计算机图形学 —— 区域扫描线Z-Buffer算法

11821095 葛林林

2019 年 1 月 8 日

1 预备知识

1.1 obj文件

obj文件并不考虑物体的大小，所以不同的物体读入的坐标范围可能变化很大，因此为了显示的方便需将其转为当前绘制坐标系中。

顶点的表示：顶点以v开头后面跟着该顶点的x, y, z三轴坐标，示例如下

$$format. \quad v\ x\ y\ z$$

$$e.g. \quad v\ -57.408021\ 196.143694\ 2.816352$$

纹理坐标的表示：纹理坐标以vt开头。

$$format. \quad vt\ tu\ tv$$

法向量的表示：法向量的表示以vn开头。

$$format. \quad vn\ nx\ ny\ nz$$

$$e.g. \quad vn\ 5.9333\ -0.4798\ -1.8985$$

面的表示：面以f开头，代表”face”的意识，格式为“f 顶点索引/ 纹理坐标索引/ 顶点法向量索引”，如下所示

$$format. \quad f\ v/vt/vn\ v/vt/vn\ v/vt/vn$$

$$e.g. \quad f\ 1/1/1\ 2/2/2\ 3/3/3$$

1.2 OpenGL预备知识

1.2.1 OpenGL的坐标系

本报告中用到的OpenGL坐标系有世界坐标系、当前绘制坐标系和屏幕坐标系。其中当前绘制坐标系，在为使用glTranslatef()，glScalef(), glRotatef()等函数进行操作之前的初始状态时与世界坐标系是重合的，其具体的定义如下表表格所示：

Table 1: 坐标系的介绍

坐标系名称		轴方向定义
世界坐标系 $O_wX_wY_w$	原点O_w	屏幕中心为原点(0,0,0)
	X_d轴	从左往右为X_s轴正方向，可见范围为[-1,1]
	Y_d轴	从下往上为Y_s轴正方向，可见范围为[-1,1]
	Z_d轴	由屏幕内部指向外部
初始状态时当前绘制坐标系 $O_dX_dY_dZ_d$	原点O_d	屏幕中心为原点(0,0,0)
	X_d轴	从左向右为X_d轴正方向
	Y_d轴	从下往上为Y_d轴正方向
	Z_d轴	由屏幕内部指向外部
屏幕坐标系 $O_sX_sY_s$	原点O_s	屏幕左上角为(0,0)
	X_d轴	从左往右为X_s轴正方向
	Y_d轴	从下往上为Y_s轴正方向

1.2.2 OpenGL工程的搭建

OpenGL是一种跨平台的图形渲染编程接口，下面总结了搭建OpenGL工程的过程。

```
void OpenGLFunc(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH | GLUT_RGBA | GLUT_STENCIL);
    glutInitWindowSize(800, 600);        //set window size
    glutInitWindowPosition(100, 150);    //set window position
    glutCreateWindow("Display");         //window name
    glutDisplayFunc(DisplayFunc);        //call self defined display function
    glutIdleFunc(IdleFunc);          //call self-defined idle function
    glutKeyboardFunc(KeyboardFunc); //call self-defined keyboard function
    glutSpecialFunc(SpecialFunc);        //call self-defined special function
    glutMouseFunc(MouseFunc);        //call self-defined mouse function
    glutMotionFunc(MotionFunc);      //call self-defined mouse motion function
    glutPassiveMotionFunc(PassiveMotionFunc);//call self-defined passive mouse motion function
    glutMainLoop();
}
```

上述代码是使得OpenGL程序能够正常运行的一个模板，该段程序为OpenGL产生的窗口设置回调函数：显示回调函数、空闲回调函数、键盘回调函数、特殊键回调函数，鼠标回调函数、鼠标按下移动回调函数和鼠标移动回调函数，他们分别定义在如下所示的函数中：

```
    void DisplayFunc(...){...}
    void IdelFunc(...){...}
    void KeyboardFunc(...){...}
    void SpecialFunc(...){...}
    void MouseFunc(...){...}
    void MotionFunc(...){...}
    void PassiveMotionFunc(...){...}
```

2 算法

2.1 数据结构

2.1.1 边表

边表用来记录所有该模型包含的所有面片的边信息，边的数据结构如下所示：

- x:边上端点的x坐标
- x_c：在当前扫描线中，该边的x坐标
- dx：相邻两个扫描线之间的x坐标之差
- dy：该边剩余的扫描线数
- dy_{max}:该边涉及的所有扫描线数
- id：该边对应的多边形id号

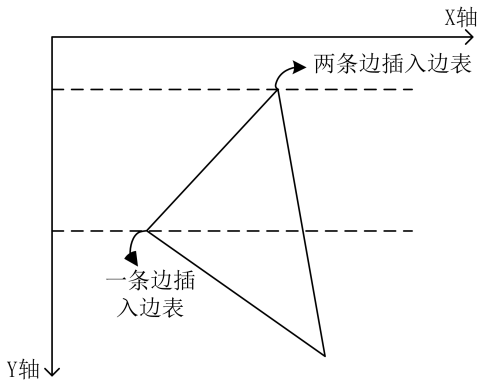

Figure 1: 边表初始化示意图

2.1.2 活化边表

活化多边形表的结构和边表相同，与之不同的是活化边表记录了当前扫描线所在行涉及到的边。因此该表在扫描线进行移动时需要不断的清空画完的边并且不断地更新加入新的边。

2.1.3 多边形表

多边形表存储了该模型包含的所有面片所在面的信息，多边形表的数据结构如下所示：

- a, b, c, d:当前多边形所在面的方程系数
- $color$：当前多边形的颜色

2.2 算法流程

如下图所示是本报告中所用算法的算法流程图：

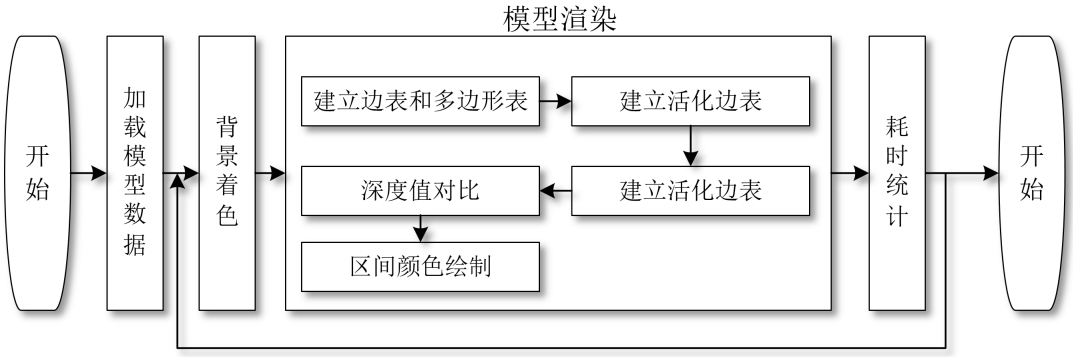

Figure 2: 算法流程图

2.3 算法的加速

2.3.1 边状态判断加速

由于边状态的判断比较复杂会有大量不必要的计算，考虑到多边形边的性质，既同一条扫描线中的同一个多边形有且仅有两条边过该扫描线，并且x值小的为in状态，x值大的为out状态，所以我们可以以边对于的多边形id对活化边表进行排序，并且保证同一个多边形的边尽量靠近，以防止存在刚好x值相等的边影响排序结果。

2.3.2 背景色加速

由于本报告中加入了背景色，如果每次都要重新计算背景色会耗费不必要的CPU时间，因此本报告中初始化时将背景色的值存入一组数组中，在绘制时直接读取。

3 实验结果

3.1 实验环境

本次实验的环境如下：

Table 2: 实验环境参数

参数	描述
System	Windows 10 64bit
CPU	Intel(R) Core(TM) i5-2410M CPU @2.30GHz 2.3GHz（4核）
RAM	6GB
IDE	Visual Studio 2017
Library	OpenGL

3.2 简单案例的设计

在实际调试过程中会出现很多问题，为了方便调试，设计了如下所示的一些简单实例帮助调试：

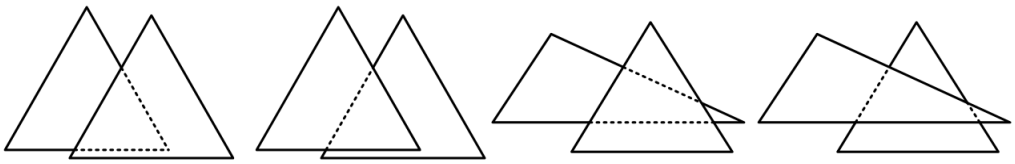

Figure 3: 简单案例示意图

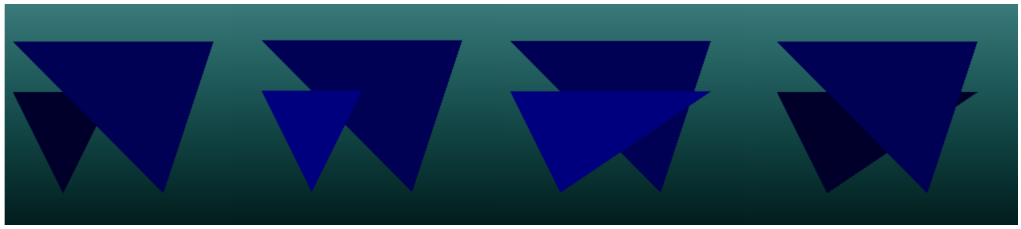

Figure 4: 简单案例测试效果图

3.3 数据集的测试

测试部分显示的窗口大小为600×600，得到的结果如下表所示：

Table 3: 案例测试

文件	三角片数	顶点数	优化前帧速(fps)
cat.obj	2755	5506	11.7883
duck.obj	791	3957	6.0306
bunny.obj	69451	208353	1.9299

如下所示是各个模型显示的效果图：

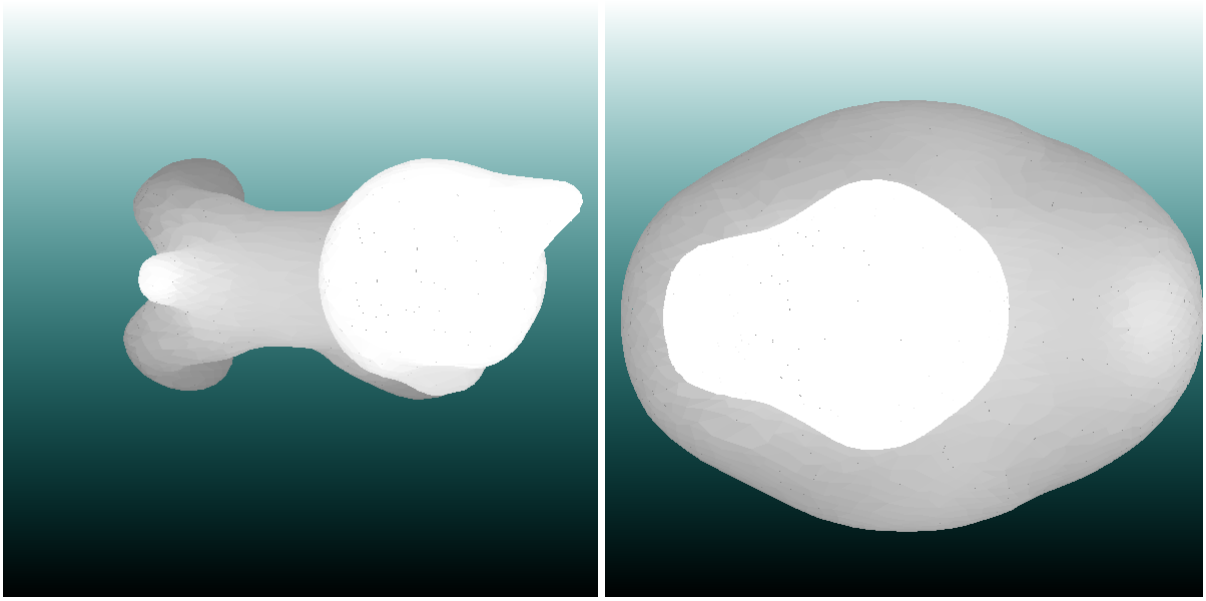

Figure 5: cat和duck模型显示效果图

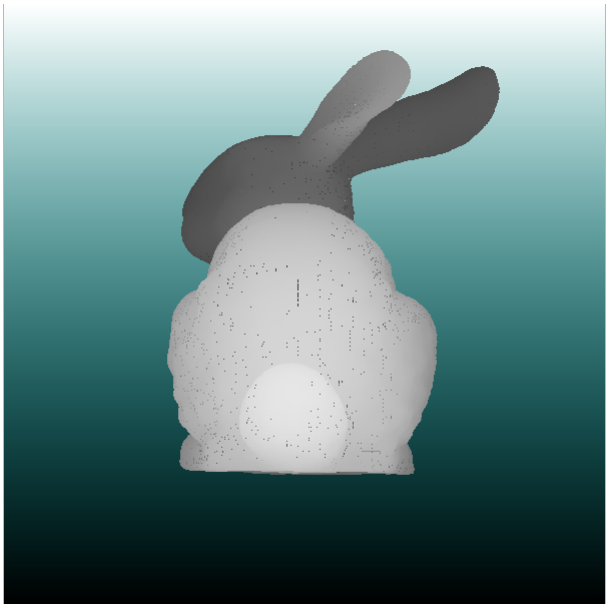

Figure 6: bunny模型显示效果图

从效果图中可以较清晰的看出模型的轮廓和纹理(由于为加光照模型，没有阴影效果)，除此之外效果图中还有一些斑点，可能的原因是多边形端点处考虑依然不够全面，有待进一步的完善。

4 关于一些问题的讨论

4.1 边状态的判定

方案一： 该方案是利用边状态的一些性质进行判断，如下所示总结出边状态的两大性质：

- 边的in和out状态的个数必须对应；
- 对于扫描线扫到的同一个三角片中的边状态必定in在out之前，将活化边表按照id进行排序，这样同个三角形的in和out的边就相邻存放，x_c小的则为in的边，x_c大的则为out的边。

方案二： 如下图所示是线段P_1P_2为in状态的情况，假设P_1, P_2, P_3点对应的坐标分别为$(x_1, y_1), (x_2, y_2), (x_3, y_3)$。则线段$P_1P_2$的斜率为

$$k = \frac{y_1 - y_2}{x_1 - x_2} \tag{1}$$

而线段P_1P_2对应的直线方程为：

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1} \tag{2}$$

令

$$f(x, y) = \frac{y - y_1}{y_2 - y_1} - \frac{x - x_1}{x_2 - x_1} \tag{3}$$

则当满足如下公式时则为in状态

$$sign(y_2 - y_1)kf(x_3, y_3) < 0 \tag{4}$$

Figure 7: $f(x_3, y_3) > 0, k < 0$

Figure 8: $f(x_3, y_3) < 0, k > 0$

总结： 对比方案一和方案二不能看出方案一的思路更为简单计算量较小，因此算法中采用了**方案一**进行边状态的判断。

4.2 模型着色的考虑

由于自己写光照模型会比较复杂，因此本报告中采用了较为简单的方案，及将z轴的值转化到[0,1]之间，以该值作为灰度值对模型进行着色。