Color;
            vertex_in_modelview_space = (gl_ModelViewMatrix *
gl_Vertex);
            vertex_to_light_vector = vec3(gl_LightSource[0].po
sition - vertex_in_modelview_space);

    })";
```

在一开始没有怎么接触glsl语言的时候，我对于要怎样链接这样一个编写好的顶点着色器的文件都很懵逼。经过向同学请教，我这才知道可以直接将整个着色器文件编写成字符串放入主文件中，然后通过的字符串的方式直接编译链接。

由于作业中给出的要求是要使用gl_points来进行一个球的绘制，经过查询资料，我了解到opengl中有“点精灵”这样的概念。通过更改顶点着色器来将一个点的大小进行修改，这样我们能够直接通过画点就能够画出一个球的形状。由于作业中同时要求我们实现近大远小的视觉效果，所以我们根据画点的位置来进行点的大小的修改。这里我是根据原始的右手坐标系进行修改大小。在z轴为正的情况下相对的大，为负的时候相对的小。

而vertex_in_modelview_space，以及vertex_to_light_vector都是我们之后在片段着色器中所需要的顶点坐标以及顶点到光源之间的向量。这里与老师pdf中给出的形式相

同。

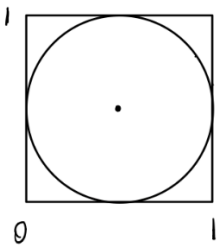
2.片段着色器

```
const char *frChar = R"(
    #version 120
    varying vec3 vertex_to_light_vector;
    varying vec4 vertex_in_modelview_space;
    void main()
    {
            vec2 n_xy = gl_PointCoord - vec2(0.5, 0.5);

            float dist = sqrt(dot(n_xy,n_xy));//距离
            if (dist >= 0.5)discard;

            vec3 n = normalize(vec3(n_xy, sqrt(pow(0.5,2)-
pow(dist,2))));   // 法向量
            vec3 l = normalize(vertex_to_light_vector);   // 入
射方向
            vec3 r = normalize(reflect(-l, n));  // 反射方向
            vec3 v = normalize(vec3(0, 0, 1) - vertex_in_model
view_space.xyz);     // 观察方向
            vec3 h = normalize(v + l);

            vec4 ambient = gl_LightSource[0].ambient;
            vec4 diffuse = gl_LightSource[0].diffuse;
            vec4 specular = gl_LightSource[0].specular;

            //漫反射
            float diffuse_term = clamp(max(dot(n, l),0.0), 0.0
, 1.0);
            //镜面反射
            int a = 5;
            float specular_term = max(pow(dot(r, v), a), 0.0);

            //Blinn-phong
            float specular_term1 = max(pow(dot(n, h), a), 0.0)
;


            if (dot(r,v) < 0)
                gl_FragColor = (ambient + diffuse*diffuse_term
+0.0) * gl_Color;
            else
```

```
                gl_FragColor = (ambient + diffuse*diffuse_term
 + 0.5*specular*specular_term) * gl_Color;
    }
)";
```

gl_PointCoord是片元着色器的内建只读变量，它的值是当前片元所在点图元的二维坐标。点的范围是0.0到1.0。

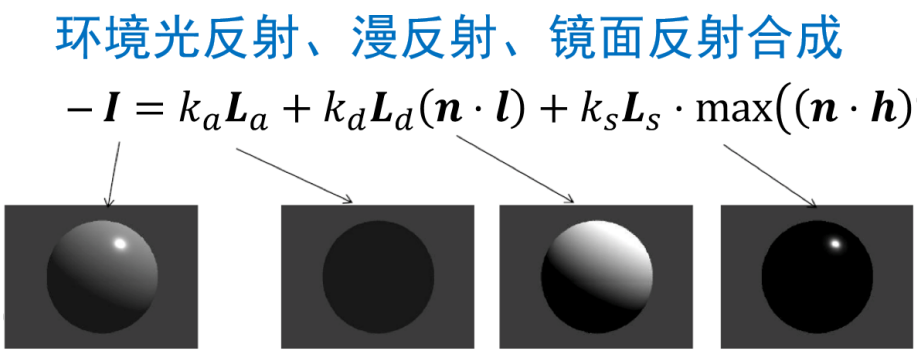
由于要实现球体，我们需要将投影的矩形变成一个球体。



如上图所示，我们只用将xy平面的圆之外的部分舍弃，不着色就可以实现球体。

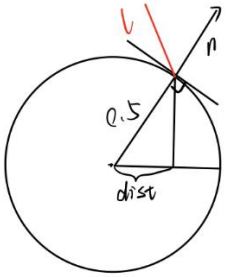
实现了球体之后，我们就需要实现phong shading模型。

Phong shading模型需要环境光，漫反射光，以及镜面反射光这三种光照的着色。



环境光不用进行修改，直接使用环境光*着色的颜色就可以得到。

漫反射光需要我们计算入射方向的向量以及法向量。入射方向在顶点着色器中已经得到，就是利用varying声明的vertex_to_light_vector。



我们通过计算表面片元的在z轴上的高度就可以和投影在xoy平面的坐标组合在一起来

得到法向量。

镜面反射光需要反射方向和观察方向的向量。观察方向的向量可以直接通过顶点着色器中声明的vertex_in_modelview_space和观察点的坐标做差得到。反射向量可以通过法向量和入射向量和可以直接使用的reflect函数直接得到。

老师的上课的pdf中还给出了一种Blinn-phong模型，这里我也进行了一下尝试，但是在实验结果上没有发现太大的区别。

- $\boldsymbol{h} = \frac{\boldsymbol{l}+\boldsymbol{v}}{|\boldsymbol{l}+\boldsymbol{v}|}$
- $\boldsymbol{I}_s = k_s \boldsymbol{L}_s \cdot \max((\boldsymbol{n} \cdot \boldsymbol{h})^{\alpha}, 0)$

得到各种需要向量之后，就可以通过计算得到想要的矩阵，从gl_LightSource[0]中读出光照的参数，然后分别将三种光照加起来可以得到我们的光照。这里需要注意r和v相乘为负的情况，这种情况下镜面反射不起作用。

3.编译链接两种着色器

```
void MyGLWidget::complie_shader()
{
    //创建一个着色器对象
    GLuint vertexShader = glCreateShader(GL_VERTEX_SHADER);
    GLuint fragmentShader = glCreateShader(GL_FRAGMENT_SHADER)
;

    //替换着色器对象中的源代码
    glShaderSource(vertexShader, 1, &vsChar, 0);
    glShaderSource(fragmentShader, 1, &frChar, 0);

    //编译一个着色器对象
    glCompileShader(vertexShader);
    glCompileShader(fragmentShader);

    //创建一个program
    GLuint program = glCreateProgram();

    //将着色器对象附加到program 对象
    glAttachShader(program, vertexShader);
    glAttachShader(program, fragmentShader);

    //连接一个program 对象
    glLinkProgram(program);
```

```
    //指明在接下来的绘制中使用program所代表 的着色器程序
    glUseProgram(program);
}
```

这里完全参照老师pdf中给出的参考代码。

4.初始化gl

```
void MyGLWidget::initializeGL()
{
    initializeOpenGLFunctions();

    // vertexShader = glCreateShader(GL_VERTEX_SHADER);
    // Your Implementation

    glViewport(0, 0, width(), height());
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

    //材料特性决定对光的反射 设定材料的反射系数
    GLfloat mat_ambient[] = {0.8f, 0.8f, 0.8f, 1.0f};
    GLfloat mat_diffuse[] = {0.8f, 0.0f, 0.8f, 1.0f};
    GLfloat mat_specular[] = {1.0f, 0.0f, 1.0f, 1.0f};
    GLfloat mat_shininess[] = {50.0f};
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

    //设置光源的位置
    GLfloat light_position[] = {0.0f, 1.0f, 0.0f, 0.0f};
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);

    //分别设置环境光，漫反射光，镜面光
    GLfloat ambient[] = {0.2f, 0.2f, 0.2f, 1.0f};
    GLfloat diffuse[] = {1.0f, 1.0f, 1.0f, 1.0f};
    GLfloat specular[] = {1.0f, 1.0f, 1.0f, 1.0f};
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, specular);

    //打开光照
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
```

```
    glEnable(GL_VERTEX_PROGRAM_POINT_SIZE);//更改点的大小
    glEnable(GL_POINT_SPRITE);//启用点精灵
    glEnable(GL_DEPTH_TEST);
    complie_shader();
}
```

在这里初始化了单光源的光源的环境光，漫反射光，镜面反射的参数，以及使用了glMaterialfv()来设置材料特性。这里参照老师上课时讲的东西，参考老师给出的pdf。

5.绘制场景

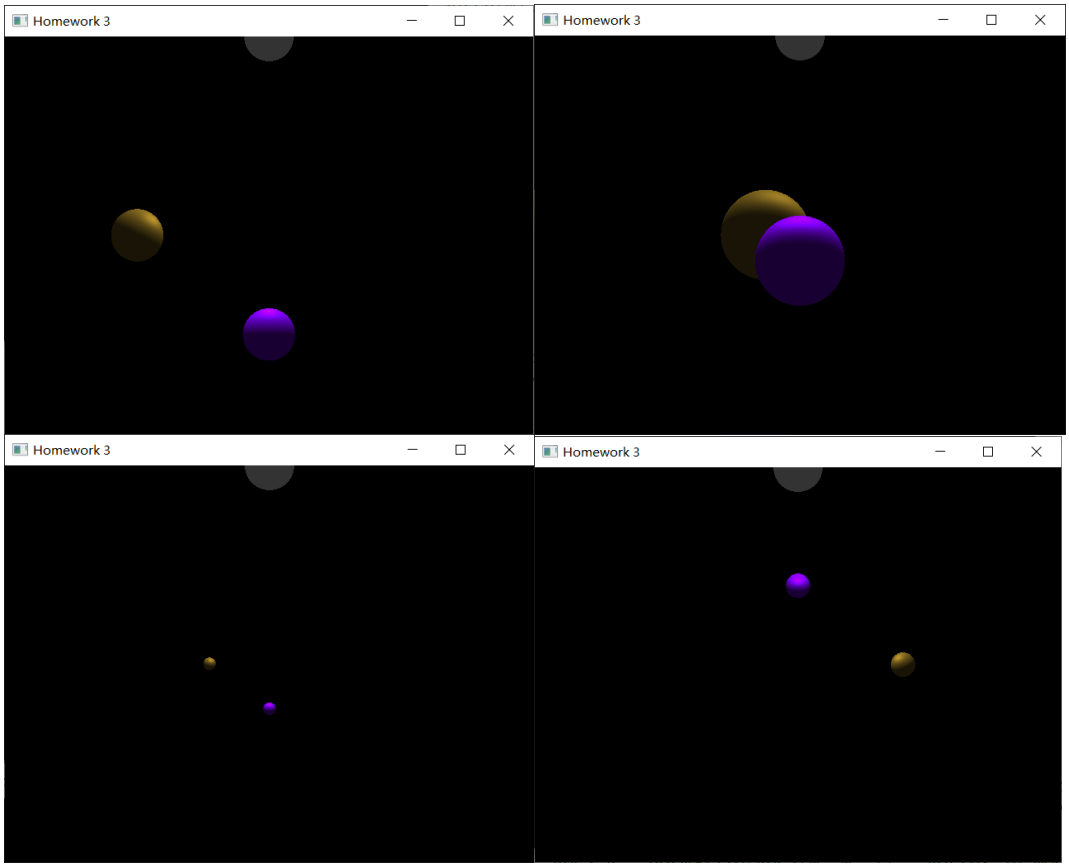
```
void MyGLWidget::paintGL()
{
    // Your Implementation
    glClear(GL_COLOR_BUFFER_BIT);

    theta %= 360;//theta 绘制圆型轨迹的极坐标角度
    theta += 1;

    //绘制光源的位置，但是没有加上贴纸
    glPushMatrix();
    glBegin(GL_POINTS);
    glColor3d(1, 1, 1);
    glVertex3f(0, 1, 0);
    glEnd();
    glPopMatrix();


    //绘制竖向的转动的球
    glPushMatrix();
    glBegin(GL_POINTS);
    glColor3d(0.5, 0, 1);
    glVertex3f(0, R * cos(theta * PI / 180), R * sin(theta * P
I / 180));
    glEnd();
    glPopMatrix();

    //绘制横向的转动的球
    glPushMatrix();
    glBegin(GL_POINTS);
    glColor3d(0.489, 0.3903, 0.12);
    glVertex3f(R * cos(theta * PI / 180), 0, R * sin(theta * P
I / 180));
    glEnd();
```

```
    glPopMatrix();
}
```

为了标记一下光源的位置，我在光源的位置也使用了gl_points画了一个球，但是这个球并没有得到光照的着色，原因应该是它直接放在光源上了。同时我又画了两个球，其中一个球绕着y轴转，另外一个绕着x轴转。绘制过程很简单，调用gl_points绘制即可。

三.实验结果

以下为运行过程中的一些截图，可以看出光照的效果和近大远小的效果。其中褐色的球绕着y轴旋转，紫色的球绕着x轴旋转。



(上面那个灰色为没有光照着色的光源)

四.实验总结

在进行这一次作业的时候，确实遇到了许多的困难。这里十分感谢老师和助教给出ddl的延迟。邻近期末考试，既有大作业又要进行复习，确实时间不够充足。首先遇到的问题就是不熟悉glsl的语法，虽然都是c语言，但是这确实是一种不同于以往的语言，并且和我们这学期学的opengl也不大相同。只有大概了解了这种语言之后，才能够知道如何实现老师要求的gl_points画球的基本逻辑。这是入手这个作业遇到的困难。之后又遇到链接的问题，用文件链接的写法，一直报了许多错误，可能是操作不当。所幸请教大佬后，可以直接使用字符串来编译链接。

经过重重bug之后能够实现近大远小的功能之后，实现光照的部分也遇到了许多困难。虽然老师课上的pdf给出了算法，但是向量的求解也困扰了我很久，通过反复的修改，才比较正确实现了光照的模型。

总的来说，这次作业算一次困难的实践，由于对于语言和模型的不了解，花费了许多

时间去理解如何实现老师给出的要求的逻辑。