# 第0章 导论

## 0.1视觉交流与计算机图形学

Richard Hamming: 计算的目的是洞察事物的本质，而不是获得数字。

计算机图形学的目的是获得信息，而不是图像本身。

## 0.2视觉交流的基本概念

使用合适的信息表达方式、图像应突出重点、使用合适的信息展示级别、采用合适的信息格式、注意图像显示的准确性、理解并尊重观众的文化背景、使用交互成为用户熟悉的高效操作

## 0.3三维几何和几何流水线

1. 场景和视图
2. 三维世界坐标系
3. 三维眼坐标系
4. 投影：正交投影和透视投影
5. 裁剪
6. 选择透视投影或正交投影
7. 二维眼坐标：隐藏面剔除技术
8. 三维屏幕坐标：窗口到视口的映射、屏幕坐标（显示坐标）



## 0.4外观属性

对象的外观(appearance)属性：顶点的坐标、顶点深度值、顶点颜色（RGB）、顶点法向量、顶点材质、顶点纹理坐标

## 0.5观察过程

模型视图变换(model view transformation)

## 0.6图形卡

加速器

图形卡与图形API的发展相互促进。

## 0.7一个简单的OpenGL程序

GLUT (Graphics Library Utility Toolkit)

GLUT完全通过事件来操作。对程序需要处理的每个事件，都需要在main()函数中定义相应的回调函数。回调函数是当相关事件发生时，系统事件处理程序调用的函数。改变窗口（Reshape）事件生成窗口，显示（display）事件调用自身的回调函数在窗口中绘制初始图像。空闲（idle）回调函数在系统空闲时间（当系统不生成图像或相应其他事件时）重新计算图像，并在终端上显示改变的图像。

#include <GL/glut.h>

// 其他需要的头文件

// 所需要的全局数据块和类型定义

// 函数声明

void doMyInit(void);

void display(void);

void reshape(int,int);

void idle(void);

// 其他函数声明

// 初始化函数

void doMyInit(void){

设置基本的OpenGL参数和环境

设置投影转换（正交投影或透视投影）

}

// 改变窗口回调函数

void reshape(int w, int h){

使用新的窗口维度w和h，设置投影转换

再显示

}

// 显示回调函数

void display(void){

设置定义几何、转换和需要再显示的外观的视口转换

}

// 空闲回调函数

void idle(void){

更新、post redisplay

}

// 主函数—建立系统，将控制权交给事件处理程序

void main(int argc, char \*\* argv){

// 通过GLUT初始化系统和自己的初始化工作

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(windW, windH);

glutInitWindowPosition(topLeftX, topLeftY);

glutCreateWindow(“A simple Program”);

doMyInit();

// 定义事件回调函数

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutIdleFunc(idle);

// 进入主事件循环

glutMainLoop();

}

两种方法理解代码：（1）几何流水线；（2）程序功能：程序是事件驱动，所有回调函数都不是程序直接调用的。

## 本章OpenGL术语表

类型

Glfloat: 和系统无关的浮点数定义

OpenGL函数

glBegin(xxx)：指定由顶点函数定义的几何模型的类型

glClear(parms): 清除由参数定义的窗口数据

glClearColor(r, g, b, a): 将图形窗口的颜色设为背景颜色

glColor3f(r, g,b): 为后续顶点调用设置的RGB值

glEnable(parms): 激活参数定义的性能

glEnd(): 几何模型定义区域的结束标记，和glBegin()配对使用

glLoadIdentity：将单位矩阵写入由glMatrixMode指定的矩阵中

glMatrixMode(parm)：指定后续操作使用到的系统矩阵

glPopMatrix(): 在glMatrixMode指定的当前矩阵栈中，将栈顶的矩阵弹出栈

glPushMatrix(): 复制当前矩阵栈中栈顶的矩阵，用于后续操作；当栈顶矩阵被glPopMatrix弹出后，该矩阵的值将恢复为最近调用的glPushMatrix栈顶矩阵值

glRotate(angle, x, y, z)：旋转几何模型，旋转轴的参数为(x,y,z)，旋转角度为angle

glScalef(dx,dy,dz): 将顶点坐标乘以指定值，对几何模型进行缩放

glTranslatef(tx,ty,tz): 顶点坐标加指定值，平移几何模型

glVertex3fv(array): 根据三维矩阵设置几何模型顶点

glViewport(x,y,width,height): 使用整数窗口坐标，指定绘制图形的视口尺寸

GLU函数：

gluLookAt(eyepoint, viewpoint, up):

gluPerspective(fileOfview, aspect, near, far): 基于观察环境参数，给定定义视域体的四个参数以定义透视投影

GLUT函数：

glutCreateWindow(title): 创建图形窗口，并给出窗口名

glutDisplayFunc(function): 为显示事件指定回调函数

glutIdleFunc(function): 为空闲事件指定回调函数

glutInit(parms): 根据main()函数的部分参数初始化GLUT系统

glutInitDisplayMode(parms): 根据传入的符号参数设置系统显示模式

glutInitWinodwPosition(x,y): 设置窗口左上角顶点屏幕坐标

glutMainLoop(): 进入GLUT事件处理循环

glutPostRedisplay(): 设置重绘事件，触发再次显示事件

glutReshapeFunc(function): 为改变窗口事件指定回调函数

glutSwapBuffers(): 后台颜色缓存中的内容交换到前台颜色缓存中用于显示

参数：

GL\_COLOR\_BUFFER\_BIT:与glclear一起使用，表明清空颜色缓存

GL\_DEPTH\_BUFFER\_BIT: 与glclear一起使用，表明清空深度缓存

GL\_DEPTH\_TEST: 指定使用深度测试

GL\_MODELVIEW：指定使用模型视角矩阵

GL\_QUAD\_STRIP：指定所用顶点是连续有序的四边形条带的顶点

GLUT\_DEPTH：指定窗口使用景深缓存（从而可以进行深度测试）

GLUT\_DOUBLE：指定窗口使用后台缓存（从而可以使用双缓存）

GLUT\_RGB: 指定窗口使用RGB颜色模型

# 第1章 视图变换和投影

## 1.1简介

决定所看图像的因素有：眼睛的位置、观察的目标点、观察视图的向上方向、观看的宽度、视图的高宽比。在计算机图形学中，必须指定这些参数才能正确定义图像。

模型变换、视图变换和投影等机制，通过图形API来管理，所以，图形编程的任务就是为这些API提供正确的信息，并且按照正确的顺序调用这些API函数。

## 1.2视图变化的基本模型

建立视图环境、定义投影、定义窗口和定义视图

## 1.3管理视图的其他方面

隐藏面：深度缓冲、z缓冲、画家算法

双缓存：前缓存和后缓存，这对动画是必需的。

立体视图：双目立体视图

视图变换和视觉交流

## 1.4在OpenGL中实现视图变换和投影

（1）定义窗口和视图：

glutInitWindowSize(); // 放在MyInit()函数中

glutInitWindowPosition();

glutCreateWindow();

（2）改变窗口的形状

glutReshapeFunc(); // 放在reshape()函数中

（3）设置视图变换的环境

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(); // 放在display()函数中

（4）定义透视投影

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(); // 或者glFrustum()

（5）定义正交投影

glOrtho();

（6）隐藏面的处理

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glEnable(GL\_DEPTH\_TEST);

如果想关闭深度测试，可以使用glDisable( )函数。

（7）设置双缓存

glutInitDisplayMode(GLUT\_DOUBLE

## 1.5本章的OpenGL术语

OpenGL函数：

glDepthFunc(parm): 通过符号参数指定计算函数，该函数决定顶点是否替换当前绘制缓冲中的顶点。

glDisable(parm): 通过符号参数表示禁用OpenGL的某些功能。

glFrustum(): 通过参数左和右、上和下、前和后裁剪平面来指定透视投影的视图平截头体。

glGetFloatv(parm, \*parms): 通过符号名称指定参数要返回什么值，并指定保存该值的变量。

glLoadMatrixf(array)

glOrtho();

glViewport()

GLUT函数：

glutGetWindow(): 返回当前活动的窗口值；

glutSetWindow(winName): 设置当前活动窗口值（通常为标识名）。

参数：

GL\_DEPTH\_TEST:

GL\_MODELVIEW\_MATRIX

GL\_PROJECTION

GL\_PROJECTION\_MATRIX

# 第2章 建模原理

建模是图形流水线的第一步。

顶点、线段、多边形、多面体、变换矩阵

变换和建模：

translate();

scale();

rotate();

drawBall();

图例和标签、精确度

场景图(scene graph)和建模图

scene graph定义：基于矢量图形编辑的应用，是一种常见的数据结构，可按照一定逻辑和空间表征的图形场景。是图或树状结构中的一系列节点。

# 第3章 在OpenGL中实现建模

## 3.1指定几何体的OpenGL模型

采用顶点列表的方式定义几何体：

glBegin(mode);

// 顶点列表：在mode所指定的绘制模式下创建基原对象的顶点数据

// 外观信息比如颜色、法线和纹理坐标也可以在这里指定

glEnd();

这种glBegin(mode)…glEnd()的顶点列表方式，使用不同的mode值来说明顶点列表创建图像的方式。

在OpenGL中，顶点数据是通过一系列以glVertex\*(…)命名的函数来定义的。这些函数将顶点坐标值送入OpenGL流水线，转换成图像信息。

如glVertex3f(x,y,z)表示3个浮点数。

在OpenGL中，glBegin(mode)和glEnd()之间可以调用自己的函数来指定顶点列表的顶点。还有其他很多信息可以放在两者之间，如顶点法向量、纹理坐标等。

3.1.1点和多点模型

glPointSize(2.0) // 设置每个点的像素数，默认是1.0

glBegin(GL\_POINTS);

for (i=0;i<(int)(3\*N);i++)

glVertex3f(2.0\*sin(step\*i), 2.0\*cos(step\*i), -1+zstep\*i);

glEnd();

3.1.2直线段

glBegin(GL\_LINES);

glVertex3f(0.,0.,0.);glVertex3f(5.,5.,.5.); // 第1条直线段

glVertex3f(1.,0.,1.);glVertex3f(6.,5.,.5.); // 第2条直线段

。。。。。。

glEnd();

3.1.3其他

线段序列：GL\_LINE\_STRIP

封闭线段：GL\_LINE\_LOOP

三角形：GL\_TRIANGLES

三角形序列：GL\_TRIANGLE\_STRIP、GL\_TRIANGLE\_FAN

四边形：GL\_QUADS:

for(i=0;i<XSIZE-1;i++)

for(j=0;j<jSIZE-1;j++)

// 四边形序列：点(i,j), (i+1,j), (i+1,j+1), (I,j+1)

glBegin(GL\_QUADS);

glVertex3f(XX(i),vertices[i][j],zz(j));

glVertex3f( )

glVertex3f( )

glVertex3f( )

glEnd();

}

四边形条带：GL\_QUAD\_STRIP

普通多边形：GL\_POLYGON

3.1.10顶点数据

单独的顶点处理（glVertex()）低效，可使用顶点数组，需要glEnable()来启用顶点数组，顶点数组可保存顶点、法线、纹理或其他信息，见第12章。

3.1.11反走样

成对使用下列API：

glEnable(GL\_LINE-SMOOTH)；

glHint(GL\_LINE\_SMOOTH\_HINT, GL\_NICEST);

## 3.2 OpenGL工具中的附加对象

3.2.5 GLUT对象

GLUT模型包括：

圆锥(glutSolidCone / glutWireCone) 实体模型和线框模型

立方体(glutSolidCube / glutWireCube)

。。。。。。

## 3.3 OpenGL中的变换

投影变换：由用户定义的投影生成的；

模-视变换：从用户定义的视图变换以及所有在程序中应用到的模型变换中得到的。

前面已经讨论了投影和视图，下面关注建模中用到的变换。

三种基本的模型变换：旋转、平移和缩放。

glRotatef(angle, x, y, z): angle是旋转角度，x,y,z指定了一个向量的坐标，所有参数都是浮点类型。

glRotatef(angle, 1.0, 0.0, 0.0)：沿着x轴旋转模型空间。

glTranslatef(Tx,Ty,Tz)

glScalef(Sx,Sy,Sz)

glGetFloatv(GL\_MODELVIEW\_MATRIX, trans)

## 3.4图标和标签

在图例和标签中的文本使用手工函数生成的，该函数集成了一些工具来显示文本。函数doRasterString(…)显示GLUT中的glutBitmapCharacter()函数定义的位图字符。

void doRasterString(float x, float y, float z, char \*s){

char c;

glRasterPos3f(x,y,z);

for(; (c=\*s)!=’\0’;s++)

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, c);

}

在display()函数中编写：

glBegin(GL\_QUADS);

glColor3f(0.,0.0,0.);

glVertex3f();

glVertex3f();

colorMap(0.3, &r, &g, &b);

……

glEnd();

sprintf(s, “5.0f”, 0.0);

doRasterString(0.1,0.1,0.,s);

glPopMatrix();

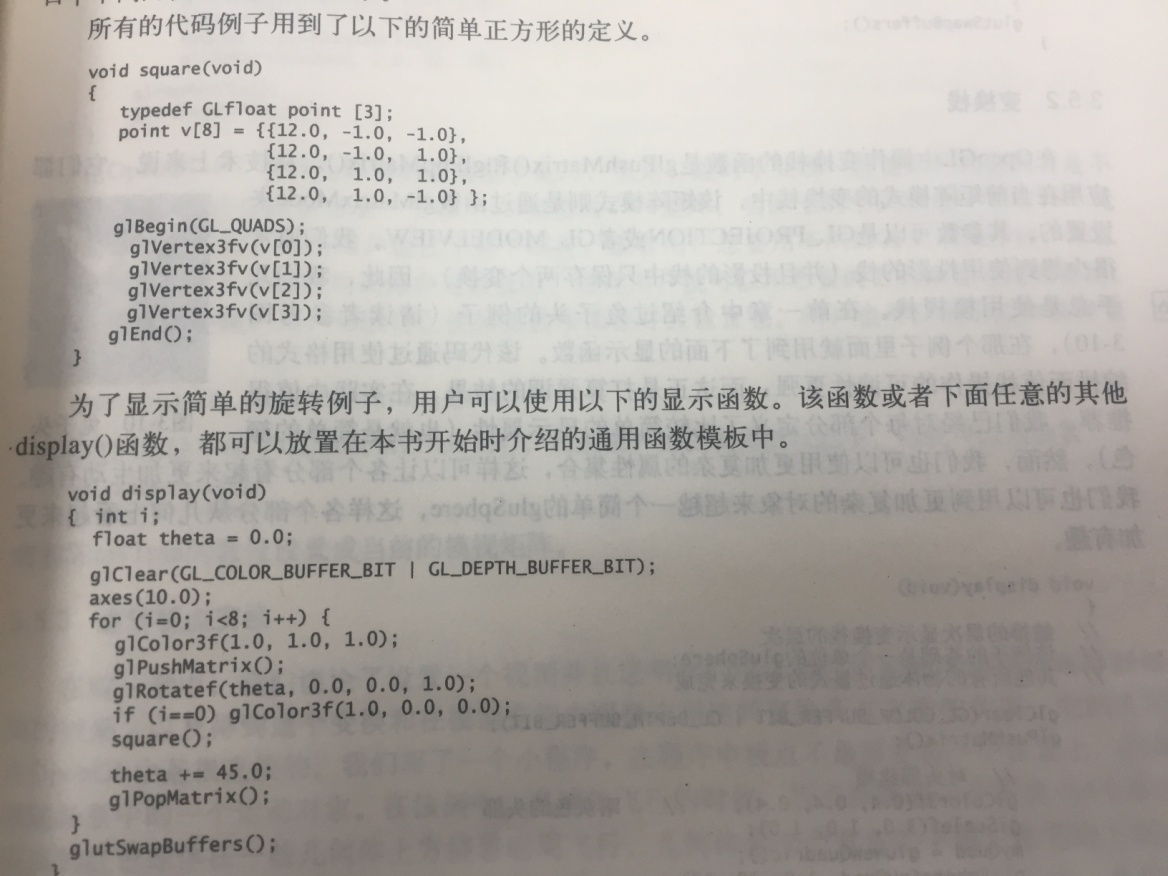
glDisable(GL\_SMOOTH);

// 现在回到主窗口来显示实际的模型

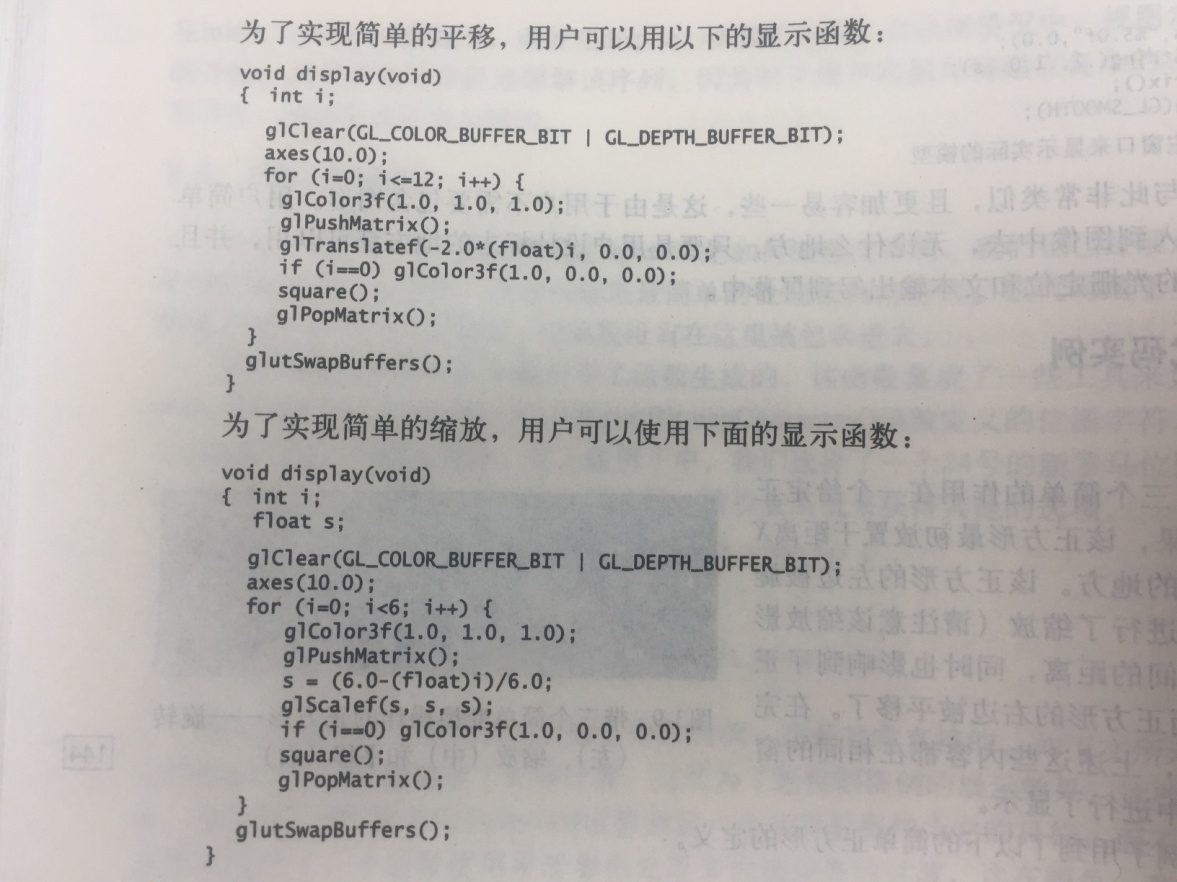
## 3.5变换的代码示例

3.5.1简单变换

在display()函数中，旋转：

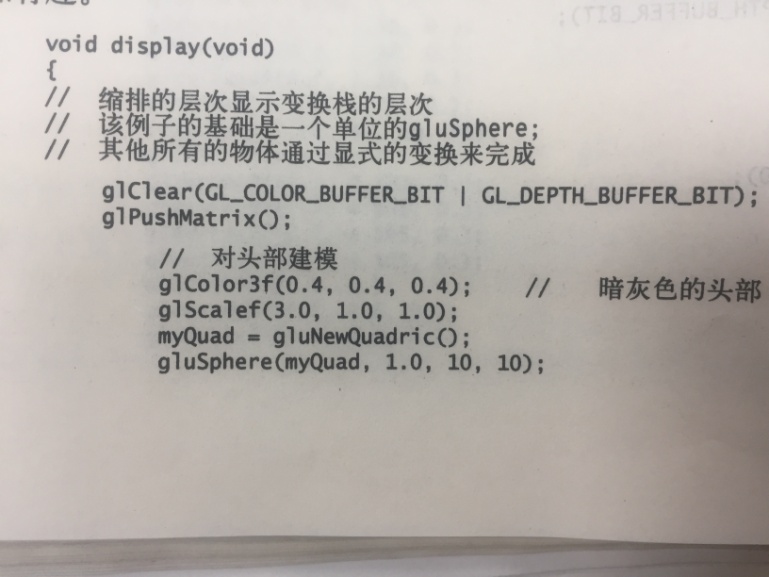


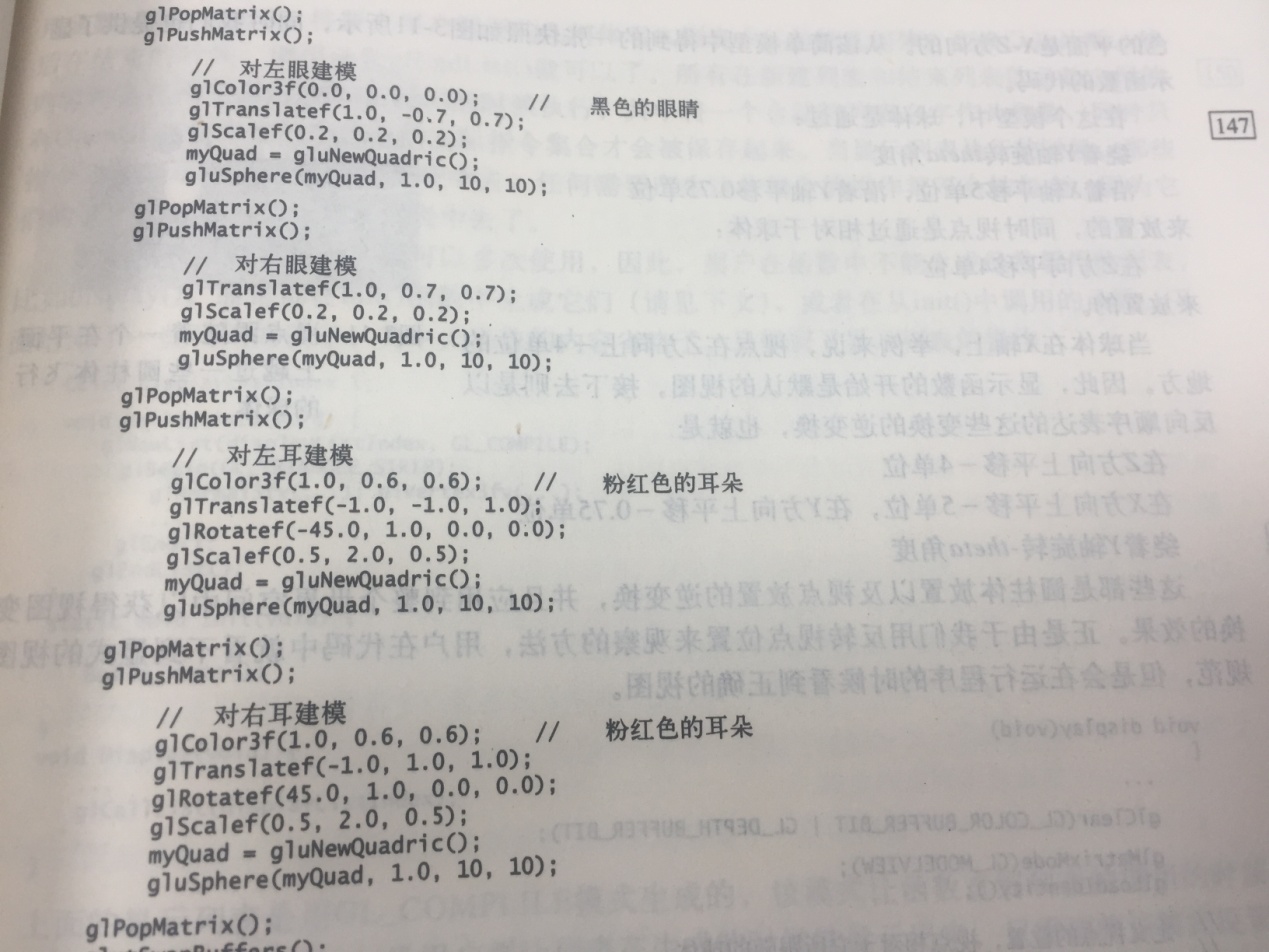
平移和缩放：



3.5.2变换栈

在OpenGL中操作变换栈的函数是glPushMatrix()和glPopMatrix()。





3.5.3逆转视点变换

视点不是固定在一个位置上，而是跟随场景中的一个运动对象。

3.5.4生成显示列表

在OpenGL中，图形对象可以编译进入显示列表中，显示列表包含了对象最后的几何体，等待显示。显示列表只要定义一次就可以多次使用，因此，不能在display()函数中，一般在init()中生成他们，或者在从init()中调用的函数。

GLint displayListIndex l;

void Build\_lists(void){

glNewList(displayListIndex, GL\_COMPILE);

glBegin(GL\_TRIANGLE\_STRIP);

glNormal3fv(…); glNormal3fv(…);

…

glEnd();

glEndList();

}

static void Init(void){

…

Build\_lists();

}

void Display(void) {

…

glCallList(displayListIndex);

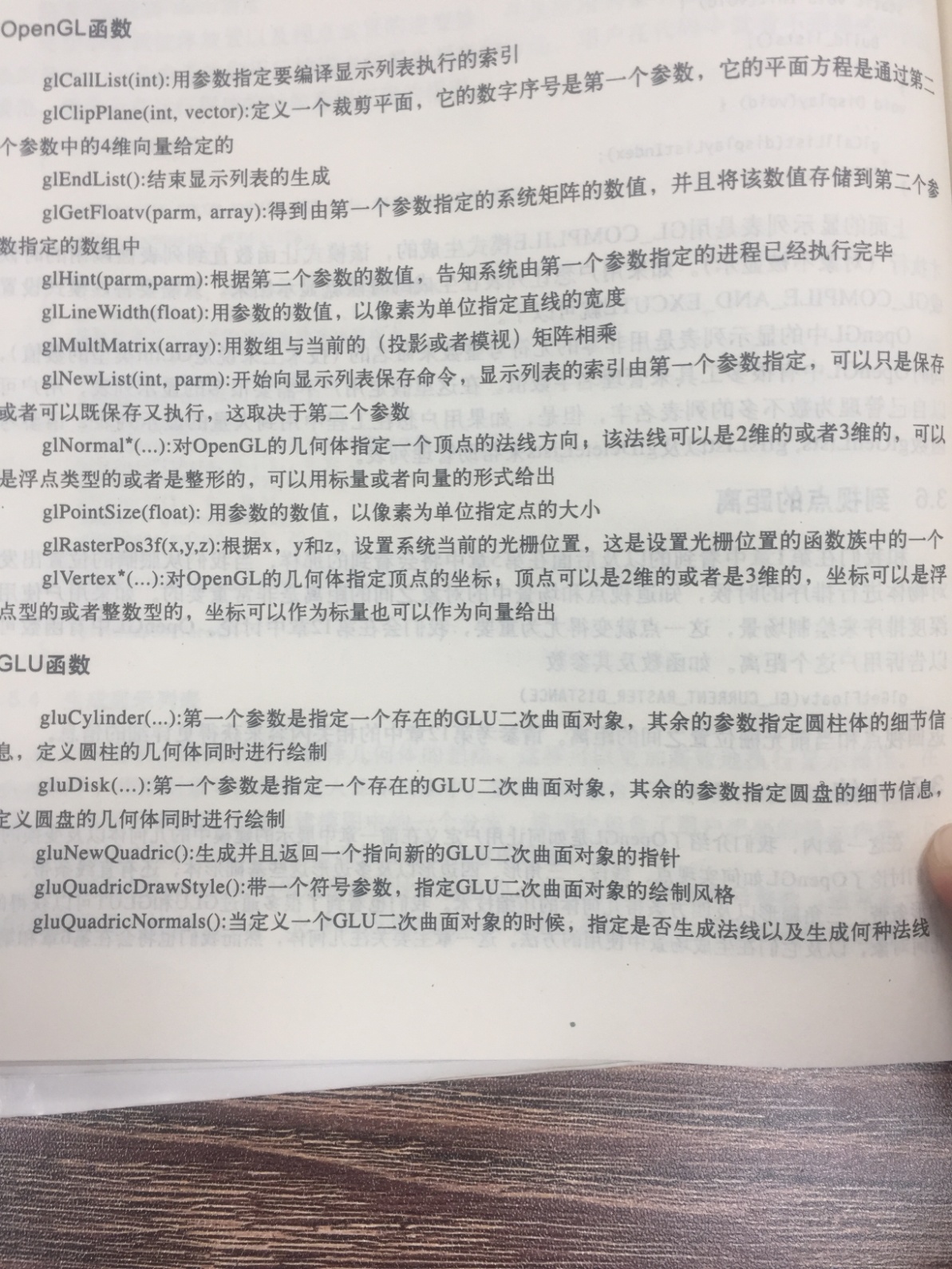
}

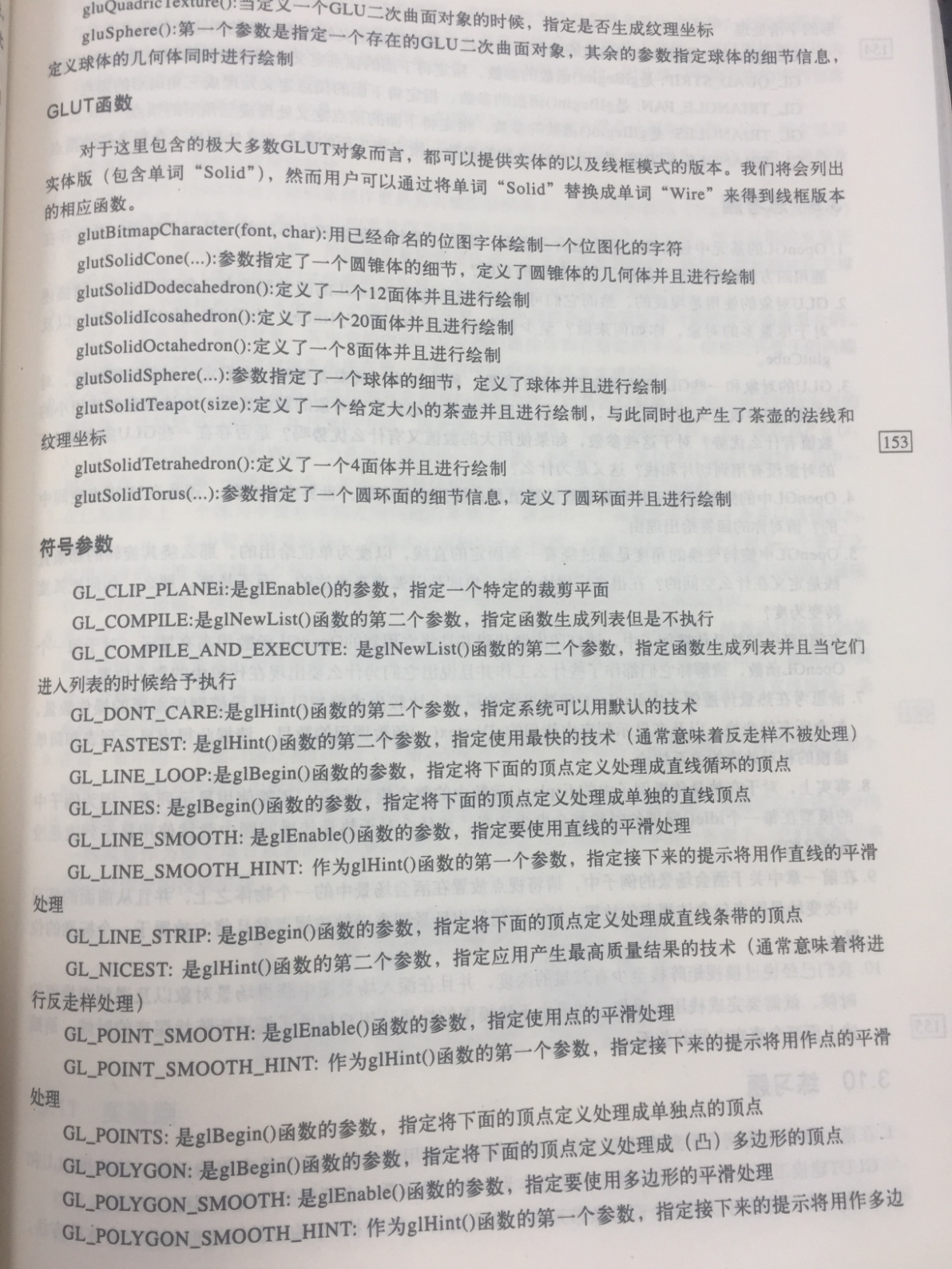
## 3.6到视点的距离

视点和场景中的对象之间的距离。

glGetFloatv(GL\_CURRENT\_RASTER\_DISTANCE)

## 本章的OpenGL术语表





# 第5章 颜色及其混合

RGB颜色模型、HSV (Hue-Saturation-Value)、HLS (Hue-Lightness-Saturation)

色度、亮度和饱和度\亮度和色弱、颜色深度、色谱、颜色混合和alpha通道

颜色和视觉交流：（1）强调色；（2）背景色；（3）自然色；（4）伪颜色和颜色渐变：把表示除图像颜色信息之外的其他值的颜色，称为伪颜色。

## 5.5 OpenGL中的颜色

5.5.1颜色定义

RGB分量值都为实数。OpenGL用glColor\*(…)函数说明颜色：

glColor3f(r, g, b): 三个实数标量颜色参数

glColor4fv(V): 或向量V的四个实数颜色参数

5.5.2使用混合

使用RGBA颜色模型时，用户必须说明是够要颜色混合，用户还必须指定颜色缓冲区中的颜色与新对象颜色的混合方式，可使用下面两个函数实现：

glEnable(GL\_BLEND);

glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);

混合函数的说明：glBlendFunc(src, dest); 源(src)和目标(dest)可以取许多值。

## 5.6代码实例

5.6.1带有全色谱的模型

typedef GLfloat color[4];

void cube(float r, float g, float b){

color cubeclr;

…

cubeclr[0]=r; cubeclr1]=g; cubeclr[2]=b; cubeclr[3]=1.0;

glColor4fv(cubeclr);

glBegin(GL\_QUADS);

…

glEnd();

}

void ribboncube(){

…

for(i=0;i<=NUMSTEPS;i++){

glPushMatrix();

glScalef(scale,scale,scale);

glTranslatef(…);

cube(…);

glPopMatrix();

}

}

## 本章的OpenGL术语表

OpenGL函数：

glColor\*(): 系统用于定义颜色的函数，系统用该函数定义的颜色画图。颜色可以有3个分量或4个分量，可声明为标量或数组。

glBlendFunc(const, const): 说明源和目标是否带变比因子来确定颜色混合。

常量：

GL\_BLEND：说明是否带颜色混合，由函数glEnable()和glDisable()使用。

GL\_ONE\_MINUS\_SRC\_ALPHA：函数glBlendFunc()使用的变比因子，说明颜色是否乘以1-alphasource。

GL\_SRC\_ALPHA: 函数glBlendFunc()使用的变比因子，说明颜色是否乘以alphasource。

# 第6章 光照处理和着色处理

创建更具有吸引力和更加真实的图像技术。它主要是关于产生增强图像效果的两个方面：一个是基于简单模型的光照处理，主要是光线和表面的交互；另一个是基于简单模型的着色处理，主要是物体表面的颜色变化。

光照处理模型主要基于光线的三个组成：间接光照、直接光照以及反射光照。

## 6.1光照处理

只考虑场景中光线的光照处理：局部光照模型。指Phong光照模型。

光照分成3个基本组成：环境光、漫反射光和镜面反射光

## 6.2材质

光照处理包括在场景中指定光源和物体与光线相关的属性。为了在场景中使用光照，需要指定以下两方面：光源的属性和物体的材质属性。

光照处理包括把光源和材质整合在一起。

## 6.3光源属性

光源颜色、位置光、聚光灯、光线衰减、方向光

## 6.4放置与移动光源

光源在场景中的位置是确定的；

光源在场景中的位置相对视点是确定的；

光源在场景中的位置相对物体是确定的；

光源在场景中的位置是到处移动的。

## 6.5用光照实现特效

## 6.6场景图中的光源

## 6.7着色处理

初级的OpenGL API，只支持简单的局部光照模型：Flat着色和平滑着色（Gouraud着色处理）。还有更成熟的着色处理。

## 6.8在视觉交流中考虑着色器

## 6.9定义

Flat着色处理：整个多边形的颜色是平坦的（不变化），或者说多边形着色的时候是平坦的，所以照亮时它的颜色不变。

不同图形API实现Flat着色都是一样的，但实现平滑着色是不一样的。最简单的平滑着色处理可以通过先计算每个顶点的颜色，然后对整个平面进行光滑的颜色插值来实现，这种方式称为**Gouraud着色处理**。

## 6.11计算每个顶点的法向

Flat着色处理的图像对**一个多边形**只使用一个法向，而用平滑着色处理的图像对**多边形每个顶点**使用独立的法向向量。对每个顶点计算法向比对每个多边形计算法向的工作量大，这就是平滑着色处理的代价。

## 6.12其他着色处理模型

Phong着色处理模型，为每个顶点设立一个法向，并且对法向进行插值，而不是颜色插值，来计算多边形中每个像素的颜色。插值法向比插值颜色更复杂。而且，Phong着色处理模型假设整个多边形都是真正的平滑表面，可以在多边形内部产生镜面反射光，而且沿着多边形的边也非常光滑。

Phong着色处理模型是基于对整个多边形法向进行连续的变化，还有一种着色处理模型是基于对整个多边形的法向进行控制，这就是**纹理映射**。

## 6.13各向异性着色处理器

上面介绍的都是简单的光照处理模型，假设光线是均匀地从表面法向反射（等方性光照）。然而，有一些材质的光照参数根据光线和眼睛围绕法向的夹角而变化。**各向异性着色处理**：不同方向的反射是不一样的。

光照计算中的那些角度，包括漫反射中的法向角度和镜像反射光反射中反射光线的角度，被一个更加复杂，称为**双向反射分布函数（BRDF）**代替，表示为，它依赖于眼睛的维度角和经度角，以及光照射的点位置：。BRDF也考虑对不同波长（或不同颜色）的光线采取不同的行为。BRDF着色处理通过修改表面法向来模拟近似的BRDF，当达到把每个像素当做一个多边形来处理的时候，就是真正的各向异性着色处理。

## 顶点和像素着色器

简介

OpenGL着色处理语言GLSL是一种高级的面向过程的语言，和C语言类似，但添加了一些反映可编程图形特性的特有的数据和操作，它代替了图形卡的某些函数。原理很简单：把顶点数据发送给图形卡，把它们转化为像素位置，通过多边形扫描线（称为片段, fragment）来设置每个像素的颜色。

主要目标：速度；次要目标：图像质量

GLSL着色器类型包括：顶点着色器（做来处理坐标、法向、纹理坐标或者颜色）；细分控制和计算着色器；几何着色器；片段着色器（用来提供更真实的犹如各种非照相真实感的效果）。

着色器变量，包括：attribute变量、uniform变量、constant变量、out和in变量

编译着色器由驱动程序完成。

整型标量以及向量类型：int, ivec2, ivec3, ivec4

实数标量以及向量类型：float, vec2, vec3, vec4

实数方阵矩阵类型：mat2, mat3, mat4

非方阵实数矩阵类型：mat3x2

布尔标量以及向量类型：bool, bvec2, bvec3, bvec4

向量分量可通过[index]访问，也可采取字母名称（即名称集合）进行访问，包括：.rgba(颜色向量)，.xyzw(几何向量)，.stpq(纹理坐标向量)。

代码实例

GLuint createShaderProgram() { // 顶点着色语言

const char \*vshaderSource =

“#version 430 \n”

“void main(void) \n”

“{ gl\_Position = vec4(0.0, …); }”;

const char \*fshaderSource = // 片段着色语言

“#version 430 \n”

“out vect4 color; \n”

“void main(void) \n”

“{ color = vec4(0.0, 0.0, …); }”;

// 创建了GL\_VERTEX\_SHADER和GL\_FRAGMENT\_SHADER两个着色器

GLuint vShader = glCreateShader(GL\_VERTEX\_SHADER); //返回整数ID，vShader

GLuint fShader = glCreateShader(GL\_FRAGMENT\_SHADER);

glShaderSource(vshader, 1, &vshaderSource, NULL); //将GLSL代码从字符串载入空着色器对象中

glShaderSource(fshader, 1, &fshaderSource, NULL);

glCompileShader(vshader); // 编译着色器

glCompileShader(fshader); // 请求GLSL编译器确保它们的兼容性

GLuint vfProgram = glCreateProgram(); //创建一个叫vfProgrm的程序对象，并存储指向它的整数ID

glAttachShader(vfProgram, vshader); // 将着色器加入程序对象

glAttachShader(vfProgram, fshader);

glLinkProgram(vfProgram);

return vfProgram;

}

void init(GLFWwindow \* window) {

renderingProgram = createShaderProgram();

glGenVertexArrays(numVAOs, vao);

glBindVertexArray(vbo[0]);

}

void display(GLFWwindow, \* window, double currentTime) {

glUseProgram(renderingProgram); //将含有2个已编译着色器的程序载入OpenGL管线阶段（在GPU上！），并没有运行着色器，只是将着色器加载进硬件。

glDrawArrays(GL\_POINTS, 0, 1); //启动管线处理，画一个点（像素为1）

}

glShaderSource有4个参数：第1个参数用来存放着色器的着色对象；第2个参数是着色器代码中的字符串数量；第3个是包含源代码的字符串指针；最后一个是没用到的参数。

**从文件中读取GLSL源代码**

当GLSL着色器代码很长时，需要从文件读入。将顶点着色器和片段着色器代码保存为vertShader.glsl和fragShader.glsl

#include <string>

#include <iostream>

#include <fstream>

string readShaderSource(const char \*filePath) {

string content;

… // C++语言读取字符串(string)的程序

return content;

}

GLuint createShaderProgram() {

// 与之前的代码相同

string vertShaderStr = readShaderSource(“vertShader.glsl”);

string fragShaderStr = readShaderSource(“fragShader.glsl”);

const char \*vertShaderSrc = vertShaderStr.c\_str();

const char \*fragShaderSrc = fragShaderStr.c\_str();

glShaderSource(vShader, 1, &vshaderSrc, NULL);

glShaderSource(fShader, 1, &fshaderSrc, NULL);

…

}

## 6.14全局光照

在周围的世界中，光线并不只是直射光源，也并不只有单个环境光值。光线被场景中的每个物体和表面反射，这些间接光源在整个场景中各不相同。解决这种问题的光照处理称为**全局光照模型**。

因为光照是针对整个场景计算的，并不和视点相关，也不是根据视点对每个多边形进行计算。全局光照过程并不包括着色处理模型，任何光线能量达到表面产生的着色处理结果都传递给另外的表面。

6.14.1辐射度方法

6.14.2光子映射

## 6.15局部光照和OpenGL

6.15.1指定和定义光源

定义场景进行单面还是双面光照：

glLightModel[f|i](GL\_LIGHT\_MODEL\_TWO\_SIDE, value)

是否进行镜面反射光计算：

glLightModel[f|i](GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, value)

全局环境光：

glLightModelf(GL\_LIGHT\_MODEL\_AMBIENT, r, g, b, a)

光源的位置和颜色：

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_pos0); // 光源 0

glEnable(GL\_LIGHT); // 使用光照模型

glDisable(…);

6.15.2选择性地使用光源

// 太阳

glEnable(GL\_LIGHT1);

glDisable(GL\_LIGHT0);

…

// 地球

glDisable(GL\_LIGHT1);

glEnable(GL\_LIGHT0);

…

6.15.3定义材质

glMaterial\*(…)

glMaterial[i|f][v](face, pname, value)

例如：

GLfloat shininess[]={ 50.0 };

GLfloat white[] = {1.0, 1.0, 1.0, 1.0};

glMaterialfv(GL\_FRONT, GL\_AMBIENT, white);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, white);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, white);

glMaterialfv(GL\_FRONT, GL\_SHININESS, shininess);

## 6.16建议

OpenGL光照模型缺乏一些非常重要的功能，这些功能可以让场景达到真正想要的真实效果，包括：创建阴影的技术；“热”色彩，OpenGL难以实现上述的真实的图像效果，需要特殊处理。

OpenGL不允许各向异性反射，需要特殊的计算机程序。

## 6.17小结

本章介绍了如何利用Phong光照模型处理顶点颜色，利用Gouraud着色处理模型进行线性平均颜色，建立在整个多边形上平滑变化的颜色，并利用它们来创建图像。至此，外观部分的简单扩展只剩下纹理映射了。

## 本章的OpenGL术语

OpenGL函数：

glLight\*(light, pname, value): 为指定的光源设置命名参数的值的函数集

glLightModel\*(pname, value): 为将要使用的光照模型设置参数值的函数集

glMaterial\*(face, pname, value): 为光照模型指定材质属性的函数集

glShadeMode(pname)： 选择Flat着色处理或者选择平滑着色处理

# 第7章 事件和交互式编程

交互式计算机图形

用户界面：应用程序中与用户交互的部分

## 7.1定义

事件：计算机系统控制状态的转换；

事件记录：

事件队列：

事件处理程序：

事件注册回调函数：回调函数和事件关联起来的过程

主事件循环

## 7.2事件的例子

按键事件：keyDown, keyUp …

菜单事件：

鼠标事件：

软件事件：程序本身发送的，目的是让程序接下来执行一个特殊的操作，如redisplay重显示事件。

系统事件：

窗口事件：比如移动窗口或改变窗口大小。

## 7.3交互的方式和方法

交互设备、离散的输入设备：鼠标和键盘

操纵杆、spaceball、gamepad (Joystick/Controller)

## 7.4对象选择

通过菜单、键盘敲击和鼠标功能等交互方式，图形API能将用户输入集成到应用程序中。这些输入操作先为选择的对象指定应完成的动作，然后操纵图像。入股偶需要在场景里识别某个对象，并赋予某一动作，就不能简单地靠鼠标点击来识别它，必须通过程序解析这个点击来完成对象选择操作。

## 7.5交互和视觉交流

如何设计和实现交互式图形应用程序，帮助用户理解显示图像的含义。这种交互就是用户和程序之间的一种交流，通过这种交流用户可以对图像施加影响，因此需要很好的视觉交流。这种交互也让用户可以通过操纵图像来理解图像要表达的信息。

一类常见的交互式应用是从不同的视点来观察整个场景或观察某个对象。它能让用户在场景中漫游，以及放大或者缩小场景。在场景中漫游的另一种实现方式是在世界坐标空间旋转场景本身，而保持视点方向不变。运用鼠标可以很自然地控制旋转操作。

另一类应用程序需要针对图像的某一部分进行部分图像操作，例如选择，并做相关的操纵。鼠标点击选中。

## 7.6事件和场景图

将事件和交互一并纳入场景图中，来管理所有的交互操作。

场景图的4种节点：组节点、变换节点、几何节点和外观节点。

## 7.7 建议

专业的应用程序需要一个专业的界面、一个由这个领域的专业人士设计、测试以及改进的专业界面。

## 7.8 OpenGL中的事件

OpenGL API中一般通过GLUT来实现对事件和窗口的处理。GLUT定义了许多事件，并提供给程序员一系列的回调函数与之相对应。在有GLUT扩展的OpenGL中，主事件循环显式地调用函数glutMainLoop( )，把它作为主程序的最后一个语句。

## 7.9回调函数的注册

下面列举一些OpenGL的事件，对于每一个事件给出相应回调函数的注册函数。

|  |  |
| --- | --- |
| 事件 | 回调函数的注册函数 |
| idle | glutIdleFunc(func\_name)，需要一个回调函数作为参数，该回调函数具有形式void func\_name(void)。这个函数是一个事件处理程序，决定每一个空闲周期做什么。通常，这个函数的结束是调用glutPostRedisplay( )，这个函数定义程序在没有其他事件处理时做什么操作，它通常用来驱动实时动画。 |
| display | glutDisplayFunc(func\_name)需要一个回调函数作为参数，该回调函数具有形式void func\_name(void)。这个函数也是一个事件处理程序，无论显示事件何时收到，它会产生一个新的显示命令。注意：这个显示函数将被事件处理程序调用，无论这个显示事件何时到达。这个事件是由glutPostRedisplay( )函数发送的，当窗口被打开、移动或者调整时发送。 |
| reshape | glutReshapeFunc(func\_name)需要一个回调函数作为参数，该回调函数具有形式void func\_name(void)。这个函数管理任何视域设置的改变，以适应新调整的窗口和新的投影定义。reshape回调函数的参数是窗口改变以后的宽度和高度。 |
| keyboard | glutKeyboardFunc(func\_name)需要一个回调函数作为参数，该回调函数具有形式void func\_name(unsigned char, int, int)。这个带参数的函数是处理键盘被按下的事件处理程序，接收哪些字符被按下以及当前的光标位置(int x, int y)。与所有包括屏幕位置的回调函数一样，该函数的屏幕位置会转化为相对窗口的坐标。同样，这个函数的结束是调用glutPostDisplay( )来重新显示某个特定键盘事件引起改变的场景。 |
| special | glutSpecialFunc(func\_name)，回调函数具有形式void func\_name(int key, int x, int y)。这个事件产生于那些“特殊键”被按下时，这些键包括功能键、方向键和一些其他按键。第一个参数是按下的键值，第2和第3个参数是窗口整型坐标，它记录了当特殊键被按下时光标的位置。 |
| menu | int glutCreateMenu(func\_name)，回调函数形式void func\_name(int)，传递给这个函数的是一个整型参数，它记录了当菜单打开并有选项被选中后，所选中的菜单项对应的编号。 |
| mouse | glutMouseFunc(func\_name)需要一个回调函数作为参数，该回调函数具有形式void func\_name(int button, int state, int mouseX, int mouseY)。这里，button表示哪一个键被按下，state表示鼠标的状态，比如GLUT\_DOWN；无论按下和松开按键，都会产生事件，还有整型值xPos和yPos来记录当事件发生时光标在窗口中的坐标。如果鼠标键定义为用来触发一个菜单，则鼠标事件将不会使用该回调函数。 |
| mouse passive motion | glutPassiveMotionFunc(func\_name)，回调函数形式void func\_name(int, int)。这两个整型参数代表了事件发生时光标相对窗口的坐标。这个事件是鼠标在鼠标键放开（即没有被按下）状态下移动时产生。 |
| joystick | glutJoystickFunc |
| timer | glutTimerFunc(msec, func\_name, value)需要一个整型参数(msec)来表示经过多少毫秒回调函数被触发；一个回调函数参数，回调函数具有形式void func\_name(int)；还需要一个整型参数(value)作为调用该回调函数时的传入参数。 |

还有其他的一些设备事件，如Joystick

glutJoystickFunc — sets the joystick callback for the current window.

void glutJoystickFunc(void(\*)(unsigned int buttons, int xaxis, int yaxis, int zaxis), int pollInterval)

参数：func:The new joystick callback function. “pollInterval”: Joystick polling interval in ms.

介绍：glutJoystickFunc sets the joystick callback for the current window.

The joystick callback is called either due to polling of the joystick at the uniform timer interval specified by pollInterval (in milliseconds) or in response to calling glutForceJoystickFunc. If the pollInterval is non-positive, no joystick polling is performed and the GLUT application must frequently (usually from an idle callback) call glutForceJoystickFunc.

The joystick buttons are reported by the callback's buttonMask parameter. The constants GLUT\_JOYSTICK\_BUTTON\_A (0x1), GLUT\_JOYSTICK\_BUTTON\_B (0x2), GLUT\_JOYSTICK\_BUTTON\_C (0x4), and GLUT\_JOYSTICK\_BUTTON\_D (0x8) are provided for programming convience.

The x, y, and z callback parameters report the X, Y, and Z axes of the joystick. The joystick is centered at (0,0,0). X, Y, and Z are scaled to range between -1000 and 1000. Moving the joystick left reports negative X; right reports positive X. Pulling the stick towards you reports negative Y; push the stick away from you reports positive Y. If the joystick has a third axis (rudder or up/down), down reports negative Z; up reports positive Z.

Passing a NULL func to glutJoystickFunc disables the generation of joystick callbacks. Without a joystick callback registered, [glutForceJoystickFunc](https://www.dei.isep.ipp.pt/~matos/cg/docs/manual/glutForceJoystickFunc.3GLUT.html) does nothing.

When a new window is created, no joystick callback is initially registered.

限制：The GLUT joystick callback only reports the first 3 axes and 32 buttons. GLUT supports only a single joystick.

Glut Implementation Notes For X11：The current implementation of GLUT for X11 supports the joystick API, but not joystick input. A future implementation of GLUT for X11 may add joystick support.

## 7.10实现细节

F1到F12的功能键：GLUT\_KEY\_F1到GLUT\_KEY\_F12

方向键：GLUT\_KEY\_LEFT

其他特殊键：GLUT\_KEY\_PAGE\_UP，。。。

在使用这些特殊键时，要用这些有含义的符号名来处理这些键值，这些键值返回给特殊键回调函数。

timer事件与其他任何事件都不一样，是一个“注册一次作用一次”的事件。

创建和操纵菜单

菜单是交互式编程的关键部分，GLUT和计算机的窗口系统结合在一起，为程序提供一个设备无关的菜单实现方法。

创建一个菜单调用glutCreateMenu(…)，通过glutAddMenuEntry(name, value)函数和每一个菜单项的名字联系起来。可以通过glutSetMenu(int)函数来激活菜单，也可以通过glutAddMenuEntry(…)函数来添加菜单项。

在OpenGL中，菜单可以被激活，也可以取消激活，可以被创建，也可以被销毁；菜单可以被添加、删除和修改。都可以在GLUT中找到相应的API。

查看当前活动的菜单编号，用：int glutGetMenu(void);

……

## 7.11代码实例

7.11.1空闲事件回调函数

假设函数cube()可以画一个简单的立方体，边长是2.0，中心在原点处。随着时间推移，通过改变坐标来移动这个立方体，这样，让idle事件回调函数来设置立方体新的坐标，然后发送一个redisplay事件。显示函数将根据idle回调函数中设置的坐标画出新的立方体。

#define deltaTime 0.05

GLfloat cubex=0.0, cubey=0.0, cubez=0.0, time=0.0;

void displsy(void) {

glPushMatrix(); // 压入栈顶矩阵

glTranslatef(cubex, cubey, cubez); // 平移立方体的坐标

cube(); // 画出一个立方体

glPopMatrix(); //

}

void animate(void) {

// 立方体的位置由基于事件的行为建模设置

// 用不同的常量与时间相乘，并观察行为的变化

time+=deltaTime; if(time > 2.0\*M\_PI) time-=2.0\*M\_PI;

cubex=sin(time);

…

glutPostRedisplay();

}

void main(int argc, char \*\* argv){

// 在以下函数之前是标准的GLUT初始化

…

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutIdleFunc(animate);

myInit();

glutMainLoop();

}

7.11.2定时器时间回调函数

定时器回调函数可以按照设置的时间表来驱动程序的动作。

本例中，用定时器回调函数来代替空闲回调函数管理立方体移动的动画过程，并允许控制动画过程的节奏。设定合适的延迟时间，可以使动画在快速的系统上不至于太快。

#define frameDelay 33

#define dTime 0.05

void timer (int i){

aTime+=dTime; if(time > 2.0\*PI) time-=2.0\*PI;

cubex=sin(2.0\*aTime);

…

glutTimerFunc(frameDelay, timer, 1);

glutPostRedisplay();

}

void main(int argc, char \*\* argv){

…

glutTimerFunc(frameDelay, timer, 1);

…

}

7.11.3键盘回调函数

从cube()函数开始，让用户通过简单的键盘按键来控制立方体的上下。

GLfloat cubex=0.0;

…

GLfloat time =0.0;

void display(void) {

glPushMatrix(); // 压入栈顶矩阵

glTranslatef(cubex, cubey, cubez); // 平移立方体的坐标

cube(); // 画出一个立方体

glPopMatrix(); //

}

void keyboard(unsigned char key, int x, int y){

ch = ‘’;

switch(key)

{

case ‘q’: case ‘Q’:

case ‘i’: case ‘I’:

ch= key; cubey -=0.1; break;

case

…

}

glutPostRedisplay();

}

void main(int argc, char \*\* argv) {

// 标准的GLUT初始化

glutDisplayFunc(display);

glutKeyboardFunc(keyboard);

myInit();

glutMainLop();

}

类似的函数glutSpecialFunc(…)可以用来读取键盘上特殊键的输入。

7.11.4菜单回调函数

同样从cube()函数开始，但这一次，不是让立方体做运动，而是定义一个菜单选择立方体的颜色。当选择了颜色后，新颜色将应用到立方体上。这个例子只用一个静态菜单，因此，glutCreateMenu(…)函数返回的值将被main( )函数忽略。

#define RED 1

#define GREEN 2

…

void cube(void) {

…

GLfloat color[4];

// 根据菜单选择设置颜色

switch(colorName) {

case RED:

color[0]=1.0; color[1]=0.0;

color[2]=0.0; color[3]=1.0; break;

case GREEN:

…

}

void display(void){

cube();

}

void options\_menu(int input) {

colorName = input;

glutPostRedisplay();

}

void main(int argc, char \*\* argv){

…

glutCreateMenu(options\_menu); // 创建选项菜单

glutAddMenuEntry(“Red”, RED); //1增加菜单项

glutAddMenuEntry(“Green”, GREEN); //2

…

glutAttachMenu(GLUT\_RIGHT\_BUTTON,”Colors”);

myInit();

glutMainLoop();

}

7.11.5鼠标移动的鼠标回调函数

使用整型坐标spinX和spinY来控制旋转。

float sinX, spinY;

int curX, curY, myX, myY;

void mouse(int button, int state, int mouseX, int mouseY) {

curX = mouseX;

curY = mouseY;

}

void motion(int xPos, int yPos) {

spinX = (GLfloat)(curX – xPos);

spinY = (GLfloat)(curY – yPos);

myX = curX;

myY = curY;

glutPostRedisplay();

}

int main(int argc, char \*\* argv) {

…

glutMouseFunc(mouse);

glutMotionFunc(motion);

myInit();

glutMainLoop();

}

7.11.6对象拾取的鼠标回调函数

## 7.12拾取的实现细节

标准的选择方法：Render模式和Selection模式

glRenderMode(GL\_SELECT): 选择场景绘制的模式

glSelectBuffer(value, buffer): 创建一个给定大小的缓存数组，保存GL\_SELECT模式下得到的信息。

# 第8章 纹理映射

纹理映射是另一种在多边形中产生颜色的方式。它可以将一副图像映射到多边形上以产生精美的画面，或者将相应信息加入到图像中而不必计算额外的几何值。

纹理映射的基本思想是：当图形几何值已经计算和显示完成后，在图像中加入附加的可视内容。

纹理映射就是将场景中对象的点与纹理图中的点对应起来，以便于使用简单的几何图产生丰富逼真的视觉效果图像。

纹理映射涉及两个空间：2D屏幕空间（即显示物体的空间）和纹理空间（该空间放置要映射到物体上去的纹理图的信息）。

最常见的是2D纹理图，可用Photoshop等软件生成原始的RGB文件。

## OpenGL中的纹理映射

glEnable(): 允许纹理映射，glDisable()表示不允许纹理映射操作；

glGenTextures(…) 生成用于纹理的一个或多个名称（参数）

glBindTexture(…)把纹理名（glGenTextures生成）与纹理对象绑定，如GL\_TEXTURE\_2D

glTexEnv\*(…)定义纹理操作环境

glTexParameter\*(…)定义纹理反走样、反卷等操作参数

glTexImage\*(…)与定义纹理参数的信息绑定：如颜色坐标数等

glTexCoord\*(…)纹理坐标与几何体顶点相关

glTexGen\*(…)控制几何体顶点纹理坐标的自动生成

glDeleteTextures(…)删除由glGenTextures生成的一个或多个纹理

# 第9章 图形在科学计算领域中的应用

## 9.1简介

科学可视化

图像🡪观察

认知🡪问题

图形化的问题解决

建立问题模型

数据和视觉交流

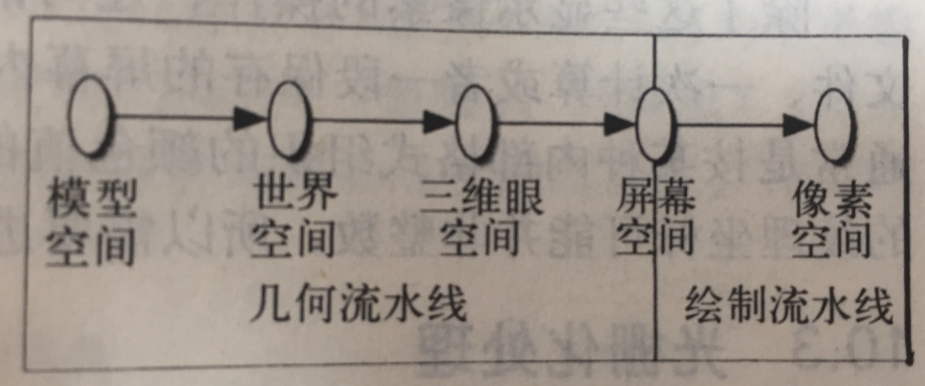
科学数据的高维度

经常把二维屏幕认为是三维空间的投影，所以也需要把高维空间投影到三维空间上。随着智能三维沉浸式观察设备的提高，将来也许没有必要投影到三维空间上。

# 第10章 绘制与绘制流水线

## 流水线

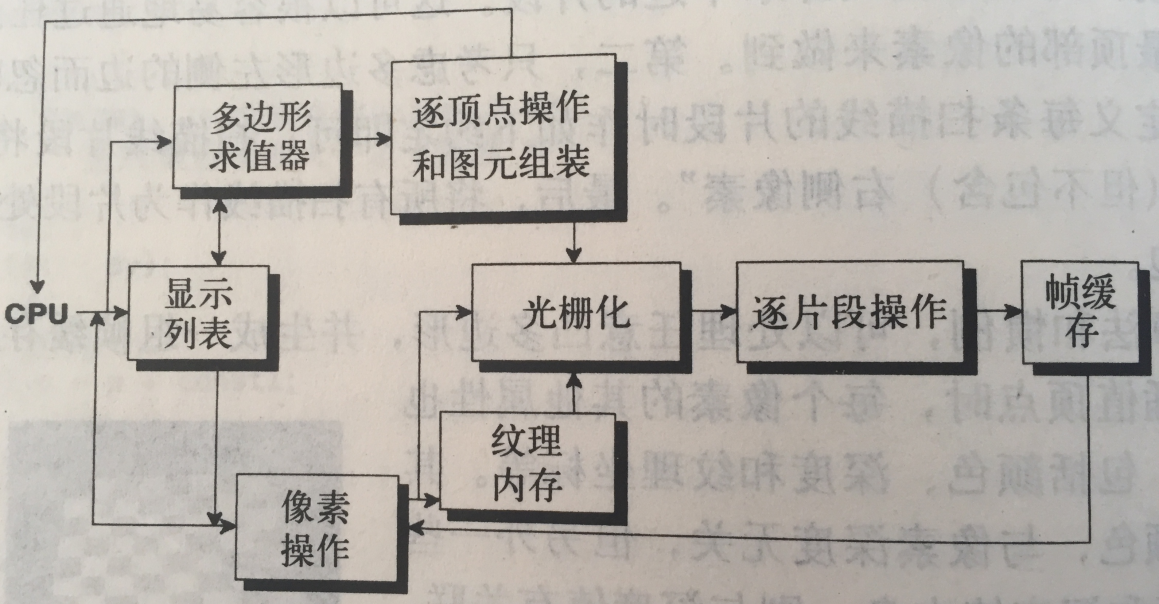
分为几个阶段：（1）对场景中的每个多边形，根据其顶点得到多边形在光栅显示设备中相应扫描线的端点，多边形内部的像素点信息可以根据这些端点插值得到；（2）插值过程进一步应用于深度测试、剪裁和颜色混合等过程。



## 光栅化处理

## OpenGL的绘制流水线

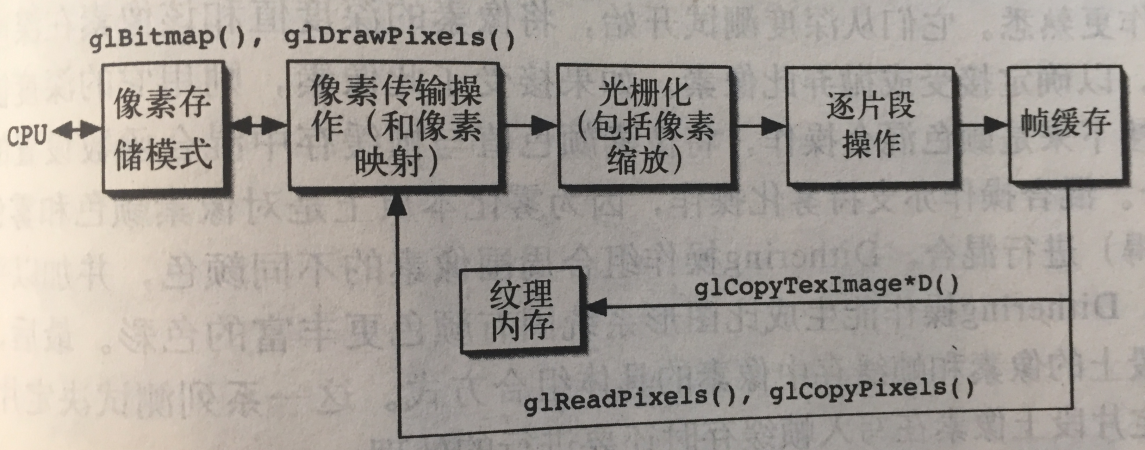
立即模式操作、显示列表方式



绘制流水线中的纹理映射

纹理映射涉及纹理内存和绘制系统的其他部分。一个纹理图可以从文件中读入，或者对帧缓存或其他来源的数据应用像素操作而得到。

下图中，从帧缓存回到像素操作的箭头表明：帧缓存中的信息可以被取出并写入帧缓存的其他部分；从帧缓存到纹理内存的箭头提示了：帧缓存中的信息甚至可以作为纹理图本身。



纹理图处理

纹理内存可以复制帧缓存中的数据，可使用glCopyTexImage\*D(…)函数或者其他像素级操作。

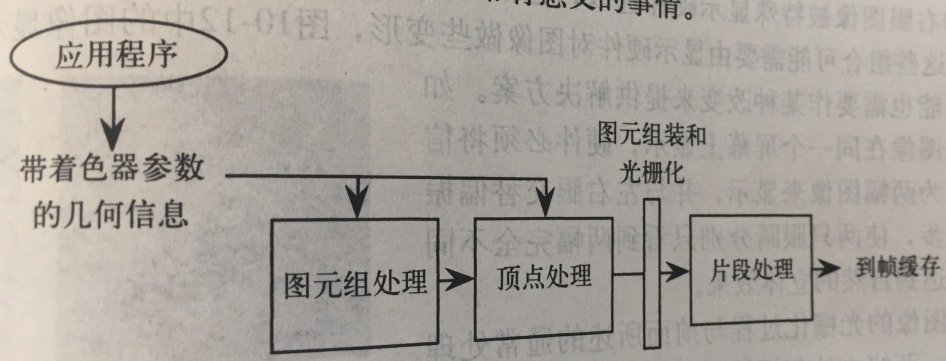
逐片段操作

OpenGL与可编程着色器

在标准的OpenGL中，当顶点进入绘制流水线时，顶点可附加包括坐标、颜色、纹理坐标等信息，甚至可以存储程序的地址，可以用来计算各向异性的着色处理以及凹凸纹理图等。图形卡开始拥有强大的逐顶点编程能力，如每个顶点带有16个或更多的4维实数向量来装载附加数据，虽然每个图形卡具有完全不同的指令集。

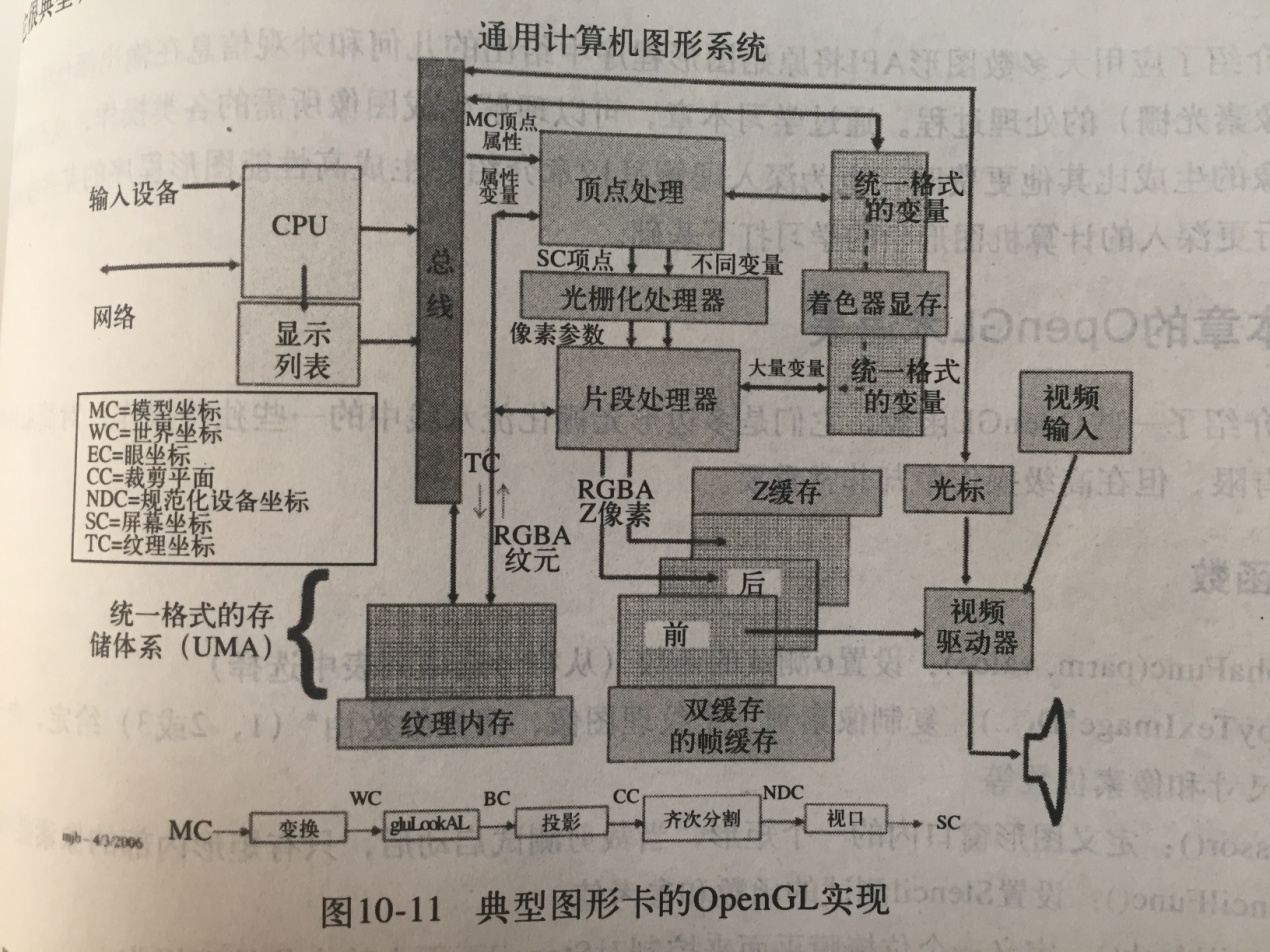
逐片段操作

使得绘制流水线变为可编程绘制流水线，带有3个可编程阶段：群组处理、顶点处理和片段处理。



OpenGL高级版，即为每个顶点提供一段程序来计算顶点属性。此类程序设计语言应独立于特定图形卡，而由图形API提供编译方式或解释执行方式为图形卡生成所需的操作。

图形卡绘制流水线实现的实例



## 图形卡的部分三维视图变换操作

交替为左右眼提供图像，使观众将两幅图像当做同一个场景的两个视图。

# 第11章 动力学与动画

动画图像是指图像不受用户或者观察者干预，可随时间的推进连续播放。

动画主要分成2类：实时动画和录制动画。

* 实时动画：通过程序的运行在屏幕上展现每一帧图像；
* 录制动画：先将每一帧图像绘制好，保存在一个可以播放的文件格式中（也可以是单独制作的电影或者视频）。

这两种动画技术都需要对模型、光照计算以及随时间变化的视点做详细的规划。为了能够高效地实施生成动画，实施动画技术采用相对简单的模型和绘制算法，来达到较高的屏幕刷新率，但是录制动画趋向于更复杂的场景模型和更细致的绘制方法。

由于采用了简单的模型和绘制方法，实时动画可能没有录制动画来的真实，而且由于图像生成速度跟不上，会使帧速率下降，从而达不到很好的实时效果。尽管这样，实时动画是由运行的程序产生动画效果，如果允许用户和动画物体做交互操作，用户就可会通过实时动画技术达到身临其境的使用效果。随着计算机速度以及图形硬件性能的不断提高，实时动画技术得到了越来越多的应用。

要理解怎样通过参数来定义场景的视图，包括尺寸、形状、位置、外观属性等。最好的方法是通过场景图来定义视图，要理解怎样逐帧地改变这些参数，来控制随时改变视图。通常采用基于时间的事件来生成新的动画帧，比如采用idle事件或timer事件，或直接使用系统时钟，在生成新的一帧图像后更新视图的参数，然后再生成下一帧图像。

## 11.1动画的分类

过程动画

动画是由计算过程驱动的。可以通过基于时间的计算，显示地控制这些参数。应用这种方法可以很容易地通过少数几个参数控制，实现简单模型的动画。根据某一科学原理计算随时间改变的对象位置或其他属性，并按其变化显示整个图像序列。直接通过计算得到动画序列每一帧的重要参数的方法是过程动画的特征。过程动画技术也可用来生成复杂的动画序列，只要能通过计算获得全部动画参数。

场景图中的动画

参数可由用户输入，或者可由定时事件或特定事件驱动的触发器来改变场景。下面给出场景图的组成部分及其变化方法：

（1）场景的几何模型：可以用参数来定义场景的几何模型，比如参数曲面方程随时间*t*的改变，场景的几何将发生变化。

（2）场景的变换：通过参数来定义场景中物体的旋转、平移和缩放，比如物件的旋转等。

（3）场景中物体的外观属性，比如颜色和纹理：可以根据需要改变颜色、纹理或者其他外观属性。比如一个曲面在不同时刻使用不同的通道。

（4）场景的视图：可以根据参数来改变视点位置、视线方向、向上方向。随着时间控制这些参数的改变可以用不同的方法观察场景，或根据需要观察场景的不同部分。

插值动画

两帧的参数向量值可以按照建模或者交流的要求，选择合适的插值方法求得，插值方法可以是线性的，也可以是非线性的，这种生成动画的方法叫做插值动画。

插值动画的应用：变形

基于帧的动画

通用简单的参数来控制整个场景的变化，然后每次更新参数，生成一帧新的动画。

帧数、关键帧、渐变。

## 11.2动画中的一些问题

11.2.1 帧速率

需要每秒24~30帧的速度才能使动画看起来连续。

动画的帧速率的数值在不同速度的机器上相差很大。当使用idle事件生成动画的时候，上述情况一定会发生。而采用timer事件生成动画会得到比较平稳的帧速率，但仍然不可预测。可使用系统事件解决上述问题。

11.2.2时间走样

扇叶逆时针旋转，达到很快的速度时，产生顺时针旋转的现象：时间走样。

11.2.3动画制作

## 11.3动画和视觉交流

## 11.4在静止帧中表示运动信息

运动轨迹法和运动模糊法。

## 11.5 OpenGL的动画例子

**在模型中移动物体：**

void animate(void) {

// 立方体的位置由基于事件的行为建模设置

time+=deltaTime; if(time > 2.0\*M\_PI) time-=2.0\*M\_PI;

cubex=sin(time);

cubey=cos(2.0\*time);

cubez=cos(time);

glutPostRedisplay();

}

**控制动画的时间：**

（1）采用idle事件来调用函数：idle事件生成每帧的时间很不均匀，动画会以完全不同的速度播放。

（2）采用timer事件调用函数：需要连续重新注册回调函数（timer回调函数每执行一次，就要注册一次），这不能让你很好的控制时间，但确实比idle事件控制时间好的多。

（3）根据直接获取的系统时钟来控制帧速率：glutGet(state)返回当前系统状态变量值，用GLUT\_ELAPSED\_TIME做参数就会得到当前的系统时钟。glutGet(GLUT\_ELAPSED\_TIME)就会得到从glutInit()开始，或者从第一次glutGet(GLUT\_ELAPSED\_TIME)开始过去了多少毫秒数。这样当生成一帧（调用glutPostRedisplay()）的时候，就可以通过调用这个函数来得到时间。当结束下一帧计算并准备生成时检查这个时间。如果已经过去足够的时间，就可以生成redisplay命令；如果过去的时间不够长，需要继续等待（可调用系统sleep事件），直到过去了足够的时间，然后再产生redisplay命令。

**移动模型的部件**：移动层次结构模型的单个部件。

**移动视点或模型的观察标架**：gluLookAt(…)函数用到视点的位置

void display(void) {

// 将视点作为变量，移动视点…

glMatrixMode(GL\_MODLEVIEW);

glLoadIdentity();

gluLookAt(ep.x, ep.y, ep.z, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

…

}

void animate(void) {

GLfloat numsteps=100.0, direction[3]={0.0, 0.0, -20.0};

if(ep.z<-10.0) whichway=1.0;

if(ep.z>1.0) whichway=-1.0;

ep.z+=whichway\*direction[2]/numsteps;

glutPostRedisplay();

}

**场景的纹理插值**

**改变模型的特征**

**生成轨迹**

**使用累积缓存**

**创建数字视频**

## 11.6用OpenGL制作动画时应注意的一些要点

处理移动视点。视图变换是模型变换的一部分，如果希望用参数控制视角，就要在合适的位置设置这些参数。在display()函数中，将模型变化矩阵设置为单位矩阵，然后调用gluLookAt()函数，变换结果保存为将来用，然后再调用旋转变换。

注意纹理走样现象。

## 11.7建议

一般来说，我们不会做高端的娱乐动画，而是关心富含信息的动画作品，即表达科学或技术工作的动画。请记住，当你观看电视或商业视频动画时，你所看到的是专业级动画，或者说是一些能感动观众的动画。这样的作品往往要求非常详细的设计，需要非常复杂的考虑，以及高端的图形系统和工具。个人级或者同事级的动画，只是表达自己的想法，即自己用，或与同事或朋友交流用，这些动画也会非常有价值。

# 第12章 高性能图形技术

建模技术：一是减少模型的多边形数量，从而简化场景，减少绘制工作量；二是增加模型特征，从而简化其绘制方法，提高绘制速度。

系统加速技术：几何压缩，如三角形条带、三角扇形、四边形条带等。

采用顶点数组和方向数组（以及颜色数组和纹理数组），是一种更为通用的系统加速技术，其速度比几何压缩技术快，但比显示列表技术慢。利用这种技术，所有的几何数据，如顶点、颜色、法向和纹理等，都只需存储一次，而使用时只需提供索引即可。在使用这种技术时，首先需要定义数组，然后将几何数据填充到数组中，最后启动需要使用的数组。在启动数组后，就可以调用glArrayElement(int )函数访问给定顶点索引的几何数组。此外，也可以使用glDrawElements(…)和glDrawRangeElements(…)函数访问顶点数组中的一组数据，从而提高访问效率。

void cube(float r, float g, float b) {

//定义顶点和颜色数据类型

color cubecolor;

point3 vertices[8] = {{-1.0, -1.0, -1.0}, {…},…};

point3 normals[8]=…;

GLubyte face1[4]={1,5,7,3}; // 面上的节点编号

…

GLubyte face6[4]={0,1,3,2};

…

glEnableClientState(GL\_VERTEX\_ARRAY); //启动客户端功能

glEnableClientState(GL\_NORMAL\_ARRAY);

glVertexPointer(3,GL\_FLOAT,0,vertices);

glNormalPointer(3,0,normals);

glDrawElements(GL\_QUADS,4,GL\_UNSIGNED\_BYTE,face1);

… //使用数组数据绘制几何图元

glDrawElements(GL\_QUADS,4,GL\_UNSIGNED\_BYTE,face6);

}

可见，只需要调用一个函数就可将顶点和法向数组发送到图形系统，绘制立方体每个面也只需要调用一个函数。

# 第15章 硬拷贝

硬拷贝：将图形计算结果输出到固定的媒介上的技术。

硬拷贝可使用物理媒介（纸张、雕塑），也可以使用数字媒体（图像、视频）。

三维图像技术

三维对象成型技术（3D打印）、STL文件

视频、数字视频

## 支持硬拷贝的OpenGL技术

OpenGL带有捕获颜色缓存的工具，利用这些工具可以随时将屏幕显示内容“导出”。关键的OpenGL函数是：

* glReadBuffer(BUFNAME): 指定读取的缓存；
* glReadPixels(…): 参数是指定读取缓存和写入陈列的方式。

一旦缓存写入阵列，就可以直接通过文件的相关技术把阵列写入文件。如果所使用的OpenGL实现或者其他工具包带有相关功能函数，则可以将陈列保存为标准格式（如JPEG）。

#define BUF\_WIDTH 512

#define BUF\_HEIGHT 512

static GLubyte bufImage[WIDTH][HEIGHT][3]

// 函数将屏幕缓冲区内容写入一个数组中，将数组名称维数传递给函数。

// 得到的文件包含原始的RGB数据，任何兼容该文件格式的应用程序（如Photoshop）均可以打开并操作文件

void savewindow(char \*outfile, int BUF\_WIDTH, int BUF\_HEIGHT) {

FILE \*fd;

GLubyte ch;

int i,j,k;

fd=fopen(outfile, “w”);

glReadBuffer(GL\_FRONT); // 设置读取前缓冲区

glReadPixels(0,0,WIDTH,HEIGHT,GL\_RGB,GL\_UNSIGNED\_BYTE,bufImage);

for (i=WIDTH; i>0; i--) { // 对于每一行

for (j=0; j<HEIGHT; j++) { // 对于每列

for(k=0; k<3; k++) { // 读取像素的RGB分量

ch=bufImage[i][j][k];

fwrite(&ch, 1, 1, fd);

}

}

}

fclose(fd);

}

## 用OpenGL生成立体图

需要左眼图像和右眼图像，两幅图像都使用RGB颜色。

首先，生成场景的左眼图像和右眼图像，保存到独立的颜色阵列中。

然后，逐个读取两幅图像中的每个像素进行组装；从左眼图像的像素中读取红色信息，从右眼图像中读取蓝色和绿色信息，然后混合为此像素点的颜色。

最后，显示混合的图像，并通过红/蓝或红/绿眼镜观看三维效果。

OpenGL实现：

首先，在后缓存中生成左眼图像并保存到阵列，像平常一样生成或载入图像，但不调用glutSwapBuffers()，这样图像就保留在后缓存中了。

用函数glReadBuffer(GL\_BACK)指定读取后缓存，然后使用函数glReadPixels(0,0,width,height,GL\_RGB,GL\_UNSIGNED\_BYTE,left\_view)将后缓存中内容读取到left\_view阵列中。

存储完左眼图像后，对右眼图像做同样的工作，也是输出到后缓存，然后存储到right\_view阵列。

现在，内存中有两个RGB颜色值的阵列。建立第3个同样数据类型的merge\_view阵列，循环遍历像素，从left\_view阵列中读取红色值，从right\_view阵列中读取蓝色和绿色值，复制到merge\_view阵列中。

现在有了混合颜色阵列merge\_view，使用glDrawPixels (width,height,GL\_RGB,GL\_UNSIGNED\_BYTE,merge\_view)将阵列写入后缓存中。接着交换缓存来显示图像，也可以将立体图像写入前面描述的文件。

## 本章的OpenGL术语表

glDrawPixels(…): 将一块像素写入帧缓存，需要大量参数指定像素写入的方式；

glReadBuffer(…): 指定一个颜色缓存作为像素读取的源

glReadPixels(…): 从帧缓存中读取一块像素到阵列中，需要大量参数指定像素读取和存储的方式；（glReadPixels读取速度很慢！）

# 参考文献

Steve Cunningham. 计算机图形学. 北京：机械工业出版社. 2008