区间扫描线 Z-Buffer 程序使用说明

傅珂杰(11824029) 空天信息技术(直博一年级)

目录

1.概述

这是傅珂杰《计算机图形学》课程大作业的程序说明，程序是基于 QT 5.11.2 用 c++ 开发的，程序是在 MacOS 上开发的，编译器为 Clang 8.0(Apple)，采用的 QT 自己的二维绘图类 QPainter 实现绘制的。已在 MacOS Mojave 10.14.1 上测试过可以正常运行，运行界面如下图所示:

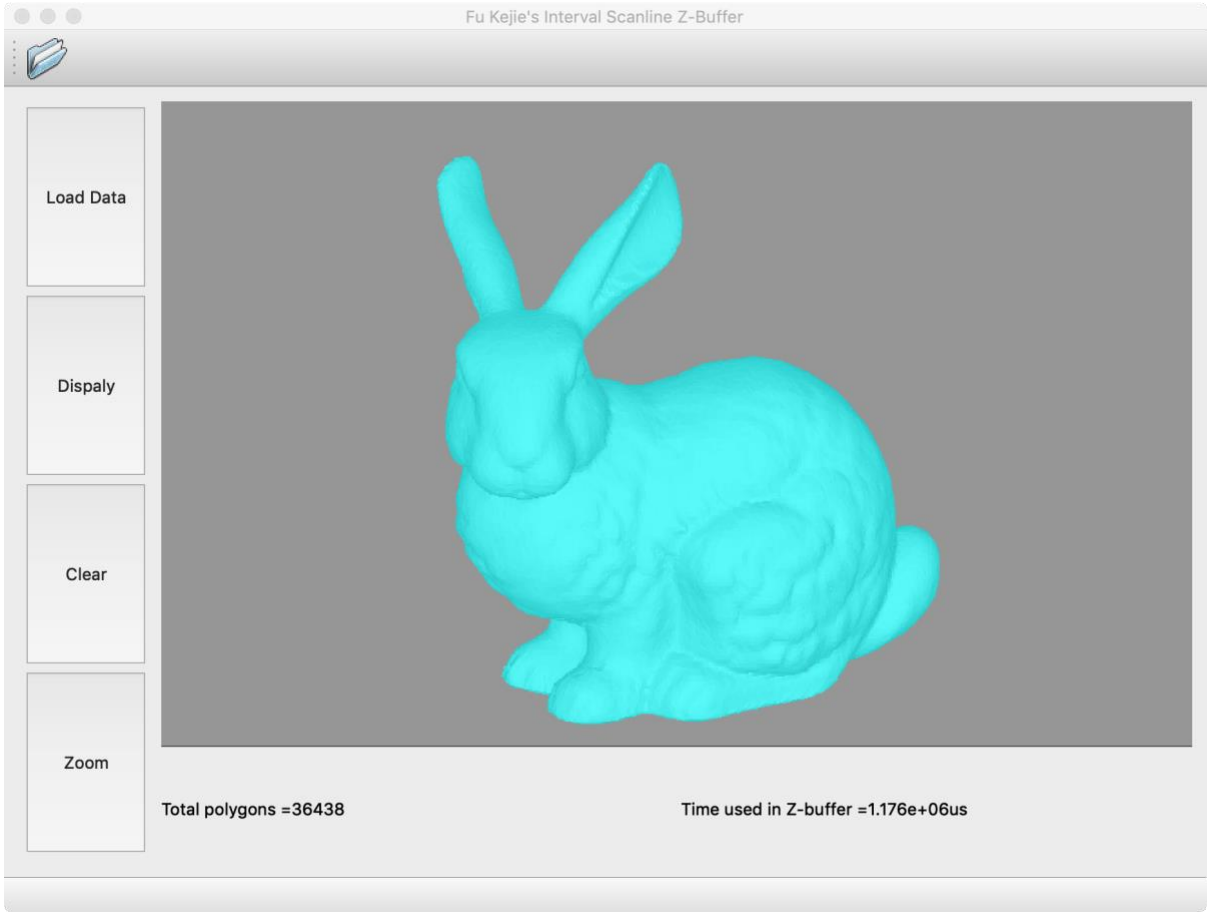


图 1 在 MacOS Mojave 下运行界面

打开压缩文件，主要有如下的 3 个文件，

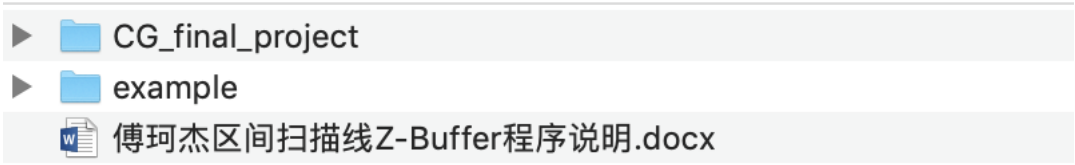


图 2 直接解压后包含的文件

其中 CG_final_project 是程序源码文件，example 是本程序测试所用到的几个例子，最后即是本篇说明文档。

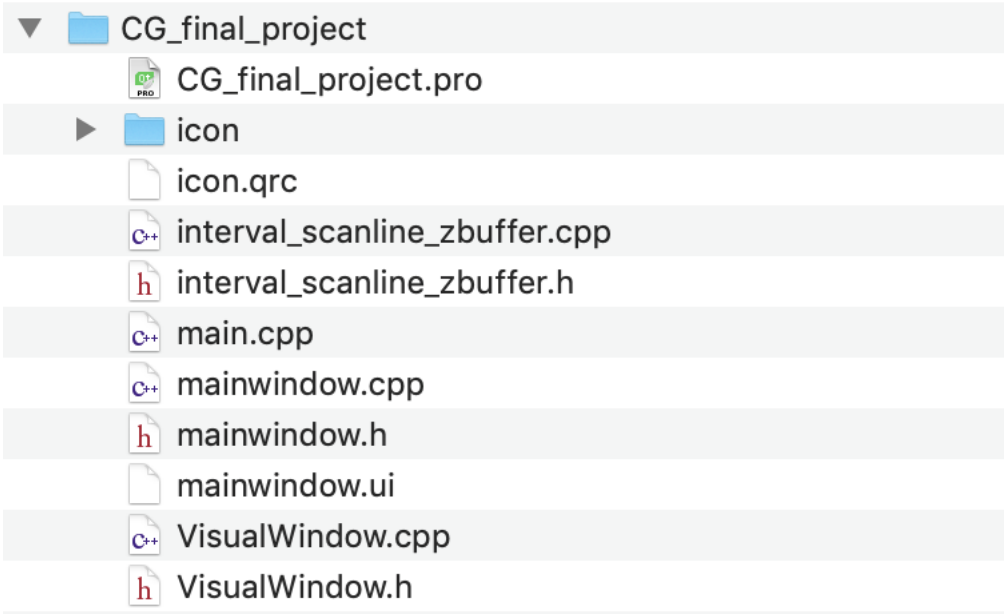


图 3 源码文件的组成

在 QT 开发环境下，直接点击 CG_final_project.pro 即可构建本程序的 QT 的工程（不要有中文路径），除了（.pro）之外，对其他文件做如下说明：

（1）icon 文件夹和 icon.qrc 是和本程序所用到的图标相关的（虽然只用到了一张图）。

（2）mainwindow.ui 是 QT 工程下本程序的 UI 界面配置文件。mainwindow.h 和 mainwindow.cpp 是和 UI 界面操作交互相关的 c++文件。

（3）VisualWindow.h 和 VisualWindow.cpp 是和可是化相关的 c++文件，程序绘图是通通过这里的代码执行的，为了本程序的需要，这个类继承了 QT 的 Widget 类，该基类利益提供一个绘图窗口，可以利用 QT 自带的二维绘图类 QPainter 进行二维绘图。

（4）interval_scanline_zbuffer.h 和 interval_scanline_zbuffer.cpp 是区间扫描线 zbuffer 算法实现 c++源码，里面包含有本程序的核心类 **class interval_scanline_zbuffer**。

本程序可以实现如下功能：

1.可以连续导入多个的 OBJ 文件，利用区间扫面线算法进行消隐显示，为了实现层次感，加了一个光照，然后一起进行可视化；

2.可以对可视化的模型进行旋转、移动、缩放等操作。

3.可以实现 Zoom 操作。

4.可以在当前窗口将所有图形全部擦除，然后重新导入新的 OBJ 文件。

需要说明的是，本程序暂时不支持面贯穿的显示，所以连续导入多个不同的模型，若发生面贯穿，显示效果会比较差。

如果编译存在问题，请联系 kjfu@zju.edu.cn,或者手机 17816862426。

2.数据结构和算法

2.1 数据结构

2.1.1 存储几何数据的结构： 点 ，环 ，面

本程序中点 Vertex 的数据结构是采用 QT 自带的函数库中 QVector3D 类，这个类可以将点的位置存储为向量(x,y,z)的形式，可以实现两个 QVector3D 对象 v0,v1 直接相减，还可以和 QT 自带的矩阵类 QMatrix4X4 相乘，进行坐标变换。

环的数据结构：

一开始为了考虑到可能存在多变形带洞的情形，为了实现欧拉操作，所以就设计了环，但是在后续因为时间原因，只做了读 obj 文件的程序，当成面内的一个存储顶点序列的列表了。

```
class loop
{
public:
    QVector<int> lVertices;//the list of the vertices's id
};
```

面的数据结构：

由于一开始是想做带洞多边形读取的，所以一个面内会有多个环，默认第一个环是外环。

```
class face
{
public:
    QVector<loop> floops;// the first loop is the outer loop (if this face has one more loops)
    int id;
};
```

2.1.2 边项（edgeItem）和边表(edgeTable)

因为我这里对于边表是用 QT 提供的类似于 STL 中 Vector 的 QVector 作为容器存储每个边项，所以原先课上讲的每个边或有一个*next 的指针指向下一个边在我的程序中就没有了。

```
class edgeItem //  edge in the sorted edge table
{
public:

    float x;
    float dx;
    int dy;
    int id;
    //tips:
    //   I use the class Qvector aa container of the edge,so i don't need pointer *next
};

class edgeTable
{
public:

    void initilize(int h);
    void clear();
    QVector<edgeItem> *edgeTableColumn;
    int height=0;

};
```

对于边表类，里面包含一个可以动态初始化边表的高度，清理边表项。

2.1.3 多边形项(polygonItem)和多边形表(polygonTable)

多边形项也是不包含指向想一个多边形项的指针，每个多边形项包含了平面方程系数 abcd 的向量，用 QT 自带的向量类 QVector4D；包含扫描线的个数 dy，多边形的 id，是否进入扫面线的标记 iflag 和进行反射系数 diffuse。

```
class polygonItem // polygon in the sorted polygon table
{
public:
    polygonItem():inflag(0){}
    QVector4D coefficient_abcd;//store the coefficients a b c d in a Qvector4D
    float dy;
    QRgb color;
    int id;
    int inflag;
    float diffuse;
    //tips:
    //   I use the class Qvector as the container of the edge,so i don't need pointer *next
};


class polygonTable
{
public:
    void initilize(int h);
    void clear();
    QVector <polygonItem> *polygonTableColumn;
    int height = 0;
};
```

多边形表也是可以进行动态高度的设置，清理整个多边形表的功能。

2.1.4 活化边列表(activeEdgeList)和活化多形列表（activePolygonList）

```
class activeEdgeList
{
public:
    QVector<edgeItem> edgeList;//contain edgeitem from the edge table
    void sort();// sort edges in the edgelist
};

class activePolygonList
{
public:
    QVector<polygonItem> polygonList;//contain polygonitem from polygon table
    polygonItem &searchPolygon(int id);//find the polygon by its id
    void resetInflag();//reset the inflag of all the polygonitem in the polygonlist
};
```

本程序里面边表和多边形表其中的表项是通过对 QVector(类似 STL vector)对象 edgelist 和 polygonlist push_back（）相应的项。对于活化边表，可以实现对边项按照 x 值由小到大排序，对于多边形表，可以实现根据 id 找到对应多边形并返回其引用，然后还有每次进入到下一条扫面线的时候，将所有活化多边形列表内的多边形项的 inflag 清 0。

2.1.5 区间项（intervalItem）和区间表(intervalTable)

我在这个程序里面定义里一个区间项和区间表，用于存储区间扫描线中间求得的区间，当完整的构建完一个区间表的时候，才开始进行渲染。在最后可视化的时候，再读取这个区间表，在 VisableWindow 这个类内，在 QT 的窗口部件上，根据每个区间项所包含的左右端点，及其属性值，用 QPainter 画出一条线。

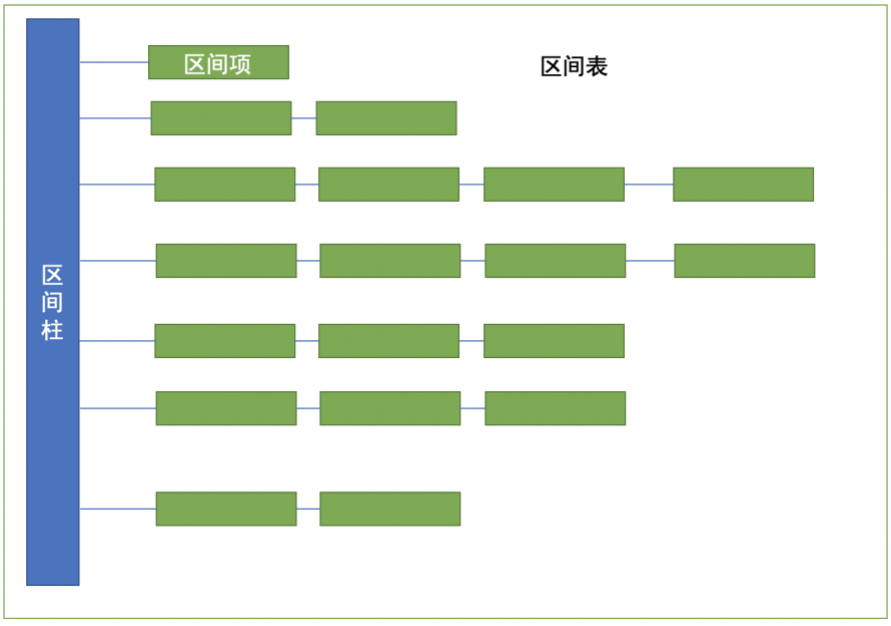


图 4 区间表和区间项的关系

上图是区间项和区间表的关系图，其实和边项和边表，多边形项和多边形表的关系类似。

```
class interval
{
public:
    interval():id(-1){}
    int x_l;//leftside of the interval
    int x_r;//rightside of the interval
    float diffuse;
    int id; //z_l is decieded by polygon id; id = -1 means the background

};

class intervalTable
{
public:
    void initialize(int height);
    void clear();
    QVector<interval> *intervalColumn;// every row in the intervalColumn is a list of intervals

    int height;

};
```

每个区间项包括左右端点的 x 值，以及该区间的反射系数，还有属于哪个多边形，id=-1 表示是属于背景的。

在实际算法实现过程中，还有一个中间的数据结构进入多边形列表（inPolygonList）。用于存储进入多边形表的 id 的，当扫面线进入多边形的时候，把它的 id 加入到这个表中，当扫描线离开多边形的时候，把它的 id 删除。

```
class inPolygonList
{
    void erasePolygonId(int id);
    QVector <int> inPolygonIdList;
};
```

2.1.6 区间扫面线 Z-Buffer 类（interval_scanline_zbuffer）

本程序关于区间扫描线 Z-Buffer 定义为一个类。

```
// interval scanline
class interval_scanline_zbuffer
{
public:

    interval_scanline_zbuffer():vertexIdBase(0), nVertices(0), nFaces(0){}

    //void setWindowSize(int w, int h);
    void initialize(int left, int top,  int width, int height);
    void clear();
    void loadMatrix(QMatrix4x4 &mat);
    void zoom();//move the model in the cencter and fit the window
    void readOBJ(QString filename);
    void readyScan();//Prepare the data structures required by span scanlin z-buffer algorithm
    void goScan();//start span scanline z-buffer: scan lines!

    edgeTable ET;//edges table
    polygonTable PT;// polygon table
    intervalTable IT;// interval table

    QMatrix4x4 transformMatrix;

    QVector<QVector3D> initialVertexArray;
    QVector<QVector3D> zoomedVertexArray;
//    QVector<QVector3D> transformedVertexArray;//the vertex array after model transform firstly and  view transform secondly
    QVector<face> FaceArray;

//private:
    int isZoomed;
    float model_ymax;
    float model_ymin;
    float model_xmax;
    float model_xmin;
    int vertexIdBase;// for read one more models, in this cycle reading file the vertices' ids and faces' ids are based on the last cycle reading fill
    int nVertices ;// the number of vertices
    int nFaces ;//the number of faces, for .obj files the number of loops equals the number of loops
    int windowWidth;
    int windowHeight;
    int top;
    int bottom;
    int left;
    int right;
};
```

其中主要的成员对象是区间表 IT，多边形表 PT，边表 ET；还有存储几何点的列表 initialVertexArray 和后续经过变换后的 zoomVertexArray；存储了面信息的 FaceArray。这个类的主要成员函数有初始化整个 Z-buffer 的 initialize（）函数，清理整个 buffer 的 clear 函数，加载可视化类传过来的转换矩阵的 loadMatrix（），还有用于居中调整的函数 zoom（），最后三个函数是最关键的，首先是 readOBJ（）能够读取 obj 文件并将信息存储在 initialVertexArray 和 FaceArray 中，接下来是 readyScan（）主要是用于生成 PT 和 ET 的，然后是 goScan（）进行区间扫面线扫描，将结构存储在区间表 IT 中。

2.2 基本流程

本程序整体流程如下：

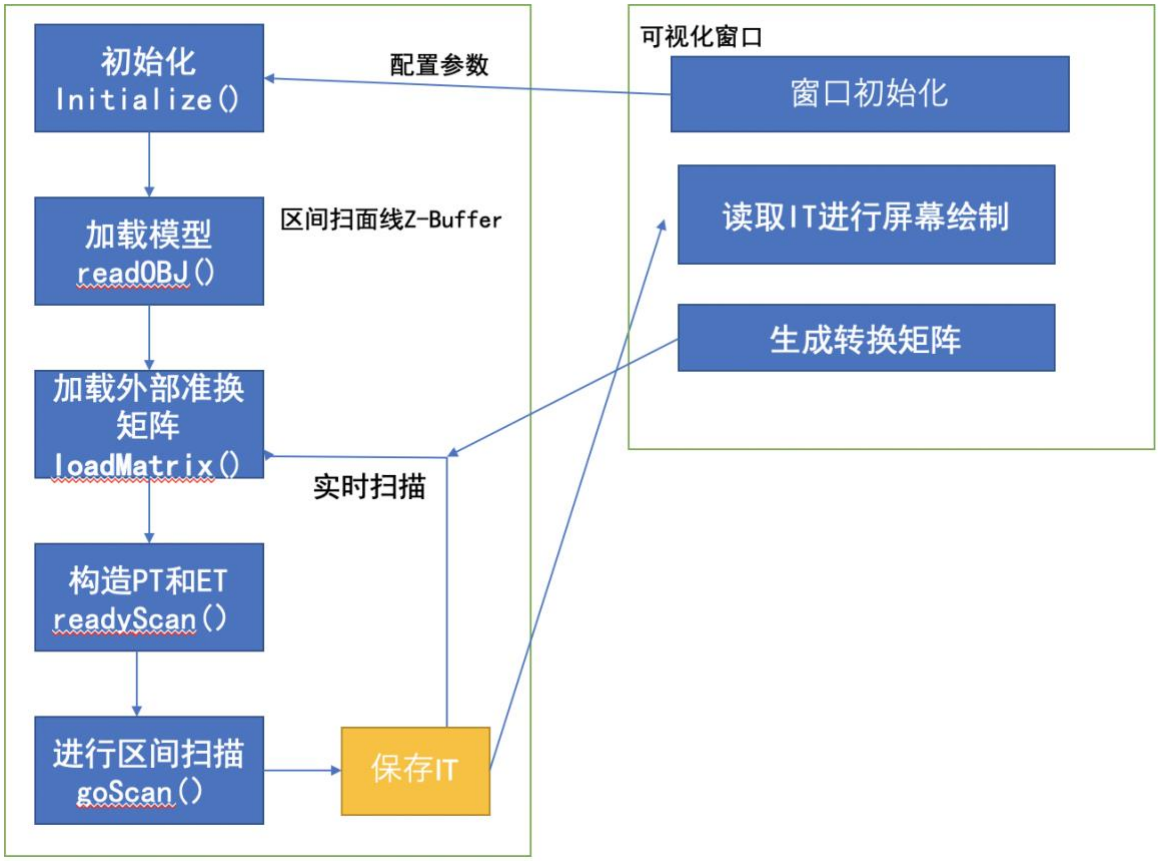


图 5 本程序整体流程

下面对比较关键的 readyScan()和 goScan（）的流程做如下说明：

readyScan():

1. 先对所有面遍历，如果面在窗口外或面在背面，则直接跳过后面的步骤，继续下一个面；
2. 对面上的点进行遍历，选择相邻的两个点，判断时候要加入到边表 ET 中，如果不用，则跳过后面的步骤，如果需要，计算 dx，dy，则生成边项，同时在这里与当前多边形的 ymax 和 ymin 进行比较；
3. 当一个面的所有点遍历完之后，计算这个面的相关量，生成多边形项，加入到 ET 中。

goScan():

1. 清空区间表 IT，生成初始活化边表 AEL 和活化多边形表 APL;
2. 自定向下开始扫描，首先将 ET 中当前高度的所有新的边项加入到 AEL 中，然后对 AEL 中的边项进行排序，将 PT 中所有新的多边形加入到 PEL 中；

3. 自左向右对 AEL 中的边遍历，这里如果里面有 N 条边，我的算法中一开始的一条边要和左边界比较，最后一条边要和右边界比较，然后确定出扫描区间。在这里生成的区间项 push_back 进 IT 中
4. 更新 AEL 和 APL，然后返回第 1 步直到扫描到底部。

2.3 加速算法

本程序做了如下加速：

1.背面剔除

如果一个多边形的法向量中，z 方向的向量为小于 0 则剔除，在具体实现过程，这个多边形就不会加入到多边形表中，在后续进行区间扫面的时候，就不会有这个多边形存在。

2.拒绝在窗口外面的多边形

当多边形整个在窗口外面的时候，就不把这个多边形出去。

3.对于存在边和窗口边界相交的情况

对于一个多边形，如果其边与窗口相交，那么对其边做如下处理，令此条边 dx=0，x=左边界和右边界，但是这种处理只针对如下两种情况，边从窗口外面进入到窗口中不考虑。这样处理之后，在后面区间判读的时候，能很快就把 x=左右边界的边进行判断，不用去求 z 值，也算是一点加速。

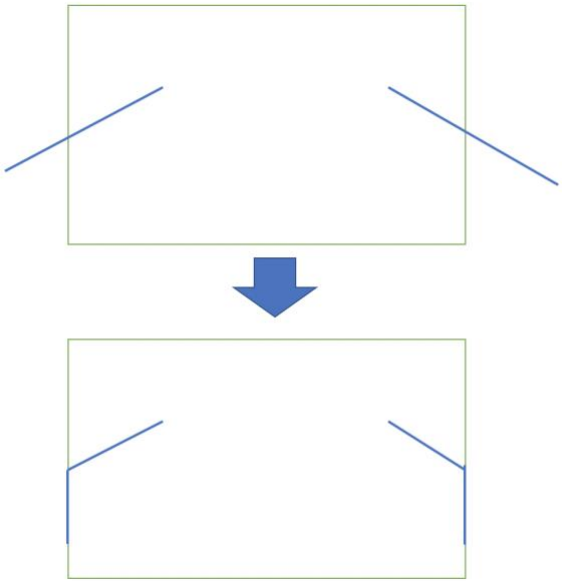


图 6 对于多边形边和窗口相交

3.程序使用及样例测试

3. 1 程序使用

本程序 涉及的按钮比较少，如下图所示，是本程序运行用户点击操作部件：

图 7 本程序运行用户点击的操作按钮

点击文件图形图标和”Load Data”按钮以及最上面的 File 都可以选择打开*。obj 文件。打开一个文件之后，还可以依次再打开更多的 obj 文件。

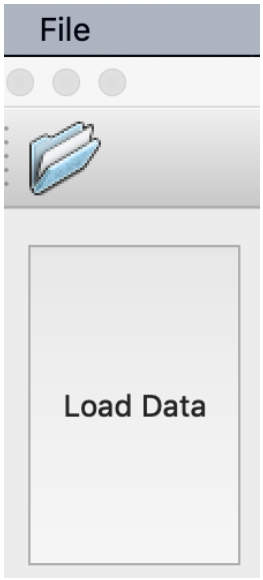


图 8 选择 obj 文件打开

需要注意的是，打开文件之后不会直接显示模型，需要点击一下“Display“按钮才能显示模型。

点击“Clear“按钮会清空当前所有显示的图像，在这之后，可以选择继续导入文件。

点击“Zoom“按钮会将已经平移，放缩，旋转的模型一键居中，并调节缩放到屏幕中央。

另外，缩放是按滚轮；按着鼠标左键移动鼠标，可以对图形跟随鼠标进行旋转；按着鼠标右键移动鼠标，可以实现图形跟随鼠标的移动。

最下面有输出信息：

```
Total polygons =36438                    Time used in Z-buffer =1.169e+06us
```

分别会输出面片数量，和 z_buffer 运行的时间，这里主要测得是开始扫描到完成扫描的时间，不包括加载模型和渲染的时间，如果进行了缩放，平移旋转，想要知道这一过程的时间，可以再点一下“Display”，更新一下时间变化量。

3.2 样例测试

3.2.1 Bunny 图 网格面片数量 36438

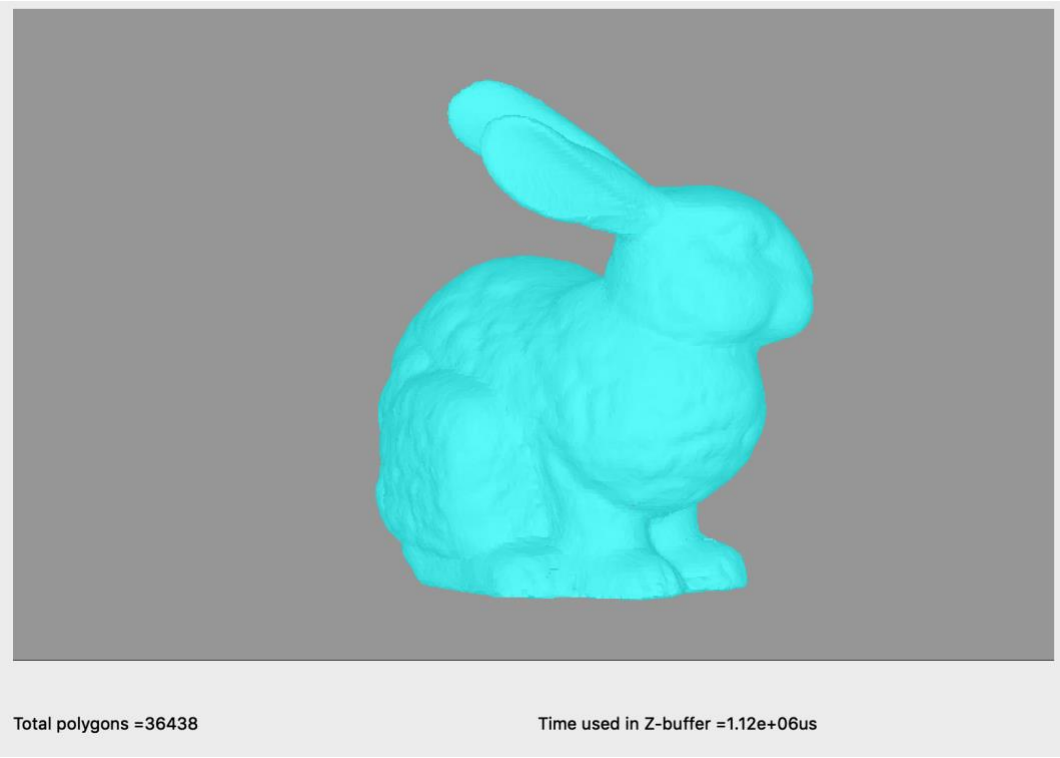


图 9 Bunny 兔显示

可以实现所有缩放，平移，旋转功能个，但是会有延迟。

3.2.2 多模型 鹿+楼房 总面片 8768

图 10 楼房+兔显示

3.2.3 带贯穿情况的模型 花模型 总面片

虽然本程序没有多实现边形贯穿的时候 Z-buffer 算法，但是为了显示一下，如果有贯穿存在，本程序是否会出什么 bug 所以就对这个模型进行了测试。

图 11 花模型显示

在发生多边形贯穿的花瓣区域，本程序的显示如下：

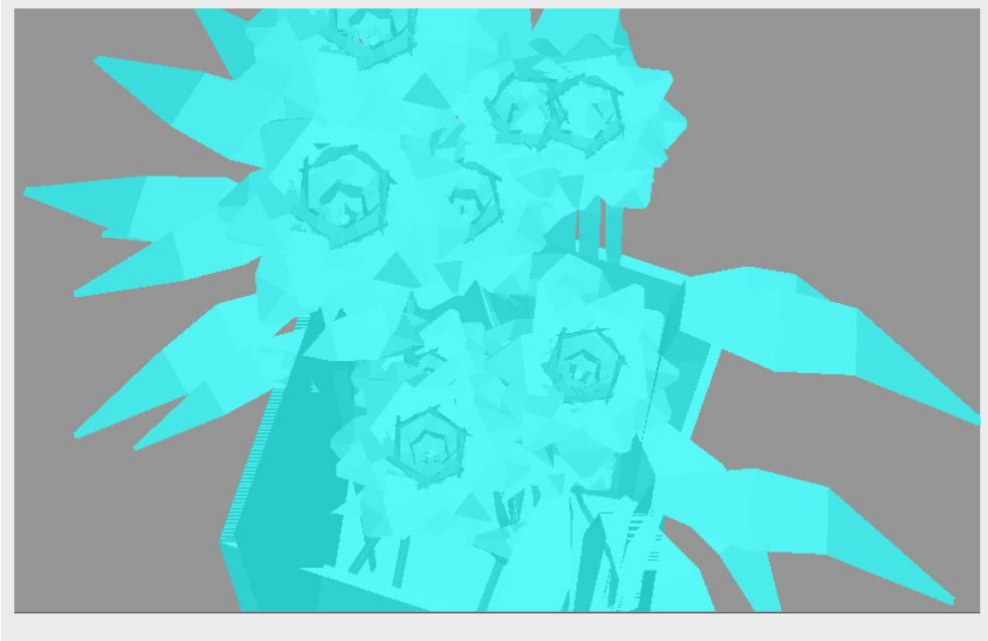
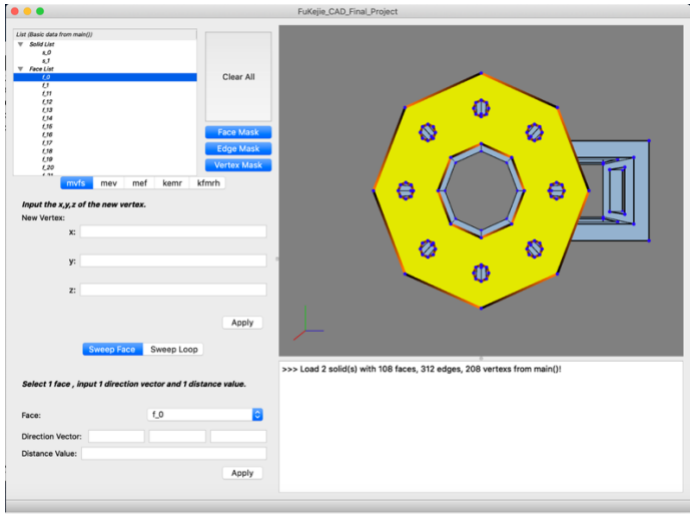


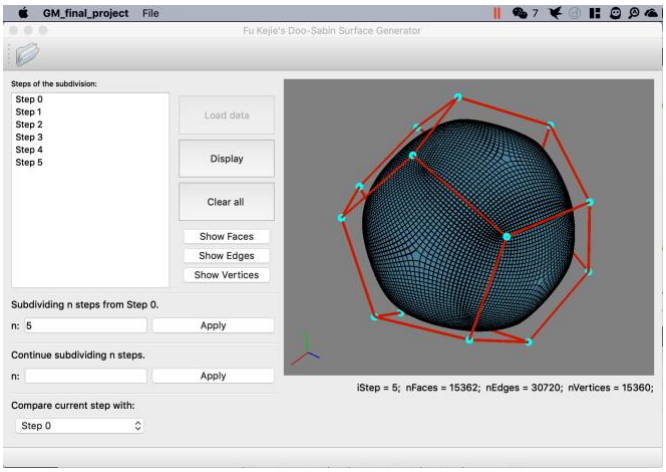
图 12 花模型花瓣处显示效果

4.展望

主要怪自己时间安排的不合理，完成这个作业的时间非常有限，而且自己编程水平和能力比较弱，各方面的因素影响下，导致这个作业最后完成的非常赶，很多一开始打算干的事情都没有机会和精力继续弄。本来还想要把我在高曙明老师《三维 CAD 建模》课上关于欧拉操作的实体建模和在冯老师《应用几何造型基础》细分曲面的生成，拿到这个程序中实现，原先那两个作业都是用 OpenGL 提供的 API 实现。真的非常想把这两次作业都用自己写的这个程序实现，甚至一开始写这个程序的时候存储面的信息想要用的环，以实现对带洞（内环）多边形的实现。



（a）欧拉操作实体建模



（b）Doo-Sabin 细分曲面生成器

图 13 其他课上用 OpenGL 实现的内容很遗憾没能用自己写的 Zbuffer 实现

希望以后能够更加合理的规划自己的时间，完成作业的时候不要留下太大的遗憾。