

计算机图形学 —— 区域扫描线Z-Buffer算法

11821095 葛林林

2018 年 12 月 25 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，因此为了现实的方便需要将其转为规范化设备坐标。

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. $v \ x \ y \ z$

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

纹理坐标的表示：纹理坐标以 vt 开头。

format. $vt \ tu \ tv$

法向量的表示：法向量的表示以 vn 开头。

format. $vn \ nx \ ny \ nz$

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

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

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

1.2 OpenGL预备知识

1.2.1 OpenGL的坐标系

OpenGL中常用的坐标系有局部坐标系、世界坐标系、视点坐标系、投影坐标系、规格化设备坐标系和屏幕坐标系。

规格化设备坐标系 $O_d X_d Y_d Z_d$:其坐标范围为 $\{x, y, z \in [-1, 1]\}$ ，它以创建的窗口的中心为原点 O_d

X_d 轴：从左向右为正方向

Y_d 轴：从下往上为正方向。屏幕坐标系 $O_s X_s Y_s$ 以屏幕左上角为坐标原点，从左往右为 X_s 轴正方向，从上往下为 Y_s 轴

正方向。

世界坐标系

原点 O_w ：以屏幕中心为原点，始终保持不变。

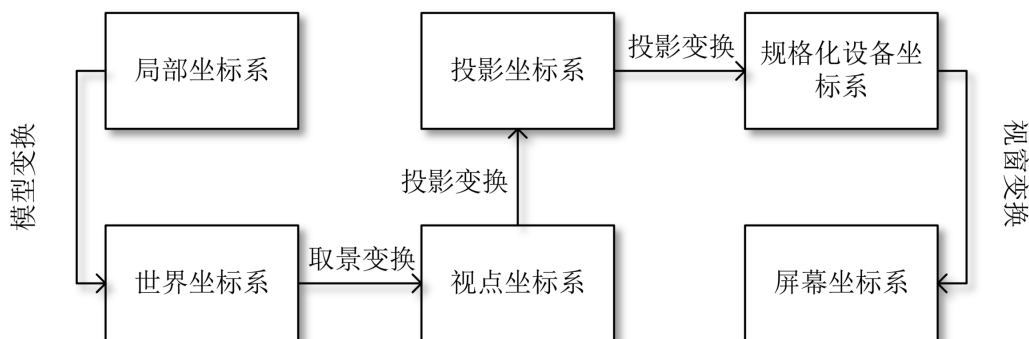


Figure 1: 常用坐标系变换

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(IdelFunc);                 //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

加载文件→获取深度值→消隐算法

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

2.1.2 活化边表

活化多边形的结构和边表相同，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 算法的加速

2.2.1 存储器的优化

2.2.2 取整的优化

在扫描线算法中，需要进行大量的取整运算，c++中可以利用round函数来实现。宏相比于函数而言能够减少函数带来的开销，因此我们可以定义如下宏来代替该函数：

```
#define ROUND(DAT) ((int)(DAT+0.5))
```

3 实验结果

3.1 实验环境

本次实验的环境如下：

Table 1: 环境参数

参数	
System	Windows 10 64bit
CPU	Intel(R) Core(TM) i5-4590 CPU @3.30GHz 3.3GHz（4核）
RAM	16GB
IDE	Microsoft Visual Studio 2017

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：

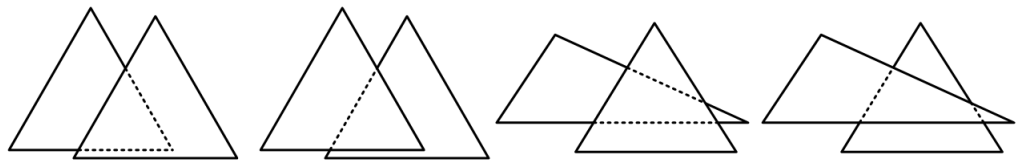


Figure 2: 简单案例示意图

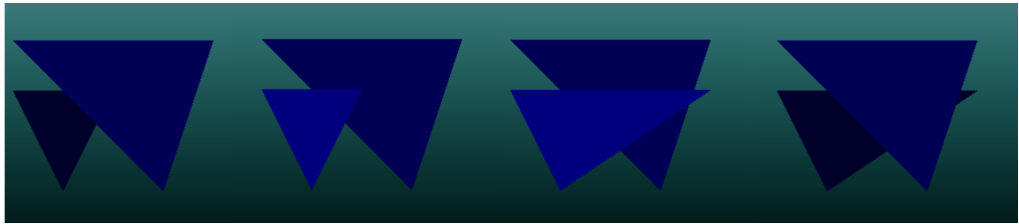


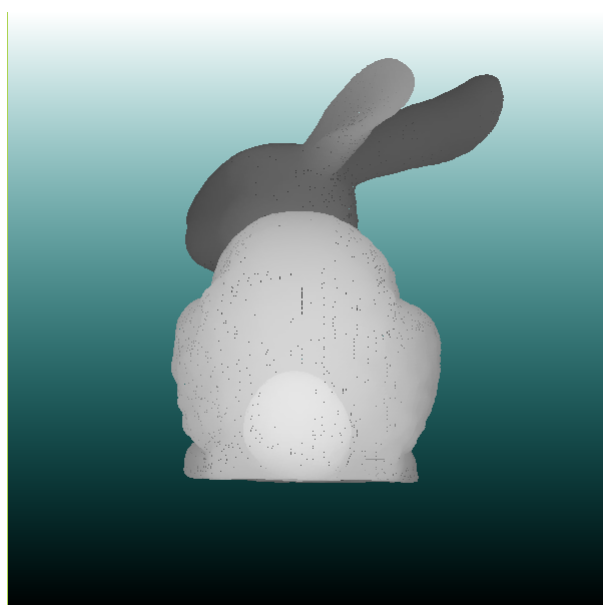
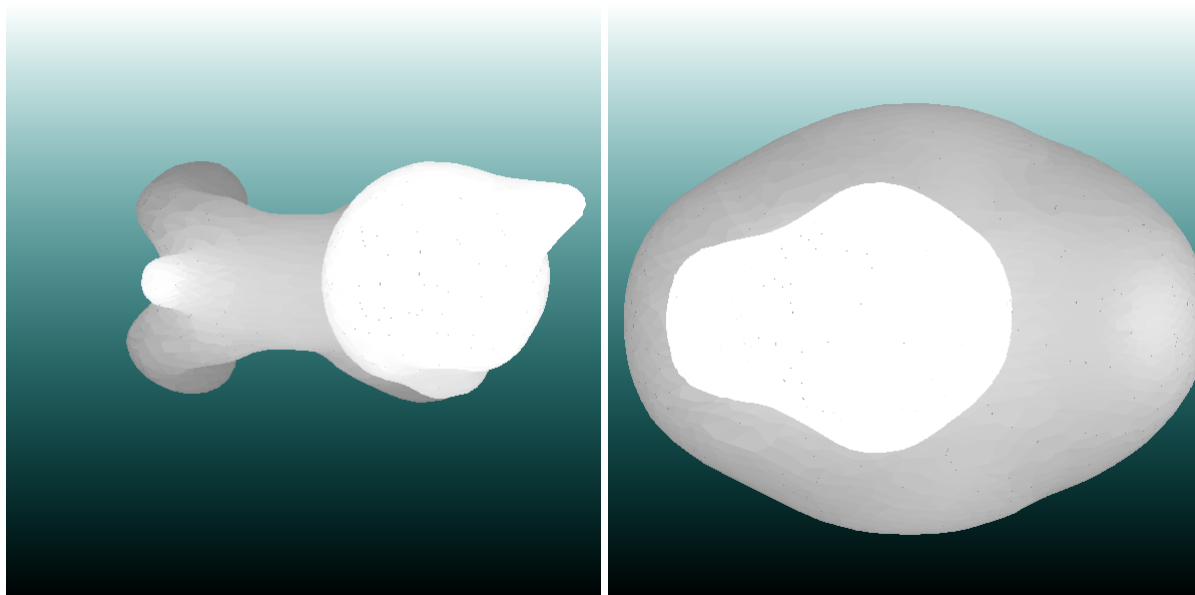
Figure 3: 简单案例测试效果图

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 2: 案例测试

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



3.4 程序分析

本次使用的是称为**bunny.obj**的文件，该文件中包含了*v*和*f*开头的的数据，代码中**load_obj.h**和**load_obj.cpp**两个文件定义了加载obj文件的方法。

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

$$k = \frac{y_1 - y_2}{x_1 - x_2} \quad (1)$$

而线段 P_1P_2 对应的直线方程为：

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1} \quad (2)$$

令

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

则当满足如下公式时则为in状态

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

Figure 4: $f(x_3, y_3) > 0, k < 0$

Figure 5: $f(x_3, y_3) < 0, k > 0$

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 结果中出现空白点

5 经验总结

- 用特殊的案例对结果进行测试