说明文档

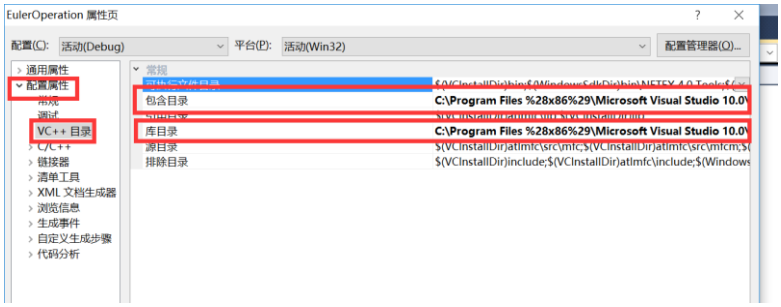
姓名：Call 偶围城 学号：XXXXXXXX

本程序实现了区间扫描线 ZBuffer 算法，充分利用了扫描线内区间的连贯性、多条扫描线之间的连贯性、深度的连贯性。本说明文档分为五个部分：编译环境的介绍、用户界面的介绍、数据结构的介绍、在实现区间扫描线 ZBuffer 算法过程中在基本算法思想之外的加速优化处理以及结果的展示。

一、编译环境

我使用的 IDE 是 VS2010，系统是 win10 64 位 4G 内存的虚拟机。在 VS 中我配置了 OpenGL，需要的配置文件在压缩包内，具体配置方法如下：

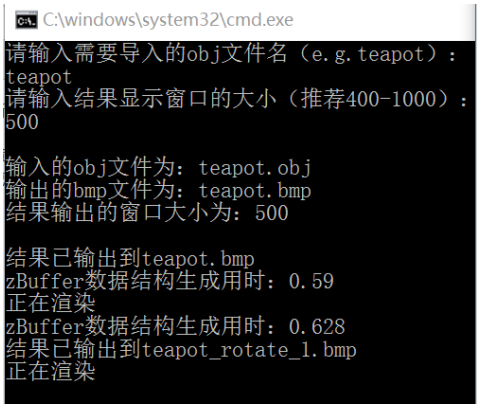
1. 将 glut.lib 和 glut32.lib 这两个静态函数库复制到文件目录的 lib 文件夹下 X:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\lib；
2. 将 glut.dll 和 glut32.dll 这两个动态库文件放到操作系统目录下面的 X:\Windows\system32 文件夹内（32 位系统）或X:\Windows\SysWOW64(64 位系统）；
3. 将解压得到的头文件 glut.h 复制到 X:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\include\GL，如果在 incluce 目录下没有 GL 文件夹，则需要手动创建；
4. 打开项目属性，设置包含目录和库目录，分别添加 gl 文件夹和 lib 文件夹所在的目录。

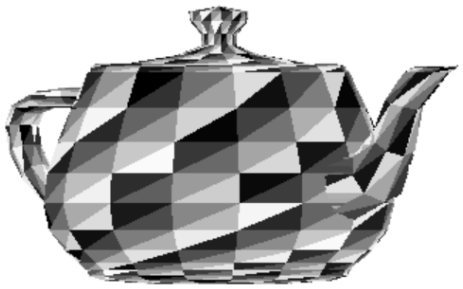


二、用户界面

用户界面如图所示，程序会先提示输入需要导入的 obj 文件名，这里不需要输入文件后缀。如果用户打开当前显示结果保存到 bmp 文件中的功能，所有的

bmp 文件也会基于此输入命名（e.g. teapot.bmp, teapot_rotate_1.bmp）。





由于输出文件需要占用一定比例的时间，有点影响旋转显示的连贯性，所以我在提交的源代码中注释了输出文件的调用部分，如果用户想保存每次旋转的结果可以在 main.cpp 和 CResult.cpp 中分别取消注释调用语句即可：

```
/**
    func：将显示结果写入bmp文件
    由于写入文件需要耗时，影响鼠标拖拽旋转物体的观察效果的连续性
    所以我这里注释了，如果需要保存物体当前状态的二维图片可以放开
*/
// result.WriteFile(path);
```

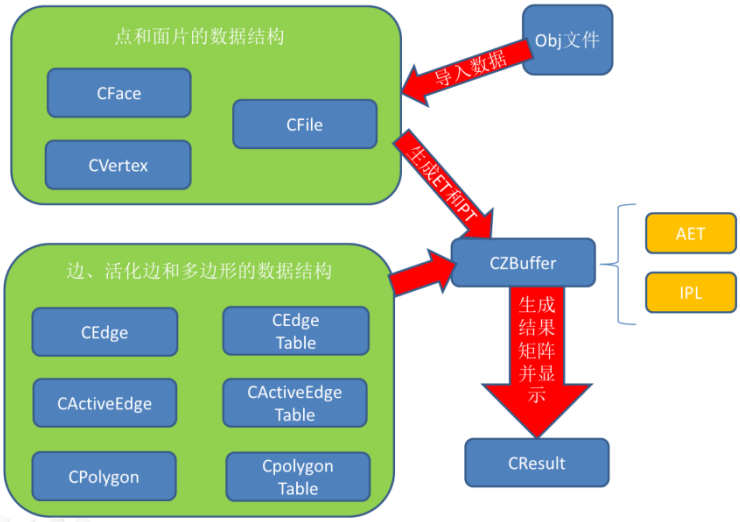
文件名输入后程序会提示输入结果显示窗口的大小，这个大小是最后 OpenGL 画的区域的边长，也是输出文件的大小。程序会根据用户输入的这个大小对 obj 文件里的坐标点做相应的缩放变换，保证显示的物体大小适应窗口大小并且显示在窗口中间。

接下来程序会提示用户输入文件的完整名称，输出文件的完整名称，旋转结果输出到了哪个文件里，区间扫描线 zbuffer 算法生成最终结果矩阵用了多长时间等。

三、数据结构

先简单的介绍一下，我的程序一共有 11 个类。最开始先读入 obj 文件生成 CFile 实例，CFile 里有两个 vector，分别存着 CFace 面片信息和 CVertex 顶点信息。之后通过顶点和面片信息初始化 CZBuffer 实例，构造 ET 和 PT，ET 和 PT 的数据结构分别是 CEdge 和 CPolyEdge，CActiveEdge 后来被弃用了，活化边的数据结构使用的依然是 CEdge。CZBuffer 通过扫描构建自己的成员变量 AET 和 IPL，其中活化边表和活化多边形表用的是 CEdgeTable 和 CPolyTable。使用 CZBuffer 中的 AET

和 IPL 初始化 CResult，生成结果矩阵，可以输出 bmp 文件也可以 OpenGL 渲染显示。用户拖动鼠标旋转物体，即对初始顶点旋转变换后再经过上述操作生成结果矩阵，最终再显示。



每个类的详细数据结构在文档中不容易叙述清楚，我以截图的形式保存在一个文件夹中，截图里面有详细的注释。其中源代码 CFile 有两个文件，原版和效果优化版，优化版是对旋转变换后进行了附加处理，效果会比原版好很多，但是非常耗时，如 1000*1000 的处理会需要多耗费将近 1 秒的时间，如果想看比较好的效果请在项目中使用 CFile 效果优化版.cpp（在项目中文件名要改回 CFile.cpp）。

四、加速优化

我的运行硬件环境是内存 4G 64 位的 win10 虚拟机，扫描大小为 500*500 时生成最终结果矩阵大约需要 0.5、06 秒左右，大小为 1000*1000 时生成最终结果矩阵大约需要 1.0-1.4 秒，有时可以降至 1 秒以内，其中我做了一些加速的处理。

```
请输入需要导入的obj文件名（e.g.teapot）：
teapot
请输入结果显示窗口的大小（推荐400-1000）：
500

输入的obj文件为：teapot.obj
输出的bmp文件为：teapot.bmp
结果输出的窗口大小为：500

zBuffer数据结构生成用时：0.575
正在渲染
zBuffer数据结构生成用时：0.506
正在渲染
zBuffer数据结构生成用时：0.5
正在渲染
zBuffer数据结构生成用时：0.679
正在渲染
zBuffer数据结构生成用时：0.506
正在渲染
zBuffer数据结构生成用时：0.733
正在渲染
zBuffer数据结构生成用时：0.461
正在渲染
```

```
请输入需要导入的obj文件名（e.g.teapot）：
teapot
请输入结果显示窗口的大小（推荐400-1000）：
1000

输入的obj文件为：teapot.obj
输出的bmp文件为：teapot.bmp
结果输出的窗口大小为：1000

zBuffer数据结构生成用时：1.388
正在渲染
zBuffer数据结构生成用时：1.209
正在渲染
zBuffer数据结构生成用时：1.093
正在渲染
zBuffer数据结构生成用时：1.084
正在渲染
zBuffer数据结构生成用时：1.098
正在渲染
zBuffer数据结构生成用时：1.133
正在渲染
zBuffer数据结构生成用时：1.125
正在渲染
zBuffer数据结构生成用时：1.072
正在渲染
```

首先，我在所有实例（点、三角形面片、边、多边形）构建时都进行了序列化的编号，开始我打算都写成链表形式，但是遍历耗费不少时间，所以后来我将链表与 vector 相结合使用，信息都存储在 vector 中，用编号进行直接访问，不需要查找。活化边表和活化多边形表使用的是链表，因为边和多边形的顺序不影响扫描，而且用链表插入和删除比较方便。

其次，我对边表 ET 数据的结构进行了修改，将边的最大 y 坐标 ymax 变为了剩余扫描线的数量 dy，原本 ymax 的信息保留在 map EdgeList 的索引里，即扫描线在 y 的位置时把 EdgeList 的第 y 个 vector 中的所有边加入活化边表即可。相应的，在构建 EdgeList 时要按照边的最大 y 坐标把边加入到 map 的相应位置。在扫描时，每次遍历活化边表把 dy 都-1，当 dy<=0 时就把该边移出活化边表。

然后，在边表 ET 的基础上，我设置了活化边表。本来想为活化边表单独写一个类 CActiveEdge，写完了之后发现 CEdge 已经满足需求了，虽然最后没有单独的活化边表数据结构，但是活化边表的思想我保留了下来。每次扫描时把新涉及的边加入，只对活化边表里的边进行扫描，并利用边的连续性算出交点（x+dx），删除也只遍历活化边表，相比于遍历整个边表 ET 计算交点和移除扫描完的边效率要高很多。

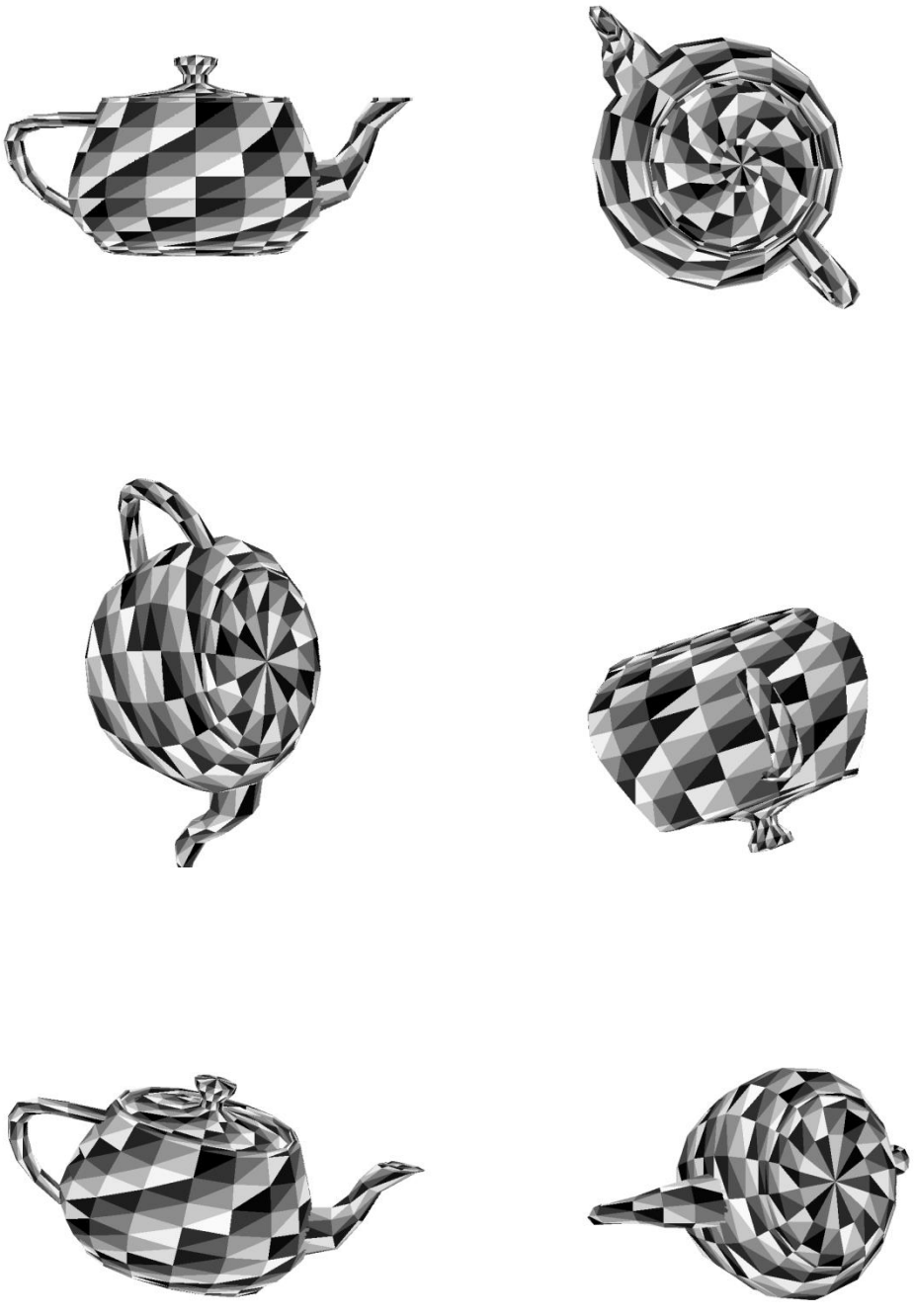
同时，活化多边形表也是我借鉴了扫描线 ZBuffer 算法的思想，在扫描线与活化边计算交点时将每个交点对应的多边形序号保留下来，对排序过的交点依次扫描，扫描到每个交点的时候就将该点对应的多边形 flag 取反，即利用奇数交点进入偶数交点离开的原则设置多边形与区域的关系。当判断一个多边形是否包含某一区域时，只要遍历活化多边形表即可，无需遍历整个多边形表 PT。

最后，在读入文件和旋转变换后我都计算了 maxx, minx, maxy, miny，即物体坐标的最值，在后续的处理中也有 sizex, sizey 等属性，在扫描边表、AET 和 IPL 生成结果矩阵等情况下，可以减少相当比例的循环次数，因为物体周围的空白区域都可以不进行处理。

五、结果展示

我在“teapot 结果图”文件夹中保存了 1 张初始效果图和 27 张旋转效果图，这里为了 bmp 文件的精度使用的是 1000*1000 的大小，推荐使用 CFile 效果优化版.cpp（压缩包工程里使用的就是这个），效果显示大小选择 500*500，这样效果和速度比较兼得，在视觉效果下和 1000*1000 的结果几乎没有区别，时间只慢

0.1-0.2 秒左右。以下我展示了初始效果以及随机选取了五张旋转图展示：



在文档的最后我要感谢冯老师一学期以来的指导，在冯老师的计算机图形学课程中我学到了很多知识，同时也开拓了自己的视野。