005--视觉班第5次课程[OpenGL专题]

一. 视觉班课程安排:

- 课程日期: 2020 年 7 月 10 日 周三 第 5 次课程 (共 21 次课程)
- 授课老师: CC 老师 (QQ: 1323177506)
- 研发老师: CC 老师
- 班主任老师:
  - 大大老师 (QQ: 188706023)
  - 朵朵老师 (QQ: 1550934962)
  - 婷婷老师 (QQ: 3470520842)
- 课程时长: 2小时
- 课程时间安排:
  - 上课: 20:00 – 21:00
  - 休息: 21:00 – 21:10
  - 上课: 21:10 – 22:00
- 课程内容:
  - 深度测试;
  - 多边形模型
  - 多边形偏移
  - 颜色混合;
  - 案例001–对甜甜圈进行 正背面剔除/深度测试解决问题;
  - 案例002--使用OpenGL 混合方程式实现颜色混合;
  - 案例003--使用objectFrame 来进行变换

- 课程作业:
  - 将深度测试 理解以及案例执行过程分析写到博客上
  - 将关于颜色混合的原理 写到博客上

二. 课程内容安排

2.1 课程快速回顾

优秀作业点评:

逻辑教育iOS学院 _ 视觉班优秀博客分享(主讲老师:CC老师)[第4次作业] --注意:不要误删内容哦~					
日期	课程次数	QQ昵称	博客主题	检阅	博客作业地址
2020/7/9	4	正在注销账号	OpenGL矩阵入栈出栈详解及金字塔案例	推荐阅读	https://blog.csdn.net/weixin_40918107/article/details/107238364
2020/7/9	4	拾遗，那一季	案例 03：金字塔、六边形、圆环的绘制	推荐阅读	https://www.jianshu.com/p/2aa2145aae94
2020/7/9	4	拾遗，那一季	五、OpenGL 渲染技巧：正背面剔除	推荐阅读	https://www.jianshu.com/p/bd3fd3551b92
2020/7/8	4	收纳箱	五、OpenGL基础变换与矩阵栈	已阅	https://www.jianshu.com/p/595f7c6211f2
2020/7/8	4	收纳箱	四、OpenGL深度缓冲区、裁剪和混合	已阅	https://www.jianshu.com/p/f8e3a6443b20

全部作业提交博客地址: https://docs.qq.com/sheet/DVFpVdkpsQ0NvWFJX?tab=79xchx

隐藏面消除

问题: 使用平面着色器来绘制甜甜圈,会出现隐藏消除吗?

问题: 为什么使用默认光源着色器会出现隐藏面消除?

2.2 课程笔记

2.2.1 深度测试

深度测试解决什么问题:

1. 当甜甜圈, 旋转时. 2个部分重叠时. 此时OpenGL 不能清楚分辨 那个图层在前 那个图层在后.则会出现甜甜圈被啃一口的现象;
2. 隐藏面消除,除了使用 正背面剔除, 还可以使用深度测试来解决.

什么是深度: 深度就是在openGL坐标系中， 像素点的 z 坐标距离观察者的距离.

当观察者可以放在坐标系的任意位置, 所以. 不能简单的说Z 数值越大或越小. 观察者就越靠近物体;

如果观察者在Z轴的正方向, Z值越大则靠近观察者;

如果观察者在Z轴的负方向, Z值越小则靠近观察者;

深度缓冲区(DepthBuffer):
深度缓冲区在哪里? 其实它存储在显存中;
深度缓存区原理, 就是把距离观察者平面(近裁剪面)的深度值 与 窗口中每个像素点1对1进行关联以及存储;
120 * 120 = 120 * 120 个存储深度值

清空深度缓存区

```
1 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

开启深度测试

```
1 glEnable(GL_DEPTH_TEST);
```

深度缓冲区(DepthBuffer)和颜色缓存区(ColorBuffer)是对应的.颜色缓存区存储像素的颜色信息,而深度缓冲区存储像素的深度信息. 在决定是否绘制一个物体表面时, 首先要将表面对应的像素的深度值与当前深度缓冲区中的值进行比较. 如果大于深度缓冲区中的值,则丢弃这部分.否则利用这个像素对应的深度值和颜色值.分别更新深度缓冲区和颜色缓存区. 这个过程称为”深度测试”

开启了深度测试,则在绘制每个像素之前, OpenGL 会把它的深度值与目前像素点对应存储的深度值,进行比较. 如果 像素点新对应深度值 < 像素点对应的深度值 . (意思比较当前2个图层时, 那个图层更加接近于观察者) 那么此时就会将该像素点的深度值进行取而代之; 反之,如果像素点上的新颜色值如果距离观察者更远,则应该会遮挡. 那么此时它所对应的深度值与颜色值就会被抛弃. 不进行绘制;

关闭深度测试

```
1 glDisable(GL_DEPTH_TEST);
```

深度缓存区的默认值为1.0. 表示最大的深度值. 深度值的范围是[0,1]之间.

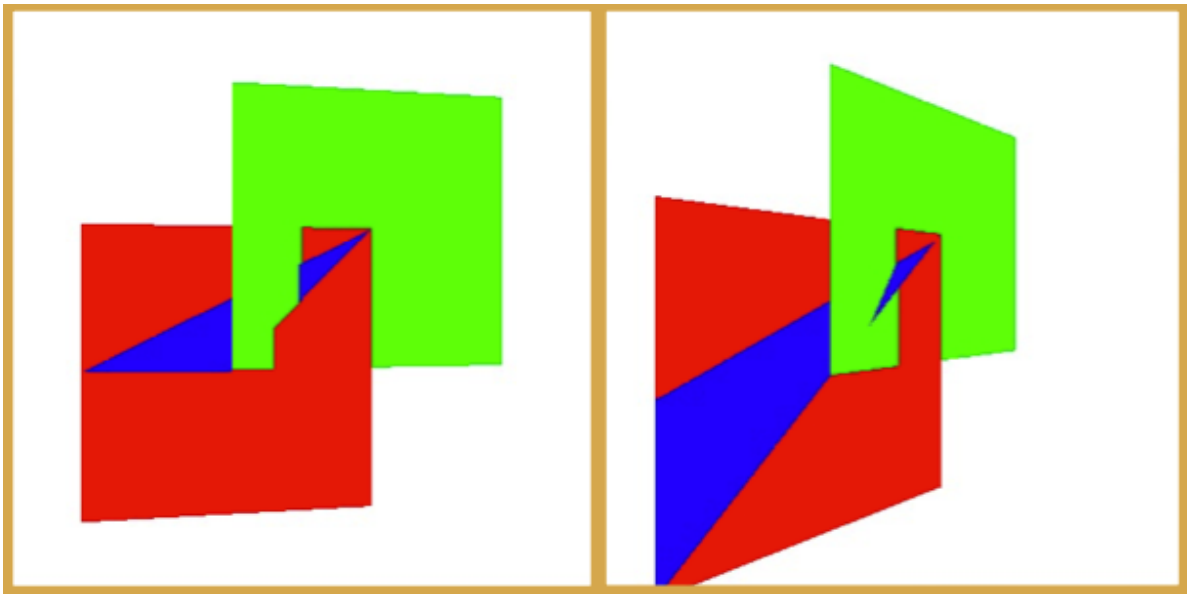
我们可以通过 `glDepthFunc(GLenum func)` **来修改 深度测试的测试规则. 那么测试的规则主要在于该函数中的枚举值;**

参数	说明
GL_ALWAYS	总是通过测试
GL_NEVER	总是不通过测试
GL_LESS	当前深度值 < 存储的深度值时通过
GL_EQUAL	当前深度值 = 存储的深度值时通过
GL_LEQUAL	当前深度值 <= 存储的深度值时通过
GL_GREATER	当前深度值 > 存储的深度值时通过
GL_NOTEQUAL	当前深度值 != 存储的深度值时通过
GL_GEQUAL	当前深度值 >= 存储的深度值时通过

疑问: 刚刚我们进行深度测试的图层都是非透明图层. 那么如果是半透明图层 OpenGL 需要怎么处理?
解答: 继续往下学习 颜色混合处理;

2.2.2 深度测试的潜在风险之 Z-fighting (Z冲突.闪烁)问题;

因为开启深度测试后,OpenGL 就不会再去绘制模型被遮挡的部分. 这样实现的显示更加真实.但是由于深度缓冲区精度的限制对于深度相差非常小的情况下.(例如在同一平面上进行2次制),OpenGL 就可能出现不能正确判断两者的深度值,会导致深度测试的结果不可预测.显示出来的现象时交错闪烁.的前面2个画面,交错出现.



同一个位置上 出现的图层,且深度值出现精确度很低时,就会容易引起 ZFighting 现象. 表示2个物体靠的非常的近,无法确定谁在前,谁在后. 而出现显示歧义;

2.2.3 如果解决ZFighting 问题:

既然, 是因为靠的太近,无法区分图层先后. 那么此时,就可以在2个图层之间加入一个微妙的间隔. 那么手动添加,复杂且不精确. 此时OpenGL 提供一个解决方案, "多边形偏移"

1. **启用多边形偏移 Polygon Offset**
    - 解决方法: 让深度值之间产生间隔.如果2个图形之间有间隔,是不是意味着就不会产生干涉.可以理解为在执行深度测试前将立方体的深度值做一些细微的增加.于是就能将重叠的2个图形深度值之前有所区分.

```
1 //启用Polygon Offset 方式
2 glEnable(GL_POLYGON_OFFSET_FILL)
3
4
5 参数列表:
6 GL_POLYGON_OFFSET_POINT      对应模式: GL_POINT
7 GL_POLYGON_OFFSET_LINE       对应模式: GL_LINE
8 GL_POLYGON_OFFSET_FILL       对应模式: GL_FILL
```

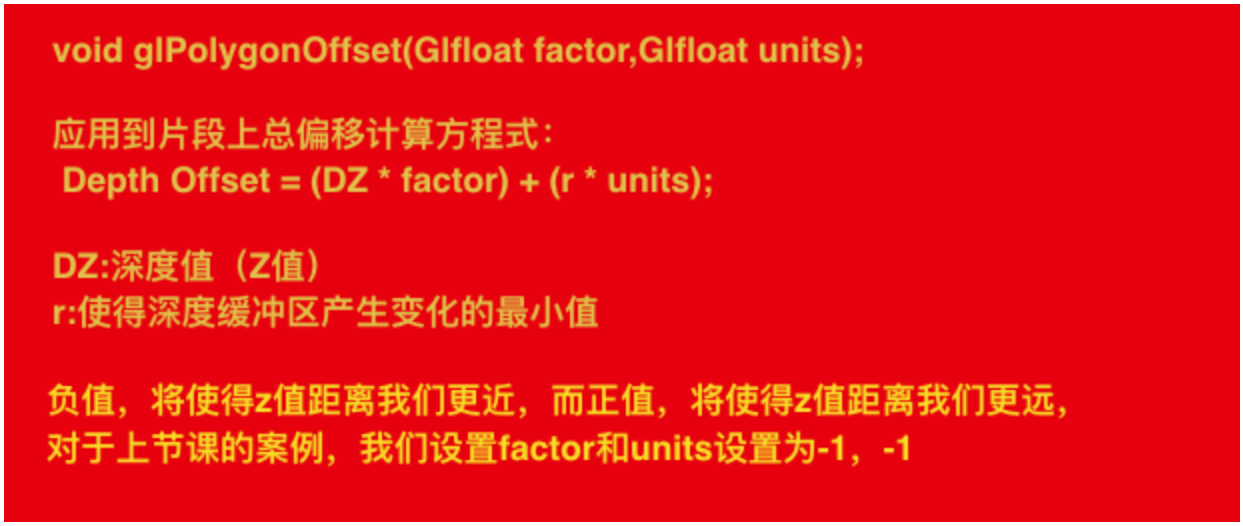
2. **指定偏移量**

- 通过glPolygonOffset 来指定.glPolygonOffset 需要2个参数: factor , units
- 每个Fragment 的深度值都会增加如下所示的偏移量:

  Offset = (m * factor) + (r * units);

  m : 多边形的深度的斜率的最大值,理解一个多边形越是与近裁剪面平行,m 就越接近于0.

  r : 能产生于窗口坐标系的深度值中可分辨的差异最小值.r 是由具体是由具体OpenGL 平台指定的一个常量.
- 一个大于0的Offset 会把模型推到离你(摄像机)更远的位置,相应的一个小于0的Offset 会把模型拉近
- 一般而言,只需要将-1 和 -1 这样简单赋值给glPolygonOffset 基本可以满足需求.



- 关闭多边形偏移 Polygon Offset

```
glDisable(GL_POLYGON_OFFSET_FILL)
```

2.2.4 如何预防ZFighting闪烁问题

- 不要将两个物体靠的太近，避免渲染时三角形叠在一起。这种方式要求对场景中物体插入一个少量的偏移，那么就可能避免ZFighting现象。例如上面的立方体和平面问题中，将平面下移0.001f就可以解决这个问题。当然手动去插入这个小的偏移是要付出代价的。
- 尽可能将近裁剪面设置得离观察者远一些。上面我们看到，在近裁剪平面附近，深度的精确度是很高的，因此尽可能让近裁剪面远一些的话，会使整个裁剪范围内的精确度变高一些。但是这种方式会使离观察者较近的物体被裁减掉，因此需要调试好裁剪面参数。
- 使用更高位数的深度缓冲区，通常使用的深度缓冲区是24位的，现在有一些硬件使用使用32/64位的缓冲区，使精确度得到提高

2.2.5 颜色混合

我们把OpenGL 渲染时会把颜色值存在颜色缓存区中，每个片段的深度值也是放在深度缓冲区。当深度缓冲区被关闭时，新的颜色将简单的覆盖原来颜色缓存区存在的颜色值，当深度缓冲区再次打开时，新的颜色片段只是当它们比原来的值更接近邻近的裁剪平面才会替换原来的颜色片段.

那么如果开启深度测试后.但是2个重叠的图层中, 有一个图层是半透明的. 有一个图层是非半透明的. 那么此时就不能进行单纯的 比较深度值,然后进行覆盖. 而是需要将2个图层的颜色进行混合;

```
1 glEnable(GL_BlEND);
```

目标颜色：已经存储在颜色缓存区的颜色值 [例如,前女友]
源颜色：作为当前渲染命令结果进入颜色缓存区的颜色值 [例如,现女友]

固定着色器/可编程着色器-> 使用开关方式 ->颜色混合(单纯的2个图层重叠进行混合)
可编程着色器->片元着色器 -> 处理图片原图颜色+薄薄的绿色(颜色值) -> 进行颜色混合方程式计算 -> 套用公式

2个 多个. 有关系 3个图层;
一个一个图层 0号图层 1号图层->混合存储混合后颜色 颜色缓存区 -> 2 号图层 -> 进混合

当混合功能被启动时，源颜色和目标颜色的组合方式是混合方程式控制的。在默认情况下，混合方程式如下所示：

```
1 当混合功能被启动时，源颜色和目标颜色的组合方式是混合方程式控制的。在默认情况下，混合方
```

```
程式如下所示:

    Cf = (Cs * S) + (Cd * D)

Cf : 最终计算参数的颜色
Cs :  源颜色
Cd : 目标颜色
S: 源混合因子
D: 目标混合因子
```

2. 设置混合因子

```
设置混合因子，需要用到glBlendFun函数
glBlendFunc(GLenum S,GLenum D);

S:源混合因子
D: 目标混合因子
```

表中R、G、B、A 分别代表 红、绿、蓝、alpha。
表中下标S、D，分别代表源、目标
表中C 代表常量颜色（默认黑色）

OpenGL 混合因子

函数	RGB混合因子	Alpha混合因子
GL_ZERO	(0,0,0)	0
GL_ONE	(1,1,1)	1
GL_SRC_COLOR	(Rs,Gs,Bs)	As
GL_ONE_MINUS_SRC_COLOR	(1,1,1) - (Rs,Gs,Bs)	1-As
GL_DST_COLOR	(Rd,Gd,Bd)	Ad
GL_ONE_MINUS_DST_COLOR	(1,1,1)-(Rd,Gd,Bd)	1-Ad
GL_SRC_ALPHA	(As,As,As)	As
GL_ONE_MINUS_SRC_ALPHA	(1,1,1)-(As,As,As)	1-As
GL_DST_ALPHA	(Ad,Ad,Ad)	Ad
GL_ONE_MINUS_DST_ALPHA	(1,1,1)-(Ad,Ad,Ad)	1-Ad
GL_CONSTANT_COLOR	(Rc,Gc,Bc)	Ac
Gl_ONE_MINUS_CONSTANT_COLOR	(1,1,1)-(Rc,Gc,Bc)	1-Ac
GL_CONSTANT_ALPHA	(Ac,Ac,Ac)	Ac
GL_ONE_MINUS_CONSTANT_COLOR	(1,1,1)-(Ac,Ac,Ac)	1-Ac
GL_SRC_ALPHA_SATURATE	(f,f,f)* f = min(As,1-Ad)	1

举例: 下面通过一个常见的混合函数组合来说明问题:

```
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
```

如果颜色缓存区已经有一种颜色红色（1.0f,0.0f,0.0f,0.0f）,这个目标颜色Cd，如果在这上面用一种alpha为0.6的蓝色（0.0f,0.0f,1.0f,0.6f）

Cd（目标颜色）= （1.0f,0.0f,0.0f,0.0f）;
Cs（源颜色）= （0.0f,0.0f,1.0f,0.6f）;
S = 源alpha值 = 0.6f
D = 1 – 源alpha值= 1–0.6f = 0.4f

方程式Cf = (Cs * S) + (Cd * D)

等价于 = （Blue * 0.6f） + (Red * 0.4f)

2.2.6. 颜色混合总结

刚刚的例子，最终颜色是以原先的红色（目标颜色）与 后来的蓝色（源颜色）进行组合。
源颜色的alpha值越高，添加的蓝色颜色成分越高，目标颜色所保留的成分就会越少。
混合函数经常用于实现在其他一些不透明的物体前面绘制一个透明物体的效果。

2.2.7 修改颜色混合方程式

默认混合方程式:

Cf = (Cs * S) + (Cd * D)

实际上远不止这一种混合方程式，我们可以从5个不同的方程式中进行选择

```
//选择混合方程式的函数:
glbBlendEquation(GLenum mode);
```

可用的混合方程模式

模式	函数
GL_FUNC_ADD	Cf =(Cs * S) + (Cd * D)
GL_FUNS_SUBTRACT	Cf = (Cs * S) – (Cd * D)
GL_FUNC_REVERSE_SUBTRACT	Cf = (Cd * D) – (Cs * S)
GL_MIN	Cf = min(Cs ,Cd)
GL_MAX	Cf = min(Cs ,Cd)

```
1 常量混合颜色，默认初始化为黑色（0.0f,0.0f,0.0f,0.0f），但是还是可以修改这个常量混合颜色。
2 void glBlendColor(GLclampf red ,GLclampf green ,GLclampf blue ,GLclampf alpha );
```