（本程序由浙江大学 13 级学生郑天季编写，学号 21721201）

程序的编程环境

visual studio 2017 community 版本，使用 C++编写

用户使用说明

直接在 main 函数里面修改 ori_model.ReadModelFromFile("2.obj");中的 obj 可以替换模型文件，编译运行以后是一个随着时间不断旋转的模型

数据结构说明

文件代码说明：

程序不依赖除了 C++标准库和 windows.h 以外的所有第三方库文件，从矩阵运算，模型读取到渲染器全部是使用自己编写的代码逻辑，没有使用 dx 和 opengl 以及对应的 shader 文件。即全部使用 CPU 渲染

代码分为四个部分，分别为 transform 文件，myMath 文件，DrawElement 文件以及 main 文件。

Transform 文件：定义世界坐标变换矩阵，实缴坐标变换矩阵，投影变换矩阵等信息，定义并实现从局部坐标系到屏幕坐标系的转换。

MyMath 文件：定义自己编写的数学操作，包括对其次向量，4 阶矩阵，顶点信息，面的信息，obj 模型的信息以及相关操作的定义和实现。
其中可能比较不好理解的是 Face 结构和 Model 结构：

```
class Face {
public:
    Face();
    ~Face();
    vector<point> pointVector;
    double A = 0, B = 0, C = 0, D = 0;
    double r, g, b;
    bool in_out_flag;
    double nowZ;
    int id;

};
```

Face 结构中，包括所有顶点的信息，包括面的 ABCD 参数值，包括面的 RGB 信息

（一个面公用一个面颜色），包括在扫描线中这个面是否是 in 还是 out，包括这个面的 id，这个面的 nowZ 用于在某一条扫描线中计算当时的 z 值

```
class Model {
public:
    void ReadModelFromFile(const char* filename);
    Model();
    ~Model();
    void initModel();
    vector<Face> faceVector;
    vector<point> pointVector;
    vector<vect> normalVector;
    int numNormals;
    int numFaces;
    int numVertex;
};
```

在 Model 结构中，包含一个从文件里面读模型的方法，模型包含所有点的 vector 数据结构，包含一个有所有点的法向的数据结构，还有一个包含所有面的数据结构。最后有三个记录所有点的数量，所有面的数量和所有法向的数量。

DrawElement 文件：定义区间扫描线以及相关画图的数据结构，包括活化边表：

```
class ActivateEdge{
public:
    double x, dx;
    int id, ymax;// 属于哪一个面
};
```

其中 x 指的是这个边当前的 x 坐标值，dx 指的是和下一次 y 扫描线的 x 差值，id 指的是这个边所对应的多边形的 id，ymax 指的是这个边的最大 y 值

DrawElement 文件还包括扫描线结构：

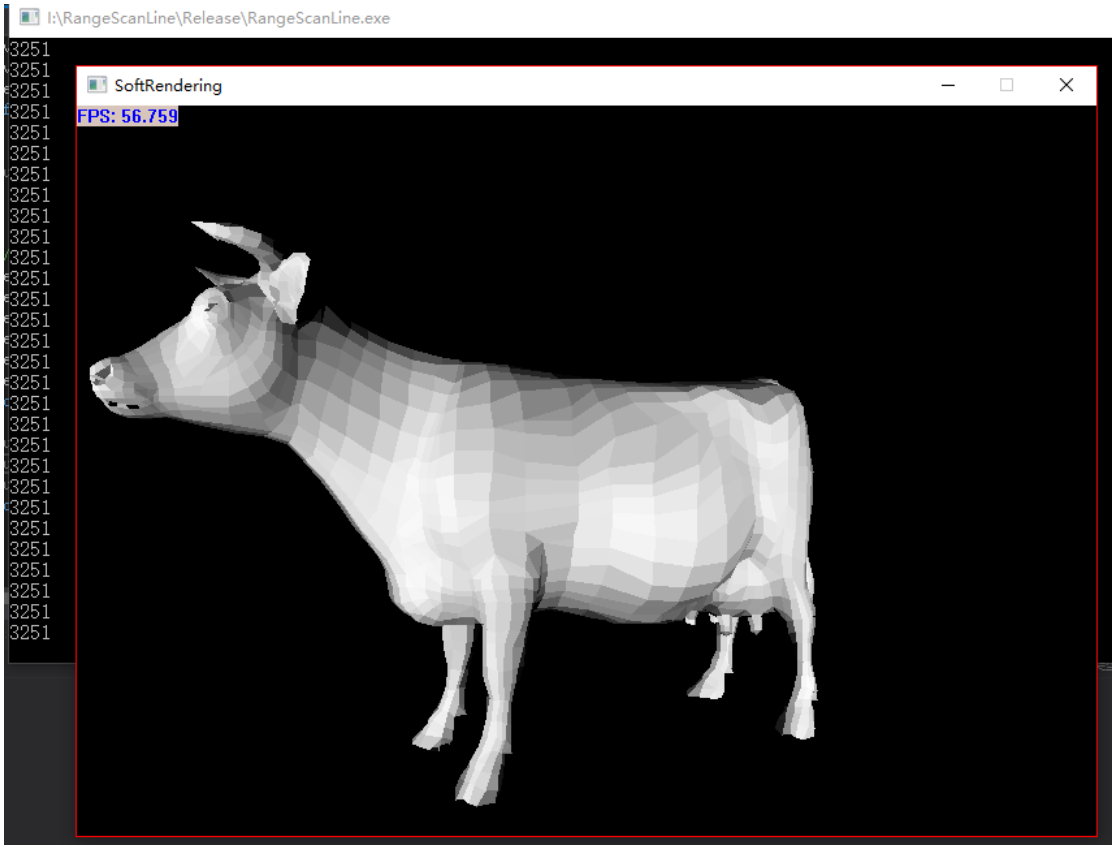
```
class Scanline {
public:
    Scanline();
    ~Scanline();
    list<ActivateEdge> edgeList;
};
```

一条扫描线只有一个活化边表的成员变量，对于活化多边形表，则定义在了 drawScanLine 方法内部。

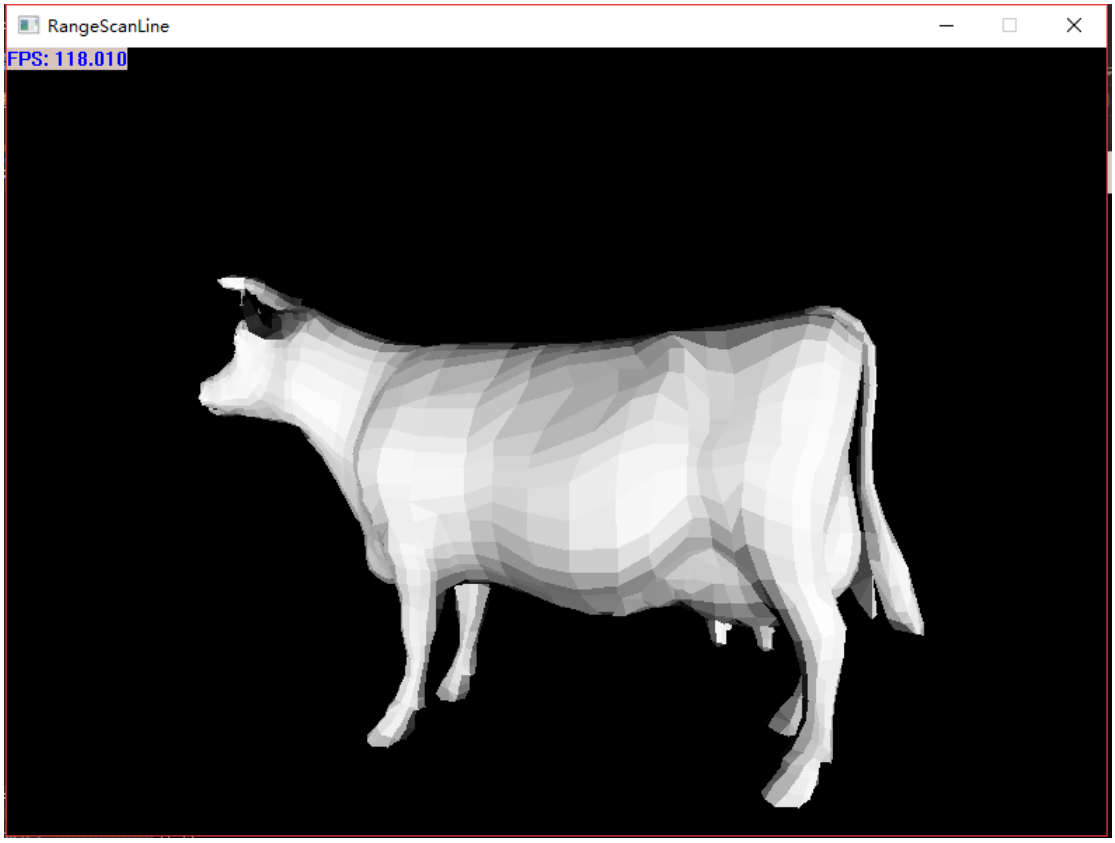
DrawElement 文件还包括整个渲染所需要的长远变量，包括光照，摄像机位置等等信息，都定义在了 DrawElement 类当中。

加速算法的使用：

效果：对于没有使用加速算法的 cow.obj 文件（3251 个面片），我们运行帧数为 56.7 帧（事实上因为之前忘记截图了，这个 56 帧是已经做了背面裁剪，pow 算法改进，sqrt 算法改进还有消除动态内存分配等的效果，一开始是只有 10 帧左右的。。）



对于使用了加速算法的代码，同样的牛，我们运行帧率为 118 帧



优化算法的方式：

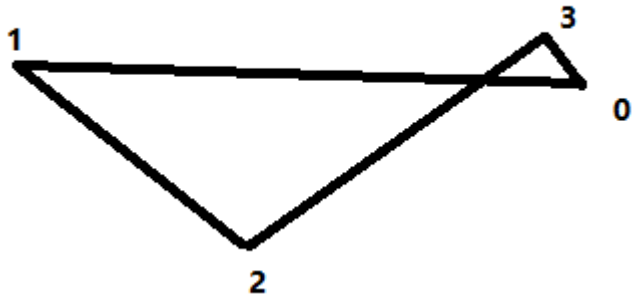
1， 增加背面剔除，同时也增加了摄像机外的 cvv 裁剪
2， 通过 visual Studio 的性能查看器，发现了区间扫描线算法的瓶颈，修改了扫描线的内部实现方式，调整了一下顺序和逻辑
3， 消除所有的 new 操作，改为使用引用的方式访问地址，同时使用了 google 的 libtcmalloc_minimal 动态优化库减少标准库里面的 new 应用
4， 尽量减少除法操作
5， 对于光照的法向计算，我发现了 pow 和 sqrt 函数的消耗十分的高，所以 sqrt 算法使用了 o（1）复杂度的 magic number 求 sqrt 倒数的算法，pow 则是改为了乘法
6， 本来是想用 openMP 对程序进行并行计算优化的，结果发现我们的区间扫描线可能并不支持并行计算，所以我觉得如果能够优化成可以并行计算的方式的话，效率会高很多

对 obj 数据的要求

因为读 obj 文件是自己写的，所以并不像其他库一样支持很多功能。首先这个程序支持多边形以及凹多边形，但是在计算法向的时候凹多边形可能不是特别容易计算，所以要求输入数据是具有法向数据的，如果没有法向数据的话会出问题。

另外因为支持多边形操作，为了保证线条的一致，所以要求模型面的顶点是

按照顺序来的，测试的时候发现有些 obj 模型的数据没有按照顺序来，例如下面这种情况：



那么我就会按照 0—1—2—3 的顺序连接，就会出现问题。解决方法当然是有的，例如将所有的多边形拆分成三角形，例如上面这个拆分成 0-1-2 和 0-2-3 两个三角形，但是如果这样做的话就不支持多边形和凹多边形了，并不能体现算法的特点，所以我选择了这种方式。最后附一张非光照的效果图：