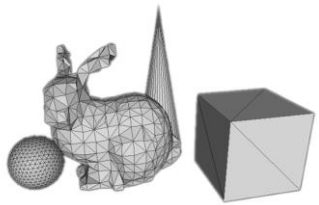
Introduction to Computer Graphics



Programming with OpenGL: 3D dimension and Interaction

上节内容回顾

- OpenGL简介
  - OpenGL是什么
    - 是开放图形程序库，应用程序接口（API），图形硬件的软件接口，不是一种编程语言
  - 从不同角度理解OpenGL
    - 程序员视点（API），状态机（State Machine），图形绘制流水线（Rendering Pipeline)
  - OpenGL的实现方式
    - 软件实现
    - 硬件实现
  - OpenGL三个主要的库及彼此关系
    - GL，GLU，GLUT

上节内容回顾

- OpenGL完整程序
  - 第一个OpenGL程序解析
    - 一个简单的程序，大量使用默认参数
  - OpenGL编程基础知识
    - OpenGL程序的结构：
      - 状态机初始化，回调函数注册，定义回调函数
    - 控制函数
      - GLUT库函数：窗口管理、事件处理循环、回调函数机制
    - 视图
      - 设置相机的内部属性、设置相机的外部属性
    - OpenGL的图元
      - 基本图元设置，用简单图元实现复杂物体
    - 属性
      - 颜色、宽度、实线虚线等

本节内容

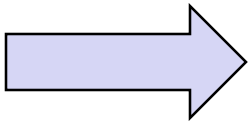
- 三维绘制（3D Rendering)
  - 如何呈现三维效果
  - 设置模型视图矩阵与投影矩阵
  - Zbuffer概念
  - GLUT/GLU相关函数
- 交互（Interaction）
  - GLUT回调函数
  - 菜单
  - 利用异或操作实现橡皮条技术

本节内容

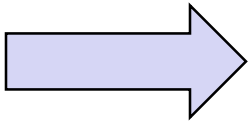
- 三维绘制（3D Rendering)
  - 如何呈现三维效果
  - 设置模型视图矩阵与投影矩阵
  - Zbuffer概念
- 交互（Interaction）
  - GLUT回调函数
  - 菜单
  - 利用异或操作实现橡皮条技术

如何呈现三维效果

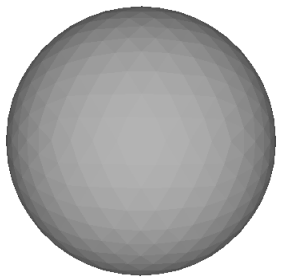
- 在计算机2D屏幕上产生和增强3D效果
  - 3D几何
  - 透视效果——近大远小
  - 隐藏面消除
  - 颜色和着色
  - 光照和阴影
  - 雾效
  - …

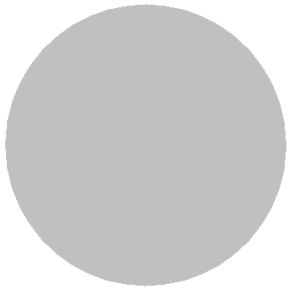


产生3D效果



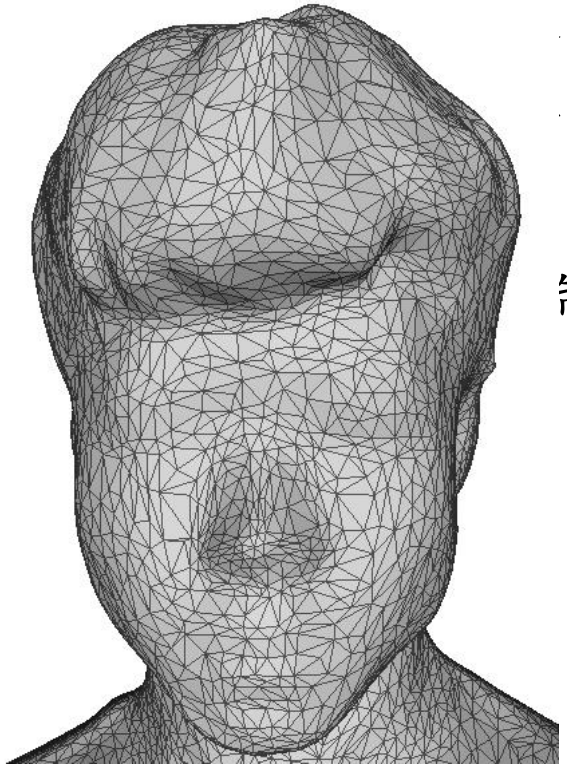
增强3D效果





3D几何

- 基本几何图元
- 用基本几何图元构建



球体.ply

```
ply
format ascii 1.0
comment VCGLIB generated
element vertex 37702
property float x
property float y
property float z
property float nx
property float ny
property float nz
element face 75404
property list uchar int vertex_indices
end_header
-109.85 -585.583 -913.65 -0.990883 0.0898327 -0.1004
-80.8779 -589.489 -965.09 -0.547469 0.203695 -0.811656
-97.4846 -599.833 -853.546 -0.516241 -0.175639 0.83824
-49.3773 -947.479 -786.993 0.475753 0.717751 0.508421
-97.3486 -588.504 -851.651 -0.529533 -0.183035 0.828307
-85.3551 -596.765 -847.027 -0.382764 -0.209772 0.899715
-30.7061 -959.997 -794.619 0.662137 0.512276 0.546944
-103.878 -957.321 -787.054 -0.565117 0.781235 0.265169
-91.6018 -948.06 -791.949 -0.465062 0.861006 0.205878
-83.9699 -946.093 -787.644 -0.324549 0.912479 0.249098
-81.7526 -943.729 -792.817 -0.296337 0.935378 0.193007
-27.2919 -593.142 -857.568 0.42749 -0.321394 0.84496
-26.6596 -586.995 -855.36 0.457365 -0.374257 0.80669
-45.2669 -594.426 -976.194 0.345885 0.188408 -0.919166
11.3998 -588.406 -906.555 0.992747 0.119602 -0.0121996
-11.6815 -590.988 -867.936 0.672804 -0.178577 0.717945
```

3D几何

- 基本几何图元
- 用基本几何图元构建
  - 稍微简单物体，编程构建：球体等
  - 复杂物体：通过建模软件构建保存成模型文件，例如Ply，Obj等模型格式
  - Task: 读入ply文件，并绘制模型
- GLU/GLUT对象

GLU和GLUT对象

- GLU和GLUT提供了一些高级几何对象
- 这些高级几何对象主要是一些二次曲面
  - 椭球体
  - 圆锥体
  - 圆柱体
  - ...
- 这些高级几何对象仍然是由多边形图元构成的。
- 用途：主要用于在开发阶段测试程序或者简单展示
- 自学：GLU中的二次曲面对象

GLU和GLUT对象

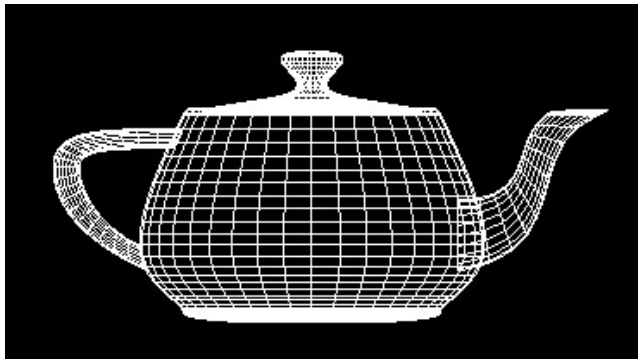
- GLUT中的高级几何对象
  - GLUT在GLU的基础上增加了更多类型(Type)并更易使用的三维对象
    - 立方体
    - 球体(Sphere)
    - 圆锥(Cone)
    - 圆环(Torus)
    - 规则多面体(Tetrahedron、Octahedron、Dodecahedron)
    - 犹他壶(Teapot)
  - GLUT为每种类型的对象提供了线框图或者实体图（填充多边形）两种绘制函数，格式为：
    - glutWire*Type*()
    - glutSolid*Type*()

GLU和GLUT对象

- 举例1：绘制球体
  - void glutWireSphere(GLdouble radius, GLint slices, GLint stacks)
  - void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks)
  - 其中：
    - radius——球体半径
    - slices——经线数
    - stacks——纬线数
    - 球心位于坐标原点

GLU和GLUT对象

- 举例2：绘制犹他壶
  - 三维图形领域最著名的几何对象
  - 广泛用于测试各种绘制算法
  - void glutWireTeapot(GLdouble size)
  - void glutSolidTeapot(GLdouble size)
  - 其中：
    - size——犹他壶的尺寸
    - 犹他壶中心位于原点
    - 由192个顶点构成



本节内容

- 三维绘制（3D Rendering)
  - 如何呈现三维效果
  - 设置模型视图矩阵与投影矩阵
  - Zbuffer概念
- 输入与交互
  - GLUT回调函数
  - 菜单
  - 利用异或操作实现橡皮条技术

设置模型视图矩阵(ModelView Matrix)与投影矩阵(Projection Matrix)

- 成像要素
  - 相机相对物体在哪里拍摄
  - 相机是何种参数相机：是否广角，胶片多大，相机能看多远等等
- 模型视图矩阵
  - 用于描述物体与相机之间的相对位置、方位关系
- 投影矩阵
  - 用于描述相机内部参数

OpenGL与合成摄像机模型

- 要让OpenGL变换能够正确的模拟真实摄像机的成像过程，就需要提供正确的变换矩阵（如投影矩阵、模型-视图矩阵）。

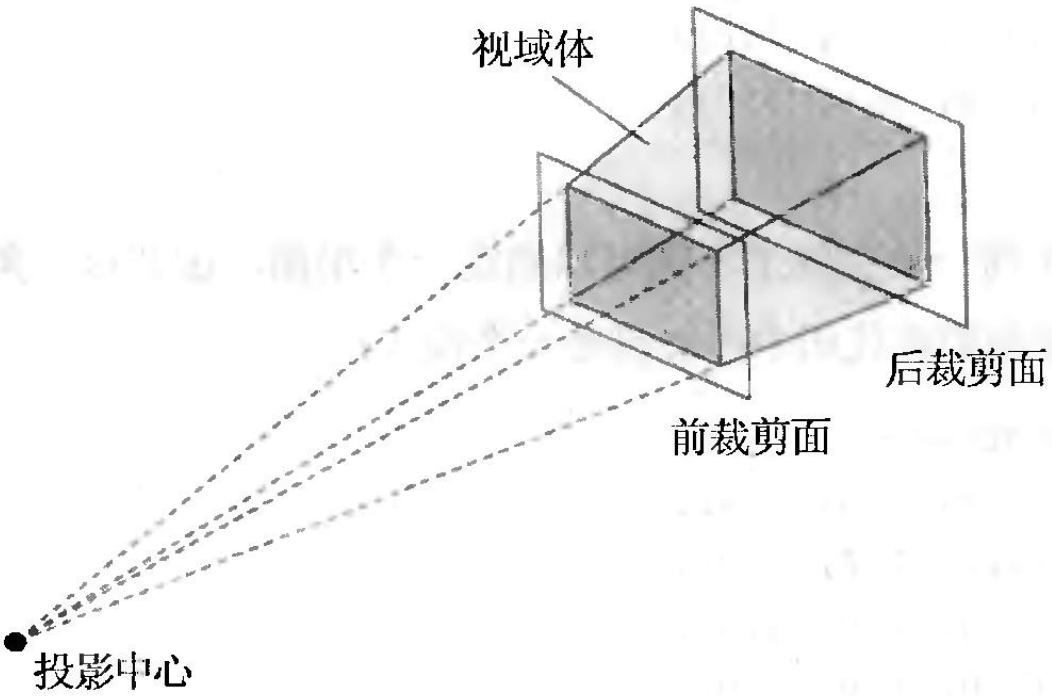
- 这些矩阵在OpenGL内部都是预定义的状态变量；

- 如何提供正确的变换矩阵？

  ——OpenGL提供了与投影变换、模型变换、视图变换等相关的API函数来自动建立和修改对应的矩阵。

OpenGL与合成摄像机模型

- OpenGL如何模拟投影过程？
  - 确定哪些对象可见，哪些对象不可见？——建立视域体
  - 确定对象在投影平面上的位置？——正交投影、透视投影



OpenGL与合成摄像机模型

- OpenGL与定位、投影过程的影响因素
  - 被观察对象
    - 位置（三个自由度）
    - 朝向（三个自由度）
  - 摄像机
    - 位置（三个自由度）
    - 朝向（三个自由度）
    - 透镜参数（长焦、广角、…）
    - 底片尺寸

glVertex*();
glTranslate*()
glRotate*()

摄像机定位、定向和调焦对准
gluLookAt()

摄像机镜头投影
glOrtho()
gluOrtho2D()
glFrustum()
gluPerspective()

OpenGL 虚拟摄像机的定位与定向

- 何为定位和定向？

  ——确定摄像机与被观察对象之间的相对位置和朝向

- 定位和定向方法一 —— 相对于被观察对象调整摄像机
  - 可以使用视图变换gluLookAt()来定位和定向；
  - gluLookAt()将修改OpenGL内部的模型-视图矩阵；
  - void gluLookAt (

    GLdouble eyex, GLdouble eyey, GLdouble eyez,

    GLdouble atx, GLdouble aty, GLdouble atz,

    GLdouble upx, GLdouble upy, GLdouble upz)

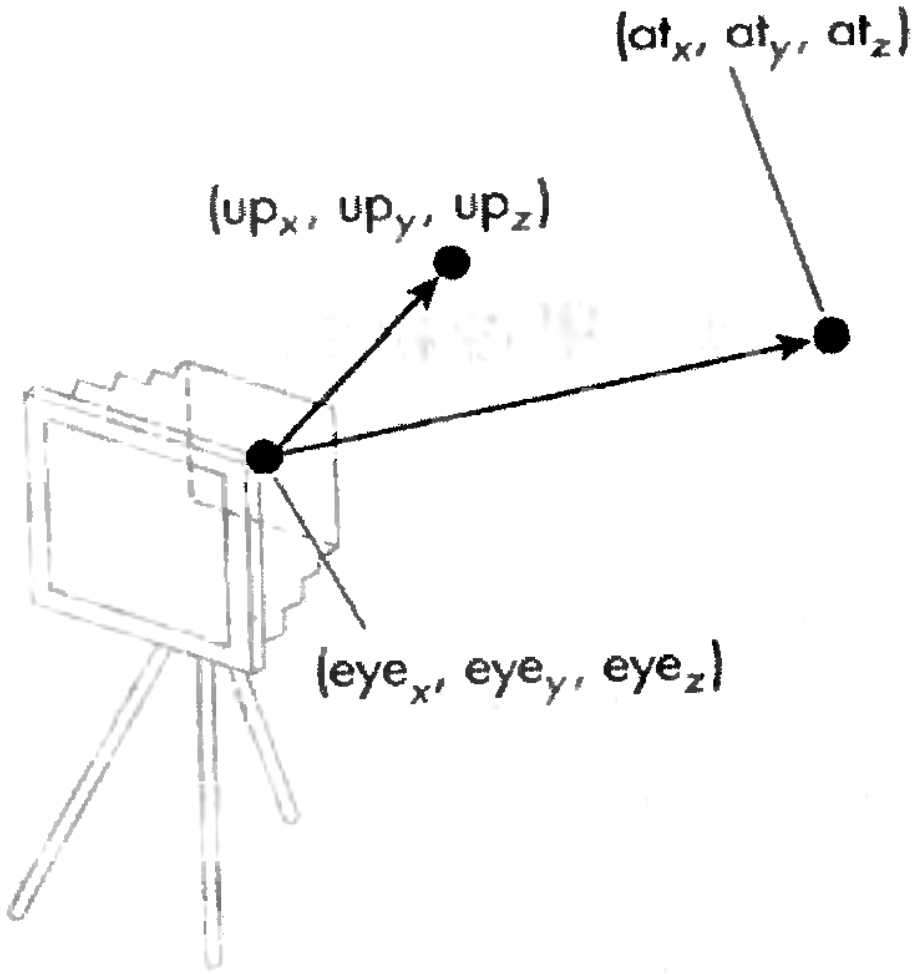
  其中： (eyex, eyey, eyez)——视点

  (atx, aty, atz)—— 被观察点

  (upx, upy, upz) —— 向上向量

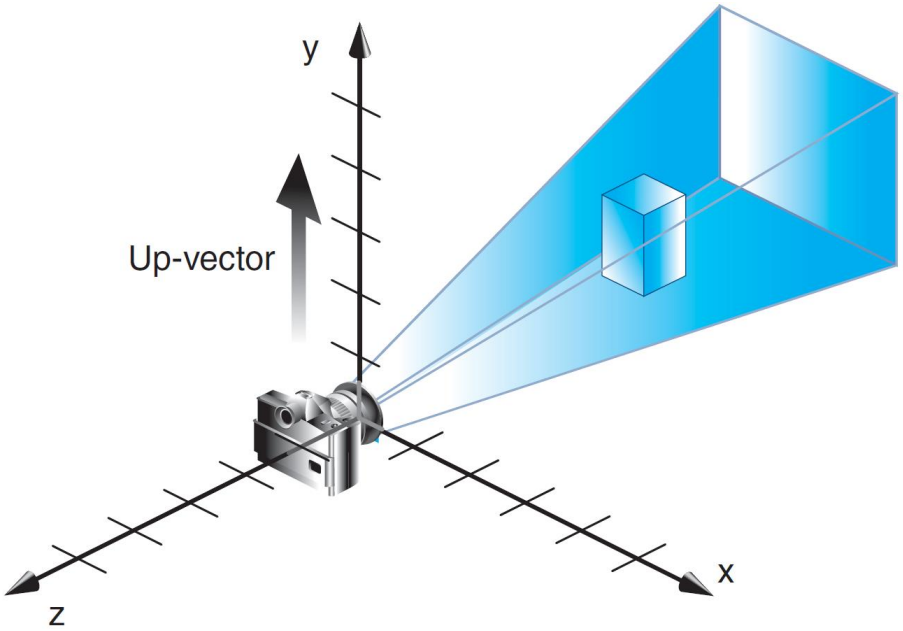
OpenGL 虚拟摄像机的定位与定向

- gluLookAt()的几何意义

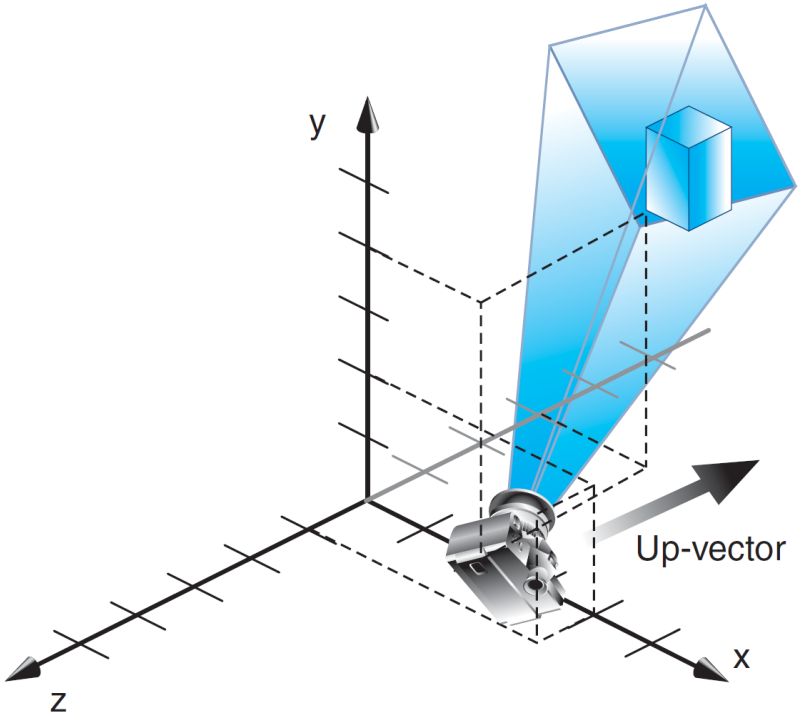


OpenGL 虚拟摄像机的定位与定向

- gluLookAt()的几何意义



默认的摄像机位置



利用gluLookAt对摄像机进行对位和定向

OpenGL 虚拟摄像机的定位和定向

- 定位和定向方法二 —— 相对于摄像机调整被观察对象
  - 可以使用与模型变换（平移、旋转和缩放）相关的API函数
  - void glTranslate<fd>(type dx, type dy, type dz)
  - void glRotate<fd>(type angle, type dx, type dy, type dz)
  - void glScale<fd>(type sx, type sy, type sz)

其中： (dx, dy, dz)——平移向量

angle——旋转角度

(sx，sy, sz) —— 放缩向量

OpenGL 虚拟摄像机的定位和定向

- 总结：实现定位和定向的原理——模型变换、视图变换
- OpenGL实现定位和定向的途径——设置模型-视图矩阵
- 为什么OpenGL不单独设置模型矩阵和视图矩阵两种状态变量？
  - 原因：上述两种定位和定向方法（分别对应视图变换和模型变换）从根本上是统一的。
- 用户可以根据需要和便利性灵活选用模型变换、视图变换函数来实现定位和定向。

OpenGL 虚拟摄像机的定位和定向

- OpenGL中如何设置和修改模型-视图矩阵

```
   ...

glMatrixMode (GL_MODELVIEW);

glLoadIdentity();

gluLookAt( 1.0, 1.0, 1.0,    // 视点位置
           0.0, 0.0, 0.0,    // 被观察点位置
           0.0, 1.0, 0.0   ); // 向上向量
//glTranslatef(1.0, 0.0, 0.0);
glBegin(...)
   .
   .
   .
glEnd()

   ...
```

指定当前矩阵为模型-视图矩阵

将当前矩阵设置为单位矩阵（初始化）

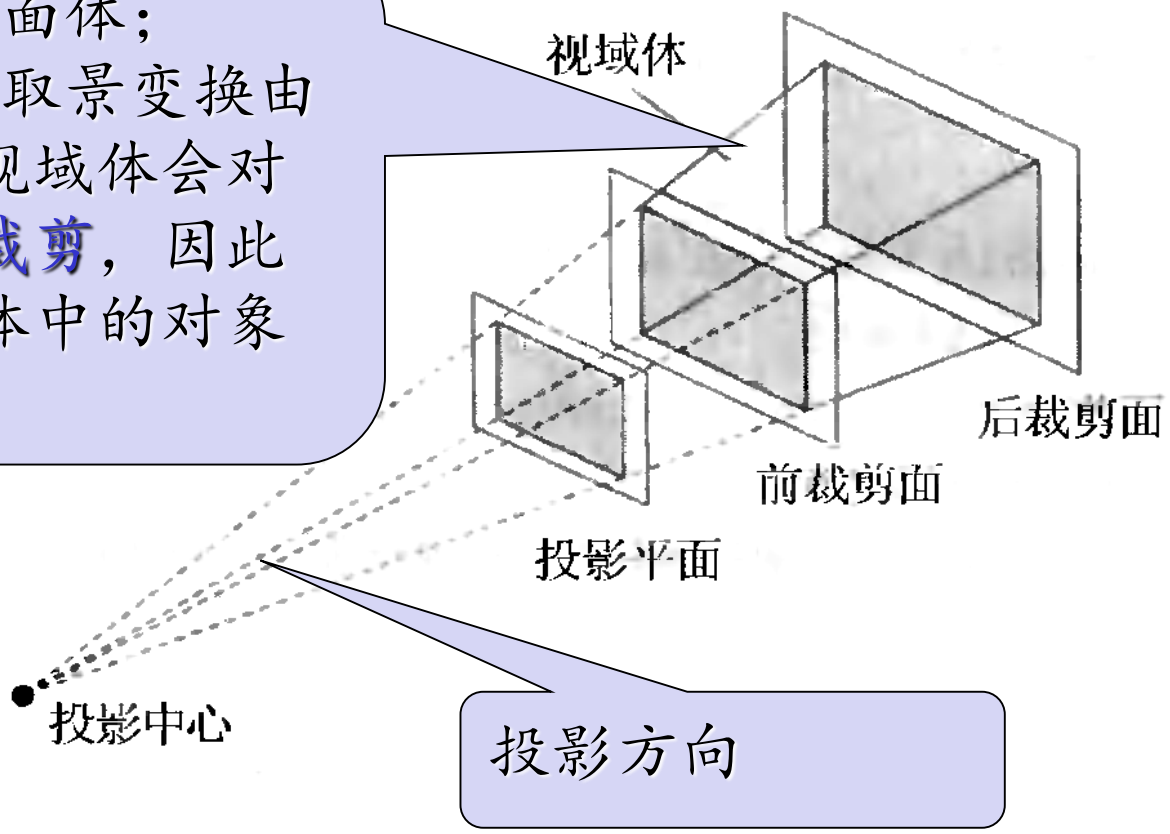
修改当前矩阵为我们期望的矩阵

提供几何图元顶点在世界坐标系中的位置

OpenGL 虚拟摄像机的投影取景

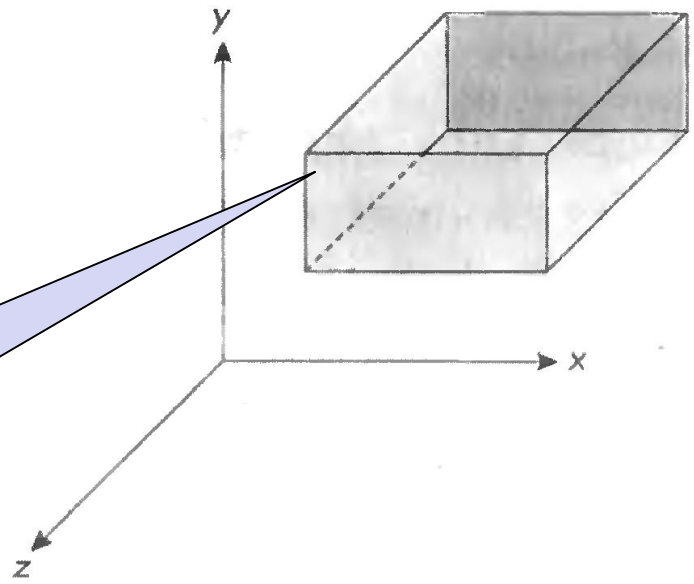
- OpenGL投影取景的实现——设置视域体（视域四棱锥）

1. 视域体为六面体；
2. 摄像机投影取景变换由视域体决定，视域体会对三维场景进行**裁剪**，因此只有位于视域体中的对象才能被观察到。



OpenGL 虚拟摄像机的投影取景

- OpenGL提供两种投影取景方式——正交投影、透视投影
- 正交投影——物距无穷大，可以理解为投影中心在无穷远处
  - 模拟长焦镜头
  - 优点：保持距离和形状
  - 缺点：缺乏立体感（没有透视效果



正交投影的视域体为平行六面体，也即长方体；

OpenGL 虚拟摄像机的投影取景

- OpenGL的正交投影变换
  - 通过六个参数设定正交视域体
  - void glOrtho (GLdouble left, GLdouble right,
    GLdoublt bottom, GLdouble top,
    GLdouble near, GLdouble far)

其中：

参数必须满足的约束条件：

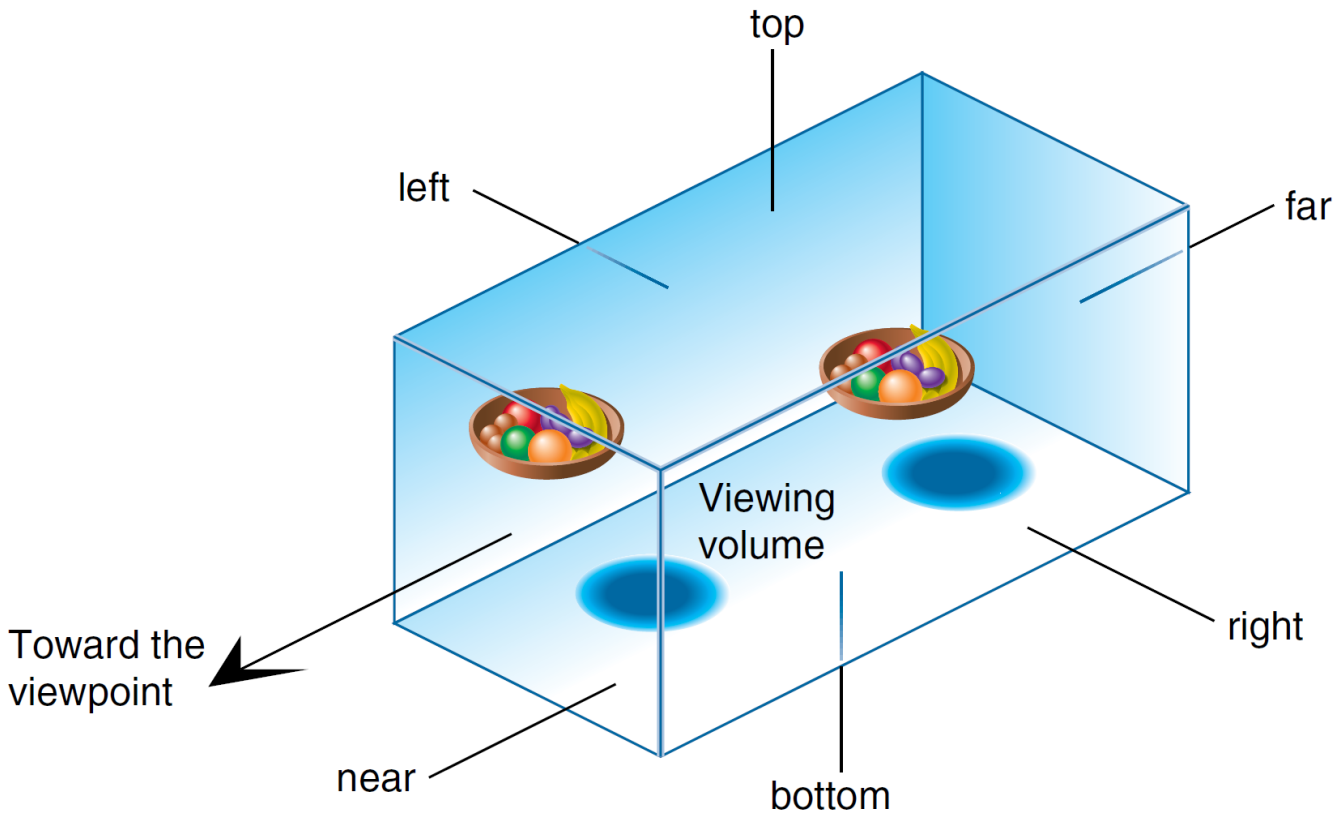
right > left , top > bottom, far > near

glOrtho的二维简化版本 (相当于near = -1, far = 1)

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

OpenGL 虚拟摄像机的投影取景

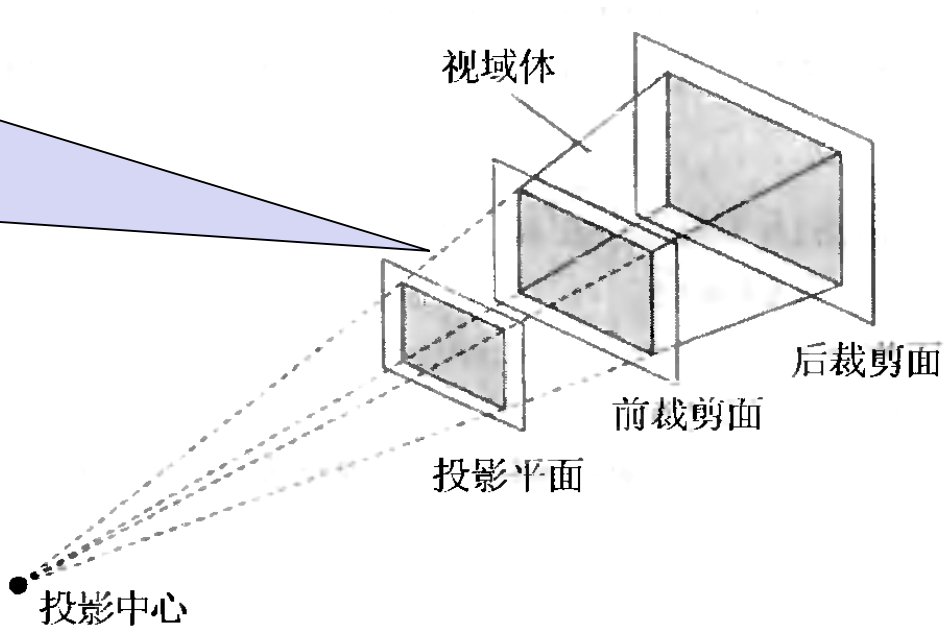
- glOrtho()中各参数的几何意义



OpenGL 虚拟摄像机的投影取景

- OpenGL的透视投影变换
  - 模拟各种透视镜头（广角、...）
  - 优点：具有立体感（透视效果）
  - 缺点：不保持距离和形状（近大远小）

1. 投影平面可以设置在投影中心与前裁剪面之间的任何位置
2. 投影中心通常也称为视点



OpenGL 虚拟摄像机的投影取景

- OpenGL的透视投影变换
  - 通过六个参数设定透视视域体
  - void glFrustum (GLdouble left, GLdouble right,
    GLdoublt bottom, GLdouble top,
    GLdouble near, GLdouble far)

  其中：

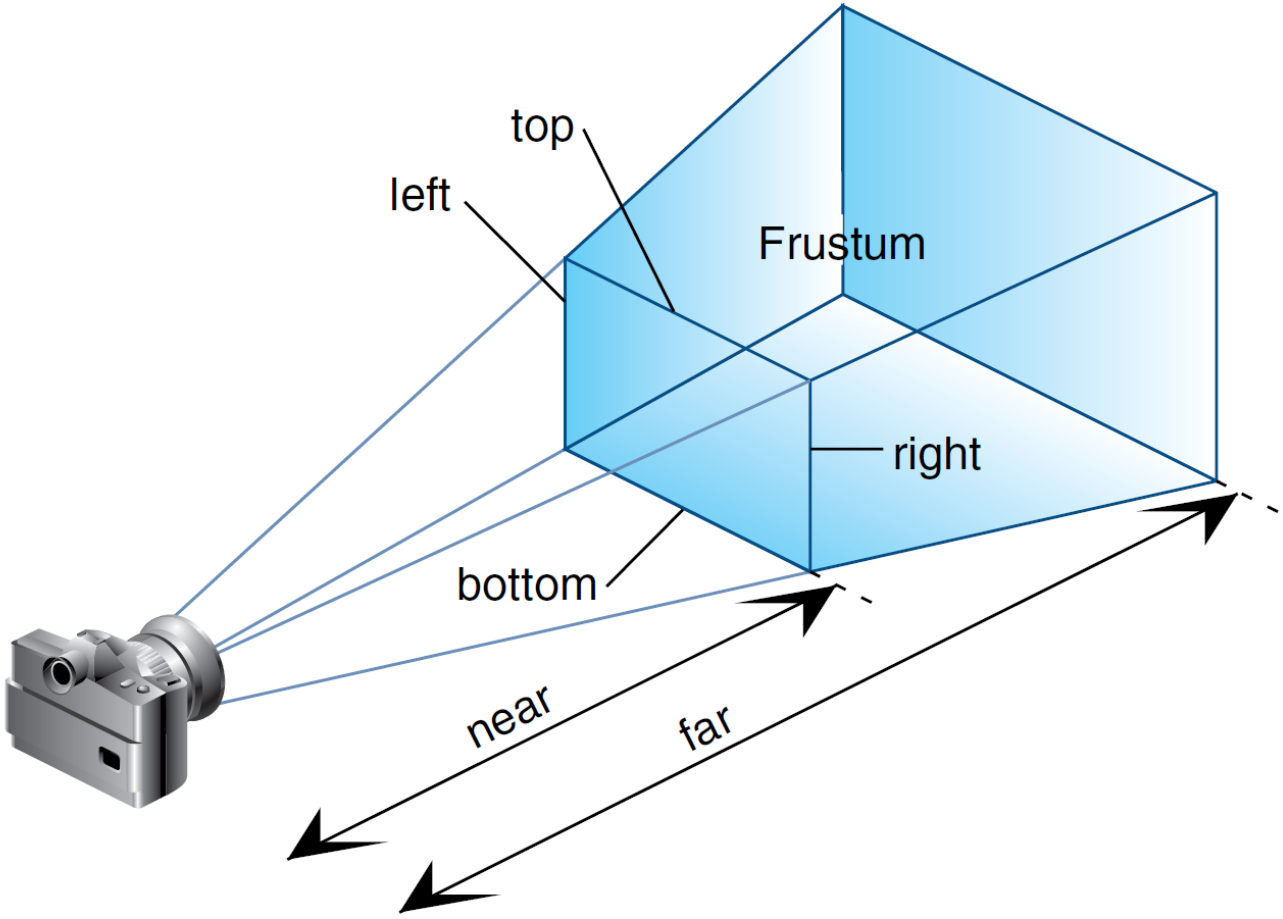
  1. 前裁剪面和后裁剪面分别由near和far确定，这些值是相对于投影中心来度量的， 且满足 far > near >0

  2. 前裁剪面的左下角和右上角在摄像机坐标系中的坐标分别为（left, bottom, near）和 (right, top, near)。

- 注意：正交投影中，摄像机可以位于裁剪体中，而在透视投影中，视域体必须位于摄像机前面。

OpenGL 虚拟摄像机的投影取景

- glFrustum()中各参数的几何意义

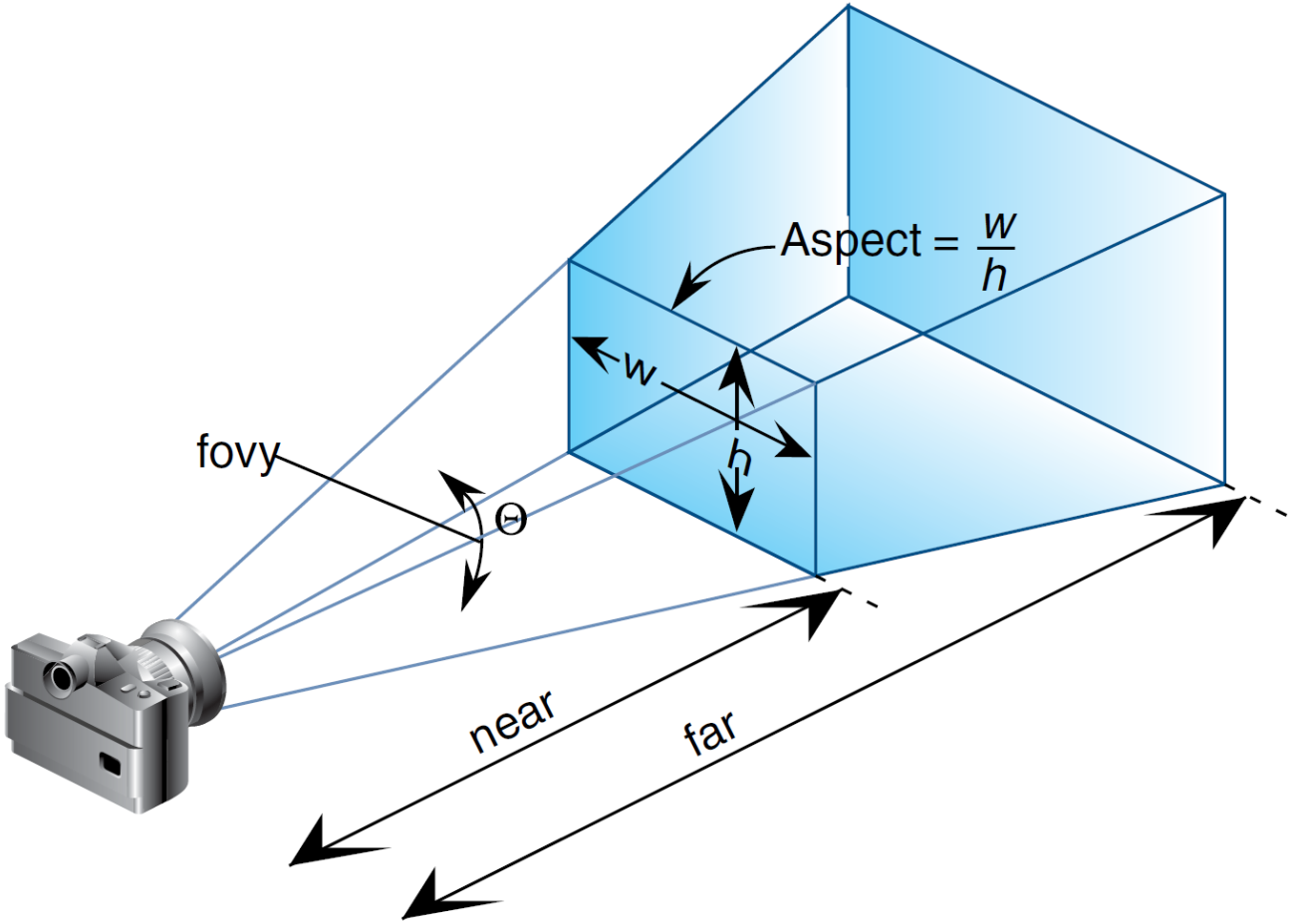


OpenGL 虚拟摄像机的投影取景

- glFrustum()是设置透视视域体最通用的方法
- GLU库在glFrustum()基础上提供了另一个辅助函数来简化设置视域体
  - 通过四个参数设定透视视域体
  - void gluPerspective (GLdouble fovy, GLdouble Aspect,
    GLdouble near, GLdouble far)
  - 相比glFrustum()，gluPerspective()所设置的透视视域体更为特殊，但却更为常用；
  - 在应用程序中是选择glFrustum()还是gluPerspective()来设置视域体要根据应用对透视视域体的要求而定。

OpenGL 虚拟摄像机的投影取景

- gluPerspective()中各参数的几何意义



OpenGL 虚拟摄像机的投影取景

- OpenGL实现投影变换的途径——设置投影矩阵

```
   ...

glMatrixMode (GL_PROJECTION);

glLoadIdentity();

//等价于gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);

   ...
```

指定当前矩阵为投影矩阵

将当前矩阵设置为单位矩阵（初始化）

修改当前矩阵为我们期望的投影矩阵

本节内容

- 三维绘制（3D Rendering)
  - 如何呈现三维效果
  - 设置模型视图矩阵与投影矩阵
  - 隐藏面消除概念
- 交互（Interaction）
  - GLUT回调函数
  - 菜单
  - 利用异或操作实现橡皮条技术

隐藏面消除

- 消除不可见表面的两种途径
- 第一种途径：多边形拣选
- 目的：将多边形图元的不可见表面剔除
- 例子：启用多边形拣选并剔除多边形的反面（当反面是不可见表面）
  - glEnable(GL_CULL_FACE)
  - glCullFace(GL_BACK)

隐藏面消除

- 第二种途径：深度缓存算法（Zbuffer）
- 目的：沿着投影方向，距离摄像机近的几何对象应该遮挡距离摄像机更远的几何对象
- 算法原理：
- 1. 增加额外的深度缓存空间来保存在绘制过程中绘制器已经绘制图元的深度信息；
- 2. 依据深度值比较结果，判断当前图元的颜色信息是否更新颜色缓存中对应像素的颜色值：若图元离投影机更近则更新颜色缓存，否则图元颜色将被丢弃。

隐藏面消除

- 结论：

1. 启用OpenGL深度缓存功能后，离摄像机更近的几何对象的绘制结果将保留在颜色缓存中，并最终可见；

2. 不启用OpenGL深度缓存缓存功能（默认），那么几何图元的绘制顺序（即程序中提交图元的顺序）将决定最终在屏幕上是否可见。

隐藏面消除

- 使用方法（三步）：

1.在main()中的窗口初始化阶段，请求一个深度缓存

```
glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |
                    GLUT_DEPTH)
```

2.在init()中OpenGL的初始化阶段，启用深度缓存功能

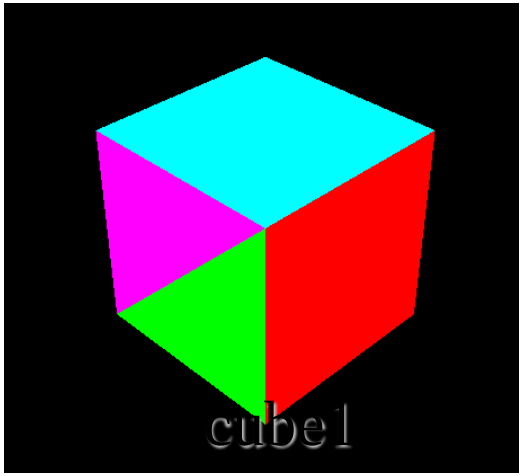
```
glEnable(GL_DEPTH_TEST)
```

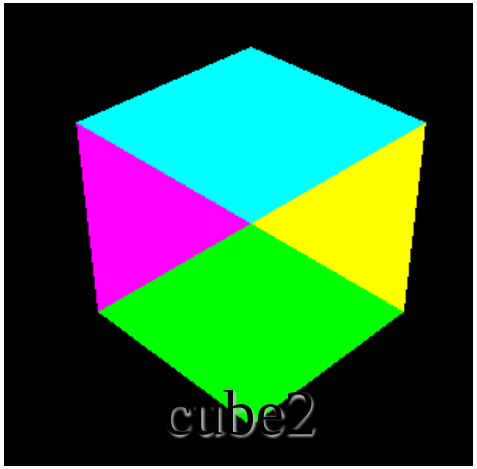
3.在display()中绘制开始前，清空颜色缓存的同时，通常也清空深度缓存

```
glClear(GL_COLOR_BUFFER_BIT |
        GL_DEPTH_BUFFER_BIT)
```

多边形拣选VS深度缓存

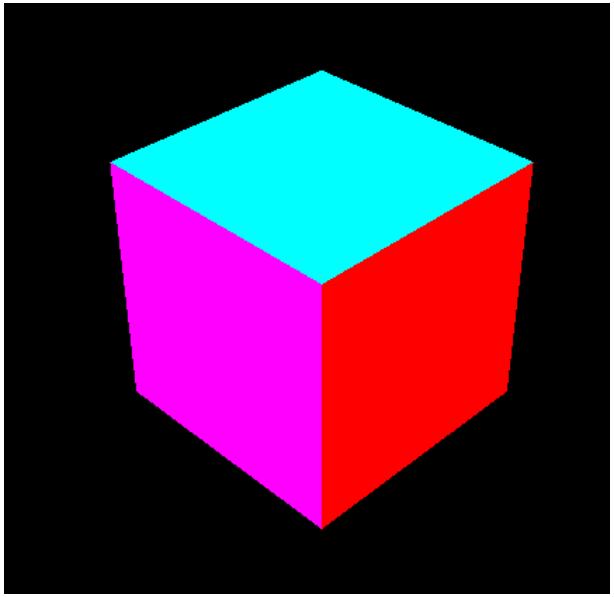
- CullFace VS DepthTest
- CullFace 和 DepthTest 功能都关闭



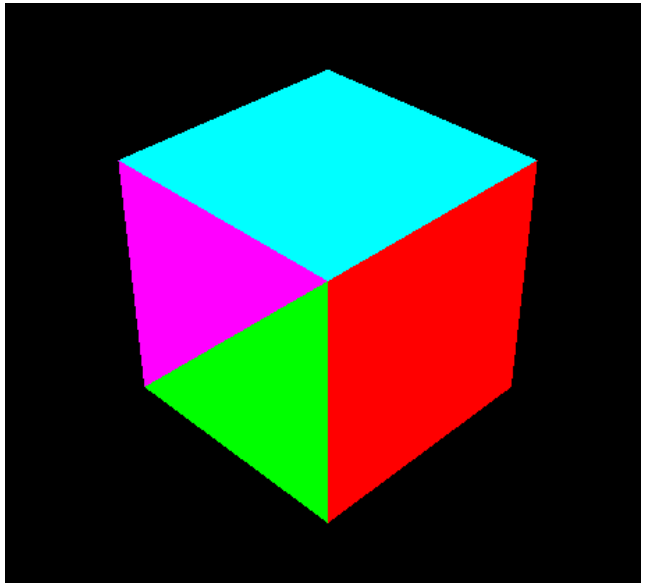


多边形拣选VS深度缓存

- CullFace 和 DepthTest 功能任一开启



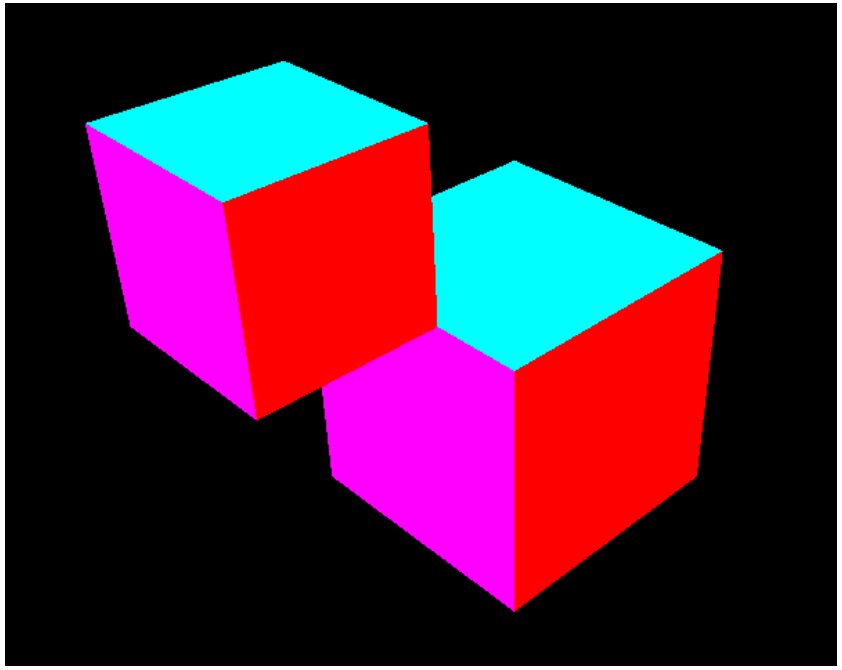
开启



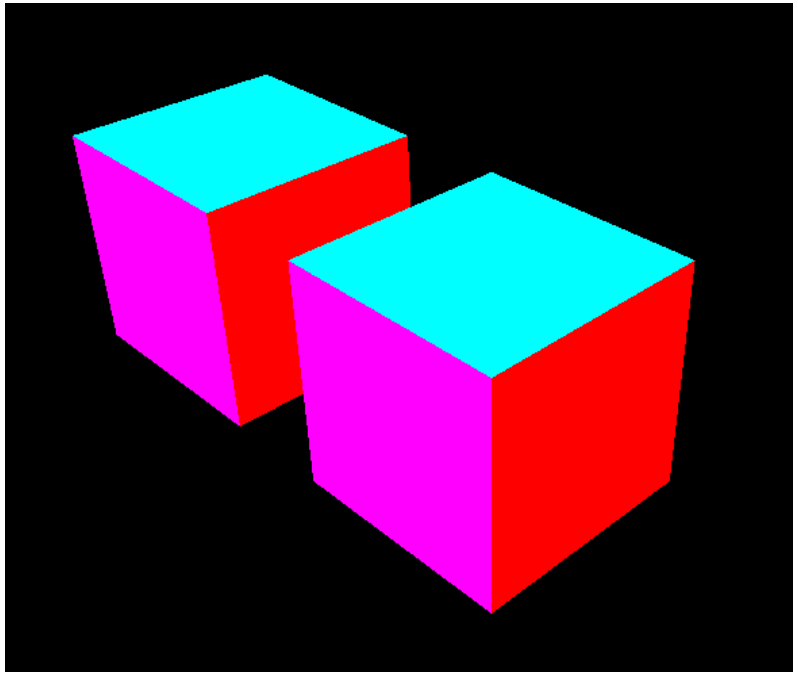
未开启

多边形拣选VS深度缓存

- CullFace VS DepthTest



CullFace



DepthTest

本节内容

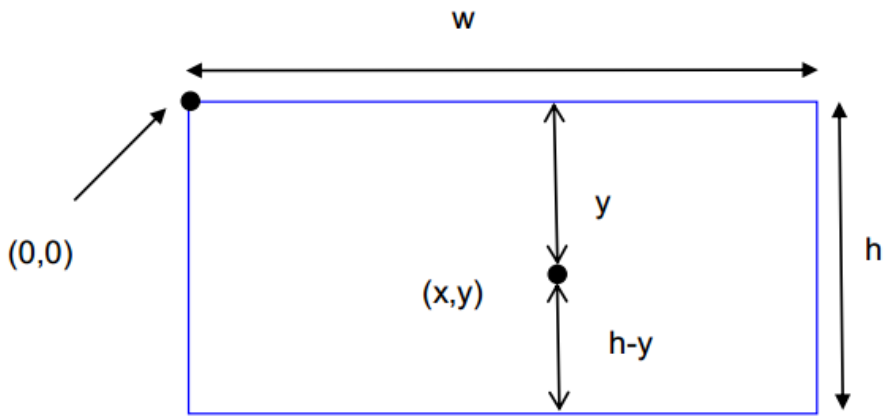
- 三维绘制（3D Rendering)
  - 如何呈现三维效果
  - 设置模型视图矩阵与投影矩阵
  - 隐藏面消除概念
- 交互（Interaction）
  - GLUT回调函数
  - 菜单

鼠标回调函数

- **glutMouseFunc(mouse)**
  void mouse(int button, int state, int x, int y)
  - 其中button的值可能是GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, GLUT_RIGHT_BUTTON表示哪个按钮导致了事件发生
  - state表示相应按钮的状态：GLUT_UP, GLUT_DOWN
  - x, y表示在窗口中的位置 (以窗口左上角为原点的坐标)

鼠标定位

- 在屏幕上的位置通常是以像素为单位的，原点在左上角 –因为显示器自顶向下刷新显示内容
- 在OpenGL中使用的世界坐标系，其原点在左下角
- 在这个坐标系中的y坐标需要从窗口高度中减去回调函数返回的y值： y = h - y



获取窗口尺寸

- 为了完成y坐标的转换，需要知道窗口的尺寸：在程序执行过程中，高度可能发生改变
  - 需要利用一个全局变量跟踪其变化
  - 新高度值返回给形状改变回调函数(见后)
  - 也可以用查询函数glGetIntegerv()和glGetFloatv() 获取，因为高度是状态的一部分

  例如：glGetIntegerv (GL_VIEWPORT, viewport)

  将视口大小传入viewport数组中

结束程序

- 在以前的程序中没有办法通过OpenGL结束当前程序
- 可以利用简单的鼠标回调函数做到这一点

```
void mouse(int btn, int state, int x, int y)
{
        if(btn==GLUT_RIGHT_BUTTON &&
        state==GLUT_DOWN)
        exit(0);
}
```

在指针处画方框

```
void mouse(int btn, int state, int x, int y) {
        if(btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
        exit(0);
        if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        drawSquare(x, y);
}

void drawSquare(GLint x, GLint y) {
        y = h-y; //转化y坐标
        glColor3ub( (char)rand()%256, (char) rand()%256,
        (char)rand()%256); // 随机颜色
        glBegin(GL_POLYGON);
                glVertex2f(x - size, y - size);
                glVertex2f(x + size, y - size);
                glVertex2f(x + size, y + size);
                glVertex2f(x - size, y + size);
        glEnd();
}
```

鼠标移动回调函数的应用

- 通过利用移动回调函数可以在不释放鼠标按钮的情况下，连续画一系列方框(square.c)
  - **glutMotionFunc(drawSquare)**
- 应用被动移动回调函数，可以不用按鼠标按钮就可以连续画方框
  - **glutPassiveMotionFunc(drawSquare)**

- **void glutMotionFunc(void (*f)(int x,int y))**
- **void glutPassiveMotionFunc(void (*f)(int x,int y))**

键盘的使用

- 当鼠标位于窗口内，并且键盘有某个键被按下或释放，就会产生键盘事件

```
glutKeyboardFunc(keyboard)
void keyboard(unsigned char key,
int x, int y)
```

  - 返回键盘上被按下键的ASCII码和鼠标位置
  - 注意在GLUT中并不把释放键做为一个事件

```
void keyboard(unsigned char key, int x, int y)
{ // 按下Q、 q或ESC键时，终止程序
        if(key == 'Q' || key == 'q' || key == '\27')
        exit(0);
}
```

特殊按键

- GLUT在glut.h中定义了特殊按键：
  - 功能键1: GLUT_KEY_F1
  - 向上方向键： GLUT_KEY_UP

    void glutSpecialFunc(void (*func)(int key, int x, int y))
  - 回调函数内

    if(key == GLUT_KEY_F1) ……

    if(key == GLUT_KEY_UP) ……

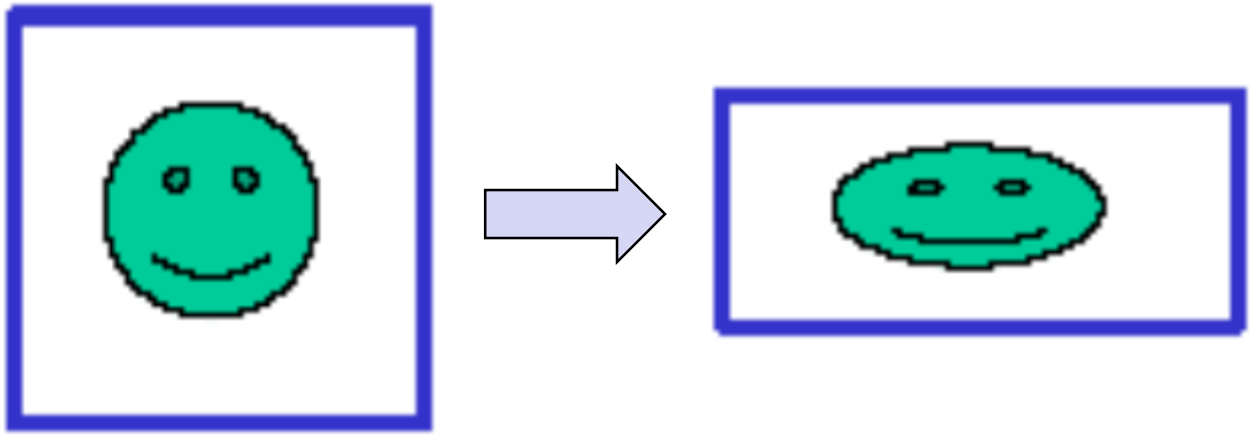
修饰键 – Ctrl/Alt/Shift

- **int glutGetModifiers()**
  - 如果在鼠标或键盘事件产生时修饰键被按下，该函数返回GLUT_ACTIVE_SHIFT, GLUT_ACTIVE_CTRL, GLUT_ACTIVE_ALT的逻辑与结果
  - 例子：让Ctrl+c或Ctrl+C终止程序，在键盘回调函数中

```
if (glutGetModifiers() == GLUT_ACTIVE_CTRL)
&&(key == 'c' || key == 'C'))
exit(0);
```

改变窗口大小

- 通过拖动窗口的角点可以改变窗口的形状和尺寸
  - 那么其中的显示内容该如何处理?
  - 必须由应用程序重新绘制
  - 结果可能是



形状改变回调函数

- **glutReshapeFunc(reshape)**
  void reshape(int w, int h)
  – 返回新窗口的宽度与高度（单位：像素）
  – 回调函数执行后自动发送刷新显示事件，触发显示回调
  – GLUT有一个缺省的形状改变的回调函数，调用glViewport(0,0,w,h)把视口设置为新窗口
- **这个回调函数是放置照相机函数的恰当地方，因为当窗口首次创建时**就会调用它。
  – 当窗口形状尺寸发生改变时，可在回调函数里对观察条件进行修改

例子

- 通过保持视口和世界窗口的长宽比一样，使得物体不变形

```
void reshape(int w, int h) {
        glViewport(0, 0, w, h); // 设置视口为整个窗口
        glMatrixMode(GL_PROJECTION); // 调整裁剪窗口
        glLoadIdentity();
        if(w<=h) // 宽小于高，裁剪矩形宽度置为4
                gluOrtho2D(-2.0, 2.0, -2.0*(GLfloat)h/(GLfloat)w,
                2.0*(GLfloat)h/(GLfloat)w);
        else // 高小于宽，裁剪矩形高度置为4
                gluOrtho2D(-2.0*(GLfloat)w/(GLfloat)h,
                2.0*(GLfloat)w/(GLfloat)h, -2.0, 2.0);
        glMatrixMode(GL_MODELVIEW); /*return to modelview
mode*/
}
```

本节内容

- 三维绘制（3D Rendering)
  - 如何呈现三维效果
  - 设置模型视图矩阵与投影矩阵
  - 隐藏面消除概念
- 交互（Interaction）
  - GLUT回调函数
  - 菜单

菜单

- GLUT支持弹出式菜单
  - 可以有子菜单
- 创建弹出式菜单的三个步骤
  - 定义菜单内各条目
  - 定义每个菜单项的行为
    - 如果条目被选择执行的操作
  - 把菜单连接到鼠标按钮上

菜单函数

- **int glutCreateMenu(void (*f)(int value))**
  – 创建一个使用回调函数f()的菜单，并返回菜单的整数标识符
- **void glutAddMenuEntry(char *name, int value)**
  – 为当前菜单增加一个名为name的菜单项； value值在选中时返回给菜单回调函数
- **void glutAttachMenu(int button)**
  – 将当前菜单关联到鼠标按钮button上 (GLUT_RIGHT_BUTTON、 GLUT_MIDDLE_BUTTON、 GLUT_LEFT_BUTTON)

定义一个简单的菜单

- 在main()函数中

会作为参数返回给回调函数mymenu

```
menu_id = glutCreateMenu(mymenu);
glutAddMenuEntry("Clear Screen", 1);
glutAddMenuEntry("Exit", 2);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

菜单回调函数

```
void mymenu(int value) {
        if(value==1)
        glClear(GL_COLOR_BUFFER_BIT);
        if(value==2) exit(0);
}
```