1. **目录**

[1．引言 2](#_Toc123394531)

[2. 主体介绍 2](#_Toc123394532)

[2.1 程序简介 2](#_Toc123394533)

[2.2 算法 2](#_Toc123394534)

[2.3 实现细节 4](#_Toc123394535)

[3. 结果与讨论 8](#_Toc123394536)

[3.1 结果 8](#_Toc123394537)

[3.2 讨论 11](#_Toc123394538)

# 1．引言

本次项目使用Visual Studio 2022进行实现，可加载obj文件,并对模型进行键盘和鼠标控制，包括但不限于平移、旋转。并添加了bezier曲面，场景中添加光照，bezier曲面上添加纹理贴图以达到个性化效果。

# 2. 主体介绍

## 2.1 程序简介

具体功能如下：

1. 加载obj文件。
2. 对模型加键盘控制，实现模型的平移和旋转。例如，按压右向箭头，模型向右移动。具体按键和其对应的功能自行设定。
3. 添加鼠标控制，主要包含以下两个功能：
   * + 1. 拖动鼠标左键实现模型的旋转，而不改变观看视角。
       2. 拖动鼠标右键实现观看视角的改变，模型本身保持固定。
4. 添加bezier曲面，至少包含5\*5个控制点
5. 为bezier曲面添加纹理贴图，图片自行选择。
6. 在场景中添加ambient light, diffuse light and specular light进行组合来产生不同的光照效果。
7. 场景中的图形能分别显示顶点模式，面片模式和线框模式。

## 2.2 算法

**1. 坐标变换**

在图形学中，坐标系分为局部坐标系、世界坐标系、视线坐标系、裁剪坐标系、屏幕坐标系。在OpenGL中，我们只需处理局部坐标系、世界坐标系、视线坐标系。首先，我们可以对物体（Object）进行各种各样的变换（平移、旋转、缩放等），将这些变换依次相乘，得到。接着，我们需要根据相机的位置坐标、相机朝向的坐标、“向上”方向的向量（以上变量均为世界坐标系中的变量）得到。然后，我们根据视角大小、窗口比例、近平面距离、远平面距离获得。最后，我们只需将上述矩阵相乘（），并将结果发送到对应的顶点着色器，让顶点着色器将结果与点相乘（）即可得到这个点经过变换后的坐标。

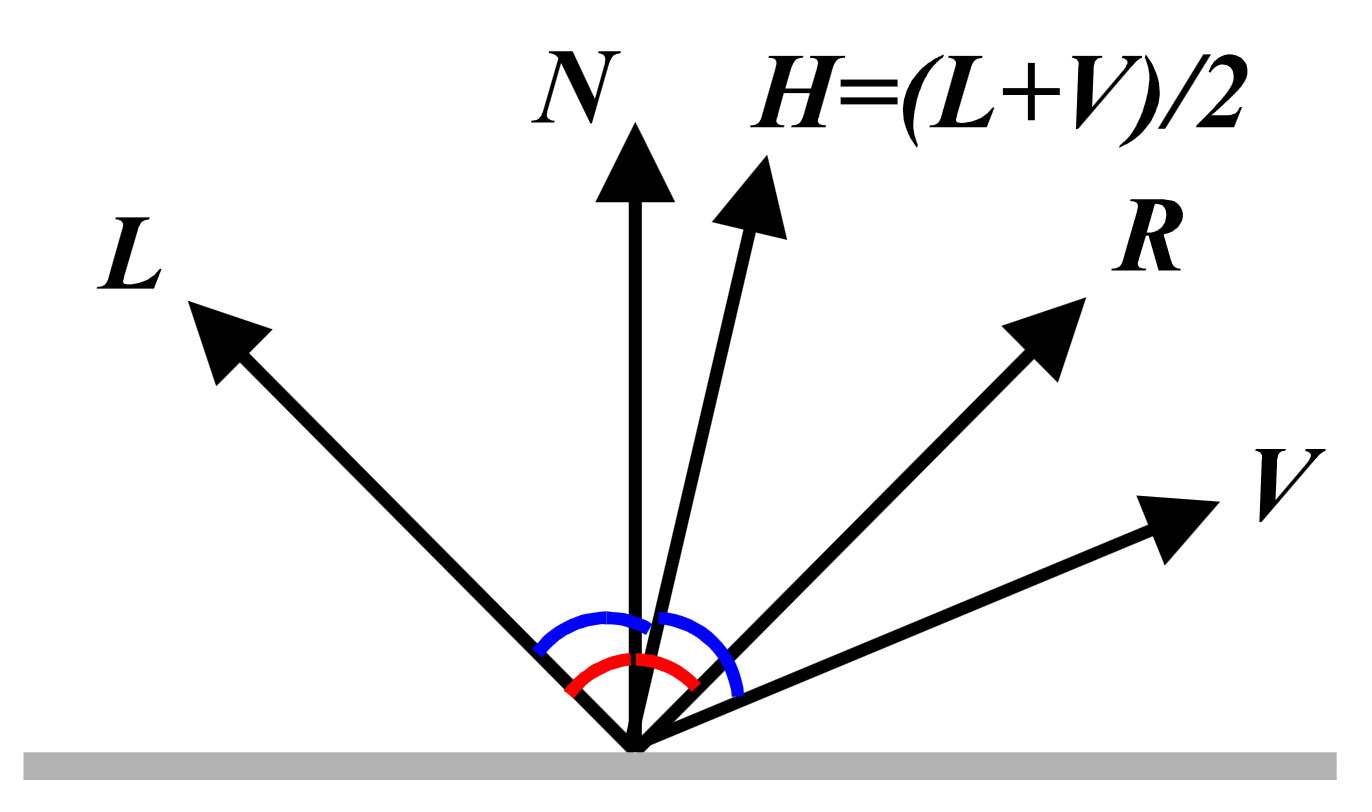
**2. Phong光照模型**

对于漫反射，我们采用Lambert模型进行渲染。Lambert模型假设物体表面足够光滑，光的分布只由物体的几何形状决定，且没有高光区域。Lambert模型的公式为。其中表示标准化的入射光线方向，表示标准化的面法向量，表示反射的颜色（），表示入射光的光强。

对于镜面反射，我们采用Phong模型进行渲染。Phong模型假设光的分布依赖于物体的材质属性，取决于光线的方向和观察的方向。Phong模型中的系数，其中代表标准化的反射向量，代表标准化的观察向量。

对于环境光，我们假设环境光为一个常量。

在Phong光照模型中，包含环境光常量、漫反射和镜面反射。公式如下：



**3. Bezier曲面**

Bezier曲线是用Pierre Bezier的名字命名的，该曲线由一组控制点定义。设控制点为，则曲线方程

其中称为Bernstein多项式

Bernstein多项式中的每一项的大小都在，多项式的各项之和为1。Bezier曲线位于控制点的凹处。当控制点的个数过多时，多项式的次数会很大，且曲线难以控制。

与Bezier曲线类似，Bezier曲面的方程为：

一个次的Bezier曲面有个控制点。

## 2.3 实现细节

本项目对obj文件中的数据进行了封装，将其分为Vertex（点）、Plane（平面）、Object（物体）。在obj文件中，“点”表示为“v (坐标){3}，“面”使用点的索引、点的纹理坐标索引、点的法向量索引，表示为“f (点的索引)/(点的纹理坐标索引)/(点的法向量索引)”。点的纹理坐标表示为“vt (坐标){2}”。点的法向量表示为“vn (坐标){3}”。本次作业中未涉及点的纹理坐标和点的法向量，故未对它们做处理。Vertex（点）类中保存了点的3个坐标（x，y，z），不接受绝对值≥1的坐标，可以将Vertex实例与Object绑定。Plane（平面）类中保存了点的索引、点的纹理坐标索引、点的法向量索引。可以将Plane实例与Object绑定。Plane实例不接受不合法的索引（索引的值超过当前绑定的Object中的对应元素的个数）。Object（物体）实例中保存了组成该物体的点、面等信息，以及该物体当前状态对应的变换矩阵，可进行一些基本变换（平移、旋转等，这些变换在物体的当前状态上叠加，平移使用glm::translate()，旋转使用glm::rotate()），可以获得该物体的原点在世界坐标系中的坐标，以及该物体绕世界坐标系中的各个坐标转过的角度。Object类提供了从obj文件生成实例的方法。可以向Object类添加着色器文件。Object类还提供了draw()方法，初始化VBO、EBO，生成VAO以及编译出ShaderProgram，其中VAO和ShaderProgram可以从外部获取。

对GLFW的相关操作封装在Scene类中。Scene类设计为单例模式，至多允许创建一个实例。使用Scene::init()来初始化GLFW库，使用Scene::createWindow()创建一个窗口，使用Scene::start()来开始渲染。可以向Scene的实例添加按键/鼠标/渲染回调函数，这些回调函数会在渲染循环块中被调用。

**读取obj文件：**根据obj文件的每一行的开头来判断内容。若开头为“#”，则该行为注释，忽略。若开头为“v”，则表明这一行的内容为点的3个坐标，将其依次读入，生成Vertex类实例，并将该实例添加到当前Object实例中。若开头为“f”，则表明这一行的内容为面。此处使用StringUtil::stringSplit()对内容按照正则表达式分割，提取出点的索引、点的纹理坐标索引、点的法向量索引，生成Plane实例，并将该实例添加到当前Object实例中。

**模型的键盘控制：**键盘控制的回调函数为main.cpp中的void onRender(GLFWwindow\* window, Object& object) 。通过glfwGetKey()函数获取指定的按键是否被按下，若按下则执行指定操作。按“Esc”为重置变换，按“→”为向右平移，按“←”为向左平移，按“↑”为向上平移，按“↓”为向下平移，按“O”为向里平移，按“I”为向外平移，按“W”为绕x轴负方向旋转，按“S”为绕x轴正方形旋转，按“A”为绕y轴负方向旋转，按“D”为绕y轴正方向旋转，按“Q”为绕z轴正方向旋转，按“E”为绕z轴负方向旋转。以上变换均调用Object实例的函数。

**模型的鼠标控制：**鼠标控制的回调函数为main.cpp中的void onMouseAction(GLFWwindow\* window, Object& object) 。使用glfwGetMouseButton()获取当前鼠标按下的键，使用glfwGetCursorPos()获取当前鼠标的坐标。变量lastCursorX、lastCursorY表示鼠标上一时刻的坐标，变量cameraX、cameraY表示相机上一时刻的坐标。按下鼠标左键时，物体会分别绕x、y轴旋转以当前鼠标的坐标与上一时刻坐标之差表示的角度，此处使用Object实例的函数。按下鼠标右键时，相机会根据鼠标移动的方向来进行移动（xOy平面）。此处使用glm::lookAt()。我添加了防误触功能，当坐标变化超过50px时不做任何操作。

执行上述操作后，调用main.cpp中的渲染回调函数void onRender(GLFWwindow\* window, Object& object) 。该函数将的结果传递给顶点着色器，并进行相关的绘制操作。

对于Bezier曲面，我们使用Tessellation来绘制。Tessellation可以让我们用较粗糙的模型来绘制较精细的图案，可以绘制更光滑的轮廓，可以根据需求来更改图像的精细程度。

Tessellation使用新的类型——Patch。Patch类型记做GL\_PATCH。N个控制点可以作为一个新的Patch。Patch中控制点的数量可以用glPatchParameteri来定义。

在Tessellation中，我们使用Tesselation Control Shader(TCS)、Primitive Generator(PG)和Tessellation Evaluation(TES)。

TCS的输入为一组控制点，输出为Patch和Tessellation Levels（即应该为此Patch生成多少三角形）。PG是一个生成新的原始类型的固定功能的阶段。该阶段由Tessellation Levels（默认值或由TCS提供）、细分后的点的间距、输入的原始类型的类别、原始类型的生成顺序决定（后3个参数由TES提供）。细分后的点的坐标保存在内置的vec3 gl\_TessCoord中。TES使用PG提供的gl\_TessCoord和TCS的输出（Patch）来计算细分产生的点，并输出新的点的坐标和贴图坐标。

对于25个控制点的Bezier曲面，点的坐标有以下公式：

对于贴图，我们应该为模型的每一个顶点指定一个对应的贴图坐标。对于模型的其他点，我们采用插值的方式来计算贴图坐标。贴图坐标的x和y的取值都在，其中左下角为，右上角为。对于超出贴图坐标的部分，采用GL\_REPEAT模式处理。为了减少贴图失真，采用了多级渐远纹理。对于贴图的缩小采用GL\_LINEAR\_MIPMAP\_LINEAR，对于贴图的放大采用GL\_LINEAR。我们使用SOIL（Simple OpenGL Image Library）来加载贴图。在片段着色器中，使用sampler2D类型来保存贴图。

对于光照，我们需要模型中每一个顶点的法向量。对于Bezier曲面，计算，其中为Bezier曲面方程，为模型的变换矩阵。若模型的变换中含有缩放操作，则应将改为。对于普通的模型，我们需要读取obj文件中的点法向量（以vn开头）。Obj文件的点法向量可由MeshLab软件生成。需要注意的是，OpenGL在计算反射向量时，reflect函数的第一个变量为光线方向向量，且方向由光源指向反射点。若模型使用了贴图，则模型的颜色为贴图的颜色而不是某个固定的颜色。

本项目添加了Shader类、Camera类、Texture类。Shader类中存储了着色器的编号、来源、类型，可以对着色器进行编译。Texture类中存储了贴图的编号、s（或t）方向超出贴图坐标范围的处理方式、放大/缩小贴图的处理方式，可以加载贴图。Camera类中存储了相机的位置、相机的朝向、相机对“上”方向的定义，可以生成视角矩阵。Shader类和Texture类被Object类使用，Camera类被Scene类使用。在Object类中适当的地方使用多线程来加快数据的处理（从封装的对象中读取数据并生成std::vector）。

# 3. 结果与讨论

## 3.1 结果

鼠标左键：

图片包含 电脑, 游戏机, 监控, 桌子

描述已自动生成

键盘操作：

图片包含 游戏机, 电脑, 食物

描述已自动生成

鼠标右键：

图片包含 游戏机, 食物

描述已自动生成

线框模式：

图片包含 黑暗, 游戏机, 监控, 灯光

描述已自动生成

顶点模式：

图片包含 黑暗, 游戏机, 监控, 灯光

描述已自动生成

## 3.2 讨论

1. 对象之间相互调用导致链接错误

在两个类的头文件中互相声明对方，且在函数声明中只使用对方的指针。

2. 对象的计数功能不能正常工作

原本判断索引是否合法是判断索引大小是否小于点（Vertex）的个数，Vertex的个数记录在Vertex类的static变量中，运行时个数变成了负数。这是由于vector的push\_back()的机制引起的。使用push\_back()往vector里添加对象时，添加的是对象的副本。在vector析构时，里面的每个对象总共会析构3次。后来增加了把点（Vertex）和平面（Plane）添加到物体（Object）中的功能，让Object的实例来对点（Vertex）的个数进行计数，进而来判断索引是否合法。

3. glBufferData()不兼容new出来的数组

我们无法预先确定obj文件中点和面等变量的个数，因此传入glBufferData()的数组长度也无法确定。若传入new出来的数组，则glBufferData()无法获取数组中的数据，导致无法绘制物体。后来改用vector的下标访问来代替数组。数组的元素个数为vector::size()，数组的起始地址为&vector::operator[](0)。

3. 读取obj文件速度慢

本项目读取了含顶点法向量的eight.uniform.obj，由于文件较大，导致程序的启动时间长达12秒。我将文件按行读取改成先把文件一次性读取到缓存中（如stringstream），再进行后续操作，并将从Vertex类、Plane类中读取模型信息的操作进行了并行化处理，即读取顶点坐标、读取顶点坐标索引、读取顶点法向量、读取顶点法向量索引分别在独立的线程进行。std::thread在使用成员函数时，应使用std::mem\_fn来传递函数，并多加一个this参数。最后将编译参数改为/O2（最大优化（优选速度）），启动时间缩短至4秒。