

傅珂杰的程序使用说明目录

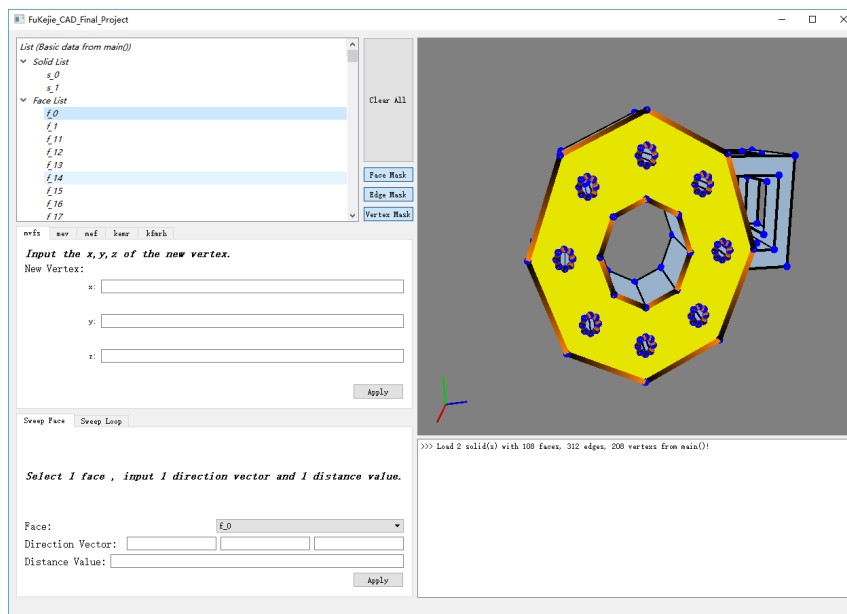
| | |
|---|----|
| 一、概述 | 2 |
| 二、程序输入（交互）方式 | 4 |
| 2.1 基于主函数直接输入 | 4 |
| 2.1.1 <i>B_rep</i> 数据结构 | 5 |
| 2.1.2 欧拉操作函数 | 8 |
| 2.1.3 <i>Sweep</i> 函数 (<i>sweep face</i> 和 <i>sweep loop</i>) | 9 |
| 2.2 基于 GUI 输入 | 10 |
| 2.2.1 <i>GUI</i> 布局介绍 | 10 |
| 2.2.2 欧拉操作界面使用 | 15 |
| 2.2.3 扫成操作界面使用 | 17 |
| 三、未在本说明中详细叙述的编程细节 | 18 |
| 3.1 非凸带洞多边形分格化 | 18 |
| 3.2 鼠标拖动三维物体旋转缩放 | 19 |

程序使用说明

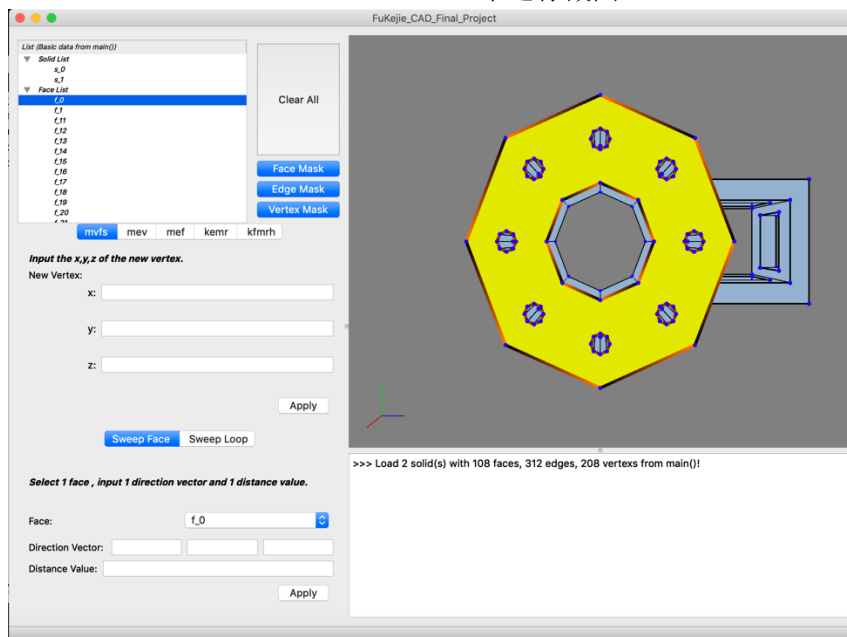
傅珂杰 11824029

一、概述

这是傅珂杰的《三维 CAD 建模》期末作业，程序是基于 QT 5.11.2 用 c++开发的，已经在 Windows 10 1709 和 MacOS Mojave 10.14.1 上测试过可以正常运行，运行界面分别如下图所示：



(a) Windows 10 下运行截图



(b) MacOS Mojave 下运行截图

图 1 Win10 系统和 MacOS 下程序运行截图

打开压缩文件（忽略隐藏文件）可以看到如下所示的文件：

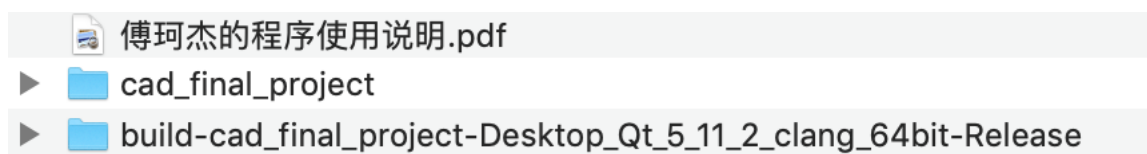


图 2 解压后的文件

一份就是本说明书，一个 `cad_final_project` 文件夹是源码文件，后面一个文件夹是已经发布的可执行文件，点击里面的 `cad_final_project.dmg` 即可以在 MacOS 运行，但是无法进行基于 `main()` 内的构建，只能基于 GUI 进行输入（交互）。

对于源码文件 `cad_final_project` 文件由于 QT 自带 OpenGL 库和 GLU 库，所以提交的文件中不包含其他动态库，源码文件夹如下图所示：

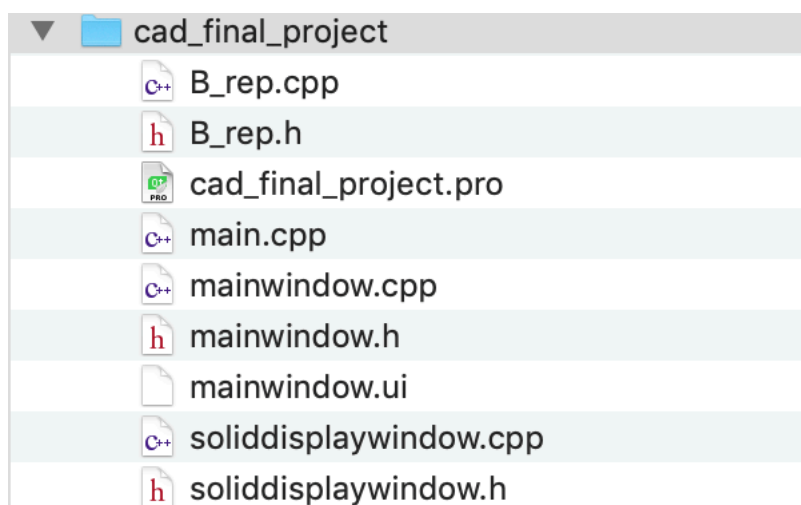


图 3 源码文件的组成

这里基于边界表示的欧拉操作数据结构及算法在 `B_rep.h` 和 `B_rep.cpp` 中；`main.cpp` 是本程序的主程序，在后面会提到可以选择在 `main()` 中直接输入欧拉操作函数，再编译运行，里面包含两个 demo，都是基于欧拉操作实现，可以选择注释掉这两个 demo 再编译运行，也可以选择 GUI 中利用图形按钮在运行；`soliddisplaywindow.h` 和 `soliddisplaywindow.cpp` 是基于 QT 自身包含的 OpenGL 库和 glu 工具库以及其他 qt 包含的库写成的 OpenGL 窗口类；`mainwindow.h`，`mainwindow.cpp` 和 `mainwindow.ui` 是 QT 上用于用户界面(GUI)代码（其中 `mainwindow.h` 和 `mainwindow.cpp` 是基于 c++编写的）；`cad_final_project.pro` 是本程序在 QT 上的工程文件。本程序的核心部分是 `B_rep.h` 和 `B_rep.cpp`，但是为了实现图形交互，在界面上也根据欧拉操作的相关输入输出做出了调整。

本程序的输入可以基于用户在编译前在 `main()` 函数通过欧拉操作函数实现输入，也可在编译运行后，在 GUI 中实现输入。

本程序经测试能够实现多个柄的模型，非凸模型的构建，能够同时构建多个分离体（但是分立体之间的相对位置无法动态调整），但是必须建立在用户遵守后面章节提到的正确输入上，否则错误输入会造成程序异常中断（虽然已经有许多错误输入的情况已经尽可能的被程序识别）。本程序不支持 undo 和 redo；在 MacOS 中放大程序窗口可能存在异常。

如果编译存在问题，请联系 kjfu@zju.edu.cn，或者手机 17816862426。Linux 上理论上也能编译，但是需要加载一下 QT 的自带 OpenGL 库。本源码已经用 Git 管理了，请放心调试。

二、程序输入（交互）方式

2.1 基于主函数直接输入

主函数内容如下图所示：

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.setWindowTitle("FuKejie_CAD_Final_Project");
    w.show();

    SolidModelingProject newProject;

    /* _____ TWO DEMO _____ */
    /* If not needed,
       * you can comment them out or in the GUI click the button "Clear All" */

    /* Demo nut */
    demo_nut(newProject);

    /* demo cube with handles */
    demo_cube_with_handles(newProject);

    /* _____ END TWO DEMO _____ */

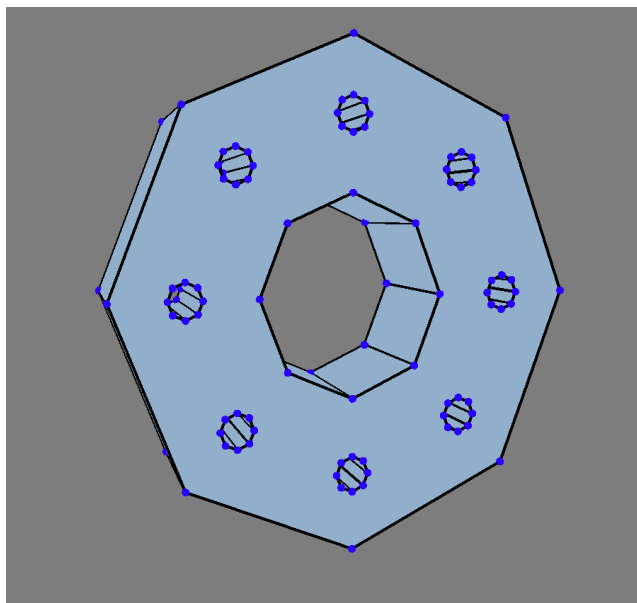
    /* Start your code following by using mvfs(), mev(), mef(), kemr(), kfmrh(), sweepFace(), sweepLoop() */

    w.linkSolid_From_main(&newProject);

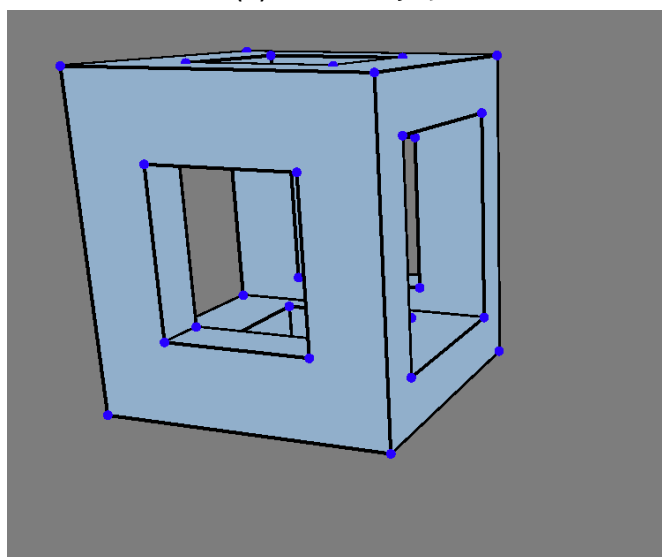
    return a.exec();
}
```

图 4 main()函数

针对主函数，做如下说明，w 变量是 QT 的主窗口；newProject 是在 B_rep 中定义的 SolidModelingProject1 类的对象，5 个欧拉操作和 Sweep 函数均是其成员函数；随后是两个 demo，都是基于欧拉操作和 sweep 操作实现的，其源码在当前函数的下方，一个是螺母，一个每个面都贯通的立方体（五个柄），效果图如下：



(a) demo1 螺母



(b) demo2 全通立方体

可以在编译后继续在这两个 demo 上继续进行其他的欧拉操作也可将这两个代码直接注释掉，再针对 newProject 对象进行欧拉操作和 sweep 操作；最后面的 `w.linkSolid_From_main(&newProject)` 是把这里 newProject 的引用传递到图形界面中，并传递给 OpenGL 窗口。

下面对本程序 B_rep 的数据结构，五个欧拉操作函数和 Sweep 操作函数做一下说明，便于用户能够使用建模，具体使用可以参照 `mian()` 下方两个 demo 的中欧拉操作的使用。

2.1.1 B_rep 数据结构

自顶向下讲（虽然本源码是自底向上写的），基本数据结构命名与教材上半边结构命名一致，除了下面这个类：

(1) SolidModelingProject 类

```

class SolidModelingProject
{
public:
    SolidModelingProject():currentSolidId(0),currentFaceId(0),currentEdgeId(0),currentVertexId(0){}
    ~SolidModelingProject();

    std::vector<Solid*> globalSolidList;
    std::vector<Face*> globalFaceList;
    std::vector<Edge*> globalEdgeList;
    std::vector<Vertex*> globalVertexList;

    Solid *mvfs(double x, double y, double z);
    Solid *mvfs(Vertex *v);//override
    Solid *SolidModelingProject::mvfs(Vertex *v)
    HalfEdge *mev(Loop *old_lp, Vertex *old_v, double x, double y, double z);
    HalfEdge *mev(Loop *old_lp, Vertex *old_v, Vertex *new_v);//override

    Loop *mef(Loop *old_lp, Vertex *v1, Vertex *v2, int mode = 0);

    void kemr(Edge *old_e, Vertex *inner_v);
    void kemr(Loop *old_lp, Vertex *inner_v, Vertex *outer_v);//override

    void kfmrh(Face *keep_face, Face *delete_face);

    void sweepLoop(Loop *lp, double *vector,double distance);
    void sweepFace(Face *f, double *vector, double distance);

    int currentSolidId;
    int currentFaceId;
    int currentEdgeId;
    int currentVertexId;
};

```

这是用来建模的类，里面包括了欧拉操作和 sweep 操作，具体使用方法见下一小节。这里包含了四个列表(用 c++ vector 当容器)，分别存储 project 中所有的 solid, face, edge(物理边)，vertex 的指针，也就是说只要在当前项目中，即使有多个分离体，其内部的点，线，面都会在当前类中可见；最后面 current*Id 分别表示全局的各个点线面体的索引值，注意，如果面或者边被删除了，其索引值就空缺！中间的函数及其重载在下一小节讲。

(2) Solid 类

```

class Solid
{
public:
    Solid():ID(-1){}
    int ID;
    std::vector<Face*> sFaces;
    std::vector<Edge*> sEdges;
    std::vector<Vertex*> sVertexes;
};

```

包含其自身索引 Id（初始化为-1）和该体内面，边，点的指针列表，与高老师课堂上讲的不同，这里不用 prev 和 next 来构成面边线的环形链表，而是直接用 c++的 vector 容器（半边 HalfEdge 还是用上课讲的 prev 和 next 构成双向环形链表）。

(3) Face 类

```
class Face
{
public:
    Face():ID(-1),fSolid(nullptr), outerLoop(nullptr){}
    Face(Loop *lp);

    int ID;
    Solid *fSolid;
    Loop *outerLoop;
    std::vector <Loop*> fLoops;

};
```

包含构造函数，根据 loop 指针构造，会建立好该 loop 和当前对象的关系；自身索引 Id;还有指向体的指针和指向面内所有环的列表的指针（c++ vector 当容器），特别还有一个单独指向外环的指针。

(4) Edge 类

```
class Edge
{
public:
    Edge(): ID(-1), eHalfEdge(nullptr){}
    Edge(Vertex *v1, Vertex *v2);// build edge v1->v2

    int ID;
    HalfEdge *eHalfEdge;

};
```

包含构造函数，利用两个顶点指针构造，方向是从 v1 指向 v2；然后是自身索引；最后在包含一个半边的指针，特别要说明的是这个半边的方向就是 v1 指向 v2，要找到另外一条对偶的半边可以用过这条半边寻找。以下是半边的数据结构。

(5) HalfEdge 类

```
class HalfEdge
{
public:
    HalfEdge(): heVertex(nullptr), heLoop(nullptr), brother(nullptr), next(nullptr), prev(nullptr),heEdge(nullptr){}
    HalfEdge(Vertex *v);

    Vertex *heVertex;
    Loop *heLoop;
    HalfEdge *brother;
    HalfEdge *next;
    HalfEdge *prev;
    Edge *heEdge;

};
```

这是所有前面提到的数据结构中，唯一不依赖 c++ vector 作为容器的类，所有半边通过 prev 和 next 构成双向环形链表。这里半边有可以通过一个点的指针完成初始化，就是将后面的

heVertex 指针指向这个点，在本说明后面的描述中，直接描述为“该半边指向该点”；还有指向环，指向下一条半边，上一条半边的指针；另外该半边的对偶边，通过 brother 指针指向，就是一条物理边的两条半边分别用各自的 brother 指针指向对方。

(6) Vertex 类

```
class Vertex
{
public:
    // build Vertex object by x,y,z
    Vertex(double x, double y, double z);

    //build Vertex Object by the difference from another Vertex object;
    Vertex(const Vertex &v, double *vector, double distance);
    int ID;
    double vCoord[4];
};
```

包含基于三维坐标的构造函数，和基于一个已知点产生位移的重载构造函数（主要为了用于 sweep 操作而写的），demo 和图形用户界面中都未曾直接使用这个构造函数。

2.1.2 欧拉操作函数

欧拉操作函数主要算法高老师都已经在课堂上讲过了，这里不在细究，主要讲一下本程序中的欧拉操作使用时需要注意的细节。

(1) mvfs () 两个重载函数

```
Solid *mvfs(double x, double y, double z);
Solid *mvfs(Vertex *v);//override
```

都是输出新建的 Solid，不同的输入，第一个输入三维坐标 x, y, z；第二个输入指向 vertex 的指针，这个重载函数主要是用于 sweep 操作，这里要求 vertex 是未被加入到 solidModelingProject 类全局点列表中的对象，即和已知的半边结构没有任何拓扑上的关系，完全新建的且独立的点。

(2) mev()函数两个重载函数

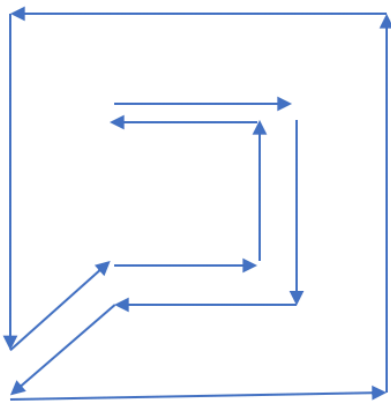
```
HalfEdge *mev(Loop *old_lp, Vertex *old_v, double x, double y, double z);
HalfEdge *mev(Loop *old_lp, Vertex *old_v, Vertex *new_v);//override
```

输出的均是半边的指针，注意，这个半边都是指向刚刚生成点，输入为一个包含原先点的环 Loop 的指针和一个在 Loop 上的点，这里注意，如果 old_v 不在 loop 上会造成程序异常中断！两个重载函数的区别在于输入新点的方式，可以是三维坐标，也可以是一个完全独立的新的点！

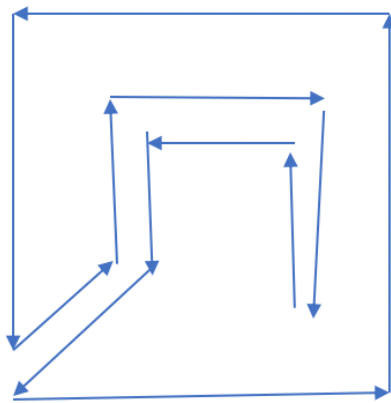
(3) `mef()` 函数特别注意!

```
Loop *mef(Loop *old_lp, Vertex *v1, Vertex *v2, int mode = 0);
```

这个函数输出是新生成的环，输入时一个已经存在的环，然后是环上的两个点，要求是这两个点都在环上，否则程序会异常中断，在 GUI 中用户可选择点和 Loop 来自行判断；这里要求边生成的方向为从 `v1` 到 `v2`；`mode` 默认情况下是 0 但是在特殊情况下，需要调到 1，就是新生成的面是为了后续步骤中生成内环的，新的面的走向与外在的大面的走向一致的时候，需要设置为 1 (否则生成内环时，内环旋转方向会和外环一致)!



Mode = 1



Mode = 0

图 5 `mef()` 中 `mode` 的选择示意图

(4) `kemr()` 1 两个重载函数

```
void kemr(Edge *old_e, Vertex *inner_v);
void kemr(Loop *old_lp, Vertex *inner_v, Vertex *outer_v); //override
```

第一个重载函数，基于输入的需要删除的边，和在物理边上位于要生成内环的点；第二个重载函数，基于输入一个内环，以及位于内环上的内部点（将要位于内环上），外部点（将要位于外环上）。

(5 kfmrh

```
void kfmrh(Face *keep_face, Face *delete_face);
```

第一个变量是要保留的面，第二个变量是要删除的面，两者不可输入反。

2.1.3 Sweep 函数 (sweep face 和 sweep loop)

(1) sweep face

```
void sweepFace(Face *f, double *vector, double distance);
```

输入为需要扫成的面，方向向量，方向向量的模，中间生成的点，线，面均会放入到全局点线面列表中。

(2) sweep loop

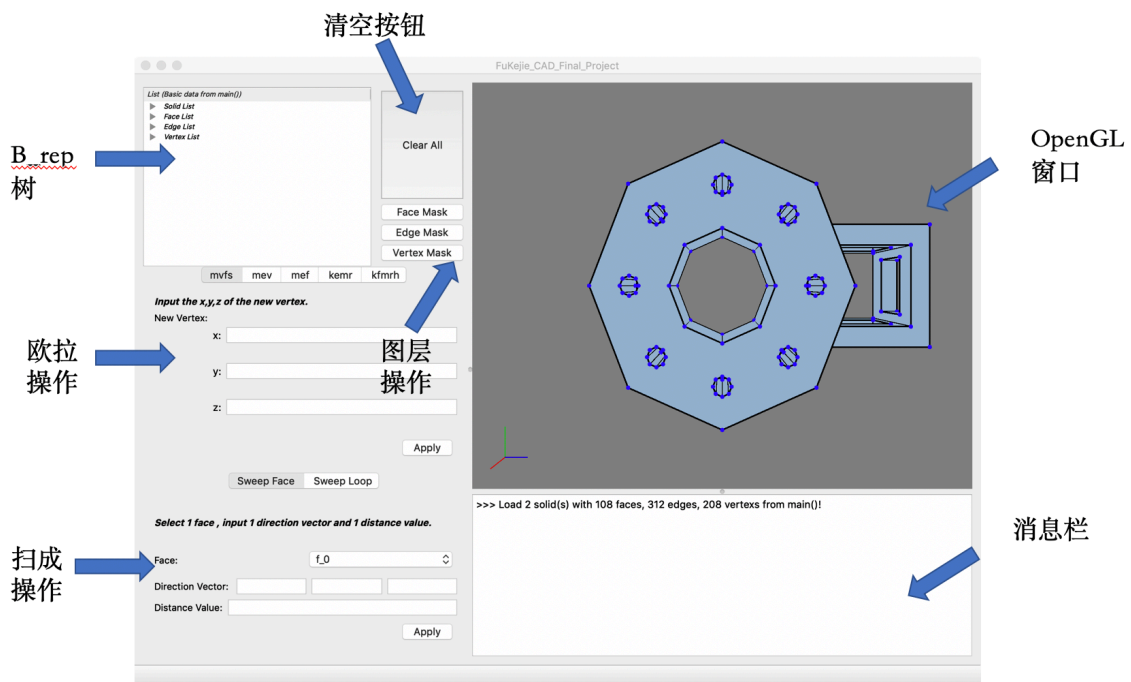
```
void sweepLoop(Loop *lp, double *vector, double distance);
```

输入为需要扫成的环，方向向量，方向向量的模，中间生成的点，线，面均会放入到全局点线面列表中。这个函数主要是为 sweep face 服务，但是，对于希望内外环不在同一个平面上的情况下，这个函数可以排上用场。

2.2 基于 GUI 输入

2.2.1 GUI 布局介绍

主要布局如下：



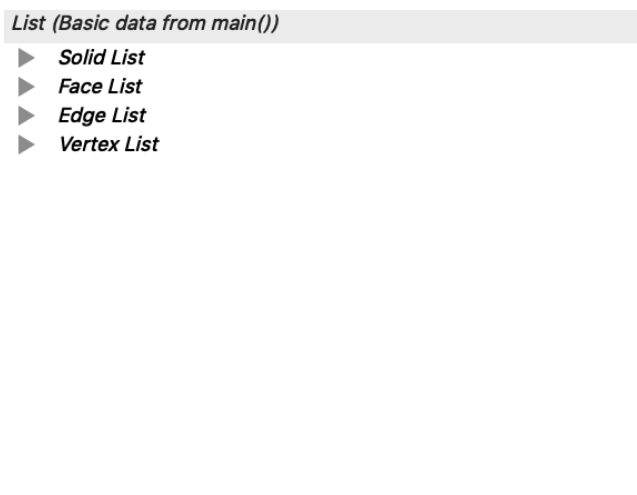
(1) 清空按钮

按下后会清空 OpenGL 窗口和消息栏，所有下拉菜单也会清空，可以用于重新开始建模。

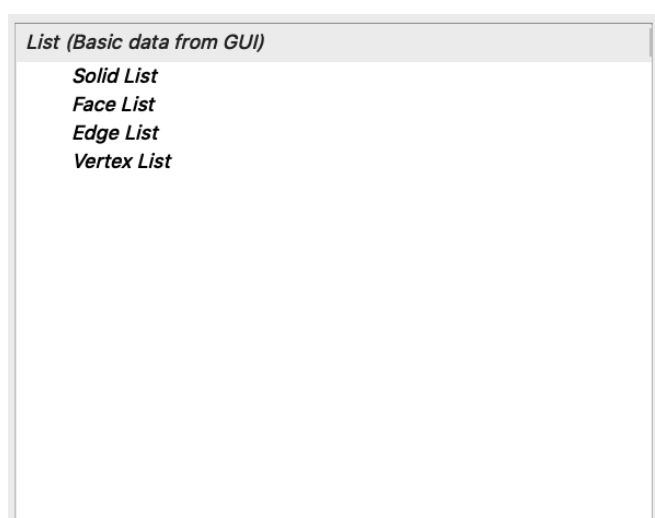
(2) B_rep 树

包含了所有的 solid, face, edge, vertex, 包括从 main () 函数中加载的。

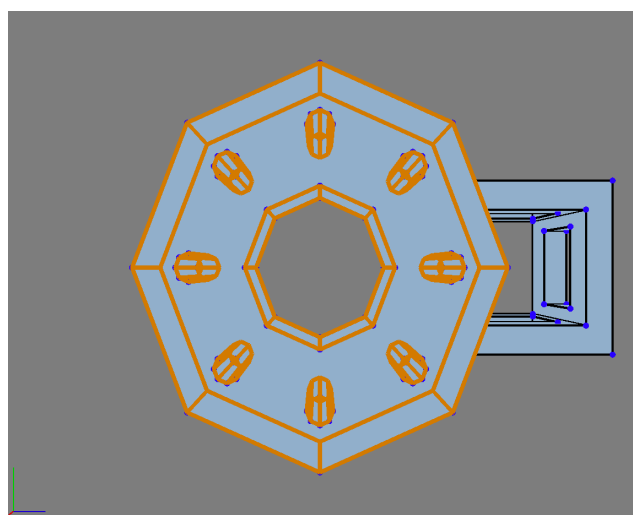
如果未注释掉 main() 里面的 demo 或者是基于 main () 里面构造的模型，树列表窗口会显示如下：



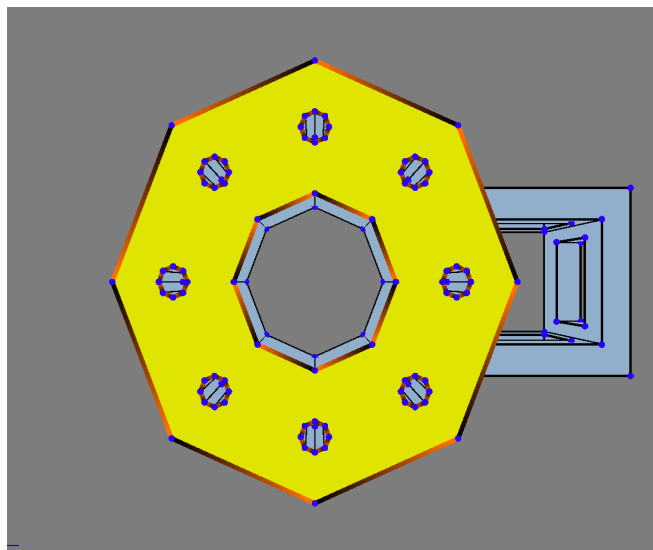
如果直接在 GUI 界面中建模，不依赖于 main（）中已经构建好的模型，会显示如下图所示：



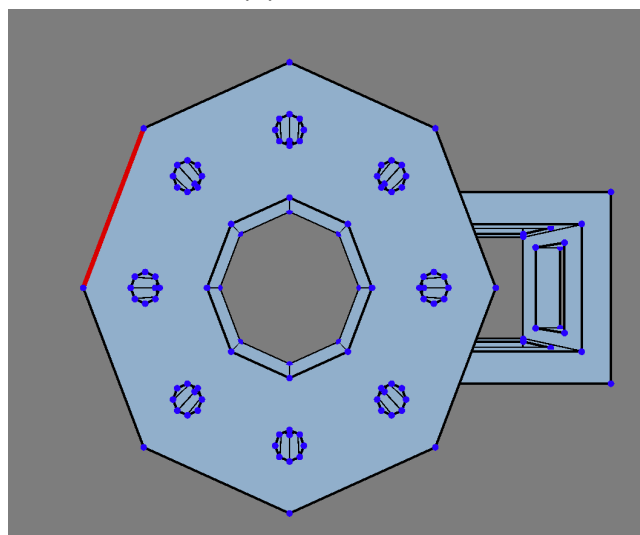
展开列表，会在 OpenGL 窗口中较粗并且变色显示选中的对象



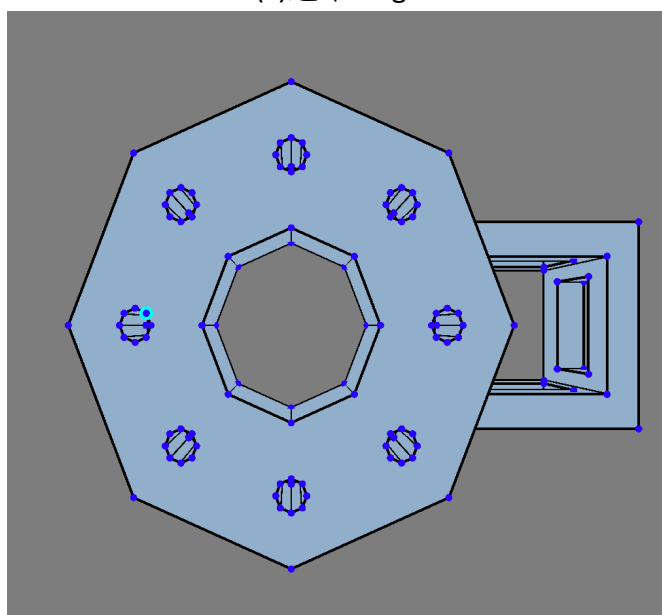
(a) 选中 solid



(b) 选中 face



(c)选中 edge



(d) 选中 vertex

图 6 B_rep 树中选中时候的变化

特别针对面做如下展开：

面内部环的方向用黑色到红色（橙色）的渐变表示，半边的方向由黑色指向红色，如下图所示，面的方向符合右手螺旋方向：

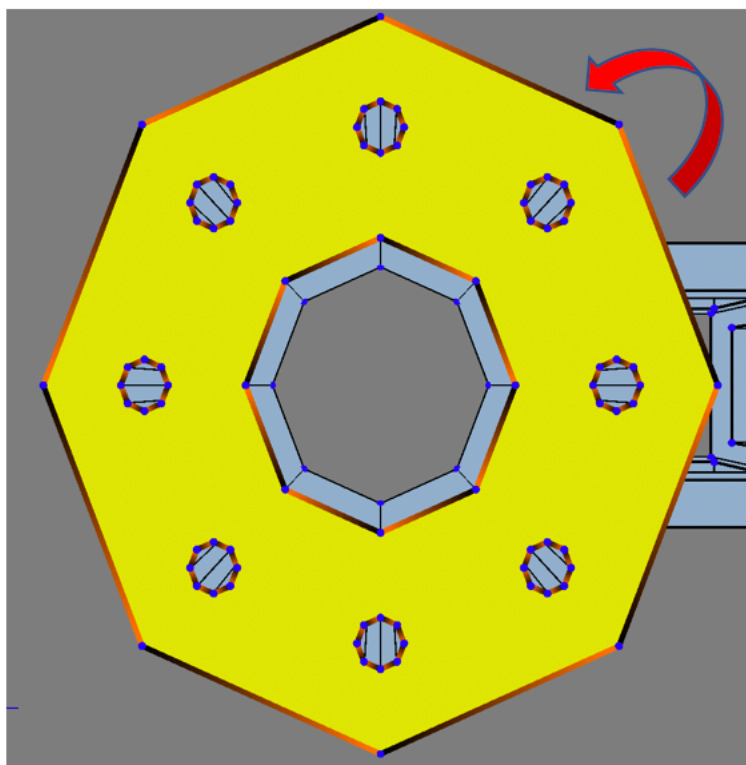


图 7 面内环的方向表示

（3）图层操作

默认三个图层操作 mask 都是打开的，当某个未打开时其相关信息消失，如下图所示为 facemask 没有被打开的情形：

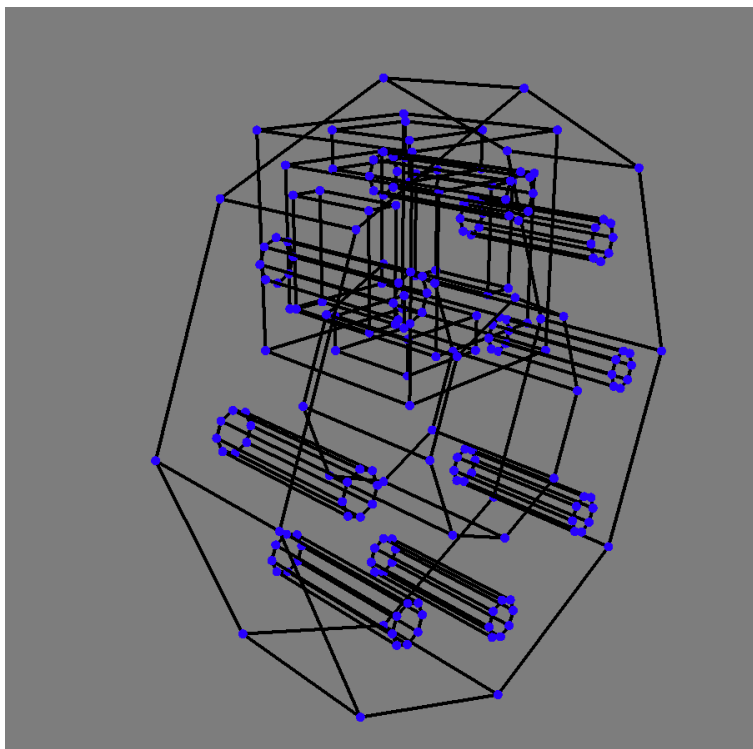
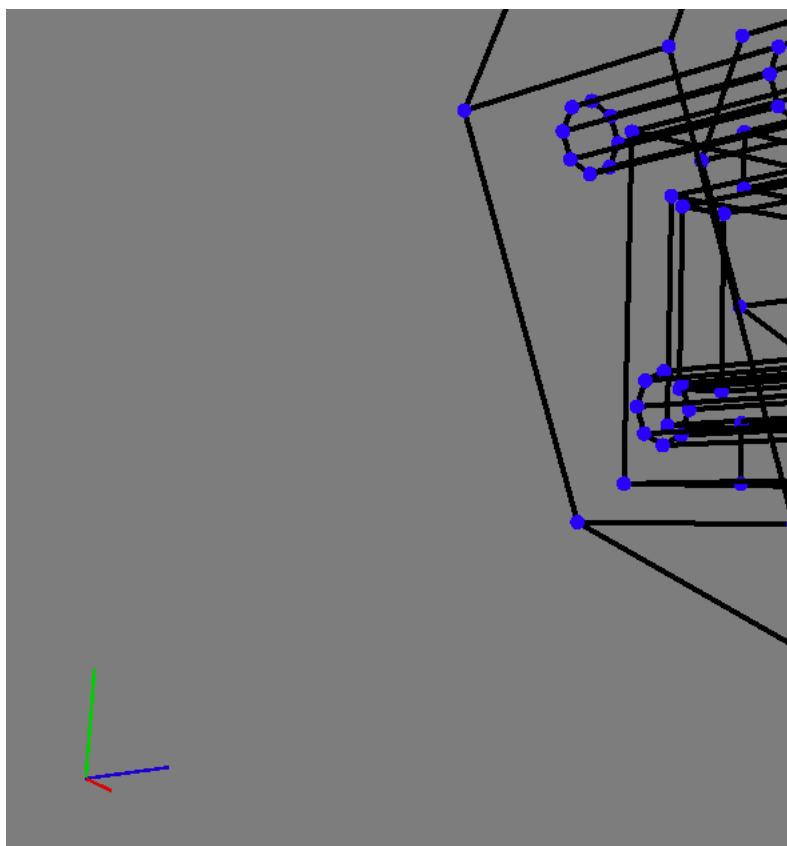


图 8 未打开 FaceMask 的情形

(4) OpenGL 窗口

支持放大旋转拖动。还有绝对坐标轴指明方向



红轴 z 方向，蓝轴 x 方向，绿轴 y 方向。鼠标滚轮可以实现放大，按住鼠标左键移动鼠标可以实现旋转，按住鼠标右键移动鼠标可以实现 oxy 明面的平移。

(5) 消息栏

输出每次操作点线面体的变化情况，由于时间原因没有涉及大量的消息反馈能力。

```
>>> Load 2 solid(s) with 108 faces, 312 edges, 208 vertexs from main(!)

[GUI]>>> mvfs() builds: s_2, f_122, v_208.

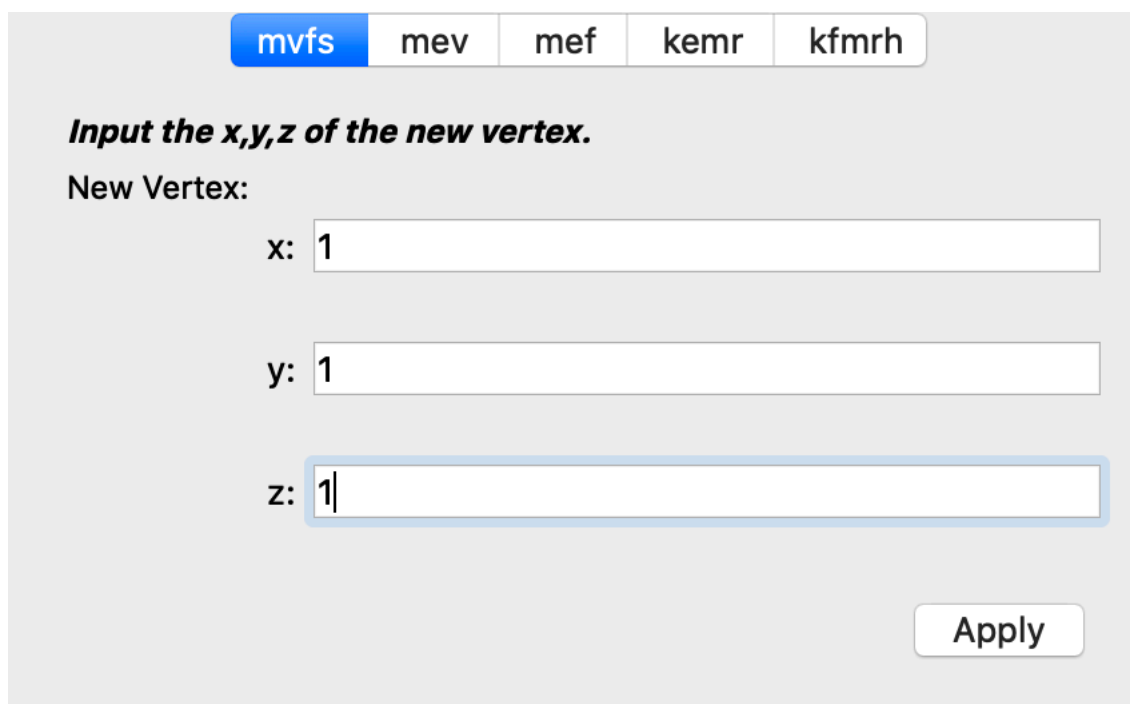
[GUI]>>> mvfs() builds: s_3, f_123, v_209.

[GUI]>>> mvfs() builds: s_4, f_124, v_210.
```

2.2.2 欧拉操作界面使用

选择任意下拉菜单对象，都会在 OpenGL 窗口加粗变色。

(1) 输入三维坐标即可，最后点 apply。



mvfs mev mef kemr kfmrh

Input the x,y,z of the new vertex.

New Vertex:

x: 1

y: 1

z: 1

Apply

(2) 选择面中的环，再旋转环上的点，再输入坐标 xyz 最后点 apply

mvfs mev **mef** kemr kfmrh

Select 1 loop and 1 vertex, input x, y, z of the new vertex.

Loop: f_0 lp_0

Old Vertex: v_0

New Vertex: x:
y:
z:

Apply

(3) 选择面中的环，在选择一个起始点，和一个终止点，特殊情况下勾选最后一个圆环，当且仅当内部面上边的走向和外部面的环的走向不同的时候！

mvfs mev **mef** kemr kfmrh

Select 1 loop and two vertex (v1->v2).

Loop: f_0

Start Vertex: v_0

End Vertex: v_0

☐ Special choice for make a inner face

Apply

(4) 两种模式，选择环+两个点，或者边+一个点

mvfs

mev

mef

kemr

kfmrh

☒ **a. Select 1 loop, 1 outer vertex and 1 inner vertex**

Loop:

f_0

Outer Vertex:

v_0

 Inner Vertex:

v_0

☐ **b. Select 1 edge to be deleted and 1 inner vertex**

Edge:

e_0

 Vertex:

v_0

Apply

(5) 选择要保留的面和要删除的面，点 apply 即可。

mvfs

mev

mef

kemr

kfmrh

Select 1 face to be reserved and 1 face to be deleted.

Face to be reserved:

f_0

Face to be deleted:

f_0

Apply

2.2.3 扫成操作界面使用

(1) 选择面，输入法向量，模既可以点击 apply，特别说明，可以直接在 demo 上进行 sweep。

Sweep Face

Sweep Loop

Select 1 face , input 1 direction vector and 1 distance value.

Face:

Direction Vector:

Distance Value:

Apply

(2) 选择环，输入法向量，模既可以点击 apply，特别说明，可以直接在 demo 上进行 sweep。

Sweep Face

Sweep Loop

Select 1 loop, input 1 direction vector and 1 distance value.

Loop:

Direction Vector:

Distance Value:

Apply

三、未在本说明中详细叙述的编程细节

3.1 非凸带洞多边形分格化

本程序中用的是 glu 工具库中的 gluNewTess() 函数，通过设置回调函数，以绘制类似 GL_POLYGON 的方式绘制外边界，内边界，函数自动绘制成非凸带洞多边形，并在 OpenGL 窗口中显示，在回调函数中设置多边形颜色，本程序中 GUI 中选择面和绘制整个实体的边界面都是采用

这个方法，只不过设置了两个关于点着色的回调函数，为了画出不同的面，详解 `soliddisplaywindow.cpp`。

本来是打算利用维尔斯特拉斯应用分析与随机研究所（WIAS）斯杭老师用于教学的二维 Delaunay 网格剖分器 `detri2` 来对所有面进行三角化，然后利用可编程 shader 进行渲染，但是改程序只能针对 z 值相等的情形，三维空间任意面片需要经过空间变换到平面中在进行剖分，如果要使用开源的三维 Delaunay 三角剖分又会设计加载大量的库，不如之间用自带的库实现方便，就放弃了使用其他开源库对网格进行三角化后，再渲染。

3.2 鼠标拖动三维物体旋转缩放

QT 自带的库能捕捉鼠标的信息，对于 OpenGL 窗口旋转缩放平移部分使用的是：

```
glMatrixMode(GL_PROJECTION);  
glLoadMatrixf(projection.data());  
glMatrixMode(GL_MODELVIEW);  
glLoadMatrixf(modelview.data());
```

由于多边形分格化使用的是固定渲染管线的方法，所以这里使用的也是固定管线的方法，通过传递投影变化矩阵，模型变换矩阵和视角变换矩阵，相关矩阵的计算使用的 QT 自带的矩阵库 `QMatrix4x4`。