Final Project Document

118033930145 雷沁欣

1. 配置环境（在 VS2017 下配置 OpenGL）和运行环境说明

环境：Windows 10、OpenGL、C++

开发工具：Visual Studio 2017（确保已安装 C/C++开发组件、NuGet 包管理器）

glut 的库可以点击下面的链接下载：

https://www.opengl.org/resources/libraries/glut/glutdlls37beta.zip

下载解压后：

把 glut.h 放到...\VC\Tools\MSVC\14.10.25017\include\gl 下（没有 gl 文件夹就新建一个）

把 glut.lib,glut32.lib 放到...\VC\Tools\MSVC\14.10.25017\lib\x86 下

把 glut.dll,glut32.dll 放到 C:\Windows\SysWOW64 下

glm 的库可以点击下面的链接下载：

https://github.com/g-truc/glm/releases

下载后直接将 glm 文件夹放到...\VC\Tools\MSVC\14.10.25017\include 下

在 VS2017 新建项目时管理 NuGet 程序包搜索 nupengl，将搜索到的两个都安装

最后生成的可执行文件在...\CG project\Release 文件夹中，可以在 windows 环境下直接执行

2. 项目及其代码介绍

此项目使用最基本的光线追踪算法实现了必须要求的 ambient，diffuse， specular high light 和 shadows。另外，简单实现了一下选择要求的 transparency 和 refraction。详细的实现说明见如下代码说明部分：

loadTGA 函数用于加载一些特殊问题贴图或场景贴图图像文件，这个功能选择性使用。

color.h 和 color.cpp 定义了项目中使用到的颜色信息

```
#define __HEADER_COLOR__
#define __HEADER_COLOR__

class Color
{

public:
    float r, g, b;
    static const Color WHITE;
    static const Color BLACK;
    static const Color RED;
    static const Color GREEN;
    static const Color BLUE;
    static const Color GRAY;
    static const Color YELLOW;
    static const Color MAGENTA;
    static const Color PURPLE;

    Color() : r(1), g(1), b(1) {};
```

```
        Color(float rCol, float gCol, float bCol) : r(rCol), g(gCol), b(bCol) {};

        void scaleColor(float);

        void combineColor(Color);

        void combineColor(Color, float);

        Color phongLight(Color, float,    float);
};
#endif //__HEADER_COLOR__

#include "color.h"

void Color::scaleColor(float scaleFactor)
{
        r = r * scaleFactor;
        g = g * scaleFactor;
        b = b * scaleFactor;
}

void Color::combineColor(Color col)
{
        r *= col.r;
        g *= col.g;
        b *= col.b;
}

void Color::combineColor(Color col, float scaleFactor)
{
        r +=    scaleFactor * col.r;
        g +=    scaleFactor * col.g;
        b +=    scaleFactor * col.b;
}


Color Color::phongLight(Color ambientCol, float diffTerm, float specTerm)
{
        Color col;
        col.r = (ambientCol.r) * r + diffTerm * r + specTerm;
        col.g = (ambientCol.g) * g + diffTerm * g + specTerm;
        col.b = (ambientCol.b) * b + diffTerm * b + specTerm;
        return col;
```

```
}

const Color Color::WHITE = Color(1, 1, 1);
const Color Color::BLACK = Color(0, 0, 0);
const Color Color::RED = Color(1, 0, 0);
const Color Color::GREEN = Color(0, 1, 0);
const Color Color::BLUE = Color(0, 0, 1);
const Color Color::GRAY = Color(0.2f, 0.2f, 0.2f);
const Color Color::YELLOW = Color(1, 1, 1, 0);
const Color Color::MAGENTA = Color(1, 0, 1);
const Color Color::PURPLE = Color(0.5, 0.5, 0.5);
```

cone.h 和 cone.cpp ， cylinder.h 和 cylinder.cpp， plane.h 和 plane.cpp， sphere.h 和 sphere.cpp，tetrahedron.h 和 tetrahedron.cpp，pentagonal_pyramid.h 和 pentagonal_pyramid.cpp 分别定义和实现了圆锥体，圆柱体，平面图形，球体、四面体和五棱锥，另外，通过平面的组合定义实现了一个立方体。这些模型都继承了两个重要的数据定义，一个是 object.h 和 object.cpp，另一个是 vector.h 和 vector.cpp。Object 类定义实现了模型最基本的属性设置获取方法，而 Vector 则定义实现了场景中模型坐标的基本变换操作:

```
#include <math.h>
#include "vector.h"

Vector::Vector() : x(0), y(0), z(0) {}

Vector::Vector(float a_x, float a_y, float a_z)
: x(a_x), y(a_y), z(a_z) {}


const Vector& Vector::operator*=(float scale)
{
    x *= scale;
    y *= scale;
    z *= scale;
    return *this;
}

Vector Vector::operator*(float scale) const
{
    return Vector(*this) *= scale;
}

const Vector& Vector::operator/=(float scale)
{
    return operator*=(1.0f / scale);
}
```

```
Vector Vector::operator/(float scale) const
{
	return Vector(*this) *= 1.0f / scale;
}

const Vector& Vector::operator+=(const Vector rhs)
{
	x += rhs.x;
	y += rhs.y;
	z += rhs.z;
	return *this;
}

Vector Vector::operator+(const Vector rhs) const
{
	return Vector(x + rhs.x, y + rhs.y, z + rhs.z);
}

const Vector& Vector::operator-=(const Vector rhs)
{
	x -= rhs.x;
	y -= rhs.y;
	z -= rhs.z;
	return *this;
}

Vector Vector::operator-(const Vector rhs) const
{
	return Vector(x -rhs.x, y - rhs.y, z - rhs.z);
}

const bool Vector::operator<(const Vector rhs)
{
	float l1 = length();
	float l2 = rhs.length();
	return (l1 < l2);
}

void Vector::scale(float scale)
{
	operator*=(scale);
}
```

```
Vector Vector::cross(const Vector rhs) const
{
	float tx = y * rhs.z - z * rhs.y;
	float ty = z * rhs.x - x * rhs.z;
	float tz = x * rhs.y - y * rhs.x;
	return Vector(tx, ty, tz);
}
float Vector::dot(const Vector rhs) const
{
	return x*rhs.x + y*rhs.y + z*rhs.z;
}
float Vector::dist(const Vector rhs) const
{
	Vector vdif = (*this)-rhs;
	return vdif.length();
}
float Vector::length() const
{
	return sqrt(dot(*this));
}

void Vector::normalise()
{
	scale(1.0f / length());
}
```

main.cpp 是本项目的主程序，在这个文件中， 以下代码定义了两个光源和物体交互产生的阴影信息：

```
		PointBundle shadow = closestPt(q.point, l);
		PointBundle shadow2 = closestPt(q.point, l2);
		Vector reflectionVector = ((n*2)*(n.dot(v)))-v;
		if (shadow.index == -1){
			colorSum = col.phongLight(backgroundCol, lDotn, spec);
		}
		if (shadow.index != -1)
		{
			if (q.point.dist(shadow.point) < q.point.dist(light)){
				colorSum = col.phongLight(backgroundCol, lDotn/6, 0);
			}
		}
		if (shadow2.index == -1){
			colorSum = col.phongLight(backgroundCol, l2Dotn, spec);
		}
		if (shadow2.index != -1)
```

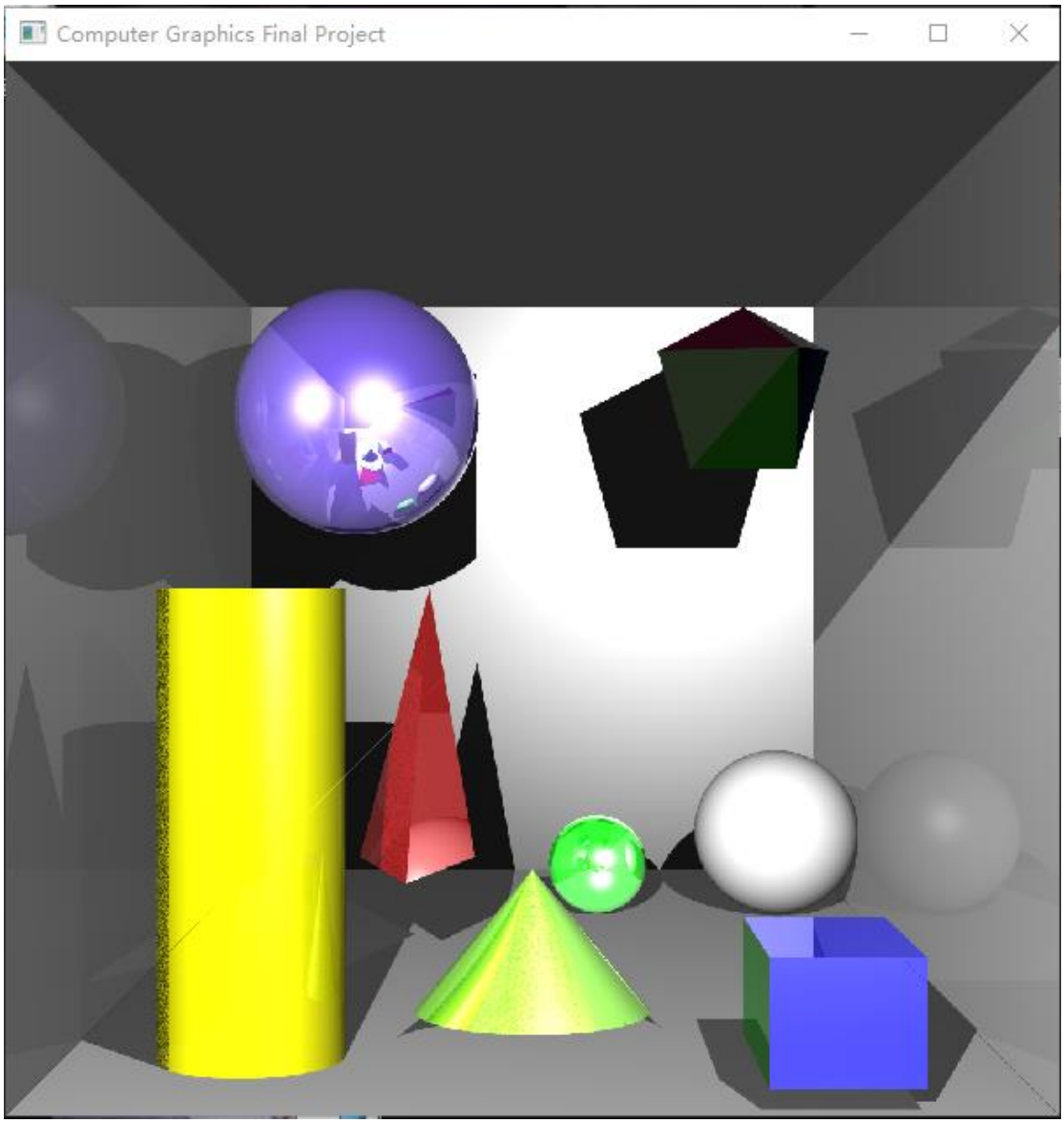
```
        {
            if (q.point.dist(shadow2.point) < q.point.dist(light2)){
                colorSum = col.phongLight(backgroundCol, l2Dotn/6, 0);
            }
        }
```

以下代码定义了场景中需要使用到的模型种类个数以及位置颜色等信息：

```
void initialize()
{
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(XMIN, XMAX, YMIN, YMAX);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glClearColor(0, 0, 0, 1);
    //这里定义了三个球体的大小、位置和颜色信息
    Sphere *sphere0 = new Sphere(Vector(4, -17, -130), 3.0, Color::GREEN);
    sceneObjects.push_back(sphere0);
    Sphere *sphere1 = new Sphere(Vector(15, -15, -130), 5.0, Color::WHITE);
    sceneObjects.push_back(sphere1);
    Sphere *sphere2 = new Sphere(Vector(-10, 10, -120), 7.0, Color(0.3,0.2,0.7));
    sceneObjects.push_back(sphere2);
    //这里定义了圆柱体和圆锥体的大小、 位置和颜色信息
    Cylinder *cylinder3 = new Cylinder(Vector(-12, -20, -90), 4, 20.0, Color::YELLOW);
    sceneObjects.push_back(cylinder3);
    Cone *cone4 = new Cone(Vector(0, -18, -90), 5.0, 6.0, Color(0.5,0.9,0.2));
    sceneObjects.push_back(cone4);
    //这里定义了场景中几个平面的位置和颜色信息
    Plane *plane5 = new Plane(Vector(-20,-20,-30),Vector(20,-20,-30),Vector(20,-20,-
150),Vector(-20,-20,-150), Color(1,1,1));
    sceneObjects.push_back(plane5);
    Plane *plane_B_6 = new Plane(Vector(-20,-20,-150),Vector(20,-20,-150),Vector(20,20,-
150),Vector(-20,20,-150), Color::GRAY);
    sceneObjects.push_back(plane_B_6);
    Plane *plane_L_7 = new Plane(Vector(-20,-20,-30),Vector(-20,-20,-150), Vector(-20, 20,-
150),Vector(-20, 20,-30), Color(1,1,1));
    sceneObjects.push_back(plane_L_7);
    Plane *plane_R_8 = new Plane(Vector(20,20,-30),Vector(20,20,-150), Vector(20, -20,-
150),Vector(20, -20,-30), Color(1,1,1));
    sceneObjects.push_back(plane_R_8);
    Plane *plane9 = new Plane(Vector(-20,20,-30),Vector(20,20,-30),Vector(20,20,-
150),Vector(-20,20,-150), Color(1,1,1));
    sceneObjects.push_back(plane9);
    //使用几个平面定义了一个立方体和具体的位置颜色等信息
    Vector a = Vector(9,-14,-80);Vector b = Vector(15,-14,-80); Vector c = Vector(15,-14,-90);
```

```
Vector d = Vector(9,-14,-90);
    Vector e = Vector(9,-19,-80); Vector f = Vector(15,-19,-80); Vector g = Vector(15,-19,-90);
Vector h = Vector(9,-19,-90);
    Plane *front_10 = new Plane(a,e,f,b,Color::BLUE);
    sceneObjects.push_back(front_10);
    Plane *back_11 = new Plane(d,c,g,h,Color::RED);
    sceneObjects.push_back(back_11);
    Plane *top_12 = new Plane(a,b,c,d,Color::BLUE);
    sceneObjects.push_back(top_12);
    Plane *bottom_13 = new Plane(e,f,g,h,Color::BLUE);
    sceneObjects.push_back(bottom_13);
    Plane *left_14 = new Plane(h,e,a,d,Color::GREEN);
    sceneObjects.push_back(left_14);
    Plane *right_15 = new Plane(f,g,c,b,Color::GREEN);
    sceneObjects.push_back(right_15);
    //此处定义了一个四面体的位置和颜色等信息
    Vector ta = Vector(-6,-14,-100); Vector tb = Vector(-3,-14,-110); Vector tc = Vector(-9,-14,-
110); Vector top = Vector(-5.5,0,-112.5);
    Tetrahedron *left_16 = new Tetrahedron(ta,top,tc,Color::RED);
    sceneObjects.push_back(left_16);
    Tetrahedron *right_17 = new Tetrahedron(ta,tb,top,Color::RED);
    sceneObjects.push_back(right_17);
    Tetrahedron *back_18 = new Tetrahedron(tc,tb,top,Color::BLACK);
    sceneObjects.push_back(back_18);
    Tetrahedron *bottom_19 = new Tetrahedron(ta,tb,tc,Color::BLACK);
    sceneObjects.push_back(bottom_19);
    //此处定义了一个五棱锥的位置和颜色等信息
    Vector pa = Vector(12,16,-120); Vector pb = Vector(16.86,13.48,-120); Vector pc =
Vector(15,6.8,-120);
    Vector pd = Vector(9,6.8,-120); Vector pe = Vector(7.14,13.48,-120); Vector px =
Vector(12,10.9,-95);
    Pentagonal_pyramid *plane_1 = new Pentagonal_pyramid(pa,pb,px,Color(0.4,0.3,0.1));
    sceneObjects.push_back(plane_1);
    Pentagonal_pyramid *plane_2 = new Pentagonal_pyramid(pb,pc,px,Color(0.1,0.5,0.9));
    sceneObjects.push_back(plane_2);
    Pentagonal_pyramid *plane_3 = new Pentagonal_pyramid(pc,pd,px,Color(0.2,0.8,0.1));
    sceneObjects.push_back(plane_3);
    Pentagonal_pyramid *plane_4 = new Pentagonal_pyramid(pd,pe,px,Color(0.7,0.9,0.6));
    sceneObjects.push_back(plane_4);
    Pentagonal_pyramid *plane_5 = new Pentagonal_pyramid(pe,pa,px,Color(0.8,0.1,0.4));
    sceneObjects.push_back(plane_5);
}
```

3. 实验结果



从实验结果来看，项目基本上实现了 project 的要求，两个光源及其阴影，transparency 和 refraction 也得到了简单实现，项目也包含了基本的 polyhedral and spherical objects。

4. 遇到的问题和解决办法

1）一开始在其它的平台上配置环境老是出错，最后发现用 VS2017 是很方便的。

2）对于一些多面体的定义不太了解，在 Stack Overflow 上找了一些类似的代码进行学习。

3）在项目的程序设计中，坐标计算比较麻烦，对于多面体比如五棱锥需要单独计算它每个点的坐标，可以通过编写程序计算来完成。

4）关于 transparency 和 refraction 参考了 GitHub 上一些相关代码的经验，同时也翻阅了《全局光照技术》，对其进行了一定的了解。