目录

[一、基本概念: 1](#_Toc526091880)

[二、OpenGL绘图入门： 3](#_Toc526091881)

[三、OpenGL基本图元画法 4](#_Toc526091882)

[四、与鼠标和键盘的交互 5](#_Toc526091883)

[五、程序中的菜单设计与使用 6](#_Toc526091884)

[六、世界窗口和视口 6](#_Toc526091885)

[七、裁剪 7](#_Toc526091886)

[八、正多边形、圆和圆弧 7](#_Toc526091887)

[九、曲线的参数形式 8](#_Toc526091888)

[十、图形学中的向量工具 8](#_Toc526091889)

[十一、直线和平面的图形学表示法 10](#_Toc526091890)

[十二、物体变换 11](#_Toc526091891)

[十三、绘制三维场景 14](#_Toc526091892)

[十四、使用多边形网格建模 15](#_Toc526091893)

[十五、用网格逼近光滑物体 17](#_Toc526091894)

[十六、三维观察 19](#_Toc526091895)

[十七、几何体的真实感渲染 23](#_Toc526091896)

[十八、光栅显示工具 31](#_Toc526091897)

[十九、曲线和曲面设计 33](#_Toc526091898)

[二十、颜色理论 35](#_Toc526091899)

[二十一、光线跟踪基础 36](#_Toc526091900)

使用图书《计算机图形学(OpenGL版)（第3版）》

【进一步阅读】

Foley和Van Dam【Foley93】和David Rogers【Rogers98】提供了许多图形输入和输出设备的附加技术细节

【Gems】图形学珍宝，汇集全世界图形学研究者和实践者的众多新思想和珍宝

【IEEE Computer Graphics and Applications】常常介绍一些受图形学影响的新领域进展情况的文章

【TOGS】美国计算机学会图形学会刊

【Proceedings of SIGGRAPH】计算机图形学专业兴趣小组年会

书中材料下载地址www.fshiljr.com

SDL场景描述语言 见书中附录3

紫色 代表示例

!! 代表“注意”

# 一、基本概念:

SIGGRAPH：图形学专业兴趣小组，www.siggraph.org

体可视化：利用图形来展示海量的数据

图元（primitive）：组成计算机绘制的图像的基本元素

图元分类：点、线、折线、文本、填充区域、光栅图像

API：图形应用编程接口

帧：用来连续播放以形成动画的图片

**制作图像的基本元素**

1.折线

a.一条折线可以看起来像平滑的曲线

b.如果折线首尾顶点被同一条边相连，就形成一个多边形

b1.如果任意两条边除了在顶点外都不相交，这种多边形就称为简单多边形

c.重要属性：颜色，粗细，虚线的方式，粗边在端点混合的方式

c1.一般一条折线所有边都应具有相同的属性

2.文本

a.一些图形设备有两种不同的显示模式：文本模式和图形模式；文本模式只能输出内部

已有的各种字符，而且位置只能放在一些规则的网格中；图形模式的形状和位置都是随

意的，也可以以图形方式绘制文本

b.重要属性：字体，色彩，大小，间距，倾斜度

c.字符的形状可以用折线或点阵（即像素点阵）表示

3.填充区域

填充图元是指填充时的颜色或图案

4.光栅图像

由许多非常小的不同颜色和亮度的单元（像素）组成的图像

a.位图（或称像素图）：每个元素用来存储每个对应像素的值的数组整体

b.失真：当视图放大到像素可见时的状况

c.锯齿：光栅图像放大后，像素块边缘形成的失真特点

d.光栅图像制作：

•手工设计图像：手工定义每个像素点的数值

•计算机生成图像：由计算机计算从光源处或者反射光到图像（假设是相机胶卷）

每个像素处所形成的色彩和亮度

•扫描图片：即数字化照片或视频图像；将虚拟网格放在原始图像上，然后每个网

格上设备读取匹配的最相近的颜色

4.1灰度光栅图像

只有两种像素值，如0或1，或称为二值图；或称为每像素1位的图像，1比特信息足

以区分两个值

a.像素深度为n（即每个像素n位的存储量）的图像可以有2n个灰度级

常见组合：2bits/pixel=4个灰度级；4bits/pixel=16个灰度级；8bits/pixel=256个

b.条带化：降低灰度级之后，原本渐变的颜色区域变成一系列固定值的色块

4.2 彩色光栅图像

每个像素有一个彩色值，由红绿蓝不同强度组合的三色光混合而成，每个像素值是一个

3元组

a.色彩深度：用来表示每个像素色彩的比特数；3元组每个元素都有一个比特数，色彩

深度是3个数的和；常见组合是，每个元素8位，每个像素24位

**图形显示设备**

1.线画显示（早期设备）

2.光栅显示器

a.帧缓冲区：存储器中一块足够容纳要显示图片的区域，（一般在显卡里）

b.扫描控制器：自动运行（非程序控制）引发帧缓冲区通过转换器将每个像素输送到显

示平面的物理位置

c.阴极射线管（CRT），数模转换器（DAC）都是彩色视频显示系统的部件

3.视频卡/3D加速器

过去的硬件叫做“显示适配器”

a.显示卡性能：最大像素值，刷新率（每秒整个显示的图像被重新绘制多少次），支持

的颜色数

b.图形管线（graphics pipeline）：无论何时图像成形，定义场景的几何数据必须经过大

量的处理步骤，这些步骤即为图形管线

b1.GeForce256是第一个可编程的视频卡，而更早期的图形卡有固定功能的图形管道

4. 其他光栅显示设备

液晶显示器（LCD）：每个像素被一个水平格线和一个垂直格线定位，要打开一个像素则激活对应的格线，在这个像素位置制造一个电场，电场改变了LCD材料中长水晶分子的极性，允许光穿过平面，或阻止光穿过

等离子显示器：板内每个像素位置放置了一个非常小的氖管，这个管子通过电场控制开关。这种显示器不需要刷新屏幕

有源矩阵板：每个像素位置都对应一个微型晶体管的液晶显示器，晶体管响应电场，根据电场的比例调整液晶，决定显示亮度的不同级别；晶体管能够存储液晶的调整状态，于是不再需要刷新

5.硬拷贝光栅设备

激光打印机、点阵打印机（收款台打印收据使用）、喷墨绘图机、胶片记录仪

**图形输入的基本单元和设备**

1.逻辑上的输入图形基元类型

字符串：键盘

赋值：赋值设备产生一个介于0.0-1.0之间的实数值

定位：定位输入设备即鼠标，返回给程序坐标值和触发值

选取：选取输入设备用来识别图像的某个部分以便于进一步处理

2.物理输入设备类型

键盘、鼠标、按钮、写字板、操纵杆、跟踪球、数据手套

# 二、OpenGL绘图入门：

**4种OpenGL库**

基本GL库

GLUT库：GL实用工具包（the GL Utility Toolkit）

GLU库：GL实用库（the GL Utility Library）

GLUI库：用户接口库（the User Interface Library）

**5个初始化显示屏幕窗口的函数**

（放在主函数开头部分）

glutInit(&argc, argv) //初始化工具包

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB) //指明显示屏幕窗口如何被初始化

GLUT\_SINGLE 表示分配单个显示缓存

GLUT\_RGB 表示颜色由所需的红绿蓝三色的数量来指定

glutInitWindowSize(640, 480) //指定屏幕窗口初始尺寸

glutInitWindowPosition(100,150) //指定窗口位置，相对于显示器左上角的像素坐标

glutCreateWindow(“my first attempt”) //打开并显示窗口，标题为“my first attempt”

（放在主函数最后）

glutMainLoop() //事件循环处理

**5种基本事件类型**

1. 注册回调函数的glut成员名一般都带有Func

2. 用户无法用指令自己直接调用回调函数，因为回调函数是在遇到特定事件时，系统自动根据事件类型调用的；但用户可以间接调用，比如glutPostRedisplay()

glutDisplayFunc(myDisplay)：注册重新绘制事件的回调函数

!!一般将所有的绘图指令部分放在这个回调函数中，其他的回调函数只用于处理事 件和传递数据

glutPostRedisplay()：强制GLUT调用重新绘制的回调函数

glutReshapeFunc(myReshape)：珠层改变窗口形状的事件

glutMouseFunc(myMouse)：注册鼠标事件发生时所调用的函数

glutMotionFunc(myMotionFunc)：注册鼠标运动事件，鼠标按钮不释放

glutPassiveMotionFunc(myPassiveMotion)：鼠标按钮没有按下并进入一个窗口

glutKeyboardFunc(myKeyboard)：注册按下或释放键盘按键的事件的函数

**OpenGL函数的命令格式**

gl【gl库】Vertex【基本命令】2【参数数目】i【参数类型】(…)

glVertex\*() 【仅仅使用基本命令而不去考虑函数的参数特性】

**OpenGL数据类型**

|  |  |  |  |
| --- | --- | --- | --- |
| 命令后缀 | 数据类型 | 典型的C/C++类型 | OpenGL类型名 |
| b | 8位整型 | signed char | GLbyte |
| s | 16位整型 | short | GLshort |
| i | 32位整型 | int/long | Glint, GLsizei |
| f | 32位浮点型 | float | GLfloat, GLclampf |
| d | 64位浮点型 | double | GLdouble, GLclampd |
| ub | 8位无符号整型 | unsigned char | GLubyte, GLboolean |
| us | 16位无符号整型 | unsigned short | GLushort |
| ui | 32位无符号整型 | unsigned int/long | GLuint,GLenum,GLbitfield |

**OpenGL状态设定语句——其他初始化语句举例**

（一般封装在一个单独的函数里，在主函数中调用）

glPointSize(size) //设定点的大小，size是浮点变量

glColor3f(red, green, blue) //设定画图颜色，3种色值在0.0-1.0之间变化

glClearColor(red, green, blue, alpha) //设定背景颜色，alpha用来设置透明度

glLineWidth(4.0) //设定线条粗细，默认宽度是1.0

（下面的一般放在重画回调函数中）

glClear(GL\_COLOR\_BUFFER\_BIT) //将整个窗口清除成背景色，清屏

# 三、OpenGL基本图元画法

**概述**

1.画图语句执行的机制是：通过将绘图所需坐标（glVertex2i）发送到图形管道中，按照约定的处理方式进行处理（glBegin(…)），然后统一输出结果（glFlush或glutSwapBuffers）

2. “将绘图所需坐标发送 （glVertex2i）”的语句可以用循环体来发送

3. “处理方式的选择（glBegin(…)）”语句可以用条件语句选择，即不同的条件表达式结果，不同的处理方式语句生效

4. 画图元的语句可以封装成函数，进行更复杂的图形绘制，甚至可以为函数加入参数，进行更为简单普适的绘画

5. 送入图形管道中的坐标不一定是整型（glVertex2i或glVertex2f），这个坐标是窗口坐标，是连续的世界坐标，而非是像素坐标（见后面“窗口和视口”部分）

**画点的语句/画图语句——即回调函数的语句**

glBegin(GL\_POINTS); //如果画其他图元：GL\_LINES, GL\_POLYGON

glVertex2i (100, 50); //点的坐标表示

glEnd();

**画直线的语句**

glBegin(GL\_LINES);

glVertex2i(40,100);

glVertex2i(202,96); //坐标须成对（偶数个），每对坐标之间画一条独立直线

… //（不是每两个相邻坐标画一条线，只是两点一线指令的平行重复）

glEnd();

**画折线**

glBegin(GL\_LINE\_STRIP); //如需将折线首尾相连，用GL\_LINE\_LOOP，自动连接

glVertex2i(20,10);

glVertex2i(50,10);

glVertex2i(20,80); //一个有序点列表来描述

glEnd();

（用GL\_LINE\_LOOP绘制的多边形不能填充颜色或图案，需用GL\_POLYGON）

**自定义函数画线p57**

moveTo(x,y)

lineTo(x,y)

GLintPoint CP

**画边校正的矩形（边与坐标轴平行）**

（边校正矩形的长宽比：宽/高）

glRecti(Glint x1, Glint y1, Glint x2, Glint y2); //用两对角点绘制，以当前颜色填充

**画填充多边形**

（凸多边形：多边形中任意两点连线完全位于多边形内部）

（OpenGL支持用图案或彩色填充一般的多边形，但必须是凸多边形）

glBegin(GL\_POLYGON);

glVertex2f(x0,y0);

glVertex2f(x1,y1);

…

glEnd(); //通过顶点列表来绘制一个凸多边形，以当前颜色填充

**其他图元**

GL\_TRIANGLES：一次使用3个顶点，每次绘制一个独立的三角形

GL\_QUADS：一次使用4个顶点，每次绘制一个独立的四边形

GL\_TRIANGLE\_STRIP：基于三个一组的顶点（每相邻三个顶点），绘制一系列三角形

GL\_TRIANGLE\_FAN：基于三个一组的顶点（始终选取第一个点），绘制一系列与第一 个点相连的三角形

GL\_QUAD\_STRIP：基于四个一组的顶点（每相邻四个顶点），绘制一系列四边形即四 边形条带，奇数号顶点组成条带的一边，偶数号顶点组成另一条边

# 四、与鼠标和键盘的交互

**用鼠标交互（点击）**

1. 每次用鼠标进行一个动作，系统会自动调用鼠标回调函数，并向函数发送参数

2. 在鼠标回调函数中，经常使用static局部变量来保留上次回调函数的数据以便后用

3. 回调函数原型只是一个结构性要求，在定义/声明函数时，形参名称可以自定义

回调函数注册：glutMouseFunc(myMouse)

回调函数原型：void myMouse (int button, int state, int x, int y)

button取值范围：GLUT\_LEFT\_BUTTON, GLUT\_MIDDLE\_BUTTON, GLUT\_RIGHT\_BUTTON（即发生动作的是哪个按键）

state取值范围：GLUT\_UP, GLUT\_DOWN（即按下或弹起鼠标按键）

x,y含义：x是距离窗口左边的像素数，y是距离程序窗口顶端的像素数

**鼠标运动**

回调函数注册：

glutMotionFunc(myMovedMouse) //按住一个或几个鼠标按钮在窗口内移动时

glutPassiveMotionFunc(myMovedMouse) //没有按下按钮时的移动

回调函数原型：void myMovedMouse(int x, int y) //x，y都是事件发生时鼠标位置

**键盘交互**

1. 大多数键盘交互回调函数的实现方法都由很长的switch语句组成，一个case对应一个感兴趣的键（如快捷键），记住每个case要用break来结束（如果该case无其他指令跳出）

回调函数注册：glutKeyboardFunc(myKeyboard)

回调函数原型：void myKeyboard(unsigned int key, int x, int y)

key：就是按键对应的ASCII值

x，y：事件发生时鼠标所在位置

# 五、程序中的菜单设计与使用

**GLUT菜单**

1.添加菜单和菜单条目（在主函数中）

glutCreateMenu(processMenuEvents); //创建菜单并注册菜单处理函数

glutAddMenuEntry(“Red”,RED);

… //添加菜单条目的显示名称和对应的枚举常量

glutAttachMenu(GLUT\_RIGHT\_BUTTON); //菜单系在鼠标按钮上（可选左中右）

2.定义菜单处理函数（一般为switch结构）

void processMenuEvents (int option) {

switch (option){

case RED: red=1.0; green=0.0; blue=0.0; break;

case…} //根据不同的选择，形参有不同的枚举值，进行不同的指令处理

glutIdleFunc()

**GLUI菜单**

比GLUT更加复杂，暂不介绍

# 六、世界窗口和视口

**概述**

1. 将程序中用于描述对象几何信息的数值，和用于表示对象在屏幕中大小的位置和数值区分开来。前者为建模(modeling)，后者为观看(viewing)

2. 世界坐标系：场景中物体在实际世界中的坐标

3. 世界窗口(或简称窗口world window)：世界坐标系中定义的一个对齐的矩形；世界窗口指明要绘制世界的哪一部分，之外的内容不绘制

4. 视口(viewport)：在显示器屏幕窗口上定义一个对齐的矩形，视口内容与窗口内容相同，即视口用于在显示器屏幕上对窗口内容表达

5. 屏幕(screen)坐标和屏幕窗口(screen window)坐标：像素在显示器上的坐标和像素在程序窗口（和屏幕窗口含义相同）中的坐标；前者用于初始化程序窗口glutInitWindowSize和glutInitWindowPosition，后者用于在程序窗口内定位视口；前者原点在显示器左上角，后者原点在程序窗口左下角

6. 窗口的内容通过映射完全呈现在视口中，意味着必要时存在图象的等比例（线性）拉伸和压缩；窗口内容是真实图象（却只能通过视口来间接虚拟表达），视口内容是可见的，但却是变形的，同时清晰度受分辨率限制

7. 漫游：相机的术语，也叫作“全视”；即镜头（窗口）顺着场景滑行，在不同的时间拍摄不同的部分

**窗口和视口**

1.映射关系

x,y是世界窗口中对象在世界坐标系中坐标；sx,sy是视口内像素在屏幕坐标系中的坐标

,

,

,

2.窗口和视口设定

（二维条件的函数原型）

窗口设定：void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)

视口设定：void glViewport(Glint x, Glint y, Glint width, Glint height)

2.1窗口设定的参数是窗口四个边界在世界坐标系下的坐标

2.2 视口设定的参数是视口左下角的屏幕坐标（基于建立的窗口中）和视口的宽度与高度（均为像素尺度）；视口内的坐标沿用窗口的定义坐标（即同样使用世界坐标的同一部分区间）；设定一个视口后，将暂时局限于该视口进行作图，虽然仍可以在视口外的世界坐标空间中作图，但是并不会显示出来（相当于被裁剪掉）

2.3 gluOrtho2D()必须在glMatrixModel和glLoadIdentity之后（在三维作图部分解释）

2.4 如果没有使用glViewport命令，默认的视口即为整个屏幕窗口

2.5 视口的高度不能为负数，但窗口的边界坐标可以颠倒达到相同目的

**双缓存显示模式**

1. 定义：当用户注视当前图像时，在一个“屏幕外内存”绘制新的图像，然后瞬间将绘制好的新图像显示在显示器上

2. 特点：使用双缓冲或双缓存可以让当前帧和下一帧快速平滑转换，在函数glutSwapBuffers作用下，显示不断从一个缓冲区转换到另一个缓冲区，新图像在当前不可见的缓冲区中绘制，直到绘制完成才切换。用户不会看到显示器清屏操作和新图像一行行被绘制的过程

3. 设置语句和相关命令

glutInitDisplayModel(GLUT\_DOUBLE|GLUT\_RGB) //GLUT\_SINGLE对应glFlush()

glutSwapBuffers() //对应双缓存模式，将缓存中图片转移到屏幕窗口上，替换glFlush()

# 七、裁剪

见算法整理文档

# 八、正多边形、圆和圆弧

**正多边形**

1.简单多边形：如果一个多边形没有两条边互相交叉；或只有相邻的边能相互接触，而且只在它们的端点处接触

**正n边形的变种**

1.花环：每一个顶点都和其他所有顶点相连的正多边形

**圆和圆弧**

1.三种画圆的算法

圆心和半径

圆心和圆上一点

圆上三点

**曲线的逐次细化**

Koch曲线和Koch雪花 p105/144

**自相似**

1.定义：在图像的某一部分放置一个小的窗口，观察其粗糙的形状，然后再将窗口无限缩小进行观察，新窗口内的曲线仍然和原来的曲线有着类似的形状和粗糙程度；一般来说只有极限情况下才存在着真正的自相似

2.图形学中的近似：只要保证在最大程度上放大时，最小的线段长度也比两个像素之间的距离要小即可

# 九、曲线的参数形式

**直线**

**椭圆**

**抛物线**

**双曲线**

**绘制参数曲线**

算法：用非常紧凑的间隔采集t的样本，从而仅使用直线段来很好的近似曲线进行绘制；曲线变化剧烈的间隔，采集点应更密集，平滑的地方可以稀疏

**极坐标形状**

圆：f(θ)=K

心脏线：f(θ)=K(1+cosθ)

玫瑰曲线：f(θ)=Kcos(nθ) (n指定玫瑰的花瓣个数)

阿基米德螺线：f(θ)=Aθ

圆锥曲线：f(θ)=1/(1±ecosθ) (e为离心率，e=1抛物线，e>1双曲线，0≤e<1椭圆)

对数螺旋线（等角螺旋线）f(θ)=Kexp(aθ); a=cot(α) (在一个常数角度α处切割所有径向线)

# 十、图形学中的向量工具

**概述**

1.左手系右手系判断：拇指指向z正方向，四指由x正向到y正向

2.右手系应用：数学、物理、工程；用来建立对象模型

3.左手系应用：用于处理图形学中观察系统和摄像机系统

**向量基础**

1.两个点的差是一个向量

**向量的线性组合**

1.线性组合定义：

2.向量的仿射组合：线性组合的系数之和等于1

2.1为了使两个向量成为一个仿射组合，可以令一个系数为任意标量t，另一个为(1-t)

3.向量的凸组合：线性组合的系数之和为1，且所有系数均为非负值

**向量的度量和单位向量**

1.向量归一化：归一化后的向量表示为

2.归一化一个向量之前，必须确认它的长度为非零

**向量的点积**

1.定义：

2.性质：

2.1对称性：

2.2线性：

2.3同质性：

2.4

3.在仿射变换下，点积可以通过将第一个向量转置成行矩阵，第二个保持列矩阵形式，以矩阵相乘的方式表达，即，后面n,v是矩阵，而且使用齐次坐标

**两个向量的夹角**

重要关系：

**正交性**

1.名称：正交(perpendicular)=直交(orthogonal)=垂直(normal)

2.定义：给定，则与**a**逆时针正交

3.性质：

3.1线性：；

3.2

4.正交点积性质：p172/133

4.1反对称性：

**正交投影和点到直线距离**

把**c**分解到**v**和方向上

点C到经过点A且方向为**v**的直线的距离是

**投影的应用：反射**

**a**为入射光向量，**n**为单位法向量，**r**为出射光向量

则

**向量的叉积**

1.性质：（没有结合律）

1.1同质性

1.2线性

1.3反对称性

2.几何解释

3.仿射变换中的性质

**求平面法向量**

1.一般的叉积运算用到减法，所以计算时会受到计算机表示数字精度的影响，当两个向量的夹角很小时尤其明显

更加鲁棒的法向量计算法见算法整理

**坐标系统和坐标框架**

1.坐标框架：一个叫做原点的特定点ϕ，和三个两两正交的单位向量；和坐标系统相比，将原点明确化

2.坐标框架中图元表示

2.1向量：

2.2点：

3.齐次表示

,

3.1 OpenGL中所有的运算都是在四维齐次坐标下进行的，所有的二维三维点或向量均被自动转换为四维齐次坐标

3.2 向量的齐次表示适用于向量的一切线性组合

**点的仿射组合**

1.概念：点的相加、点的任意线性组合是不合理的，存在几何错误或者说坐标系选择相关；只有点的仿射组合才是一个合理的点，与坐标系选取无关（合理的点必然与坐标系选取无关，是客观存在的固定点，坐标可以变化但是空间位置不能变）

1.1 若点的线性组合系数之和为0，组合成一个向量

2. 点的仿射和都可以写成一个点加上一个向量，反之亦然，证明仿射和的合理性与存在性

3. 多个点或多边形顶点的位置的平均值，即为其质心，也是一个仿射组合

**两点的线性插值**

1.定义：；获得A点与B点的线性插值

2.一维的对于数值的线性插值函数声明为float lerp(float a, float b, float t)；二维三维的对于点的线性插值声明为Tween()；当参数在0到1之间变化时，称为内插，否则称为外插

**三点的二次线性插值**

也称为ABC三点的贝塞尔曲线

**四点的三次线性插值**

将分成四个部分，作为四个点的系数

# 十一、直线和平面的图形学表示法

1.曲线的参数表示和曲线上的运动路径是不同的，后者可以表示出曲线上的点的运动速度，对于一条给定的运动路径，有无穷多种参数表示方法

**直线的参数表示**

1. C为直线上一点，**b**为直线方向上的向量

2. t的取值范围不同可以决定是线段，射线，或直线

3. 直线上点的运动速度是 |**b**|

4. 直线的参数表示可以写成坐标形式，因为等式右边结果是一个点

**直线的点-法向量表达形式**

**n•**(R-C)=0

1.R为直线上任意一点，C为直线上一个已知点，**n**为法向量  
2.这种表达形式只能用于2维

3. 这种形式和一般形式等价：Ax+By=1，则(A,B)即为直线法向量，因此一般形式Ax+By+Cz=1，也不能用于三维表示直线，而是平面

**平面的参数表示**

P(s,t)=C+s**a**+t**b**

P(s,t)=sA+tB+(1-s-t)C

1.第一种是平面上一点和两个不共线向量的组合，s和t是两个参数

2.第二种是平面上三点的仿射组合形式

3.可以通过限制s,t的取值范围来截取一个平面面片

# 十二、物体变换

**点和物体变换（二维）**

1.定义

1）映射函数：即一个变换，写作T()，将一个点按照某一指定的等式或算法变为一个新的点，写作Q=T(P)

2）基本变换：每次只产生一种变换效果的变换，如纯缩放、剪切、旋转等

2.仿射变换

在齐次坐标系下，即线性变换加上一个平移量，表达式

1）仿射变换可以很容易对物体实施缩放、旋转、剪切和平移，所有的线性变换都可以表示成仿射变换（在齐次坐标系下）

2）一系列仿射变换可以组合成一个单一的仿射变换

3）仿射变换可以用一个紧凑的矩阵形式表示出来

3.平移

1）如果平移的是向量，那么平移没有任何效果，在表达式中也可以体现

2）a，b即为两个方向上平移的量

3）如果不使用齐次坐标系，则没有办法用矩阵操作进行平移

4.缩放（或特殊情况下的“反射”）

1）特别的，当缩放因子S为负数时，表示相对某个轴有反射（对称）作用

2）当两个缩放因子一样时，称作均匀变换；或者叫做关于原点的放大，放大系数为|S|；若缩放因子不一样，称为非均匀缩放

3）这是一个关于原点的缩放

4）关于任意过原点直线的反射，变换矩阵为（β为直线与x轴正方向夹角）

5.旋转

1）如果旋转角度是正值，则旋转方向是逆时针（CCW）

2）这是一个纯粹绕原点的旋转矩阵

3）关于任意点的旋转

6.剪切

1）这是一个x轴方向上的剪切，将矩阵转置后则为y轴方向上剪切

7.逆变换

仿射变换的逆变换矩阵就是原变换矩阵的逆矩阵

8.组合变换

1）仿射变换的组合仍然是仿射变换

2）当使用齐次坐标时，仿射变换组合可以通过矩阵乘法实现

**仿射变换的性质**

1.仿射变换保持点的仿射组合

2.仿射变换保持共线和共面关系

内在性：仿射变换前，如果一个点处于另两点之间，变换后仍然在该两点之间

3.保持直线和平面的平行性

4.矩阵的列揭示出了变换后的坐标系

变换矩阵按列拆分成若干向量和一个点之后，向量部分是基矢量变换后的矢量，点是原点变换后的位置（从原坐标系的视角下看去，即使用原坐标系）

5.仿射变换保持相互比例

即保持线段长度的相互比例

6.如何影响一个图形的面积

仿射变换后，图形面积变为原来的|det M|倍（即变换矩阵行列式的绝对值）

7.每个仿射变换都由基本仿射变换组成（顺序由右至左）

二维：M=（剪切）（缩放）（旋转）（平移）

三维：M=（缩放）（旋转）（剪切1）（剪切2）（平移）

8.仿射变换并非是坐标系无关的

**三维变换**

1.平移（最后一列即为平移向量）

2.缩放

3.剪切（最基本的）

4.旋转（分解成简单旋转）

x-roll绕x轴旋转β角度（逆着x轴正方向看去，逆时针方向为β正方向）

y-roll

z-roll

5.绕任意轴**u**（**u**为该轴的方向向量）旋转β角

1）经典方式：先进行两个旋转使**u**和z轴重合，然后进行z-roll至β角度，然后逆旋转回到原来**u**方向

2）构造性方法：在旋转平面建立一个二维坐标系统，使用由P点表示出的基矢量来表示Q

c=cosβ；s=sinβ

6.找出旋转轴和角度

7.三维仿射变换性质

同二维性质，只是面积变化改为体积变化

**坐标系变换**

坐标系经过仿射变换M后，同一个点的新旧坐标关系 Pold=MPnew

推导：

（一般向量推导有两种表示法，坐标表示和基向量表示；坐标表示法必须保证所有坐标是相对于同一坐标系而言，基向量表示不必，见下）

P点可表示为

使用世界坐标（旧坐标）：

**OpenGL中使用仿射变换**

1.基本原理

OpenGL采用一个模型视图矩阵（modelview matrix）或称为CT（当前变换），每个通过图形管线的点都要乘以这个矩阵，然后才会进行窗口到视口的变换操作

2.基本指令

glMatrixMode(GL\_MODELVIEW); //通知OpenGL对CT矩阵进行操作

glLoadIdentity(); //将上一指令中指定的矩阵初始化为单位矩阵

变换指令（每个指令使用之前仍需用glMatrixMode指定所操作的矩阵）

glScaled(sx,sy,sz); //对CT后乘一个缩放矩阵，Scaled的最后一个字母代表double

glTranslated(dx,dy,dz); //对CT后乘一个平移矩阵

glRotatef(angle,x,y,z); //对CT后乘一个旋转矩阵，xyz代表转轴方向向量（转轴过原点， 方向向量自动被归一化），angle代表绕轴转角

glRotated(angle,x,y,z); //同上，不同数据类型

3.注意事项

指令的操作是对矩阵后乘，但是操作的原理是对图形的变换，因此为了达到所需的变换顺序，在指令的输入顺序上应反过来

4.变换栈

1）原理：栈顶的矩阵就是实际的CT

2）指令：（使用之前必须先用glMatrixMode指定要操作的矩阵）

glPushMatrix(); //将所指定的矩阵压入栈顶（一个拷贝）

glPopMatrix(); //将栈顶矩阵弹出丢弃

glGet(GL\_MODELVIEW\_STACK\_DEPTH) //函数返回栈中元素个数，大于1才能进行弹出操作

# 十三、绘制三维场景

**基本概念**

1.视景体：一个平行六面体，四个侧面由窗口的边界决定，前后两个面称为近平面和远平面；视景体中的点沿着与z轴平行的直线投射到窗口上，窗口平面即为x-y平面

2.平行投影和透视投影：透视投影更加真实，近大远小

3.图形管道（简化版）：

→模型视点矩阵VM→投影矩阵P→clip→视口矩阵Vp→视口

从坐标角度理解：

世界坐标系→视点坐标系（或称摄像机/眼睛坐标系）→为裁剪设计的过渡坐标系→适宜正确显示的坐标系

4.模型视点矩阵（CT或modelview）

结合两种效果，一是对物体的一系列模型变换，二是对空间中摄像机位置和朝向的变换；可以理解为两个矩阵先后作用，模型矩阵先，视点矩阵后，即VM

M矩阵用于对图形本身进行变换，V矩阵用于把坐标系变换成视点坐标系

5.视点坐标系

摄像机的视点位于原点，视景体和z轴对齐，视景体边界和xyz轴平行

6.投影矩阵glMatrixMode(GL\_PROJECTION)

以一种特殊的方式旋转平移每一个顶点，使得在视景体中的所有点将来出现在一个每一维都从-1到1的标准立方体中。这个矩阵将视景体压缩为中心位于原点的立方体。这个立方体各个面可以有效裁剪物体。使用透视投影时，该矩阵作用更大

7.视口矩阵

把标准立方体中裁剪剩余的部分映射到一个三维的视口。标准立方体变为长方体，xy坐标等于视口，同时z坐标从0到1保持每一点的深度信息

**OpenGL中的建模和视点工具**

1.设置模型矩阵M

见OpenGL仿射变换部分的基本指令

2.设置视点矩阵V

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(eye.x,eye.y,eye.z,look.x,look.y,look.z,up.x,up.y,up.z); //视点矩阵设置应在模型矩阵之前，eye和look分别定义眼睛位置和观察点位置，两点位置之差(eye-look)即为眼睛坐标系的一个基向量**n**，up定义眼睛坐标系的另一个基向量；

注意：一般估算时先猜测up为(0,1,0)，即与y轴平行，在设置矩阵时即可使用up=(0,1,0)，函数会自行计算并创建合理的**u**,**v**,**n**

3.设置投影矩阵

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(left, right, bottom, top, near, far); //创建一个平行六面体视景体，LR对应x，BT对应y，NF对应z，这些坐标对应的是眼睛坐标系，因为设置投影矩阵时，坐标系已经变为眼睛坐标系，默认的摄像机位置是原点，朝向-z方向

p262

# 十四、使用多边形网格建模

**定义多边形网格**

1.基本思想

1）每个多边形面片的每个顶点都单独指定一个法向量比单纯给整个面片指定一个法向量效果要好，除非物体的每个面本来就是平面，而非用平面近似曲面得到

2.表示方式

1）使用三个列表，顶点列表记录网格中不同顶点的位置，每个顶点有一个索引（编号），法线列表记录模型中存在的不同的法线向量，每个向量有一个索引，面片列表通过记录每个面片包含的顶点（顺序是“站在物体外部观察，以逆时针方向遍历多边形”）与对应的法向量来表示每个面片（只记录顶点与法向量的索引来节约空间），每个面片有一个编号

2）如果面片带洞，则可增加两条重合的虚拟路径进入洞的边界，遍历方法类似面积分

3）其他保存3D网格信息的方式举例：qs, 3DS, VRML, PLY, MS3D, OBJ

**网格的性质**

并非是固有不变的性质，而是可以描述某个网格特性的角度

1.实体性：网格的面片包围一块有限体积的空间

2.连通性：不连通的网格一般表示的对象数多于一个

3.简单性：是一个实体，并且不含有洞

4.平直性：网格是平直的则每个面片都是平面多边形，三角形天生就是平直的

5.凸性：连接此物体任意两点的线段也包含在物体中

**程序实现**

p287

**多面体概述**

1.定义：多面体是一个连通的网格，它由简单的平面多边形组成并且包围了一块有限体积的空间。

1）每条边恰好由两个面共享

2）在每个顶点处至少有三条边相交于此

3）面之间不能相互贯穿，两个面要么不接触，要么只沿着公共边相接

Schlegel图：p252/291

2.棱柱

棱柱是把一个多边形沿着一个向量扫掠或拉伸的结果，如果向量本身垂直于原多边形所在平面，则为直角棱柱

3.柏拉图实体（p253/292-p257/296）

多面体的每一个面片都相同，且为正则多边形，称为正则多面体（或称柏拉图实体）；只有5种。面片是等边三角形的有3种（正四面体，正八面体，正二十面体），正方形的有一种（正六面体），正五边形的有一种（正十二面体）

柏拉图实体的面法向量可以通过连接实体的中心和每个面片的中心来简易获得

4.对偶多面体

对偶多面体的每个顶点是原多面体每个面片的中心，相邻的两个面片的中心相连构成了对偶多面体

1）正四面体本身对偶

2）正立方体，正八面体互为对偶

3）正十二面体，正二十面体互为对偶

5.阿基米德立体（半正则立体）p258/297

每个面都是正则多边形，但是不是每个面都相同，只存在13种阿基米德立体

**拉伸形体**

定义：给定一个多边形作为底面，然后对该多边形使用变换得到顶面，然后将对应顶点相连

1.应用一：“铺砖块”组合

OpenGL只能可靠地绘制渲染出凸多边形，可以先将多边形分解离散化为凸多边形，然后再拉伸每个凸多边形；缺点是会多余绘制相接的面

2.应用二：四边形条带

四边形条带是“铺砖块”的一个特例，不会造成多余的面被绘制

3.带“扭结”的拉伸

即对拉伸后的底面使用仿射变换，来产生放大缩小或扭转的效果

4.管状体与蛇形体

每一次拉伸都使用自己原来的变换结果，然后将端面相接形成管状体

Frenet标架

5.离散扫掠形成的旋转曲面p268/307

# 十五、用网格逼近光滑物体

**曲面的表示**

曲面的参数形式：P(u,v)=(X(u,v),Y(u,v),Z(u,v)) //曲面和平面一样，需要两个参数

u/v轮廓线：固定一个参数不动，变化另一个参数得到的曲线

曲面的隐函数等式：F(x,y,z)=0 //隐函数形式为F(x,y,z)=…

对于一个有意义的封闭曲面：F(x,y,z)>0表示曲面外部，反之表示内部

**曲面的法向量（参数形式）**

**曲面的法向量（隐式形式）**

如果同时拥有曲面的隐式形式和参数形式，则先用隐式形式求法向量，再用u,v参数替换x,y,z

**仿射变换的影响**

参数形式变换后

隐函数形式

法向量

**三个通用形体**

1.通用球面

定义：球心在原点半径为1的球面称为通用球面

隐函数形式：F(x,y,z)=x2+y2+z2-1 或F(P)=|P|2-1

参数形式：P(u,v)=(cos*v*cos*u*,cos*v*sin*u*,sin*v*) //u为经度，v为纬度,

参数形式2：P(u,v)= //u表示经度，v代表z

法向量：**n**(x,y,z)=(x,y,z) 或 **n**(u,v)=**p**(u,v)

2.通用柱面

定义：中心轴与z轴重合，横截面是半径为1的圆形，沿着z从0拉伸到1

引申为一端逐渐变小的柱面（随着z增加，s为变化端的半径）

隐函数形式：F(x,y,z)=x2+y2-(1+(s-1)z)2, 0<z<1

参数形式：P(u,v)=((1+(s-1)v)cosu,(1+(s-1)v)sinu,v)

顶面参数形式：P(u,v)=(vcosu,vsinu,1)

法向量：**n**(x,y,z)=(x,y,-(s-1)(1+(s-1)z)) 或 **n**(u,v)=(cosu,sinu,1-s)

3.通用锥面

即一端变小的柱面当s=0的时候的特例

**构造弯曲曲面的多边形网格**

1.概述

•把曲面替换为一堆三角形和四边形的集合，这些多边形的顶点位于曲面自身，并且用线段相连。每个顶点指定曲面的法向量，这个法向量是在此顶点处原始曲面的法线方向（注意和多面体网格的区别）

•nSlices和nStacks：沿着赤道方向或者环绕z轴切成nSlices条（经度用参数u表示），从南极到北极或沿着z方向分成nStacks层（高度或者纬度用v表示）

2.通用球面和通用柱面的网格

p277/316

3.直纹面

定义：一个曲面，在其上的任意一点都可以找到一条通过它的直线使之完全落在曲面上

举例：锥面和一端逐渐变小的柱面

•锥面：（P0为尖端，P1(u)为底面曲线轮廓）

锥面上的点：P(u,v)=(1-v)P0+vP1(u) //将曲线线性拉伸到一个点导致的结果

•柱面：（不一定是圆柱，一般的不一定封口的柱面）

柱面上的点：P(u,v)=P0(u)+**d**v //**d**为柱面的拉伸方向向量

条件：P0(u)必须在一张平面内

•其他直纹面：

双螺旋面（两条螺旋线）、摩比斯带子p281/320

4.旋转曲面

定义：在xz坐标平面内有一条轮廓线，用参数表示为C(v)=(X(v),Z(v))，把轮廓线围绕z坐标轴进行旋转并且用参数u控制旋转的角度

•经线：曲线绕着轴旋转在不同位置形成的曲线（曲面和过旋转轴的平面形成的交线）

•纬线：v为常数的圆环轮廓线（曲面和垂直于旋转轴的平面形成的交线）

参数表示：P(u,v)=(X(v)cosu,X(v)sinu,Z(v))

•C(v)绕任意轴**r**扫掠形成的旋转曲面（齐次坐标系）,Rr(u)为任意轴旋转矩阵

法向量：

构造网格的方法：按照nSlices,nStacks的思想

5.基于3D曲线的管状体

同前，只是在顶点处计算原始曲面的法线

6.基于双变量显式函数的曲面

定义：一个曲面在其中一个维度上是单值的，即比如x,y坐标给定，那么z的坐标只有一个值视为此种情况（类似于高度场）；可以用双独立变量显式函数来表示

参数形式：P(u,v)=(u,f(u,v),v) //其中一例

**粒子系统与基于物理的系统**

1.粒子系统概述

在动画中应用，一个场景包含很多基本粒子（通常简单表示为圆点），圆点存储于3D空间中，并且具有颜色、生命周期、速度和大小等属性；在动画中，根据这些属性来进行显示

2.基于物理的系统概述

和粒子系统类似，用于动画中对于物体进行真实的模拟，

# 十六、三维观察

**透视投影概述和详议摄像机**

1.基本概念

•视点：或叫观察参考点，VRP

•投影线：从点P到视点的直线叫做P点的投影线（所有的投影线都交汇于视点）

•视平面法线（VPN）：视景体的轴，和近平面、远平面和视平面垂直；视点和look点的差eye-look定义了向量**n**（归一化后）和VPN

•观察角度（θ）：视景体的张角

•投影点：P’是P的投影线和视平面的交点，也是P的投影

•翻转（度）：飞机关于纵向轴的旋转，关于水平面的旋转角度，决定了机翼的倾斜程度

•倾斜（度）：飞机的纵向轴-**n**和水平面的夹角，决定了飞机是攀升还是俯冲

•偏离（度）：也就是航向或方位角

•无翻转摄像机：通常设置的是零翻转摄像机，即其**u**轴是水平的和世界坐标y轴垂直

2.摄像机的定位和定向

平行投影和透视投影的唯一区别就在于投影矩阵，它定义出了视景体的形状；其他矩阵如视点变换矩阵，定义方法类似

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(eye.x,eye.y,eye.z,look.x,look.y,look.z,up.x,up.y,up.z);

注：可以用gluLookAt()这样的专门功能的函数来更改一些特定矩阵的值，也可以用glLoadMatrixf(m)这样的通用矩阵操作函数来更改当前被操作矩阵的值（后乘），f代表数据类型float，括号内的唯一参数是乘数矩阵，用“一维16元素”的数组表达

3.gluLookAt()函数

函数用来计算得出**n,u,v**，计算后进行归一化

3.1 **up**向量是一个用户自定义的用来指示摄像机上方的示意性向量，一般对于零翻转摄像机设定up=(0,1,0)，因此求出来的u轴一定与y垂直

3.2 当up=(0,1,0)

3.3 视点变换矩阵V

4.在程序中指定一个摄像机

用一个Camera类来记录摄像机参数和操作摄像机，代码始于p302/341

5.p345 SDL

**三维物体的透视投影**

1.详议图形绘制管道

顶点和顶点法向>>Modelview matrix(VM)>(eye coordinates)>Projection matrix(P)>(Clip coordiates)> Clip>>Perspective division>(Normalized device coordinates)>Viewport matrix(Vp)> (window coordinates)>显示点

•为了提升效率，一般将所有的操作矩阵合成为一个单一矩阵，再和所有顶点（成千上万）相乘

2.点的透视投影

在视点坐标系下，P点在近平面上的投影坐标为（近平面为二维）：(近平面距离原点为N)

1）所有平行于近平面的平面都可以作为视平面，因为仅仅改变图像的大小，而实际上显示出来的图像大小只由视口决定

3.直线的透视投影

1）直线投影仍然为直线，和视平面平行的平行线投影后仍然平行，不平行于视平面的平行线投影后不平行，而是相交于“灭点”

2）不平行于视平面且方向相同的所有的平行线共享一个灭点，即灭点位置只和平行线的方向有关，而与平行线的位置无关

直线的参数方程：

代入投影表达式，在t趋向于无穷大时，即为灭点坐标（视平面上坐标）

3）为了防止出现错误，一般对于直线中比近平面更靠近视点的部分会在投影之前就被裁剪掉

4.增加伪深度

概念：投影到同一个平面甚至同一个点上的不同对象在实际空间中距离视平面一般不相同，伪深度用以区别对象在空间中的前后位置，可以识别被遮盖的部分以采取处理

方法：将投影坐标增加一个z\*维度来记录伪深度

a,b的选取应使伪深度处于-1到1之间；所以定义当Pz=-N时，伪深度为-1，Pz=-F时，伪深度为1（F为远平面的位置）

注意：当-Pz越来越接近F时，伪深度变化越来越慢变化量很小，会造成计算机无法识别，尤其是当N远小于F时，可以通过适当提高N的值改善

5.使用齐次坐标

概念：使用齐次坐标后，就可以用坐标系（包括原点）作为一组基来表示点和向量，并且可以用矩阵表示仿射变换，图形绘制管道中的操作都可以用矩阵乘法完成，这也要求伪深度表达式拥有和坐标一致的分母；矩阵乘法的硬件运算很高效，通过这种方式也就提高了图形处理的效率

概念扩展：把一个点的所有齐次坐标分量同等程度地放缩后依然表示这个点，但是该点的普通坐标仍然必须将第四个分量化归为1后获得

6.透视变换

概念：变换矩阵第四个行向量不是(0,0,0,1)，即不是仿射变换的广泛变换；由于并未降低点的维度，所以是变换而不是投影

操作：

•先进行一般透视变换，变换后视景体变为平行六面体（如上式）

•再进行伸缩和平移操作，将视景体变为一个正立方体（每一维都从-1到1），称为正则视景体(CVV)，完整的透视矩阵包括这两部分操作：（RLTB为窗口的四个边界名称缩写，代表其位置）

•进行裁剪操作（裁剪算法见算法整理，使用正则视景体裁剪最大化的简化了视景体边界的描述方式，只有-1和1，与窗口的大小选择也没有关系，因此提高了裁剪效率）

•执行隐藏面消除操作

•将第四个分量归一化，这个步骤称为透视除法（裁剪之后再做除法可以避免出现除数为零，即z坐标为零的点或与视点共面的点；在其他方面也提高了效率，比如判断与CVV位置关系时，只需判断符号正负即可）

•之后再进行一次视口变换（将xy坐标转化为屏幕坐标系，即屏幕上显示位置，z坐标变换到[0,1]区间，仍作为深度）和正交投影

透视投影=透视变换+正交投影

性质：

1）通过视点的直线（平面）被映射为平行于z坐标轴的直线（平面）

2）与z坐标轴垂直的直线（平面）映射后依然垂直于z坐标轴

3）具有保平面性和保包含性

OpenGL指令：

操作的矩阵是投影矩阵，但是实际上称为透视矩阵

glMatrixMode

glFrustum(left,right,bott,top,N,F);

或gluPerspective(viewAngle,aspect,N,F);

两种指令作用相同，参数不同，关系如下：

**生成立体视图**

采用两个摄像机，间距是人眼间距，绘制左右两个视图p326/365

**投影的分类**

1.一点两点以及三点透视

•主轴：世界坐标系的轴称为主轴，世界坐标系平面称为主平面

•定义：依据主轴在视平面投影所产生的有限元处的灭点数量来定义几点透视，即与**n**不垂直的主轴数量

2.平行投影的类型

•特点：所有的投影线都是同一个方向

•点P的投影点p的位置：（**d**为投影线向量，**n**为视平面法向量，B为视平面上一点）

•分类方法：

1）斜投影：投影方向**d**与**n**不平行

2）正交投影：**d**与**n**平行

将实际视景体投影到CVV，正交投影的投影矩阵（实际就是一个放缩位移矩阵）

说是投影，OpenGL的实际操作都是变换，保留第三个坐标用于深度测试

•执行代码：glOrtho() //生成上述矩阵

2.1）多视角正交投影：工程制图的一般方法，用俯视图，主视图，侧视图三种图来共同表达物体实际特征

2.2）轴测视图：为了使物体形状的3D感表达更强，在这种视图中，**n**不与任何主轴平行，使平行于三个坐标平面的物体面的特征共同显示出来，一般保持一个主轴在视图中处于垂直方向（不是与**n**垂直）；主轴与视平面法线**n**形成的角度如果为α，那么sinα称为缩短因子，即该轴方向的线段在投影图中的缩短比例

•••等测图：所有主轴被同等程度地缩短（）

•二测图：两个主轴被同等程度缩短（，或）

•三测图：三个主轴的缩短程度各不相同

# 十七、几何体的真实感渲染

**概念概述**

一般意义上，渲染模型或着色模型是计算像素颜色的方式（包括绘制轮廓和隐藏覆盖面等），光照模型一般是针对要显示的像素模拟各种光照效果所使用的近似物理模型（也可以理解为一种附加效果，否则像素颜色就是单纯的纹理颜色或背景色）。不同的光照模型可以作为渲染模型的一部分嵌入，即用来渲染由于光照而导致的像素颜色成分。阴影等特殊的渲染效果一般不属于光照模型，是渲染模型的一种特殊的独立的可附加可消除的部分，或称为一种后期效果。

•着色模型（shading model）：试图描述光源发出的光线和场景中的物体是如何相互作用的；光的散射和反射的全部物理原理非常复杂，决定了不同复杂程度和真实感的模型

•线框渲染（wireframe）：最低端的真实感渲染技术，只画出物体的边界或者是网格边界，着色模型的真实感比线框渲染的真实感高一个层次

⊙可见线框渲染技术：线框技术加隐藏面消隐技术，将被遮挡的线条抹去，提高真实感

•平着色（flat shading）：即平面明暗处理，每个面上的反射光量只根据其上某点计算一次，同一个面上的所有点灰度一致

•平滑着色（smooth shading）：同一个面上不同点的灰度不同是通过插值实现的，例如Gouraud shading

•镜面反射光（specular light）：高光（highlight）可以使物体看上去更加有光泽

•阴影，纹理，光线跟踪……

**渲染模型（Rendering Model）**

1.光源种类：点光源和环境光源（ambient light）

2.入射光和表面的相互作用

•吸收称为热能

•表面反射：

⊙漫反射（diffuse scattering）入射光浅浅地穿透物体表面然后向各个方向均匀散射， 因为散射光与表面有强烈的相互作用，所以光的颜色通常被表面材质的属性所影响

⊙镜面反射（specular reflection）有很强的方向性，入射光不穿透物体表面，而是 直接在外表面反射，产生高光；简单模型中，反射光与入射光颜色相同，类似塑料， 复杂模型中，颜色在高光的不同区域会有变化，例如金属表面光泽

•折射进入物体内部

3.计算漫反射分量

**m**反射面的法向量；**v**反射点P指向视点的向量；**s**反射点P指向光源的向量

•光强和面片相对于视点的朝向没有关系（只要视点不在面片的背光面后）

•光强正比于面片的投影面积，也就是与面片相对光源的朝向有关（Lambert法则）

•当面片背向光源，光强为0

•光强在现实中与光源到面片距离有关，但是基于简化的光源建模（点光源或无穷远光源），距离因素如果在物理定律层面被考虑（有时因为其他原因有必要加入距离因素），会产生更加不真实的效果

•是漫反射光源的亮度；为漫反射系数，真实情况下，其与入射光波长，夹角，表面物理特性（和颜色）有关，为了简化而忽略。一般为每个表面选择一个合适的值，需要和真实效果对比反复试验确定

4.镜面反射

1）Phong【Phong75】模型（适合塑料与玻璃，不适合金属）

•一般物体的反射光并非只有一个方向，而是围绕镜面反射方向分布，镜面反射方向光强最大，光强随着偏离镜面反射方向角度增大迅速减小

•镜面反射光随角度衰减的分布见p344/383，f一般取1-200之间，由实验确定

镜面反射**：**

•是镜面反射光源强度；是镜面反射系数，也通过实验确定，一般会依赖于入射光波长和入射角度

2）Blinn【Blinn77】半角向量加速

•这个改进算法不需要计算**r**，如果光源和视点都非常远，那么**s**和**v**对于物体上不同的表面都是不变的，则只需要对于每个面计算一次**hm**即可

3）Schlick【Schlick94】加速

用更容易计算的式子代替指数计算，仍采用Phong基本模型

5.环境光源与环境反射

•光源没有确定位置，光在各个方向上均匀分布

•环境光源强度用表示，环境光反射系数为，通常与漫反射系数相等

6.综合光的贡献

•漫反射光源强度和镜面反射光源强度是两个值，但通常设为相等

7.增加颜色因素

•将光分成红绿蓝三种分量，每个颜色的光分别使用一套参数（光源强度和反射系数）和上述公式

•漫反射和镜面反射的光源颜色一般也相同

•环境光和漫反射系数依赖于表面本身的颜色（即白光照射下的漫反射颜色）

如果可以知道物体的颜色构成（红绿蓝比例），则该物体的漫反射系数也应遵照该比例

•镜面反射通常不改变光的颜色（不绝对），所以一般镜面反射系数对于三色光都设为一样的值，则其作用更类似于“灰度”

8.增加材质因素

反射系数表p349/388

•材质因素决定了反射系数和指数f的值

•真实情况下，很多材质的镜面反射光并非完全是和入射光颜色一样，即三色镜面反射系数不相同（比如金和铜）

**在OpenGL中使用光源**

1.创建光源

1）OpenGL允许最多定义八个光源，依次用GL\_LIGHT0、GL\_LIGHT1

2）实现语句

先定义一些存储参数的数组，然后通过glLightfv()指令将数组（或用glLightf()传递单个值）赋给特定光源的特定参数（fv代表float vector）

①定义位置

GLfloat myLightPosition[]={3.0,6.0,5.0,1.0}; //定义坐标数组且用齐次坐标，例子是点光源，如果使用向量表示方向光或平行光（无限远光源）

glLightfv(GL\_LIGHT0,GL\_POSITION,myLightPosition);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

②定义颜色和光源强度

GLfloat amb0[]={0.2,0.4,0.6,1.0}; //分别定义环境光，漫反射和镜面反射光源强度与颜色

GLfloat diff0[]={0.8,0.9,0.5,1.0};

GLfloat spec0[]={1.0,0.8,1.0,1.0}; //颜色使用RGBA格式，即红绿蓝和透明度（alpha）

glLightfv(GL\_LIGHT0,GL\_AMBIENT,amb0);

glLightfv(GL\_LIGHT0,GL\_DIFFUSE,diff0);

glLightfv(GL\_LIGHT0,GL\_SPECULAR,spec0);

//对于所有光源，环境光在默认状态下是黑色（昏暗），对于LIGHT0，漫反射和镜面反射光源默认是白光，对于其他光源漫反射和镜面反射光源默认是黑色

③定义光照方向

glLightf(GL\_LIGHT0,GL\_SPOT\_CUTOFF,45.0); //定义光锥的半张开角

glLightf(GL\_LIGHT0,GL\_SPOT\_EXPONENT,4.0); //定义衰减因子的指数，coseβ为与光源的连线和光源集中方向的夹角为β的点的光强衰减率（探照灯效果）

GLfloat dir[]={2.0,1.0,-4.0};

glLightfv(GL\_LIGHT0,GL\_SPOT\_DIRECTION,dir);

direction vector

//这部分参数的默认值**d**=(0,0,-1),α=180°,ε=0，即默认条件下光源是一个全方向的点光源

④定义光随距离衰减

衰减因子表达式：

glLightf(GL\_LIGHT0,GL\_CONSTANT\_ATTENUATION,2.0);

glLightf(GL\_LIGHT0,GL\_LINEAR\_ATTENUATION,2.0);

glLightf(GL\_LIGHT0,GL\_QUADRATIC\_ATTENUATION,2.0);

//默认条件下，kc=1，其他的都为零，即没有衰减

⑤定义光照模型的一般规则

•全局环境光设定

GLfloat amb[]={0.2,0.3,0.1,1.0};

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT,amb);

//全局环境光独立于所有普通光源，默认值为(0.2,0.2,0.2,1.0)，即在不设定任何光源的默认条件下场景就有光

•视点是局部还是无限远

glLightModeli(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER,GL\_TRUE);

//通过此设定设置对于每个顶点是否采用局部的视点坐标（真实情况下每个顶点的**v**一般都不同），还是采用默认的对于每个顶点都一样的无穷远视点，其默认的视点方向**v**=(0,0,1)，即视点坐标系的z轴方向（相同的**v**和**s**可以提高渲染效率）

•多边形两面是否都需要着色

glLightModeli(GL\_LIGHT\_MODEL\_TWO\_SIDE,GL\_TRUE);

//设定为GL\_TRUE时，OpenGL会对所有背面的法向取反，从而会对所有的多边形面片背面着色（适用于存在对面片背面观察的情况）；默认值为GL\_FALSE，即不对背面着色（正反面依靠顶点的顺序判断）

⑥光源的移动和位置详议

光源的位置同样受到模型视点矩阵作用，但是仅仅在glLightfv(GL\_LIGHT0,GL\_POSITION, position)语句调用时完成变换，因此该语句的位置不同可以产生不同的结果

•利用压栈/退栈操作可以单独变换光源位置，退栈之后模型视点矩阵可以用于改变图形，而光源位置不再受到影响

p355/394有问题

2.材质属性

GLfloat myDiffuse[]={0.8,0.2,0.0,1.0}

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

反射系数的格式仍采用RGBA

•第一个参数可选：GL\_FRONT,GL\_BACK,GL\_FRONT\_AND\_BACK,为正面、反面或正反两面设置反射系数

•第二个参数：GL\_AMBIENT,GL\_DIFFUSE,GL\_SPECULAR,GL\_AMBIENT\_AND\_DIFFUSE, GL\_EMISSION,设置环境光、漫反射或镜面反射系数，或环境光和漫反射系数相同时的值，或设置表面的辐射颜色（emissive color），即面片自身的辉光颜色（独立于所有光源）

3.SDLp356/395

**平面着色与平滑着色**

1.概述

•有的书中，照明模型（illumination model）和着色模型（shading model）意思是类似的

•填充器（filer）：多边形填充程序，一般在一个凸多边形内部，由下至上由左至右一个像素一个像素地着上适当的颜色

2.平面着色和马赫带

为一个面片上的所有像素着相同的颜色，通常选取面片的第一个顶点的颜色；适合于面片本身是平的，并且离光源相距很远的时候，这种着色方法效率非常高

1）语句

glShadeModel(GL\_FLAT);

2）马赫带

一种视觉现象，叫做一侧抑制(lateral inhibition)

3.平滑着色

1）Gouraud明暗处理（Gouraud插值）【Gouraud71】

•对于在ys处的扫描线，首先计算扫描线最左端的点（即扫描线与面片边界的左侧交点）的颜色，利用该点所在边界的两个顶点插值计算出

•f是扫面线端点所在位置相对于边界两顶点的比例，同样对右侧的端点重复此步骤

•因此，扫描线上的其他的点可以通过两个端点的插值得到

•为了提高效率，每个像素的颜色可以采用增量计算

•语句：

glShadeModel(GL\_SMOOTH);

•缺点：不能很好地处理高光，因此使用时镜面反射分量经常被去掉

2）Phong Shading明暗处理【Phong75】

•过程和Gouraud类似，只是将所有的颜色参数改为法向量，即通过顶点的法向量插值得到扫描线端点的法向量，再插值得到每个像素的法向量，再进行计算颜色

•缺点：耗时长

•改进和加速算法

篱笆明暗处理（fence shading）【Benhrens94】

对于扫描线端点仍然采用插值得到法向量的方法，之后计算这两个端点的颜色，然后类似Gouraud一样沿着扫描线插值两个颜色

（其他比如【Bishop86，Claussen90】）

**增加隐藏面消隐**

1.深度缓存方法（depth buffer或z-缓存，z-buffer）

1）原理

•采用一个深度缓存和帧缓存对应的方式，对于帧缓存内每个像素p[i][j]，对应一个深度缓存中的量d[i][j]，d数组中每个元素的大小12-30比特

•渲染过程中，d数组储存对应的像素遇到的最近物体的伪深度，p数组则存储该最近的点的颜色

•d数组初始值为1.0，即伪深度最大值，帧缓存的初始值则为背景色

2）计算每个像素的伪深度

方法类似于Gouraud插值

3）语句和指令

glutInitDisplayMode(GLUT\_DEPTH| GLUT\_RGB); //初始化显示时指定深度缓存模式

glEnable(GL\_DEPTH\_TEST); //同时启用深度测试

渲染每一幅图像之前要初始化：

glClear(GL\_COLOR\_BUFFER\_BIT| GL\_DEPTH\_BUFFER\_BIT); //清屏

**为面片添加纹理**

1.概述

纹理空间（texture space）：以s和t为参数，两个参数取值为0-1之间

纹理函数：s和t为参数，函数值为颜色或亮度值

2.纹理来源

1）位图纹理

•矩阵一样的像素数组，记为txtr[r][c]，c列r行，存储着颜色值，称为纹理元素(texels)

•需要先将txtr[r][c]的值和0-1之间的st值联系起来

2）数学函数

即texture(s,t)类型的纹理函数

3.添加纹理的机理

Ttw：从纹理坐标到世界坐标的变换

Tws：从世界坐标到屏幕坐标的变换

像素点：(sx,sy)=Tws(Ttw(s,t))

渲染过程： //即由像素坐标可以求得对应纹理坐标，进而得到颜色

4.平面上贴纹理

1）语句与指令

glBegin(GL\_QUADS); //纹理坐标和顶点先后成对输入图形管线，绑定在一起

glTexCoord2f(0.0,0.0);

glVertex3f(1.0,2.5,1.5);

…

glEnd();

•纹理坐标可以选取处于0-1区间之外的值，但是一般（默认）OpenGL会忽略整数位，选取对应的0-1之间的值替代，也就是反复平铺纹理的效果

•面上的其他点对应的纹理坐标由顶点插值确定

5.网格模型添加纹理坐标（数据组织）

1）平面片组成的网格

•面片顶点的数量

•面片法向的索引

•顶点索引的列表

•纹理坐标索引的列表

2）逼近平滑物体的网格

•面片顶点的数量

•顶点索引的列表

因为对于平滑物体网格，每个顶点对应一个特殊的法向量和纹理坐标

6.渲染纹理

过程仍然和Gouraud明暗处理类似，沿着扫描线逐个像素渲染，但是线性插值不再适合，因为纹理和投影的尺度并非是线性对应关系（投影存在着近大远小的情况），也称透视校正插值perspective-correct interpolation。

1）推导g和f的关系（变换前后的插值比例关系）p372/411【Segal92】

已知：,,

,,,,

·其中r(f)就应该理解为其字面含义：对a,b两点进行表观比例为f的线性插值（这与不应该对a,b两点直接进行线性插值来求顶点属性不矛盾，r(f)只是一个示意性函数）。后面的推导建立在r=R的情况下。

•第二个等式说明屏幕坐标中比例为f的点对应原三维空间中的哪个点，这样仍然可以对变换后的属性进行插值，只是插值的结果不能用线性插值的原始式子直接计算，而是用这个式子。

·在透视变换的条件下(, )，有另一种方式表达R（真实的插值）、f（变换后的表象比例）、A、B（原始顶点坐标或其他顶点属性）之间的关系：

•如果变换是仿射变换，屏幕上直线ab上等距的间隔对应到三维空间中AB上依旧是等距的，即a4=b4=1

2）增量渲染

•1）中的最终式子称为双线性插值，或理性线性绘制（rarional linear rendering）【Heckbert91】，或双曲线插值（hyperbolic interpolation）【Blinn92】

•1）中式子A,B,R可以理解为A或B或R(s,t)，即纹理坐标，a,b,r即为经过变换后透视除法之前的空间坐标

•为了高效计算(s,t)，仍然可以根据1）中式子改写出增量形式（以s坐标为例）；s1s2为对应y值时的等式分子和分母

•对于扫描线y的端点和扫描线上其他点x的对应纹理，都是采用这个插值

**渲染、纹理和图形管线**

•着色shade；透视投影perspective；裁剪clip；透视除法perspective division；视口viewport；视椎view volume（类似视景体）

•裁剪过程中产生的新的顶点的颜色，通过线性插值来计算p351/390

•伪深度计算中使用线性插值，这些都不是使用双线性插值的地方

**渲染纹理的语句与指令**

p415

**纹理调和**

纹理映射得到的数值在渲染计算中如何使用？

1.创建一个发光体

1）原理

设置可见光亮度I等于该点的纹理值，分别对红绿蓝设定

2)指令

为了让OpenGL这样贴纹理

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_REPLACE);

2.调和反射系数画纹理

1）原理

物体的颜色比例和其漫反射系数比例相同，即可以用纹理的颜色调和反射系数

2）指令

glTexEnvf(GL\_TEXTURE\_ENV,GL\_TEXTURE\_ENV\_MODE,GL\_MODULATE);

例子p416

**在曲面上卷绕纹理**

1）收缩包裹（Bier & Sloane【Bier86】）：对于旋转曲面物体，将纹理卷绕贴在一个足够大的包围在物体周围的圆柱面上（对于类球体，则为足够大的球面），然后纹理面和物体表面顶点按照一定的规则进行对应（类似于将圆柱面或球面进行收缩贴在物体表面）。

①物体中心对应规则：物体中心C和纹理面上一点P的连线与物体表面交于一点V，则P和V对应

②物体法向对应规则：物体表面一点V的法线与纹理面交于P点

③纹理面法向：纹理面上一点P的法线与物体表面交于一点V

**反射映射**

1）概念

反射映射：物体周围的场景在物体上的反射，随着观察角度不同，图像会在物体上“流动”；而一般的法向纹理贴图是固定在物体上的

2）分类

铬映射：一个代表周围环境的粗糙并模糊的图像被反射在物体上

环境映射（Blinn & Newell【Blinn76】）：在物体的反射中可以看到一个可辨识的周围环境的图像

3)原理（p388/427）

•将需要反射的环境纹理包裹在一个巨大的球体上或者是立方体【Greene86】上，该球体或立方体包围被观察物体乃至整个空间；纹理面的生成可以通过从物体位置并且从不同角度观察环境来绘制

•若光泽物体位于包围球体或立方体中心，且相对非常小，则可以简化反射纹理的计算过程，即可将反射光的方向视为从物体中心发出

4）语句和指令

OpenGL中提供的情况：纹理包裹在一个非常大的立方体上的反射映射

glTexGenf(GL\_S,GL\_TEXTURE\_GEN\_MODE,GL\_CUBE\_MAP);

glTexGenf(GL\_T,GL\_TEXTURE\_GEN\_MODE,GL\_CUBE\_MAP);

glEnable(GL\_TEXTURE\_GEN\_S);

glEnable(GL\_TEXTURE\_GEN\_T);

5）使用环境映射模拟高光

原理：创建一个高亮度的斑点纹理，即将斑点“画在”物体上

**给物体添加阴影**

1）方法一：

①基本思想

先绘制阴影所在平面，再分别绘制物体不同的面片在平面上的投影，最后画物体本身（可能存在阴影被物体本身遮蔽的情况），整合之后即为阴影

②原理

点光源S，顶点V，平面

投影点

在OpenGL中，这个公式可以利用矩阵计算，即V’和V的关系可以利用矩阵表达

2）方法二：

①基本思想

对每个光源利用一个辅助的二次深度缓冲区，称为阴影缓冲区，该缓冲区记录的是以光源为视点所有“可见”的点的伪深度（按照深度缓冲区的工作原理，对于每一个缓冲区元素记录下最小的伪深度）；在渲染着色时，通过检验每个视点可见的点的阴影缓冲区深度便可得知是否是处于阴影中；阴影中的点只需要环境光设置，反之需要所有类型光源设置

②特点

可以计算非平面上的阴影；需要大量存储空间

3）其他方法：

对于非点光源，存在全影和半影，算法将更加复杂（【Watt92】）

**辐射度与光线跟踪**

是两种高级的提高图片光照真实感的算法，其中辐射度算法的等级更高【Paul Nettle】

**OpenGL2.0和着色语言（GLSL）**

更高级的OpenGL版本，具有可编程着色扩展，使用GLSL语言（配套的术语有顶点处理器和顶点着色代码）

1）凹凸贴图（Jim Blinn【Blinn88】）

通过纹理扰动表面的原始法向量，进而在渲染过程中形成复杂的贴图，例如形成橘皮状坑洼的表面或者浮雕般的视觉效果；贴图附着的几何体或网格仍然是光滑或平滑的结构

•进一步阅读

【www.paulsprojects.net】Paul Baker

【A Practical and Robust Bump-Mapping Technique for Today’s GPUs】Mark Kilgard

2）非真实感绘制

绘制出的物体带有非真实的特性，但却仍然具有该物体的本质特征；比如带有卡通效果的图像，水墨画效果的图像或是雕刻等等

# 十八、光栅显示工具

**概述**

1）生成图像的方法：

①扫描和数字化一幅已经存在的照片或者电视图像

②用程序计算像素值，例如场景绘制

2）扫描转换（scan conversion）或光栅化（rasterization）：通过帧缓冲区一块区域的顶点位置和颜色等高级信息确定区域内部像素颜色的过程；在图形管线中，视口变换之后就要进行光栅化过程

3）片段（fragment）：光栅化过程生成的不是简单的像素，而是片段；片段由颜色、深度和纹理坐标对组成，在把片段作为像素值写入帧缓冲区前要进行一系列的操作和测试。也可以理解片段为场景中物体表面图形的一小部分，如果顺利通过管道，则其将成为一个像素或提供一个像素的一部分信息，反之可能由于被遮挡而丢弃。

4）像素映射（pixmap，pixelmap）：矩形的数值数组，每个元素代表一个像素，不同类型的像素映射使用的表示像素的方法不同效果也不同；像素映射可以存储在内存中任意区域，也可以复制到帧缓冲区决定显示器上每个像素的颜色亮度；像素既指每个矩形数组的元素，也表示显示器上每个亮点

5）颜色深度（color depth）：一个像素数据的比特长度，决定了每个像素有多少种取值，亦即多少种颜色亮度

**处理像素映射**

1）对像素映射的操作种类

•绘制图像

•复制内存映射

glReadPixels()：读取帧缓冲区中的一个区域到常规内存

glCopyPixels()：在帧缓冲区中把一块区域复制到另外一个位置

glDrawPixels()：将给定的内存映射绘制到帧缓冲区中

glRasterPos2i(x,y) //设定当前光栅位置，一个位置标识，被上述函数使用

•缩放和旋转内存映射

复杂的或更一般的旋转缩放需要借助反走样技术处理结果

•比较两个内存映射

•在内存映射中表示一个区域并着色（区域填充region filling）

2）重要的像素映射类型实例

位图（bitmap）：每个像素占一位，每个像素不是开就是关

灰度级（grayscale）：每个像素占一个字节，有256种颜色

LUT索引（LUT indices）：每个像素包含一个数字，数字是颜色查找表（LUT）的索引

RGB：每个像素包含3个字节，分别表示红绿蓝三个分量，即“真彩色”

指令：struct RGB {unsigned char r,g,b;};

RGBA：每个像素包含4个字节，第四个表示不透明度

**合并像素映射**

①将两幅图逐一按像素进行合并，合并的方法有很多种，如加权平均

②一种可使用独特硬件达到极高效率的合并称为读-改-写循环，即一个像素与另一个像素合并后代替原第一个像素。

③一种特殊的读改写循环即alpha通道图像融合，也是加权平均融合的一种，alpha值即为权

④另一种合并像素的方式是将像素的值按照二进制逐位合并，即逻辑合并。有几种逻辑运算就有几种逻辑合并。XOR是较为常用的，因为两次相同对象的XOR操作会得到原对象。

BitBLT（读音bitblit）操作（位块传输bit boundary block transfer）：集成了draw，read，copy方法的函数操作

**Bresenham直线绘制算法**

优点：无需浮点运算，只有整数运算；无需舍入操作，即取整操作；算法稳定，可重复可逆

算法过程：（限于直线从左至右，斜率小于1大于0的情况）

①将初始像素位置填充

②像素坐标x以1递增

③若F<0，通过增加2H更新F，反之像素坐标y递增1，并增加-2(W-H)更新F

F(x,y) = -2W(y - ay) + 2H(x - ax) 这是直线方程等式左边乘2后的式子，用以判断像素点所在位置在真实直线之上还是之下

W = bx - ax

H = by - ay

上面3个式子的所有量均须以整数运算

1)Bresenham算法其他情况扩展

①直线从右至左——x由增递减，注意交换起始点与终点位置的算法并不提倡，因为打乱了顶点顺序，在绘制折线、虚线等其他线时难以重复使用此算法。

②斜率大于1或-1——x与y互换

③斜率为负——y由递增变为递减

**点画线和多边形点画**

定义：通过使用数字序列直接定义每个像素点的开关与否来为直线或一片区域绘制图形，对直线而言图形一般就是虚线或点线等。p427

**定义和填充像素区域**

区域定义方法有两种

像素定义法：枚举出所有像素值的表示法

符号表示法：即数学或几何学描述的抽象表示

**符号定义区域**

也有两种方法

矩形集合定义区域（也称形状表示法）：适用操作为平移和缩放，以及填充图形

路径定义区域（如数学公式，折线坐标定义法，链码法）：链码法使用起点坐标和一个表示形状轮廓的数字序列组成，数字的含义是路径的下一步移动方向  
适用于平移和缩放以及旋转，但是不适合填充

**填充多边形定义的区域**

算法p433

需要考虑水平边，相交边等特殊情况的处理

改进性能，使用制表法（保存一些信息）和经验规律加速交点计算p436

其他高效多边形填充算法【Lui00】

**走样和反走样技术**

反走样anti-aliasing技术分为3种：未看，参考《实时计算机图形学》第二版笔记

①前置滤波

②过采样

③后置滤波

**纹理反走样**

纹理四边形texture quad：不是纹理元素texel，这是像素空间的一个像素经过空间变换投影到对应纹理空间形成的四边形。该四边形的着色计算自然就决定了该对应像素的着色。

①椭圆加权平均EWA滤波【heckbert86】

近似前置滤波法，使用滤波器函数替代纹理四边形，可使用增量方法方便计算

②随机采样【Watt&Watt92】

通过在纹理坐标周围随机采样求平均来决定像素颜色，这种方法忽略纹理四边形

**OpenGL反走样**

累积缓冲区accumulation buffer：通过对视点位置添加数个小抖动jittered，多次绘制同一场景，然后将这些场景累计平均。会严重消耗性能。

**低色彩分辨率产生多种明暗效果的算法**

数字半色调digital halftone或图案法patterning【Ulichney87,Knuth87】：以像素空间分辨率换取色彩分辨率，即以相邻数个像素的不同明暗组合产生足以满足人眼效果的高色彩分辨率的效果。色彩分辨率也即色深。  
像素的一种明暗组合应该尽可能随机分布其组成像素的亮度，避免产生条纹。

误差扩散error diffusion：对每个像素正常进行舍入处理（从高颜色分辨率舍入至最近的低分辨率值），然后将舍入误差传递给周围的像素，使得局部平均颜色亮度不失真。

点扩散【Knuth87】：抖动技术和误差扩散技术的合并方法

# 十九、曲线和曲面设计

**曲线的参数连续性**

即k-阶连续的知识

**曲线的几何连续性——Gk连续**

曲线的各阶导数向量的方向连续，但是数值可以不连续，即曲线运动速率可以突变，但是曲线仍然是曲线（没有转折点）。

**多项式描述曲线**

概念：  
多项式次数是指数的最大值  
多项式阶数是系数个数，比次数大1

这里的多项式曲线指的是参数方程描述

①一次多项式曲线

必是一条直线

②二次多项式曲线

必是抛物线

二次隐式形式或一般式为圆锥曲线，即包括抛物线，椭圆和双曲线

③高次多项式曲线

【Sederberg85】对于给定的多项式参数形式曲线，总可能找到其隐式形式，只有对一次或二次隐式形式才总能找到其参数形式（参数形式不一定为多项式）。

**有理参数形式**

有理参数形式指的是曲线参数方程由多项式的比组成，其中分子分母均为二次多项式时可以组成圆锥曲线的参数式，分子分母为一次多项式时结果特征不唯一。

**交互式曲线设计**

控制顶点Control Points：设计时插入的用以决定曲线形状的自定义点

控制多边形：控制顶点的序列

曲线生成算法有两种：  
插值：最终曲线精确通过控制顶点  
逼近：最终曲线不一定经过所有控制顶点

**贝赛尔曲线曲线设计**

①德卡斯特利奥算法p464

利用一系列点按照比例t进行内插，对于所有t值0-1，所有内插点形成的轨迹即为贝赛尔曲线

三点内插形成抛物线，最常用的是4点内插，其曲线的参数形式系数为Bernstein多项式

**贝赛尔曲线的性质**

①端点插值，其他点逼近

② 仿射不变性

变换后的曲线与基于变换后的控制顶点的曲线是相同的

③凸包性

曲线不会超出所有控制顶点能够覆盖的最大多边形区域

③导数

导数曲线为建立在一组新的控制顶点上的另一条贝塞尔曲线

缺点：

①控制顶点个数不能太多，否则会难以计算

②每个控制顶点对于整个曲线都有影响，缺少局部控制能力，这个由Bernstein多项式决定，该多项式就是后面的混合函数

**寻找更好的混合函数**

支撑：函数值不为0的区间通常叫做该函数的支撑

混合函数的性质：

①易于计算，且数值稳定——多项式函数

②在区间中每个t值的和为1——必须满足生成的点集（曲线）是控制顶点的仿射变换

③具有小的支撑集以供局部控制

④插值由设计者选定的某些控制顶点

⑤足够光滑——内部至少一阶光滑，且在起点终点处导数为0，起点终点处导数可以不连续

**分段多项式**

满足上述条件的单独简单多项式不存在，应使用分段多项式分段表式混合函数子集，通过平移创建不同区间的分段函数，保证任一参数值下所有分段函数值和为1，即可构建混合函数集合。

连接点：相邻分段函数链接处的函数值

结点：连接处的t值，或参数值

样条函数定义：M次样条函数是每个结点处都为M-1阶光滑的M次分段多项式

封闭曲线设计p477

**样条曲线和基函数**

结点向量：结点组成的一个顺序序列

样条曲线：由样条函数作为混合函数形成的曲线

基函数：给定一结点向量，一组能够用以生成所有在该结点向量上的样条曲线的混合函数族

**B样条基函数**

B样条函数定义：线性，二次，三次，B样条的阶数m比次数大1。p480

多重节点：结点向量中包含若干个相同结点，在该处基函数会有特殊性质。

标准结点向量：m阶B样条曲线的标准结点向量为首末两个m重结点，其余结点距离相差1。具体条件见p486/525

贝塞尔曲线也是B样条曲线在m=L+1时的特殊情况。

B样条曲线的重要性质介绍：p488/527

多重控制顶点：类似多重节点，使用几个相同的顶点，当同一顶点数目足够多时，曲线必然插值该顶点。

**有理曲线和非均匀有理B样条曲线**

简称为NURBS，是一般B样条曲线的推广，即为每个顶点增加一个权重然后归一化，p490/529

优点：  
可以精确获得圆锥曲线，一般B样条不行  
可以保持投影变换下不变性，一般B样条不行

投影不变性描述p491/530

通过NURBS得到  
圆锥曲线p492/531  
圆p493/532

**曲线插值**

详见Farin【Farin88】

分段三次多项式插值：

①Hermite插值

定义：规定所有顶点处的一阶导数，并保证其一阶光滑（一阶导数连续）

②自然三次样条

定义：端点处二阶导数为零，内部顶点二阶导数连续的曲线

通过规定二阶导数连续可以求得一阶导数的值，进而通过Hermite插值计算曲线参数

自然三次样条没有局部控制性，因为其定义条件的限制。

③Catmull-Rom方法p500/539

定义：内部顶点一阶光滑，不要求二阶光滑。内部顶点的斜率和其两侧顶点形成的向量平行。

张力（内部节点处速率）、偏重、连续性（允许曲线不再一阶光滑）

**自由曲面造型**

直纹面p504/543

旋转曲面p505/544

曲面片p506/545

曲面片的拼接p507/546

# 二十、颜色理论

**人类的颜色感知**

视网膜包含两类神经感知细胞：  
视锥细胞：对颜色和亮度敏感的细胞，每个细胞感应某种特定颜色，红绿蓝。主要集中在中央凹的小块区域内，可用于注视和辨别细节  
视杆细胞：不能辨别颜色和细节，对低强度光线敏感，可感知暗淡光线下的物体轮廓，可用于暗光下侧视物体轮廓

人眼能够区分128种不同色调的颜色

**颜色匹配**

利用三原色组合形成纯光谱色的波长-强度曲线p528

使用任何可见色作为原色，都会导致描述有些可见色需要负系数，这是因为纯光谱色的光谱本质上就与组合色不一样，人眼的识别能力简化了这个问题但并没有消除。

**颜色描述**

以下介绍几种颜色描述方式：

①主导波长、饱和度、光通量

光通量：光的总能量，光谱曲线的总面积

饱和度：主导成分占总光通量的比重

**CIE标准**

CIE：国际照明委员会，CIE标准也称XYZ模型，使用的也是三原色模型，但是选取的三原色不对应任何真实存在的可见色

XYZ模型的三原色是rgb模型颜色匹配函数的仿射变换，以保证xyz三色系数全空间非负。为了方便，使y和光效函数有同样形状，使得y原色的量和颜色总亮度相等

光效函数：人眼对单位辐射亮度不同波长的单色光的反应函数

CIE色度图：用来查找给定颜色的XYZ组合分量的图，轮廓表示纯光谱色，内部表示其他不饱和色p531

应用：  
①计算互补色——相加为白色的两个颜色，也称对比色。  
②计算给定颜色的主导波长和饱和度——主导色和白色线性组合，部分颜色无主导色，用互补色的主导色加后缀c表示

**色域**

定义：即一个显示器或打印机等输出颜色的设备所能产生的颜色范围，色域通常由色度图定义。输出设备一般都建立在三原色组合的原理上，因为实际中三原色必定为真实存在的三原色，所以三原色组合原理创造的输出设备无法表示全部的可见色。

选择红绿蓝作为三原色一方面和人眼有关，另一方面也是色域较大的必然选择。两个原因之间也许并不独立。

可见色（理论上由XYZ模型完全表示）> 显示器色域（RGB模型表示）

RGB颜色空间和XYZ空间转换p535

CIE坐标下的标准三原色，由NTSC美国通信标准委员会定义【Rogers85】  
红色=（0.670,0.330）绿色=（0.210,0.710）蓝色=（0.140,0.080）

**加/减色系统**

定义：加色系统即为三原色通过合并产生结果颜色，减色系统即为三原色从白色中减去得到的结果颜色。对应的加减色系统所使用的三原色对应分量之间为互补色。

加色系统是光源型输出机制的原理，即电视机，显示器等输出设备，靠直接发出可见光组合

减色系统是反光型输出机制的原理，即打印机，绘画等，靠散射白光并吸收部分成分

常见的加色系统使用红绿蓝三原色，对应的减色系统三原色是洋红magenta，青色cyan，和黄色yellow，也就是常说的红黄蓝三原色。

**HLS颜色模型**

色调H，亮度L，饱和度S

HLS模型空间与RGB模型空间的相互转化p537  
注意这个转化的精确过程很复杂，实际情况下一般只是对特殊点与最大值进行转化，其他的一般情况线性近似处理。

**索引颜色和颜色查找表LUT**

含义：像素值不直接表示颜色，而是一个表格的索引，表格即为LUT，其中的数值直接决定颜色。LUT的表宽（二进制位数）决定了总可选颜色数目，但是像素的色深仍然决定了一次扫描或一幅图像的不同色彩数目。

**颜色量化**

定义：一个将高色深颜色组合转化为低色深颜色组合的过程。转换过程一般包含用最接近的代替者代替原颜色的步骤。

·预处理步骤：  
扫描图像中像素三元数的取值范围，即图像颜色所占用的rgb空间六面体的尺寸。

①均匀量化p541

将rgb六面体沿坐标轴等距划分，每个子块使用接近中心的颜色作为代表色

这个算法显然不精确

②流行度算法p543

首先遍历像素图像，记录每种颜色出现的次数，然后排序并选出最流行的若干种颜色，并以此为标准代替原颜色。

这个算法的时间和空间代价很大

③中值切分算法p583

将rgb六面体沿最长维进行中值切分（切分后两子块颜色数目相同，并不一定是等距或中点），重复此过程直至分割成所需数目，再以每个子块中心值为代表色，其余类似。

④八叉树量化p545

具体算法过程见书，将原N种颜色转换为八叉树结构存储，但只保留最新的K种颜色，每次颜色更新都需要进行合并和替代等操作。

八叉树的8由三原色数目决定，而其深度由每个颜色分量的位数决定。具体见书

质量上和中值切分法类似，但是更加高效。

# 二十一、光线跟踪基础

光线跟踪是一种高级的渲染方式，较早的OpenGL（本书之前）版本不支持的光照模型可以借助光线跟踪来实现。

**光线跟踪的几何描述**

光线在视域空间的表达式，点（视点）向（像素和视点）式p553/592

光线跟踪的核心就是计算光线和物体交点坐标，亦即计算出光线到达交点时的时间t，代入上述参数式即可得到交点坐标。

交点求取公式：直线参数式和曲面隐式式代入，这也是算法的核心p556/595

光线与通用平面求交p556/595

光线与通用球面求交p557/596

仿射变换的性质可以推出，逆变换光线与通用物体的相交时刻t和实际光线与实际物体的相交相同。只需将光线按照物体的变换进行逆变换即可。

**光线跟踪器构架**

简单构架伪代码p554/593

核心函数实现代码构架p560/599

求交方法具体代码构架：  
球体p565/604  
正方形p569/608  
锥形柱体（包括圆柱和圆锥体）p569/608  
立方体（通用立方体作为其他更加复杂的形状的外切包围盒，可以加速计算光线和物体相交的判定）p571/610  
任何凸多面体p573/612 立方体、凸多面体使用的是Cyrus Beck算法

!!进行渲染时可以首先编写粗糙的预览结果用于测试，粗糙意味着可以对一块区域的像素着色而非每个着色。

使用光线跟踪模型实现OpenGL明暗着色模型p577/616  
这种明暗着色模型的特点是根据物体表面材质为其着色，而材质又3个反射系数粗略概括

**基于物理的明暗着色模型——Cook Torrance模型**

Cook Torrance模型有关镜面反射因素的三个基本方面

①微平面法向量的分布

微平面的位向分布遵循Beckmann分布函数

②阴影与遮挡p582/621

考虑到出射光和入射光被其他微平面部分遮挡的因素

③菲涅尔Fresnel系数

非理想镜面会反射和吸收部分光线，这个特征由该系数描述，该系数和入射角与材料折射率有关。

·Phong模型和Cook Torrance模型的比较，注意掠射角大时，反射光的强度最大值不一定在镜面反射角处。

完整的Cook Torrance光照模型表达式p584/623

颜色偏移：入射光波长的不同会导致折射率不同，进而导致菲涅尔系数不同。如果是非纯色谱光，那么不同的光分量会遭遇不同程度的“衰减”，进而影响出射光组合结果的颜色。

·为了达到更逼真的效果，可以采用更多波长的光来渲染场景（根据颜色理论，红绿蓝三色光只是一个“近似”），这就涉及到显示设备是否能够支持。

**添加表面纹理**

曲线面知识