回顾

- **1-1 课程目的和任务**
- **1-2 计算机技术在土木建筑工程中的应用**
  - CAD技术；科学计算可视化；有限元方法；虚拟现实技术；GIS/GPS/RS技术；多媒体技术和网络技术；
- **1-3 CAE概念和研究范围**
  - CAE是面向产品生成或工程建设生命周期（Life Cycle）的计算机系统，提供计算机辅助生成产品或工程建设过程的标准和概念以及相应的技术基础，提供一个支持设计、生产、管理全过程的集成环境。
  - 数据获取；概念设计；模型建立；分析计算；优化设计；性能评价；文档管理；与生产、生产管理、市场运行等其他过程的集成
  - 工程信息的获取、表示、管理和应用
- **1-4 CAE软件**

土木建筑工程CAE

第二章 计算机图形学

清华大学土木工程系

胡振中

huzhenzhong@tsinghua.edu.cn

Tsinghua University

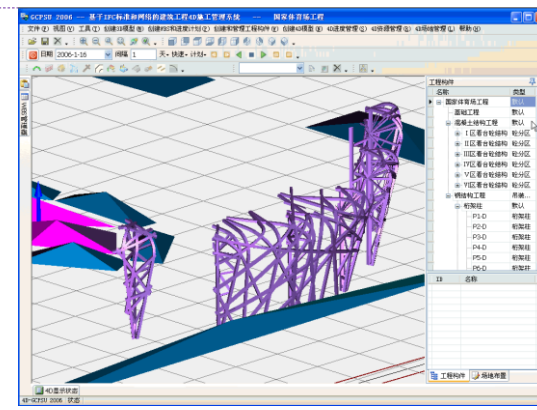
第二章 计算机图形学

第二章 计算机图形学

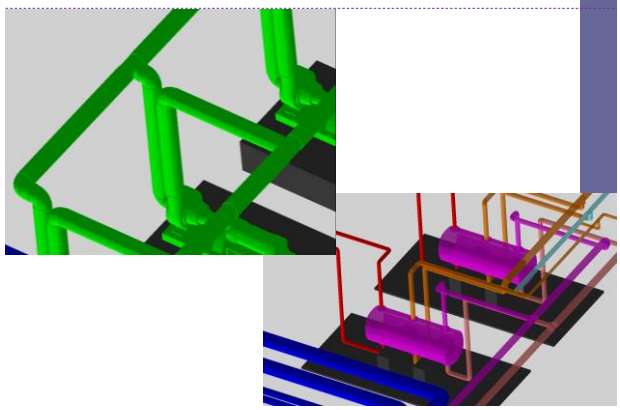
2-1 土木建筑工程建模技术发展

	二维制图	三维几何建模	信息建模
面向流程	电子制图	工程设计	工程项目全生命期
表示形式	二维图形	三维实体模型	集成三维实体模型和非几何信息的信息模型
成果	相互独立的图纸	从单一模型中生成的协调一致的施工图集	项目全生命期所需的各种图档和报表

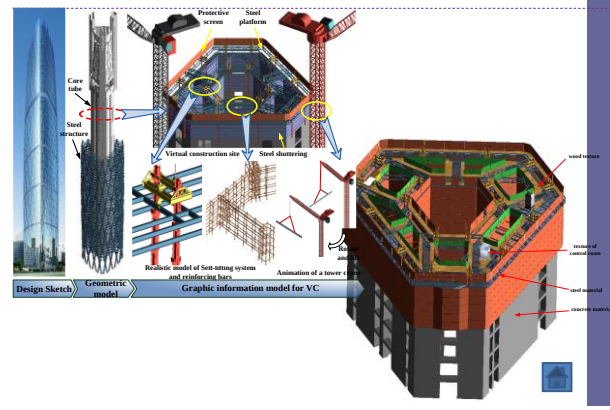
2-1 土木建筑工程建模技术发展



2-1 土木建筑工程建模技术发展



2-1 土木建筑工程建模技术发展



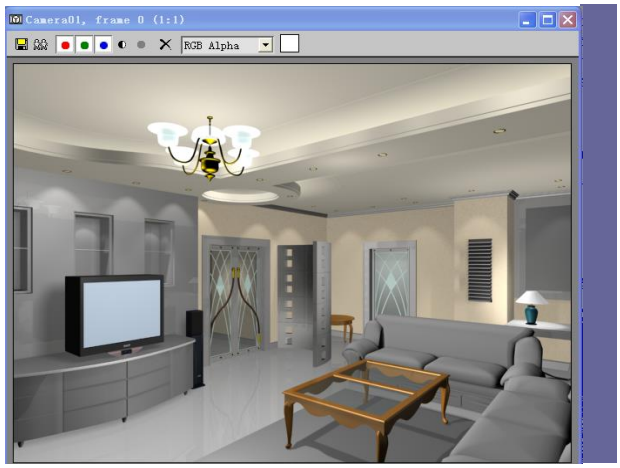
2-2 三维几何模型

- **三维几何造型**

  是通过对点、线、面、体等几何元素，经过平移、旋转、变化等几何变换和并、交、差等集合运算，产生实际的或想象的物体模型的过程。

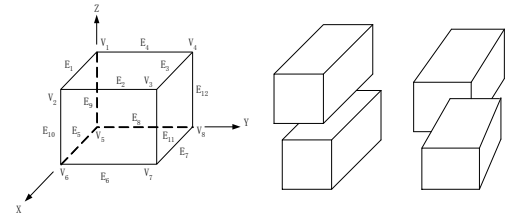
- **三维几何模型的真实感表现**

  对计算机中建立的三维几何模型进一步的隐藏线隐藏面消除，并模拟现实世界中的光照效果求出三维几何模型的明暗和纹理效应，从而使计算机中的三维几何模型具有真实感。



三维几何造型

- **三维物体的几何模型**
  - **线框模型（Wire-frame）**
    - 用物体的轮廓边来描述物体的几何形状（将物体看成三维空间中线段的集合）。
    - 可从任意方向获得三视图、透视图、不适于处理剖面图；
    - 表示三维物体时会产生多义性。



三维几何造型

– **线框模型（Wire-frame）**

- **数据结构**
  - ✓ **顶点表：**反映各个顶点的坐标
  - ✓ **边　表：**反映各条边以及每条边的起点和终点
  - ✓ **边类型：**反映边的类型信息，如直线、圆弧等

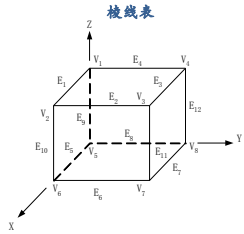
顶点表

顶点	坐标值		
	x	y	z
1	0	0	1
2	1	0	1
3	1	1	1
4	0	1	1
5	0	0	0
6	1	0	0
7	1	1	0
8	0	1	0

边表

棱线	顶点号	
1	1	2
2	2	3
3	3	4
4	4	1
5	5	6
6	6	7
7	7	8
8	8	5
9	1	5
10	2	6
11	3	7
12	4	8

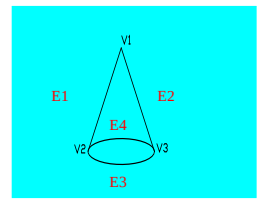
棱线表



三维几何造型

思考：

顶点表				边表			边类型	
顶点号	X	Y	Z	边号	起点号	终点号	边号	类型
V1	x_1	y_1	z_1	E_1	V_1	V_2	E_1	直线
V2	x_2	y_2	z_2	E_2	V_1	V_3	E_2	直线
V3	x_3	y_3	z_3	E_3	V_2	V_3	E_3	弧
				E_4	V_3	V_2	E_4	弧



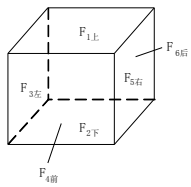
- **线框模型的特点**
  - ✓ 若顶点数和边数增加，会受到计算机内存和运算速度的限制；
  - ✓ 用大量棱线近似曲面显得不自然；
  - ✓ 难以计算形体的几何特征；
  - ✓ 可产生不确定理解----多义性；
  - ✓ 结构简单，处理容易；

三维几何造型

– **表面模型（Surface Model）**

- 用棱线所围成的封闭部分定义实体的表面，再由面包络构成实体（多边形平面的集合来逼近空间物体的轮廓面）。
- 表面模型的面元素可以是平面、圆柱面、球面等。
- 缺点：缺乏描述表示形体内部结构的能力，不能求物体的重量、惯性矩等。

顶点表和棱边表同线框模型

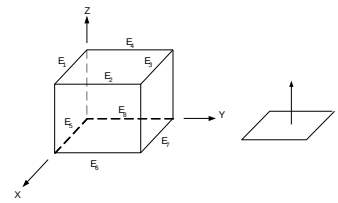


表面	棱线号			
1	1	2	3	4
2	5	6	7	8
3	1	10	5	9
4	2	11	6	10
5	3	12	7	11
6	4	9	8	12

三维几何造型

– **实体模型（Solid Model）**

- 在表面模型的基础上，对实体部分予以明确定义。
- 实体模型具有描述几何形状的完备信息，可以对物体形状作操作处理（取某个剖面、简单体素的拓扑操作。



该棱线所围的外法向矢量与z正向一致

表面	棱线号			
1	1	2	3	4
2	-5	-6	-7	-8
3	-1	-10	-5	-9
4	2	10	6	11
5	3	12	7	11
6	-4	-9	-8	-12

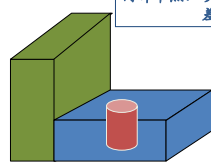
除顶点表、棱线表同线框模型外，面表中棱线号是有向的。

三维几何造型

- **计算机内部的表示模式**
  - 体素调用
  - 空间点列
  - 单元分解
  - **扫描变换**
  - **边界表示**
  - **结构实体几何(CSG)**
- **复合表示模式**
  - CSG与边界表示
  - CSG与扫描变换
  - CSG、边界表示与扫描变换

- **结构实体几何（CSG）**
  - 数据结构: 有序二叉树
    - 内部节点：并、交、差等集合运算
    - 叶结点：体素

根节点：L型立体
叶结点：三个体素实体
内部节点：实体的并、差集合运算



2-3 三维图形显示的关键技术

- 数学工具
- 物体变换与三维观察
- 真实感渲染
- 曲线和曲面

数学工具

- **向量（Vector）**
  - 向量是具有长度和方向的实体，但是没有位置
    - 力、位移、速度
    - 向量运算法则（加、减、点积、叉积）
    - 向量的长度和单位向量
    - 向量的夹角
    - 正交向量
    - 法向量

$$|\vec{a}| = \sqrt{\vec{a}\cdot\vec{a}} = \sqrt{a_1^2 + a_2^2 + a_3^2}$$
$$\vec{a} + \vec{b} = (a_1 + b_1, a_2 + b_2, a_3 + b_3)$$
$$\vec{a} - \vec{b} = (a_1 - b_1, a_2 - b_2, a_3 - b_3)$$

- **坐标系**
  - 从原点出发的三个轴，是位于“世界”中的某一位置
    - 通常采用两两互相正交的直角坐标系
    - 点和向量的齐次表示（为了用代数的方法研究映射几何）

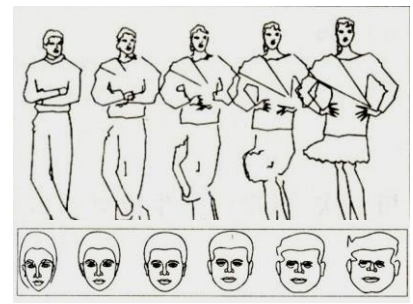
$$v = (v_1, v_2, v_3, 0) \quad P = (p_1, p_2, p_3, 1)$$

数学工具

- **两个点的线性插值**

$$P = A(1-t) + Bt$$

  - 关键帧动画



数学工具

- **求两个平面的交线**

设两平面上的已知顶点为P_s、P_t，法向量为U、V。设P为交线上的一点，则不妨设$P_z = 0$，可求得：

$$\begin{cases} P_x = (P_s \cdot UV_y - P_t \cdot VU_y)/(U_xV_y - V_xU_y) \\ P_y = (P_s \cdot UV_x - P_t \cdot VU_x)/(U_yV_x - V_yU_x) \\ P_z = 0 \end{cases}$$

否则，可假设$P_x = 0$或$P_y = 0$，同理求得P点坐标。

求得P点坐标后，令交线方向为N，则：

$$N = U \times V$$

- **求线面的交点**

已知面的法向量N及其上任意一点P，直线上两顶点P_1，P_2，直线与平面的交点P_t。则：

$$P_t = P_1 + t(P_2 - P_1) \Rightarrow t = -(P_1 - P)\cdot N/(P_2 - P_1)\cdot N$$

其中，若分母为零，则直线与平面平行。若分子为零，则P1为交点或这个直线位于平面上。

数学工具

- **三维空间中点到平面的垂足**

已知平面法向量N及平面上任意一点P_{on}，已知一顶点P到平面的垂线交点P_t。则：

$$P_t = P + t\cdot N \Rightarrow t = -(P - P_{on})\cdot N / N \cdot N$$

若分子为零，说明顶点与平面上顶点构成的向量与平面平行。

- **两条直线之间的距离**
- **凸多边形的面积**
- **直线与平面的交点**
- **点到平面的距离**
- **直线到平面的距离**
- **平面上点与多边形包含关系**
- **三维顶点集合凸包**
- **……**

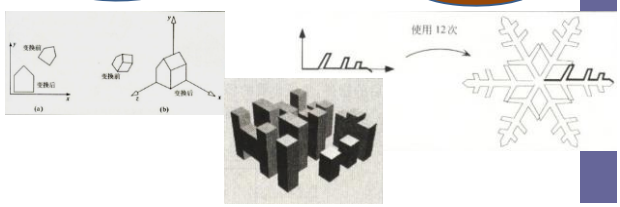
物体变换与三维观察

- **基本变换**
  - 变换矩阵T

$$P'(x', y', z', 1) = P(x, y, z, 1)\cdot T$$

比例、旋转、剪切　平移　投影　整体比例

$$T = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$



物体变换与三维观察

- **基本变换**
  - 平移变换

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

  - 比例变换

$$T = \begin{bmatrix} x_x & 0 & 0 & 0 \\ 0 & x_y & 0 & 0 \\ 0 & 0 & x_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - 绕x轴旋转变换

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - 绕y轴旋转变换

$$T = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - 绕z轴旋转变换

$$T = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

物体变换与三维观察

- **组合变换**

$$P'(x',y',z',1) = P(x,y,z,1)\cdot T_1\cdot T_2\cdots\cdots T_n$$

– **三维点绕P_1P_2轴旋转**

假设三维点P坐标为（x,y,z），绕P_1P_2旋转角度α后的坐标为P'（x',y',z'）；P_1、P_2坐标为（x_1,y_1,z_1）、（x_2,y_2,z_2），令旋转矩阵T，则有：

$$T = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} R(\alpha) \begin{bmatrix} 1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & y_1 \\ 0 & 0 & 1 & z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$c = \cos(\alpha)$
$s = \sin(\alpha)$
$\vec{u} = \overrightarrow{P_2} - \overrightarrow{P_1}$

$$R(\alpha) = \begin{bmatrix} c+(1-c)\vec{u}_x^2 & (1-c)\vec{u}_y\vec{u}_x - s\vec{u}_z & (1-c)\vec{u}_z\vec{u}_x + s\vec{u}_y & 0 \\ (1-c)\vec{u}_x\vec{u}_y + s\vec{u}_z & c+(1-c)\vec{u}_y^2 & (1-c)\vec{u}_z\vec{u}_y - s\vec{u}_x & 0 \\ (1-c)\vec{u}_x\vec{u}_z - s\vec{u}_y & (1-c)\vec{u}_y\vec{u}_z + s\vec{u}_x & c+(1-c)\vec{u}_z^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

物体变换与三维观察

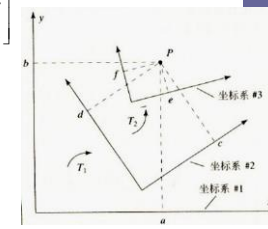
- **坐标系变换**

– **变换公式** $\begin{bmatrix} a \\ b \\ 1 \end{bmatrix} = M \begin{bmatrix} c \\ d \\ 1 \end{bmatrix}$

– **连续变换** $\begin{bmatrix} a \\ b \\ 1 \end{bmatrix} = M_1 \begin{bmatrix} c \\ d \\ 1 \end{bmatrix} = M_1 M_2 \begin{bmatrix} e \\ f \\ 1 \end{bmatrix}$

– **变换点** $M = M_2 \times M_1$

– **变换坐标系** $M = M_1 \times M_2$



物体变换与三维观察

- **摄像机与视景体**

产生目标场景视图的变换过程，类似于用照相机进行拍照，其步骤大致如下：

– **把照相机固定在三脚架上，并让他对准场景（视图变换）**

– **对场景中的物体进行安排（模型变换）**

– **选择照相机镜头，并调整放大倍数（投影变换）**

– **确定最终照片的大小（视口变换）**

物体变换与三维观察

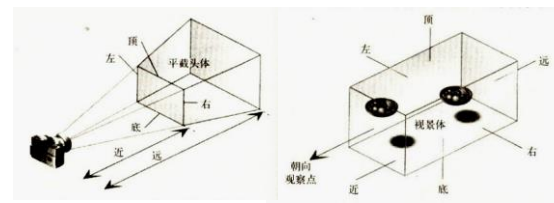
- **摄像机与视景体**

– **透视投影**

模仿真实世界的视觉效果，最显著的特征是物体距离照相机越远，在最终图像中看上去就越小。

– **正投影**

模仿虚拟世界中的真实尺寸，常用于建筑蓝图和计算机辅助设计。



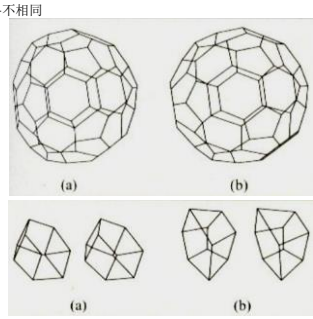
物体变换与三维观察

- **裸眼立体视图**

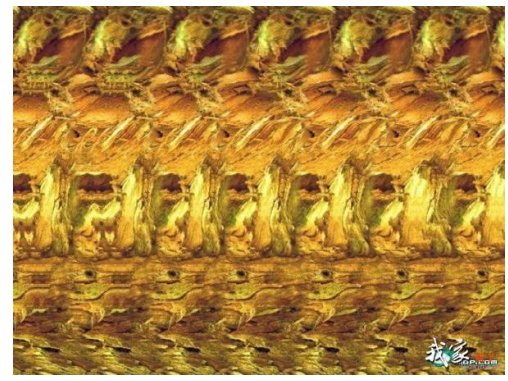
要想制作立体视图，一左一右的两幅图像要分别用略有差异的照相机生成：

– 所有照相机都LookAt点是同一个点

– 视点位置各不相同



物体变换与三维观察

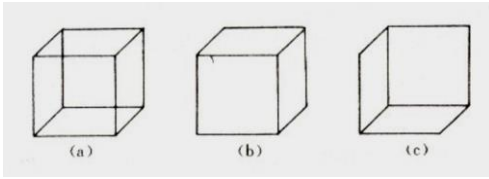


真实感渲染

- **消隐**

为了消除二义性，使图形准确地、形象地表现出来，就需要在绘制时消除实际不可见的线和面，习惯上称作消除隐藏线和隐藏面，简称消隐。

算法中，需要反复地进行直线、射线、线段以及平面之间的求交运算。



真实感渲染

- **颜色**

物体的颜色不仅取决于物体本身，它与光源、周围环境的颜色、以及观察者的视觉系统有关系。

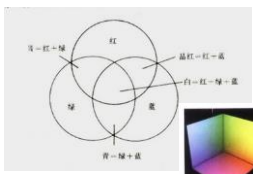
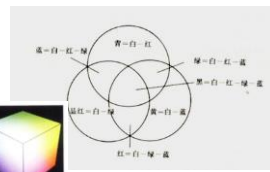
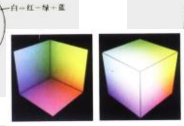
从视觉的角度，颜色包含三个要素。

– **色彩（Hue）：**即通常所说的红、绿、蓝等。
– **饱和度（Saturation）：**指颜色的纯度，如果添加白色，相当于减少了饱和度。
– **亮度（Lightness）：**即光的强度。

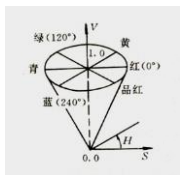
– **颜色模型**

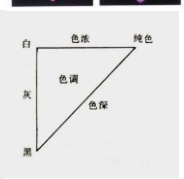
- **RGB：**红、绿、蓝三维直角坐标颜色系统中的一个单位正方体。
  应用广泛，如彩色阴极射线管、彩色光栅图形显示器……
- **CMY：**红、绿、蓝的补色青（Cyan）、品红（Magenta）、黄（Yellow）为原色。
  用于从白光中滤去某种颜色，也称为减性原色系统，原色为黑。
- **HSV：**面向用户的颜色模型，对应于圆柱坐标系中的一个圆锥形子集。
- **HLS：**色彩、亮度、饱和度。圆柱形坐标系的双圆锥子集。

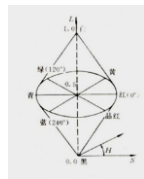
真实感渲染

RGB　　CMY







HSV　　HLS

真实感渲染

- **颜色**

– **颜色模型的转换**

HSV -> RGB
RGB -> HSV
HLS -> RGB
RGB -> HLS

```
HSV_TO_RGB(h,s,v,r,g,b)
{
  if (s == 0) {
     r = g = b = v;
  } else {
     if (h == 360) {
          h = 0;
     }

     h = h / 60;
     i = (int)h;
     f = h - i;
     p = v * (1 - s);
     g = v * (1 - s * f);
     t = v * (1 - s * (1 - f));

     switch (i){
        case 0: (r,g,b) = (v,t,p);break;
        case 1: (r,g,b) = (g,v,p);break;
        case 2: (r,g,b) = (p,v,t);break;
        case 3: (r,g,b) = (p,g,v);break;
        case 4: (r,g,b) = (t,p,v);break;
        case 5: (r,g,b) = (v,p,q);break;
        default: break;
     }
  }
}
```

真实感渲染

- **光照与阴影**

– **光色效应的模拟**
  - 能反映物体表面颜色和亮度的细微变化；
  - 能表现物体表面的质感；
  - 能表现光照下的物体阴影，充分体现场景的深度感和层次感，以及物体间的遮挡关系；
  - 能模拟透明物体的透明效果和镜面物体的镜象效果。

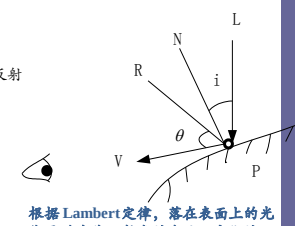
– **光照模型**
  - 模拟光能在物体间传递，并到达观察者眼中引起视觉的复杂过程。
  - 一般用一个光照度公式来计算和显示可见面上某一点的明暗程度。

真实感渲染

- **光照与阴影**

– **表现三维物体的真实感**
  - 冯氏光照模型PhongShading
    ✓ 环境反射、漫反射、镜面反射

– **Phong氏光照模型**
  - L 光线向量
  - V 视线向量
  - R 镜面反射向量
  - N 法线（过P点）
  - i 光线入射角
  - θ 镜面反射方向与视线方向夹角



根据Lambert定律，落在表面上的光能是随光线入射角的余弦而变化的。

对于非理想的反射面，镜面反射光的强度会随视线与反射线之间的夹角θ的增加而急剧地减少。

真实感渲染

- **光照与阴影**

$$I = k_a I_{pa} + \sum [k_d I_{pd} \cos i + k_s I_{ps} \cos^n \theta]$$

亮度　环境反射(与光源无关)　对所有点光源和平行光源求和　漫反射　镜面反射

式中：
- n　镜面反射光的会聚指数
- I_{pa}　环境反射光亮度
- I_{pd}　漫反射光亮度
- I_{ps}　镜面反射光亮度
- k_a　环境反射比例系数
- k_d　漫反射比例系数
- k_s　镜面反射比例系数

真实感渲染

- **光照与阴影**
  - **Phong氏模型的扫描线算法示意**

```
begin
  for屏幕上的每条扫描线y do
    begin
      将color数组初始化成y扫描线的背景值
      for y扫描线上的每一可见区间段s中的
      每点（x,y）do
        begin
          设（x,y）对应的空间可见点为P;
          求出P点的单位法向量N_o;
          求出P点的单位入射光向量L_o;
          求出P点的单位视线向量V_o;
```

真实感渲染

- **光照与阴影**
  - **求出L_o在P点的单位镜面反射向量R_o**

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} = k_a \begin{bmatrix} r_{pa} \\ g_{pa} \\ b_{pa} \end{bmatrix} + \sum [k_d \begin{bmatrix} r_{pd} \\ g_{pd} \\ b_{pd} \end{bmatrix} (L_o \cdot N_o) + k_s \begin{bmatrix} r_{ps} \\ g_{ps} \\ b_{ps} \end{bmatrix} (R_o \cdot V_o)^n]$$

$$color(x) := (r, g, b)$$

```
      end
      显示color
  end
end
```

真实感渲染

- **光照与阴影**
  - **场景渲染**
    - Phong模型可以产生物体的真实感图形，但是基于插值的，所以不能用来表示物体表面的细节，不易模拟光的折射、反射和阴影等（特别是阴影）。
    - 为了生成更真实的图象，正确显示反射、折射、阴影和表面纹理等，必须用更精确的图形算法。
    - 光线跟踪法（Ray-tracing）即为此类算法。

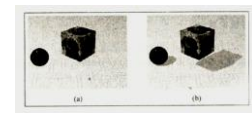
真实感渲染

- **光照与阴影**
  - **光线跟踪法思路**
    - 从视点出发，引一条视线向物体空间延伸，通过计算求得与它相交的物体。
    - 视线可能与多个物体相交，存在多个交点。
    - 比较各点与视点的距离，求得离视点最近的交点—可见点。
    - 计算该点的法向量。
    - 判断该点是否处在阴影中：从该点向光源引射线，看射线是否与某个不透明物体相交。
    - 查找表面系数表：表面的颜色属性、反射率、透明性、粗糙度等（由表面材质性质而来）。

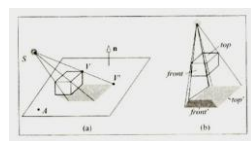
如果物体表面反射性强（如玻璃、磨光金属等），则其他物体可通过可见点，反射或折射到视点。所以在求得可见点后，必须沿反射线方向或折射线方向继续跟踪，看在该方向上是否有物体存在——间接视线——求间接可见点——递归过程。

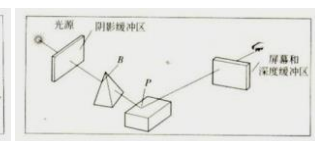
真实感渲染

- **光照与阴影**
  - **阴影的生产方法**
    - **构造投影面**
    - **利用阴影缓冲区**
    - **光线跟踪法**



造成歧义

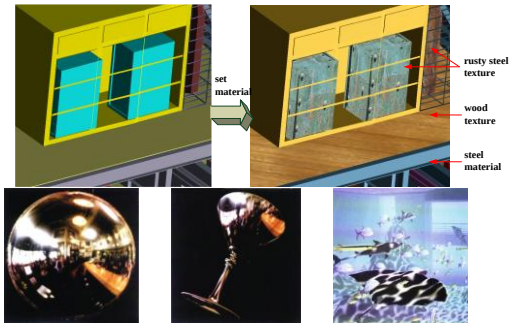




真实感渲染

- **纹理映射（Texture Mapping）**

不显著增加计算量的前提下较大幅度地提高图形的真实感。



真实感渲染

- **纹理映射（Texture Mapping）**
  - **现实世界中的物体表面有各种纹理：**
    - 颜色纹理——颜色色彩或明暗度的变化
    - 几何纹理——不规则的细小凹凸
  - **建立颜色纹理的过程：**
    - 在纹理平面区域（纹理空间）预定义一个纹理图象（图案制作，或通过扫描获得）；
    - **建立物体表面的点与纹理空间的点之间的对应（映射）；**
    - 当物体表面的可见点确定后，以纹理空间的对应点的值乘以亮度值。

真实感渲染

- **纹理映射（Texture Mapping）**
  - **映射计算：**

三维坐标(x,y,z) → 二维坐标(u,v)

球面 $x^2+y^2+z^2=1$

$$\begin{aligned} x &= \cos(2\pi u)\cos(2\pi v) \\ y &= \sin(2\pi u)\cos(2\pi v) \\ z &= \sin(2\pi v) \end{aligned} \quad (u,v)=\begin{cases} (0,0) & ,(x,y)=(0,0) \\ \left(\frac{1-\sqrt{1-(x^2+y^2)}}{x^2+y^2}x,\frac{1-\sqrt{1-(x^2+y^2)}}{x^2+y^2}y\right) & ,\text{其他情况} \end{cases}$$

三角形和平行四边形

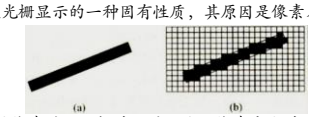
$$(x,y,z)=(u,v,1)\begin{bmatrix} A & D & G \\ B & E & H \\ C & F & 1 \end{bmatrix}$$

指定三个顶点u,v值，即可算出系数矩阵中各个系数的值

真实感渲染

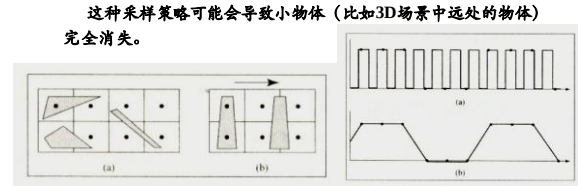
- **走样与反走样**
  - **走样**

走样是光栅显示的一种固有性质，其原因是像素本质上是离散的。



覆盖大量像素的矩形相对平滑　使用像素中点对矩形进行采样

这种采样策略可能会导致小物体（比如3D场景中远处的物体）完全消失。



真实感渲染

- **走样与反走样**
  - **反走样技术**

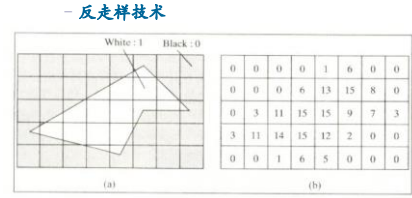
采用分辨率更高的显示设备，配更好的绘制算法—— 问题还是存在;

对于在白色背景中的黑色矩形，在边界处掺入一些灰色像素来“模糊”

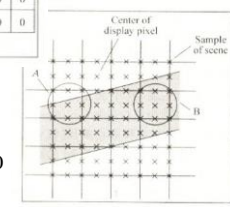
- **前置滤波(prefiltering)**
  根据物体的覆盖率计算像素的颜色，对于非多边形的形状，计算代价非常大。
- **过采样(supersampling)**
  对场景进行超过显示像素数目的亮度采样，然后取其平均值来形成要显示的像素值。
- **后置滤波(postfiltering)**
  每个显示像素的数值是场景中一个适当规模的相应样本集合的加权平均。

真实感渲染

- **走样与反走样**
  - **反走样技术**



前置滤波(prefiltering)



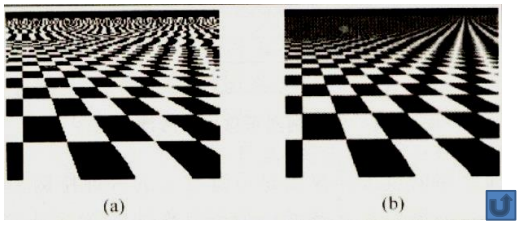
过采样(supersampling)

真实感渲染

- **走样与反走样**
  - **反走样技术**

纹理在小像素区域内的计算变形；

为了减小走样效果，每个屏幕像素着色时，应使用位于对应的纹理四边形内部的若干颜色均值作为该像素的颜色值。



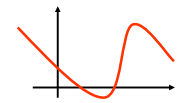
曲线和曲面

- **曲线**
  - **三种坐标表示方式**
    - 直角坐标

1) 显式： $y = f(x)$ 如 $y = \sin(x)$

2) 隐式： $f(x, y) = 0$

$$x^{2/3} + y^{2/3} = R^{2/3}$$



$$\begin{cases} x = R\cos^3\theta \\ y = R\sin^3\theta \end{cases}$$

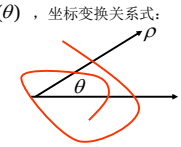
参数坐标表达式

曲线和曲面

- **曲线**
  - **三种坐标表示方式**
    - **极坐标**

对于任一坐标曲线 $\rho = \rho(\theta)$ ，坐标变换关系式：

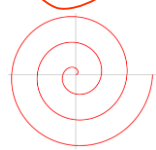
$$\begin{cases} x = \rho\cos(\theta) \\ y = \rho\sin(\theta) \end{cases}$$



例：阿基米德螺线

$$\rho = \alpha\theta$$

$$\begin{cases} x = \alpha\theta\cos(\theta) \\ y = \alpha\theta\sin(\theta) \end{cases}$$



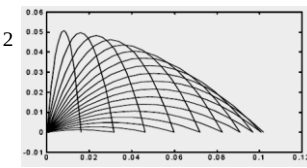
曲线和曲面

- **曲线**
  - **三种坐标表示方式**
    - **参数坐标**

$$\begin{cases} x = x(t) \\ y = y(t) \end{cases}$$

例：弹道曲线：

$$\begin{cases} x = V_0 t\cos\alpha \\ y = V_0 t\sin\alpha - gt^2/2 \end{cases}$$

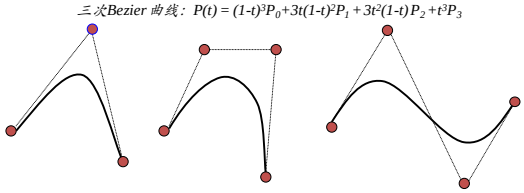


曲线和曲面

- **曲线**
  - **参数样条曲线**
    - **二次参数样条曲线** $P(t) = A_0 + A_1 t + A_2 t^2$
    - **三次参数样条曲线** $P(t) = A_0 + A_1 t + A_2 t^2 + A_3 t^3$
    - **贝塞尔(Bezier)曲线**

二次Bezier 曲线： $P(t) = (1-t)^2P_0 + 2t(1-t)P_1 + t^2P_2$

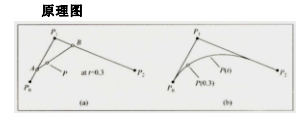
三次Bezier 曲线： $P(t) = (1-t)^3P_0 + 3t(1-t)^2P_1 + 3t^2(1-t)P_2 + t^3P_3$



曲线和曲面

- **曲线**
  - **参数样条曲线**
    - **贝塞尔(Bezier)曲线**

原理图



- ✓ B-样条多项式次数独立于控制点的个数
- ✓ B-样条允许曲线可以局部控制

- **B样条曲线**

$$\begin{cases} B_{i,1}(t) = \begin{cases} 1, & when \quad t \in [t_i, t_{i+1}] \\ 0, & other \end{cases} \\ B_{i,k}(t) = \dfrac{t - t_i}{t_{i+k-1} - t_i} B_{i,k-1}(t) + \dfrac{t_{i+k} - t}{t_{i+k} - t_{i+1}} B_{i+1,k-1}(t), \end{cases}$$

$$P(t) = \sum_{i=1}^{n} P_i B_{i,k}(t), \qquad t \in [t_{k-1}, t_{n+1}]$$

曲线表达式

上述基函数中，i = 0, 1, …n

曲线和曲面

- 曲线
  - 参数样条曲线
    - 非均匀有理B样条(NURBS)曲线

$$R(t)=\frac{\sum_{i=1}^{n}h_iP_iB_{i,k}(t)}{\sum_{i=1}^{n}h_iB_{i,k}(t)}$$

其中P_i(i=1,2,...,n)为控制顶点，h_i(i=1,2,...,n)称为权或权因子，分别与控制顶点相联系。其中首、末权因子大于零，其余权因子不小于零。控制顶点顺序连成控制多边形。其节点向量是一般非均匀的。当所有权因子均为1时，*NURBS*曲线就成为*B*样条曲线。

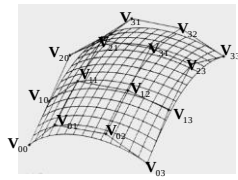
曲线和曲面

- 曲线
  - 参数样条曲线
    - 非均匀有理B样条(NURBS)曲线
      - ✓ 对标准的解析形状（如圆锥曲线、二次曲面、回转面等）和自由曲线、曲面提供了统一的数学表示，而且对二次曲线曲面的表示是精确的。
      - ✓ 由操纵控制顶点和权因子为各种形状设计提供了充分的灵活性。
      - ✓ 计算稳定且速度较快。
      - ✓ NURBS在比例、旋转、平移、剪切以及平行和透视投影变换下是不变的。
      - ✓ NURBS是非有理B样条形式以及Bezier形式的合适的推广。

曲线和曲面

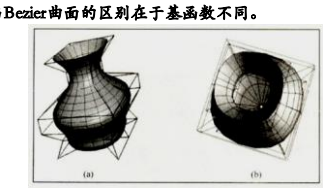
- 曲面
  - 贝塞尔(Bezier)曲面
    - 用控制多边形网格（特征网格）替代点矢、切矢与扭矢构造贝塞尔曲面。
    - 可以认为控制网格是曲面$P(u,w)$大致形状的勾画；$P(u,w)$是对控制网格的逼近。

✓ 以逼近为基础的曲面设计方法。它先通过控制顶点网格勾画出曲面的大体形状，然后通过修改控制顶点的位置修改曲面的形状。
✓ 比较直观，易为工程设计人员所接受。
✓ 但这种方法不具有局部性，即修改任意一个控制顶点都会影响整张曲面的形状。



曲线和曲面

- 曲面
  - B样条曲面
    - 用控制多边形网格（特征网格）替代点矢、切矢与扭矢构造B样条曲面。
    - 与Bezier曲面的区别在于基函数不同。

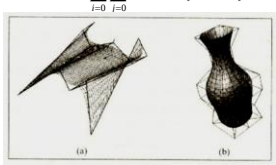


✓ 构造方法是Bezier曲面方法的推广。
✓ 用B样条基函数代替Bezier基函数来反映控制顶点对曲面形状的影响。
✓ 解决了Bezier曲面设计中存在的局部性修改问题。

曲线和曲面

- 曲面
  - 非均匀有理B样条曲面（NURBS）
    - 给定一张(m+1)x(n+1)的网格控制点P_{ij} (i=0,1,...,m; j=0,1,...,n),以及各控制网格点的权值W_{ij} (i=0,1,...,m; j=0,1,...,n)，则其确定的NURBS曲面的表达式为：

$$P(u,w)=\frac{\sum_{i=0}^{m}\sum_{j=0}^{n}N_{i,k}(u)N_{j,k}(w)W_{ij}P_{ij}}{\sum_{i=0}^{m}\sum_{j=0}^{n}N_{i,k}(u)N_{j,k}(w)W_{ij}}$$



✓ 规则曲面与自由曲面精确的统一的数学表示。
✓ 有多种方式定义曲面，但构造这些曲面的数学基础以及在造型系统中存储它们的方法是相同的。
✓ NURBS方法已成为众多CAD/CAM系统的基本几何表达式和数据交换标准。

2-4 专业图形程序接口(OpenGL)

- OpenGL简介

SGI公司推出的OpenGL因性能优越的交互式三维图形建模能力，得到了Microsoft、IBM、DEC、Sun、HP等大公司的认同。因此，OpenGL已经成为一种**三维图形开发标准**，是从事三维图形开发工作的必要工具。

OpenGL被设计成独立于硬件，独立于窗口系统（如MacOS、UNIX、Windows、Linux等），在运行各种操作系统的各种计算机上都可用，并能在网络环境下以客户/服务器模式工作，是专业图形处理、科学计算等高端应用领域的**标准图形库**。各种流行的编程语言都可以调用OpenGL中的库函数（如C、C++、C#、Java等）。

OpenGL是一个图形应用程序编程接口或函数库。

API： Application Programmer's Interface,
应用程序编程接口

2-4 专业图形程序接口(OpenGL)

- **OpenGL简介**
  - OpenGL是什么

从程序员的角度

图形应用程序的3要素
- 指定要绘制的对象
- 描述这些对象的属性
- 定义观察这些对象的方式

OpenGL是图形渲染管道

2-4 专业图形程序接口(OpenGL)

- **OpenGL简介**
  - OpenGL有什么

图元函数 ← 指定输入

属性函数
视窗函数
控制函数
→ 改变状态

OpenGL库函数
- (1) OpenGL核心库
- (2) OpenGL实用库
- (3) OpenGL辅助库
- (4) OpenGL工具库
- (5) Windows专用库
- (6) Win32 API函数库

2-4 专业图形程序接口(OpenGL)

- **OpenGL简介**
  - OpenGL库函数

(1) OpenGL核心库，包含有115个函数，函数名的前缀为gl。
这部分函数用于常规的、核心的图形处理。由于许多函数可以接收不同数据类型的参数，因此派生出来的函数原形多达300多个。

(2) OpenGL实用库，包含有43个函数，函数名的前缀为glu。
这部分函数通过调用核心库的函数，为开发者提供相对简单的用法，实现一些较为复杂的操作。如：坐标变换、纹理映射、绘制椭球、茶壶等简单多边形。OpenGL中的核心库和实用库可以在所有的OpenGL平台上运行。

(3) OpenGL辅助库，包含有31个函数，函数名前缀为aux。
这部分函数提供窗口管理、输入输出处理以及绘制一些简单三维物体。OpenGL中的辅助库不能在所有的OpenGL平台上运行。

2-4 专业图形程序接口(OpenGL)

- **OpenGL简介**
  - OpenGL库函数

(4) OpenGL工具库，包含大约30多个函数，函数名前缀为glut。
这部分函数主要提供基于窗口的工具，如多窗口绘制、空消息和定时器，以及一些绘制较复杂物体的函数。由于glut中的窗口管理函数是不依赖于运行环境的，因此OpenGL中的工具库可以在所有的OpenGL平台上运行。

(5) Windows专用库，包含有16个函数，函数名前缀为wgl。
这部分函数主要用于连接OpenGL和Windows95/NT，以弥补OpenGL在文本方面的不足。Windows专用库只能用于Windows95/98/NT环境中。

(6) Win32API函数库，包含有6个函数，函数名无专用前缀。
这部分函数主要用于处理像素存储格式和双帧缓存。这6个函数将替换Windows GDI中原有的同样的函数。Win32API函数库只能用于Windows/95/98/NT环境中。

2-4 专业图形程序接口(OpenGL)

- **OpenGL简介**
  - OpenGL中的数据类型

缩写	数据类型	C	OpenGL
b	8位整数	signed char	GLbyte
ub	8位无符号整数	unsigned char	GLubyte GLboolean
s	16位整数	short	GLshort
us	16位无符号整数	unsigned short	GLushort
i	32位整数	int	GLint GLsizei
ui	32位无符号整数	unsigned int	GLuint GLenum GLbitfield
f	32位浮点数	float	GLfloat GLclampf
d	64位浮点数	double	GLdouble GLclampd
		void	GLvoid

2-4 专业图形程序接口(OpenGL)

- **OpenGL简介**
  - OpenGL中函数的多种形式

```
glVertex{234}{sifd}(TYPE coords, …)
glVertex{234}{sifd}v(TYPE *coords)

GLint i, j;
GLfloat x, y, z, point[3];
                …
glVertex2i(i, j);
glVertex2f(x, y);
glVertex3f(x, y, z);
glVertex3fv(point);
```

详见 gl.h中定义

2-4 专业图形程序接口(OpenGL)

- **OpenGL简介**
  - **OpenGL中的部分常数及其含义**

字符	含义
GL_POINTS	绘制单个顶点集
GL_LINES	绘制多组独立的双顶点线段
GL_AMBIENT	设置RGBA模式下的环境光
GL_POSITION	设置光源位置
GL_FLAT	设置平面明暗处理模式
GL_SMOOTH	设置光滑明暗处理模式
……	……

2-4 专业图形程序接口(OpenGL)

- **OpenGL绘图功能**
  - **OpenGL中的绘制方式**

（1）线框绘制方式（Wire frame）：绘制三维物体的网格轮廓线。
（2）深度优先线框绘制方式（Depth cued）：采用线框方式绘图，使远处的物体比近处的物体暗一些，以模拟人眼看物体的效果。
（3）反走样线框绘制方式（Antialiased）：采用线框方式绘图，绘制时采用反走样技术，以减少图形线条的参差不齐。
（4）平面明暗处理方式（Flat shading）：对模型的平面单元按光照进行着色，但不进行光滑处理。
（5）光滑明暗处理方式（Smooth shading）：对模型按光照绘制的过程进行光滑处理，这种方式更接近于现实。

2-4 专业图形程序接口(OpenGL)

- **OpenGL绘图功能**
  - **OpenGL中的绘制方式**

（6）加阴影和纹理的方式（Shadow and Texture）：在模型表面贴上纹理甚至加上光照阴影效果，使三维场景像照片一样逼真。
（7）运动模糊绘制方式（Motion blured）：模拟物体运动时人眼观察所觉察到的动感模糊现象。
（8）大气环境效果（Atmosphere effects）：在三维场景中加入雾等大气环境效果，使人有身临其境之感。
（9）深度域效果（Depth of effects）：类似于照相机镜头效果，模拟在聚焦点处清晰。

2-4 专业图形程序接口(OpenGL)

- **OpenGL绘图功能**
  - **OpenGL中的绘图功能**

（1）建模功能　　**（2）变换功能**
（3）颜色模式设置　　**（4）光照和材质设置**
（5）反走样、融合、雾化　　**（6）位图显示和图像增强**
（7）纹理映射　　**（8）双缓存动画**

2-4 专业图形程序接口(OpenGL)

- **OpenGL绘图功能**
  - **（1）建模功能**

真实世界里的任何物体都可在计算机中用简单的点、线、多边形描述，OpenGL除了提供基本的点、线、多边形的绘制函数外，还提供了比较复杂的三维物体(如球、锥体、多面体、茶壶等)以及复杂曲线和曲面(如Bezier、Nurbs等曲线和曲面)绘制函数，从而可方便构建虚拟三维世界。

2-4 专业图形程序接口(OpenGL)

- **OpenGL绘图功能**
  - **（2）变换功能**

无论多复杂的图形都是由基本图元组成并经过一系列变换来实现的。OpenGL的模型变换有平移、旋转、缩放等多种变换。投影变换有透视投影和正交投影两种变换。

  - **（3）颜色模式设置**

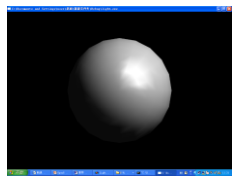
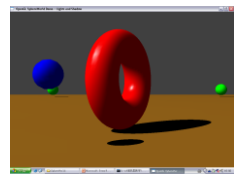
OpenGL提供两种物体着色模式:
- ✓ RGBA颜色模式
- ✓ 颜色索引模式(Color Index)

2-4 专业图形程序接口(OpenGL)

- **OpenGL绘图功能**

（4）光照和材质设置

OpenGL光源属性有辐射光(Emitted Light)、环境光(Ambient Light)、漫反射光(Diffuse Light)和镜面光(Specular Light)等。材质是用光反射率来表示。场景中物体最终反映到人眼的颜色是光的RGB分量与材质的RGB分量反射率相乘后形成的颜色。

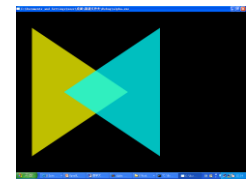
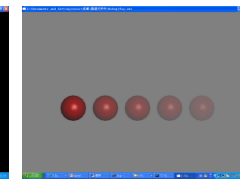



2-4 专业图形程序接口(OpenGL)

- **OpenGL绘图功能**

（5）反走样、融合、雾化

OpenGL提供了点、线、多边形的反走样技术。为了使三维图形更加有真实感，经常需要处理半透明或透明的物体图像，这就用到融合技术。OpenGL提供了雾的基本操作来对场景进行雾化处理效果。

2-4 专业图形程序接口(OpenGL)

- **OpenGL绘图功能**

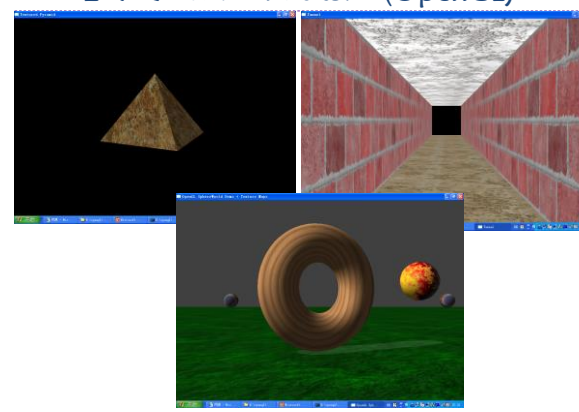
（6）位图显示和图像增强

在图形绘制过程中，图像和位图是非常重要的一个方面，OpenGL提供了一系列函数来实现位图和图像的操作。

（7）纹理映射

在计算机图形学中，将包含颜色、alpha值、亮度等数据的矩形数组称为纹理。而纹理映射可理解为将纹理粘贴在所绘制的三维模型表面，以使三维图形显得更加生动。

2-4 专业图形程序接口(OpenGL)



2-4 专业图形程序接口(OpenGL)

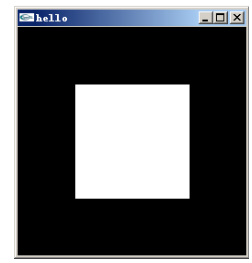
- **OpenGL绘图功能**

（8）双缓存动画

OpenGL提供了双缓存技术来实现动画绘制。双缓存即前台缓存和后台缓存，后台缓存计算场景、生成动画，前台缓存显示后台缓存已画好的画面。

2-4 专业图形程序接口(OpenGL)

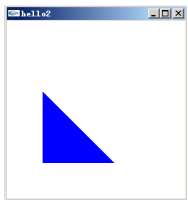
- **程序实例**




```
#include <GL/glut.h>

void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POLYGON);
                glVertex2f(-0.5, -0.5);
                glVertex2f(-0.5, 0.5);
                glVertex2f(0.5, 0.5);
                glVertex2f(0.5, -0.5);
        glEnd();
        glFlush();
}

void main(int argc, char * argv[])
{
        glutInit(&argc,argv);
        glutCreateWindow("Simple.C");
        glutDisplayFunc(display);
        glutMainLoop();
}
```

2-4 专业图形程序接口(OpenGL)

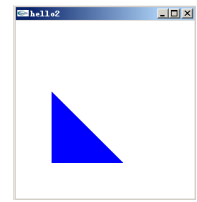
- 程序实例




```
/*
 * hello2.c
 * This is a simple flat color draw blue tirangle | OpenGL program.
 */
#define GLUT_DISABLE_ATEXIT_HACK
#include <GL/glut.h>

void display(void)
{
/* clear all pixels  */
   glClear (GL_COLOR_BUFFER_BIT);
   glShadeModel(GL_FLAT);

/* draw Blue triangle  with corners at
 * (0.2,0.2) , (0.6, 0.2) and (0.2,0.6)
 */

   glBegin(GL_TRIANGLES);
        glColor3f(0.0, 0.0, 1.0);
        glVertex2f(0.2, 0.2);
        glVertex2f(0.6, 0.2);
        glVertex2f(0.2, 0.6);
   glEnd();

/* don't wait!
 * start processing buffered OpenGL routines
 */
   glFlush ();
}
```

2-4 专业图形程序接口(OpenGL)

- 程序实例

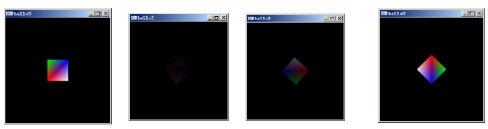



```
void init (void)
{
/* select clearing color    */
   glClearColor (1.0, 1.0, 1.0, 0.0);

/* initialize viewing values  */
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

/*
 * Declare initial window size, position, and display mode
 * (single buffer and RGBA).  Open window with "hello2"
 * in its title bar.  Call initialization routines.
 * Register callback function to display graphics.
 * Enter main loop and process events.
 */
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (250, 250);
   glutInitWindowPosition (100, 100);
   glutCreateWindow ("hello2");
   init ();
   glutDisplayFunc(display);
   glutMainLoop();

}
```

2-4 专业图形程序接口(OpenGL)

- 程序实例

绘制一个黑窗口，并显示一个彩色的平滑模式绘制四边形，鼠标左键点击后，目标会进行旋转，右键点击旋转停止。此外，各个点随着旋转，颜色保持不变，深浅要变化。



开始　　旋转过程出现深浅变化　　右键点击旋转停止 颜色变化也停止

2-4 专业图形程序接口(OpenGL)

- 程序实例

```
/*
 *  double.c
 *  This is a simple double buffered program.
 *  Pressing the left mouse button rotates the rect.
 *  Pressing the right mouse button stops the rotation.
 */
#define GLUT_DISABLE_ATEXIT_HACK
#include <GL/glut.h>
#include <stdlib.h>

static GLfloat spin = 0.0;
static GLfloat chagecolor=1.0;

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glPushMatrix();
   glRotatef(spin, 0.0, 0.0, 1.0);

   glBegin(GL_POLYGON);
        glColor3f(chagecolor, 0.0, 0.0);
        glVertex2f(-10.0,-10.0);
        glColor3f(0.0,chagecolor, 0.0);
        glVertex2f(-10.0, 10.0);
        glColor3f(0.0, 0.0,chagecolor);
        glVertex2f(10.0, 10.0);
        glColor3f(chagecolor, chagecolor, chagecolor);
        glVertex2f(10.0, -10.0);
    glEnd();

   glPopMatrix();

   glutSwapBuffers();
}
```

2-4 专业图形程序接口(OpenGL)

- 程序实例

```
void spinDisplay(void)
{
   spin = spin + 0.02;
   if (spin > 360.0)
      spin = spin - 360.0;
   chagecolor=chagecolor+0.01;
   if (chagecolor > 1.0)
      chagecolor = chagecolor - 1.0;
   glutPostRedisplay();
}

void init(void)
{
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glShadeModel (GL_SMOOTH);
}
```

```
void reshape(int w, int h)
{
   glViewport (0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

void mouse(int button, int state, int x, int y)
{
   switch (button) {
      case GLUT_LEFT_BUTTON:
         if (state == GLUT_DOWN)
            glutIdleFunc(spinDisplay);
         break;
      case GLUT_MIDDLE_BUTTON:
      case GLUT_RIGHT_BUTTON:
         if (state == GLUT_DOWN)
            glutIdleFunc(NULL);
         break;
      default:
         break;
   }
}
```

2-4 专业图形程序接口(OpenGL)

- 程序实例

```
/*
 *  Request double buffer display mode.
 *  Register mouse input callback functions
 */
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
   glutInitWindowSize (250, 250);
   glutInitWindowPosition (100, 100);
   glutCreateWindow ("hello3");
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutMouseFunc(mouse);
   glutMainLoop();

}
```

2-4 专业图形程序接口(OpenGL)

- 程序实例

第1行，三条线，每个的sittple都不同，具体为：
重复次数　　模式
1　　0x0101
1　　0x00FF
1　　0x1C47

第2行，三条线，stipple同上要求，但线宽要求为5.0

第3行，1条线，使用一个stipple
重复次数　　模式
5　　0x1C47

2-4 专业图形程序接口(OpenGL)

- 程序实例

```
/*
 *  lines.c
 *  This program demonstrates geometric primitives and
 *  their attributes.
 */
#define GLUT_DISABLE_ATEXIT_HACK
#include <GL/glut.h>
#include <stdlib.h>

#define drawOneLine(x1,y1,x2,y2)  glBegin(GL_LINES);  \
   glVertex2f ((x1),(y1)); glVertex2f ((x2),(y2)); glEnd();

void init(void)
{
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glShadeModel (GL_FLAT);
}
void display(void)
{
   int i;

   glClear (GL_COLOR_BUFFER_BIT);

/* select white for all lines  */
   glColor3f (1.0, 1.0, 1.0);

/* in 1st row, 3 lines, each with a different stipple  */
   glEnable (GL_LINE_STIPPLE);


   glLineStipple (1, 0x0101);  /*  dotted  */
   drawOneLine (50.0, 125.0, 150.0, 125.0);
   glLineStipple (1, 0x00FF);  /*  dashed  */
   drawOneLine (150.0, 125.0, 250.0, 125.0);
   glLineStipple (1, 0x1C47);  /*  dash/dot/dash  */
   drawOneLine (250.0, 125.0, 350.0, 125.0);

/* in 2nd row, 3 wide lines, each with different stipple */
```

2-4 专业图形程序接口(OpenGL)

- 程序实例

```
   glLineWidth (5.0);
   glLineStipple (1, 0x0101);  /*  dotted  */

   drawOneLine (50.0, 100.0, 150.0, 100.0);
   glLineStipple (1, 0x00FF);  /*  dashed  */

   drawOneLine (150.0, 100.0, 250.0, 100.0);
   glLineStipple (1, 0x1C47);  /*  dash/dot/dash  */

   drawOneLine (250.0, 100.0, 350.0, 100.0);
   glLineWidth (1.0);

/* in 3rd row, 1 line, with dash/dot/dash stipple    */
/* and a stipple repeat factor of 5                  */
   glLineStipple (5, 0x1C47);  /*  dash/dot/dash  */
   drawOneLine (50.0, 75.0, 350.0, 75.0);

   glDisable (GL_LINE_STIPPLE);
   glFlush ();
}

void reshape (int w, int h)
{
   glViewport (0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   gluOrtho2D (0.0, (GLdouble) w, 0.0, (GLdouble) h);
}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (400, 150);
   glutInitWindowPosition (100, 100);
   glutCreateWindow ("lines");
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutMainLoop();
   return 0;
}
```

学习重点

- **三维模型的三种数据模型**
  - 线框模型、表面模型、实体模型
- **三维图形显示的关键技术**
  - 数学工具、物体变换与三维观察、真实感渲染、曲线和曲面
- **OpenGL的开发技术**

主要参考文献

- Francis S Hill, Stephen M Kelley 著, 胡事民等译. 计算机图形学(OpenGL版)(第3版). 清华大学出版社：2009
- Dave Shreiner 著，李军等译. OpenGL编程指南(原书第7版). 机械工业出版社：2010
- 孙家广等. 计算机图形学(第三版）. 清华大学出版社：1998

提高内容参考

- 具备拓扑关系的三维实体模型的数据结构设计，并实现其布尔运算；
- 开发一个基于点插值技术的关键帧动画制作程序，并实现一个约10秒钟长度的小动画（或动漫）；
- 针对基于三角形网格描述的实体模型，提出一个高效的碰撞检测算法；
- 基于OpenGL，实现多光源下的阴影显示；
- 开发一个贝塞尔曲线和曲面建模的程序；
- 基于OpenGL，开发一个能通过截面拉伸和旋转建立三维模型的程序。

谢谢！

清华大学土木工程系

胡振中

huzhenzhong@tsinghua.edu.cn