OpenGL图形编程

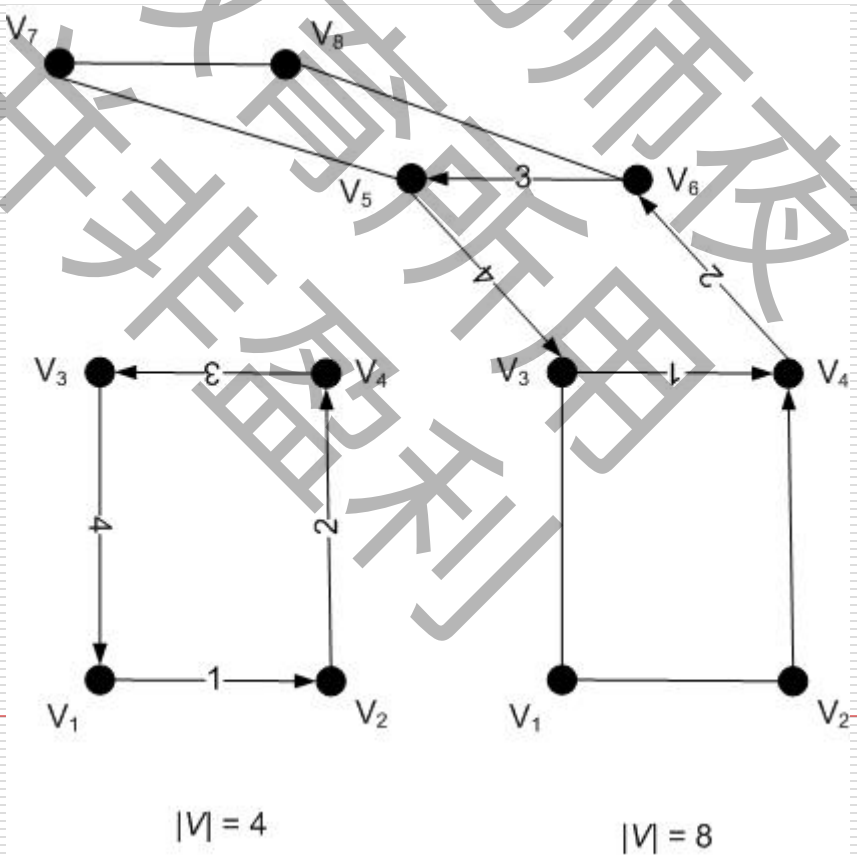
授课教师：少书师夜

2.4 多边形面的绘制

- 三角形面的绘制
  - GL_TRIANGLES
  - GL_TRIANGLE_STRIP
  - GL_TRIANGLE_FAN
- 四边形面的绘制
  - GL_QUADS
  - GL_QUAD_STRIP
- 多边形面的绘制（GL_POLYGON）

四边形面的绘制

- GL_QUADS
- GL_QUAD_STRIP



应用

- 组合
- 画圆

OpenGL图形编程

授课教师：少书师夜

OpenGL的缓存

OpenGL进行图形显示需要四种缓存：

1. 颜色缓存
2. 深度缓存
3. 模板缓存
4. 累计缓存

颜色缓存

- 由红、绿、蓝、Alpha位等位平面(Bitplane)组成的，有前缓存(Front Buffer)、后缓存(Back Buffer)、左前(Front_Left)和右前(Front_Right)缓存、左后(Back_Left)和右后(Back_Right)缓存。
- 左前缓存是必须的颜色缓存。前缓存是可见缓存，后缓存是不可见缓存。前后缓存技术可以实现动画操作
- 与颜色缓存相关的主要函数有：
- 清除颜色缓存：glClear(GL_COLOR_BUFFER_BIT)，用于清除当前显示缓冲区内容，为开始新的绘制做好准备。

- 设置清除颜色：glClearColor(red，green，blue，alpha)，用当前颜色(red, green, blue, alpha)清除当前显示缓冲区内容，为开始新的绘制做好准备。
- 屏蔽颜色缓存：glColorMask()，分别设置红、绿、蓝、alpha的可写属性。
- 选择颜色缓存：glDrawBuffer()，用于对双缓存中一个进行选择。
- 交换颜色缓存：swapBuffer()，交换前后缓存中的颜色，以实现动画

深度缓存

- 记录每个像素点所对应的物体点到视点的距离，由此决定表面的可见性。在进行消隐的时候，OpenGL必须知道各物体间的相对位置关系，从而模拟出物体相互遮挡得效果，因此，需要进行深度测试。而深度测试的结果就生成了深度缓存。

与深度缓存相关的OpenGL操作主要有：

- ✓ 清除深度缓存：glClear(GL_DEPTH_BUFFER_BIT)，用于清除当前显示缓冲区内容，为开始新的绘制做好准备；
- ✓ 设置清除值：glClearDepth(1.0)；

- ✓ 屏蔽深度缓存：glDepthMask(GL_TRUE)，表示可以写深度缓存；glDepthMask(GL_FALSE)，表示禁止写深度缓存；
- ✓ 启动和关闭深度测试：glEnable(GL_DEPTH_TEST)，表示开启深度测试；glDisable(GL_DEPTH_TEST)，表示禁止深度测试。
- ✓ 确定测试条件：glDepthFunc()，根据函数参数确定测试方式，具体的参数说明请参考OpenGL手册。
- ✓ 确定深度范围：glDepthRagne(Glclampd zNear, Glclampd zFar)，参数zNear和zFar分别说明视景体的前景面和后景面向窗口坐标映射的规格化坐标，便于后续使用。

模板缓存和累积缓存主要用于图形的特殊效果绘制。

- **模板缓存**：存放像素的模板值。可用于控制像素是否被改写，因而其可以禁止在屏幕的某些区域绘图。模板缓存可以用于多种复杂图形的绘制，例如：屏蔽屏幕区域(凸区域或凹区域，因而也可以绘制凹多边形等)；遮盖物体；制作物体的交集等。

- **累积缓存**：是一系列绘制结果的累积，可以用来实现场景的反走样、景深模拟、运动模糊等。例如为了实现全局反走样，可多次绘制场景，每次绘制时轻微移动场景(相当于在空间上抖动场景)，把多次绘制的结果进行累积并最后一次输出，结果场景的边界会变得模糊，从而实现全局反走样。

3. OpenGL二维观察与三维变换

- 3.1二维观察
- 3.2三维变换

3.1OpenGL二维观察

- 观察流水线：将该场景的世界坐标描述经各种处理变换到一个或多个输出设备参照系来显示的过程；
- 裁剪窗口（世界窗口、观察窗口、viewing window）：二维场景中要显示的部分；
- 视口（viewport）：控制在显示窗口的定位；窗口选择要看什么，而视口指定在输出设备的什么位置进行观察
- 二维观察变换（two-dimensional viewing transformation）有时称为窗口到视口的变换或窗口变换**：场景的描述从二维世界坐标系到设备坐标系的映射
- 规范化设备坐标：为了使观察处理独立于输出设备，图形系统将对象描述转化到规范设备坐标系并提供裁剪程序；

3.1OpenGL二维观察

实现二维观察的步骤：

- 3.1.1指定矩阵堆栈
- 3.1.2指定裁剪窗口
- 3.1.3指定视区

3.1.1指定矩阵堆栈

- 指定当前操作的是投影矩阵堆栈

```
glMatrixMode (GL_PROJECTION) ;
```

- 初始化，即指定当前操作的矩阵堆栈的栈顶元素为单位矩阵。

```
glLoadIdentity();
```

3.1.2指定裁剪窗口

- 定义二维裁剪窗口

```
gluOtho2D(xwmin, xwmax, ywmin, ywmax);
```

其中，双精度浮点数xwmin，xwmax，ywmin，ywmax分别对应裁剪窗口的左、右、下、上四条边界。

默认的裁剪窗口，四条边界分别为wxl=-1.0，wxr=1.0,wyt=-1.0,wyb=1.0。

3.1.3指定视区

- 指定视区

```
glViewPort (xvmin, yvmin, vpWidth, vpHeighht) ;
```

xvmin和yvmin指定了对应于屏幕上显示窗口中的矩形视区的左下角坐标，单位为像素。

整型值vpWidth和vpHeighht则指定了视区的宽度和高度。

默认的视区大小和位置与显示窗口保持一致。

3.2OpenGL中的变换

- 3.2.1变换种类
- 3.2.2模型视图矩阵
- 3.2.3矩阵操作
- 3.2.4矩阵堆栈
- 3.2.5投影变换
- 3.2.6剪切变换

3.2.1变换种类

- 视图变换：指定观察者或摄影机的位置；
- 模型变换：在场景中移动对象；
- 模型视图变换：描述视图变换与模型变换的对偶性；
- 投影变换：对视见空间进行修剪和改变大小；
- 视见区变换：对窗口的最终输出进行缩放；

我们生活在一个三维的世界——如果要观察一个物体，我们可以：

1、从不同的位置去观察它。（视图变换）
2、移动或者旋转它，当然了，如果它只是计算机里面的物体，我们还可以放大或缩小它。（模型变换）
3、如果把物体画下来，我们可以选择：是否需要一种“近大远小”的透视效果。另外，我们可能只希望看到物体的一部分，而不是全部（剪裁）。（投影变换）
4、我们可能希望把整个看到的图形画下来，但它只占据纸张的一部分，而不是全部。（视口变换）

3.2.1变换种类

- 视图变换：指定观察者或摄影机的位置；

视图变换是第一个应用到场景上的变换。它用于确定场景的有利位置。默认情况下，观察点位于原点，顺着z轴的负向看。视图变换使你可以将观察点放置在任何期望的位置上，从任何方向进行观察。确定视图变换相当于在场景中放置一部摄像机并确定其指向。

在工作时间表上，必须先指定视图变换再指定其他任何变换。

3.2.1变换种类

- 模型变换：在场景中移动对象；

模型变换用于处理模型和模型内的特定对象。这些变换把对象移动到合适的位置，旋转它们，并对它们进行缩放。模型变换实际上就是几何变换。

3.2.1变换种类

- 模型视图变换：描述视图变换与模型变换的对偶性；

从内部效果和它们对场景最终外观的效果来说，视图变换和模型变换是相同的，把这两者分开完全是为了程序员的方便，向后移动对象和向前移动参考系之间并没有本质差别。术语“模型视图”表示你可以把这类变换视为模型变换或视图变换，但实际上并无区别，因此称它为模型视图变换。

3.2.1变换种类

- 投影变换：对视见空间进行修剪和改变大小；

投影变换应用到最终的模型视图方向。投影实际上定义了视见空间并建立了修剪平面。更明确一些，投影变换指定已完成场景如何转换成屏幕上的最终图象。常用的投影变换包括 正投影和透视投影。

3.2.1变换种类

- 视见区变换：对窗口的最终输出进行缩放；

把场景的一个二维投影映射到屏幕上某处的窗口。这个到物理窗口坐标的映射是最后完成的一项变换，它称为视见区变换。

3.2.1变换种类

在计算机图形学中，所有的变换都是通过矩阵乘法来实现的，即将物体顶点构成的齐次坐标矩阵乘以三维变换矩阵就得到了变换后的物体顶点的齐次坐标矩阵，这样只要求出物体的三维变换矩阵，就可以得到变换后的物体。

在OpenGL中，对象的坐标变换也通过矩阵来实现的（虽然并不一定要自己定义矩阵）。通常，在OpenGL中使用了两个重要的矩阵：模型视图矩阵和投影矩阵，其中模型视图矩阵用于物体的模型视图变换，投影矩阵用于投影变换。

3.2.1变换种类

有了模型视图矩阵和投影矩阵，在OpenGL中从未加工的顶点数据到屏幕坐标的过程就包括如下步骤：

1. 将顶点坐标转换为规范化齐次坐标矩阵；
2. 将顶点的规范化齐次坐标矩阵乘以模型视图矩阵，生成变换后的顶点齐次坐标矩阵；
3. 将变换后的顶点齐次坐标矩阵乘以投影矩阵，生成修剪坐标矩阵，这样就有效地排除了视见空间之外的所有数据。
4. 由修剪坐标除以w坐标，得到规格化的设备坐标。注意，在变换过程中，模型视图矩阵和投影视图矩阵都有可能改变坐标的 w 值。
5. 坐标三元组被视见区变换影射到一个2D平面。这也是一个矩阵变换，但在OpenGL中不能直接指定或修改它，系统会根据指定给 glViewport 的值在内部设置它。

3.2.2模型视图矩阵

模型视图矩阵用于指定场景的视图变换和几何变换，模型视图矩阵是一个4×4的矩阵，它代表了用来放置和定位对象的经过变换的坐标系。

在计算机图形学中，物体进行的几何变换可以分解成多个基本几何变换，而三维几何变换矩阵是多个基本几何变换矩阵相乘的结果。

OpenGL提供了一些函数用于把任意矩阵设置成当前操作的矩阵（模型视图矩阵或投影矩阵）。

3.2.2模型视图矩阵

```
glfloat m[] = { 1.0f, 0.0f,3.0f,0.0f,
                0.0f,1.0f,0.0f,1.0f,
                0.0f,0.0f,1.0f,1.0f,
                0.0f,0.0f,0.0f,1.0f, };
glMatrixMode(GL_MODELVIEW);
glLoadMatrixf(m);
```

这段程序中，先声明了一个数组来保存 4×4 矩阵的值，注意这里矩阵按列优先顺序保存，这意味着先从上往下遍历每一列；

然后使用glLoadMatrix函数将定义的矩阵设置为当前操作的矩阵。

3.2.2模型视图矩阵

如果需要执行变换，即把定义的矩阵乘到模型视图矩阵中。

可以使用函数 glMultmatrix，其函数原型如下：

```
void glMultMatrix{fd}(const TYPE *m);
```

参数 m 为一个以列优先顺序保存16个连续值的数组。

3.2.2模型视图矩阵

- 平移
- 旋转
- 比例

3.2.2模型视图矩阵

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

□平移

void glTranslate{fd}(TYPE x，TYPE y，TYPE z);

参数x,y,z就是目标分别沿三个轴的正向平移的偏移量。这个函数用这三个偏移量生成的矩阵完成乘法。

当参数是(0.0,0.0,0.0)时，生成的矩阵是单位矩阵，此时对物体没有影响。

3.2.2模型视图矩阵

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

旋转

```
void glRotate{fd}(TYPE angle, TYPE x, TYPE y,
    TYPE z );
```

函数中第一个参数是表示目标沿从原点到指定点(x,y,z)的方向矢量逆时针旋转的角度，后三个参数则是指定旋转方向矢量的坐标。这个函数表示用这四个参数生成的矩阵完成乘法。

当角度参数是 0.0 时，表示对物体没有影响。

3.2.2模型视图矩阵

比例

```
void glScale{fd}(TYPE x, TYPE y, TYPE z);
```

三个参数值就是目标分别沿三个轴向缩放的比例因子。这个函数用这三个比例因子生成的矩阵完成乘法。

这个函数能完成沿相应的轴对目标进行拉伸、压缩和反射三项功能。

当参数是(1.0,1.0,1.0)时，对物体没有影响。

当其中某个参数为负值时，表示将对目标进行相应轴的反射变换，且这个参数不为1.0，则还要进行相应轴的缩放变换。

最好不要令三个参数值都为零，这将导致目标沿三轴都缩为零。

3.2.3矩阵操作

- 矩阵堆栈选择

```
glMatrixMode(GLenum mode);
```

参数mode用于确定将哪个矩阵堆栈用于矩阵操作。

GL_MODELVIEW：模型视图矩阵堆栈

GL_PROJECTION：投影矩阵堆栈

GL_TEXTURE：纹理矩阵堆栈

3.2.3 矩阵操作

通过上述的高级矩阵函数，可以很方便地实现变换。

问题：在调用函数时，修改的是当前的模型视图矩阵。新的矩阵随后将成为当前的模型视图矩阵并影响此后绘制的图形。这样模型视图矩阵函数在调用时，就会造成效果的积累。

3.2.3矩阵操作

- 效果积累

一段代码：

```
//沿 x 轴正向移动 10 个单位
glTranslatef(10.0f, 0.0f, 0.0f);
glutSolidSphere(1.0f, 15, 15);
//沿 y 轴正向移动 10 个单位
glTranslatef(0.0f, 10.0f, 0.0f);
glutSolidSphere(1.0f);
```

这段代码绘制了两个球，第一个绘制的球的球心沿x轴正向移动了10个单位，而第二球不仅沿y轴正向移动10个单位，也沿x轴正向移动10个单位。这就是效果积累。

3.2.5投影变换

- OpenGL中只提供了两种投影方式，一种是平行投影（正射投影），另一种是透视投影。在投影变换之前必须指定当前处理的是投影变换矩阵：

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
```

3.2.5投影变换

- 平行投影

视景体是一个矩形的平行管道，也就是一个长方体，其特点是无论物体距离相机多远，投影后的物体大小尺寸不变。

OpenGL中平行投影函数共有两个：

glOrtho

gluOrtho2D

3.2.5投影变换

void glOrtho (GLdouble left，GLdouble right，

GLdouble bottom，GLdouble top，

GLdouble near，GLdouble far)；

函数的操作是创建一个正射投影矩阵，并且用这个矩阵乘以当前矩阵。

近裁剪平面是一个矩形，矩形左下角点三维空间坐标是（left，bottom，-near），右上角点是（right，top，-near）；

远裁剪平面也是一个矩形，左下角点空间坐标是（left，bottom，-far），右上角点是（right，top，-far）。所有的near和far值同时为正或同时为负。

如果没有其他变换，正射投影的方向平行于Z 轴，且视点朝向Z负轴。这意味着物体在视点前面时far和near都为负值，物体在视点后面时far和near都为正值。

3.2.5投影变换

void gluOrtho2D (GLdouble left，GLdouble right，
GLdouble bottom，GLdouble top)；

一个特殊的正射投影函数，主要用于二维图像到二维屏幕上的投影。

其near和far缺省值分别为-1.0和1.0，所有二维物体的Z坐标都为0.0。

裁剪面是一个左下角点为（left，bottom）、右上角点为（right，top）的矩形。

3.2.5投影变换

- 透视投影

透视投影的特点是距离视点近的物体大，距离视点远的物体小，远到极点即为消失，成为灭点。它的视景体类似于一个顶部和底部都被切除掉的棱椎，也就是棱台。

OpenGL透视投影函数也有两个：

glFrustum

gluPerspective

3.2.5投影变换

```
void glFrustum (GLdouble left, GLdouble right,
                GLdouble bottom, GLdouble top,
                GLdouble near, GLdouble far);
```

此函数创建一个透视投影矩阵，并且用这个矩阵乘以当前矩阵。

参数只定义近裁剪平面的左下角点和右上角点的三维空间坐标，即（left，bottom，-near）和（right，top，-near）；最后一个参数far是远裁剪平面的Z负值，其左下角点和右上角点空间坐标由函数根据透视投影原理自动生成。near和far表示离视点的远近，它们总为正值。

3.2.5投影变换

void gluPerspective (GLdouble fovy, GLdouble aspect,
GLdouble near, GLdouble far);

它也创建一个对称透视视景体，但它的参数定义于前面的不同，其操作是创建一个对称的透视投影矩阵，并且用这个矩阵乘以当前矩阵。

参数fovy定义视野在X-Z平面（垂直方向上的可见区域）的角度，范围是[0.0, 180.0]；参数aspect是投影平面的纵横比（宽度与高度的比值）；参数near和far分别是远近裁剪面沿Z负轴到视点的距离。

3.2.5投影变换