计算机图形学实验报告

Assignment0

课程名称： 计算机图形学

实验名称： Assignment0

学生学院： 泰山学堂

学生班级： 2017级计算机取向

学生学号： 201705301350

学生姓名： 宋建涛

提交日期： 2019.10.26

目录

[实验目标 3](#_Toc27032995)

[实验概览 3](#_Toc27032996)

[读取模型文件并保存相关数据 3](#_Toc27032997)

[认识obj模型 3](#_Toc27032998)

[存储内容的结构 4](#_Toc27032999)

[将读取的模型通过OpenGL重新绘制 6](#_Toc27033000)

[初始化 6](#_Toc27033001)

[绘制函数 7](#_Toc27033002)

[适应窗口改变函数 8](#_Toc27033003)

[实现模型颜色的改变 9](#_Toc27033004)

[实验代码结构 9](#_Toc27033005)

[实验结果和收获 10](#_Toc27033006)

[实验源代码 10](#_Toc27033007)

# 实验目标

本实验是图形学系列试验中的第一个实验，本实验的实验要求主要由两个：

1. 读一个三角形网格数据文件（包含网格纹理）
2. 使用OpenGL绘制该模型（要求能够实现颜色的变换和简单的移动）

# 实验概览

首先根据实验要求，我们可以将大实验简单的分为四个阶段：

1. 读取模型文件并保存相关数据
2. 将读取的模型文件通过OpenGL绘制出来
3. 实现模型颜色的改变
4. 实现模型简单的移动（旋转）

接下来我们就分这四部分来看一下我们需要做什么。

## 读取模型文件并保存相关数据

### 认识obj模型

OBJ文件是Wavefront公司为它的一套基于工作站的3D建模和动画软件"Advanced Visualizer"开发的一种文件格式，这种格式同样也以通过Maya读写。

OBJ文件是一种文本文件，可以直接用写字板打开进行查看和编辑修改。

Obj文件中每一行的行首通过关键字来说明本行说明的内容类型，常见的关键字如下

1.v顶点

2.vt纹理坐标

3.vn顶点法向量

4.f 面

一、顶点

格式：v x y z

意义：每个顶点的坐标

二、纹理坐标

格式：vt u v w

意义：绘制模型的三角面片时，每个顶点取像素点时对应的纹理图片上的坐标。纹理图片的坐标指的是，纹理图片如果被放在屏幕上显示时，以屏幕左下角为原点的坐标。

注意：w一般用于形容三维纹理，大部分是用不到的，基本都为0。

三、顶点法向量

格式：vn x y z

意义：绘制模型三角面片时，需要确定三角面片的朝向，整个面的朝向，是由构成每个面的顶点对应的顶点法向量的做矢量和决定的（xyz的坐标分别相加再除以3得到的）。

四、面

格式 ：f v/vt/vn v/vt/vn v/vt/vn（f 顶点索引 / 纹理坐标索引 / 顶点法向量索引）

意义：绘制三角面片的依据，每个三角面片由三个f构成，由f可以确定顶点、顶点的对应的纹理坐标（提取纹理图片对应该坐标的像素点）、通过三个顶点对应的顶点法向量可以确定三角面的方向。

补充：有些模型可能会出现四边形的绘制方式，那样的模型关于面的数据描述是这样的 f v/vt/vn v/vt/vn v/vt/vn v/vt/vn ，比三角面绘制方式多一项数据。

### 存储内容的结构

为了存储从文件中读取到的信息，我们一共定义了三种结构如下，

1. **template**<**typename** T>
2. **class** Ternary {
3. **public**:
4. T x, y, z;
5. Ternary() {};
6. Ternary(T a, T b, T c);
7. };
9. **class** Surface {
10. **public**:
11. Ternary<**int**> vertex;
12. Ternary<**int**> texture;
13. Ternary<**int**> normal;
14. Surface(Ternary<**int**>, Ternary<**int**>, Ternary<**int**>);
15. };
17. **class** ObjLoader {
18. **public**:
19. vector<Ternary<**float**>> obj\_vertexe;
20. vector<Ternary<**float**>> obj\_texture;
21. vector<Ternary<**float**>> obj\_normal;
22. vector<Surface> obj\_surface;
23. ObjLoader(string filename);//构造函数
24. };

其中Ternary仅代表一个包含三个数据的结构，它可以使float类型的也可以是string类型的。

Surface存储着从文件中读取到的面结构，在之前的介绍中我们简单了解了obj文件中对三角面片结构的描述，对每一个面片我们要存储三个点，每个点都有它的顶点索引、顶点纹理索引、顶点法向量索引，所以我们在Surface中定义了三个Ternary<int>类型的变量来分别存储这些信息。

ObjLoader存储着我们从obj文件中读取出来的所有信息，包含顶点坐标obj\_vertexe、顶点纹理坐标obj\_texture、顶点法向坐标obj\_normal和三角面片信息obj\_surface。知道了这些信息我们就能使用OpenGL重绘读取的模型了。

读取文件并存储的函数实现是在ObjLoader的重构函数中，我们期望能够在模型定义的过程中完成obj文件的读取并存储操作，ObjLoader的重构函数实现如下：

1. ObjLoader::ObjLoader(string filename)
2. {
3. string line;
4. fstream f;
5. f.open(filename, ios::in);
6. **if** (!f.is\_open()) {
7. cout << "Something Went Wrong When Opening Objfiles" << endl;
8. }
10. **while** (!f.eof()) {
11. getline(f, line);//拿到obj文件中一行，作为一个字符串
12. vector<string> data;
13. split(line, data);
14. **if** (data.size() < 4) {
15. **continue**;
16. }
17. **if** (data[0] == "v") {//读入的一行是点的坐标信息
18. obj\_vertexe.push\_back(Ternary<**float**>(stringToNum<**float**>(data[2]), stringToNum<**float**>(data[3]), stringToNum<**float**>(data[4])));
19. }
20. **else** **if**(data[0] == "vn"){//读入的一行是点的法向量
21. obj\_normal.push\_back(Ternary<**float**>(stringToNum<**float**>(data[1]), stringToNum<**float**>(data[2]), stringToNum<**float**>(data[3])));
22. }
23. **else** **if** (data[0] == "vt") {//读入的一行是点的纹理坐标
24. obj\_texture.push\_back(Ternary<**float**>(stringToNum<**float**>(data[1]), stringToNum<**float**>(data[2]), stringToNum<**float**>(data[3])));
25. }
26. **else** **if** (data[0] == "f") {//读入的一行是面的信息
27. vector<string> tmp;
28. Ternary<**int**> tmp\_surface;
29. Ternary<**int**> tmp\_texture;
30. Ternary<**int**> tmp\_vn;
31. split(data[1], tmp, '/');
32. tmp\_surface.x = stringToNum<**int**>(tmp[0]);
33. tmp\_texture.x = stringToNum<**int**>(tmp[1]);
34. tmp\_vn.x = stringToNum<**int**>(tmp[2]);
36. split(data[2], tmp, '/');
37. tmp\_surface.y = stringToNum<**int**>(tmp[0]);
38. tmp\_texture.y = stringToNum<**int**>(tmp[1]);
39. tmp\_vn.y = stringToNum<**int**>(tmp[2]);
41. split(data[3], tmp, '/');
42. tmp\_surface.z = stringToNum<**int**>(tmp[0]);
43. tmp\_texture.z = stringToNum<**int**>(tmp[1]);
44. tmp\_vn.z = stringToNum<**int**>(tmp[2]);
46. obj\_surface.push\_back(Surface(tmp\_surface, tmp\_texture, tmp\_vn));
47. }
48. //cout << "ok" << endl;
49. }
50. f.close();
51. }

## 将读取的模型通过OpenGL重新绘制

在这个阶段我们主要需要做的事如下：定义相机顶点坐标、创建窗口并初始化OpenGl窗口参数、对模型进行绘制。

### 初始化

首先相机坐标的定义我们可以用到之前的Ternary结构，毕竟三维空间中说明一个点也是用到三个数字，只不过我们这里需要的坐标是float类型。

之后我们需要进行初始化操作，在这里我们把初始化操作封装成函数void init():

1. **void** init() {
2. camera.x = -20;
3. camera.y = 30;
4. camera.z = 20;
5. glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);
6. glutInitWindowSize(500, 500);
7. glutInitWindowPosition(100, 100);
8. glutCreateWindow("ObjLoader");
10. //材质反光性设置
11. GLfloat mat\_specular[] = { 1.0, 1.0, 1.0, 1.0 };  //镜面反射参数
12. GLfloat mat\_shininess[] = { 50.0 };               //高光指数
13. GLfloat light\_position[] = { 1.0, 1.0, 1.0, 0.0 };
14. GLfloat white\_light[] = { 1.0, 1.0, 1.0, 1.0 };   //灯位置(1,1,1), 最后1-开关
15. GLfloat Light\_Model\_Ambient[] = { 0.2, 0.2, 0.2, 1.0 }; //环境光参数
17. glClearColor(0.0, 0.0, 0.0, 0.0);  //背景色
18. glShadeModel(GL\_SMOOTH);           //多变性填充模式
20. //材质属性
21. glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);
22. glMaterialfv(GL\_FRONT, GL\_SHININESS, mat\_shininess);
24. //灯光设置
25. glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);
26. glLightfv(GL\_LIGHT0, GL\_DIFFUSE, white\_light);   //散射光属性
27. glLightfv(GL\_LIGHT0, GL\_SPECULAR, white\_light);  //镜面反射光
28. glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, Light\_Model\_Ambient);  //环境光参数
30. glEnable(GL\_LIGHTING);   //开关:使用光
31. glEnable(GL\_LIGHT0);     //打开0#灯
32. glEnable(GL\_DEPTH\_TEST); //打开深度测试
33. }

### 绘制函数

Glut运行过程的绘制函数原型为void glutDisplayFunc(void (\*func)(void));这里需要我们写的是绘制函数void(\*func)(void),在我们函数中的体现就是绘制函数void Draw()。我们来看一下代码：

1. **void** Draw() {
3. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
4. //开始绘制
5. **for** (**int** i = 0; i < objModel.obj\_surface.size(); i++) {
6. glBegin(GL\_TRIANGLES);
8. Ternary<**float**> normal = objModel.obj\_normal[objModel.obj\_surface[i].normal.x - 1];
9. glNormal3f(normal.x, normal.y, normal.z);
10. Ternary<**float**> texture = objModel.obj\_texture[objModel.obj\_surface[i].texture.x - 1];
11. glTexCoord3f(texture.x, texture.y, texture.z);
12. Ternary<**float**> vertex = objModel.obj\_vertexe[objModel.obj\_surface[i].vertex.x - 1];
13. glVertex3f(vertex.x, vertex.y, vertex.z);
15. normal = objModel.obj\_normal[objModel.obj\_surface[i].normal.y - 1];
16. glNormal3f(normal.x, normal.y, normal.z);
17. texture = objModel.obj\_texture[objModel.obj\_surface[i].texture.y - 1];
18. glTexCoord3f(texture.x, texture.y, texture.z);
19. vertex = objModel.obj\_vertexe[objModel.obj\_surface[i].vertex.y - 1];
20. glVertex3f(vertex.x, vertex.y, vertex.z);
22. normal = objModel.obj\_normal[objModel.obj\_surface[i].normal.z - 1];
23. glNormal3f(normal.x, normal.y, normal.z);
24. texture = objModel.obj\_texture[objModel.obj\_surface[i].texture.z - 1];
25. glTexCoord3f(texture.x, texture.y, texture.z);
26. vertex = objModel.obj\_vertexe[objModel.obj\_surface[i].vertex.z - 1];
27. glVertex3f(vertex.x, vertex.y, vertex.z);
29. glEnd();
30. }
31. glFlush();
32. }

### 适应窗口改变函数

因为在本次试验中我们可能会改变观察视角和窗口大小，所以重绘函数void reshape()是很重要的，本函数主要完成的事情包括判断窗口大小是否在规定范围内、在视角改变时重新计算相机坐标和视角开度，话不多说，直接上代码：

1. **void** reshape(**int** w, **int** h)
2. {
3. **float** size = 80;
4. glViewport(0, 0, (GLsizei)w, (GLsizei)h);
6. glMatrixMode(GL\_PROJECTION);
7. glLoadIdentity();
8. **if** (w <= h)
9. glOrtho(-size, size, -size \* (GLfloat)h / (GLfloat)w, size\*(GLfloat)h / (GLfloat)w, -size, size);
10. **else**
11. glOrtho(-size \* (GLfloat)w / (GLfloat)h, size\*(GLfloat)w / (GLfloat)h, -size, size, -size, size);
13. //设置模型参数--几何体参数
14. gluLookAt(camera.x, camera.y, camera.z, 0, 20, 0, 0, 1, 0);
15. glMatrixMode(GL\_MODELVIEW);
16. glLoadIdentity();
17. }

## 实现模型颜色的改变

这里我们使用鼠标右键点击控制模型颜色改变，主要做法是右击是用随机数产生三个0~1数字，分别替代原画笔颜色重的RGB值，本部分用到的代码如下：

1. default\_random\_engine e;
2. uniform\_real\_distribution<**double**> u(0, 1);
3. **if** (button == GLUT\_RIGHT\_BUTTON && state == GLUT\_DOWN)
4. {
5. glColor3f(u(e), u(e), u(e));
6. Draw();
7. }

在改变画笔颜色之后，再重绘模型时就会导致模型颜色的改变。

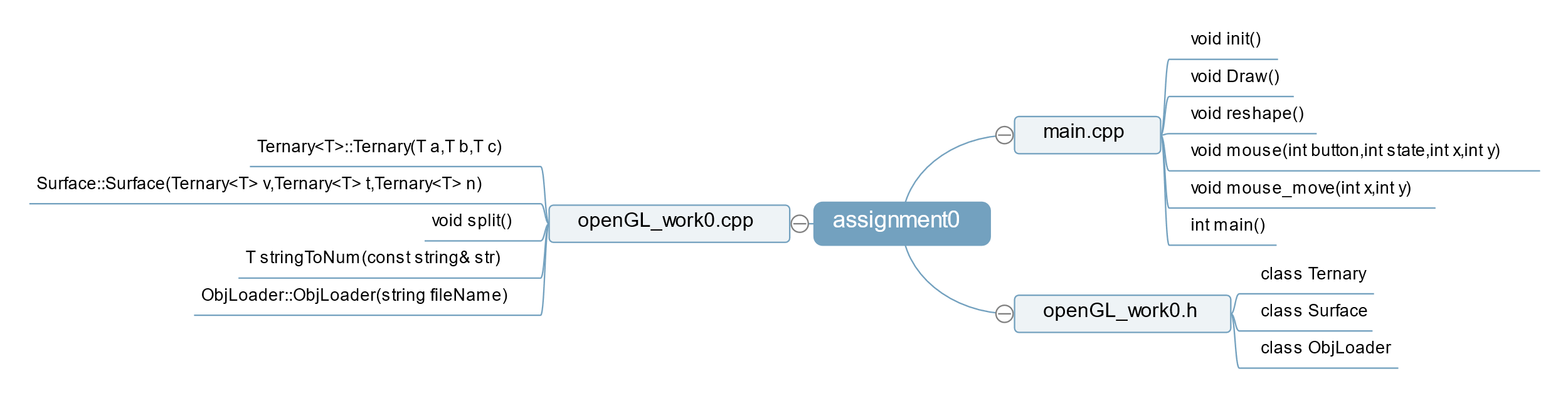
## 实现控制模型移动（旋转）

在本实验中，我们控制模型旋转（移动）的本质实际上是对相机坐标和角度的变化来实现的，在这里我们使用一组坐标past\_x和past\_y来记录鼠标左键按下时的坐标，然后在鼠标移动事件检测函数void mouse\_move(x,y)中根据当前鼠标坐标x,y和之前坐标past\_x，past\_y的差来判断模型应该旋转的角度。Void mouse\_move()函数的实现如下：

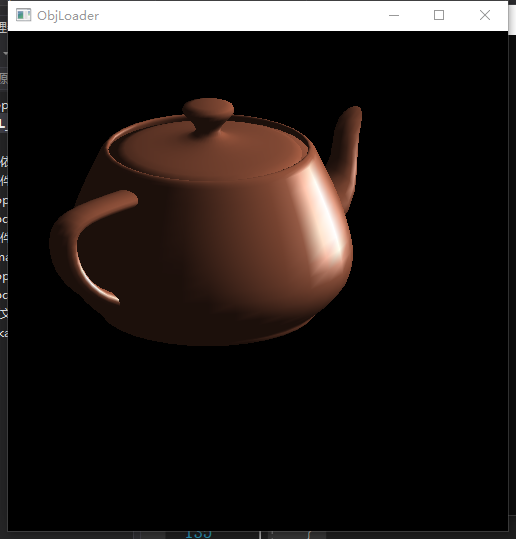
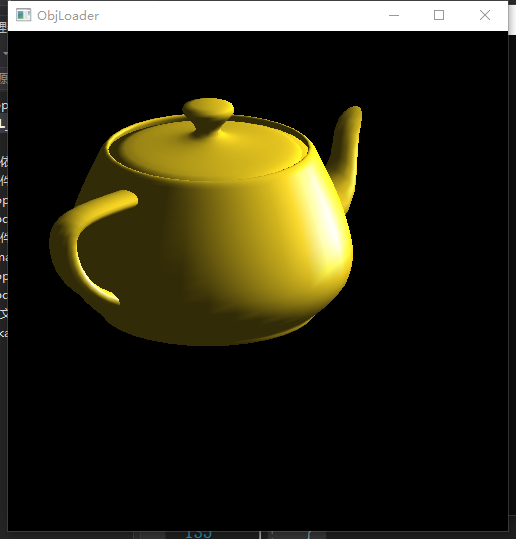
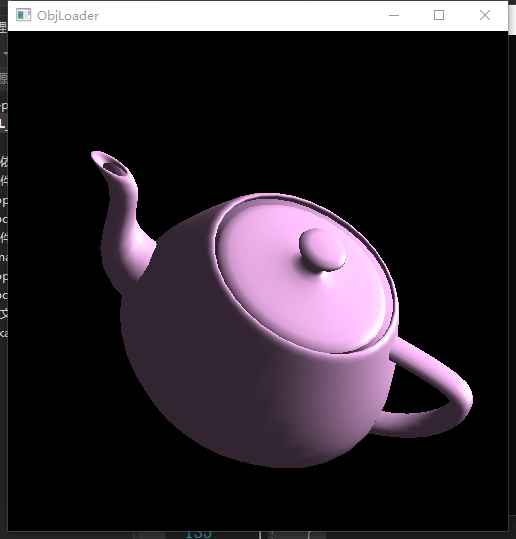
1. **void** mouse\_move(**int** x, **int** y)
2. {
3. **float** speed = 0.3;
4. **if** (get\_pos == 1)
5. {
6. glRotatef(speed\*(x - past\_x), 0, 1, 0);
7. glRotatef(speed\*(y - past\_y), 1, 0, 0);
8. Draw();
9. past\_x = x;
10. past\_y = y;
11. }
12. }

至此，我们所有得小目标都已实现，将所有的代码组合起来我们就能得到可以运行的代码。

# 实验代码结构



# 实验结果和收获

上面三张图片从左至右依次展示了模型显示、颜色变化、模型变换的功能。

通过本次实验，我对上课屠老师讲解的图形学基本知识有了初步的了解，了解了模型文件obj的存储方式，复习了鼠标事件的使用方法，为接下来的几个大实验起到了抛砖引玉的作用。

# 实验源代码

https://github.com/yuemos/Computer-Graphics.git