第1章 项目背景

1.1 项目介绍

三维模型的表示法有很多，如点云表示法，参数曲面表示法，体素表示法，隐式曲面表示法，多边形网格表示法等，其中多边形网格表示法利用众多的微小多边形面逼近真实几何体的实际表面,其中又以最常用的三角形网格表现能力丰富，由于多边形都可分为多个三角形，因此三角网格表示法式多边形网格表示法德基础，许多图形硬件的绘制都以此为基础。从 CAD，电影 CG 的制作到动画，再到游戏以及一些办公用的模拟软件，都要用到各式各样的几何模型，利用三维扫描仪器可以获得物体表面的三维点坐标，由计算机进行三维重构并方便的转化为三角网格模型。网格形变是网格模型研究的一个重要应用方向，而三角网格模型的变形编辑在几何造型，电影 CG，动画，游戏以及模拟等领域都有广泛的应用。依赖于信息获取技术的进步，利用三维扫描仪器得到物体表面的三维坐标，再用计算机进行三维重构来得到，重建出复杂的几何模型的逆向几何工程已经越来越简便。这些几何模型不仅真实反映了具体对象的实际形状，而其为设计师提供了进一步编辑，造型的基础。因此，处理和重用原有的几何模型，已成为解决几何设计效率问题的重用途径，而三维几何数据重用的核心是模型的编辑与形变技术。

鉴于以往的网格形变算法往往需要过多的用户输入和交互，并且难以保持原三维模型的表面几何细节，在这篇毕业设计中将采用拉普拉斯算法。几何细节作为三维模型的内在属性，可以用拉普拉斯坐标来表示，拉普拉斯坐标也称为微分坐标或平均曲率法向，首先在[4]中被提出，是最近出现的进行网格模型形变的工具。几何细节被认为是网格表面的内在性质,在拉普拉斯坐标系统中，相比网格在空间的绝对坐标，我们更关心一个点与周围的点存在什么样的联系，这种联系决定了在形变操作后的网格模型的几何细节保持程度。拉普拉斯坐标是我们实现这个平台中形变功能用到的核心数学工具之一。相比于自由形变算法与蒙皮算法，用户不再被限制在长方体框架及“骨骼”的控制点上，而且拉普拉斯形变算法能更好的保持几何细节。但拉普拉斯坐标在形变中只有平移不变性，在旋转和拉伸操作后往往不能够保持不变，这篇毕业设计将以三角网格

模型作为构建和分析的对象，实现算法研究的辅助平台，主要实现对拉普拉斯坐标在旋转和拉伸操作后的局部优化算法[3]，并在已有基础上进行改进和测试。

1.2 同行研究情况

在 3D max 或者 maya 中，人们可以用自由形变算法或者蒙皮算法[1]进行模型的形变，在自由形变中，先构造一个长方体的框架，将模型放入长方体中，通过对长方体上的控制顶点实施操作使模型的形状也随之改变，这种方法的缺点是，由于物体的形变是由框架控制顶点产生的，因此精确移动物体上的某一点并想达到用户预期的效果很困难，并且这种方法对模型表面细节的保持也不尽人意。蒙皮算法[1]是具有骨骼语义的几何模型操作，用户指定一块“骨骼”，该算法将所有点绑定到这个“骨骼”的一系列控制点上，每个点有不同的权值，与离控制点的距离有关，用户操作这些控制点或者指定变形参数。控制点的形变将扩散到所有绑定的点即整个几何模型上，这一算法在动画，设计师设计草图如等场合得到广泛应用。蒙皮算法的主要缺点是不适合表达骨骼语义不明显的形变，如几何模型的一个球状突起或者气球膨胀或缩小等。而多分辨率形变算法[5]不仅需要多次简化与重构的过程，而且在几何细节保持达不到满意效果，泊松方程算法[6]只适合表达这样一种语义，即在一个固定坐标只有一个高度值的曲面最好，其他语义所产生的形变后原来模型的表面细节保持很难让人满意。近几年出现的拉普拉斯形变算法[8]很大程度上解决了这个问题，然而普通的拉普拉斯算法[8]虽然能够在坐标平移时保持不变，但在拉伸或旋转时却没有不变性。

1.3 本人工作

拉普拉斯坐标是最近出现的可以用于网格形变的新的数学工具，目前一些优秀的形变算法如 [2]都用到拉普拉斯坐标, 于是我们以拉普拉斯坐标为基础，实现对三角网格模型形变后表面细节的高质量保持的优化拉普拉斯算法，以往的拉普拉斯形变方法虽然可以在一定程度上保持三角网格模型的表面细节，但由于拉普拉斯坐标的不变性是针对坐标平移，在坐标发生拉伸和旋转后往往不能保持不变，因此基于拉普拉斯坐标在形变前与形变后保持不变的算法在拉伸与旋转后在保持原有模型的表面细节仍有欠缺，

在这篇毕业设计中，我们将主要实现优化拉普拉斯形变算法[3]的造型平台，即通过构造局部优化框架将拉普拉斯坐标进行拉伸和旋转，使形变前的拉普拉斯坐标与形变后的拉普拉斯坐标在方向和大小上尽量保持不变，并购造过渡区使得形变区域的过渡更加自然。三角网格模型拉普拉斯形变平台集成了对三角网格模型的常用操作和常用算法，可以对读入的三角网格模型进行一系列的操作并保存数据。

第2章 项目实施方案

2.1 实施方案

三角网格模型拉普拉斯形变平台,针对目前流行的3D模型造型编辑平台3D max和maya等在处理三角网格模型形变时所采用的自由形变算法和蒙皮算法在保持原来模型表面细节方面的不足，以及普通拉普拉斯形变虽然相比自由形变算法和蒙皮算法在保持原来模型表面细节方面有很大进步，但在坐标旋转和拉伸时，不能像平移那样保持不变而导致表面细节的不足而设计，因此，我们的主要目的是实现优化拉普拉斯形变算法[3],并对优化拉普拉斯算法的求解过程以及公式进行改进,将过渡区加入该算法中。首先搭建 OpenGL 平台，该平台为用户提供了实现各种功能（如平移，旋转，网格细分，网格简化）的菜单及按钮。可以读入三角网格模型数据文件obj,搭建好平台后，将首先实现三角网格细分，网格简化功能，网格细分的目的是，如果读入的三角网格模型存在局部过稀疏的情况，网格细分操作将稀疏的三角网格细分为足够数量的小三角网格。网格简化可以看成网格细分的逆操作，即读入的三角网格模型存在过密的情况，网格简化删除不重要的边，面，使得局部三角网格数量趋于平均。细分和简化过的三角网格模型可能存在不光滑情况，网格平滑操作使得三角网格模型能够变得更为平滑。当用户想进行拉普拉斯形变操作时，首先用鼠标左键单击几何模型，被选择的处于近用户侧的三角形网格在以网格显示的模式下将处于高亮状态，用户通过选择一环封闭的三角形网格来确定形变区域，即选中三角形网格内侧的区域为形变区域，而这一环封闭的三角形为控制点，用户还要选择形变的点，通过鼠标左键单击一环封闭三角形网格内的形变区域来获得这些点，然后可以控制这些点进行平移，拉伸，旋转。当用户输入移动形变点指令后，程序将计算形变点移动后的坐标，并计算三角形网格其他点的最优坐标，及使得最小二乘误差最小。最后用户可以选择保存形变后的网格模型。

2.2 软件架构与环境

三角网格模型拉普拉斯形变平台由c++语言编写，使用OpenGL图形库，并用开源的QT编写GUI,编写好的程序由mingw(gnu for windows)编译，程序的开

发过程中所使用的都为开源或免费库，QT 的可移植性，使得程序具有良好的移植性，可在 windows,linux,unix,mac,symbian 等平台编译后运行，三角网格模型拉普拉斯形变平台的基本软件架构如下:

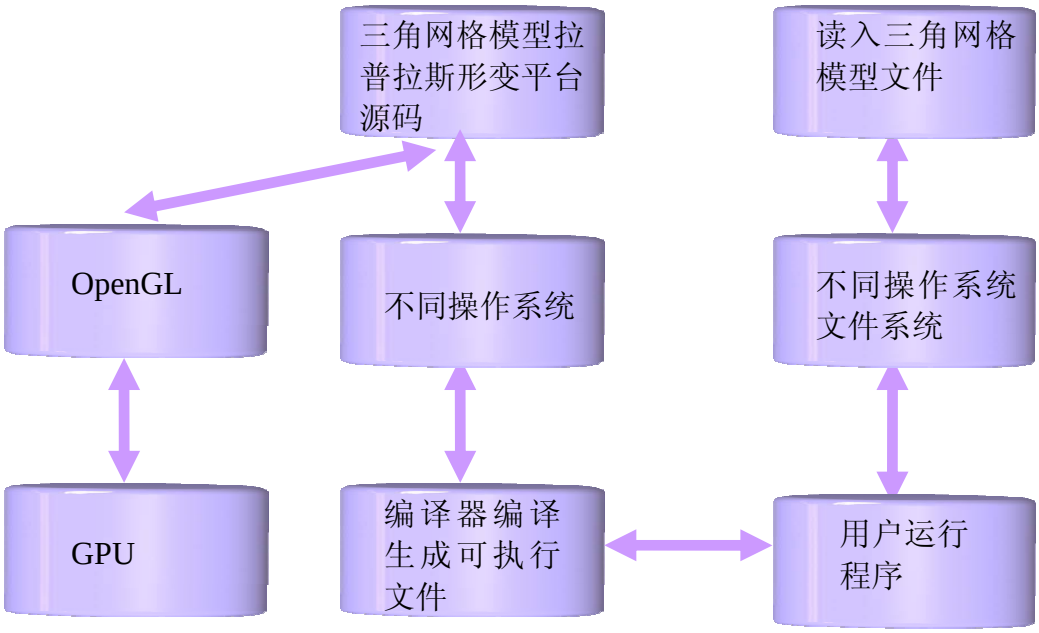


图 2-1 三角网格模型拉普拉斯形变平台架构图

2.3 功能描述

三角网格模型拉普拉斯形变平台将包括以下功能：三角网格模型的读入与显示，以点云，网格，普通方式显示模型，提供光照，用户可以对模型平移，旋转，缩放，用户可以选择视角缩放，可以选择模型上的三角面，可以对模型简化，细分，平滑，实施普通拉普拉斯形变，优化拉普拉斯形变，并可保存操作后的数据。

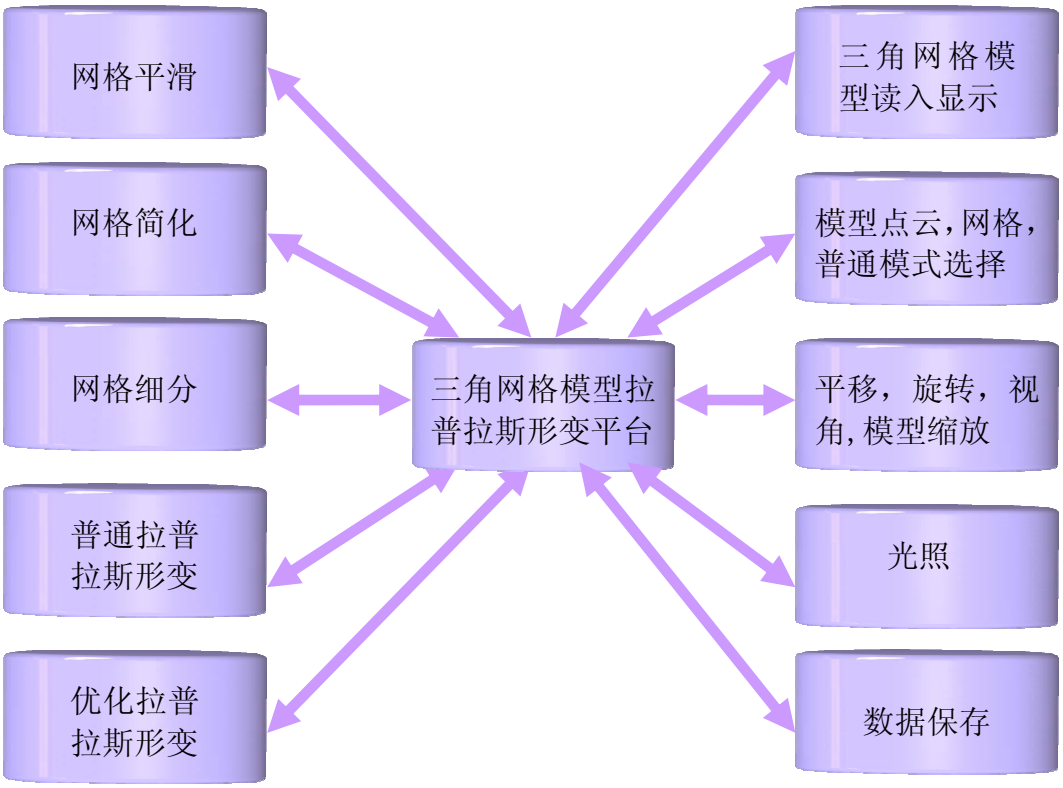


图 2-2 三角网格模型拉普拉斯形变平台功能图

第3章 在项目中负责的具体工作

3.1 负责的具体工作

这篇毕业设计与三角网格模型作为构建与分析的对象，搭建三维网格模型编辑平台，能够读取三角网格模型，并对读入的模型提供对网格模型的视图(点云，三角网格，正常模型)，平移，旋转，视角缩放，模型缩放，选取三角面等基本操作，并集成了网格简化，网格细分，网格平滑，普通拉普拉斯形变算法以及优化拉普拉斯形变算法的本人改进算法，可以对读入的三角网格模型进行以上操作并保存数据供以后重用。

这篇毕业设计的软件架构的设计，实验平台的搭建，数据设计，用例分析，原型图绘制，类的设计以及全部功能和程序中各种算法的实现和 7000 行左右的代码的编写与测试，以及相关的文档全部由本人一人完成。

3.2 数据设计

3.2.1 主要数据表

表 3-1 主要数据结构表

序号	数据名称	数据类型	数据描述
1	vertex	vector<class points>	存储三角网格模型中的点
2	face	vector<class faces>	存储三角网格模型中的面
3	edge	vector<class edges>	存储三角网格模型中的边
4	mp	map<pair<int, int>, int>	边的端点与边的索引号的对应集
5	normal	vector<class normals>	存储三角网格模型中的法线

6	v	vector<class points>	存储由用户选定的区域内的点的集合
7	pointSet	vector<int>	存储由用户选定的区域内的点的集合，点的索引号与 vertex 相同
8	ma	map<int,int>	点集 vertex 与点集 v 的索引号对应集
9	laplacian	vector<vector<double> >	生成由点集 v 构成的拉普拉斯矩阵

3.2.2 读入数据分析

在三角网格模型数据中，常用的网格模型文件有 obj,ply,3ds 等，在三角网格模型拉普拉斯形变平台中，我们支持读取的三角网格模型文件 obj。格式为以# 开头代表注释，以 v 开头代表点的三维坐标，以 vn 开头代表法向量，以 f 开头代表三角形，并以”//”隔开代表,”//”前面代表三角形的索引号，从 1 开始，”//”后面代表法向量的索引号，从 1 开始。例如：

```
# .obj format.
v    -0.013562    0.596860    -0.379364
v    -0.008161    0.602476    -0.378438
v    -0.014803    0.590327    -0.383195
vn   -0.853331    0.511423    -0.101349
vn   -0.690315    0.723412    -0.011760
vn   -0.967483    0.108675    -0.228397
vn   -0.000000    -0.000000   -1.000000
f    1//3         2//4         3//2
```

其中 vn 并不是必须的,程序中可以自动生成法向量。生成的法向量数量与点的数量对应，法向量的索引号与点的索引号对应。

3.2.3 数据结构分析

```
class points
{
```

```
public:
    double coordinate[3];
    double deformcoordinate[3];
    double normal[3];
    vector<int> neighbourlist;
    vector<int> neighbourface;
    vector<vector<double> > q;
};

class normals
{
public:
    double coordinate[3];
};

class edges
{
public:
    int point[2];
    double q;
    double newv[4];
    bool key;
    double length;
};

class faces
{
public:
    int point[3];
    int normal[3];
    double abcd[4];
```

bool key;

};

points 的成员中中包括三维坐标 coordinate[3],控制点形变三维坐标 deformcoordinate[3],程序自动生成的法向量 normal[3],邻接点索引号容器 neighbourlist，邻接面索引号容器 neighbourface，邻接面的耗散矩阵 q。

normals 的成员中中包括所有从 obj 文件中读入的法向量。

edges 的成员中包括端点的索引号 point[2],边的合并耗散值 q，合并的最优坐标 newv[4],边是否被删除的标识符 key,边的耗散长度（将边长调整到与耗散值相同的数量级）length。

faces 的成员中包括以逆时针方向构成面的三个点的索引号 point[3],与构成面的三个点相对应的法向量 normal[3],面的平面方程 $ax+by+cz+d=0$ 的系数 a,b,c,d,abcd[4],其中 $a^2+b^2+c^2=1$,面是否被删除的标识 key。

从上面的存储结构可以看出，点是这种存储方式的核心，通过 points 的成员 neighbourlist 可以找到它的邻接点索引号，通过 map 容器 mp 可以找到对应边的索引号，通过 points 的成员 neighbourface 可以找到它的邻接面。通过 edges 的成员 point[2]可以找到它的 2 个端点的索引号，再由端点的索引号可以寻找点，线，面。通过面 faces 的成员 point[3]可以找到三个顶点的索引号，通过 map 容器 mp 可以寻找构成面的三条边。

3.2.4 DFD 图

数据流图显示了 obj 文件中的数据，用户通过 GUI 输入的数据，以及通过程序的算法处理后的数据走向。

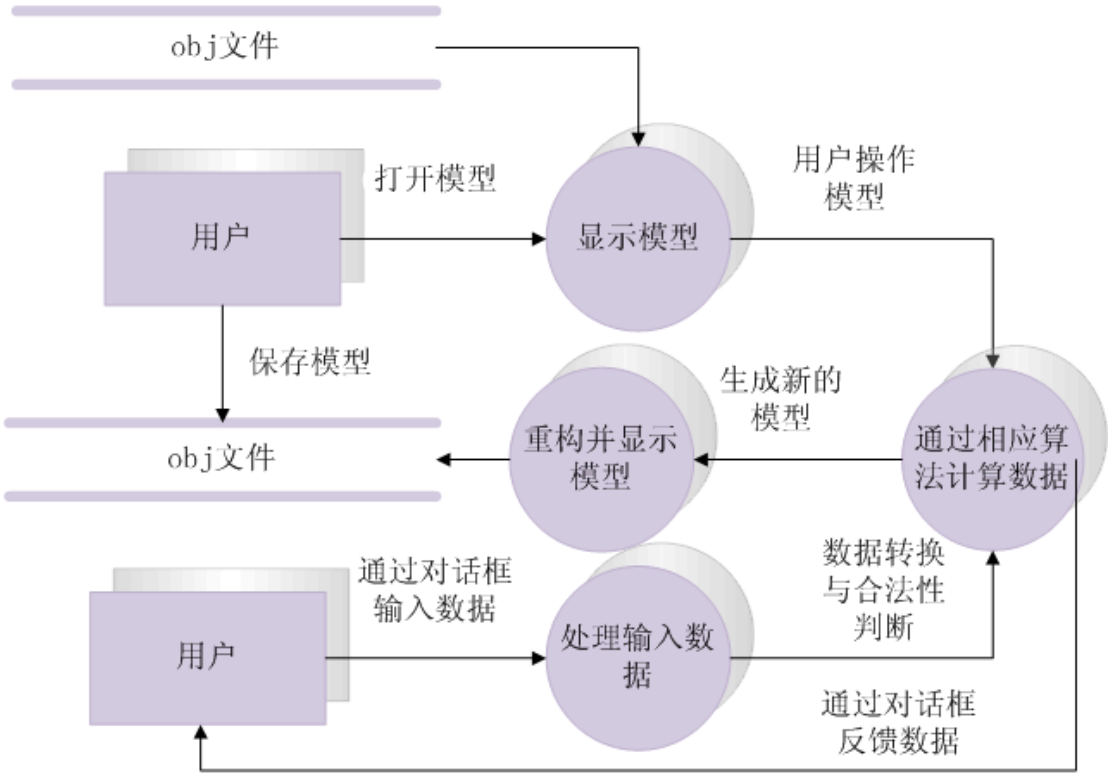


图 3-1 数据流图

3.3 用例设计

以下是用户对本系统操作的用例图，包括查看模型，使光照获得光照，平移模型，旋转模型，缩放模型，视角缩放，选择模型上的三角面，平移模型上的点或点群，简化模型，细分模型，平滑模型，对模型实施普通拉普拉斯形变，对模型实施优化拉普拉斯形变，保存模型。

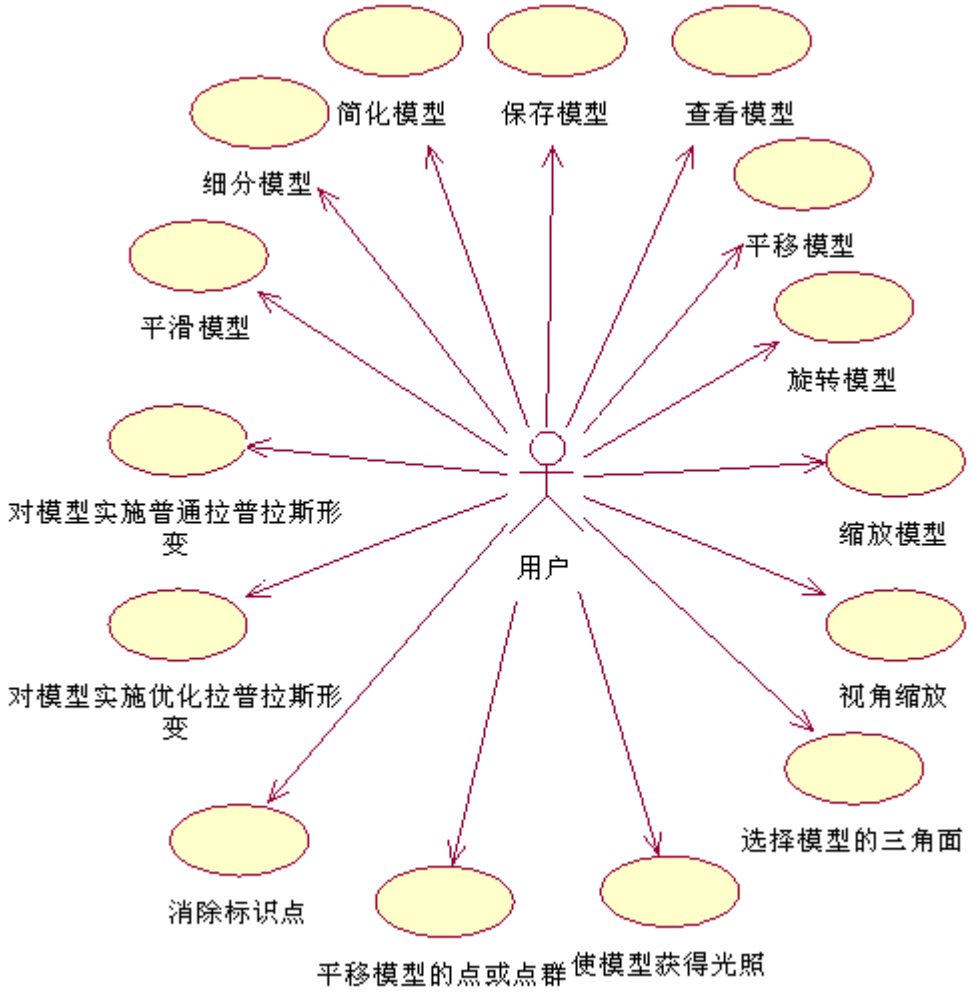


图 3-2 用例图

3.4 类设计

以下是三角网格模型拉普拉斯形变平台的类图，一共有 16 各类，每个类的成员，成员函数以及类之间的关系如下图所示。

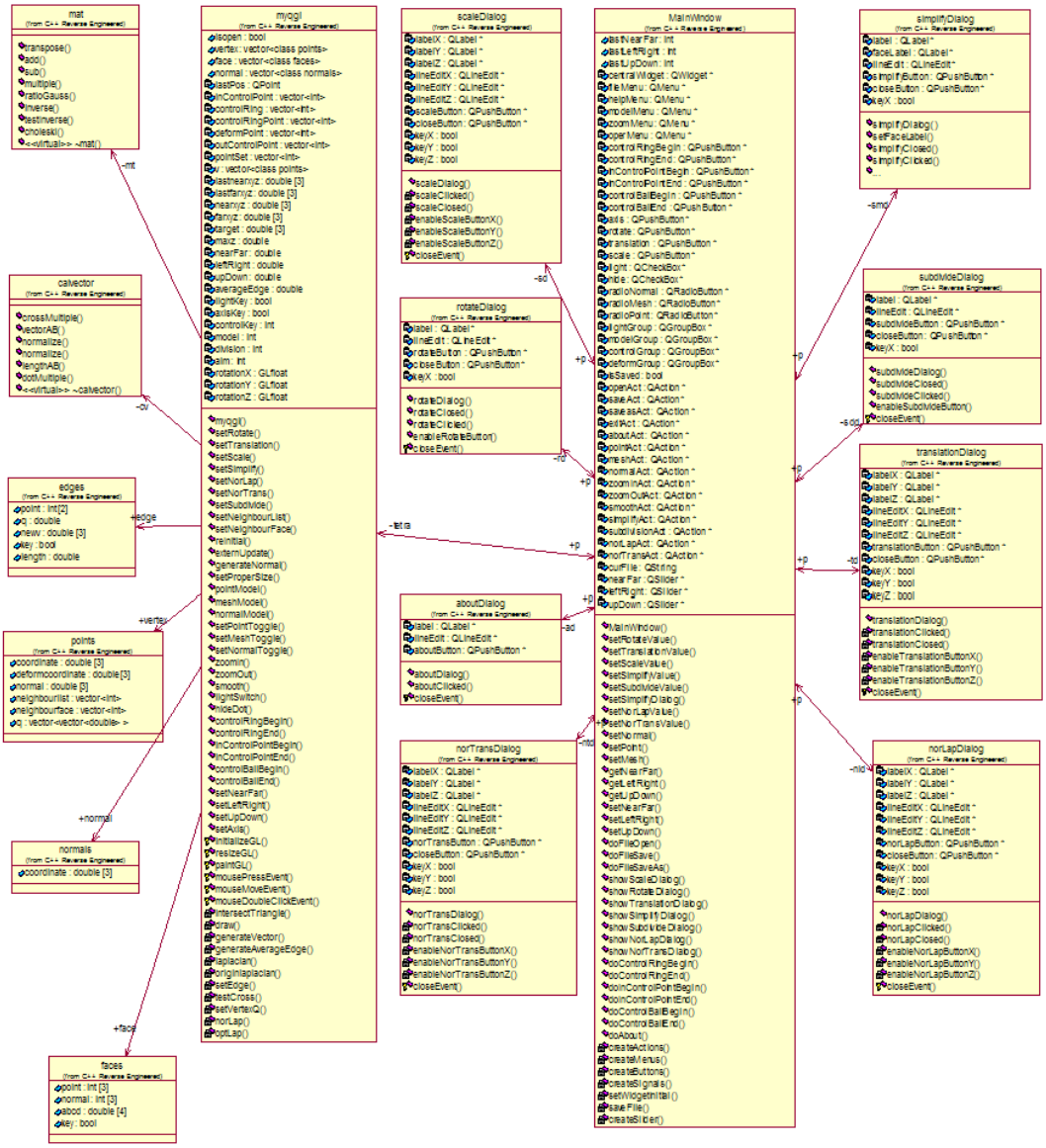

mat
transpose()
add()
sub()
multiple()
ratioGauss()
inverse()
testinverse()
cholesk()
~mat()
myqgl
isopen : bool
vertex : vector<class points>
face : vector<class faces>
normal : vector<class normals>
lastPos : QPoint
lnControlPoint : vector<int>
controlRing : vector<int>
controlRingPoint : vector<int>
deformPoint : vector<int>
outControlPoint : vector<int>
pointSet : vector<int>
w : vector<class points>
lastnarryz : double [3]
lastfarryz : double [3]
nearryz : double [3]
farryz : double [3]
target : double [3]
maxz : double
nearFar : double
leftRight : double
upDown : double
averageEdge : double
rightKey : bool
altKey : bool
controlKey : int
model : int
pMvision : int
pain : int
rotationX : GLfloat
rotationY : GLfloat
rotationZ : GLfloat
myqgl()
setRotate()
setTranslation()
setScale()
setSimplify()
setNorLap()
setNorTrans()
setSubMode()
setNeighbourList()
setNeighbourFace()
reinitial()
enterUpdate()
generateNormal()
setProperSize()
pointModel()
meshMode()
normalMode()
setPointToggle()
setMeshToggle()
setNormalToggle()
zoomIn()
zoomOut()
smooth()
lightSwitch()
hideDot()
controlRingBegin()
controlRingEnd()
lnControlPointBegin()
lnControlPointEnd()
controlBallBegin()
controlBallEnd()
setNearFar()
setLeftRight()
setUpDown()
setAxis()
initializeGL()
resizeGL()
paintGL()
mousePressEvent()
mouseMoveEvent()
mouseDoubleClickEvent()
intersectTriangle()
draw()
generateVector()
generateAverageEdge()
laplacian()
originlaplacian()
setEdge()
testCross()
setVertexQ()
norLap()
optLap()
scaleDialog
labelX : QLabel *
labelY : QLabel *
labelZ : QLabel *
lineEditX : QLineEdit *
lineEditY : QLineEdit *
lineEditZ : QLineEdit *
scaleButton : QPushButton *
closeButton : QPushButton *
keyX : bool
keyY : bool
keyZ : bool
scaleDialog()
scaleClicked()
scaleClosed()
enableScaleButtonX()
enableScaleButtonY()
enableScaleButtonZ()
closeEvent()
MainWindow
lastNearFar : int
lastLeftRight : int
lastUpDown : int
centralWidget : QWidget *
fileMenu : QMenu *
helpMenu : QMenu *
modelMenu : QMenu *
zoomMenu : QMenu *
operMenu : QMenu *
controlRingBegin : QPushButton *
controlRingEnd : QPushButton *
lnControlPointBegin : QPushButton *
lnControlPointEnd : QPushButton *
controlBallBegin : QPushButton *
controlBallEnd : QPushButton *
axis : QPushButton *
rotate : QPushButton *
translation : QPushButton *
scale : QPushButton *
light : QCheckBox *
hide : QCheckBox *
radioNormal : QRadioButton *
radioMesh : QRadioButton *
radioPoint : QRadioButton *
lightGroup : QGroupBox *
modelGroup : QGroupBox *
controlGroup : QGroupBox *
deformGroup : QGroupBox *
isSaved : bool
openAct : QAction *
saveAct : QAction *
saveasAct : QAction *
exitAct : QAction *
aboutAct : QAction *
pointAct : QAction *
meshAct : QAction *
normalAct : QAction *
zoomInAct : QAction *
zoomOutAct : QAction *
smoothAct : QAction *
simplifyAct : QAction *
subdivisionAct : QAction *
norLapAct : QAction *
norTransAct : QAction *
curFile : QString
nearFar : QSlider *
leftRight : QSlider *
upDown : QSlider *
MainWindow()
setRotateValue()
setTranslationValue()
setScaleValue()
setSimplifyValue()
setSubdivideValue()
setSimplifyDialog()
setNorLapValue()
setNorTransValue()
setNormal()
setPoint()
setMesh()
getNearFar()
getLeftRight()
getUpDown()
setNearFar()
setLeftRight()
setUpDown()
loadFileOpen()
loadFileSave()
loadFileSaveAs()
showScaleDialog()
showRotateDialog()
showTranslationDialog()
showSimplifyDialog()
showSubdivideDialog()
showNorLapDialog()
showNorTransDialog()
loadControlRingBegin()
loadControlRingEnd()
loadPointPointBegin()
loadPointPointEnd()
loadControlBallBegin()
loadControlBallEnd()
loadAbout()
createActions()
createMenus()
createButtons()
createSignals()
setWidgetInitial()
baseFile()
createSlider()
simplifyDialog
label : QLabel *
faceLabel : QLabel *
lineEdit : QLineEdit*
simplifyButton : QPushButton *
closeButton : QPushButton *
keyX : bool
simplifyDialog()
setFaceLabel()
simplifyClosed()
simplifyClicked()
subdivideDialog
label : QLabel *
lineEdit : QLineEdit *
subdivideButton : QPushButton *
closeButton : QPushButton *
keyX : bool
subdivideDialog()
subdivideClosed()
subdivideClicked()
enableSubdivideButton()
closeEvent()
translationDialog
labelX : QLabel *
labelY : QLabel *
labelZ : QLabel *
lineEditX : QLineEdit *
lineEditY : QLineEdit *
lineEditZ : QLineEdit *
translationButton : QPushButton *
closeButton : QPushButton *
keyX : bool
keyY : bool
keyZ : bool
translationDialog()
translationClicked()
translationClosed()
enableTranslationButtonX()
enableTranslationButtonY()
enableTranslationButtonZ()
closeEvent()
rotateDialog
label : QLabel *
lineEdit : QLineEdit *
rotateButton : QPushButton *
closeButton : QPushButton *
keyX : bool
rotateDialog()
rotateClosed()
rotateClicked()
enableRotateButton()
closeEvent()
aboutDialog
label : QLabel *
lineEdit : QLineEdit *
aboutButton : QPushButton *
aboutDialog()
aboutClicked()
closeEvent()
norTransDialog
labelX : QLabel *
labelY : QLabel *
labelZ : QLabel *
lineEditX : QLineEdit *
lineEditY : QLineEdit *
lineEditZ : QLineEdit *
norTransButton : QPushButton *
closeButton : QPushButton *
keyX : bool
keyY : bool
keyZ : bool
norTransDialog()
norTransClicked()
norTransClosed()
enableNorTransButtonX()
enableNorTransButtonY()
enableNorTransButtonZ()
closeEvent()
norLapDialog
labelX : QLabel *
labelY : QLabel *
labelZ : QLabel *
lineEditX : QLineEdit *
lineEditY : QLineEdit *
lineEditZ : QLineEdit *
norLapButton : QPushButton *
closeButton : QPushButton *
keyX : bool
keyY : bool
keyZ : bool
norLapDialog()
norLapClicked()
norLapClosed()
enableNorLapButtonX()
enableNorLapButtonY()
enableNorLapButtonZ()
closeEvent()
calvector
crossMultiple()
vectorAB()
normalize()
normalize()
lengthAB()
dotMultiple()
~calvector()
edges
point : int[2]
cq : double
new : double [3]
key : bool
length : double
points
coordinate : double [3]
deformcoordinate : double[3]
normal : double [3]
neighbourlist : vector<int>
neighbourface : vector<int>
cq : vector<vector<double> >
normals
coordinate : double [3]
faces
point : int [3]
normal : int [3]
abod : double [4]
key : bool
-mt
-cv
+edge
+vertex
+normal
+face
-sd
-rd
-ad
-nld
-smd
-sdd
-td
+p

图 3-3 类图

3.5 原型图设计

3.5.1 主界面

三角网格模型拉普拉斯形变平台的 GUI 由开源的 QT 编写，主界面如下所示：

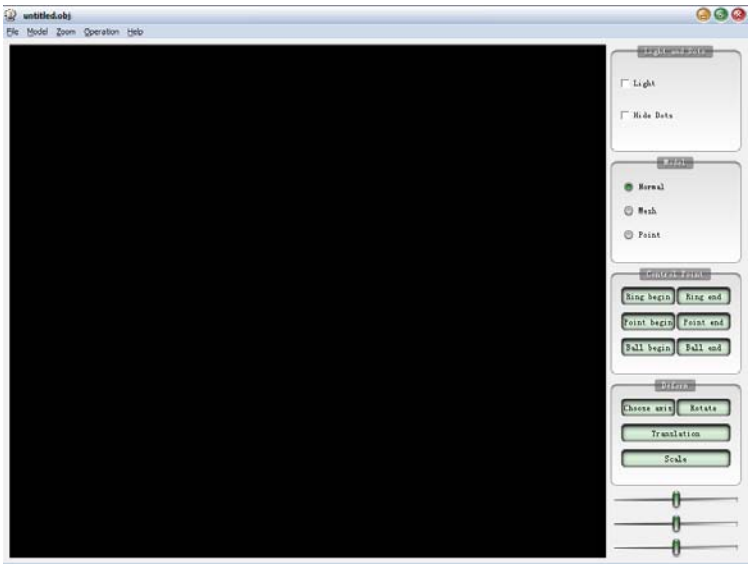


图 3-4 主界面图

主窗口的标题为载入三角网格模型 obj 文件的绝对路径与文件名。菜单栏由 File,Model,Zoom,Operation,Help 共 5 个菜单项构成。工作区的左侧为 OpenGL 窗口，用于显示三角网格模型和与用户交互，右侧为工具栏，从上到下依次为 Light and Dots 组，Model 组,Control Point 组,Deform 组。Light and Dots 组子项包括 Light,Hide Dots，当用户选择 Light 时给模型添加光照，不选择 Light 时关闭光源，当用户选择 Hide Dots 时，将隐藏程序自动生成的提示四面体与提示环，不选择 Hide Dots 时，显示程序自动生成的提示四面体与提示环。Model 组子项包括 Point,Mesh,Normal,分别实现模型的点云显示，网格显示以及普通显示模式。Control Point 组子项包括 ring begin,ring end,point begin,point end,ball begin ball end，用户点击这些按钮将分别实现开始控制环选取，结束控制环选取，开始控制点选取，结束控制点选取，开始控制球选取，结束控制球选取的功能。3 个滑块从上到下的功能分别是视角缩放，x 轴方向平移，y 轴方向平移（这里的三维

坐标系为右手笛卡尔坐标系)。

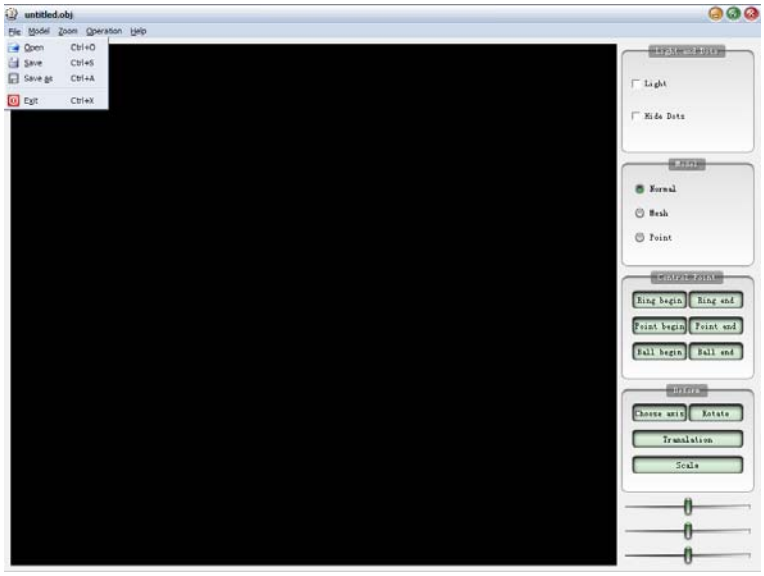


图 3-5File 菜单

File 菜单包括 Open,Save,Save as,Exit 子项，分别实现打开文件，保存文件，文件另存为和退出程序的功能，Save as 与 Exit 之间有分隔线，为每一个子项添加快捷键和图标。

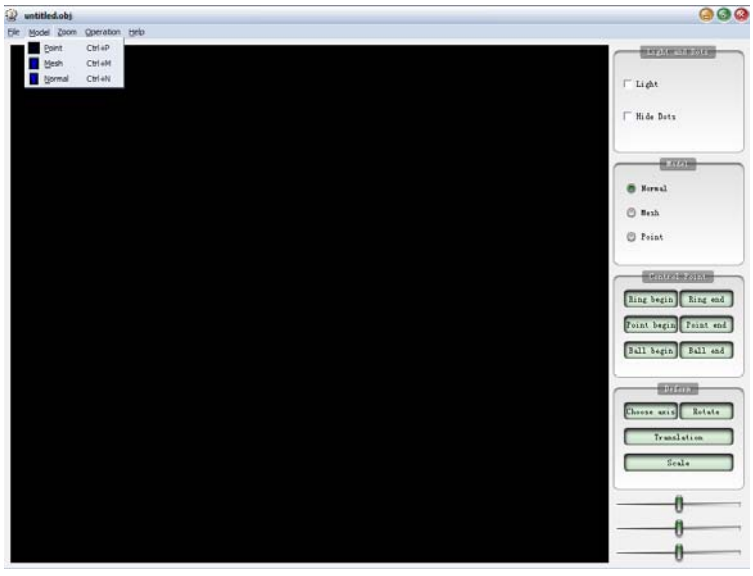


图 3-6Moel 菜单

Model 子项包括 Point,Mesh,Normal,分别实现模型的点云显示，网格显示以及普通显示模式，为每一个子项添加快捷键和图标。

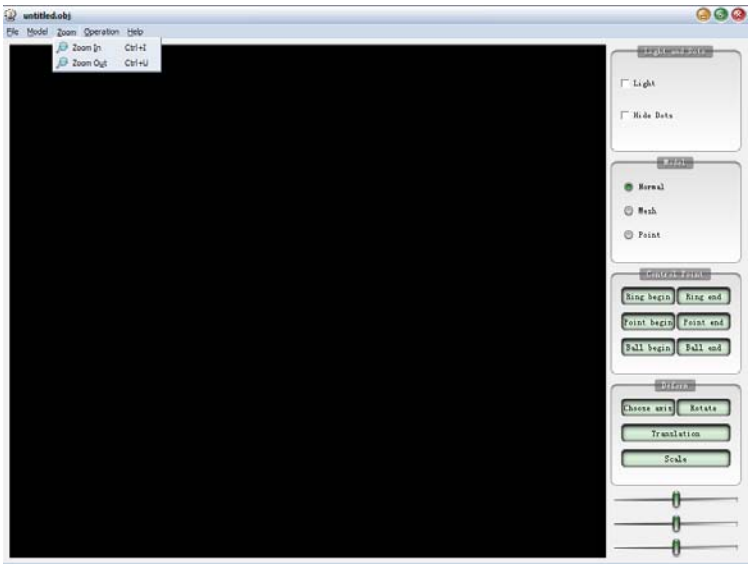


图 3-7Zoom 菜单

Zoom 菜单的子项包括 Zoom In,Zoom Out，分别实现对模型的放大与缩小，为每一个子项添加快捷键和图标。

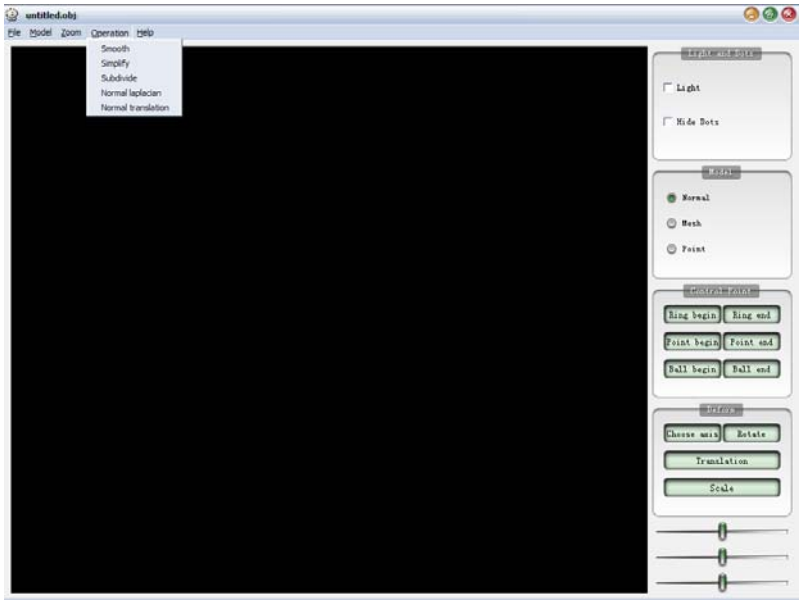


图 3-8Operation 菜单

Operation 菜单子项包括 Smooth,Simplify,Subdivide,Normal laplacian,Normal translation,分别实现网格平滑，网格简化，网格细分，普通拉普拉斯形变，控制点平移（不会影响其他点）的功能

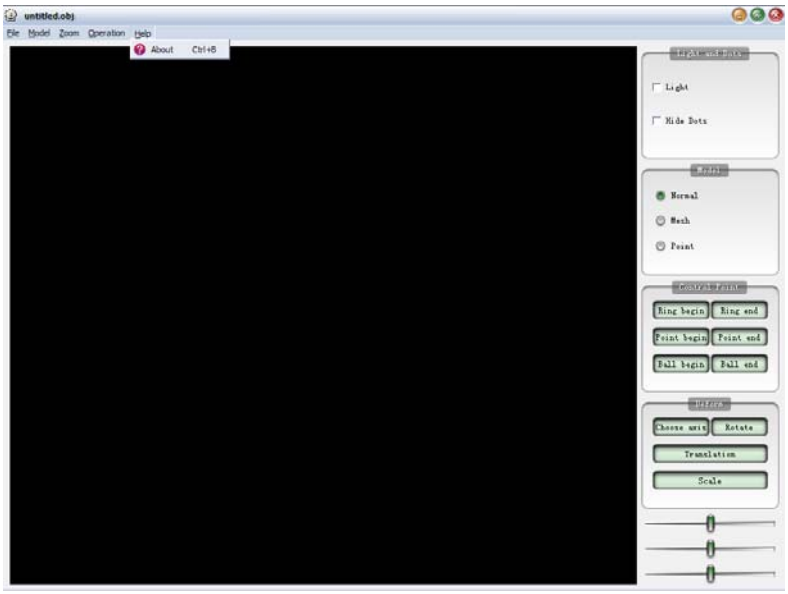


图 3-9About 菜单

Help 菜单的子项包括 About，是关于这个软件的作者（本人）的声明，添加快捷键和图标。

3.5.2 对话框

图 3-10Simplify 对话框

网格简化对话框需要用户输入保留网格剩余的面的数量 Remain faces，对话框可以检测用户输入数据的合法性以判断是否激活 Simplify 按钮。网格简化完成后，对话框返回简化后面的数量。按 Close 关闭对话框。对话框运行时，主界面的控件处于不能使用状态。

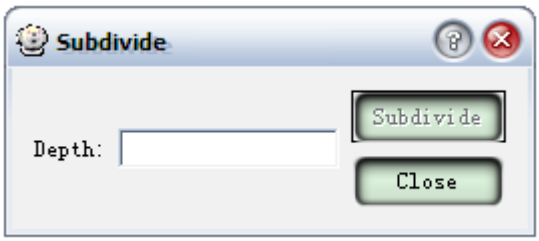


图 3-11Subdivide 对话框

网格细分对话框需要用户输入细分递归的深度 Depth，对话框可以检测用户输入数据的合法性以判断是否激活 Subdivide 按钮。按 Close 关闭对话框。对话框运行时，主界面的控件处于不能使用状态。

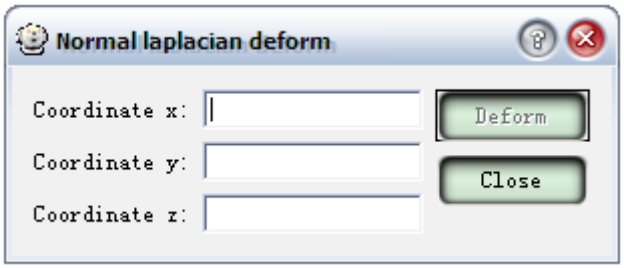


图 3-12Normal laplacian deform 对话框

普通拉普拉斯形变对话框需要用户输入平移坐标 x,y,z，对话框可以检测用户输入数据的合法性以判断是否激活 Deform 按钮。按 Close 关闭对话框。对话框运行时，主界面的控件处于不能使用状态。

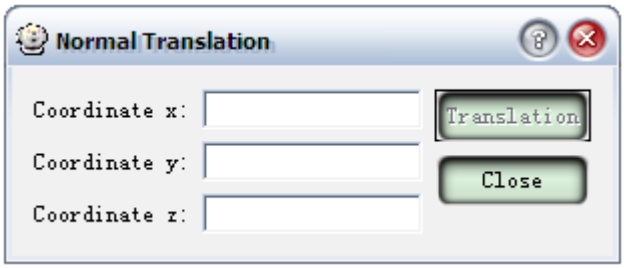


图 3-13Normal Translation 对话框

普通平移对话框需要用户输入平移坐标 x,y,z，对话框可以检测用户输入数据的合法性以判断是否激活 Translation 按钮。按 Close 关闭对话框。对话框运行时，主界面的控件处于不能使用状态。

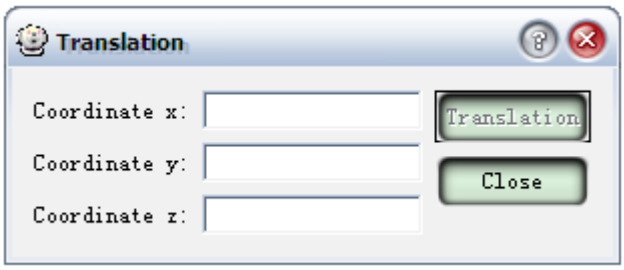


图 3-14Translation 对话框

优化拉普拉斯平移对话框需要用户输入平移坐标 x,y,z，对话框可以检测用户输入数据的合法性以判断是否激活 Translation 按钮。按 Close 关闭对话框。对话框运行时，主界面的控件处于不能使用状态。

图 3-15Rotate 对话框

优化拉普拉斯旋转对话框需要用户输入逆时针旋转角度 Rotate angel，对话框可以检测用户输入数据的合法性以判断是否激活 Rotate 按钮。按 Close 关闭对话框。对话框运行时，主界面的控件处于不能使用状态。

图 3-16Scale 对话框

优化拉普拉斯拉伸对话框需要用户输入拉伸因子 x,y,z，对话框可以检测用户输入数据的合法性以判断是否激活 Scale 按钮。按 Close 关闭对话框。对话框运行时，主界面的控件处于不能使用状态。

3.6 算法设计

3.6.1 网格简化

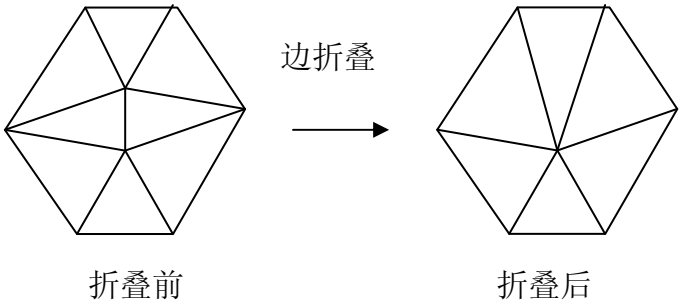


图 3-17 边折叠示意图

Garland 在[11]中比较了以前出现的网格简化算法,点簇算法（即八叉树点合并）和边消去算法，指出点簇[12]算法虽然可以简化任意边数的多边形网格，但在简化剩余面少时，简化效果往往很差。Garland 基于边消去算法为每个边构造耗散函数，然后根据耗散函数来计算耗散值，将耗散值小的边进行点合并并循环这个过程直到剩余面的数量达到某个值，这种基于耗散值的简化方法取得了很好的效果，依次 DirectX 实用库和其他一些简化库都使用这个耗散函数。

然而 Garland 的网格简化算法在简化后部分区域仍然有”较短”边的情况，我们将边长调整到与耗散值同一数量级并赋予权重作为 stl 的 heap 比较函数 cmp 的一部分来优化这个问题。用户点击网格简化交互对话框输入简化后面的数量。程序将会根据用户的输入简化到小于等于用户输入面的数量中的最大值(简化过程为每次删除 1-2 个面，所以有时不会得到简化后的准确值)。具体实现如下:

$$q_i = \sum_{j=1}^{j=n} K_{ij} = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ q_{14} & q_{24} & q_{34} & q_{44} \end{bmatrix} \quad (3\text{-}1)$$

其中 q_i 表示点 i 的耗散矩阵，为对称矩阵，n 为与点 i 邻接的面的数量。K_{ij} 表示与点 i 邻接的面的耗散矩阵。q_i 作为 points 的成员，以 vector<vector<double> >的类型存储。

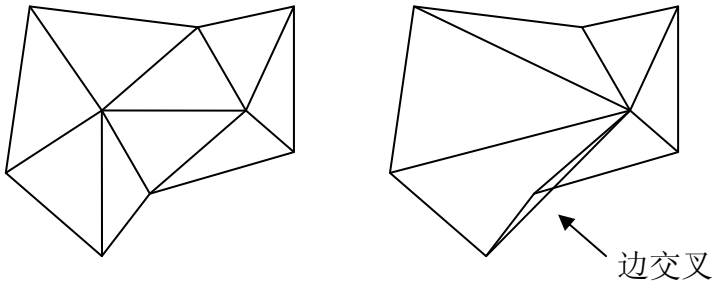
$$K_{ij}=\begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix} \quad (3\text{-}2)$$

其中 a,b,c,d 分别代表与点 *i* 相邻的一个面的平面方程 $ax+by+cz+d=0$ 的系数 a,b,c,d,且 $a^2+b^2+c^2=1$。a,b,c,d 存储于 faces 的成员 abcd[4]数组中，对应下标分别为 0,1,2,3。

预处理：首先遍历 face 容器中所有三角面面，将 map<pair<int,int>,int> mp 赋值，mp 用来通过 2 个端点查找边的索引号。计算每一个三角面的平面方程，将系数存放于 faces.abcd[4]中，计算所有点的耗散矩阵，存放于 points. q 中;为每一条边计算合并后点的耗散矩阵 $q=q_1+q_2$(q_1,q_2 为边的两个端点的耗散矩阵), 边折叠后的一个问题是寻找合适的折叠坐标，设 v=[v_x,v_y,v_z,1],对耗散函数 v^tqv 求最小值，得到合并后点的坐标存储于 edges 的成员 newv[4]:

$$newv=\begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad (3\text{-}3)$$

计算合并边耗散值 $edges.q=newv^t\ q\ newv$;将所有 edge 放入 stl 的 heap 中,形成小根堆。



折叠前 折叠后

图 3-18 边交叉示意图

每次删除堆顶边 edge[0]，首先检验合并后点的邻接面的法向量方向是否与合并前相同（即是否发生边交叉）如果相同，直接删除而不进行后面的操作。

删除 edge[0]的第二个顶点,把第二个顶点的所有邻接边,临接面中,既不是第一个顶点也不是第一个顶点的邻接点，临接面的索引 push 到第一个顶点的 vector<neighbourlist>与 vector<neighbourface>中，是第一个顶点的邻接点和邻接面的点从 vector<neighbourlist>与 vector<neighbourface>中删除,将是第二个顶点邻接点的点的 vector<neighbourlist>中删除它,将是第二个点的邻接点的 vector<neighbourface>中删除已经删除的面。查找堆中的边，将含有第二个顶点索引的边删除。重复删除 edge[0],得到简化后的模型。

3.6.2 网格细分

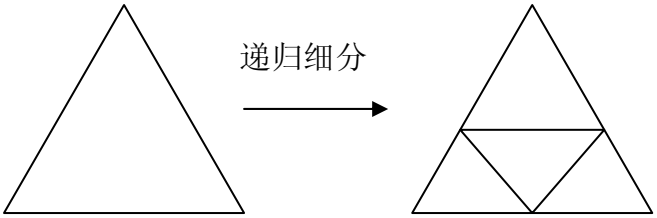


图 3-19 网格递归细分示意图

我们这里提供一种简单的细分，不需要用户选定控制顶点，用户输入细分深度 Depth,程序将输入模型的每个三角形的每条边的中点生成新的顶点，并把原来的三角形分为 4 个新的三角形，然后”删除”细分前的三角形(这里将 face.key 标为 false),细分过程中重排细分面的索引号顺序，使得法向量方向得以保持为原来面的方向，depth 将决定细分递归的次数,最后删除 face.key 为 false 的面，根据面重排顶点，顶点邻接点的索引号，顶点邻接面索引号。

3.6.3 拉普拉斯坐标

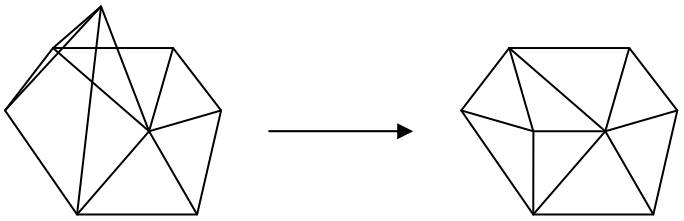
拉普拉斯坐标首先在[4]中被提出，是最近出现的进行网格模型形变的工具。几何细节被认为是网格表面的内在性质,在拉普拉斯坐标系统中，相比网格在空间的绝对坐标，我们更关心一个点与周围的点存在什么样的联系，这种联系决定了在形变操作后的网格模型的几何细节保持程度。拉普拉斯坐标通常定义为：

$$\delta_i = v_i - w_i \sum_{j \in N_i} v_j \quad (3\text{-}4)$$

其中 w_i 为拉普拉斯坐标的权值，[7]讨论了拉普拉斯权值问题，并提出了余切权值，由于余切权值需要额外的计算，我们在下面三种算法中都用简单的度权值，度 d_i 即点 i 的邻接点数量。设 $N(v_i)$表示点 i 的邻接点集，则网格的拉普拉斯矩阵 L(v)为：

$$L_{ij} = \begin{cases} 1 & i = j \\ -\dfrac{1}{d_i} & v_j \in N(v_i) \\ 0 & \text{其它} \end{cases} \quad (3\text{-}5)$$

3.6.4 网格平滑



平滑前　　平滑后

图 3-20 网格平滑示意图

网格平滑，对 SorKine 的算法进行实现。SorKine 评论了通过随机法，贪婪法，K 步贪婪法进行控制点的选择的优缺点。我们这里采用控制球选择，控制球的半径由选择区域所有边的平均变长决定，用户可以选择多个控制球来保持模型的特征区域。这样既可以使用户平滑自己的目标区域，且省去了贪婪法计算控制点的时间。

用户首先用鼠标在模型上画条封闭的环，鼠标经过的三角网格会被特殊颜色标识，然后用户用鼠标点选环内的控制点（平滑控制点在平滑过程中会尽量保持原来的坐标与凹凸程度），程序会选择控制点与控制环之间的点作为待平滑

点，生成操作这些点的容器 v:

$$E(V') = \sum_{i=1}^{n} \| L(v_i') \|^2 + \sum_{i=m}^{n} \| v_i' - u_i \|^2 \quad (3\text{-}6)$$

其中 v_i 表示容器中第 i 个点,v_i'表示平滑后的容器中的第 i 个点, n-m+1 为控制点数量，n 为点的数量。拉普拉斯矩阵 $L(v_i')$通过 myqgl 类中的 laplacian()函数来够造，以上算式中只有 v_i 为未知，最后得到||Ax-b||,通过解线性方程$(A^tA)x=A^tb$来求解 v_i。

3.6.5 普通拉普拉斯形变

普通拉普拉斯形变，普通拉普拉斯形变算法在速度上相对优化拉普拉斯形变算法有很大优势，由于不用为每个顶点计算形变矩阵，因此在对形变质量要求不高时可以使用。:用户首先用鼠标在模型上画条封闭的环，鼠标经过的三角网格会被特殊颜色标识，然后用户用鼠标点选环内的控制点，程序会选择控制点与控制环之间的点作为待形变点，选好后程序会对环内区域以及从环内区域向非控制点的方向遍历得到形变的过渡区，误差方程可表述为:

$$E(V') = \sum_{i=1}^{n} \| \delta_i - L(v_i') \|^2 + \sum_{i=m}^{n} \| v_i' - u_i \|^2 + k \sum_{i=n}^{p} \| v_i' - w_i \| \quad (3\text{-}7)$$

其中 v_i 表示容器中第 i 个点,v_i'表示形变后的容器中的第 i 个点,V’表示形变后点的集合， δ_i 是点 v_i 形变前的拉普拉斯坐标，v_i 是 n 是用户选择的控制点，待形变点，和控制环点的数量的和。n-m+1 为控制点与控制环点的和。p-n+1 为控制环外的过渡区点的数量。k 代表过渡区点保持原来坐标的权重,这个值随着过渡区点与控制环在模型上的最短路径而变化，最短路径越短，则这个值越小。最后得到||Ax-b||,通过解线性方程$(A^tA)x=A^tb$ 来求解 v_i。

3.6.6 优化拉普拉斯形变

优化拉普拉斯形变算法，实现 Lipman 的拉普拉斯算法,对求解过程进行优化，并对原来公式进行改进，加入过渡区。具体实现方法如下所述:用户首先用鼠标在模型上画条封闭的环，鼠标经过的三角网格会被特殊颜色标识，然后用户用鼠标点选环内的控制点，程序会选择控制点与控制环之间的点作为待形变点，选好后程序会对环内区域以及从环内区域向非控制点的方向遍历得到形变的过

渡区，生成操作这些点的容器 v，误差方程可表述为:

$$E(V') = \sum_{i=1}^{n} \| T_i(V')\delta_i - L(v_i') \|^2 + \sum_{i=m}^{n} \| v_i' - u_i \|^2 + k\sum_{i=n}^{p} \| v_i' - w_i \| \quad (3\text{-}8)$$

其中 v_i 表示容器中第 i 个点,v_i'表示形变后的容器中的第 i 个点,V'表示形变后点的集合，T_i 表示点 v_i 的形变矩阵，δ_i 是点 v_i 形变前的拉普拉斯坐标，v_i 是 n 是用户选择的控制点，待形变点，和控制环点的数量的和。n-m+1 为控制点与控制环点的和。p-n+1 为控制环外的过渡区点的数量。k 代表过渡区点保持原来坐标的权重,这个值随着过渡区点与控制环在模型上的最短路径而变化，最短路径越短，则这个值越小。误差方程的第一项可解释为对带形变点原有拉普拉斯坐标进行旋转，使得得到的结果与形变后点的拉普拉斯坐标平方差最小。第二项与第三项则是保持控制点，控制环点，过渡区点形变后的坐标与形变前的平方差最小。

T_i(V')的估计方法，首先同向拉伸与旋转的形变矩阵为:

$$T = s\exp(H) \quad (3\text{-}9)$$

s 为伸缩因子，H 为斜对称矩阵:

$$H = \begin{bmatrix} 0 & -h_3 & h_2 \\ h_3 & 0 & -h_1 \\ -h_2 & h_1 & 0 \end{bmatrix} \quad (3\text{-}10)$$

将 exp(H)进行幂级数展开，在 h1,h2,h3 较小时，$h1^2+h2^2+h3^2 \approx 0$,由此可得:

$$s\exp(H) = \begin{bmatrix} s & -h_3 & h_2 & t_x \\ h_3 & s & -h_1 & t_y \\ -h_2 & h_1 & s & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3\text{-}11)$$

$s,h_1,h_2,h_3,t_x,t_y,t_z$ 都为未知量，为此估计点 i 及其邻接点的局部框架，由下面算式求解:

$$\left\| \begin{bmatrix} v_{k_x} & 0 & v_{k_z} & -v_{k_y} & 1 & 0 & 0 \\ v_{k_y} & -v_{k_z} & 0 & v_{k_x} & 0 & 1 & 0 \\ v_{k_z} & v_{k_y} & -v_{k_x} & 0 & 0 & 0 & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} [s_i, h_1, h_2, h_3, t_x, t_y, t_z]^T - \begin{bmatrix} v'_{k_x} \\ v'_{k_y} \\ v'_{k_z} \\ \cdots \end{bmatrix} \right\|^2 \quad (3\text{-}12)$$

其中 $k \in \{i\} \cup N_i$,N_i 为点 v_i 的邻接点集合。由此得到最后得到 $s,h_1,h_2,h_3,t_x,t_y,t_z$ 与 v_i 的线性关系，然后将原来的算式化为$\|Ax-b\|$,解线性方程$(A^tA)x=A^tb$。其中 A^TA 为对称正定矩阵，由 Cholesky 分解得 $A^TA=L^tL$,于是方程变为$(L^tL)x=A^tb$,迭代以求得 v_i。

第4章 项目成果

4.1 网格简化

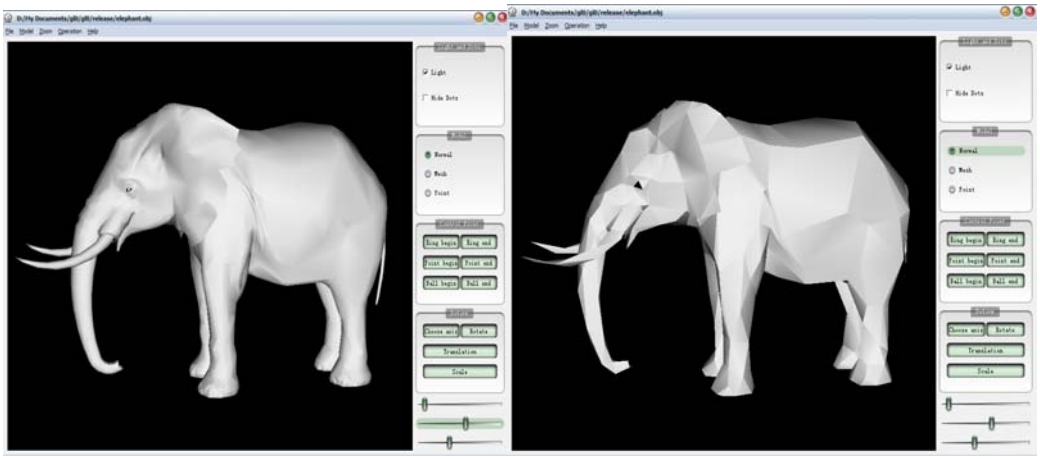


图 4-1 简化大象模型 6000 面到 800 面

对大象模型的简化效果，可以看到包括象牙等特征都得到了良好的保持。

4.2 网格细分

图 4-2 细分维纳斯模型 1762 面到 7048 面

对维纳斯模型的网格递归细分深度 1 的效果。

4.3 网格平滑

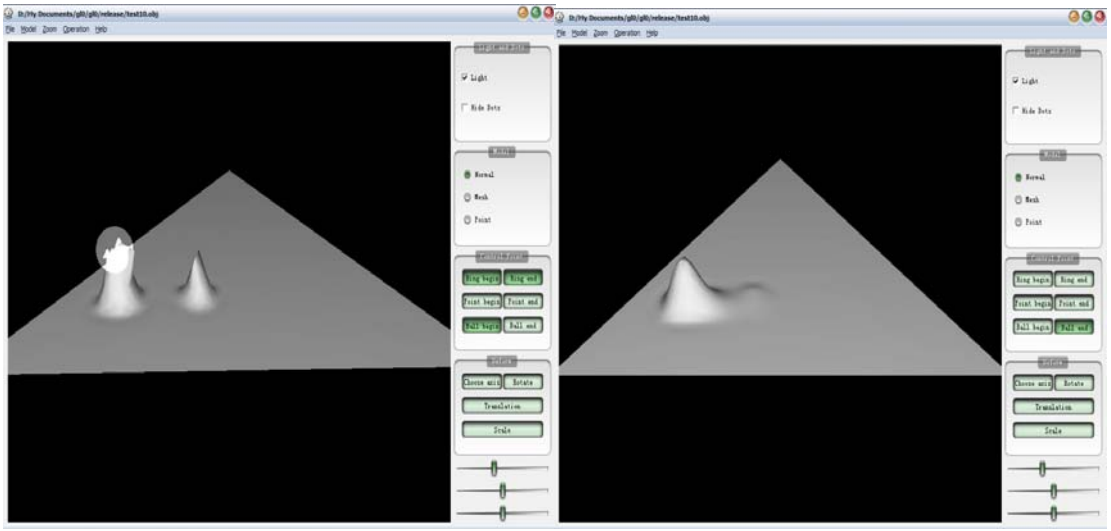


图 4-3 对 test10 模型的网格平滑效果

用控制球选择左侧的突起平滑的结果。

4.4 网格普通拉普拉斯形变

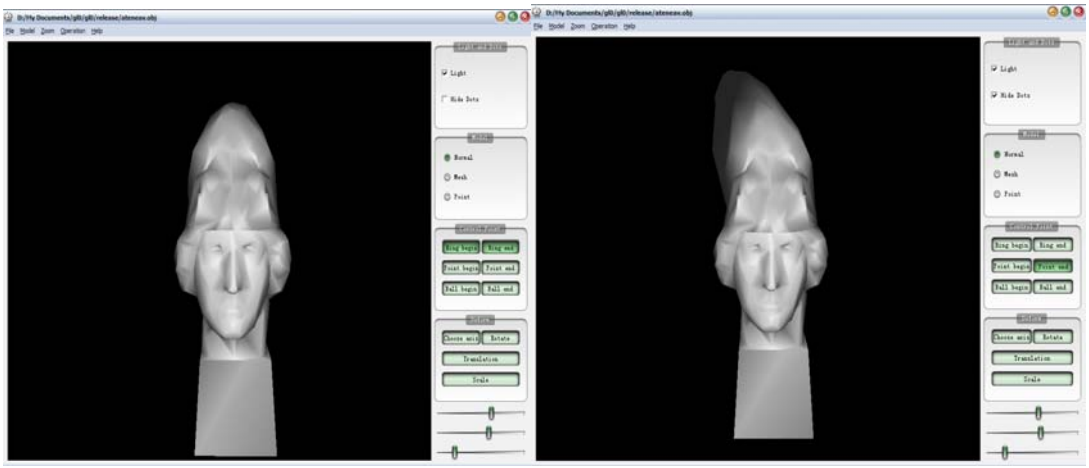


图 4-4 对雅典娜模型的普通拉普拉斯形变效果

对雅典娜模型的头部实施普通拉普拉斯形变。

4.5 网格优化拉普拉斯形变

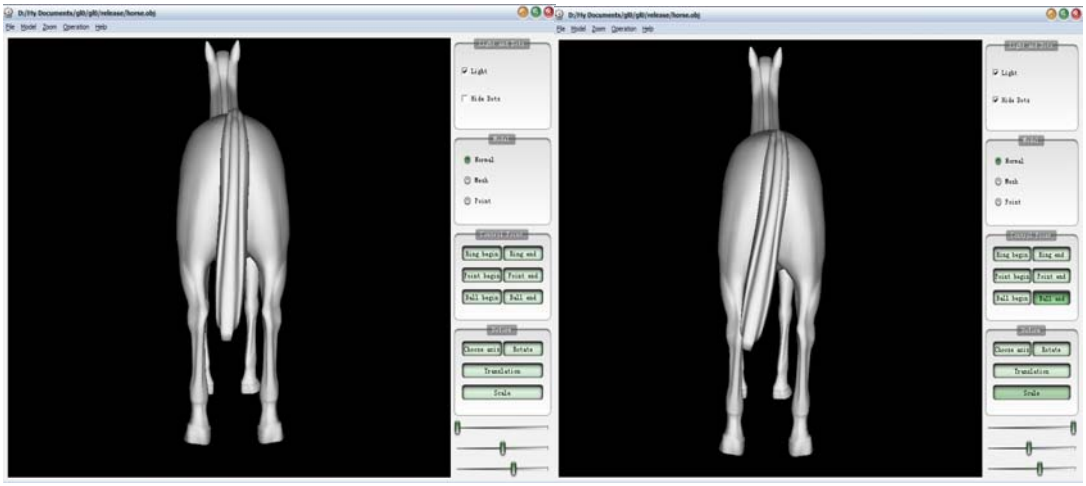


图 4-5 对马模型优化拉普拉斯形变效果

对马模型尾部的小角度旋转。可以看出表面细节良好的保持。

4.6 普通拉普拉斯形变与优化拉普拉斯形变对比

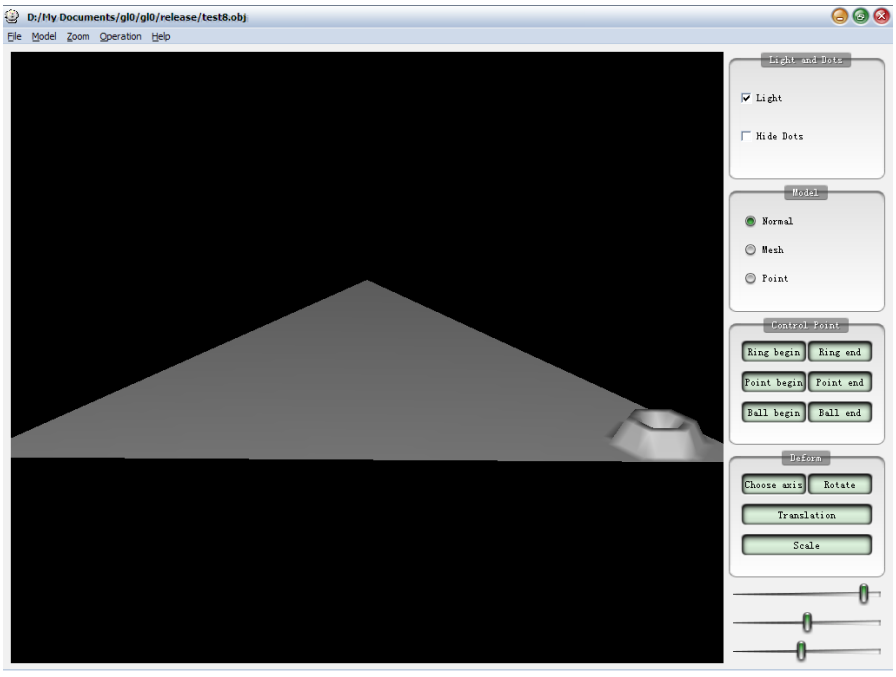


图 4-6test8 模型的原始图

原始模型，在右部有一个突起，我们以 2 种算法的截图作为对比。

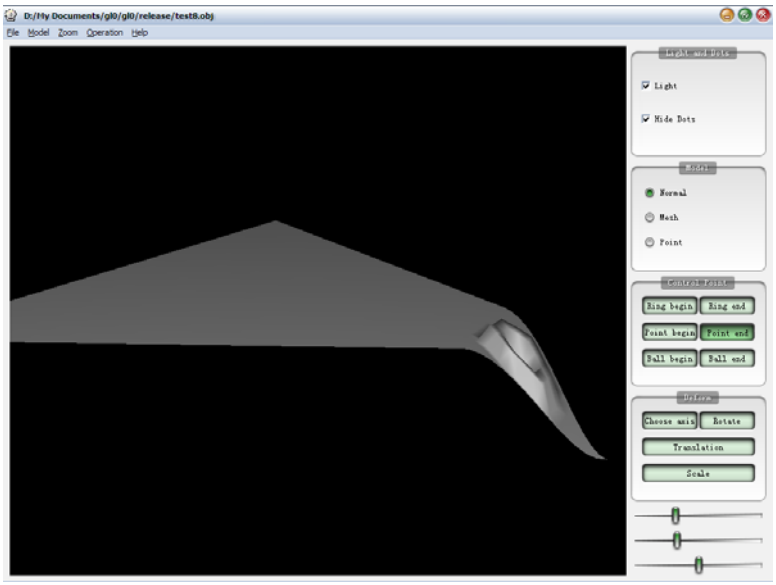


图 4-7test8 模型普通拉普拉斯形变效果（角度 1）

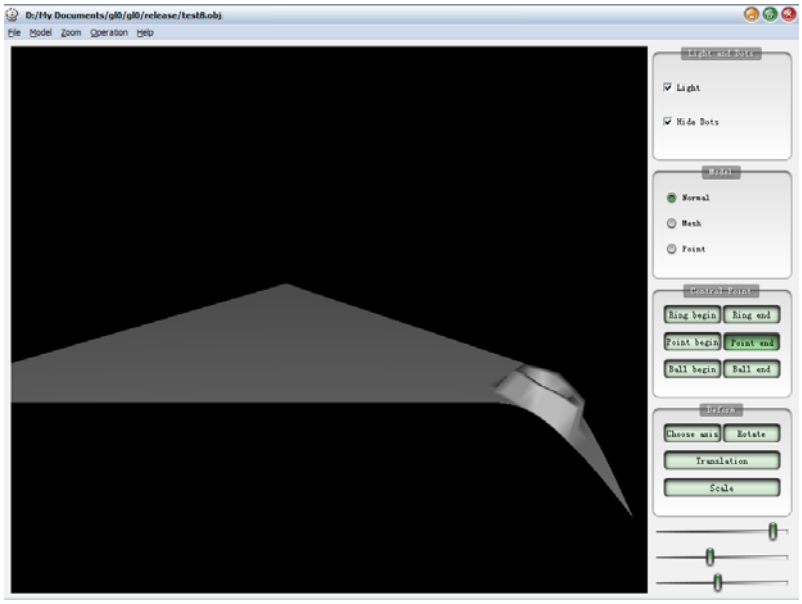


图 4-8test8 模型优化拉普拉斯形变效果（角度 1）

可以看到普通拉普拉斯算法和优化拉普拉斯算法在突起部分重构的效果。优化拉普拉斯算法更好的保持了表面细节。

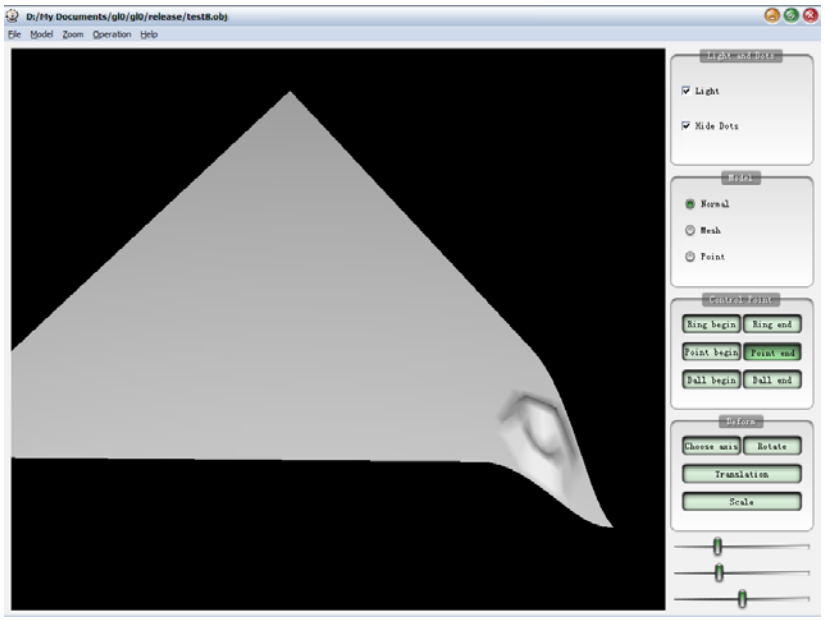


图 4-9test8 模型普通拉普拉斯形变效果（角度 2）

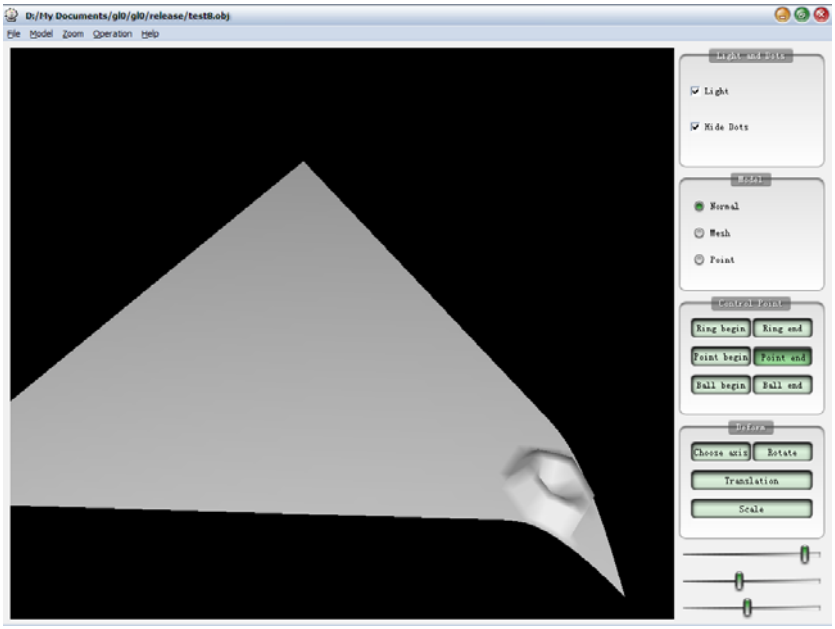


图 4-10test8 模型优化拉普拉斯形变效果（角度 2）

可以看到普通拉普拉斯算法和优化拉普拉斯算法在突起部分重构的效果。优化拉普拉斯算法更好的保持了表面细节。

4.7 时间空间复杂度分析

由于没有用到三维的容器，n 为面的数量，则所有算法的空间复杂度都为 $O(n^2)$,网格简化的时间复杂度为 O(nlogn),即建小根堆和遍历所有边并在删除边后进行 O(logn)的调整堆操作，网格细分的时间复杂度为 $O(n^2)$，要为每次细分的面的边建立 map 以便查找，网格平滑，网格普通拉普拉斯形变，网格优化拉普拉斯形变在矩阵的非 0 元较少时的时间复杂度都为 $O(n^3)$，但实际情况得到的为稀疏矩阵，时间复杂度取决于求解稀疏矩阵的求解方法的效率，一些商业软件如 MKL 的 pardiso,matlab 等都有很好求解稀疏矩阵的效率。由于是商业软件，我们的程序中求解普通拉普拉斯形变与优化拉普拉斯形变时采用的为通常的 Cholesky 分解矩阵的解法，求解网格平滑算法时没有分解矩阵，网格细分为深度为 1 的数据。以下数据为在 Inter Pentium M 1.2GHz 处理器，512MB 内存，集成显卡 82855 下测试的结果。

表 4-2 五种算法对不同模型计算的时间

模型	简化	细分(1)	平滑	普通拉普拉斯	优化拉普拉斯
维纳斯（1762面）	1.1s	0.8s	12.2s	3.0s	126.2s
大象(6000面)	4.2s	12．5s	38.6s	13.3s	367s

第5章 项目总结

在计算机图形学中，处理和重用原有的几何模型，已成为解决几何设计效率问题的重用途径，而三维几何数据重用的核心是模型的编辑与形变技术。网格形变是网格模型编辑的一个重要应用方向，在 CG，动画，游戏等领域都有广泛的应用。在三维物体的诸如点云，参数曲面，多边形网格表示方法中，由于多变形网格出色的表现能力，以及所有多边形都可以再分三角形的原因，我们选择以三角网格表示法，而比较以往的网格形变算法，在保持原来三角网格模型的表面细节方面都有着不同程度的不足，而且常常需要很多来自用户的数据输入和控制，用户的数据输入与控制很大程度影响了形变的效果，而拉普拉斯形变算法在解决表面细节方面有很大进步，但由于拉普拉斯坐标在形变中只有平移不变性，在旋转和拉伸操作后往往不能够保持不变。因此我们以三角网格模型作为构建和分析的对象，实现了算法研究的辅助平台，主要实现了对拉普拉斯坐标在旋转和拉伸操作后的局部优化算法，并在已有基础上进行改进和测试。这个平台能够读入三角网格模型文件 obj,并对读入的模型提供对网格模型的视图(点云，三角网格，正常模型)，平移，旋转，视角缩放，模型缩放，选取三角面等基本操作，并集成了网格简化，网格细分，网格平滑，普通拉普拉斯形变算法以及优化拉普拉斯形变算法的本人改进算法。由于在斜对称矩阵的泰勒展开时用到了小角度估计，因此此算法并不适于大尺度形变，关于大尺度形变的细节保持问题，还有待于我们继续研究。

参考文献

[1] Andrew Nealen,Olga Sorkine,Marc Alexa,Daniel Cohen-Or.Sketching Mesh Deformations. 2005 by the Association for Computing Machinery, Inc.

[2] Andrew Nealen,Olga Sorkine,Marc Alexa,Daniel Cohen-Or.A Sketch-Based Interface for Detail-Preserving Mesh Editing. Proceedings of ACM SIGGRAPH 2005.

[3] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl and H.-P. Seidel.Laplacian Surface Editing. Eurographics Symposium on Geometry Processing 2004.

[4] Marc Alexa.Differential coordinates for local mesh morphing and deformation.Published online: 14 February 2003© Springer-Verlag 2003.

[5] Denis Zorin, Peter Schröder and Wim Sweldens.Interactive Multiresolution Mesh Editing. Department of Computer Science, Caltech, January 1997.

[6] Yizhou Yu,Kun Zhou,Dong Xu, Xiaohan Shi,Hujun Bao,Baining Guo, Heung-Yeung Shum.Mesh Editing with Poisson-Based Gradient Field Manipulation.ACM SIGGRAPH 2004 conference proceedings.

[7] Mathieu Desbrun, Mark Meyer, Peter Schr¨oder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. SIGGRAPH 99,proceedings of the 26th annual conference on Computer graphics and interactive techniques.

[8] Yaron Lipman Olga Sorkine Daniel Cohen-Or David Levin Christian Rössl Hans-Peter Seidel.Differential Coordinates for Interactive Mesh Editing. Proceedings of the Shape Modeling International 2004.

[9] Olga Sorkine,Daniel Cohen-Or.Least-squares Meshes. Proceedings of the Shape Modeling International 2004.

[10]Huang Jin. Large Scale Geometric Deformation. Graduate School of Zhejiang University in Partial Fulfillment of the Requirement for the Degree of Doctor of Philosophy.

[11]Michael Garland,Paul S. Heckbert.Surface simplification using quadric error metrics. In SIGGRAPH 97.

[12]David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments.In SIGGRAPH 97.