****

**课程名称 计算机图形学**

**姓名**

**学号**

**所在学院**

# 一、算法说明

我的扫描线Z缓冲算法与课件中的算法（下称原算法）在数据结构上有所不同。

首先，原算法中的活化多边形表与分类多边形表结构完全相同，存在空间上的冗余。而且它并非必须，我们完全可以通过活化边表中的多边形ID来直接访问分类多边形表，获取相关信息，不需要检索活化多边形表。故舍弃活化多边形表。

其次，原算法中使用了分类边表，然而PPT中没有提到过该边表的作用，我的理解是用来在新多边形激活的时候，以及活化边表中的边扫描结束的时候查找新的边对时所用。然而当数据量达到巨大规模时，每一个y值对应的边数会很巨大，通过分类边表找边消耗的时间会很多，然而实际上大部分时间都消耗在了遍历不相关的多边形的边上。又考虑到对于常见的obj格式的数据而言，每一个多边形的边数往往不多（通常为3），那么我们可以直接遍历该多边形的边来查找边对，减少冗余。故舍弃分类边表。

修改后的分类多边形表如图1所示：

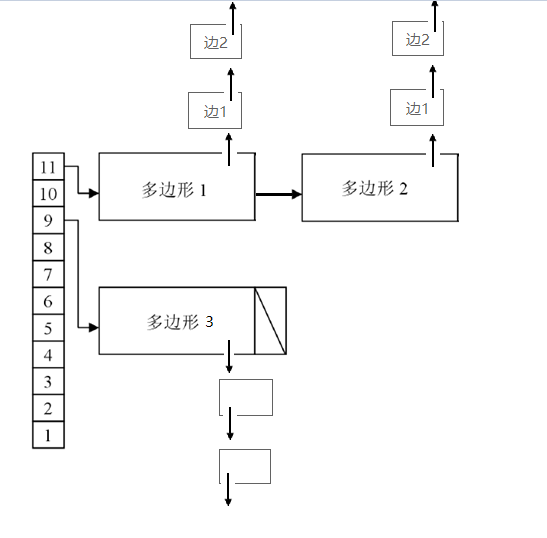


图1 分类多边形表

数据结构代码如下：

**struct Edge{**

**double hx,hy,lx,ly,dx;**

**int dy;**

**struct Edge\* next;**

**};**

**//多边形结构体**

**typedef struct Poly{**

**double a, b, c, d;**

**int id,dy;**

**double ly;**

**struct Color color;**

**struct Poly\* next;**

**struct Edge\* edge;**

**}Poly;**

**Poly polygons[MAX\_FACE];**

**//分类多边形表**

**Poly\* polylist[HEIGHT];**

**//活化边节点**

**struct AENode{**

**double xl, dxl, dy, xr, dxr, zl, dzx, dzy;**

**int id;**

**struct AENode\* next;**

**};**

**struct AENode\* AEhead; //活化边表**

核心算法如下所示是一个循环，每一趟代表一根扫描线。对于每一趟，共有三大步骤，各是一个循环。

第一步是遍历分类多边形表中高度为i的那个链表，找出边对加入到活化边表中。

第二步是将活化边表中已经用完的边对去除，加入新的边对（如果该多边形还没处理完的话）。

第三步是对活化边表进行遍历，比较各个区间的深度，设定颜色。

核心算法代码如下：

**//扫描线Z缓冲算法**

**void lineZbuffer(){**

**for (int i = HEIGHT - 1; i >= 0; i--)**

**{**

**//add new polygon from polylist to 活化边表(AE)**

**for (struct Poly\* p = polylist[i]; p != NULL;)**

**{**

**if (p->dy > 0){**

**AENode\* ae = (AENode\*)malloc(sizeof(struct AENode));**

**if (!findEdgePair(i, p, ae)){**

**//比如多边形最高点刚好是i的时候会找不到一对边，因为扫描线与两边的交点重合**

**delete ae;**

**if (i > 0){**

**//将它移到polylist的下一级中**

**p->dy--;**

**struct Poly\* temp = p->next;**

**p->next = polylist[i - 1];**

**polylist[i - 1] = p;**

**p = temp;**

**continue;**

**}**

**}**

**else{**

**//加入活化边表**

**ae->next = AEhead;**

**AEhead = ae;**

**}**

**}**

**p = p->next;**

**}**

**//处理活化边表中已经用完的边对（AENode）**

**for (struct AENode\*\* p = &AEhead; (\*p) != NULL;)**

**{**

**if ((\*p)->dy <= 0)**

**{**

**int id = (\*p)->id;**

**int ly = polygons[id].ly;**

**if (i <= ly || !findEdgePair(i, &polygons[id], (\*p)) ){ //如果多边形已经用完 或者 找不到一对边（实际未发现该情况出现）**

**//删除边对**

**struct AENode\* temp = (\*p);**

**(\*p) = (\*p)->next;**

**delete temp;**

**continue;**

**}**

**}**

**p = &(\*p)->next;**

**}**

**//初始化深度**

**for (int j = 0; j < WIDTH; j++)**

**line\_z[j] = INT\_MIN;**

**//扫描活化边表中的各边对**

**for (struct AENode\* p = AEhead; p != NULL; p = p->next)**

**{**

**double xl = p->xl, xr = p->xr, zl = p->zl;**

**int x = (int)xl + 1; //找出xl右边第一个像素点**

**if (x < 0) x = 0;**

**if (x >= WIDTH) continue;**

**if (xr > WIDTH) xr = WIDTH;**

**double z = zl + (x - xl)\*p->dzx;**

**//扫描区间**

**while (x < xr)**

**{**

**if (z > line\_z[x]){**

**line\_z[x] = z;**

**setColor(x,i, p->id);**

**}**

**z += p->dzx;**

**x++;**

**}**

**//更新信息，为下一次做准备**

**p->dy--;**

**p->zl += p->dzy;**

**p->xl += p->dxl;**

**p->xr += p->dxr;**

**}**

**}**

**}**

除了核心算法部分，程序还包括读取obj文件，缩放模型坐标，构造多边形表和活化边表，OpenGL绘图等内容，在此不一一赘述。

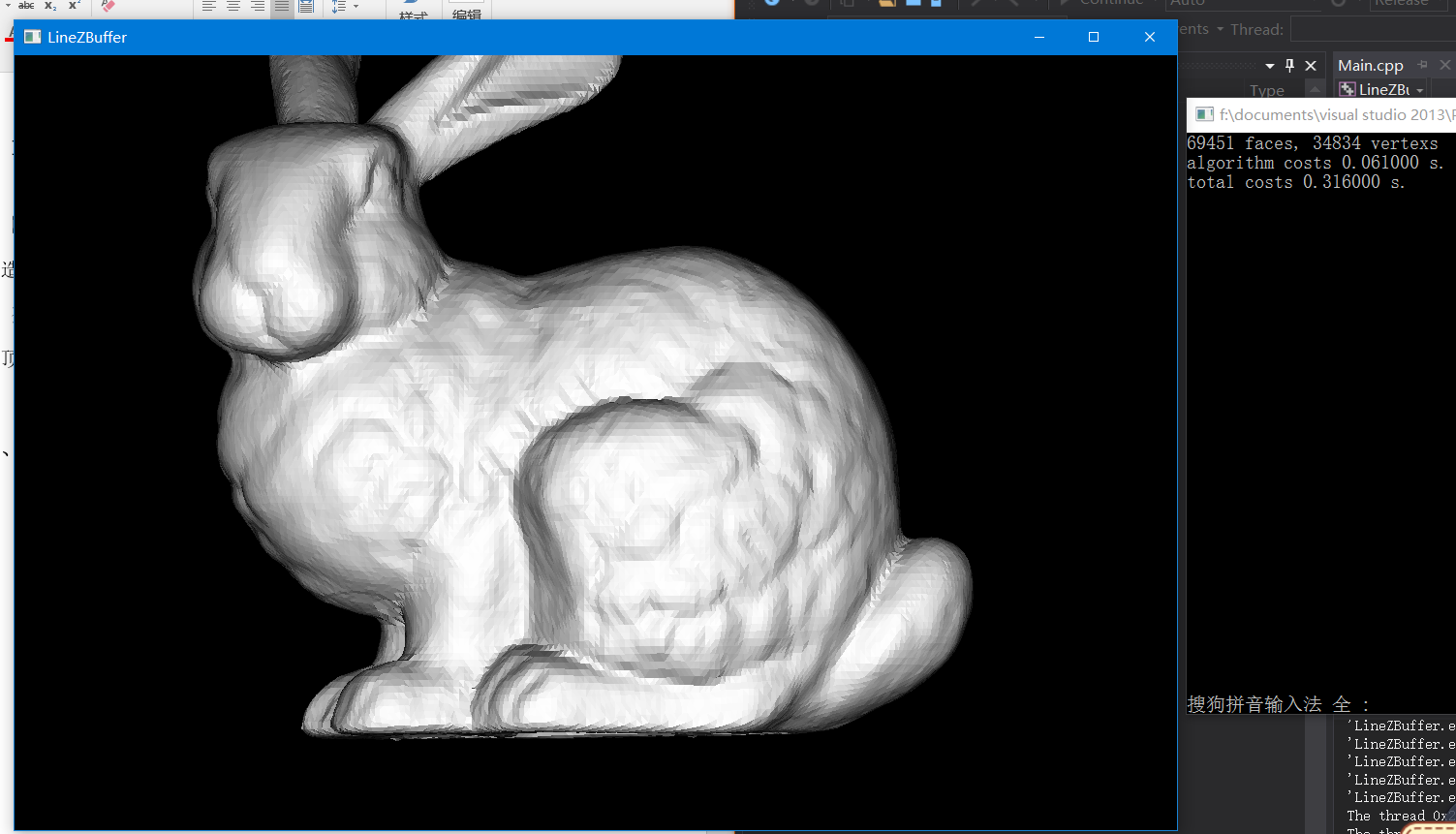
整个代码算上空行和注释共400行，处理70000个面，35000万个顶点时算法耗时约0.06秒（i7-4710HQ），整体耗时约0.30秒。

对于多边形的颜色设置，共有两种，一是彩色模式，主要用于对少量几个多边形测试算法的正确性，如最后一张图所示。另一种是灰度模式，假设有平行光从z轴正方向射向负方向，根据多边形的法向量与Z轴的夹角来设置灰度，夹角越小，颜色越亮。

# 二、运行结果展示

运行环境为Windows 10 + Visual Studio 2013 + OpenGL

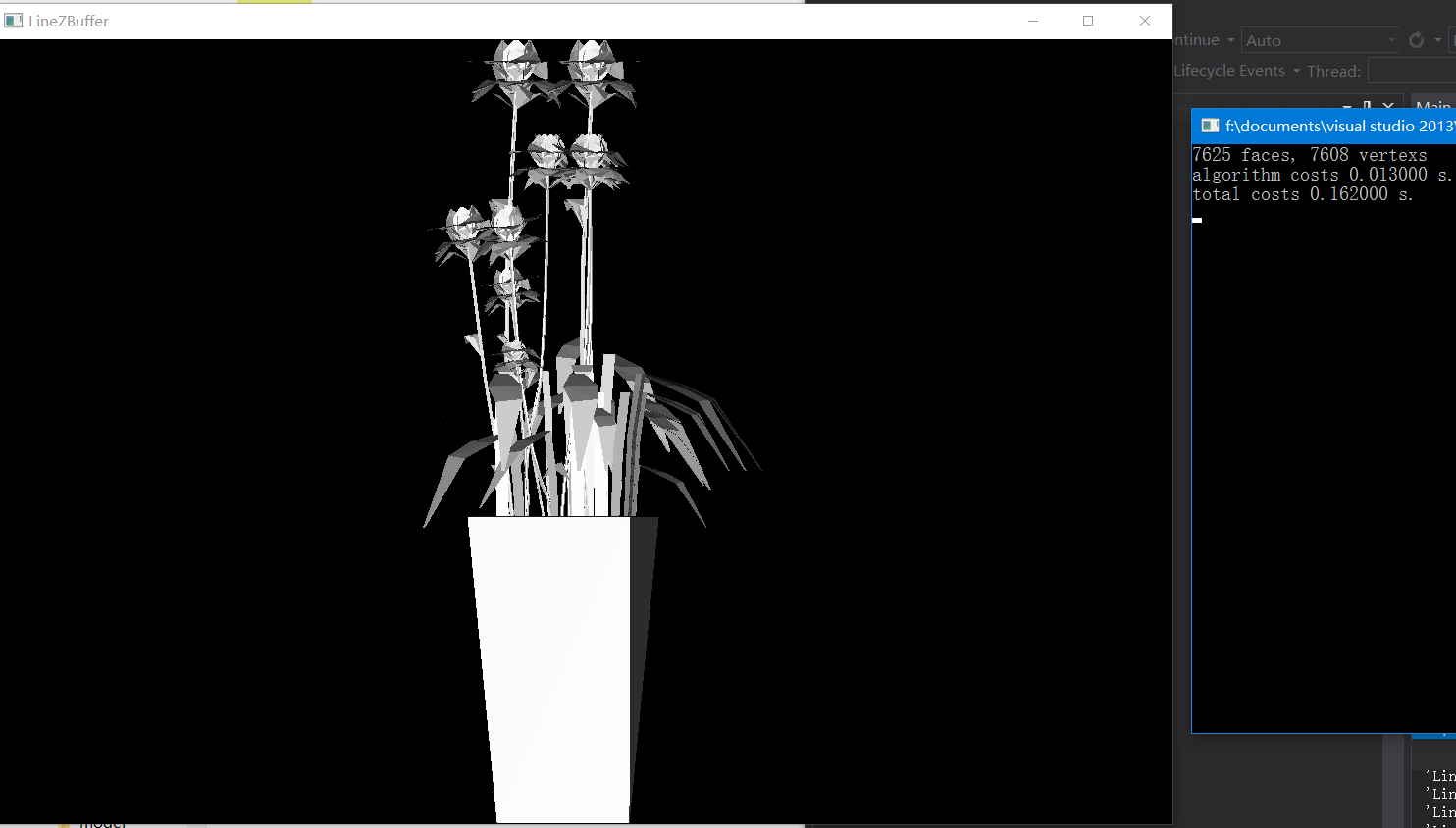
CPU i7-4710HQ 2.5GHz

1．兔子模型

69451 faces, 34834 vertexs

algorithm costs 0.061000 s.

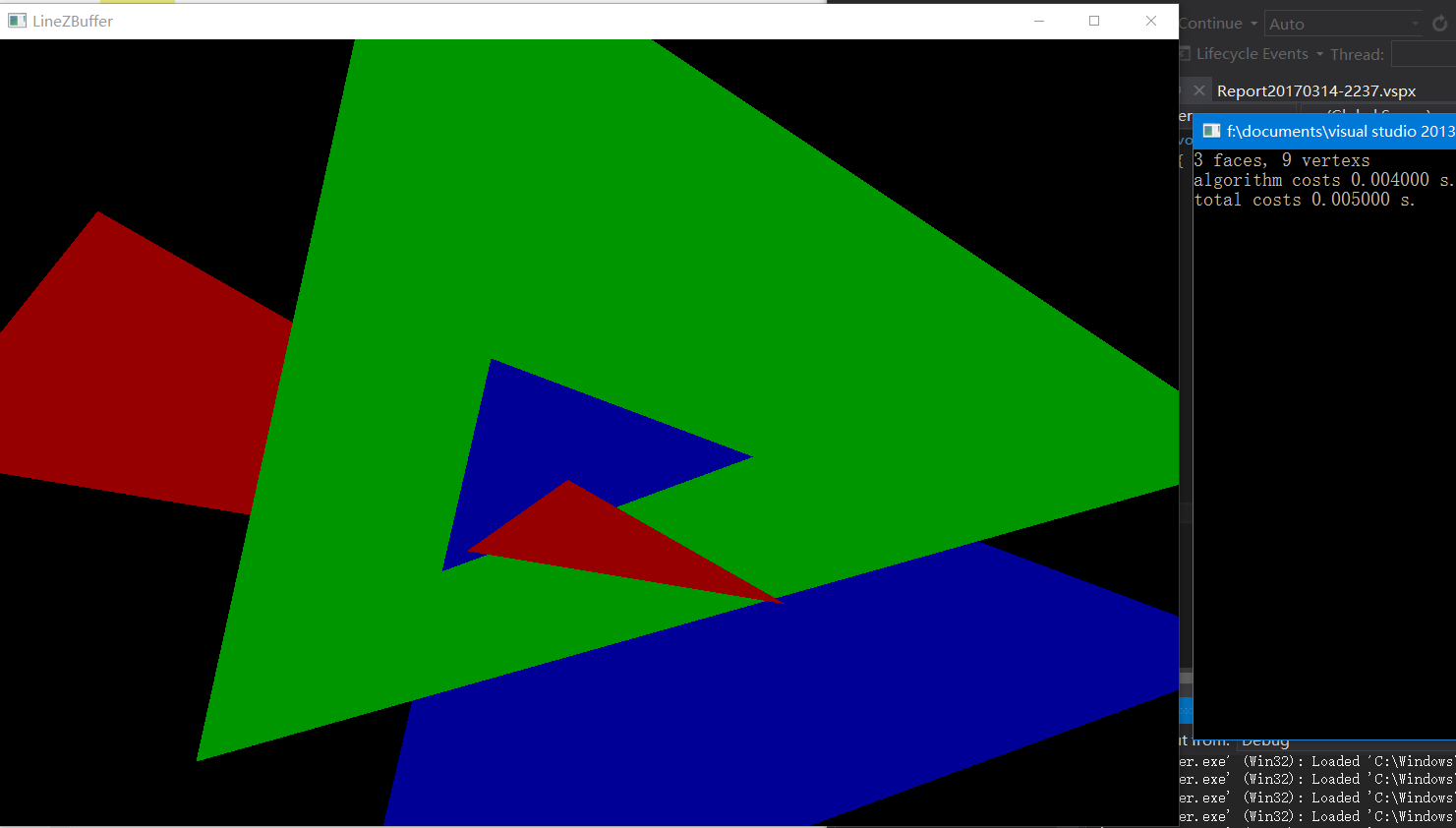
total costs 0.316000 s.

2花盆模型

7625 faces, 7608 vertexs

algorithm costs 0.013000 s.

total costs 0.162000 s.

3. 正确性测试

这是一个正确性测试