浙江大学

面向对象程序设计

大 程 序 报 告

大程名称：基于 Opengl 的 3D 建模

学号：3220105108

指导老师：李际军

2023~2024 秋冬学期 2023 年 12 月 31 日

1 大程序简介

1.1 选题背景及意义

随着计算机图形学的发展，3D建模在多个领域变得越来越重要，包括游戏开发、动画制作、工业设计、建筑规划等。传统的3D建模软件往往功能复杂且成本高昂，这对于初学者和非专业人士来说可能是一个障碍。因此，开发一个简单、易用且功能强大的3D建模工具具有重要意义。这样的工具不仅能够帮助初学者更好地理解和学习3D建模的基本概念，还能为专业人士提供一个快速原型设计的平台。

此外，OBJ文件格式作为一种广泛使用的标准3D模型格式，其在3D建模软件中的支持是必不可少的。能够读取和保存OBJ文件的能力，使得该程序可以与其他3D建模和渲染软件兼容，极大地提高了其实用性和灵活性。

1.2 目标要求

本项目的目标是开发一个基于OpenGL的3D建模程序，它应该满足以下要求：

- **基本图形建模**：能够创建和操作基本的3D图形（如立方体、球体、圆锥体等）。
- **用户交互**：提供直观的用户界面，支持用户通过鼠标和键盘与3D模型进行交互，如旋转、缩放和平移模型。
- **OBJ文件处理**：支持读取和保存OBJ格式的文件，允许用户导入和导出3D模型。
- **性能优化**：确保3D渲染流畅，对计算机资源的占用合理。
- **扩展性**：程序结构应具备一定的灵活性，便于未来添加新功能或进行改进。

1.3 术语说明

- **OpenGL（Open Graphics Library）**：一个用于渲染2D和3D矢量图形的跨语言、跨平台的应用程序接口（API）。
- **3D建模**：使用计算机图形学创建三维物体的过程。
- **OBJ文件**：一种标准的3D图像文件格式，广泛用于存储3D模型信息，包括模型的几何形状和表面纹理。
- **用户界面（UI）**：程序的操作界面，通过它用户可以与程序交互。
- **交互性**：指用户能够通过输入设备（如鼠标、键盘）与计算机程序进行交互的能力。
- **渲染**：在计算机图形学中，指的是通过计算机程序将模型转换为图像的过程。

2 需求分析

2.1 业务需求

1. **教育和自学**：程序提供了一个平台，供学生和爱好者学习和实践3D建模的基本技能。
2. **设计和原型制作**：为设计师和开发人员提供一个工具，用于快速创建和修改3D模型原型。
3. **文件格式兼容性**：通过支持OBJ文件格式的读取和保存，程序能够与其他主流3D建模工具兼容，便于在不同软件之间交换和编辑模型。
4. **用户友好性**：考虑到不同技能水平的用户，程序设计应注重直观的用户界面和简单的操作流程。
5. **性能优化**：确保即使在处理复杂模型时，程序也能保持良好的性能和响应速度。

2.2 功能需求

1. **基础3D建模功能**：能够创建和编辑基本的3D图形，如立方体、球体和圆锥体。
2. **模型编辑工具**：包括平移、旋转、缩放等基本的3D模型操作工具，以及颜色和纹理的调整功能。
3. **OBJ文件支持**：实现OBJ文件的导入和导出功能，使用户能够轻松地在不同的建模软件之间迁移和共享数据。
4. **交互式视图控制**：提供用户友好的视图操作，如通过鼠标拖动和滚轮缩放来调整视角。
5. **帮助文档和教程**：提供详细的帮助文档和教程，帮助用户快速学习如何使用程序。
6. **性能优化**：程序应针对不同的硬件配置进行优化，确保在处理大型或复杂的3D模型时依然保持良好的性能。

3 新设计类功能说明

3.1 总体框架设计

Microsoft Foundation Classes (MFC) 是一个用于构建Windows应用程序的C++库，它提供了一系列的类，用于简化Windows API的使用。MFC遵循面向对象的设计原则，使得开发Windows应用程序变得更加直观和高效。以下是MFC的整体框架设置的详细描述：

3.1.1 应用程序框架

MFC应用程序通常遵循一个标准的框架结构，包括以下主要组件：

1. **CWinApp类**：
   - 这是MFC应用程序的核心，通常由开发者继承并实现。
   - 它负责管理应用程序的启动和关闭，以及处理主消息循环。
2. **CFrameWnd类**：
   - 代表应用程序的主窗口框架。

- 它通常包含菜单栏、工具栏、状态栏以及一个或多个视图。

3. **CView类**:
   - 用于实现应用程序的用户界面和逻辑。
   - 每个视图都关联一个文档（CDocument），用于处理数据。
4. **CDocument类**:
   - 代表应用程序的数据模型。
   - 它负责数据的管理、序列化和反序列化。

3.1.2 文档/视图架构

MFC采用文档/视图架构（Document/View Architecture），将数据（文档）和用户界面（视图）分离，以支持数据的独立处理和多视图显示。

1. **文档（CDocument）**：
   - 存储和管理应用程序的数据。
   - 可以有多个视图关联到同一个文档。
2. **视图（CView）**：
   - 提供数据的展示和用户交互。
   - 可以有多种视图类型展示同一文档的数据。

3.1.3 消息映射

MFC使用消息映射机制来处理Windows消息。这是通过在类中声明消息映射宏和相应的消息处理函数来实现的。

1. **BEGIN_MESSAGE_MAP 和 END_MESSAGE_MAP**:
   - 在类定义中使用这些宏来声明消息映射。
2. **消息处理函数**:
   - 为不同的消息（如鼠标点击、键盘输入）实现具体的处理函数。

3.1.4 资源管理

MFC提供了资源编辑器，用于管理应用程序的资源，如菜单、对话框、图标等。

1. **资源文件（.rc）**：
   - 存储应用程序的资源定义。
2. **资源类（如CMenu, CDialog）**：
   - 用于在代码中操作这些资源。

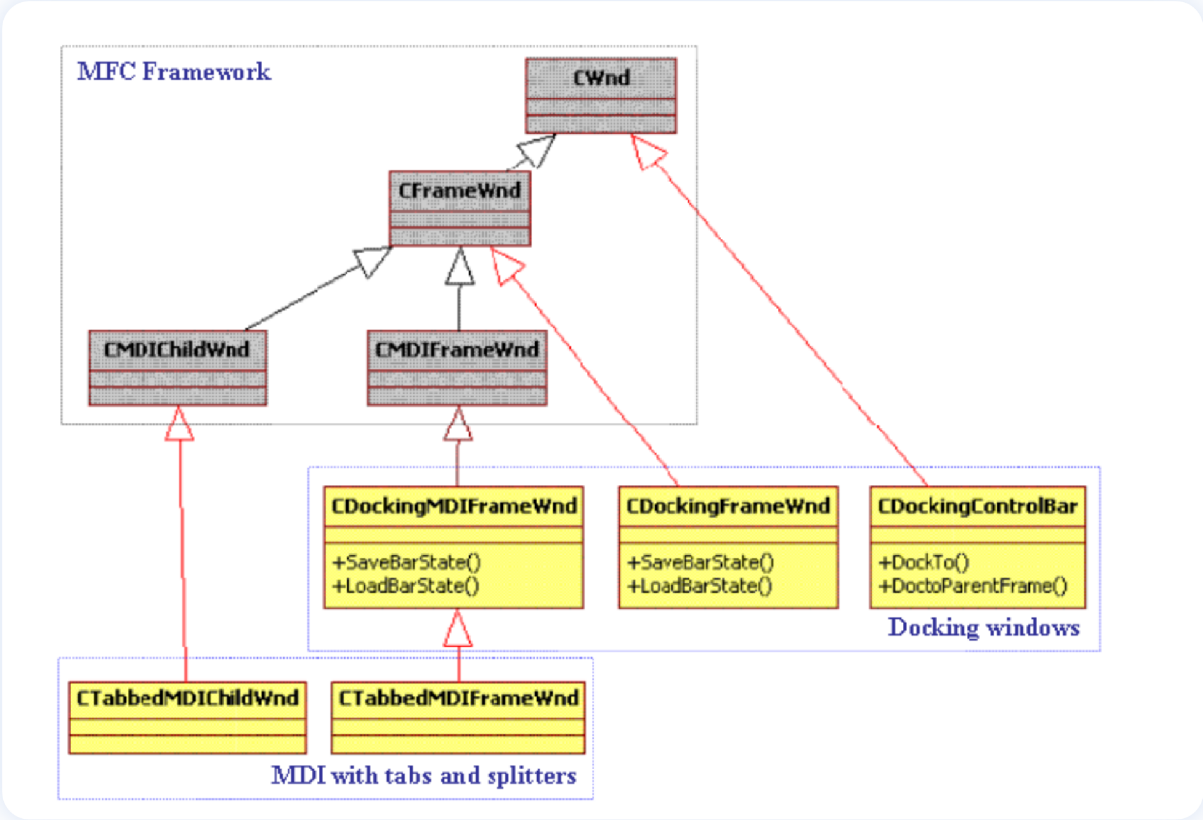
3.1.5 类向导和工具

MFC提供了类向导和其他工具，以帮助开发者快速生成和管理代码。

1. **类向导**：
   - 用于创建新的类和添加消息处理函数。
2. **AppWizard**：
   - 用于创建新的MFC应用程序，提供了多种应用程序模板。

3.1.6 异常处理

MFC支持结构化的异常处理，通过C++的异常处理机制（try, catch, throw）来管理错误和异常情况。



3.2 类模块体系设计

在我们的3D建模软件中，有两个核心类，分别是 `EnableView` 类和 `CMyOpenGLView` 类，它们共同构成了软件的基础框架，负责处理用户界面和功能操作。

1. **EnableView 类** - 这个类的主要职责是控制模型的显示效果。它就像是一个灵活的舞台导演，精心安排着每一个场景的展示。无论是调整窗口的大小，还是设置观看模型的视角，`EnableView` 类都能够应对自如。它能够调整模型的大小，确保模型在屏幕上的显示既清晰又精确。此外，这个类还负责模型的颜色渲染，让模型呈现出丰富多彩的外观。更为重要的是，它还掌

握着关于模型旋转的各种参数，使用户能够通过鼠标操作来旋转和查看模型的不同角度，就像在现实中观察一个实体物体一样自然和直观。

2. **CMyOpenGLView 类** - 这个类则是功能转换的大师。它通过一系列的按钮和键盘输入，让用户能够轻松地在不同功能之间切换，就像是一个多功能的遥控器。无论用户想要进行哪种操作，`CMyOpenGLView` 类都能迅速响应，提供相应的功能。更加引人注目的是，这个类还承担着OBJ文件的读入和存储任务。它就像是一个聪明的图书管理员，将所有的3D模型数据整齐地存放在一个名为 `vector` 的数据结构中。这不仅使得数据的存取变得高效，而且保证了数据的安全和稳定。

3.3 源代码文件组织设计

```
.
├── Debug
│   ├── myopengl.exe
│   └── myopengl.pdb
├── Release
│   ├── cat.obj
│   ├── cow.obj
│   ├── glut.dll
│   ├── glut32.dll
│   ├── myopengl.exe
│   ├── myopengl.iobj
│   ├── myopengl.ipdb
│   ├── myopengl.pdb
│   ├── palm.obj
│   └── rectangle.obj
├── libigl库类图.png
├── myopengl
│   ├── Debug
│   │   ├── MainFrm.obj
│   │   ├── enableview.obj
│   │   ├── myopengl.exe.recipe
│   │   ├── myopengl.ilk
│   │   ├── myopengl.obj
│   │   ├── myopengl.pch
│   │   ├── myopengl.res
│   │   ├── myopengl.tlog
│   │   │   ├── CL.command.1.tlog
│   │   │   ├── CL.read.1.tlog
│   │   │   ├── CL.write.1.tlog
│   │   │   ├── Cl.items.tlog
│   │   │   ├── link.command.1.tlog
│   │   │   ├── link.read.1.tlog
│   │   │   ├── link.write.1.tlog
│   │   │   ├── myopengl.lastbuildstate
```

```
│   │   │   ├── rc.command.1.tlog
│   │   │   ├── rc.read.1.tlog
│   │   │   └── rc.write.1.tlog
│   │   ├── myopenglDoc.obj
│   │   ├── myopenglView.obj
│   │   ├── stdafx.obj
│   │   ├── vc143.idb
│   │   └── vc143.pdb
│   ├── GLAUX.H
│   ├── GLAUX.LIB
│   ├── MainFrm.cpp
│   ├── MainFrm.h
│   ├── Palm_01.obj
│   ├── Release
│   │   ├── MainFrm.obj
│   │   ├── enableview.obj
│   │   ├── myopengl.exe.recipe
│   │   ├── myopengl.iobj
│   │   ├── myopengl.ipdb
│   │   ├── myopengl.log
│   │   ├── myopengl.obj
│   │   ├── myopengl.pch
│   │   ├── myopengl.res
│   │   ├── myopengl.tlog
│   │   │   ├── CL.command.1.tlog
│   │   │   ├── CL.read.1.tlog
│   │   │   ├── CL.write.1.tlog
│   │   │   ├── Cl.items.tlog
│   │   │   ├── link.command.1.tlog
│   │   │   ├── link.read.1.tlog
│   │   │   ├── link.write.1.tlog
│   │   │   ├── link.write.2u.tlog
│   │   │   ├── myopengl.lastbuildstate
│   │   │   ├── myopengl.write.1u.tlog
│   │   │   ├── rc.command.1.tlog
│   │   │   ├── rc.read.1.tlog
│   │   │   └── rc.write.1.tlog
│   │   ├── myopenglDoc.obj
│   │   ├── myopenglView.obj
│   │   ├── stdafx.obj
│   │   ├── vc140.pdb
│   │   ├── vc141.pdb
│   │   ├── vc142.pdb
│   │   └── vc143.pdb
│   ├── cat.obj
│   ├── cow_color.obj
```

```
    ├── enableview.cpp
    ├── enableview.h
    ├── glut.h
    ├── glut32.lib
    ├── myopengl.aps
    ├── myopengl.cpp
    ├── myopengl.h
    ├── myopengl.rc
    ├── myopengl.vcxproj
    ├── myopengl.vcxproj.filters
    ├── myopengl.vcxproj.user
    ├── myopenglDoc.cpp
    ├── myopenglDoc.h
    ├── myopenglView.cpp
    ├── myopenglView.h
    ├── rect.obj
    ├── res
    │   ├── Toolbar.bmp
    │   ├── myopengl.ico
    │   ├── myopengl.rc2
    │   └── myopenglDoc.ico
    ├── resource.h
    ├── stdafx.cpp
    ├── stdafx.h
    └── targetver.h
├── myopengl.sln
├── 功能完成评价表.docx
└── 面向对象程序设计大程报告.pdf
```

3.4 重点类及函数设计描述

`CmyopenglView` 类是基于 MFC (Microsoft Foundation Classes) 框架和 OpenGL 的一个3D建模视图类。这个类继承自 `enableview` 类，并实现了一系列的方法来处理3D图形的绘制、交互和文件操作。以下是各个类和主要函数的作用和功能描述：

3.4.1 类和全局变量

1. **CmyopenglView**:
   - 这是主要的视图类，负责3D图形的渲染和用户交互。
2. **全局变量**:
   - `model` 和 `type`：用于控制绘制的3D模型类型和渲染方式（如线框、填充）。
3. **光照相关的全局变量**:
   - 如 `mat_ambient`，`mat_diffuse` 等，用于设置OpenGL中的材质和光照属性。

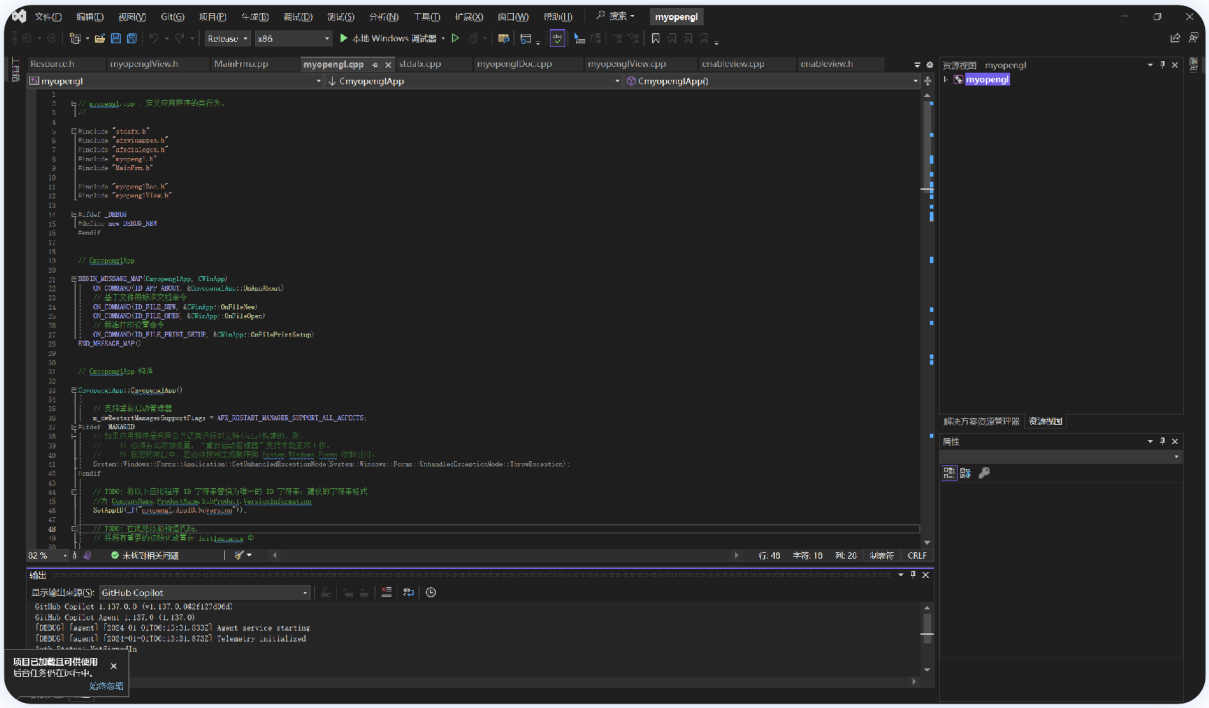
3.4.2 主要函数

1. **构造函数和析构函数（`CmyopenglView::CmyopenglView`，`CmyopenglView::~CmyopenglView`）**：
   - 初始化和清理视图类的实例。
2. `PreCreateWindow`：
   - 在窗口创建之前调用，用于修改窗口的样式或类名。
3. `OnDrawGL`：
   - 核心渲染函数，使用OpenGL命令绘制3D图形。
4. `ordination`：
   - 辅助函数，用于设置和调整OpenGL的一些渲染状态。
5. `InitalLigt`：
   - 初始化OpenGL的光照设置。
6. `PreTranslateMessage`：
   - 捕获并处理键盘事件，用于实现模型的平移和旋转等交互功能。
7. `ReadObj`：
   - 从OBJ文件中读取3D模型数据。
8. `OnReadobj`：
   - 处理读取OBJ文件的命令，调用 `ReadObj` 函数。
9. `Draw_obj`：
   - 根据读取的OBJ文件数据绘制3D模型。
10. **`OnKeyDown` 和其他 `On` 开头的函数**：
    - 这些函数处理各种用户输入事件，如键盘按键和菜单命令。
11. `OnCreate`：
    - 在视图创建时调用，用于执行初始化操作。

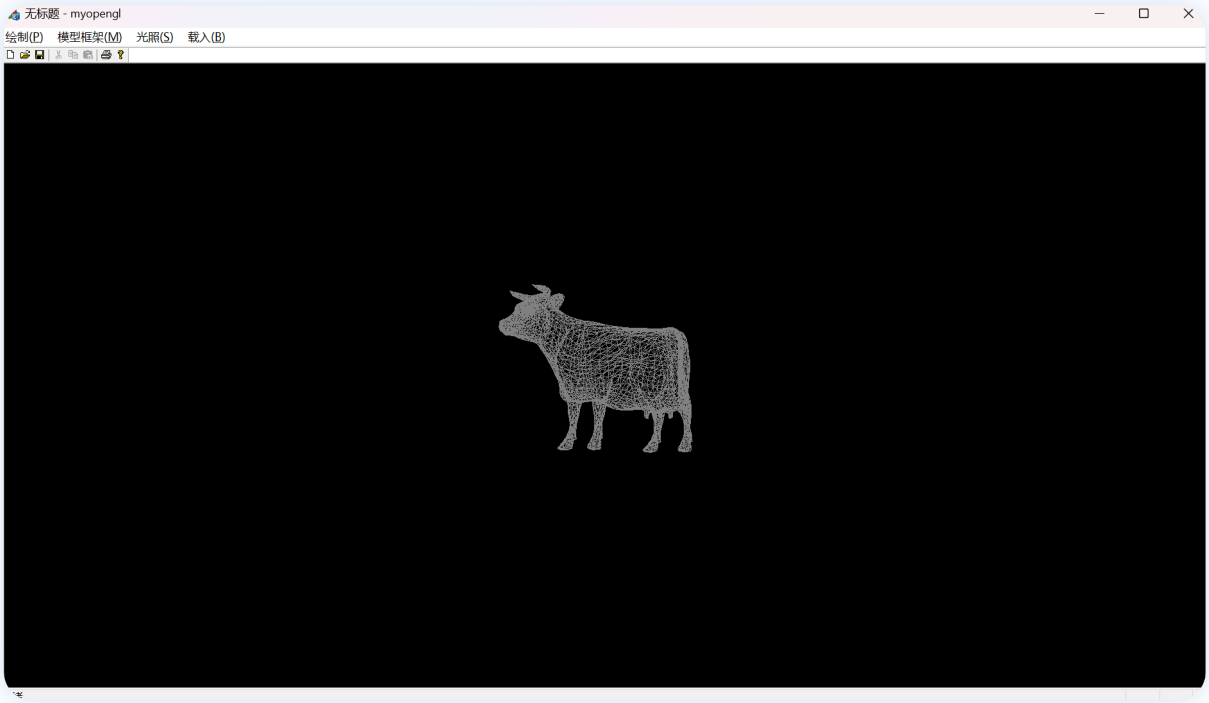
4 部署运行和使用说明

4.1 编译安装

用VS2020打开 `myopengl.sln`，请注意一定要使用 `Debug X86`



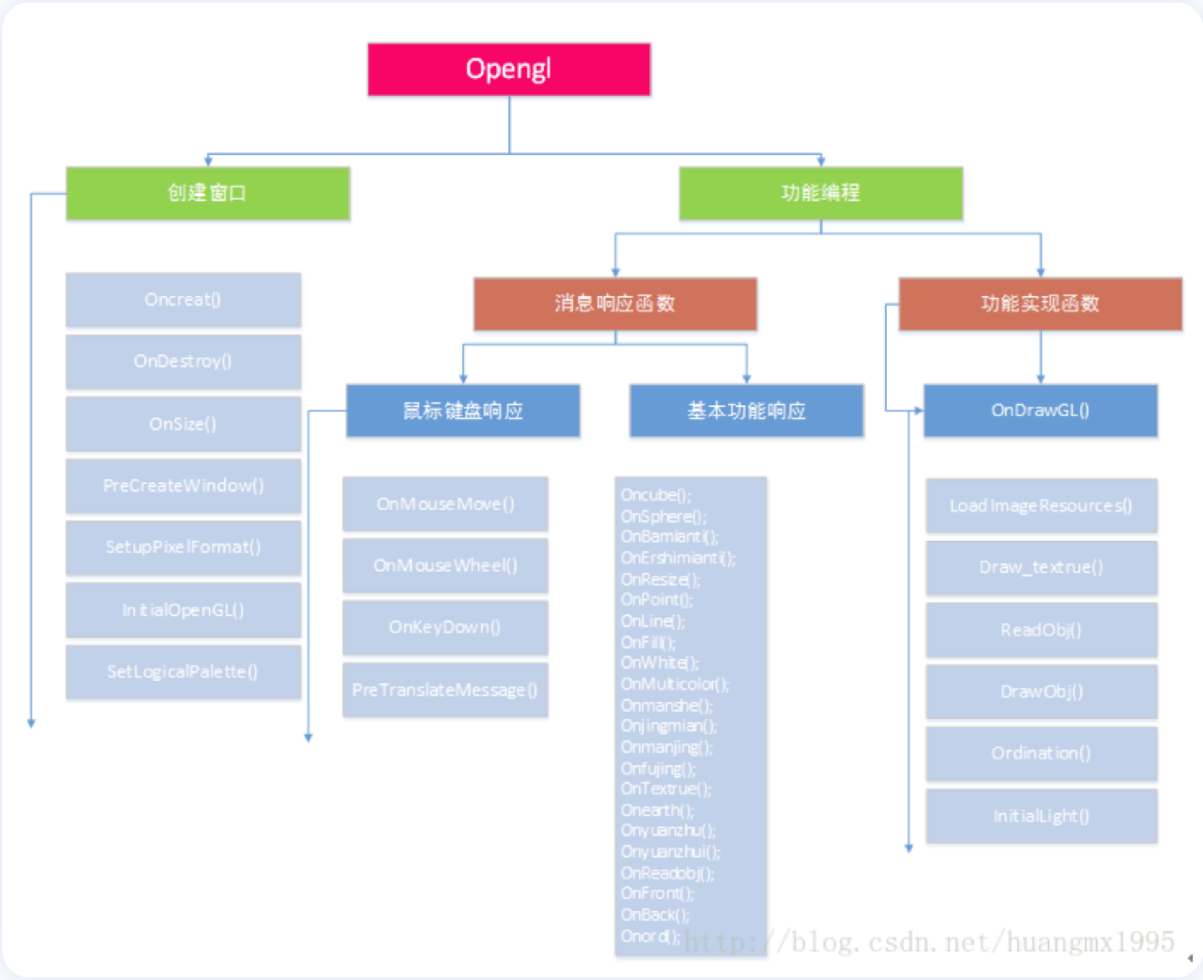
编译运行即可：



4.2 运行测试

4.2.1 类图

在project/libigl库类图.png（由于图片太大这里放不下）



4.2.2 算法实现

4.2.2.1 读取obj文件：

代码如下：

```
void CmyopenglView::OnReadobj(){
    // 设置模型标识为6
    model = 6;

    // 设置文件对话框的过滤器，只显示.obj和所有文件类型
    wchar_t filters[] = L"3D模型文件(*.obj)|*.obj|所有文件(*.*)|*.*||";
    CFileDialog fileDlg(TRUE, NULL, NULL, OFN_HIDEREADONLY, filters);

    // 打开文件对话框，让用户选择文件
    if (fileDlg.DoModal() == IDOK){
```

```
        // 获取用户选择的文件路径
        CString strBuf = fileDlg.GetPathName();
        USES_CONVERSION;
        // 将文件路径从宽字符转换为多字节字符
        char *Filename = T2A(strBuf.GetBuffer(0));
        // 读取.obj文件
        ReadObj(Filename);
    }

    // 使用字符串流来构建消息
    stringstream ss;
    ss << "加载完成! ";
    string str;
    ss >> str;
    CString s;
    s = str.c_str();
    // 显示消息框，告知用户文件加载完成
    MessageBox(s);

    // 初始化最小和最大坐标值
    float min_x, min_y, min_z, max_x, max_y, max_z;
    min_x = min_y = min_z = 10000000;
    max_x = max_y = max_z = -1000000;

    // 遍历所有顶点，找到边界坐标
    for (int i = 0; i < V.size(); i++){
        min_x = min(min_x, V[i].x);
        min_y = min(min_y, V[i].y);
        min_z = min(min_z, V[i].z);
        max_x = max(max_x, V[i].x);
        max_y = max(max_y, V[i].y);
        max_z = max(max_z, V[i].z);
    }

    // 计算模型的世界坐标中心
    worldx = (min_x + max_x) / 2;
    worldy = (min_y + max_y) / 2;
    worldz = (min_z + max_z) / 2;

    // 设置渲染类型为1
    type = 1;

    // 使视图无效并重绘
    Invalidate();

    // 获取设备上下文并进行OpenGL绘制
```

```
    CDC* ppDC = GetWindowDC();
    OnDrawGL(ppDC);

    // TODO: 在此添加命令处理程序代码
}
```

主要步骤如下：

1. 打开文件对话框，让用户选择.obj文件。
2. 读取并处理.obj文件。
3. 计算模型的边界和中心坐标。
4. 使用OpenGL进行渲染。

```
void CmyopenglView::ReadObj(char* Filename){
    // 清空之前的数据
    VN.clear();
    V.clear();
    VT.clear();
    F.clear();
    FQ.clear();

    // 打开文件
    ifstream in(Filename);
    string aline;
    string erase;

    // 逐行读取文件
    while (getline(in, aline)){
        // 如果行以 'v' 开头，表示顶点数据
        if (aline[0] == 'v'){
            // 如果接下来是 'n'，表示法向量
            if(aline[1] == 'n'){
                istringstream sin(aline);
                Vertex v;
                sin >> erase >> v.x >> v.y >> v.z;
                VN.push_back(v);
            }
            // 如果接下来是 't'，表示纹理坐标
            else if(aline[1] == 't'){
                istringstream sin(aline);
                Texture v;
                sin >> erase >> v.s >> v.t;
```

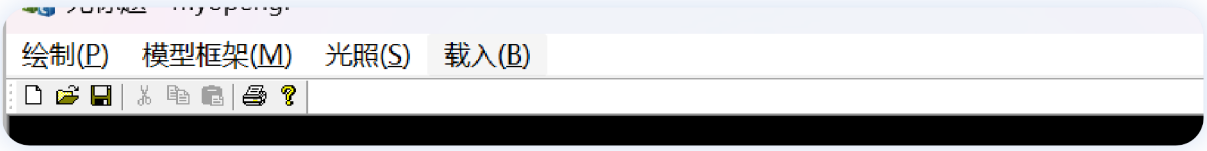
```
                VT.push_back(v);
            }
            // 否则，是顶点坐标
            else{
                istringstream sin(aline);
                Vertex v;
                sin >> erase >> v.x >> v.y >> v.z;
                V.push_back(v);
            }
        }
        // 如果行以 'f' 开头，表示面数据
        else if(aline[0] == 'f'){
            istringstream sin(aline);
            sin >> erase;
            vector<string> strvector;
            string temp;
            while(sin >> temp) strvector.push_back(temp);

            // 如果是三角形面
            if (strvector.size() == 3){
                Face fff;
                for (int count = 0; count < 3; count++) {
                    string kkk = strvector[count];
                    int i = 0;
                    int num = 0;
                    // 解析顶点索引
                    for (; i < kkk.size() && kkk[i] != '/'; i++)
                        num = num * 10 + kkk[i] - '0';
                    fff.v[count] = num;
                    i++;
                    num = 0;
                    // 解析纹理坐标索引（如果有）
                    for (; i < kkk.size() && kkk[i] != '/'; i++)
                        num = num * 10 + kkk[i] - '0';
                    fff.vt[0] = num;
                    i++;
                    num = 0;
                    // 解析法向量索引（如果有）
                    for (; i < kkk.size() && kkk[i] != '/'; i++)
                        num = num * 10 + kkk[i] - '0';
                    fff.vn[count] = num;
                }
                F.push_back(fff);
            }
            // 如果是四边形面
            else if (strvector.size() == 4){
```

```
                FaceQ fff;
                for (int count = 0; count < strvector.size(); count++) {
                    string kkk = strvector[count];
                    int i = 0;
                    int num = 0;
                    // 解析顶点索引
                    for (; i < kkk.size() && kkk[i] != '/'; i++)
                        num = num * 10 + kkk[i] - '0';
                    fff.v[count] = num;
                    i++;
                    num = 0;
                    // 解析纹理坐标索引（如果有）
                    for (; i < kkk.size() && kkk[i] != '/'; i++)
                        num = num * 10 + kkk[i] - '0';
                    fff.vt[0] = num;
                    i++;
                    num = 0;
                    // 解析法向量索引（如果有）
                    for (; i < kkk.size() && kkk[i] != '/'; i++)
                        num = num * 10 + kkk[i] - '0';
                    fff.vn[count] = num;
                }
                FQ.push_back(fff);
            }
        }
    }
}
```

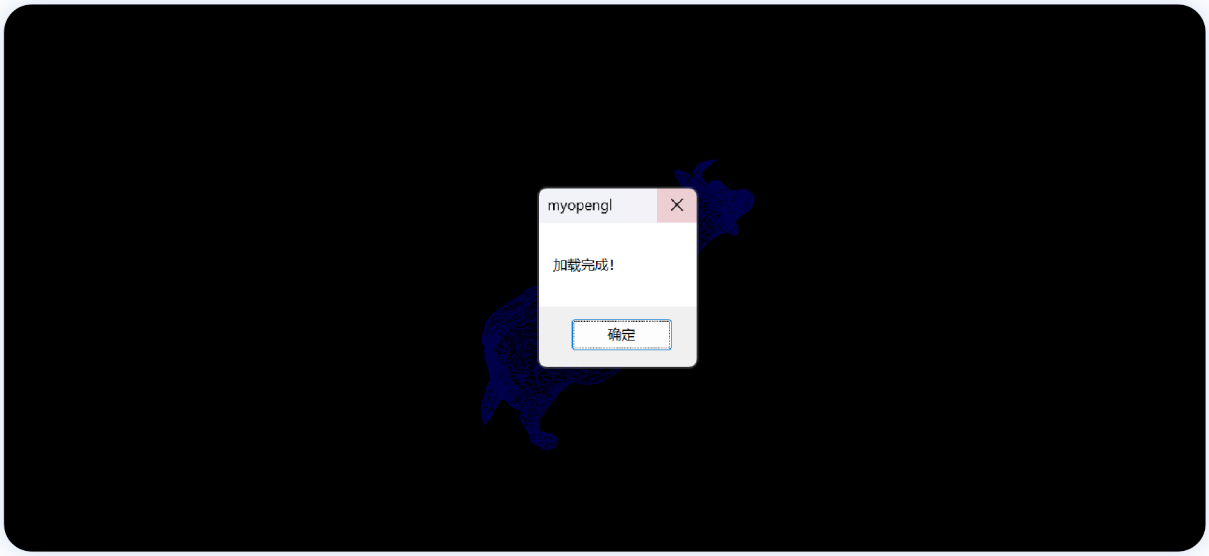
1. 清空之前存储的顶点、纹理坐标、法向量和面的数据。
2. 打开指定的 `.obj` 文件并逐行读取。
3. 根据行的首字符判断数据类型（顶点、纹理坐标、法向量或面）。
4. 解析每行数据，并将解析后的数据存储在相应的数据结构中。

效果如下：

从菜单栏中选中 `载入`，选择 `载入模型`，从文件夹中选择 `obj` 文件，点击载入。



载入成功：



4.2.2.2 多面光渲染

伪代码如下：

```
函数 InitalLigt():

    定义六个光源的位置和方向
    light_position1 = [-52, -16, -50, 0]
    light_position2 = [-26, -48, -50, 0]
    light_position3 = [16, -52, -50, 0]
    direction1 = [52, 16, 50]
```

```
direction2 = [26, 48, 50]
direction3 = [-16, 52, 50]
light_position4 = [52, 16, 50, 0]
light_position5 = [26, 48, 50, 0]
light_position6 = [-16, 52, 50, 0]
direction4 = [-52, -16, -50]
direction5 = [-26, -48, -50]
direction6 = [16, -52, -50]

设置背景颜色和启用深度测试
设置背景颜色为白色
启用深度测试

如果 color_type 等于 0 (彩色灯光):
    设置六个光源的颜色
    color1 = [1, 0, 0, 1]
    color2 = [0.5, 1, 0, 1]
    color3 = [0, 0, 1, 1]
    color4 = [1, 0, 0, 1]
    color5 = [0.5, 1, 0, 1]
    color6 = [0, 0, 1, 1]

    设置环境光和材质属性
    ambient = [0.3, 0.3, 0.3, 1.0]
    material_color = [1, 1, 1, 0.5]
    material_specular = [0.5, 0.5, 0.5, 0.5]
    material_ambient = [0.0, 0.0, 0.0, 0.0]

    配置和启用光源3、4、5
    设置光源3的位置、方向、漫反射和镜面反射颜色
    设置光源4的位置、方向、漫反射和镜面反射颜色
    设置光源5的位置、方向、漫反射和镜面反射颜色

    设置环境光模型和材质属性
    设置材质的镜面反射、漫反射、环境光属性和光泽度

    启用光照并禁用光源0
    启用光源3、4、5
    禁用颜色材质

如果 color_type 等于 1 (白色灯光):
    禁用光源3、4、5和光照

    设置光源0的位置和属性
    light_position0 = [0.0, 10.0, 10.0, 1.0]
    ambientLight = [0.25, 0.25, 0.25, 1.0]
```

```
        diffuseLight = [0.5, 0.5, 0.5, 1.0]
        specularLight = [0.5, 0.5, 0.5, 1.0]

        启用和配置光照和光源0
        启用光照
        设置环境光模型
        设置光源0的环境光、漫反射、镜面反射属性和位置
        启用光源0

        启用颜色材质并设置其属性
```

效果如下：

4.3 使用操作

1. 按住鼠标左键，来回移动可实现模型的旋转。
2. 滑动鼠标滚轮，可实现模型的缩放。
3. `W` `A` `S` `D` 四键可实现模型的上下左右移动。
4. `Q` 键可实现模型快速放大，`R` 键可实现模型快速缩小

4.4 收获感言

经过一个月的努力和学习，我终于完成了基于OpenGL的3D建模程序，这个过程对我来说既是挑战也是一次宝贵的学习经历。在这个项目中，我不仅深入理解了计算机图形学的核心概念，还学会了如何将这些理论应用于实际的软件开发中。

在开发过程中，我遇到了许多挑战，包括复杂的数学问题、性能优化以及调试复杂的渲染错误。每一个挑战都让我更加深入地理解了OpenGL的工作原理以及3D建模的细节。通过不断的实践和学习，我逐渐掌握了顶点缓冲、着色器编程、纹理映射等关键技术。

此外，这个项目也锻炼了我的问题解决能力和编程技巧。在遇到问题时，我学会了如何有效地利用资源，包括在线文档、社区论坛和同行的建议。这些资源在我解决问题和学习新技术时提供了巨大的帮助。

最让我感到自豪的是，通过这个项目，我不仅提升了自己的技术能力，还增强了自信心。看到自己从头开始构建的3D建模程序运行起来，感觉非常满足。这个项目让我意识到，只要有决心和努力，就没有什么是不可能的。

最后，我要感谢所有在这个过程中给予我支持和帮助的人。没有他们的鼓励和指导，我无法完成这个项目。我期待着将我在这个项目中学到的知识应用到未来的工作中，并继续在计算机图形学领域探索和成长。