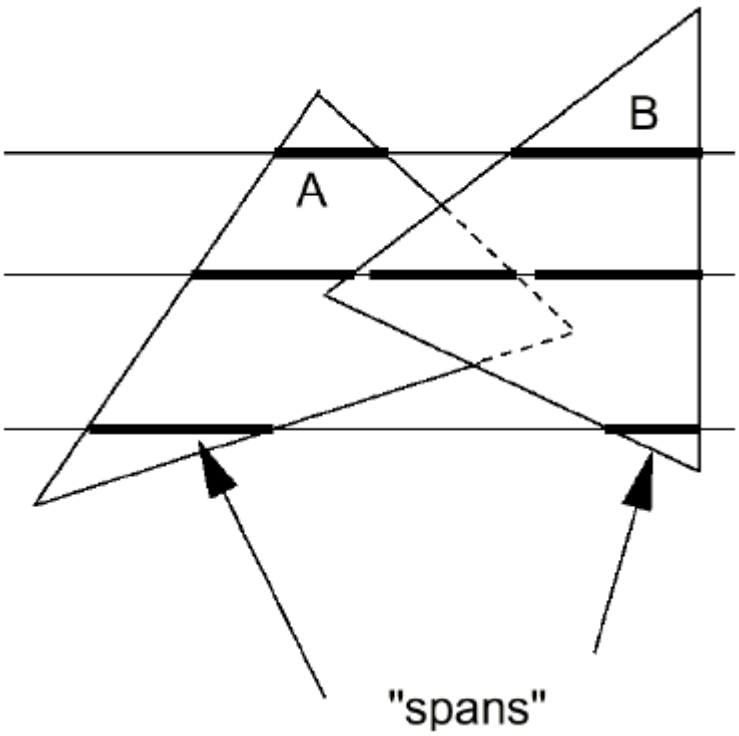
区间扫描线算法

编程环境

VS2013+opencv2.4.13

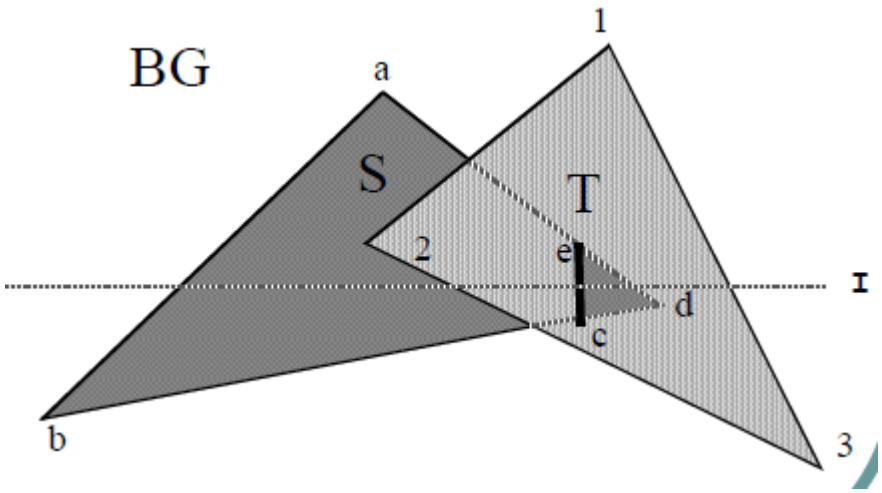
算法思想

不采用 z-buffer 的逐点判断，而是对每一条扫描线分割成几个区间，然后确定每一个区间属于哪多边形，用多边形的颜色对离视点最近区间着色。



概述

对于每一条扫描线，从左到右穿过活化多边形，通过奇偶性记录每个区间包含的多边形，然后依据多边形的参数计算深度来判断每一个区间的颜色值，通过draw()函数绘制出图像矩阵，其中还可以使用rotation()函数旋转模型来检测算法的鲁棒性。由于仅仅通过活化边表来获得区间对于多边形有贯穿的情况会出现错误，因此可以选择再draw()函数提取区间时加入区间多边形求交判断来进一步细分区间。



数据结构

Point_t记录每个顶点的坐标

```
struct Point_t
{
    double x, y, z;
};
```

Line记录每条边的信息

```
struct Line
{
    int x;               // start x
    int point1,point2;
    double dx;           // x1-x2
    int dy;              // number of crossed scaning-line(y1-y2)
    int id;              // polygon id
    Poly* poly;          // polygon pointer
};
```

Active_Line记录活化多边形信息

```
struct Active_Line
{
    Poly* poly;          // polygon pointer
    int id;              // polygon id
    double x;            // start x
    double dx;           // dx(-1/k)
    int dy;              // number of crossed scaning-line
    double z;            // depth of start point
    double dzx;          // depth variance along x, dzx = -a/c (c!= 0)
    double dzy;          // depth variance along y, dzy = b/c (c!= 0)
};
```

Color记录颜色信息

```
struct Color
{
    unsigned char r, g, b,a;
};
```

Poly记录多变形信息

```
struct  Poly
{
    int id;                         // polygon id
    double a, b, c, d;              // ax + by + cz + d = 0
    int dy;                         // number of crossed scaning-line
    Color color;                    // polygon color
    bool flag;
    Poly() : flag(false){}
    std::vector <int> point;      // record the id of points belonging to this polygon
};
```

算法实现

核心通过Draw（）函数实现对指定大小的图像矩阵像素进行赋值，其中可以选择是否判断区间内多变形是否有交点

```
void draw()
{
    //对整个图像从上到下一条一条扫描
    for (y = HEIGHT - 1; y >= 0; y--) {
        // initialization of background color
        memset(color, 0, sizeof(color));

        // add new active edges
        for (vector <Line>::iterator it = line[y].begin(); it != line[y].end();it++)
        {
            //扫描边表对活化边表增加新的活化边
        }

        //delete invalid edges
        for (list <Active_Line>::iterator it = aet.begin(); it != aet.end();)
        {
            //删除已经无效的活化边 (if dy == 0)
        }

        //if have no active edges please skip
        if (aet.size() <= 0)
            continue;

       //对活化边表进行重新排序
        aet.sort(cmp);

       //从左向右对两个相邻两个活化边表区间判断区间颜色值

        for (int j = 0; j < aet.size() - 1; j++)
```

```
        {
            //判断当前区间有哪些多边形，将其放在链表idx中

            //对idx中的多边形进行深度判断看此区间属于哪个多边形
            //如果需要判断此区间内是否有多边形相交，则对此区间进行进一步细分
#ifdef use_crosspoint
            //对idx中的多变形两两求交点并将交点放入interpoint链表中
#endif
            //对interpoint中的区间判断idx中哪个多边形的深度大则对此区间颜色赋予此多边形颜色
        }
    }
}
```

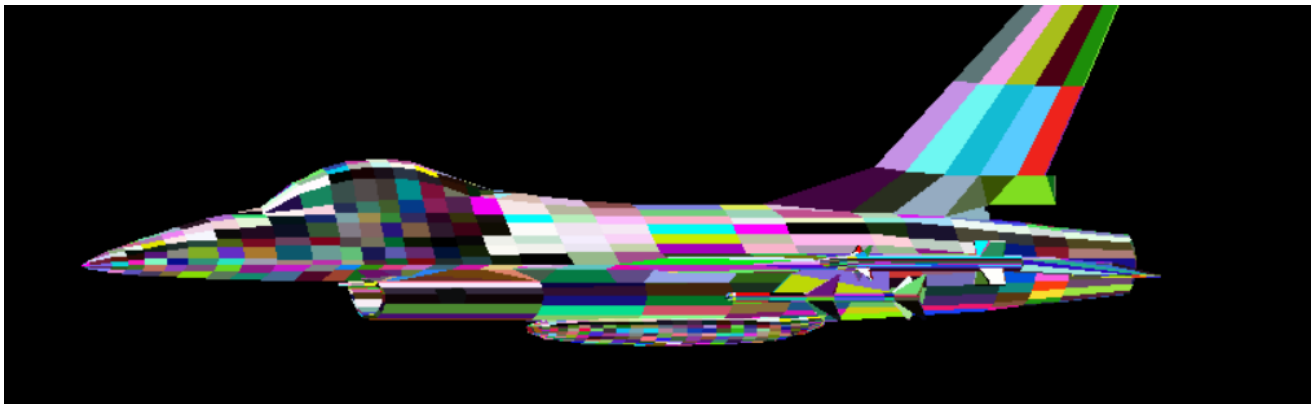
函数rotation（）是对原数据进行旋转以便对模型进行多维度观看和对算法检测鲁棒性

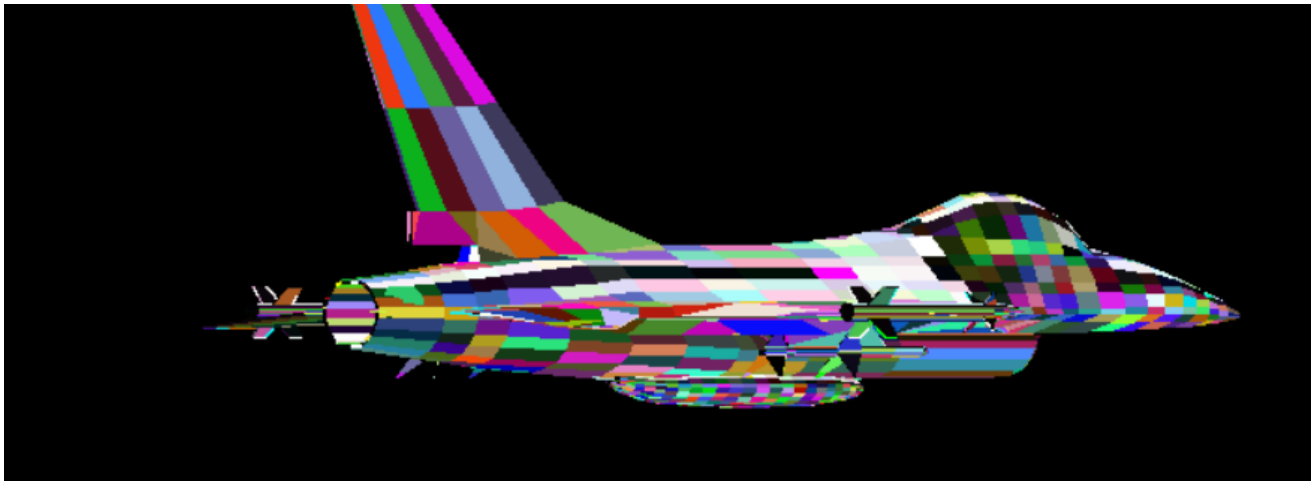
```
void rotate(double thetax, double thetay)
{
    //对每一个点做旋转变换
    for (vector <Point_t> ::iterator it = point.begin(); it != point.end(); ++it)

    //重新计算多变形表中的平面方程参数并同时重新构建边表
    for (int i = 0; i < MAX_HEIGHT;i++)
        for (vector <Poly*> ::iterator it = poly[i].begin(); it != poly[i].end(); ++it)
        {
            //依据多边形存储的对应顶点id重新计算多边形的参数

             //依据多边形存储的对应顶点id填充边表
             addLine((*it), (*it)->id, lineidx, v, s);
        }

    //重新绘制图像矩阵
    memset(image, 0, sizeof(image));
    draw();
}
```

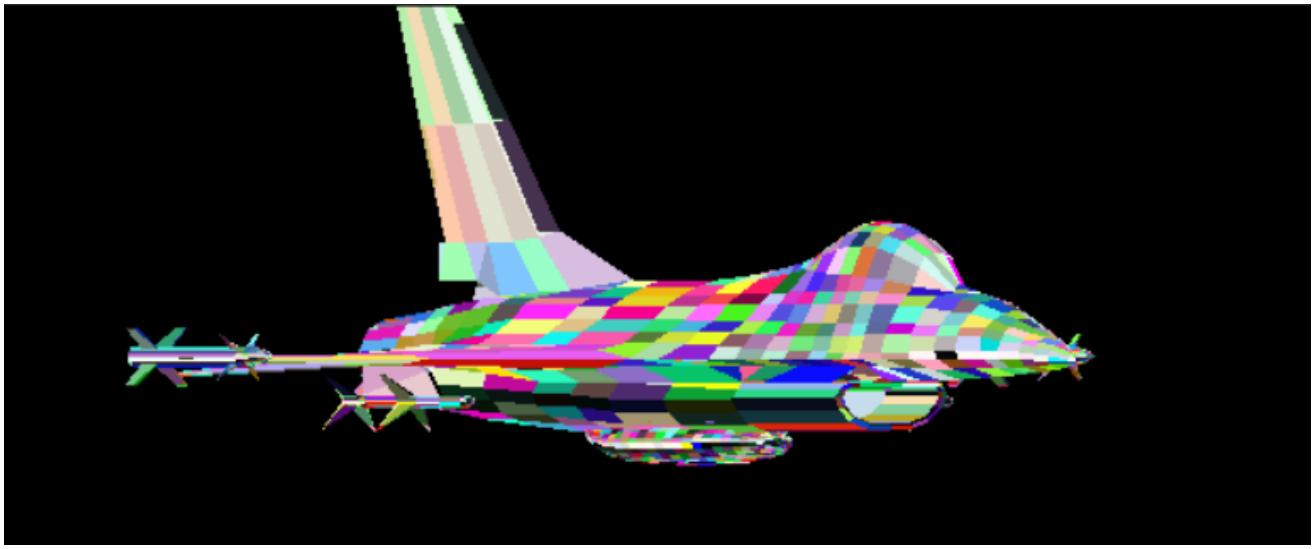
结果展示

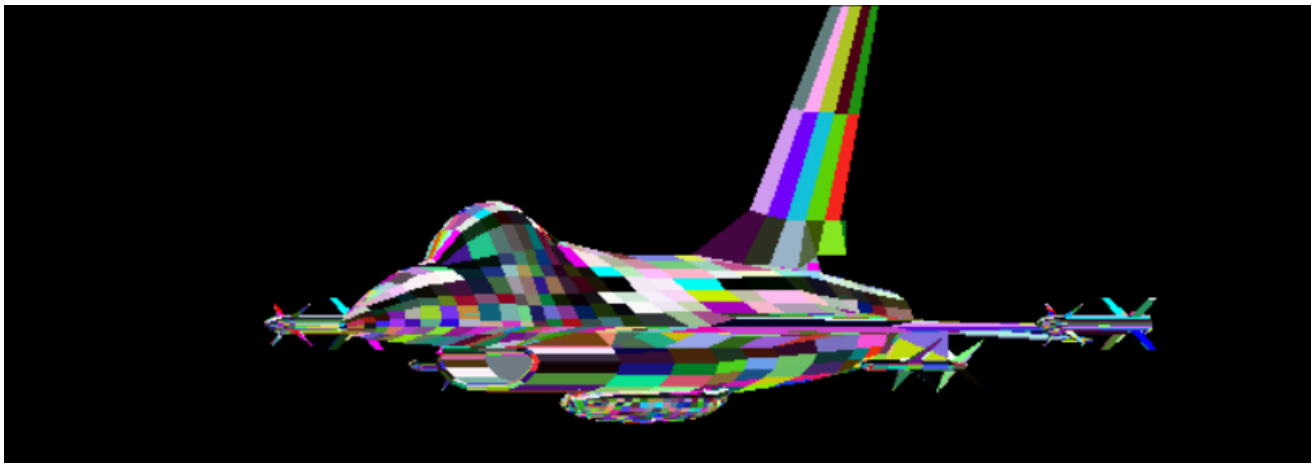
对于每个面片采用随机数生成颜色，并加入简单光照，光源在左上角。

f-16飞机的不同旋转角度（2366个面片）









雕像模型(43357个面片)