

图形学复习

第一章 计算机图形学概论

- 1.1 全书概述
- 1.2 计算机图形学定义
- 1.3 计算机图形学研究内容
- 1.4 计算机图形学的发展历史
- 1.5 计算机图形学的应用领域
- 1.6 计算机图形系统组成
- 1.7 图形显示设备
- 1.8 图形学相关概念
- 1.9 图形图像的区别及存储格式

第二章 光栅图形学算法

- 2.1 直线段的扫描转换算法
 - 2.1.1 直线段扫描转换算法概述
 - 2.1.2 数值微分法 (DDA)
 - 2.1.3 中点画线法
 - 2.1.4 Bresenham 算法
- 2.2 多边形的扫描转换
 - 2.2.1 多边形表示方法
 - 2.2.2 X-扫描线算法
 - 2.2.3 边缘填充算法
 - 2.2.4 栅栏填充算法
 - 2.2.5 边界标志算法
- 2.3 区域填充
 - 2.3.1 区域填充基础概念
 - 2.3.2 简单四连通种子填充算法
 - 2.3.3 区域填充与多边形扫描转化对比
- 2.4 反走样
 - 2.4.1 走样现象
 - 2.4.2 反走样技术

第三章 裁剪

- 3.1 裁剪概述
- 3.2 Cohen-Sutherland 算法
- 3.2 中点分割算法
- 3.3 Liang-Barsky 算法
- 3.4 多边形、字符裁剪
 - 3.4.1 多边形的裁剪
 - 3.4.2 文字裁剪

第四章 消隐

- 4.1 消隐概述
 - 4.1.1 消隐定义
 - 4.1.2 消隐分类
- 4.2 物体空间的消隐算法
 - 4.2.1 Roberts 算法
 - 4.2.2 光线投射法
- 4.3 Z 缓冲区算法 (Z-Buffer)
 - 4.3.1 Z-Buffer 算法概述
 - 4.3.2 Z-Buffer 算法改进
- 4.4 区间扫描线算法
- 4.5 Warnock 消隐算法
- 4.6 光栅扫描算法小结

第五章 二维图形变换

- 5.1 向量基础
- 5.2 图形坐标系
- 5.3 二维图形变换原理
 - 5.3.1 图形变换概述
 - 5.3.2 仿射变换
 - 5.3.3 齐次坐标
- 5.4 基本几何变换
 - 5.4.1 平移变换
 - 5.4.2 比例变换
 - 5.4.3 对称变换
 - 5.4.4 旋转变换
 - 5.4.5 错切变换
 - 5.4.6 复合变换
 - 5.4.7 任意参考点的几何变换

- 5.5 二维变换矩阵
 - 5.5.1 二维变换矩阵概述
 - 5.5.2 二维图形几何变换的计算
- 5.4 窗口、视图及变换
 - 5.4.1 窗口和视区
 - 5.4.2 观察变换
 - 5.4.3 窗口到视区的变换

第4章 三维图形变换

- 4.1 三维图形几何变换
 - 4.1.1 几何变换概述
 - 4.1.2 平移变换
 - 4.1.3 比例变换
 - 4.1.4 旋转变换
 - 4.1.5 对称变换
- 4.2 投影变换分类
- 4.3 平行投影(三视图、轴测图)
 - 4.3.1 平行投影概述
 - 4.3.2 三视图
 - 4.3.3 正轴侧
- 4.4 透视投影
 - 4.4.1 透视投影概述
 - 4.4.2 一点透视
 - 4.4.3 多点透视
 - 4.4.4 生成透视投影图的方法
 - 4.4.5 一点透视投影实例
 - 4.4.6 二点透视投影实例：

第七章 曲线曲面(一)

- 7.1 几何造型简史
- 7.2 参数曲线基本概念
 - 7.2.1 曲线和曲面的三种表示方法
 - 7.2.2 参数曲线的基本概念
- 7.3 Bezier曲线与曲面
 - 7.3.1 Bezier曲线的背景和定义
 - 7.3.2 Bernstein基函数的性质
 - 7.3.3 Bezier曲线的性质

第八章 曲线曲面(二)

- 8.1 Bezier曲线生成算法
 - 8.1.1 根据定义直接生成Bezier曲线

- 8.1.2 Bezier曲线的递推算法
 - 8.1.3 de Casteljau算法几何作图
 - 8.2 Bezier曲线的拼接及升降阶
 - 8.2.1 Bezier曲线的拼接
 - 8.2.2 Bezier曲线的升阶与降阶
 - 8.3 Bezier曲面
 - 8.3.1 Bezier曲面的定义
 - 8.3.2 Bezier曲面性质
 - 8.3.3 递推算法(曲面的求值)
 - 8.4 B样条曲线产生背景及定义
 - 8.4.1 Bezier曲线缺点
 - 8.4.2 B样条方法
 - 8.4.3 B样条的递推定义和性质
 - 8.4.4 de Boor-Cox 递推定义
 - 8.4.5 B样条基函数定义区间及节点向量
 - 8.5 B样条曲线性质及类型划分
 - 8.5.1 B样条基函数的主要性质
 - 8.5.2 B样条函数的主要性质
 - 8.5.3 B样条曲线类型的划分
 - 8.6 B样条曲面
- 第九章 真实感图形学
- 9.1 真实感图形学概述
 - 9.2 颜色模型
 - 9.2.1 颜色及颜色模型概述
 - 9.2.2 RGB颜色工业模型
 - 9.2.3 其他颜色工业模型
 - 9.2.4 颜色视觉模型
 - 9.3 简单光照模型
 - 9.3.1 光照模型的发展
 - 9.3.2 光照背景物理知识
 - 9.3.3 Phong光照模型
 - 9.4 增量式光照模型
 - 9.4.1 光暗处理
 - 9.4.2 Gouraud明暗处理
 - 9.4.3 Phong明暗处理
 - 9.5 局部光照模型
 - 9.5.1 局部光照模型概述
 - 9.5.2 理论基础
 - 9.5.3 局部光照明模型
 - 9.6 光透射模型
 - 9.6.1 光透射模型概述
 - 9.6.2 颜色调和法
 - 9.6.3 Whitted 光透射模型
 - 9.7 整体光照模型
 - 9.7.1 整体光照模型的诞生
 - 9.7.2 光线跟踪基本过程
 - 9.7.3 光线跟踪的停止
 - 9.7.4 光线跟踪的加速
 - 9.8 纹理映射
 - 9.8.1 纹理映射作用
 - 9.8.2 纹理分类
 - 9.8.3 纹理定义
 - 9.8.4 纹理映射
 - 9.9 阴影处理

- 9.9.1 阴影定义
- 9.9.2 阴影分类
- 9.9.3 阴影体法
- 9.9.4 阴影图法

第十章 绘制技术

- 10.1 基于图像的绘制技术IBR
 - 10.1.1 传统图像绘制与IBR比较
 - 10.1.2 IBR发展方向
 - 10.1.3 IBR的绘制过程
- 10.2 基于点的建模与绘制
 - 10.2.1 点建模与绘制方法
 - 10.2.2 基于点的建模与绘制的发展趋势

图形学复习

- 题型
 - 5道填空或选择 (5*2)
 - 6道大题 (6*15)
- 知识点
 - 直线扫描线
 - 扫描线填充
 - 窗口填充
 - Z-Buffer
 - 三次Hermite插值
 - 系数的概念及其推导
 - de Casteljau算法
 - 作图
 - Bezier和Bernstein基函数
 - 曲线升降阶
 - 几何变换
 - 渲染
 - 碰撞

6个大题

直线扫描线算法 必考

多边形裁剪 多边形填充扫描线 必考

Bezier BSpline 重点 推导生成 Harrmit插值

曲线的几何性质，比如怎么画图、脑率

矩阵变换，平移、旋转、投影(透视)变换(一点透视、两点透视、三点透视)

phong光照模型 重点

第一章 计算机图形学概论

1.1 全书概述

- 图形学基本知识
 - 光栅图形学
 - 扫描转换、区域填充、裁减、反走样、消隐
 - 二维、三维图形变换及观察
 - 几何造型
 - 参数曲线曲面基本概念、Bezier曲线曲面、B样条曲线等
 - 真实感图形学
 - 颜色模型、简单光照模型、纹理映射、光线跟踪
-

1.2 计算机图形学定义

- 计算机图形学
 - 计算机图形是计算机产生的图像。
 - 计算机图形学就是研究如何在计算机中表示图形、以及利用计算机进行图形的计算、处理和显示的相关原理和算法。
 - IEEE定义：**Comput graphics** is the art or science of producing graphical images with the aid of computer.
 - 计算机图形学的发展和应用在某种意义上已成为计算机软、硬件发展水平的标志。
-

1.3 计算机图形学研究内容

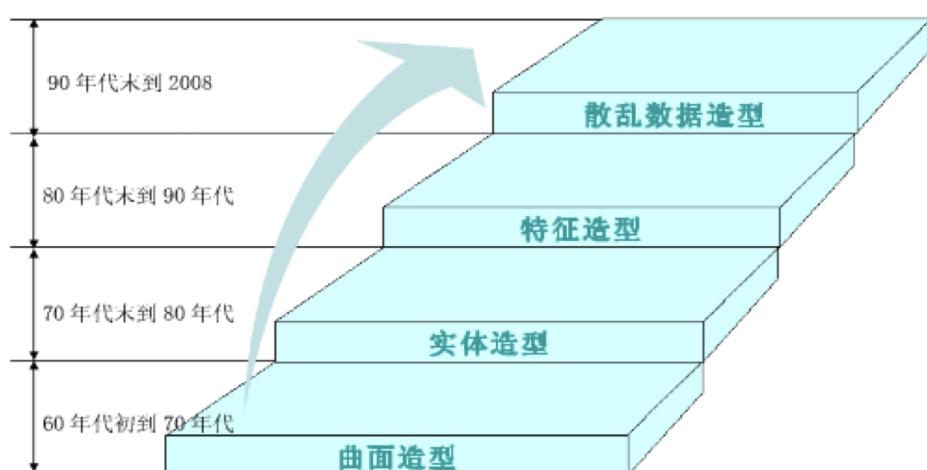
- 主要研究内容
 - 如何在计算机中表示图形、以及利用计算机进行图形的计算、处理和显示的相关原理与算法，构成了计算机图形学的主要研究内容。
- 计算机生成一副表示物体的图形的三个步骤
 - **造型技术**
 - 在计算机中建立所要生成图像的物体的模型，即给出表示该物体的几何数据和拓扑关系
 - 比如教室里的桌子、椅子、墙，用圆柱、平面等表示出来。
 - **光照模型**
 - 希望用一些简单的数学模型来近似、代替那些物理学的模型，为模拟物体表面的光照物理现象的数

学模型叫光照模型。

- 绘制（渲染）技术
 - 选择适当的绘制算法来把这个场景画（渲染）出来。
 - 绘制一幅三维物体图像所涉及的知识，实际上就是计算机图形中每个像素看上去应该是什么颜色的问题。
 - 计算机图形的发展方向
 - 准确性 -> 真实性 -> 实时性
-

1.4 计算机图形学的发展历史

- 1950年，第一台图形显示器作为美国麻省理工学院 (MIT) 旋风 I 号计算机的附件诞生。
- 1963年，Sutherland 发表博士论文。其中第一次提出了 graphics 这个词。
 - Sutherland 被公认为开创交互式图形技术的奠基人，被称为“计算机图形学之父”，并于 1988 年获“图灵奖”。
- 1962年，雷诺汽车公司的工程师 Bezier 提出 Bezier 曲线、曲面的理论，成为 CAGD (计算机辅助几何设计) 的先驱。
- 1964年，MIT教授 Steven A. Coons 提出了超限插值的新思想，通过插值四条任意的边界曲线来构造曲面。
- 70年代，光栅显示器出现了。光栅显示器屏幕是由像素组成的，由此诞生了大量算法，如区域填充、裁剪、消隐等基本图形概念、及其相应算法。
 - 真实感图形和几何造型技术这个时候也开始出现了。
- 1975年，Phong 提出了著名的简单光照模型 - Phong 模型（标志着真实感图形的出现和实用化，直到现在 Phong 模型还被大量的采用）
- 1980年，Whitted 提出了光透视模型 - Whitted 模型，成为第一次提出光线跟踪算法的范例。
- 几何造型技术：通俗地讲，该技术就像小孩搭积木，用简单的一些体素来构建复杂的模型。
-

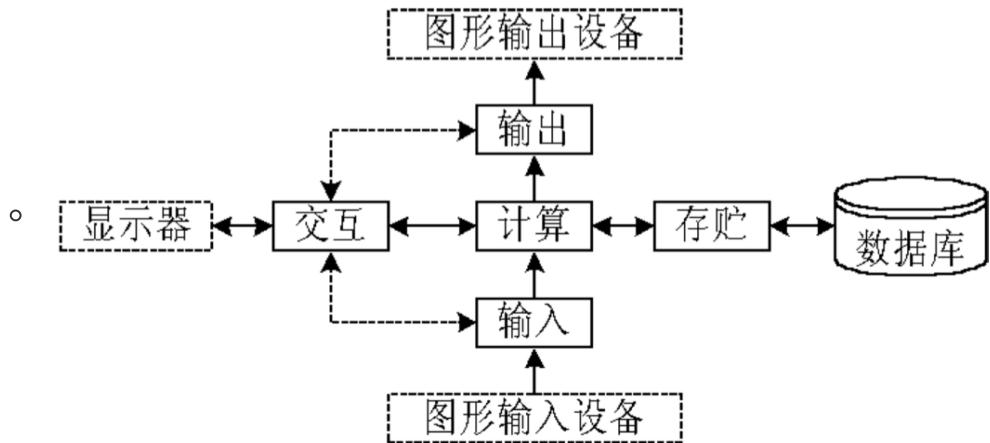


1.5 计算机图形学的应用领域

- 人机交互和图形用户界面
 - 最理想的是开发 "能听、能说、能理解人类语言" 的计算机，人们可以和计算机交谈，而不像现在这样仅限于窗口、图标、鼠标、指针 (WIMP) 界面。
- 计算机辅助设计与制造 (CAD/CAM)
 - CAD/CAM 是计算机图形学在工业界最广泛、最活跃的应用领域。
 - 飞机、汽车、船舶、宇宙飞船的外形设计
 - 发电厂、化工厂等的布局
- 真实感图形实时绘制与自然景物仿真
 - 计算机中重现真实世界的场景叫做真实感绘制。
- 计算机动画、游戏、电影
- 计算机艺术
 - 计算机艺术是科学与技术相结合的一门新兴的交叉学科，是计算机应用的一个崭新、富有时代气息的领域。
- 计算机仿真
 - 计算机仿真是计算机技术建立被仿真系统的模型，并在某些实验条件下对模型进行动态实验的一门综合性技术。
- 科学计算可视化
- 虚拟现实
 - 虚拟现实是利用计算机模拟现实的场景，使参与者获得与现实一样的感觉。
 - 准确地说，是利用电脑模拟产生一个三维空间的虚拟世界，提供使用者关于视觉、听觉、触觉等感官的模拟，让使用者如同身历其境一般，可以及时、没有限制地观察三度空间内的事物。
- 地理信息系统
 - 地理信息系统是建立在地理图形之上的关于各种资源的综合信息管理系统，是计算机图形学的一个重要应用领域。
- 农业上的应用
 - 借助计算机图形生成技术来保存和再现不同作物种类和不同生长期的植物形态，模拟植物的生长过程，从而合理地进行选种、播种、田间管理以及收获等。

1.6 计算机图形系统组成

- 五大功能
 - 一个交互式计算机图形系统应具有计算、存储、对话、输入和输出等 5 个方面的功能。



- 图形系统
 - 图形软件
 - 图形应用数据结构：对应一组图形数据文件，其中存放着欲生成的图形对象的全部描述信息。
 - 图形应用软件
 - 解决某种应用问题的图形软件，是图形系统中的核心部分，包括了各种图形生成和处理技术。如：photoshop、3Dmax等。
 - 图形支撑软件：大多数图形应用程序是建立在一定的图形支撑软件上。图形支撑软件需具有规范接口。
 - 图形硬件

1.7 图形显示设备

- 阴极射线管
 - 使用广泛：现在的图形显示设备绝大多数是基于阴极射线管 (CRT) 的显示器。
 - 阴极射线管的技术指标
 - 分辨率：一个阴极射线管在水平和垂直方向单位长度上能识别出的最大光点数称之为分辨率。光点亦称之为像素 (pixel)。
 - 显示速度
 - 要保持荧光屏上有稳定的图像就必须不断地发射电子束。只有刷新频率高到一定值后，图像才能稳定显示。大约达到每秒 60 帧即 60Hz 时，人眼才能感觉到屏幕不闪烁，要人眼觉得舒服，一般必须有 85Hz 以上的刷新频率。
- 彩色阴极射线管
 - 三只电子枪，分别涂有红、绿、蓝三种颜色的光。
- CRT图形显示器
 - 随机扫描的图形显示器 (画线设备)
 - 电子束的扫描轨迹随显示内容而变化，只在需要的地方扫描，而不必全屏扫描，因此速度快，图像清晰。
 - 一条线一条线地画图，因此也称为向量显示器。
 - 随机扫描系统是为画线应用设计的，因此不能显示逼真的有阴影场景。
 - 光栅扫描显示器 (画点设备)
 - 不能直接从一个可编地址的像素画一条直线到另一个可编地址的像素，只可能用尽可能靠近这条直

线路径的像素点来近似地表示这条直线。

- 在光栅扫描系统中，电子束横向扫描屏幕，一次一行，从上到底顺次进行。当电子束横向沿每一行移动时，电子束的强度不断变化来建立亮点的图案。
- 由于光栅扫描系统具有存储每一个屏幕点亮度信息的能力，所以，最适合显示浓淡和色彩图形。

- 例题

举例：显卡有2MB显存，当分辨率是 1024×768 时，可支持的色彩数又是多少？

$$2\text{MB} = 2 \times 1024 \times 1024 = 2097152 \text{ (字节)}$$

○

$$1024 \times 768 = 786432 \text{ (个像素)}$$

每个像素如果需要3个字节表示，将超过2MB显存，最多只需要2个字节表示，故只能支持64K色彩数

1.8 图形学相关概念

- 分辨率
 - 光点
 - 光点指电子束打在显示器荧光屏上，显示器能够显示的最小的发光点，一般用其直径来表明光点的大小。
 - 像素点
 - 像素点是指图形显示在屏幕上时候，按当前的图形显示分辨率所能提供的最小元素点。
 - 屏幕分辨率
 - 屏幕上显示的像素个数，以(水平像素数 * 垂直像素数)表示。
 - 显示分辨率
 - 是计算机显示控制器所能够控制的显示模式分辨率，简称显示模式。
 - 对于文本显示方式，显示分辨率用水平和垂直方向上所能显示的字符总数的乘积来表示。
 - 对于图形显示方式，则用水平和垂直方向上所能显示的像素点总数的乘积来表示。
 - 显卡分辨率
 - 显卡分辨率就是表示显卡输出给显示器，并能在显示器上描绘像素点的数量。
 - 一台电脑的最高分辨率是由显卡和显示器共同决定的。显示器最高分辨率是可以显示出来的最大分辨率。显卡的最大分辨率是最大能支持多少分辨率。
 - 电脑的最高分辨率取决于显卡和显示器最低的一个。
- 显示器的点距
 - 指相邻像素点之间的距离。两点之间的距离越小越好。
 - 15寸显示器，点距达到0.28mm就足够。17寸显示器，需要0.27mm、0.25mm等。
- 显示卡的作用与性能指标
 - 显示卡的基本作用就是显示图文，显示卡和显示器构成了计算机的显示系统。
 - 除了CPU和内存外，显卡对计算机的显示性能起着决定性的作用。

1.9 图形图像的区别及存储格式

- 图形图像的区别
 - 说法一
 - 图形是由计算机绘制而成的，而图像则是人为的用外部设备所捕捉到的外部的景象。
 - 说法二
 - 图形是矢量图，而图像是位图(点阵图)
- 图形(像)的构成属性
 - 几何属性
 - 刻画对象的轮廓、形状。包括点、线、面、体等。
 - 非几何属性
 - 视觉属性，刻画对象的颜色、材质等。包括明暗、色彩、纹理、透明性、线型、线宽。
 - 从构图要素上看，将图形分为两类
 - 几何属性有突出作用：工程图、等高线地图、曲面的线框图
 - 非几何属性有突出作用(明暗图)：真实感图形
- 位图和矢量图定义
 - 位图(点阵图)
 - 点阵图或像素图，计算机屏幕上的图是由屏幕上的像素构成的，每个点用二进制数据来描述其颜色与亮度等信息。
 - 矢量图
 - 面向对象的图形或绘图图形，是用数学方式描述的曲线及曲线围成的色块制作的图形。
 - 矢量文件中的图形元素称为对象。每个对象都是一个自成一体的实体，它具有颜色、形状、轮廓、大小和屏幕位置等属性。
- 位图和矢量图区别
 - 存储方式的区别
 - 点阵文件存储图的各个像素点的位置信息、颜色信息以及灰度信息。
 - 矢量文件是用数学方程、数学形式对图形进行描述，通常使用图形的形状参数和属性参数来表示图形。
 - 因此，点阵文件存储空间比矢量文件大。
 - 缩放的区别
 - 点阵文件与分辨率有关，即在一定面积的图像上包含有固定数量的像素。
 - 矢量图形与分辨率无关，可以将它缩放到任意大小和以任意分辨率在输出设备上打印出来，不会影响清晰度。
 - 存储格式的区别
 - 位图存储格式：BMP、TIFF、GIF、JPEG、PNG
 - 矢量图存储格式：DXF、SVG、EPS、WMF、EMF
 - 总结
 -

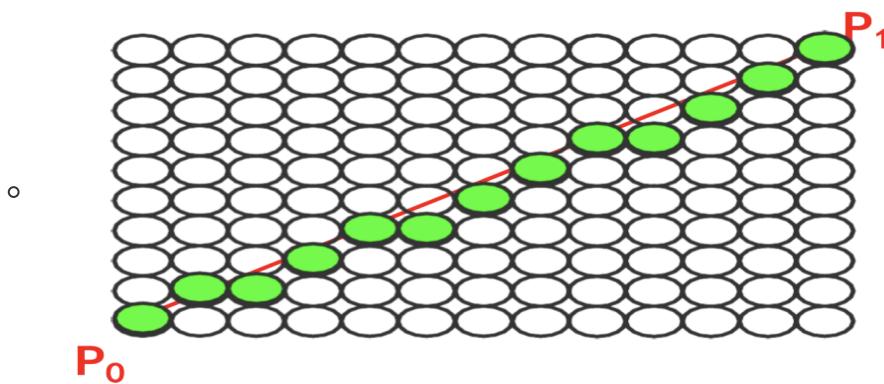
	位图	矢量图
存储内容	各像素点位置信息、颜色信息以及灰度信息	数学方程
存储空间	大	小
常见存储格式	BMP、TIFF、GIF、JPEG、PNG	DXF、SVG、EPS、WMF、EMF
图形缩放	失真	不失真
真实感图形效果	容易实现	不容易实现

第二章 光栅图形学算法

2.1 直线段的扫描转换算法

2.1.1 直线段扫描转换算法概述

- 光栅显示器屏幕上的直线
 - 核心方法：用离散像素点逼近直线



- 直线绘制的三个著名算法
 - 数值微分法 (DDA)
 - 中点画线法
 - Bresenham 算法

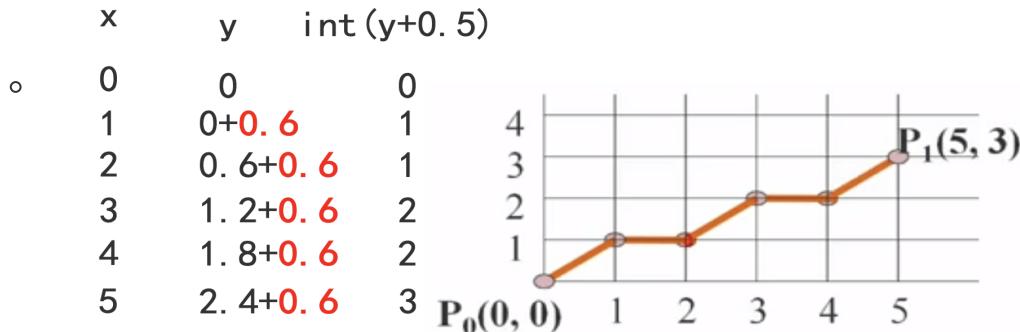
2.1.2 数值微分法 (DDA)

- 数值微分法 (DDA - Digital Different Analyzer)
 - 核心思想：增量思想
 - $y_i = kx_i + b, \quad y_{i+1} = kx_{i+1} + b$
 - $y_{i+1} = kx_{i+1} + b = kx_i + k + b = y_i + k, \quad \text{即 } y_{i+1} = y_i + k, \quad k \text{ 即为增量。}$
 - 因此即可将原式中的乘法换成加法。
- 算法过程

- 当 $|k| < 1$ 时， x 均匀增加，每次求出 x 对应的 y 值，再将 $y^* = (\text{int})(y + 0.5)$ 下取整 y^* ，得到 (x, y^*) 像素点坐标，再将该像素点涂色即可。

用DDA扫描转换连接两点 $P_0(0, 0)$ 和 $P_1(5, 3)$ 的直线段。

$$k = \frac{y_1 - y_0}{x_1 - x_0} = \frac{3 - 0}{5 - 0} = 0.6 < 1 \quad y_{i+1} = y_i + k$$



- 算法总结

- 当 $|k| \leq 1$ 时， $x_{i+1} = x_i + 1, y_{i+1} = y_i + k$ ，每次令 $y^* = (\text{int})(y_i + 0.5)$ ，将像素点 (x_i, y^*) 涂色。
- 当 $|k| > 1$ 时， x, y 互换，即 $y_{i+1} = y_i + 1$ 。
 - 因为 $|k| > 1$ 时， y 坐标范围更广，因此令 y 递增求 x 。

2.1.3 中点画线法

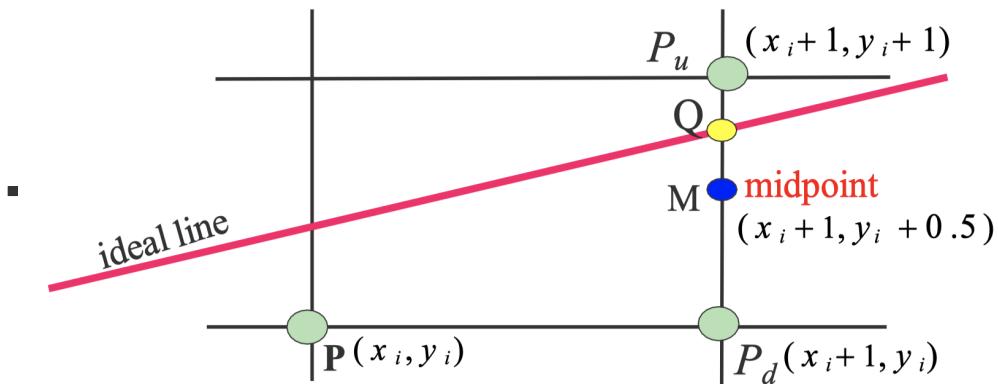
- 中点画线法

- 直线一般式方程

- $F(x, y) = 0, Ax + By + C = 0$
 - 直线上的点: $F(x, y) = 0$
 - 直线上方的点: $F(x, y) > 0$
 - 直线下方的点: $F(x, y) < 0$

- 核心思想

- 每次在最大位移方向走一步，而另一个方向是走还是不走取决于中点误差项的判断。
 - 假定: $0 \leq |k| \leq 1$ 。因此，每次在 x 方向上加 1， y 方向上加 1 或不变需要判断。



- 当 M 在 Q 的下方，即取 P_u 为下一个像素点。否则取 P_d 。

- 算法推导过程

- 将 M 带入直线方程， $F(x_m, y_m) = Ax_m + By_m + C$
 - $d_i = F(x_m, y_m) = F(x_i + 1, y_i + 0.5) = A(x_i + 1) + B(y_i + 0.5) + C$
 - 当 $d_i < 0$ 时， M 在 Q 下方，取 P_u 。
 - 当 $d_i > 0$ 时， M 在 Q 上方，取 P_d 。
 - 当 $d_i = 0$ 时， M 在直线上，取 P_d 或 P_u 均可。

- $y = \begin{cases} y + 1 & (d < 0) \\ y & (d \geq 0) \end{cases}, d_i = A(x_i + 1) + B(y_i + 0.5) + C$

- 算法递推过程

- 判断下一个点的位置，用增量来表示直线方程
- $d_{new} = \begin{cases} d_{old} + A + B & (d < 0) \\ d_{old} + A & (d \geq 0) \end{cases}, d_0 = A + 0.5 * B$
- 可以将 d_0 改为 $2d_0$ ，避免浮点数运算。

- 总结

- 中点画线法将算法提高到整数加法，优于 DDA 算法。

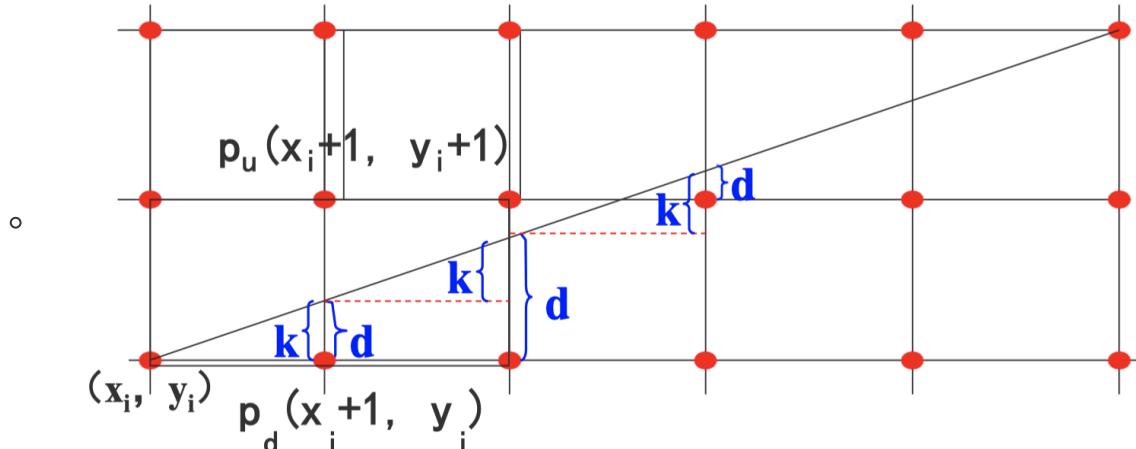
2.1.4 Bresenham 算法

- 算法特点

- 提供了一个更一般的算法。该算法不仅有好的效率，而且有更广泛的适用范围。

- Bresenham 算法基本思想

- 算法思想是通过各行、各列像素中心构造一组虚拟网格线，按照直线起点到终点的顺序，计算直线与各垂直网格线的交点，然后根据误差项的符号确定该列像素中与此交点最近的像素。



- 假设每次 $x + 1, y$ 的递增(减)量为 0 或 1，它取决于实际直线与最近光栅网格点的距离，这个距离的最大误差为 0.5。

- 误差项 d 初值 $d_0 = 0, d = d + k$ 。一旦 $d \geq 1$ ，就把它减去 1，保证 d 的相对性，且在 0、1 之间。

- $\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = \begin{cases} y_i + 1 & (d > 0.5) \\ y_i & (d \leq 0.5) \end{cases} \end{cases}$

- 改进 1：令 $e = d - 0.5$

- $\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = \begin{cases} y_i + 1 & (e > 0) \\ y_i & (e \leq 0) \end{cases} \end{cases}, e_0 = -0.5$, 每走一步有 $e = e + k$ 。
◦ if ($e > 0$) then $e = e - 1, k = \frac{dy}{dx}$

- 改进 2：用 $e * 2 * \Delta x$ 来替换 e

- $e_0 = -\Delta x$, 每走一步有 $e = e + 2 * \Delta y$ 。
◦ if ($e > 0$) then $e = e - 2\Delta x$

- 算法步骤

- ① 输入直线的两端点 $P_0(x_0, y_0)$ 和 $P_1(x_1, y_1)$ 。
◦ ② 计算初始值 $\Delta x, \Delta y, e = -\Delta x, x = x_0, y = y_0$ 。
◦ ③ 绘制点 (x, y) 。
◦ ④ e 更新为 $e + 2\Delta y$ ，判断 e 的符号。若 $e > 0$ ，则 (x, y) 更新为 $(x + 1, y + 1)$ ，同时将 e 更新为

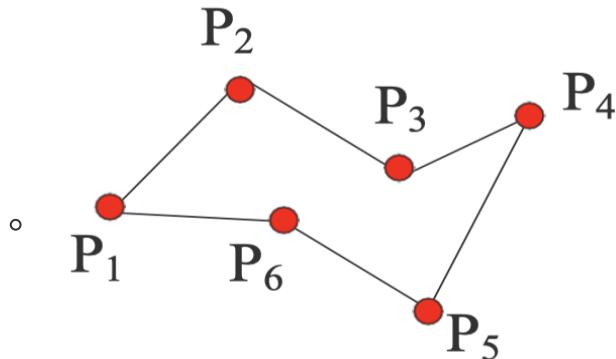
$e - 2\Delta x$ 。否则 (x, y) 更新为 $(x + 1, y)$ 。

- ⑤ 当直线没有画完时，重复步骤③、④。否则结束。
-

2.2 多边形的扫描转换

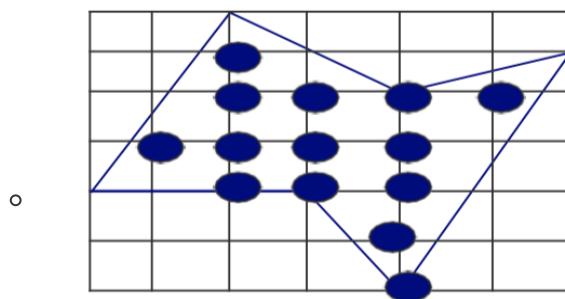
2.2.1 多边形表示方法

- 顶点表示
 - 定义：顶点表示是用多边形的顶点序列来表示多边形。这种表示直观、几何意义强、占内存少，易于进行几何变换。
 - 缺点：由于没有明确指出哪些像素在多边形内，故不能直接用于面着色。



顶点表示

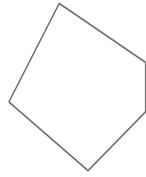
- 点阵表示
 - 定义：点阵表示是用位于多边形内的像素集合来刻画多边形。这种表示丢失了许多几何信息（如边界、顶点等），但它却是光栅显示系统显示时所需的表示形式。



点阵表示

- 多边形的扫描转换
 - 把多边形的顶点表示转换为点阵表示，即为多边形的扫描转换。
- 多边形分类
 - 凸多边形

- 任意两顶点间的连接均在多边形内



- 凹多边形

- 任意两顶点间的连线有不在多边形内的



- 含内环的多边形

- 多边形内包含多边形



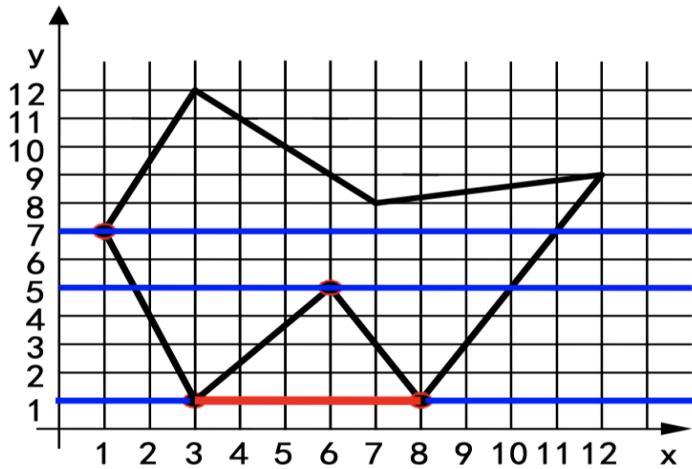
2.2.2 X-扫描线算法

- 基本思想

- X-扫描线算法填充多边形的基本思想是按扫描线顺序，计算扫描线与多边形的相交区间，再用要求的颜色显示这些区间的像素，即完成填充工作。
- 区间的端点可以通过计算扫描线与多边形边界线的交点获得。

- 算法过程

- 算法核心：按 X 递增顺序排列交点的 X 坐标序列。
 - ① 确定多边形所占有的最大扫描线数，得到多边形顶点的最小和最大 y 值 (y_{min} 和 y_{max})
 - ② 从 $y = y_{min}$ 到 $y = y_{max}$ ，每次用一条扫描线进行填充。
 - ③ 对一条扫描线填充的过程分为以下四步：
 - a. 求交：计算扫描线与多边形各边的交点
 - b. 排序：把所有交点按递增顺序进行排序
 - c. 交点配对：第一个与第二个，第三个与第四个
 - d. 区间填色：把这些相交区间内的像素置成不同于背景色的填充色
 - 扫描线与多边形顶点相交时，交点的取舍问题（交点的个数应保证为偶数个）
 - a. 若共享顶点的两条边分别落在扫描线的两边，交点只算 1 个
 - b. 若共享顶点的两条边在扫描线的同一边，交点个数为 0 个或 2 个。需要检查两条边另外两个端点的 y 值来判断。
 -



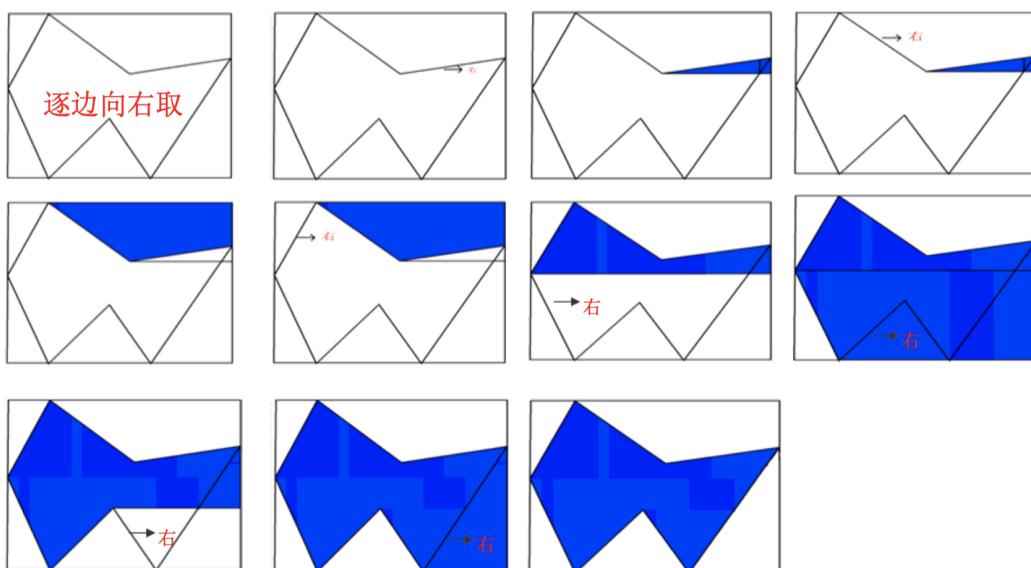
- 两个重要思想
 - 扫描线：当处理图形图像时按一条条扫描线处理
 - 增量的思想
- 扫描线中的数据结构
 - 活性边表 (AET)
 - AET：把与当前扫描线相交的边称为活性边，并把它们按与扫描线交点 x 坐标递增的顺序存放在一个链表中。
 - 节点内容
 - x : 当前扫描线与边的交点坐标
 - Δx : 从当前扫描线到下一条扫描线间 x 的增量， $\Delta x = \frac{1}{k}$
 - y_{max} : 该边所交的最高扫描线的坐标值 y_{max} ，可以判断何时 "抛弃" 该边
 - | | | | |
|-----|------------|-----------|------|
| x | Δx | y_{max} | next |
|-----|------------|-----------|------|
 - 新边表 (NET)
 - 首先构造一个纵向链表，链表的长度为多边形所占有的最大扫描线数，链表的每个节点，称为一个吊桶，对应多边形覆盖的每一条扫描线。
 - NET挂在与该边低端 y 值相同的扫描线桶中，即存放在该扫描线第一次出现的边。
 - NET 节点内容
 - y_{max} : 该边的 y_{max}
 - x_{min} : 该边较低点的 x 坐标值 x_{min}
 - $\frac{1}{k}$: 该边的斜率 $\frac{1}{k}$
 - | | | | |
|-----------|-----------|-------|------|
| y_{max} | x_{min} | $1/k$ | next |
|-----------|-----------|-------|------|
 - 每做一次新的扫描线时，要对已有的边进行三个处理：
 - 是否被去除掉
 - 如果不被去除，第二就要对它的数据进行更新。所谓更新数据就是要更新它的 x 值，即：

$$x = x + \frac{1}{k}$$
 - 看有没有新的边进来，新的边在 NET 里，可以插入排序插进来。
 - 优点：避免求交运算。
 - 小结

- 扫描线法可以实现已知多边形域边界的填充。该填充算法是按扫描线的顺序，计算扫描线与待填充区域的相交区间，再用要求的颜色显示这些区间的像素，即完成填充工作。
- 提高算法效率的方法
 - 增量的思想
 - 连贯性思想
 - 构建了一套特殊的数据结构
- 缺点：带填充区域的边界线必须事先知道，因此它的缺点是无法实现对未知边界的区域填充。

2.2.3 边缘填充算法

- 基本思想
 - 按任意顺序处理多边形的每条边。在处理每条边时，首先求出该边与扫描线的交点，然后将每一条扫描线上交点右方的所有像素取补。多边形的所有边处理完毕之后，填充即完成。
- 算法过程
 - 对于每条边，将其上下 y_{max} 、 y_{min} 所构成的矩形区域划分开，取矩形区域的右半部分像素取补。
 -



- 特点
 - 算法简单，但对于复杂图形，每个像素可能被访问多次。输入和输出量比有效边算法大得多。

2.2.4 栅栏填充算法

- 算法原理
 - 栅栏指的是一条过多边形顶点且与扫描线垂直的直线。它把多边形分成两半。在处理每条边与扫描线的交点时，将交点与栅栏之间的像素取补。

2.2.5 边界标志算法

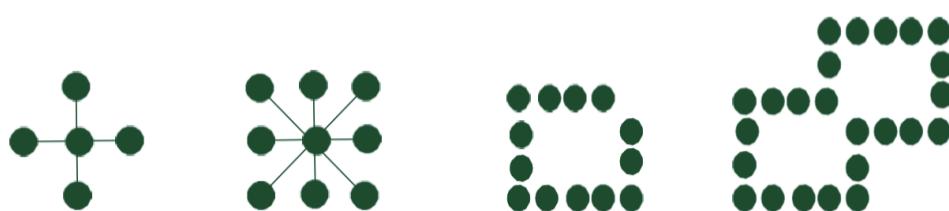
- 算法原理
 - 帧缓冲器中对多边形的每条边进行直线扫描转换，亦即对多边形边界所经过的像素打上标志。
 - 然后再采用和扫描线算法类似的方法将位于多边形内的各个区段着上所需颜色。
 - 由于边界标志算法不必建立维护边表以及对它进行排序，所以边界标志算法更适合硬件实现，这时它的

执行速度比有序边表算法快一至两个数量级。

2.3 区域填充

2.3.1 区域填充基础概念

- 区域
 - 指已经表示成点阵形式的填充图形，是像素的集合。
- 区域填充
 - 指将区域内的一点(常称种子点)赋予给定颜色，然后将这种颜色扩展到整个区域内的过程。
- 区域可采用内点表示和边界表示两种表示形式。
 - 内点表示
 - 枚举出区域内部的所有像素，内部的所有像素着同一个颜色，边界像素着与内部像素不同的颜色。
 - 边界表示
 - 枚举出边界上的所有像素，边界上的所有像素着同一个颜色，内部像素着与边界像素不同的颜色。
- 连通区域
 - 区域填充算法要求区域是连通的，因为只有在连通区域中，才可能将种子点的颜色扩展到区域内的其它点。
 - 4向连通区域
 - 从区域上一点出发，可通过四个方向，即上、下、左、右移动的组合，在不越出区域的前提下，到达区域内的任意像素。
 - 8向连通区域
 - 从区域上一点出发，可通过八个方向，即上、下、左、右、左上、右上、左下、右下这八个方向的移动的组合来到达。
 -



四个方向运动 八个方向运动 四连通区域 八连通区域

2.3.2 简单四连通种子填充算法

- 种子填充算法原理
 - 假设在多边形区域内部有一像素已知，由此出发找到区域内的所有像素，用一定的颜色或灰度来填充。
 - 假设区域采用边界定义，即区域边界上所有像素均具有某个特定值，区域内部所有像素均不取这一特定值，而边界外像素则可具有与边界相同的值。
- 种子填充算法实现

- dfs 遍历区域，或者 bfs 遍历，推荐使用 bfs 。

2.3.3 区域填充与多边形扫描转化对比

- 基本思想不同
 - 多边形扫描转换是指将多边形的顶点表示转化为点阵表示
 - 区域填充只改变区域的填充颜色，不改变区域表示方法
 - 基本条件不同
 - 在区域填充算法中，要求给定区域内一点作为种子点，然后从这一点根据连通性将新的颜色扩散到整个区域。
 - 扫描转换多边形是从多边形的边界(顶点)信息出发，利用多种形式的连贯性进行填充的。
-

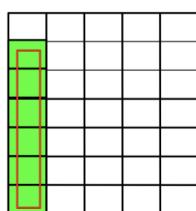
2.4 反走样

2.4.1 走样现象

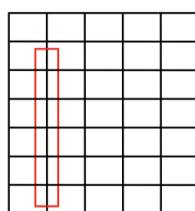
- 走样定义
 - "锯齿"是"走样"的一种形式。而走样是光栅显示的一种固有性质。产生走样现象的原因是像素本质上是离散性的。
- 走样现象
 - ① 光栅图形产生的阶梯形(锯齿形)



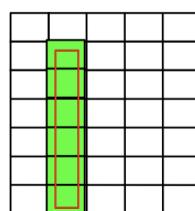
- ② 小物体由于"走样"而消失。图形中包含相对微小的物体时，这些物体在静态图形中容易被丢弃或忽略。
 - 动画序列中时隐时现，产生闪烁。
 - 矩形从左向右移动，当其覆盖某些像素中心时，矩形被显示出来，当没有覆盖像素中心时，矩形不被显示。



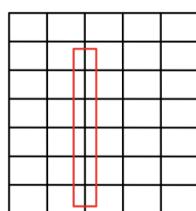
(a) 显示



(b) 不显示



(c) 显示



(d) 不显示

- 简单地说，如果对一个快速变化的信号采样频率过低，所得样本表示的会是低频变化的信号：原始信号的频率看起来被较低的"走样"频率所代替。

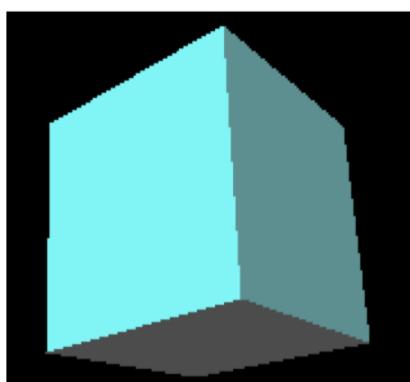
2.4.2 反走样技术

- 反走样定义
 - 反走样技术，即减少或消除走样效果的技术。
 - 采用分辨率更高的显示设备，对解决走样现象有所帮助，因为可以使锯齿相对物体更小一些。
- 非加权区域采样方法
 - 原理
 - 根据物体的覆盖率计算像素的颜色。覆盖率是指某个像素区域被物体覆盖的比例。
 - 方法
 - 被多边形覆盖了一半的像素的亮度赋为 $\frac{1}{2}$ ，覆盖三分之一的像素亮度赋值为 $\frac{1}{3}$ ，以此类推。
 - 缺点
 - 像素的亮度与相交区域的面积成正比，而与相交区域落在像素内的位置无关，这仍然会导致锯齿效应。
 - 直线条上沿理想直线方向的相邻两个像素有时会有较大的灰度差。
- 加权区域采样方法
 - 原理
 - 将直线段看作是具有一定宽度的狭长矩形。当直线段与像素有交时，根据相交区域与像素中心距离来决定其对像素亮度的贡献。
 - 直线段对一个像素的亮度的贡献正比于相交区域与像素中心的距离。设置相交区域面积与像素中心距离的权函数(高斯函数)反映相交面积对整个像素亮度的贡献大小。
 - 离散计算方法
 - 将一个像素划分为 $n = 3 * 3$ 个子像素，加权表可以取为：
$$\begin{bmatrix} w1 & w2 & w3 \\ w4 & w5 & w6 \\ w7 & w8 & w9 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

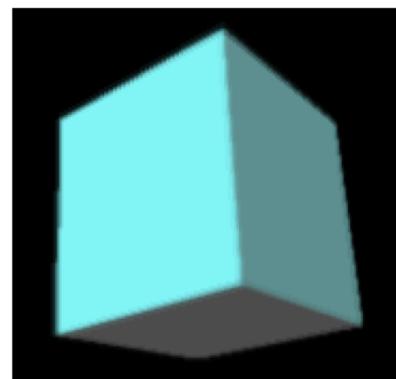
- 加权方案：中心子像素的加权是角子像素的 4 倍，是其他像素的 2 倍。对九个子像素的每个网格所计算出的亮度进行平均，然后求出所有中心落于直线段内的子像素。最后计算所有这些子像素对原像素亮度贡献之和。

- 反走样是图形学中的一个根本问题，不可能避免；是图形学中的一个永恒问题。

-



原 图

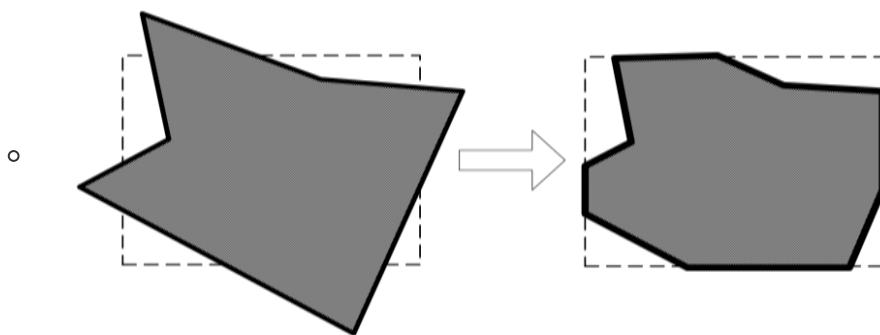


反走样图

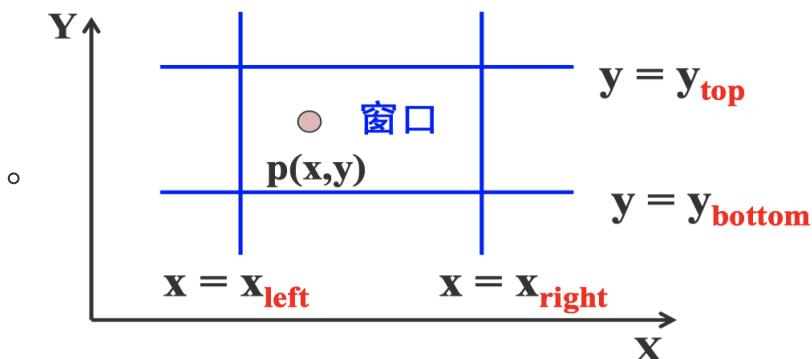
第三章 裁剪

3.1 裁剪概述

- 裁剪
 - 计算机内部存储的图形往往比较大，而屏幕显示的只是图形的一部分。
 - 因此需要确定图形哪些部分落在显示区之内，哪些落在显示区之外。这个选择的过程就称为裁剪。
 - 最简单的裁剪方法是把各种图形扫描转换为点之后，再判断点是否在窗口内。



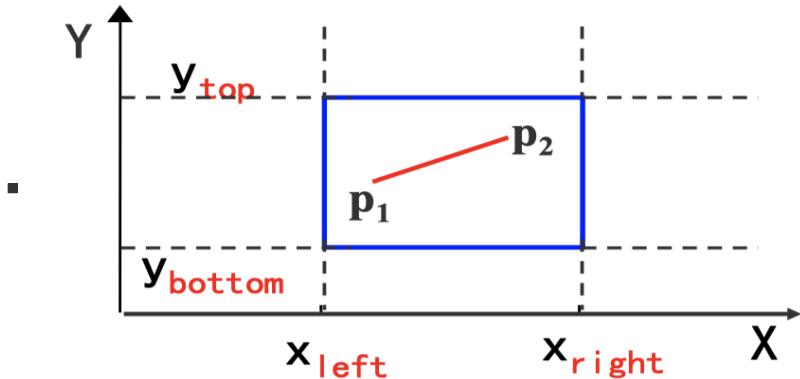
- 点的裁剪
 - 对于任意一点 $P(x, y)$ ，若满足 $x_{left} \leq x \leq x_{right}$ 且 $y_{bottom} \leq y \leq y_{top}$ ，则点 P 在矩形窗口内，否则点 P 在矩形窗口外。



- 缺点：太费时，不可取。
- 直线段的裁剪
 - 直线段裁剪算法是复杂图形裁剪的基础。

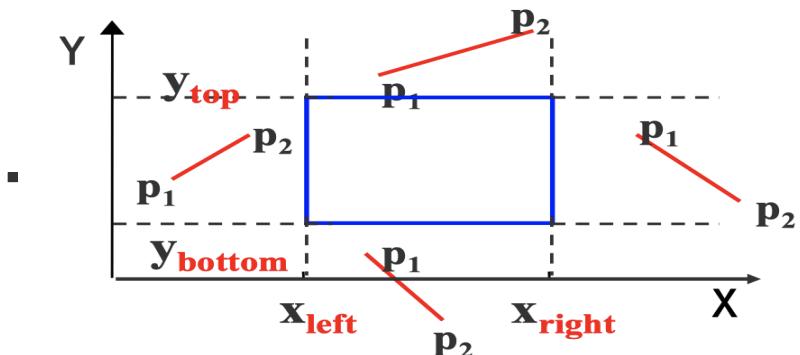
3.2 Cohen-Sutherland 算法

- 算法概述
 - 此算法又称为**编码裁剪方法**，算法的基本思想是对每条直线段分三种情况处理。
- 算法原理
 - ① 点 p_1 和 p_2 完全在裁剪窗口内，“简取”之。



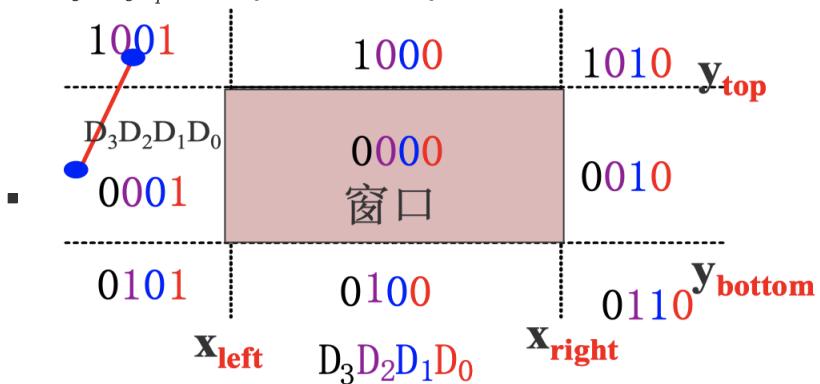
- ② 若点 $p_1(x_1, y_1)$ 和 $p_2(x_2, y_2)$ 均在窗口外，且满足下列四个条件之一，“简弃”之。

- $x_1 < x_{left}$ 且 $x_2 < x_{left}$
- $x_1 > x_{right}$ 且 $x_2 > x_{right}$
- $y_1 < y_{bottom}$ 且 $y_2 < y_{bottom}$
- $y_1 > y_{top}$ 且 $y_2 > y_{top}$

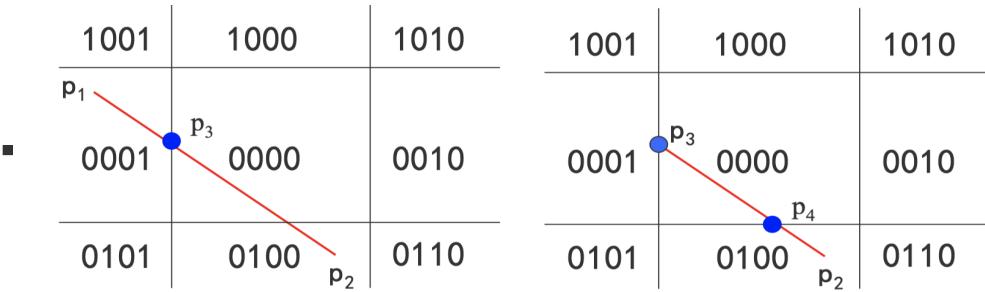


- ③ 其它情况需要对直线段按交点进行分段，分段后判断是“简取”还是“简弃”。

- 每条线段的端点都赋以四位二进制码 $D_3 D_2 D_1 D_0$ ，编码规则如下
- 若 $x < x_{left}$, 则 $D_0 = 1$, 否则 $D_0 = 0$
- 若 $x > x_{right}$, 则 $D_1 = 1$, 否则 $D_1 = 0$
- 若 $y < y_{bottom}$, 则 $D_2 = 1$, 否则 $D_2 = 0$
- 若 $y > y_{top}$, 则 $D_3 = 1$, 否则 $D_3 = 0$



- 裁剪一条线段时，先求出端点 p_1 和 p_2 的编码 $code_1$ 和 $code_2$ ，然后进行二进制“或”运算和“与”运算。若 $code_1 | code_2 = 0$ ，简弃之；若 $code_1 \& code_2 \neq 0$ ，简取之。
- 若上述两条件均不成立，则需求出直线段与窗口边界的交点，并在交点处把线段一分为二。
- 如下所示，先求出交点 p_4 ，舍弃 $p_1 p_3$ 之后，再次求出交点 p_4 ，舍弃 $p_4 p_2$ ，得到 $p_3 p_4$ 。



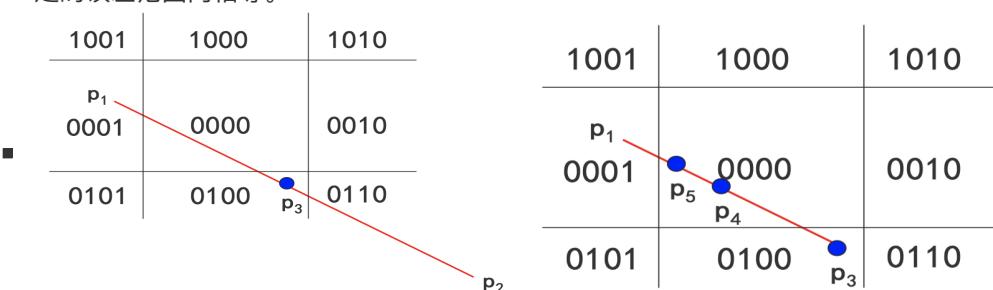
- 总结

- Cohen-Sutherland 算法体现了**编码思想**, 意义重大。
 - 比较适合两种情况: 大部分线段完全可见、大部分线段完全不可见。
-

3.2 中点分割算法

- 算法概述

- 首先将直线段的端点进行编码, 将线段和窗口的关系分成三种情况:
 - 完全在窗口内
 - 完全在窗口外
 - 和窗口有交点
- 核心思想: 通过二分逼近来确定直线段与窗口的交点。
- 注意
 - 若中点不在窗口内, 则把中点和离窗口边界最远点构成的线段丢掉, 以线段上的另一点和该中点再构成线段求其中点。
 - 如中点在窗口内, 则又以中点和最远点构成线段, 并求其中点, 直到中点与窗口边界的坐标值在规定的误差范围内相等。

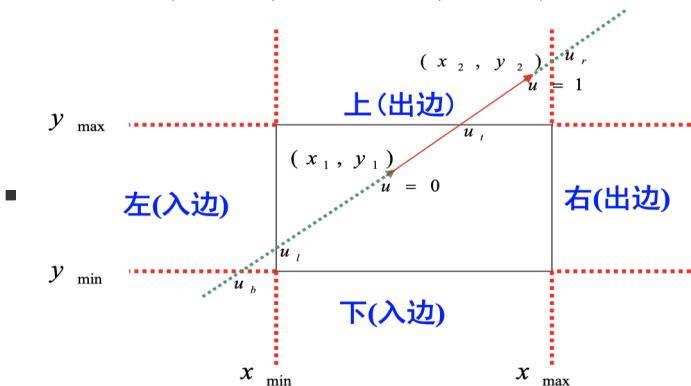


3.3 Liang-Barsky 算法

- 算法过程

- 用参数方程表示一条直线, $x = x_1 + u * (x_2 - x_1) = x_1 + \Delta x * u$,
 $y = y_1 + u * (y_2 - y_1) = y_1 + \Delta y * u$, $0 \leq u \leq 1$ 。
- 把被裁剪的红色直线段看成是一条有方向的线段, 把窗口的四条边分成两类: 入边和出边
 - 裁剪结果的线段起点是直线和两条入边的交点以及始端点三个点里最前面的一个点, 即参数 u 最大的那个点。

- 裁剪线段的终点是和两条出边的交点以及端点最后面的一个点，取参数 u 最小的那个点。
- 当 u 从 $-inf \sim inf$ 遍历直线时，首先对裁剪窗口的两条边界直线（下边和左边）从外面向里面移动，再对裁剪窗口两条边界直线（上边和右边）从里面向外面移动。
- 用 u_1 和 u_2 分别表示线段 ($u_1 \leq u_2$) 可见部分的开始和结束
 - $u_1 = \max(0, u_l, u_b), u_2 = \min(1, u_t, u_r)$

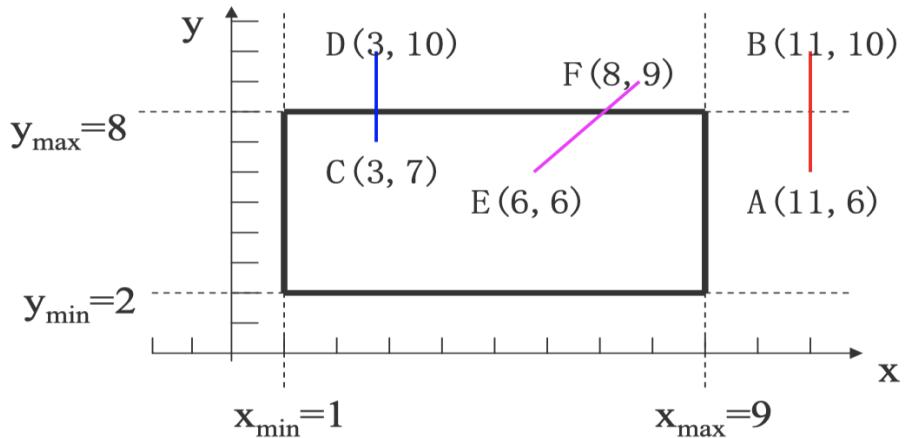


● 具体步骤

- $p_1 = -\Delta x, p_2 = \Delta x, p_3 = -\Delta y, p_4 = \Delta y$
- $q_1 = x_1 - x_{left}, q_2 = x_{right} - x_1, q_3 = y_1 - y_{bottom}, q_4 = y_{top} - y_1$
- ① **输入**直线段的两端点坐标 (x_1, y_1) 、 (x_2, y_2) ，以及窗口的四条边界坐标： wxl 、 wxr 、 wyb 、 wyt
 -
- ② **若 $\Delta x = 0$** ，则 $p_1 = p_2 = 0$ ，此时进一步判断是否满足 $q_1 < 0$ 或 $q_2 < 0$ ，若满足，则该直线段不在窗口内，算法转 ⑦ 结束。否则，满足 $q_1 \geq 0$ 且 $q_2 \geq 0$ ，则进一步计算 u_{max} 和 u_{min} 。
 - $u_{max} = \max(0, u_k | p_k < 0), u_{min} = \min(1, u_k | p_k > 0)$ 。
 - 其中 $u_k = \frac{q_k}{p_k}$, ($p_k \neq 0, k = 3, 4$)。算法转 ⑤。
- ③ **若 $\Delta y = 0$** ，则 $p_3 = p_4 = 0$ ，此时进一步判断是否满足 $q_3 < 0$ 或 $q_4 < 0$ ，若满足，则该直线段不在窗口内，算法转 ⑦ 结束。否则，满足 $q_3 \geq 0$ 且 $q_4 \geq 0$ ，则进一步计算 u_{max} 和 u_{min} 。
 - $u_{max} = \max(0, u_k | p_k < 0), u_{min} = \min(1, u_k | p_k > 0)$ 。
 - 其中 $u_k = \frac{q_k}{p_k}$, ($p_k \neq 0, k = 1, 2$)。算法转 ⑤。
- ④ **若上述两条均不满足**，则有 $p_k \neq 0$ ($k = 1, 2, 3, 4$)，此时计算 u_{max} 和 u_{min} 。
 - $u_{max} = \max(0, u_k | p_k < 0), u_{min} = \min(1, u_k | p_k > 0)$ 。
 - 其中 $u_k = \frac{q_k}{p_k}$, ($p_k \neq 0, k = 1, 2, 3, 4$)。
- ⑤ **求得 u_{max} 和 u_{min} 后，进行判断：**
 - 若 $u_{max} > u_{min}$ ，则直线段在窗口外，算法转 ⑦
 - 若 $u_{max} \leq u_{min}$ ，利用直线参数方程求出裁剪后的直线端点。
 - $x = x_1 + u * (x_2 - x_1), y = y_1 + u * (y_2 - y_1)$
- ⑥ 绘制直线段。
- ⑦ 算法结束。

● 举例

-



- 直线 AB

- $p_1 = 0, q_1 = 10; p_2 = 0, q_2 = -2; p_3 = -4, q_3 = 4; p_4 = 4, q_4 = 2$
- AB 完全在右边界之右

- 直线 CD

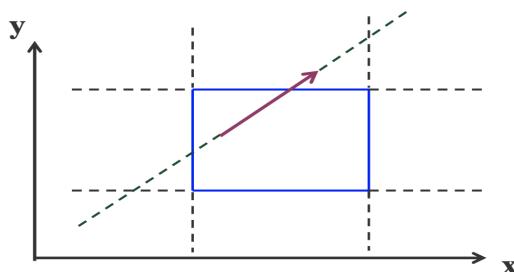
- $p_1 = 0, q_1 = 2; p_2 = 0, q_2 = 6; p_3 = -3, q_3 = 5; p_4 = 3, q_4 = 1$
- $u_3 = -\frac{5}{3}, u_4 = \frac{1}{3}$
- $u_{max} = \max(0, u_3) = 0, u_{min} = \min(1, u_4) = \frac{1}{3}$
- 裁剪后两个端点为 $(3, 7)$ 和 $(3, 8)$

- 直线 EF

- $p_1 = -2, q_1 = 5; p_2 = 2, q_2 = 3; p_3 = -3, q_3 = 4; p_4 = 3, q_4 = 2$
- $u_1 = -\frac{5}{2}, u_2 = \frac{3}{2}, u_3 = -\frac{4}{3}, u_4 = \frac{2}{3}$
- $u_{max} = \max(0, u_1, u_3) = 0, u_{min} = \min(1, u_2, u_4) = \frac{2}{3}$
- 裁剪后两个端点为 $(6, 6)$ 和 $(7.33, 8)$

- 总结

- 直线段看成是有方向的



- 直线参数化

- $x = x_1 + \Delta x * u, y = y_1 + \Delta y * u, 0 \leq u \leq 1$ 。
- Liang-Barsky 裁剪算法和 Cohen-Sutherland 算法均只能应用于矩形窗口。

3.4 多边形、字符裁剪

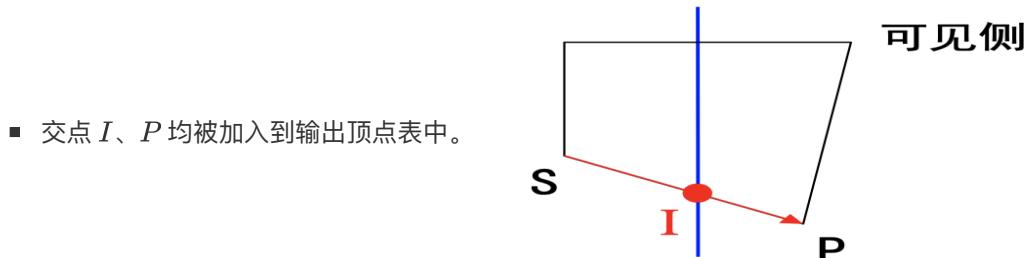
3.4.1 多边形的裁剪

- Sutherland-Hodgeman 多边形裁剪
 - 基本思想

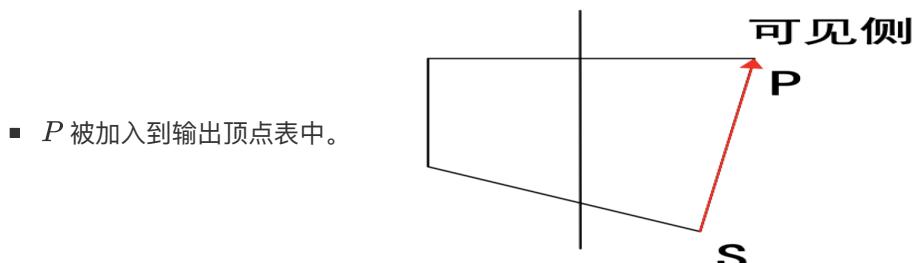
- 将多边形边界作为一个整体，每次用窗口的一条边对要裁剪的多边形和中间结果多边形进行裁剪，体现一种分而治之的思想。
- 包含有窗口区域的一个域称为可见侧，不包含窗口区域的域为不可见侧。

- 算法具体过程

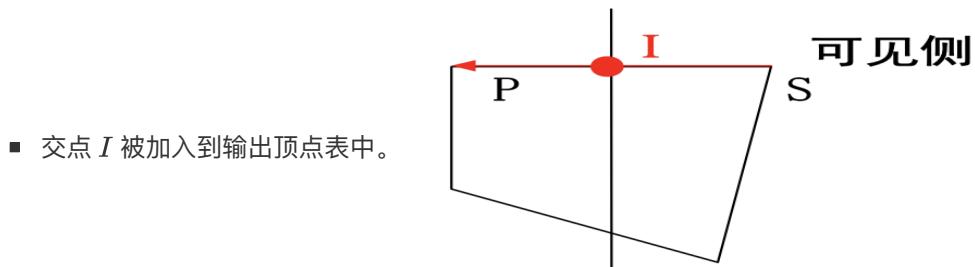
- 第一点 S 在不可见侧面，而第二点 P 在可见侧。



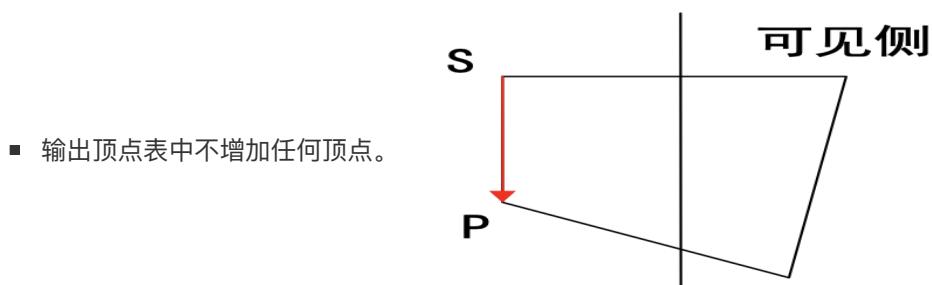
- 第一、二点 S 和 P 都在可见侧。



- S 在可见侧，而 P 在不可见侧。

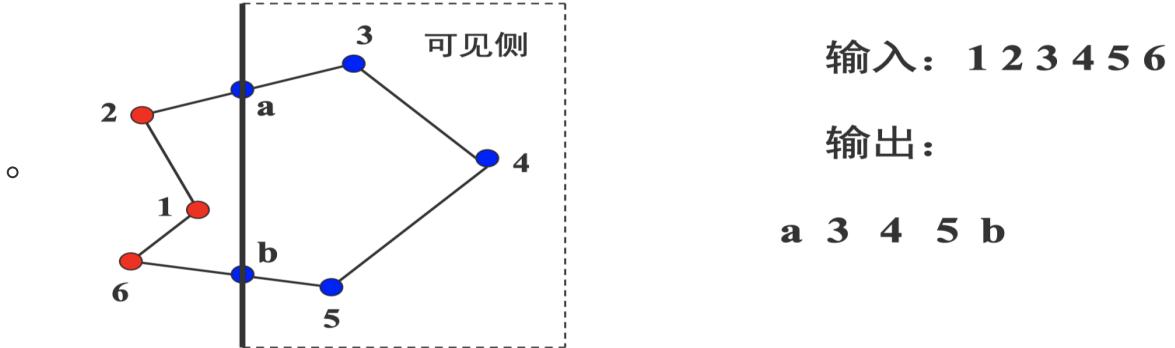


- S 和 P 都在不可见侧。



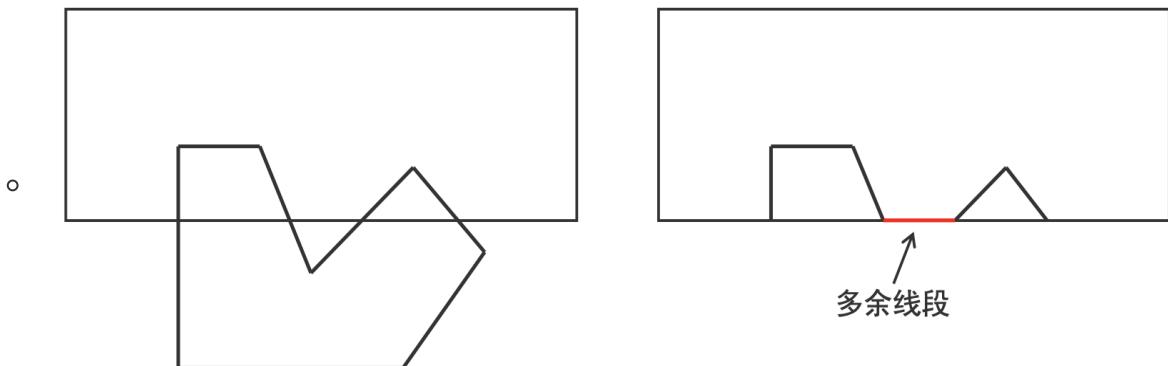
- 在窗口在一条裁剪边界处理完所有顶点后，其输出顶点表将用窗口的下一条边界继续裁剪。

- 举例



- 缺点

- 可能会有多余线段产生。



3.4.2 文字裁剪

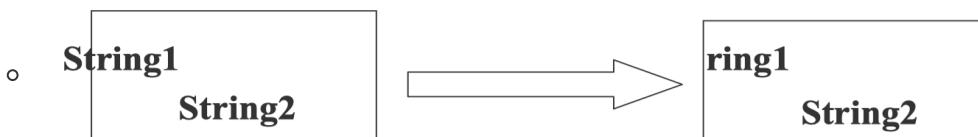
- 串精度裁剪

- 当字符串中的所有字符都在裁剪窗口内时，就全部保留它，否则舍弃整个字符串。



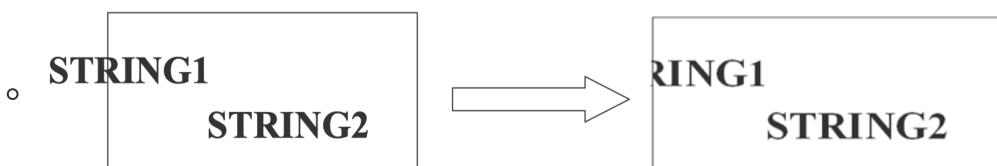
- 字符精度裁剪

- 任何与窗口有重叠或落在窗口边界以外的字符都被裁减掉。



- 笔划/像素精度裁剪

- 将笔划分解成直线段对窗口作裁剪。需判断字符串中各字符的哪些像素、笔划的哪一部分在窗口内，保留窗口内部分，裁剪掉窗口外的部分。



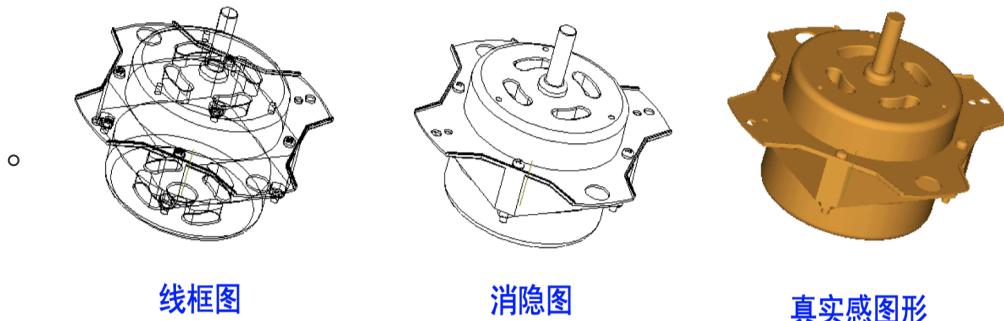
第四章 消隐

4.1 消隐概述

4.1.1 消隐定义

- 消隐

- 如果把可见和不可见的线都画出来，会对视觉造成多义性。
- 要消除二义性，就必须在绘制时消除被遮挡的不可见的线或面，即消除隐藏线和隐藏面，简称消隐。



4.1.2 消隐分类

- 按消隐对象分类

- 线消隐
 - 消隐对象是物体上的边，消除的是物体上不可见的边。
- 面消隐
 - 消隐对象是物体上的面，消除的是物体上不可见的面，通常做真实感图形消隐时用面消隐。

- 按消隐空间分类

- 物体空间的消隐算法
 - 以场景中的物体为处理单元。假设场景中有 k 个物体，将其中一个物体与其余 $k-1$ 个物体逐一比较，仅显示它可见表明以达到消隐的目的。
 - 此类算法常用于线框图的消隐。
 - Roberts 算法、光线投射法
- 图像空间的消隐算法
 - 消隐算法的主流。
 - Z-buffer 算法
 - 扫描线算法
 - Warnock 消隐算法

4.2 物体空间的消隐算法

4.2.1 Roberts 算法

- 算法步骤

- 逐个的独立考虑每个物体自身，找出为其自身所遮挡的边和面（自消隐）
- 将每一物体上留下的边再与其它物体逐个的进行比较，以确定是完全可见还是部分或全部遮挡（两两物体消隐）
- 确定由于物体之间的相互贯穿等原因，是否要形成新的显示边等，从而使被显示各物体更接近现实。

- 算法特点

- 数学处理严谨，计算量甚大。算法要求所有被显示的物体都是凸的，对于凹体要先分割成多个凸体的组合。

4.2.2 光线投射法

- 定义

- 光线投射是求光线与场景的交点，该光线就是所谓的视线（如视点与像素连成的线）

- 特点

- 一条视线与场景中的物体可能有许多交点，求出这些交点后需要排序，在前面的才能被看到。人的眼睛可以一目了然，但计算机做需要大量的运算。

4.3 Z 缓冲区算法 (Z-Buffer)

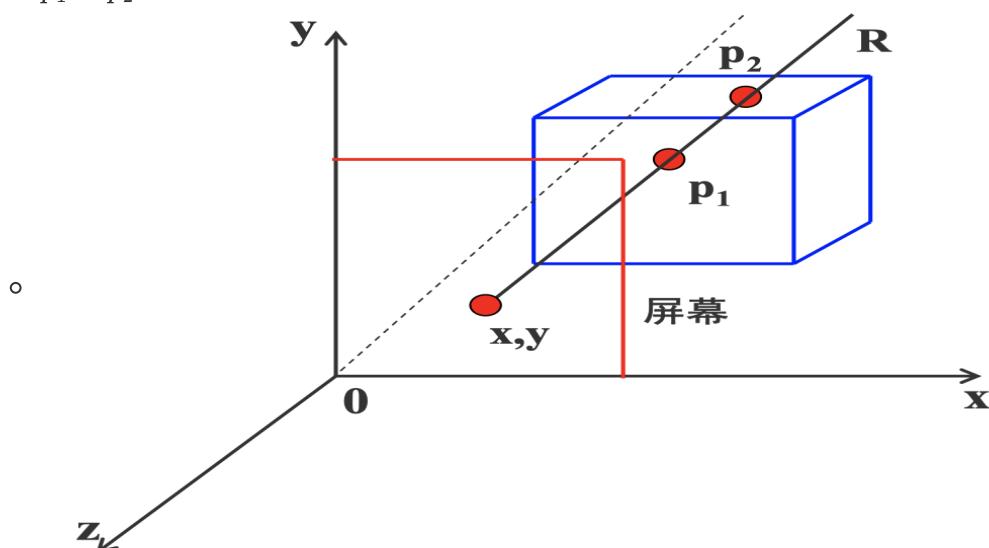
4.3.1 Z-Buffer 算法概述

- 算法介绍

- 该算法由 Edwin Catmull 独立开发，能够跟踪屏幕上每个像素深度的算法。
- 该算法有帧缓冲器和深度缓冲器。
 - $intensity(x, y)$ —— 属性数组（帧缓冲器），存储图像空间每个可见像素的光强或颜色
 - $depth(x, y)$ —— 深度数组（z-buffer），存放图像空间每个可见像素的 z 坐标

- 深度

- 以 xoy 面为投影面， z 轴为观察方向。
- 过屏幕上任意像素点 (x, y) 作平行于 z 轴的射线 R ，与物体表面相交于 p_1 和 p_2 点。
- p_1 和 p_2 点的 z 值称为该点的深度值。



- 算法思想

- 先将 Z 缓冲器中各单元的初始值置为最小值。当要改变某个像素的颜色值时，首先检查当前多边形的深度值是否大于该像素原来的深度值(保存在该像素所对应的 Z 缓冲器的单元中)
- 如果大于原来的 Z 值，说明当前多边形更靠近观察点，用它的颜色替换像素用来的颜色。

```
1 Z-Buffer算法 () {  
2     帧缓存全置为背景色，深度缓存全置为最小z值；  
3     for(每一个多边形) {  
4         扫描转换该多边形；  
5         for( 该多边形所覆盖的每个象素(x,y) ) {  
6             计算该多边形在该象素的深度值z(x,y)；  
7             if( z(x,y) 大于 z缓存(x,y) 的值 ) {  
8                 把z(x,y)存入z缓存中(x,y)处；  
9                 把多边形在(x,y)处的颜色值存入帧缓存的(x,y)处；  
10            }  
11        }  
12    }  
13 }
```

- Z-Buffer 算法的优点

- Z-Buffer 算法比较简单，也很直观。
- 在像素级上以近物取代远物。与物体在屏幕上的出现顺序是无关紧要的，有利于硬件实现。

- Z-Buffer 算法的缺点

- 占用空间大
- 没有利用图形的相关性与连续性(严重缺陷)
- 该算法是在像素级上的消隐算法(很大缺陷)

4.3.2 Z-Buffer 算法改进

- 算法改进

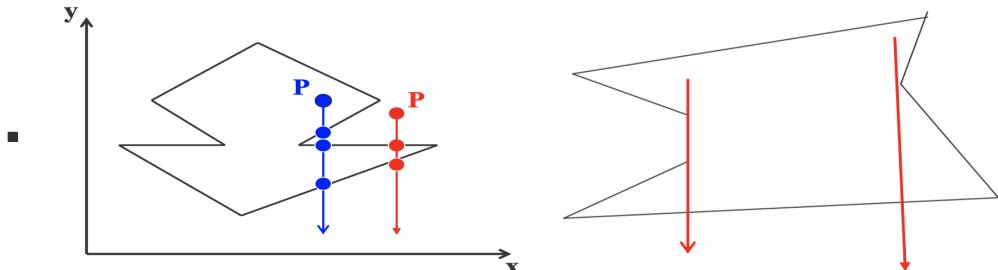
- 只用一个深度缓存变量 zb 代替与图像大小相等的缓存数组，节省了空间。

```
1 z-Buffer算法 () {  
2     帧缓存全置为背景色  
3     for( 屏幕上的每个象素(i,j) ) {  
4         深度缓存变量zb置最小值 minValue;  
5         for(多面体上的每个多边形Pk) {  
6             if(象素点(i,j)在pk的投影多边形之内) { //如何查看一点是否在多边形中  
7                 计算Pk在(i,j)处的深度值depth; //如何计算深度值  
8                 if(depth大于zb) {  
9                     zb = depth;  
10                    indexp = k; (记录多边形的序号)  
11                }  
12            }  
13        }  
14        if(zb != minValue)  
15            计算多边形 P 在交点 (i,j) 处的光照颜色并显示  
16    }  
17 }
```

- 计算 P_k 在点 (i, j) 处的深度
 - 多边形 P_k 的平面方程为 $ax + by + cz + d = 0$, $depth = -\frac{ai + bj + d}{c}$

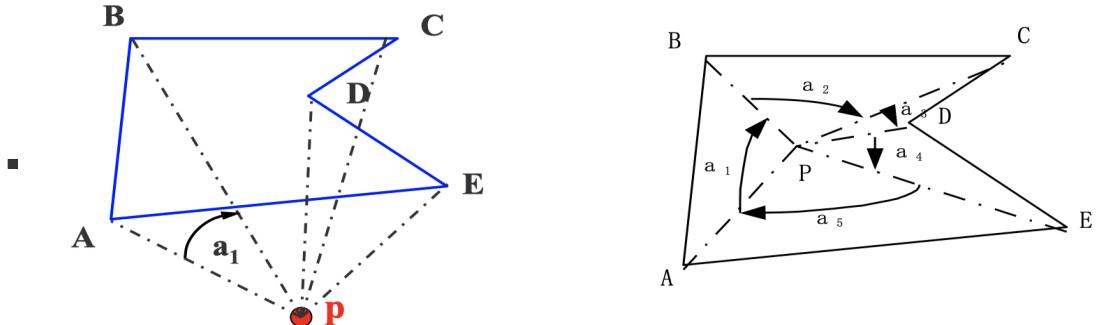
- 计算一点是否在多边形中
 - 射线法

- 核心：由被测点 P 处向 $y = -inf$ 方向作射线，交点个数是奇数，则被测点在多边形内部；交点个数是偶数表示在多边形外部。
- 若射线正好经过多边形的顶点，则采用“左开右闭”的原则来实现。即：当射线与某条边的顶点相交时，若边在射线的左侧，交点有效，计数；若边在射线的右侧，交点无效，不计数。



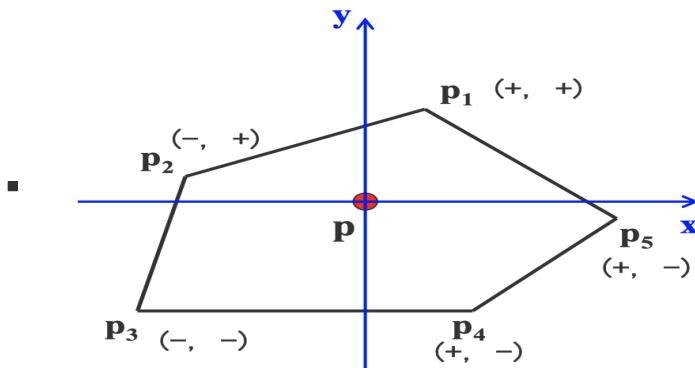
- 缺点：计算量大、不稳定（浮点误差）
- 弧长法

- 核心：以 p 点为圆心，作单位圆，把边投影到单位圆上，对应一段段弧长，规定逆时针为正，顺时针为负，计算弧长代数和。
- 代数和为 0，点在多边形外部；代数和为 2π ，点在多边形内部；代数和为 π ，点在多边形边上。



- 以顶点符号为基础的弧长累加方法

- 核心： p 是被测点，按照弧长法， p 点的代数和为 2π 。不要计算角度，做一个规定来取代原来的弧长计算。
- 在边上与内部均为 2π ，外部为 0。



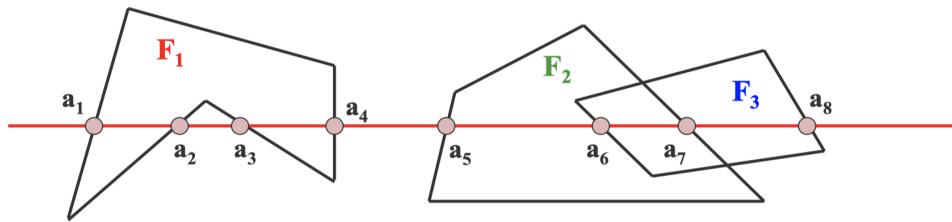
弧长变化 象限变化

(+ +)	(+ +)	0	I → I
(+ +)	(- +)	$\pi/2$	I → II
(+ +)	(- -)	$\pm\pi$	I → III
(+ +)	(+ -)	$-\pi/2$	I → IV
...

(x_i, y_i)	(x_{i+1}, y_{i+1})	弧长变化	象限变化
(+ +)	(+ +)	0	1 → 1
(+ +)	(- +)	$\pi/2$	1 → 2
(+ +)	(- -)	π	1 → 3
(+ +)	(+ -)	$-\pi/2$	1 → 4
(- +)	(+ +)	$-\pi/2$	2 → 1
(- +)	(- +)	0	2 → 2
(- +)	(- -)	$\pi/2$	2 → 3
(- +)	(+ -)	π	2 → 4
(- -)	(+ +)	$-\pi$	3 → 1
(- -)	(- +)	$-\pi/2$	3 → 2
(- -)	(- -)	0	3 → 3
(- -)	(+ -)	$\pi/2$	3 → 4
(+ -)	(+ +)	$\pi/2$	4 → 1
(+ -)	(- +)	$-\pi$	4 → 2
(+ -)	(- -)	$-\pi/2$	4 → 3
(+ -)	(+ -)	0	4 → 4

4.4 区间扫描线算法

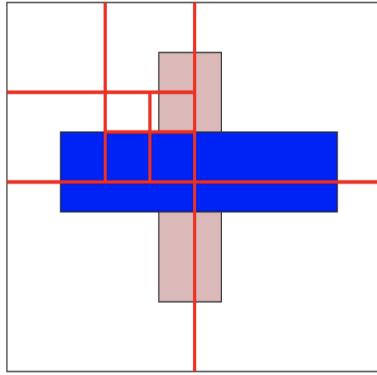
- 算法概述
 - 考虑Z-Buffer没有利用图形的相关性和连续性的缺陷，该算法放弃了Z-Buffer的思想（一个像素可能被多个多边形覆盖，即一个像素要多次判别，效率极低），是消隐算法中最快的算法之一。
- 算法过程
 - 把扫描线和多边形的这些交点都求出来，对每个区间，只判一个像素的颜色，那么整个区间都是该颜色
 - 像素计算 -> 逐段计算，效率大大提高。
 -



- 确定小区间的颜色：
 - 小区间无任何多边形，如[a4, a5]，用背景色显示
 - 小区间仅有一多边形，如[a1, a2]，显示该多边形颜色
 - 小区间存在两个以上多边形，如[a6, a7]，用深度检测
- 算法实现问题
 - 如何求交点
 - 利用增量算法简化求交
 - 每段区间上要求z值最大的面，如何得知区间与哪些多边形相关
 - 利用区间扫描转换算法中的AET与NET得知

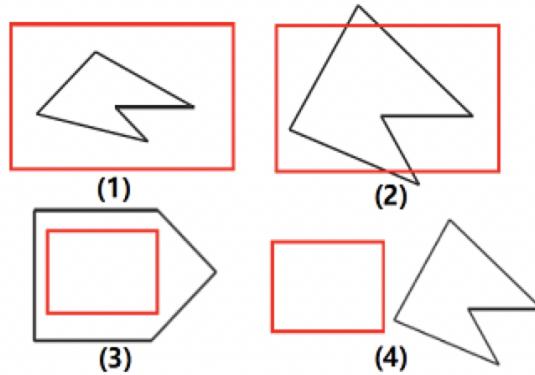
4.5 Warnock消隐算法

- 算法概述
 - 区域子分割算法，发明人 Warnock (Adobe创始人)，图像空间中非常经典的算法，其重要性不体现在其效率，而是体现在分治思想和堆栈数据结构的运用。
- 算法思想
 - 把物体投影到全屏幕窗口
 - 递归分割窗口，直到窗口内目标足够简单（可以显示）
 - 若窗口分割至像素级别，仍有两个以上的面，则不必再分割。取窗口内最近的可见面颜色或所有可见面平均颜色即可。
 -



- 如何判断窗口内图形足够简单

- 四种简单窗口图形



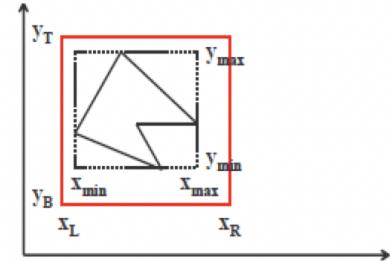
当满足下列条件时，多边形被窗口包含：

- 窗口中仅包含一个多边形

$$x_{\min} \geq x_L \quad \& \quad x_{\max} \leq x_R$$

&

$$y_{\min} \geq y_B \quad \& \quad y_{\max} \leq y_T$$



- 窗口与一个多边形相交，且窗口内无其它多边形 —— 直线方程作为判别函数

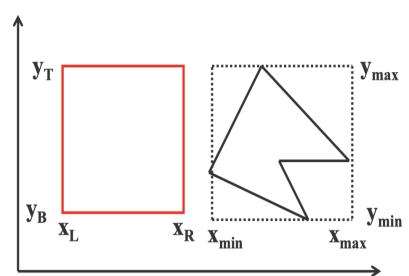
- 窗口为一个多边形所包围

当满足下列条件时，多边形和窗口分离：

- 窗口与一个多边形相分离

$$x_{\min} > x_R \quad \text{or} \quad x_{\max} < x_L$$

$$y_{\min} > y_T \quad \text{or} \quad y_{\max} < y_B$$



4.6 光栅扫描算法小结

- 核心思想

- 增量思想

- 通过增量算法可以减少计算量。

- 编码思想

- 符号判别 → 整数算法
 - 尽可能地提高底层算法的效率，底层上提高效率才是真正解决问题。
 - 图形连贯性
 - 利用连贯性可以大大减少计算量。
 - 分而治之
 - 把一个复杂对象进行分块，分到足够简单再进行处理。
-
-

第五章 二维图形变换

5.1 向量基础

- 向量功能
 - 图形学中，处理三维物体，以及绘制对象的形状、位置、方向。有两大基本工具：向量分析、图形变换。
 - 向量：点和方向的实体（没有位置）
- 向量的表示
 - 两点之差是一个向量，方向指向被减点。 $v = Q - P$
 - n 维向量就是一个 n 元组
 - $w = (w_1, w_2, \dots, w_n)$, $w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$
- 向量基本运算
 - 加减、数乘
 - 归一化
 - 即将向量转变为单位向量，方向不变，长度 = 1
 - 点积
 - $a = (a_1, a_2) \cdot b = (b_1, b_2)$
 - $a \cdot b = a_1 b_1 + a_2 b_2 = |a||b|\cos\langle a, b \rangle$, 用向量描述新闻，新闻相似，则向量夹角余弦接近于 1
 - $a \cdot b > 0, \theta < 90^\circ ; a \cdot b = 0, \theta = 90^\circ ; a \cdot b < 0, \theta > 90^\circ ;$
 - 叉积
 - $a = (a_x, a_y, a_z) \cdot b = (b_x, b_y, b_z) \cdot a \times b = \begin{vmatrix} i & j & k \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix} = |a||b|\sin\langle a, b \rangle$
 - $a \times b$ 和 a 、 b 两个向量都正交，因此可以利用叉积求平面的法向量
 - $a \times b$ 的长度等于由 a 和 b 决定的平行四边形面积
- 向量线性组合
 - 向量两种特殊线性组合 ($w = a_1 v_1 + a_2 v_2 + \dots + a_n v_n$) :
 - 仿射组合：线性组合的系数和等于 1, $\sum_{i=1}^n a_i = 1$
 - 凸组合：线性组合的系数和等于 1，且各系数非负， $\sum_{i=1}^n a_i = 1$ ($a_i \geq 0$)

5.2 图形坐标系

- 坐标系的基本概念
 - 坐标系：建立图形和数之间对应联系的参考系
 - 建模 (modeling)：程序中用于描述对象几何信息的数值
 - 观察 (viewing)：表示对象中大小和位置的数值
 - 二维观察变换的一般方法是在世界坐标系中指定一个观察坐标系统，以该系统为参考通过选定方向和位置来指定矩形裁剪窗口。
- 坐标系分类
 - 维度分类
 - 一维、二维、三维坐标系
 - 坐标轴之间的空间关系分类
 - 直角坐标系、极坐标系、圆柱坐标系、球坐标系
 - 计算机图形学坐标系分类
 - **世界坐标系**：公共坐标系，现实中物体或场景的统一参照系。计算机图形系统中涉及的其它坐标系都是参照它进行定义的。
 - **建模坐标系**：又称局部坐标系，每个物体(对象)有它自己的局部中心和坐标系，独立于世界坐标系来定义物体的几何特性。
 - **观察坐标系**：依据观察窗口的方向和形状在世界坐标系中定义的坐标系。观察坐标系用于指定图形的输出范围。
 - **设备坐标系**：适合特定输出设备输出对象的坐标系，比如屏幕坐标系。设备坐标是整数。
 - **规范化坐标系**：规范化坐标系独立于设备，能容易地转变为设备坐标系，是一个中间坐标系。归一化后的坐标，坐标轴取值范围 0~1。

5.3 二维图形变换原理

5.3.1 图形变换概述

- 用途
 - 各种变换：比例、旋转、镜像、错切、平移
 - 由一个基本的图案，经过变换组合成另外一个复杂图形
 - 用很少的物体组成一个场景
 - 可以通过图形变换组合得到动画效果
- 基本原理
 - 图形变化，但原图形的连边规则没有改变
 - 图形变化，是因为顶点位置改变决定
 - 变换几何关系，保持原拓扑关系。

5.3.2 仿射变换

- 仿射变换 (Affine Transformation / Affine Map)

- 基本功能
 - 平直性：直线变换后仍是直线
 - 平行性：平行线变换后仍平行，且直线上点的位置顺序不变
- 二维仿射变换

- x' 和 y' 都是原始坐标 x 和 y 的线性函数
- $\begin{cases} x' = a_1x + b_1y + c_1 \\ y' = a_2x + b_2y + c_2 \end{cases}$
- 矩阵形式： $[x^* \ y^*] = [x \ y \ 1] \cdot \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \\ c_1 & c_2 \end{bmatrix}$

5.3.3 齐次坐标

- 二维平面中用 (x, y) 表示一个点，不妨说是一个向量 (x, y) 表示一个点。所以可以用第 3 维为常数的 $(x, y, 1)$ 表示二维平面上的向量。
 - 这种 $n+1$ 维表示 n 维的方法称为——齐次坐标表示法， n 维向量 (p_1, p_2, \dots, p_n) 表示为 $(hp_1, hp_2, \dots, hp_n, h)$ ，其中 h 称为哑坐标，特别的 $h=1$ 时称齐次坐标为规范化坐标。
 - 二维仿射变换，齐次坐标表示： $[x^* \ y^* \ 1] = [x \ y \ 1] \cdot \begin{bmatrix} a_1 & a_2 & 0 \\ b_1 & b_2 & 0 \\ c_1 & c_2 & 1 \end{bmatrix}$
 - 不使用齐次坐标可以做比例、对称、旋转变换，但做不到平移变化，无法增加常数项。
-

5.4 基本几何变换

5.4.1 平移变换

- 不产生变形而移动物体的刚体变换，即物体上的每个点移动相同数量的坐标
- 坐标形式： $\begin{cases} x^* = x + T_x \\ y^* = y + T_y \end{cases}$
- 齐次坐标形式： $[x^* \ y^* \ 1] = [x \ y \ 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix} = [x + Tx \ y + Ty \ 1]$

5.4.2 比例变换

- 相对于坐标原点沿 x 方向放缩 S_x 倍，沿 y 方向放缩 S_y 倍。 $S > 1$ 放大， $S < 1$ 缩小。
- 坐标形式： $\begin{cases} x^* = x \cdot S_x \\ y^* = y \cdot S_y \end{cases}$
- 齐次坐标形式： $[x^* \ y^* \ 1] = [x \ y \ 1] \cdot \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = [x \cdot S_x \ y \cdot S_y \ 1]$
- 当 $S_x = S_y$ 时，为整体比例变换， $[x^* \ y^* \ 1] = [x \ y \ 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & S \end{bmatrix} = [x \ y \ S]$ ， $S > 1$ 放大， $0 < S < 1$ 缩小， $S < 0$ 发生关于原点的对称等比变换。

5.4.3 对称变换

- 也称镜像变换或反射变换。有关于x轴、y轴、原点、某条直线的对称变换。
- 关于x轴对称: $[x^* \ y^* \ 1] = [x \ y \ 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [x \ -y \ 1]$
- 关于y轴对称: $[x^* \ y^* \ 1] = [x \ y \ 1] \cdot \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix} = [-x \ y \ 1]$
- 关于原点对称: $[x^* \ y^* \ 1] = [x \ y \ 1] \cdot \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix} = [-x \ -y \ 1]$

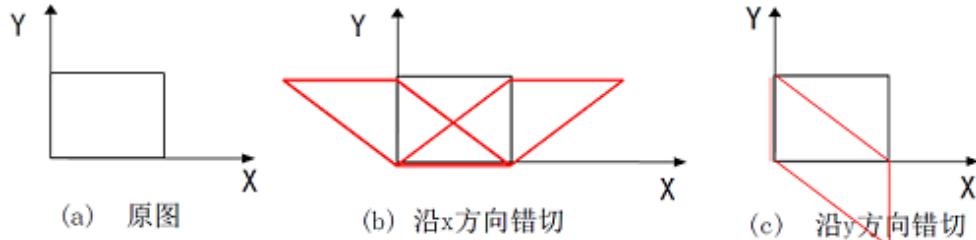
5.4.4 旋转变换

- 将点绕原点旋转角度 θ , 逆时针为正, 顺时针为负
- 坐标形式(逆时针):

$$\begin{cases} x^* = r \cdot \cos(\alpha + \theta) = r \cdot \cos\alpha \cdot \cos\theta - r \cdot \sin\alpha \cdot \sin\theta \\ y^* = r \cdot \sin(\alpha + \theta) = r \cdot \cos\alpha \cdot \sin\theta + r \cdot \sin\alpha \cdot \cos\theta \end{cases} \Rightarrow \begin{cases} x^* = x \cdot \cos\theta - y \cdot \sin\theta \\ y^* = x \cdot \sin\theta + y \cdot \cos\theta \end{cases}$$
- 齐次坐标形式(逆时针):
 $[x^* \ y^* \ 1] = [x \ y \ 1] \cdot \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = [x \cdot \cos\theta - y \cdot \sin\theta \ x \cdot \sin\theta + y \cdot \cos\theta \ 1]$
- 顺时针只要将 $\theta = -\theta$ 即可。

5.4.5 错切变换

- 弹性物体的变形处理



- 变换矩阵中的非对角线元素大都为零, 若变换矩阵中的非对角元素不为0, 则意味着 x 、 y 同时对图形的变换起作用。也就是说, 变换矩阵中非对角线元素起着把图形沿 x 或 y 方向错切的作用。
- x 值或 y 值越小, 错切量越小; x 值或 y 值越大, 错切量越大。

- 齐次坐标形式: $[x^* \ y^* \ 1] = [x \ y \ 1] \cdot \begin{bmatrix} 1 & b & 0 \\ c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [c + cy \ bx + y \ 1]$
- 沿 x 方向错切, 即 $b = 0$: $[x^* \ y^* \ 1] = [x \ y \ 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [c + cy \ y \ 1]$

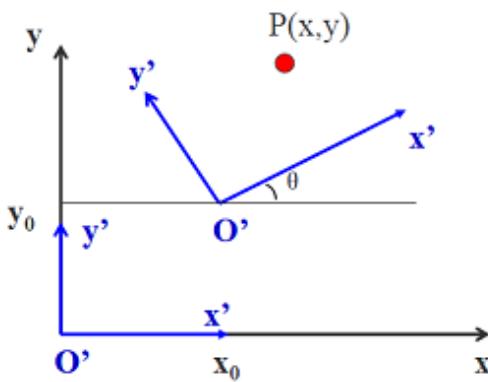
5.4.5 复合变换

- 图形作大于一次的变换, $P^* = P \cdot T = P \cdot (T_1 \cdot T_2 \cdots \cdots T_n)$, $n > 1$, 矩阵相乘不可交换! 任何一个复杂的几何变换都可以看作基本几何变换的组合形式。

- 二维复合平移: $T = T_{t1} \cdot T_{t2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_{x1} & T_{y1} & 1 \\ S_{x1} & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_{x2} & T_{y2} & 1 \\ S_{x2} & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_{x1} + T_{x2} & T_{y1} + T_{y2} & 1 \\ S_{x1} \cdot S_{x2} & 0 & 0 \end{bmatrix}$
 - 二维复合比例: $T = T_{s1} \cdot T_{s2} = \begin{bmatrix} 0 & S_{y1} & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & S_{y2} & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & S_{y1} \cdot S_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$
 - 二维复合旋转:
- $$T = T_{r1} \cdot T_{r2} = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 & 0 \\ -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_2 & \sin\theta_2 & 0 \\ -\sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_1 + \theta_2) & \sin(\theta_1 + \theta_2) & 0 \\ -\sin(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5.4.6 坐标系变换

- 图形变换经常需要从一个坐标系变换到另一个坐标系, 如下图从 $x0y$ 变换到 $x'0'y'$



- 上图变换分两步完成, $x'0'y' \xrightarrow{\text{平移}} x'0y' \xrightarrow{\text{旋转}} x0y$, 注意是从目标到源
 - 平移变换——将 $x'0'y'$ 坐标系的原点平移至 $x0y$ 坐标系的原点
 - 旋转变换——将 x' 轴旋转到 x 轴上
- $T = T_t \cdot T_r = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_0 & -y_0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(-\theta) & \sin(-\theta) & 0 \\ -\sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

5.4.7 任意参考点的几何变换

- 在以往的变换中, 以 $(0, 0)$ 为参考点, 倘若以任意点为参考点, 则:
 - 将参考点移到原点 (平移)
 - 针对原点进行二维几何变换 (变换)
 - 将原点移到参考点 (反平移)

5.5 二维变换矩阵

5.5.1 二维变换矩阵概述

- 二维空间中某点的变化可以表示成点的齐次坐标与 3 阶的二维变换矩阵 T_{2d} 相乘

$$\begin{bmatrix} x^* & y^* & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot T_{2d} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} a & b & p \\ c & d & q \\ l & m & s \end{bmatrix}$$

•

$$T_1 = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \text{对图形进行比例、旋转、对称、错切等变换;}$$

$$T_2 = \begin{bmatrix} l & m \end{bmatrix} \quad \text{对图形进行平移变换}$$

$$T_3 = \begin{bmatrix} p \\ q \end{bmatrix} \quad \text{是对图形作投影变换}$$

$$T_4 = [s] \quad \text{是对图形作整体比例变换}$$

$$\begin{array}{ccc|c} a & b & p \\ c & d & q \\ \hline l & m & s \end{array}$$

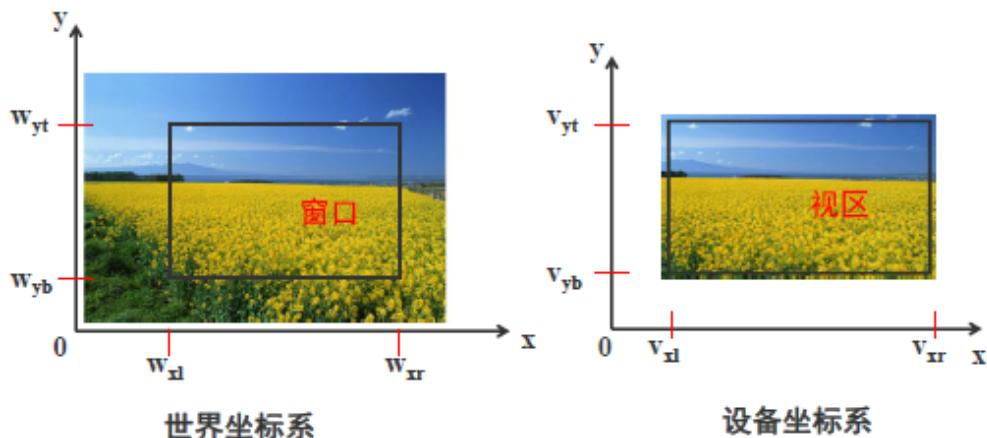
5.5.2 二维图形几何变换的计算

- 点的变换: $[x^* \ y^* \ 1] = [x \ y \ 1] \cdot T$
- 直线的变换 (两端点的变换) : $\begin{bmatrix} x_1^* & y_1^* & 1 \\ x_2^* & y_2^* & 1 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix} \cdot T$
- 多边形的变换 (每个顶点的变换) : $p = \begin{bmatrix} x_1^* & y_1^* & 1 \\ x_2^* & y_2^* & 1 \\ \dots & \dots & \dots \\ x_n^* & y_n^* & 1 \end{bmatrix}$

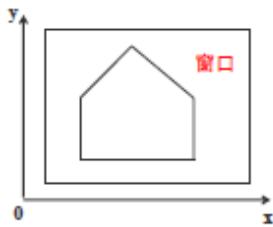
5.4 窗口、视图及变换

5.4.1 窗口和视区

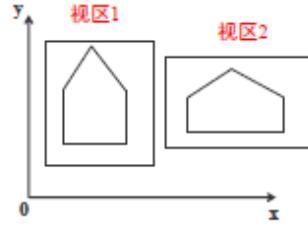
- 窗口: 世界坐标系中要显示的区域
- 视区: 窗口映射到显示器上的区域
- 窗口定义显示什么; 视区定义在何处显示; 二者需要进行坐标变换



- 世界坐标系中的一个窗口可以对应于多个视区



世界坐标系



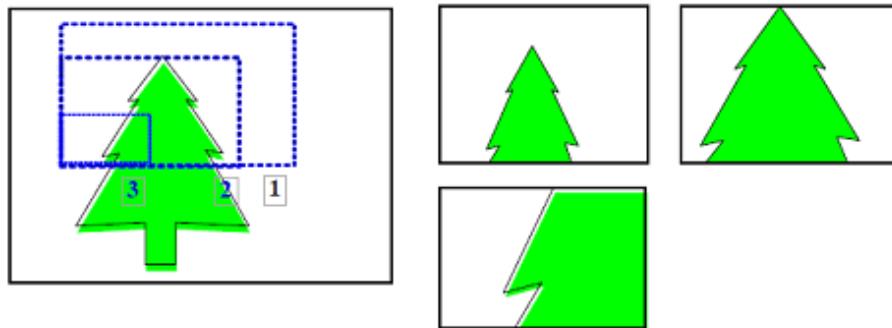
屏幕坐标系

观察变换

- 窗口 \longrightarrow 视区

5.4.2 观察变换

- 观察变换 (Viewing Transformation)
- 变焦距效果
 - 窗口放大/缩小，视区不变，图形缩小/放大



- 整体缩放效果

- 窗口不变，视区放大/缩小，图形放大/缩小

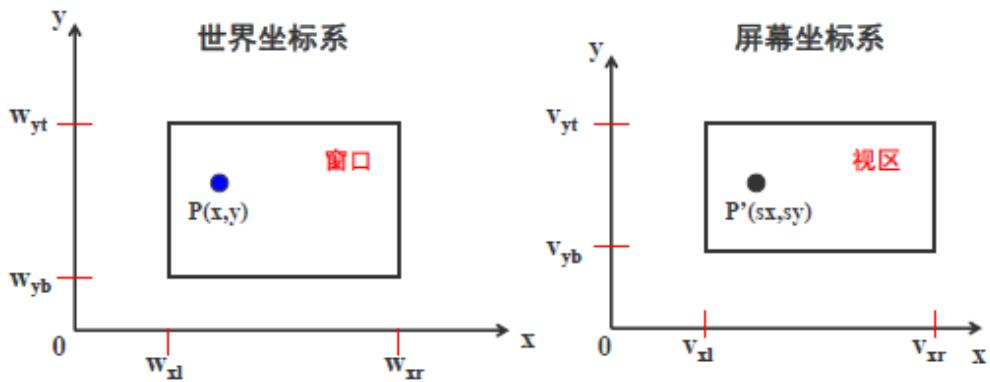


原图及窗口

- 漫游效果
 - 把一个固定大小的窗口在一幅大图形上移动，视区不变。

5.4.3 窗口到视区的变换

- 窗口的点 \rightarrow 视区的点



- 保持比例的映射

- 保持比例：映射之后，中心点仍然在中心，边界点仍然在边界
- $\begin{cases} sx = A \times x + C \\ sy = B \times y + D \end{cases}$
- 比例保持： $\frac{x-w_{xl}}{w_{xr}-w_{xl}} = \frac{sx-v_{xl}}{v_{xr}-v_{xl}} \Rightarrow sx = \frac{x-w_{xl}}{w_{xr}-w_{xl}}(v_{xr} - v_{xl}) + v_{xl}$
- 根据倍数关系： $sx = \frac{v_{xr}-v_{xl}}{w_{xr}-w_{xl}}x + (v_{xl} - \frac{v_{xr}-v_{xl}}{w_{xr}-w_{xl}}w_{xl}) = Ax + C$ ，
 $A = \frac{v_{xr}-v_{xl}}{w_{xr}-w_{xl}}$, $C = v_{xl} - A \times w_{xl}$
- 同理， $B = \frac{v_{yt}-v_{yb}}{w_{yt}-w_{yb}}$, $D = v_{yb} - B \times w_{yb}$

第4章 三维图形变换

4.1 三维图形几何变换

4.1.1 几何变换概述

三维基本几何变换皆是相对于坐标原点和坐标轴进行的几何变换。与二维变换类似，引入齐次坐标表示，即三维空间中某点的变换可以表示成点的齐次坐标与四阶的三维变换矩阵相乘。

$$\bullet p' = [x^* \ y^* \ z^* \ 1] = [x \ y \ z \ 1] \cdot \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ g & h & i & r \\ l & m & n & s \end{bmatrix}$$

$$T_{sd} = \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ g & h & i & r \\ l & m & n & s \end{bmatrix}$$

$$T_1 = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \quad \text{对点进行比例、对称、旋转、错切变换}$$

$$T_2 = [l \ m \ n] \quad \text{对点进行平移变换}$$

$$T_3 = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad \text{作用是进行透视投影变换}$$

$$T_4 = [s] \quad \text{作用是产生整体比例变换}$$

4.1.2 平移变换

- $[x' \ y' \ z' \ 1] = [x \ y \ z \ 1] \cdot T_t = [x \ y \ z \ 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix} = [x + T_x \ y + T_y \ z + T_z \ 1]$

4.1.3 比例变换

- 局部比例变换：

$$[x' \ y' \ z' \ 1] = [x \ y \ z \ 1] \cdot T_s = [x \ y \ z \ 1] \cdot \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [ax \ ey \ iz \ 1]$$

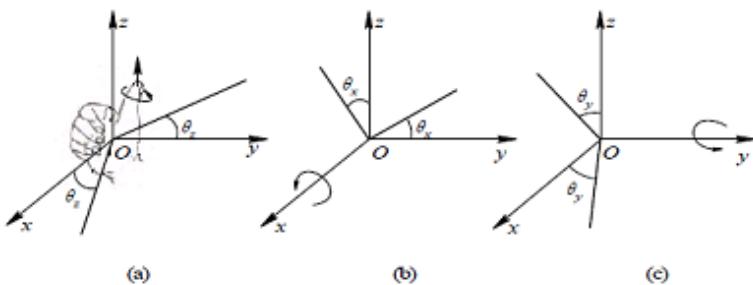
- 整体比例变换：

$$[x' \ y' \ z' \ 1] = [x \ y \ z \ 1] \cdot T_s = [x \ y \ z \ 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix} = [x \ y \ z \ s]$$

4.1.4 旋转变换

三维立体的旋转变换是指给定的三维立体绕三维空间某个指定的坐标轴旋转 θ 角度。旋转后，立体的空间位置将发生变化，但形状不变。

- 右手定则：右手大拇指指向旋转轴的正向，其余四指指向旋转角的正向



(a) 绕z轴正向旋转；(b) 绕x轴正向旋转；(c) 绕y轴正向旋转

- 绕 z 轴旋转：

$$[x \ y \ z \ 1] \cdot \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x \cdot \cos\theta - y \cdot \sin\theta \ x \cdot \sin\theta + y \cdot \cos\theta \ z \ 1]$$

- 绕 x 轴旋转：

$$[x \ y \ z \ 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x \ y \cdot \cos\theta - z \cdot \sin\theta \ y \cdot \sin\theta + z \cdot \cos\theta \ z \ 1]$$

- 绕 y 轴旋转:

$$[x \ y \ z \ 1] \cdot \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x \cdot \sin\theta + y \cdot \cos\theta \ y \ z \cdot \cos\theta - x \cdot \sin\theta \ 1]$$

- 绕任意轴旋转

- 先将轴移动到坐标原点, 再将轴旋转到坐标轴上
- 进行几何变换
- 将轴反旋转, 再反平移到原来位置

4.1.5 对称变换

- 关于坐标平面的对称

- 关于 $x0y$ 平面对称: $T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \end{bmatrix}$, $[x' \ y' \ z' \ 1] = [x \ y \ -z \ 1]$
- 关于 $y0z$ 平面对称: $T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$, $[x' \ y' \ z' \ 1] = [-x \ y \ z \ 1]$
- 关于 $z0x$ 平面对称: $T = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, $[x' \ y' \ z' \ 1] = [x \ -y \ z \ 1]$

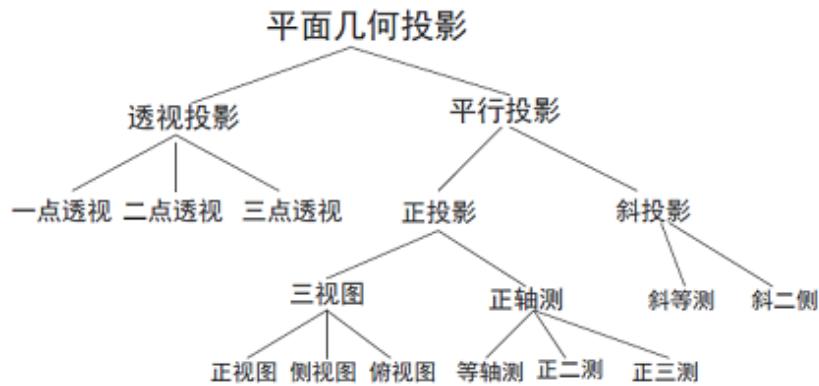
- 关于坐标轴的对称

- 关于 x 轴对称: $T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \end{bmatrix}$, $[x' \ y' \ z' \ 1] = [x \ -y \ -z \ 1]$
- 关于 y 轴对称: $T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \end{bmatrix}$, $[x' \ y' \ z' \ 1] = [-x \ y \ -z \ 1]$
- 关于 z 轴对称: $T = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, $[x' \ y' \ z' \ 1] = [-x \ -y \ z \ 1]$

4.2 投影变换分类

投影变换: 解决三维输出到二维

- 投影法分类 (区别在于投影中心到投影面之间的距离是有/无限的) :

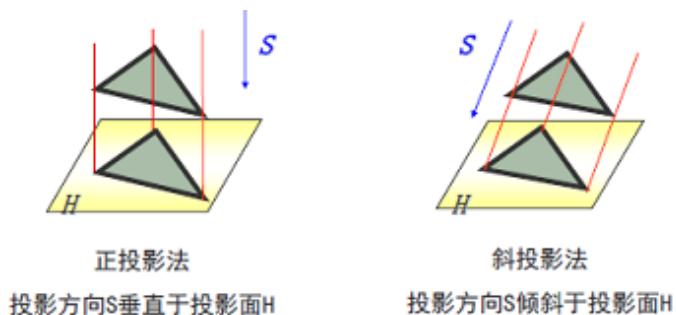


- 透视投影法（中心投影法），比如建筑透视
 - 生成真实感视图但不保持相关比例
 - 平行投影法
 - 正投影法，比如工程样图
 - 斜投影法
 - 保持物体的有关比例不变，物体的各个面的精确视图由平行投影而得，没有给出三维物体外表的真实性表示
-

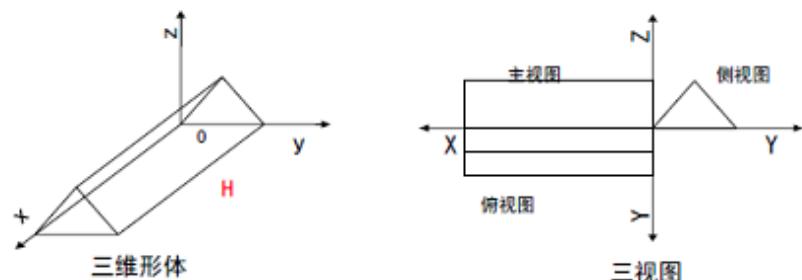
4.3 平行投影（三视图、轴测图）

4.3.1 平行投影概述

- 特点
 - 物体各个面的精确视图又平行投影而得
 - 没有给出三维物体外表的真实性但保持比例



4.3.2 三视图



- 主视图变换矩阵

◦ $T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, 投影在 $x0z$ 面, 直接将 y 置 0.

- 俯视图变换矩阵

◦ $T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -z_0 & 1 \end{bmatrix}$, 下述为推导过程

- 投影在 $x0y$ 面, 直接置 $z = 0$, $T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- 为了使俯视图与主视图都画在一个平面内, 就要使 H 面绕 x 轴顺时针转 90° , 进入 $x0z$ 面

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-90^\circ) & \sin(-90^\circ) & 0 \\ 0 & -\sin(-90^\circ) & \cos(-90^\circ) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- 为了使主视图和俯视图有一定的间距, 还要使 H 面沿 z 方向平移一段距离 $-z_0$,

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -z_0 & 1 \end{bmatrix}$$

- 侧视图变换矩阵

◦ $T = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_0 & 0 & 0 & 1 \end{bmatrix}$, 下述为推导过程

- 投影在 $y0z$ 面, 直接置 $x = 0$, $T = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- 为了使侧视图与主视图也在一个平面内, 就要使 W 面绕 z 轴正转 90° , 进入 $x0z$ 面

$$T = \begin{bmatrix} \cos(-90^\circ) & \sin(-90^\circ) & 0 & 0 \\ -\sin(-90^\circ) & \cos(-90^\circ) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- 为使主视图和侧视图有一定的间距, 还要使 W 面沿负 x 方向平移一段距离 $-x_0$,

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_0 & 0 & 0 & 1 \end{bmatrix}$$

- 主视图: $[x' \ y' \ z' \ 1] = [x \ 0 \ z \ 1]$

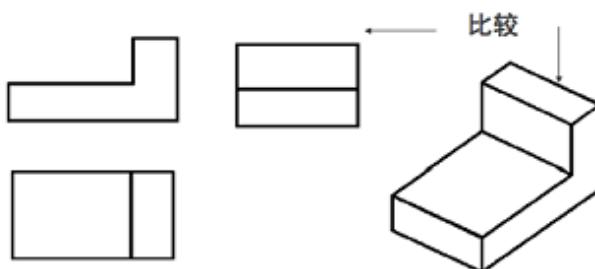
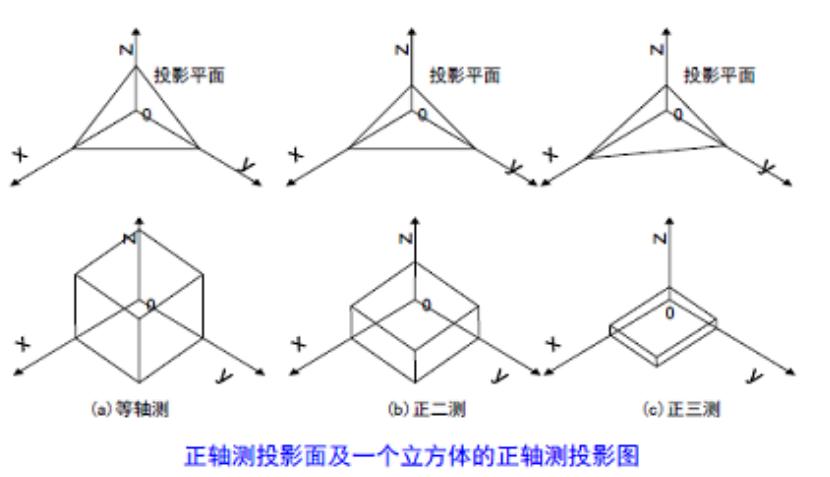
- 俯视图: $[x' \ y' \ z' \ 1] = [x \ 0 \ -(y+z_0) \ 1]$

- 侧视图: $[x' \ y' \ z' \ 1] = [-(y+x_0) \ 0 \ z \ 1]$

- 三个视图中的 y' 均为 0, 表明三个视图均落在 $x0z$ 面上

4.3.3 正轴侧

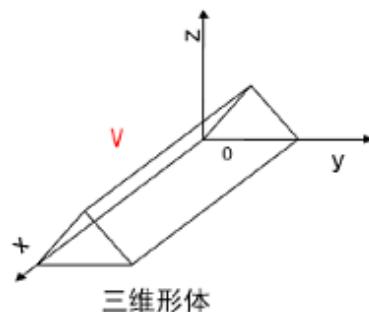
- 等轴测：当投影面与三个坐标轴之间的夹角都相等
- 正二测：当投影面与两个坐标轴之间的夹角相等
- 正三测：当投影面与三个坐标轴之间的夹角都不相等



表现力和度量性好，但直观性差 直观性好，但度量性差，辅助图样

- 正轴侧变换矩阵：

- 以 V 面为轴测投影面（三视图中的 $x0z$ 面），先将物体绕 Z 轴转 γ 角，接着绕 X 轴转 $-\alpha$ 角，最后向 V 面投影



$$T_{\pi} = T_Z \cdot T_X \cdot T_V = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 & 0 \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma\sin\alpha & 0 \\ -\sin\gamma & 0 & -\cos\gamma\sin\alpha & 0 \\ 0 & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{即 } [x' \ y' \ z' \ 1] = [x\cos\gamma - y\sin\gamma \ 0 \ -x\sin\gamma\sin\alpha - y\cos\gamma\sin\alpha + z\cos\alpha \ 1]$$

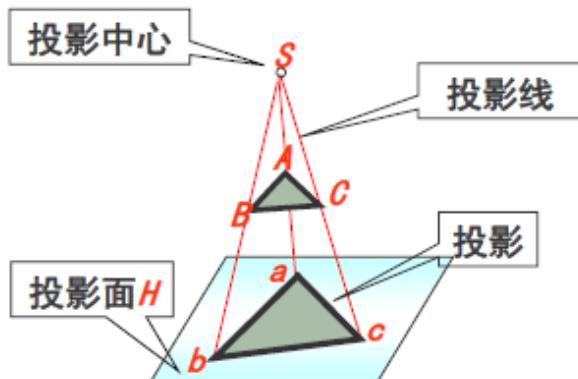
- 正等轴测图: 取 $\gamma = 45^\circ$, $\alpha = -35.26^\circ$, $T_{\text{正等轴测}} = \begin{bmatrix} 0.7071 & 0 & -0.4082 & 0 \\ -0.7071 & 0 & -0.4082 & 0 \\ 0 & 0 & 0.8165 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- 正二测图: 取 $\gamma = 20.7^\circ$, $\alpha = 19.47^\circ$, $T_{\text{正二测}} = \begin{bmatrix} 0.9354 & 0 & -0.1178 & 0 \\ -0.7071 & 0 & -0.3118 & 0 \\ 0 & 0 & 0.9428 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

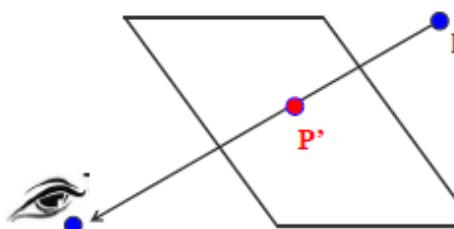
4.4 透视投影

4.4.1 透视投影概述

- 特点
 - 物体投影视图由计算投影线与观察平面之交点而得
 - 生成真实感视图但不保持比例



- 三维变换矩阵, $T_{3D} = \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ g & h & i & r \\ l & m & n & s \end{bmatrix}$, 其中 p、q、r 能产生透视变换的效果。

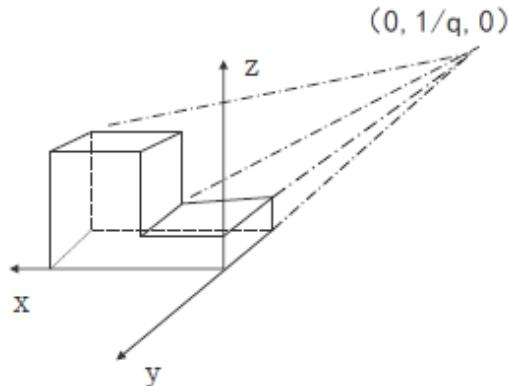


该平面为透视投影面, 穿点P' 为P的透视投影

4.4.2 一点透视

- 假设 $q \neq 0$, $p = r = 0$

- $[x \ y \ z \ 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & q \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x \ y \ z \ qy + 1]$
- 齐次化后: $[x' \ y' \ z' \ 1] = \left[\frac{x}{qy+1} \quad \frac{y}{qy+1} \quad \frac{z}{qy+1} \quad 1 \right]$
 - 当 $y = 0$ 时, $[x' \ y' \ z' \ 1] = [x \ 0 \ z \ 1]$, 即处于 $y=0$ 平面内的点变换后无变化
 - 当 $y \rightarrow \infty$ 时, $[x' \ y' \ z' \ 1] = \left[0 \quad \frac{1}{q} \quad 0 \quad 1 \right]$, 所有点都集中到 y 轴的 $1/q$ 处, 这点叫灭点

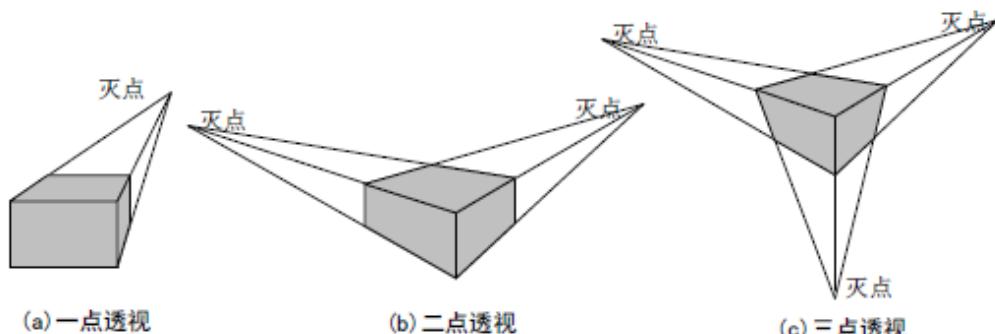


- 只形成一个灭点的透视变换即为一点透视。
- 同样的分别假设 p, r 其中一个不为 0, 另外两个为 0, 都会产生一个灭点 $(\frac{1}{p}, 0, 0)$ 、 $(0, 0, \frac{1}{r})$ 。

4.4.3 多点透视

- 根据一点透视的原理予以推广, 如果 p, q, r 三个元素中有两个为非零元素时, 将会生成两个灭点, 因此得到两点透视; 相应的三点透视概念也可得知。

- 如当 $p \neq 0, r \neq 0$, $[x \ y \ z \ 1] \cdot \begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x \ y \ z \ px + rz + 1]$
- 齐次化后: $[x' \ y' \ z' \ 1] = \left[\frac{x}{px + rz + 1} \quad \frac{y}{px + rz + 1} \quad \frac{z}{px + rz + 1} \quad 1 \right]$
- 分别看到两个灭点, $(\frac{1}{p}, 0, 0)$ 、 $(0, 0, \frac{1}{r})$

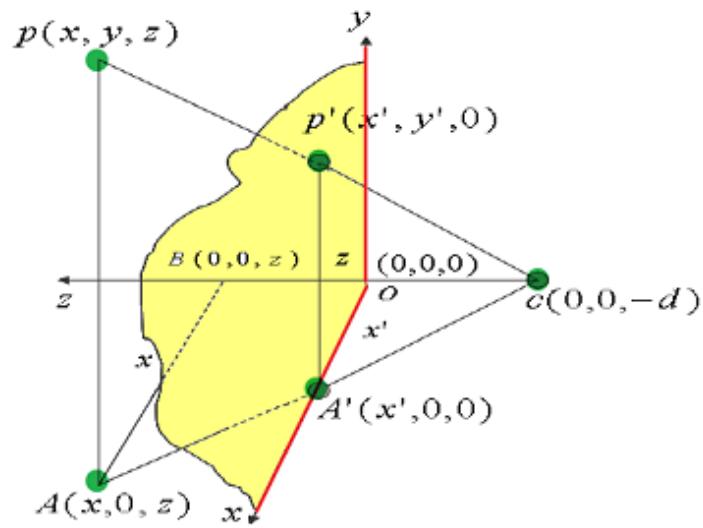


一点透视有一个主灭点, 即投影面与一个坐标轴正交, 与另外两个坐标轴平行

两点透视有两个主灭点, 即投影面与两个坐标轴相交, 与另一个坐标轴平行

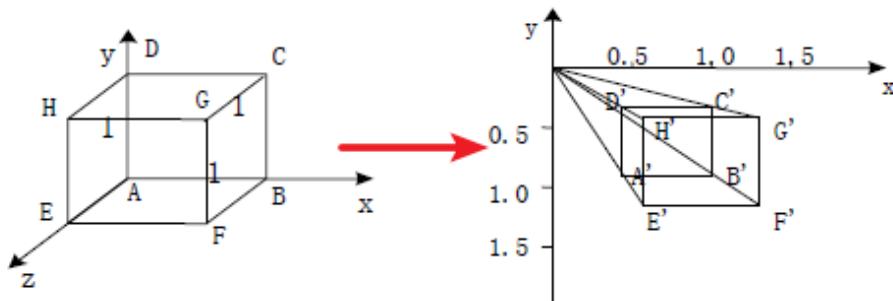
三点透视有三个主灭点, 即投影面与三个坐标轴都相交

4.4.4 生成透视投影图的方法



- 设投影中心: $c(0, 0, -d)$, 现在推空间一点 $p(x, y, z)$ 的透视投影点 $p'(x', y', z')$
- $\triangle ABC \sim \triangle A'OC \Rightarrow \frac{x'}{x} = \frac{y'}{y} = \frac{d}{d+z} \Rightarrow x' = \frac{x}{1+z/d}, y' = \frac{y}{1+z/d}, z' = 0$
- 矩阵形式是: $[x \ y \ z \ 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{1}{d} \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x \ y \ z \ \frac{z}{d} + 1]$
 - 透视坐标与 z 值成反比。即 z 值越大, 透视坐标值越小
 - d 的取值不同, 可以对形成的透视图有放大和缩小的功能。当值较大时, 形成的透视图变大; 反之缩小。
- 再乘以向投影面投影的变换矩阵, 就得到点在画面上的投影
$$[x \ y \ z \ 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{1}{d} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
 - 若投影中心在无穷远处, 则 $\frac{1}{d} \rightarrow 0$, 上式变为平行投影。

4.4.5 一点透视投影实例



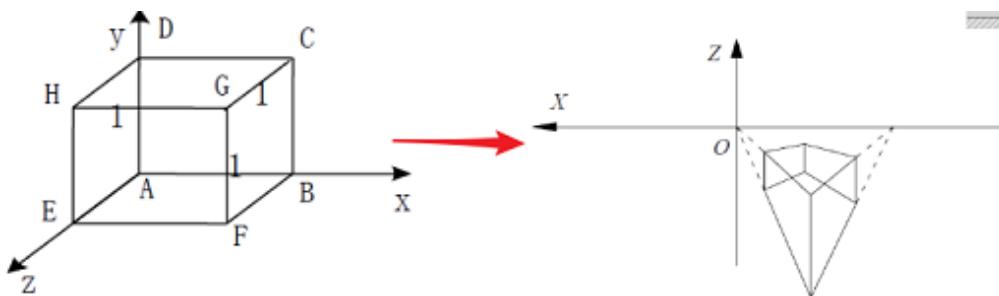
- 设 $l = -0.8, m = -1.6, n = -2$, 视距 $d = -2.5$

将上述值代入一点透视变换矩阵，得到：

$$\begin{array}{l}
 A = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 1 & 0 & 1 \end{bmatrix}, \quad D = \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix}, \quad E = \begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}, \quad F = \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix}, \quad G = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}, \quad H = \begin{bmatrix} 0 & 1 & 1 & 1 \end{bmatrix} \\
 \\
 = \begin{bmatrix} 0.8 & -1.6 & 0 & 1.8 \\ 1.8 & -1.6 & 0 & 1.8 \\ 1.8 & -0.6 & 0 & 1.8 \\ 0.8 & -0.6 & 0 & 1.8 \\ 0.8 & -1.6 & 0 & 1.4 \\ 1.8 & -1.6 & 0 & 1.4 \\ 1.8 & -0.6 & 0 & 1.4 \\ 0.8 & -0.6 & 0 & 1.4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -0.4 \\ -0.8 & -1.6 & 0 & 1.8 \end{bmatrix} = \begin{bmatrix} 0.44 & -0.89 & 0 & 1 \\ 1 & -0.89 & 0 & 1 \\ 1 & -0.33 & 0 & 1 \\ 0.44 & -0.33 & 0 & 1 \\ 0.57 & -1.14 & 0 & 1 \\ 1.29 & -1.14 & 0 & 1 \\ 1.29 & -0.43 & 0 & 1 \\ 0.57 & -0.43 & 0 & 1 \end{bmatrix} \begin{array}{l} A' \\ B' \\ C' \\ D' \\ E' \\ F' \\ G' \\ H' \end{array}
 \end{array}$$

$l = -0.8, m = -1.6, n = -2, d = -2.5$

4.4.6 二点透视投影实例：



$$T_{p^2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

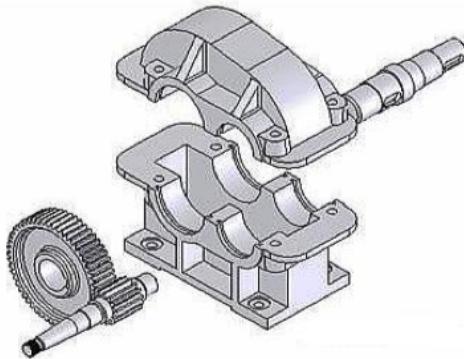
$$= \begin{bmatrix} \cos \theta & 0 & 0 & p \cdot \cos \theta - r \sin \theta \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & 0 & p \cdot \sin \theta + r \cdot \cos \theta \\ l \cdot \cos \theta + n \cdot \sin \theta & m & 0 & p(l \cdot \cos \theta + n \cdot \sin \theta) + r(n \cdot \cos \theta - l \cdot \sin \theta) + 1 \end{bmatrix}$$

- 将物体平移到适当位置 l, m, n
- 将物体绕 y 轴旋转 θ 角
- 进行透视变换
- 最后向 $x0y$ 面做正投影，即得二点透视图

第七章 曲线曲面(一)

7.1 几何造型简史

- 几何造型技术 —— 表达物体模型形状的技术
- **三类三维模型**
 - 线框模型 —— 顶点和棱边来表示物体
 - 曲面模型 (最常用) —— 描述物体表面和表面的连接关系, 不描述物体内部的点的属性
 - 实体模型 —— 不但有物体的外观而且也有物体内点的描述



7.2 参数曲线基本概念

7.2.1 曲线和曲面的三种表示方法

- **非参数表示**

- **显式表示**

$y = f(x)$, 一个 x 值对应一个 y 值。因此显式方程不能表示封闭或多值曲线。

- **隐式表示**

$f(x, y) = 0$, 易于判断一个点是否在曲线上。

- **显式或隐式表示存在的问题**

与坐标轴相关、隐函数表示不直观、作图不方便、显函数表示存在多值性、会出现斜率为无穷大的情形

- **参数表示**

- **特性**

① 用 t 表示参数, 平面曲线 $p(t) = [x(t), y(t)]$, 空间曲线 $p(t) = [x(t), y(t), z(t)]$

② 等价于笛卡尔分量表示, $p(t) = x(t) * i + y(t) * j + z(t) * k$

- **区间规范化**

① 假设曲线段对应的参数区间为 $[a, b]$, 即 $a \leq t \leq b$ 。为了方便计算, 可以将区间 $[a, b]$ 规范化为 $[0, 1]$, 参数变换为 $t' = \frac{t-a}{b-a}$, $p = p(t)$, $t \in [0, 1]$ 。

② 该形式把曲线上表示一个点的位置矢量的各个分量合写在一起当成一个整体, 考虑的是曲线上的点之间的相对位置关系而不是它们与所取坐标系之间的相对位置关系。

③ 可以把曲面表示成为双参数 u 和 v 的矢量函数, $p(u, v) = p(x(u, v), y(u, v), z(u, v))$
 $(u, v) \in [0, 1] * [0, 1]$ 。

- **参数方程的优势**

① 可以满足几何不变性的要求 —— 表示的形状不随所取坐标系而改变

② 有更大的自由度来控制曲线、曲面的形状

- ③ 直接对参数方程进行几何变换 —— 非参数方程的几何变换必须对图形上每个型值点进行几何变换，而参数表示的几何变换可对其参数方程直接进行变换
- ④ 便于处理斜率为无穷大的情形，不会因此而中断计算
- ⑤ 界定曲线、曲面的范围十分简单 —— 具有规格化的参数变量 $t \in [0, 1]$
- ⑥ 易于用向量和矩阵运算，简化计算

7.2.2 参数曲线的基本概念

- 基本数学形式

$$\begin{cases} x = x(t) \\ y = y(t) \\ z = z(t) \end{cases} \quad 0 \leq t \leq 1$$

$$p'(t) = \frac{dP}{dt} \quad p''(t) = \frac{d^2 P}{dt^2}$$

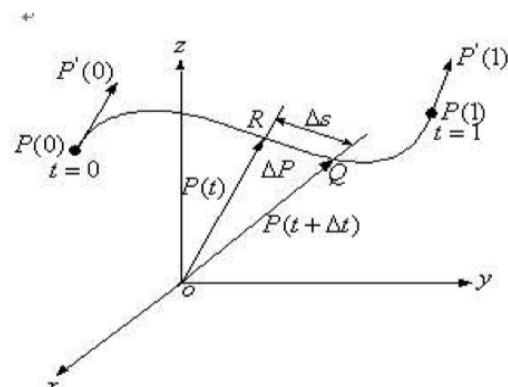
- 位置矢量

曲线上任一点的位置矢量可表示为： $p(t) = [x(t), y(t), z(t)]$

- 切矢量

**选择弧长s作为参数，当
 $\Delta t \rightarrow 0$ 时，弦长 $\Delta s \rightarrow 0$ ，但
 方向不能趋向于0**

$$T = \frac{dP}{ds} = \lim_{\Delta s \rightarrow 0} \frac{\Delta P}{\Delta s} \quad \text{单位矢量}$$



根据弧长微分公式有：

$$(ds)^2 = (dx)^2 + (dy)^2 + (dz)^2$$

引入参数 t , 可改写为:

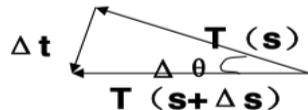
$$(ds/dt)^2 = (dx/dt)^2 + (dy/dt)^2 + (dz/dt)^2 = |P'(t)|^2$$

$$T = \frac{dP}{ds} = \frac{dP}{dt} \cdot \frac{dt}{ds} = \frac{P'(t)}{|P'(t)|}$$

即 T 是单位切矢量

- 曲率

切向量求导, 求导以后还是一个向量, 称为曲率, 其几何意义是曲线的单位切向量对弧长的转动率, 即刻画这一点曲线的弯曲程度。



$$\kappa = |T'| = \lim_{\Delta s \rightarrow 0} \left| \frac{\Delta T}{\Delta s} \right| = \lim_{\Delta s \rightarrow 0} \left| \frac{T(s + \Delta s) - T(s)}{\Delta s} \right| = \lim_{\Delta s \rightarrow 0} \left| \frac{\Delta \theta}{\Delta s} \right|$$

曲率越大, 表示曲线的弯曲程度越大

曲率 κ 的倒数 $\rho = \frac{1}{\kappa}$ 称为曲率半径

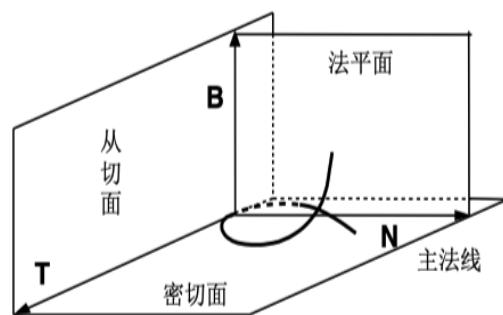
曲率半径越大, 圆弧越平缓

曲率半径越小, 圆弧越陡

- 法矢量

法矢量是与切矢量垂直的向量

N 、 B 构成的平面称为法平面, N 、 T 构成的平面称为密切平面, B 、 T 构成的平面称为从切平面



- 挠率

刻画空间曲线的扭曲程度。空间曲线的扭曲程度等价于密切平面的法矢量(即曲线的副法矢量)关于弧长的变化率。

挠率 τ 的绝对值等于副法线方向(或密切平面)对于弧长的转动率:

$$|\tau| = \lim_{\Delta s} \left| \frac{\Delta \theta}{\Delta s} \right|$$

- 插值

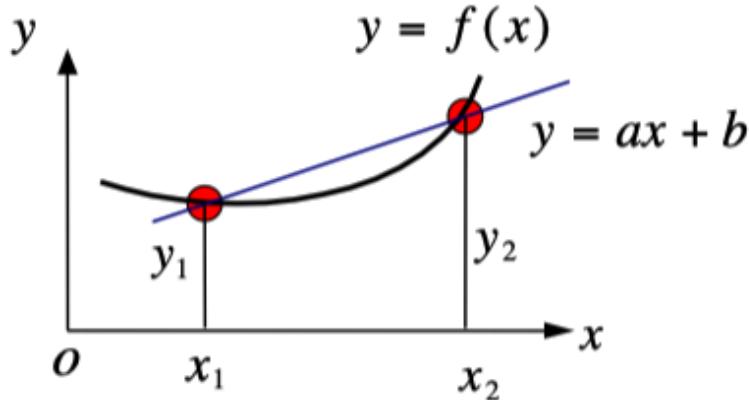
自由曲线和自由曲面一般通过少数分散的点生成，这些点叫做"型值点"、"样本点"或"控制点"。

给定一组有序的数据点 $P_i (i = 0, 1, 2, \dots, n)$ ，要求构造一条曲线顺序通过这些数据点，称为对这些数据点进行插值，所构造的曲线称为插值曲线。

构造插值曲线曲面所采用的数学方法称为曲线曲面插值法。

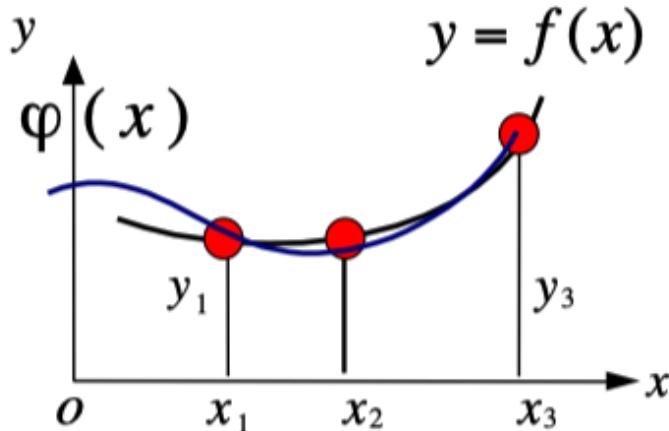
- 线性插值

假设给定函数 $f(x)$ 在两个不同点 x_1 和 x_2 的值，用一个线性函数: $y = ax + b$ ，近似代替，称为线性插值函数。



- 抛物线插值

已知三个点的坐标，要求构造一个抛物线函数。 $\phi(x) = ax^2 + bx + c$



- 拟合

构造一条曲线使之在某种意义上最接近给定的数据点(但未必通过这些点)，所构造的曲线为拟合曲线。

逼近通常指用一些性质较好的函数近似表示一些性质不好的函数。因此插值和拟合都可以视为逼近。

- 光顺

指曲线的拐点不能太多(有一、二阶导数等)。[拐点——凸曲线和凹曲线的连接点]

对平面曲线而言, 相对光顺的条件是: a. 具有二阶几何连续性(G^2)

b. 不存在多余拐点和奇异点

c. 曲率变化较小

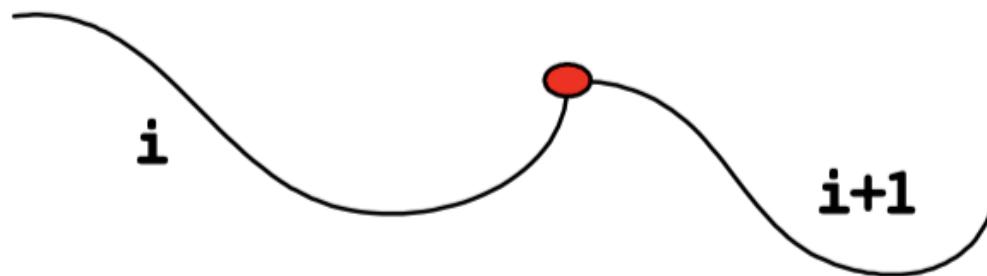
- 连续性

- 参数连续性

- 0阶参数连续性

记为 C^0 连续性, 是指曲线的几何位置连接, 即第一个曲线段在 t_{i1} 处的 x 、 y 、 z 值与第二个曲线段在 $t_{(i+1)0}$ 处的 x 、 y 、 z 值相等。

$$p_i(t_{i1}) = p_{(i+1)}(t_{(i+1)0})$$



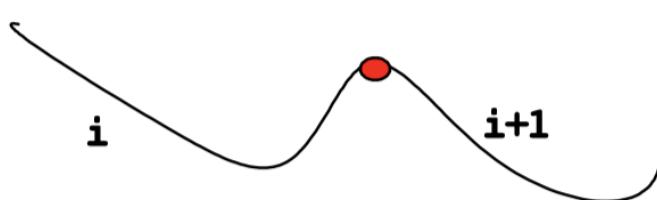
- 1阶参数连续性

记为 C^1 连续性, 指代表两个相邻曲线段的方程在相交点处有相同的一阶导数(切线)

$$p_i(t_{i1}) = p_{(i+1)}(t_{(i+1)0})$$

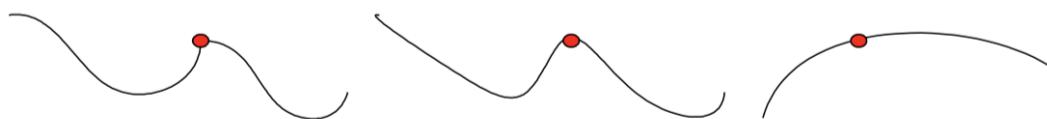
$$\text{且 } p'_i(t_{i1}) = p'_{(i+1)}(t_{(i+1)0})$$

一阶连续性对数字化
绘画及一些设计应用
已经足够



- 2阶参数连续性

记为 C^2 连续性, 指两个相邻曲线段的方程在相交点处具有相同的一阶和二阶导数。 C^2 连续性即交点处的切向量变化率相等, 即切线从一个曲线段平滑地变化到另一个曲线段。二阶连续性对电影中的动画途径和很多精密 CAD 需求有用。



(a) 0阶连续性

(b) 1阶连续性

(c) 2阶连续性

- 几何连续性

- 0阶几何连续性

记为 G^0 连续性。与0阶参数连续性的定义相同，满足 $p_i(t_{i1}) = p_{(i+1)}(t_{(i+1)0})$

- 1阶几何连续性

记为 G^1 连续性。若要求在结合处达到 G^1 连续，就是说两条曲线在结合处在满足 G^0 连续的条件下，并有公共的切矢。即切矢量方向一致。

- 2阶几何连续性

记为 G^2 连续性。就是说两条曲线在满足 G^1 连续的条件下，并有公共的曲率，即方向相同。

- 参数化

- 均匀参数化

- 节点在参数轴上呈等距分布。如0、1/10、2/10...

- 累加弦长参数化(根据长度的比例关系来确定 t)

- 这种参数法如实反映了型值点按弦长的分布情况，能够克服型值点按弦长分布不均匀的情况下采用均匀参数化所出现的问题。

- $$\begin{cases} t_0 = 0 \\ t_i = t_{i-1} + |\Delta P_{i-1}|, i = 1, 2, \dots, n \end{cases} \quad \Delta P_i = P_{i+1} - P_i$$

- 向心参数化法

- 向心参数化法假设在一段曲线弧上的向心力与曲线切矢从该弧段始端至末端的转角成正比，加上一些简化假设，得到向心参数化法。此法尤其适用于非均匀型值点分布。

$$t_0 = 0$$

- $$t_i = t_{i-1} + |\Delta P_{i-1}|^{1/2}, i = 1, 2, \dots, n$$

- 参数曲线的代数和几何形式

- 代数形式

- $$\begin{cases} x(t) = a_{3x}t^3 + a_{2x}t^2 + a_{1x}t + a_{0x} \\ y(t) = a_{3y}t^3 + a_{2y}t^2 + a_{1y}t + a_{0y} \\ z(t) = a_{3z}t^3 + a_{2z}t^2 + a_{1z}t + a_{0z} \end{cases} \quad t \in [0, 1]$$

上述代数式写成矢量式是：

$$P(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0 \quad t \in [0,1]$$

■

注意： a_3, a_2, a_1, a_0 是参数曲线的系数，但记住不是常数而是向量。 a_3 对应刚才的 a_{3x}, a_{3y}, a_{3z} 。改变系数曲线如何变化是不清楚的，这是代数形式的缺点

- 几何形式

- 几何形式是利用一条曲线端点的几何性质来刻画一条曲线。端点的几何性质，就是指曲线的端点位置、切向量、各阶导数等端点的信息。
- 对三次参数曲线，若用其端点位矢 $P(0)$ 、 $P(1)$ 和切矢 $P'(0)$ 、 $P'(1)$ 描述。需要这四个量来刻画三次参数曲线。

将 $P(0)$ 、 $P(1)$ 、 $P'(0)$ 和 $P'(1)$ 简记为：

$$P_0, P_1, p_0, p_1$$

$$P(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0 \quad t \in [0,1]$$

■

$$P_0 = a_0$$

$$\begin{aligned} P_1 &= a_3 + a_2 + a_1 + a_0 & \longrightarrow & \begin{cases} a_0 = P_0 \\ a_1 = P_1 \\ a_2 = -3P_0 + 3P_1 - 2P'_0 - P'_1 \\ a_3 = 2P_0 - 2P_1 + P'_0 + P'_1 \end{cases} \\ p_0 &= a_1 \\ p_1 &= 3a_3 + 2a_2 + a_1 \end{aligned}$$

得到：

$$P(t) = (2t^3 - 3t^2 + 1)p_0 + (-2t^3 + 3t^2)p_1 + (t^3 - 2t^2 - t)p_0' + (t^3 - t^2)p_1'$$

■

$$\text{令: } \begin{aligned} F_0(t) &= 2t^3 - 3t^2 + 1 & F_1(t) &= -2t^3 + 3t^2 \\ G_0(t) &= t^3 - 2t^2 - t & G_1(t) &= t^3 - t^2 \end{aligned}$$

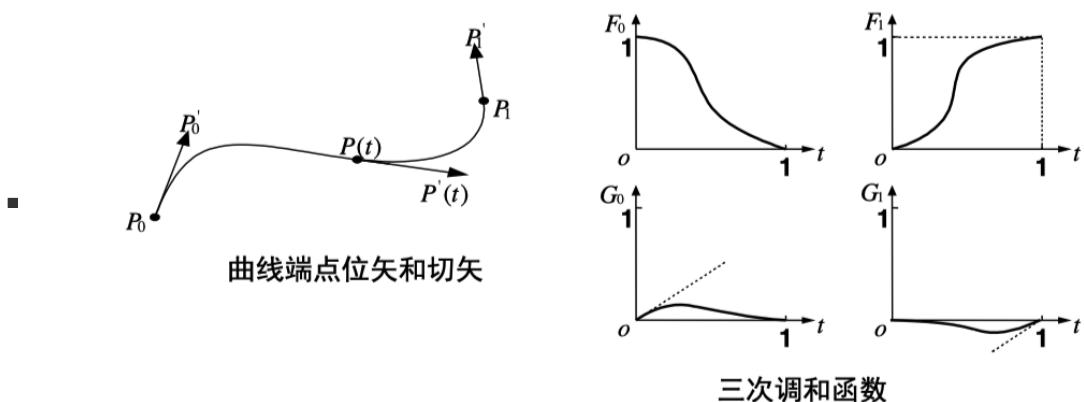
$$P(t) = F_0 P_0 + F_1 P_1 + G_0 P_0' + G_1 P_1' \quad t \in [0,1]$$

$$P(t) = F_0 P_0 + F_1 P_1 + G_0 P_0' + G_1 P_1' \quad t \in [0,1]$$

上式是三次Hermite曲线（三次哈密特曲线）的几何形式，几何系数是：

$$P_0, P_1, P_0', P_1'$$

F0、F1、G0、G1称为调和函数（或混合函数）



7.3 Bezier曲线与曲面

7.3.1 Bezier曲线的背景和定义

- Bezier曲线的背景

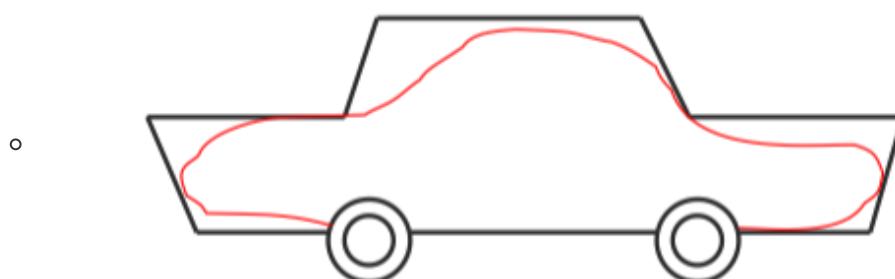
- 用曲线段拟合曲线 $f(x)$ 时，可以把曲线表示为许多小线段 $\phi(x)$ 之和，其中 $\phi(x)$ 称为基(混合)函数。

$$f(x) = \sum_{i=0}^n a_i * \phi_i(x).$$

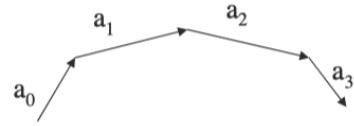
- 这些基函数是要用于计算和显示的。因此，经常选择多项式作为基函数。

$$\phi(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0.$$

- 1962年，Bezier构造了一种以逼近为基础的参数曲线和曲面的设计方法。该方法基点是在进行汽车外形设计时，先用折线段勾画出汽车的大致外形轮廓，然后用光滑的参数曲线去逼近这个折线多边形。**这个折线多边形被称为特征多边形，逼近该特征多边形的曲线被称为Bezier曲线。**



从 a_0 的末端到 a_n 的末端所形成的折线称为控制多边形或贝塞尔多边形



- $p(t) = \sum_{i=0}^n a_i f_{i,n}(t) \quad 0 \leq t \leq 1$

$$f_{i,n}(t) = \begin{cases} 1, & i = 0, \\ \frac{(-t)^i}{(i-1)!} \frac{d^{i-1}}{dt^{i-1}} \left(\frac{(1-t)^{n-1} - 1}{t} \right) & i > 0 \end{cases} \quad \text{称为贝塞尔基函数}$$

$$B_{i,n}(t) = C_n^i t^i (1-t)^{n-i} = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i} \quad (i = 0, 1, \dots, n)$$

- C_n^i 从n个不同元素中，任取*i*($i \leq n$)个元素并成一组，叫做从n个不同元素中取出*i*个元素的一个组合

Forest证明了**Bezier**曲线的基函数可以简化成伯恩斯
坦基函数

- Bezier曲线的定义

- 针对Bezier曲线，给定空间 $n+1$ 个点的位置矢量 $P_i (i = 0, 1, 2, \dots, n)$ ，则Bezier曲线段的参数方程表示如下：($0^0 = 1, 0! = 1$)

-

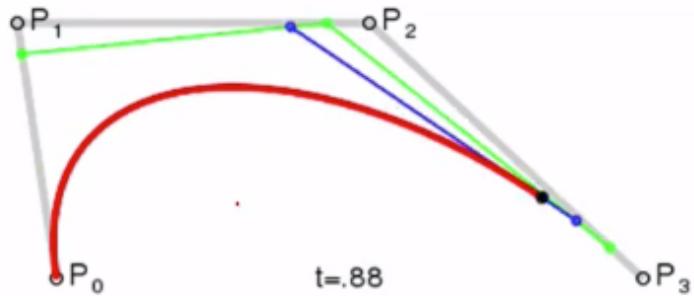
$$p(t) = \sum_{i=0}^n P_i B_{i,n}(t) \quad t \in [0, 1]$$

其中 $\mathbf{p}_i (x_i, y_i, z_i)$, $i=0, 1, 2, \dots, n$ 是控制多边形的 $n+1$ 个顶点，即构成该曲线的特征多边形； $B_{i,n}(t)$ 是Bernstein 基函数，有如下形式：

$$B_{i,n}(t) = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i} = C_n^i t^i (1-t)^{n-i} \quad (i = 0, 1, \dots, n)$$

二项式定理，又称牛顿二项式定理。该定理给出两个数之和的整数次幂的恒等式

- P_i 代表空间的很多点， t 在 0 到 1 之间，把 t 代进去可以算出一个数 —— 即平面或空间一个点。随着 t 值的变化，点也在变化。当 t 从 0 变到 1 时，就得到空间的一个图形，这个图形就是 Bezier 曲线。
-



- 一次Bezier曲线

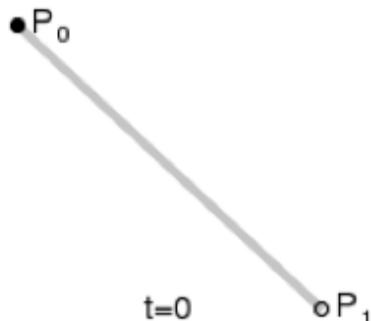
$$p(t) = \sum_{i=0}^n P_i B_{i,n}(t) \quad t \in [0,1]$$

当n=1时，有两个控制点p₀和p₁，Bezier多项式是一次多项式：

- $p(t) = \sum_{i=0}^1 P_i B_{i,1}(t) = P_0 B_{0,1}(t) + P_1 B_{1,1}(t)$

$$B_{i,n}(t) = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i}$$

- $B_{0,1}(t) = (1-t)$, $B_{1,1}(t) = t$, $p(t) = (1-t)P_0 + tP_1$, 即连接起点P₀和终点P₁的直线段。



- 二次Bezier曲线

$$p(t) = \sum_{i=0}^n P_i B_{i,n}(t) \quad t \in [0,1]$$

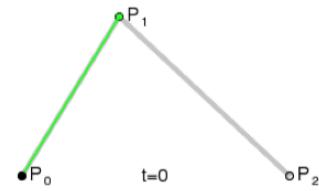
当n=2时，有3个控制点p₀、p₁和p₂，Bezier多项式是二次多项式：

$$p(t) = \sum_{i=0}^2 P_i B_{i,2}(t) = P_0 B_{0,2}(t) + P_1 B_{1,2}(t) + P_2 B_{2,2}(t)$$

- $B_{0,2}(t) = (1-t)^2$, $B_{1,2}(t) = 2t(1-t)$, $B_{2,2}(t) = t^2$, $p(t) = (P_2 - 2P_1 + P_0)t^2 + 2(P_1 - P_0)t + P_0$

二次Bezier曲线为抛物线，其矩阵形式为：

- $p(t) = [t^2 \quad t \quad 1] \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \end{bmatrix}$



○ 三次Bezier曲线

- 三次Bezier曲线由4个控制点生成，这时n=3，有4个控制点 p_0 、 p_1 、 p_2 、 p_3 ，Bezier多项式是三次多项式。

- $p(t) = \sum_{i=0}^3 P_i B_{i,3}(t) = P_0 B_{0,3}(t) + P_1 B_{1,3}(t) + P_2 B_{2,3}(t) + P_3 B_{3,3}(t)$
- $p(t) = \sum_{i=0}^3 P_i B_{i,3}(t) = (1-t^3)P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$

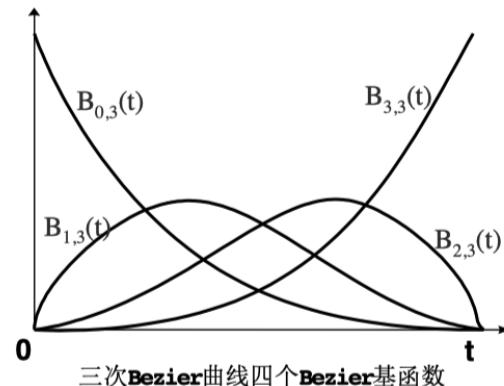
$$B_{0,3}(t) = (1-t)^3$$

其中 $B_{1,3}(t) = 3t(1-t)^2$ 为三次Bezier曲线的基函数。

$$B_{2,3}(t) = 3t^2(1-t)$$

$$B_{3,3}(t) = t^3$$

- 这四条曲线均是三次曲线，任何三次Bezier曲线都是这四条曲线的线形组合



三次Bezier曲线四个Bezier基函数

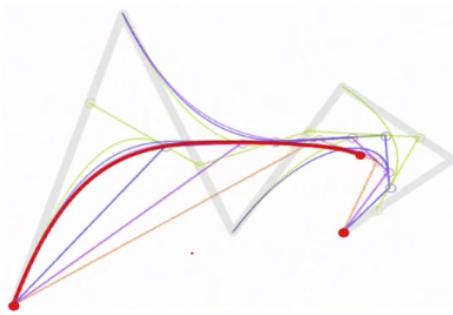
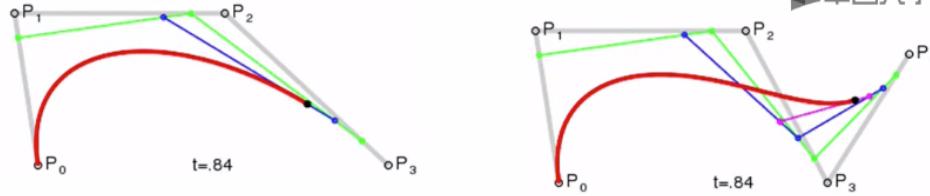
- 图中每个基函数在参数 t 的整个 $(0, 1)$ 的开区间范围内不为0。Bezier曲线不可能对曲线形状进行局部控制，如果改变任一控制点位置，整个曲线会受到影响。

把Bezier三次曲线多项式写成矩阵形式：

- $p(t) = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \quad t \in [0,1]$

$$= T \cdot M_{be} \cdot G_{be}$$

其中， M_{be} 是三次Bezier曲线系数矩阵，为常数； G_{be} 是4个控制点位置矢量。



7.3.2 Bernstein基函数的性质

- 正性(非负性)

$$\circ \quad B_{i,n}(t) = \begin{cases} = 0 & t = 0, 1 \\ > 0 & t \in (0, 1), i = 1, 2, \dots, n-1; \end{cases}$$

- 权性

基函数有n+1项，n+1个基函数的和加起来正好等于1。 $\sum_{i=0}^n B_{i,n}(t) = 1, t \in (0, 1)$ 。

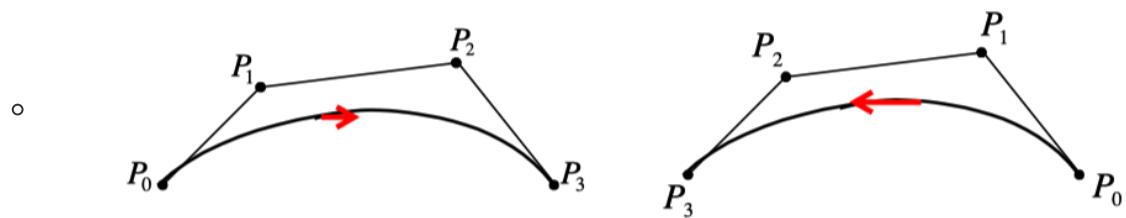
- 端点性质

$$\circ \quad B_{i,n}(0) = \begin{cases} 1 & (i = 0) \\ 0 & otherwise \end{cases}$$

$$\circ \quad B_{i,n}(1) = \begin{cases} 1 & (i = n) \\ 0 & otherwise \end{cases}$$

- 对称性

保持n次Bezier曲线控制多边形的顶点位置不变，而把次序颠倒过来，则此时曲线仍不变，只不过曲线的走向相反。



- 递推性

$$\circ \quad B_{i,n}(t) = (1-t)B_{i,n-1}(t) + tB_{i-1,n-1}(t)$$

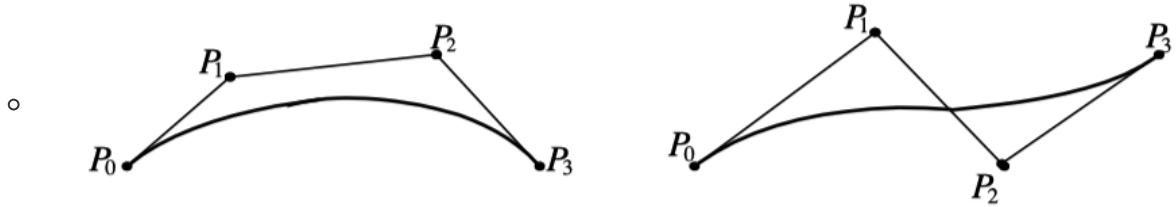
$$\circ \quad C_n^i = C_{n-1}^i + C_{n-1}^{i-1}$$

- 导函数

- $B'_{i,n}(t) = n[B_{i-1,n-1}(t) - B_{i,n-1}(t)]$, $i = 0, 1, \dots, n$
- 最大值
 - $B_{i,n}(t)$ 在 $t = \frac{i}{n}$ 处达到最大值
- 积分
 - $\int_0^1 B_{i,n}(t) dt = \frac{1}{n+1}$
- 降阶公式
 - $B_{i,n}(u) = (1-u)B_{i,n-1}(u) + uB_{i-1,n-1}(u)$
- 升阶公式
 - $B_{i,n}(n) = \frac{i+1}{n+1}B_{i+1,n+1}(t) + \frac{n+1-i}{n+1}B_{i-1,n-1}(t)$

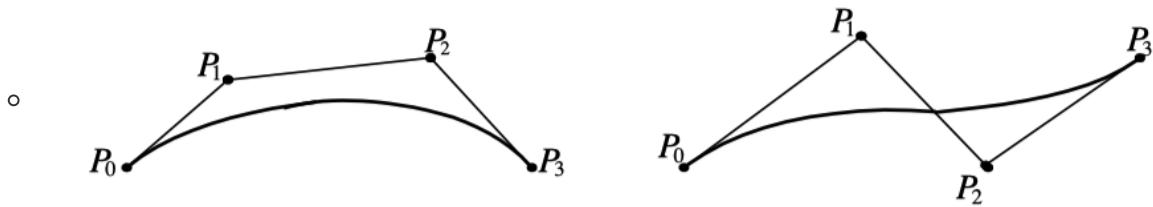
7.3.3 Bezier 曲线的性质

- 端点性质
 - 顶点 p_0 和 p_n 分别位于实际曲线段的起点和终点上。

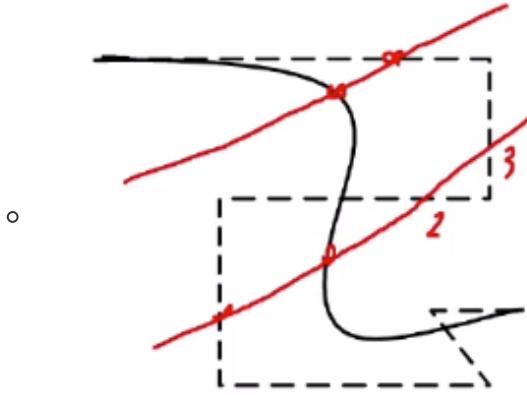


-
$$p(0) = \sum_{i=0}^n P_i B_{i,n}(0) = P_0 B_{0,n}(0) + P_1 B_{1,n}(0) + \dots + P_n B_{n,n}(0) = P_0$$
-
$$p(1) = \sum_{i=0}^n P_i B_{i,n}(1) = P_0 B_{0,n}(1) + P_1 B_{1,n}(1) + \dots + P_n B_{n,n}(1) = P_n$$

- 一阶导数
 - $p'(t) = n \sum_{i=1}^n (p_i - p_{i-1}) B_{i-1,n-1}(t)$
 - $p'(0) = n(p_1 - p_0)$, $p'(1) = n(p_n - p_{n-1})$
 - 证明 Bezier 曲线的起点和终点处的切线方向和特征多边形的第一条边及最后一条边的走向一致



- 几何不变性
 - 指某些几何特性不随坐标变换而变化的特性。Bezier 曲线的形状仅与控制多边形各顶点的相对位置有关，而与坐标系的选择无关。
- 变差缩减性
 - 若 Bezier 曲线的特征多边形是一个平面图形，则平面内任意直线与 $p(t)$ 的交点个数不多于该直线与其特征多边形的交点个数。这一性质叫变差缩减性质。
 - 此性质反映了 Bezier 曲线比其特征多边形的波动还小，也就是说 Bezier 曲线比特征多边形的折线更光滑。



第八章 曲线曲面 (二)

8.1 Bezier曲线生成算法

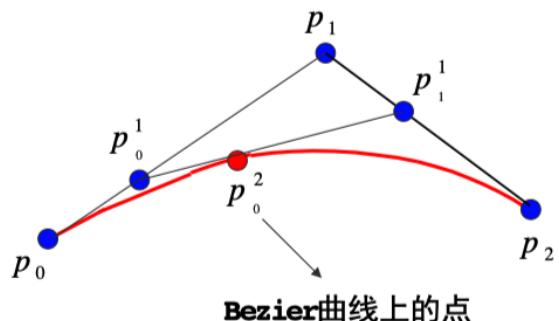
8.1.1 根据定义直接生成Bezier曲线

- 递归求出 C_n^i , $C_n^i = \frac{n!}{i!(n-i)!} = \frac{n-i+1}{i} C_n^{i-1}$, $n \geq i$ 。
- 将 $p(t)$ 表示成分量坐标形式直接计算即可。
- 缺点——计算量太大

8.1.2 Bezier曲线的递推算法

- Bezier曲线上的任一个点 $p(t)$, 都是其它相邻线段的同等比例 t 点处的连线, 再取同等比例 t 的点再连线, 一直取到最后那条线段的同等比例 t 处, 该点就是Bezier曲线上的点 $p(t)$ 。

$$\begin{aligned} P_0^1 &= (1 - t) P_0 + t P_1 \\ P_1^1 &= (1 - t) P_1 + t P_2 \\ P_0^2 &= (1 - t) P_0^1 + t P_1^1 \end{aligned}$$



•

t从0变到1, 第一、二式就分别表示控制二边形的第一、二条边, 它们是两条一次Bezier曲线。将一、二式代入第三式得:

$$P_0^2 = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$$

- 由 $(n+1)$ 个控制点 $P_i (i = 0, 1, \dots, n)$ 定义的 n 次 Bezier 曲线 P_0^n 可被定义为分别由前、后 n 个控制点定义的两条 $(n-1)$ 次 Bezier 曲线 P_0^{n-1} 与 P_1^{n-1} 的线性组合: $P_0^n = (1-t)P_0^{n-1} + tP_1^{n-1}$, $t \in [0, 1]$ 。

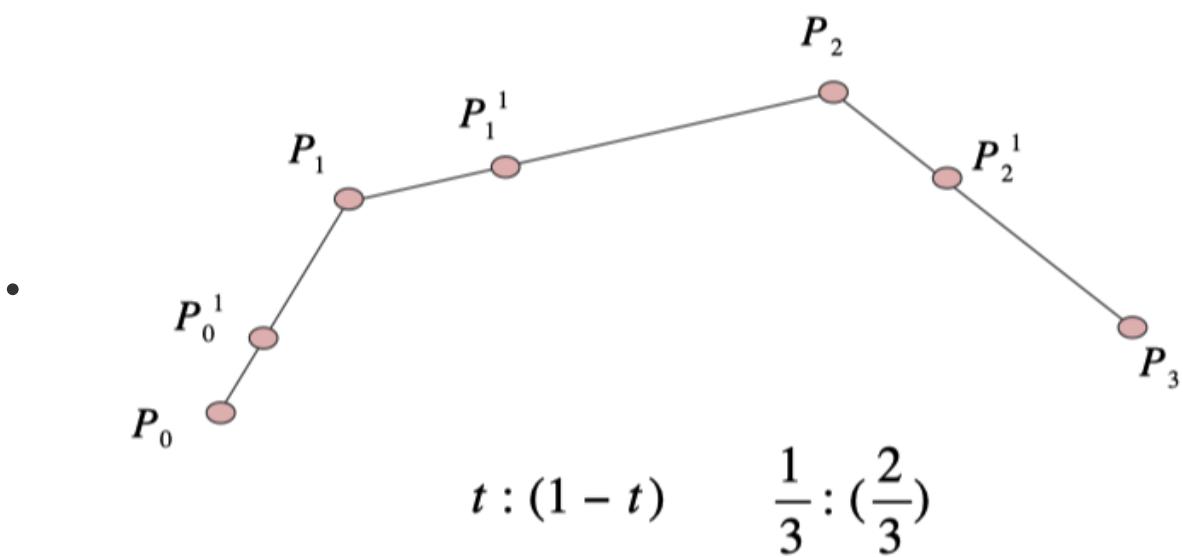
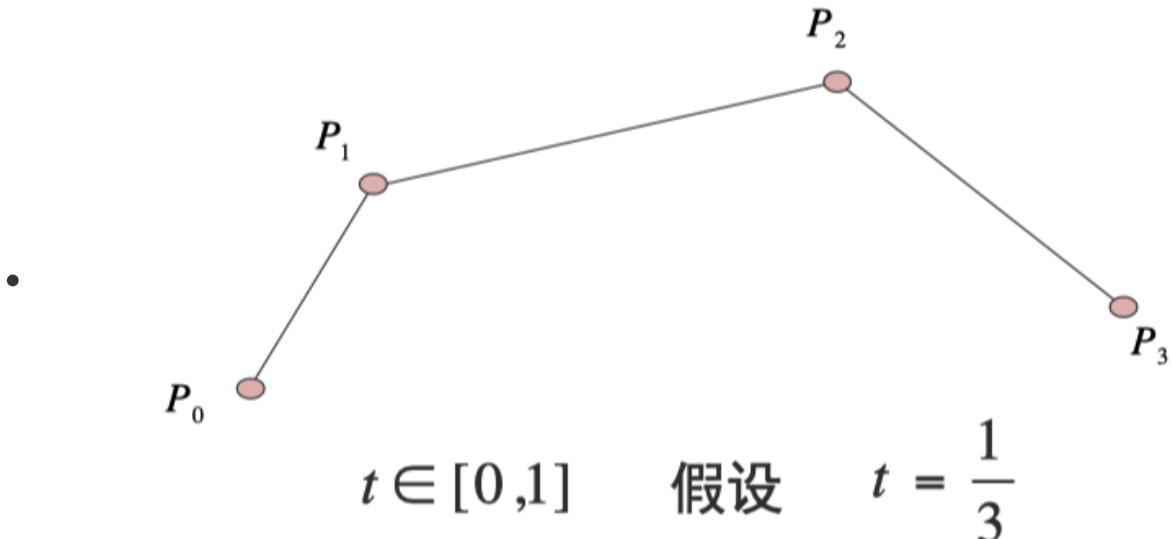
由此得到Bezier曲线的递推计算公式：

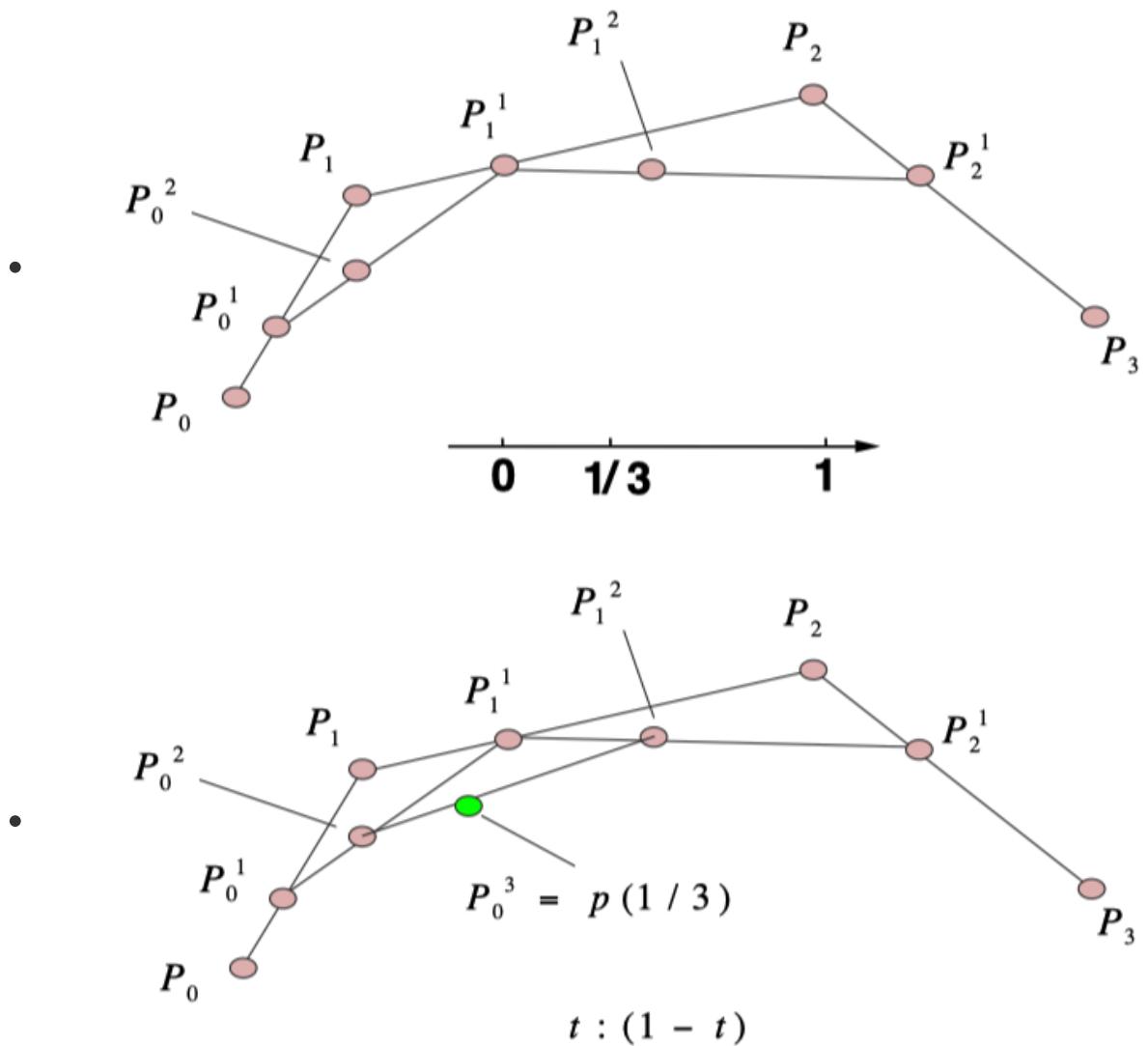
$$P_i^k = \begin{cases} P_i & k = 0 \\ (1-t)P_i^{k-1} + tP_{i+1}^{k-1} & k = 1, 2, \dots, n, i = 0, 1, \dots, n-k \end{cases}$$

这便是著名的de Casteljau算法。用这一递推公式，在给定参数下，求Bezier曲线上一点P(t)非常有效

- de Casteljau算法稳定可靠，直观简便，可以编出十分简洁的程序，是计算Bezier曲线的基本算法和标准算法。

8.1.3 de Casteljau算法几何作图

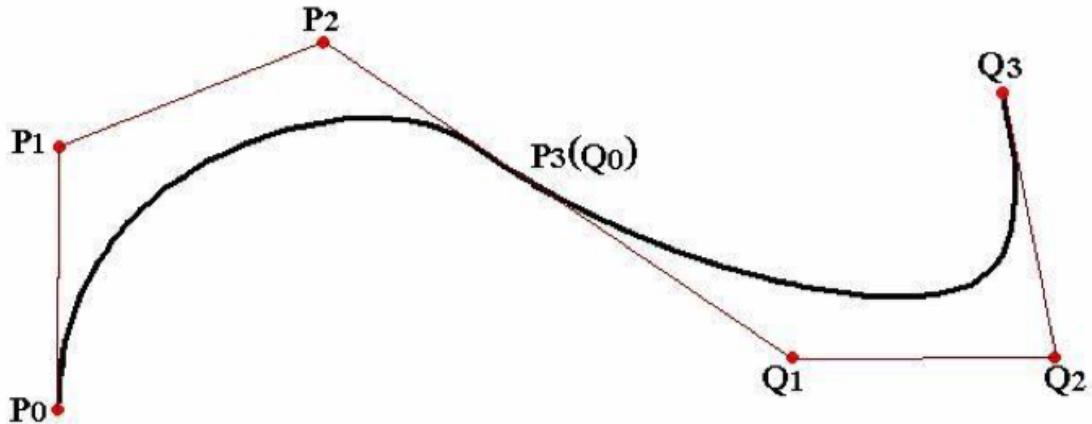




8.2 Bezier曲线的拼接及升降阶

8.2.1 Bezier曲线的拼接

- 几何设计中，有时会采用分段设计，然后将各段曲线相互连接起来，并在接合处保持一定的连续条件。
- G^0 连续，即 $P_n = Q_0$
- G^1 连续，即 P_{n-1} 、 $P_n = Q_0$ 、 Q_1 三点共线。因为Bezier曲线起点和终点处的切线方向和特征多边形的第一条边及最后一条边的走向一致。



8.2.2 Bezier曲线的升阶与降阶

- Bezier曲线的升阶
 - 升阶是指保持Bezier曲线的形状与方向不变，增加定义它的控制顶点数，即提高该Bezier曲线的次数。

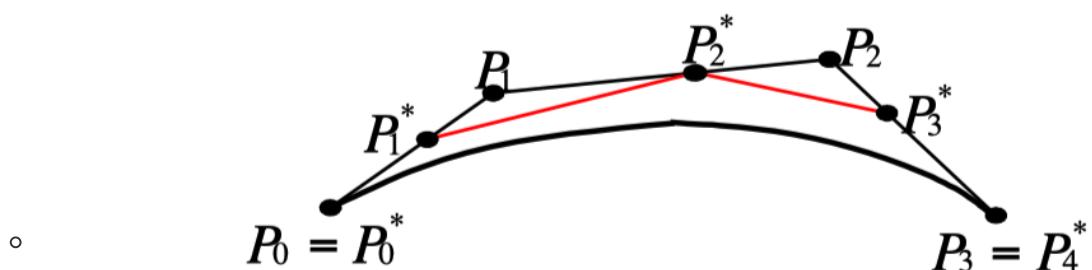
$$\sum_{i=0}^n C_n^i P_i t^i (1-t)^{n-i} = \sum_{i=0}^{n+1} C_{n+1}^i P_i^* t^i (1-t)^{n+1-i}$$

$$P_i^* C_{n+1}^i = P_i C_n^i + P_{i-1} C_{n-1}^{i-1}$$

化简即得：

$$P_i^* = \frac{i}{n+1} P_{i-1} + \left(1 - \frac{i}{n+1}\right) P_i \quad (i = 0, 1, \dots, n+1)$$

- P_i^* 在 P_i 和 P_{i-1} 的连线上， $P_0^* = P_0$, $P_{n+1}^* = P_n$ 。



新的多边形更加靠近曲线。在80年代初，有人证明如果一直升阶升下去的话，控制多边形收敛于这条曲线

- Bezier曲线的降阶
 - 降阶是升阶的逆过程， $a_0 + a_1 t + a_2 t^2 + a_3 t^3 = b_0 + b_1 t + b_2 t^2$ ，如果 a_3 不等于 0，想找一个二次多项式精确等于它是不可能的。如果一定要降下来，那只能近似逼近。

假定 P_i 是由 P_i^* 升阶得到，则由升阶公式有：

$$P_i = \frac{n-i}{n} P_i^* + \frac{i}{n} P_{i-1}^*$$

从这个方程可以导出两个递推公式：

$$P_i^* = \frac{n P_i - i P_{i-1}^*}{n-i} \quad i = 0, 1, \dots, n-1$$

$$P_{i-1}^* = \frac{n P_i - (n-i) P_i^*}{i} \quad i = n, n-1, \dots, 1$$

可以综合利用上面两个式子进行降解，但仍然不精确

- 升阶与降阶的重要性

- 它是CAD系统之间数据传递与交换的必需
- 它是系统中分段(片)线性逼近的需要。通过逐次降阶，把曲面化为直线平面，便于求交和曲面绘制。
- 它是外形信息压缩的需要。降阶处理以后可以减少存储的信息量。

8.3 Bezier曲面

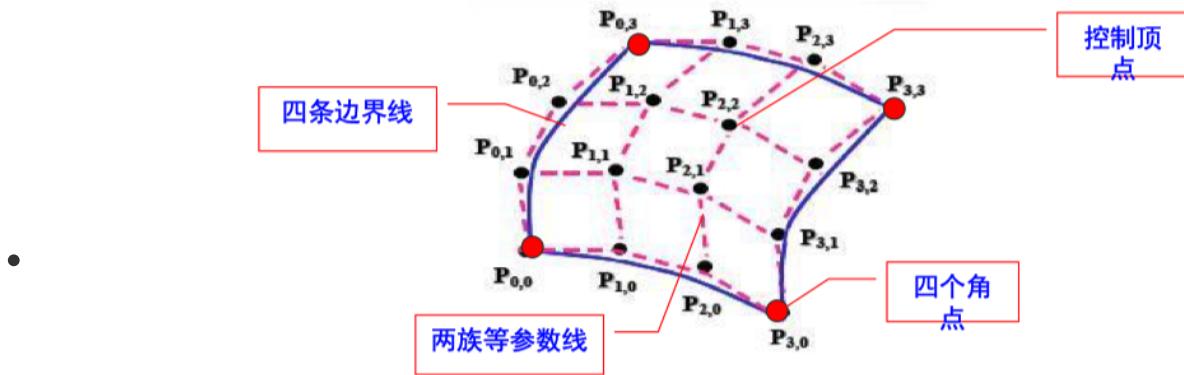
8.3.1 Bezier曲面的定义

- 设 $p_{i,j}$ ($i = 0, 1, \dots, n; j = 0, 1, \dots, m$) 为 $(n+1) * (m+1)$ 个空间点，则 $m * n$ Bezier曲面定义为：
$$P(u, v) = \sum_{i=0}^m \sum_{j=0}^n P_{i,j} B_{i,m}(u) B_{j,n}(v), \quad u, v \in [0, 1].$$
- $B_{i,m}(u) = C_m^i u^i (1-u)^{m-i}$, $B_{j,n}(v) = C_n^j v^j (1-v)^{n-j}$ 。依次用线段连接点列中相邻两点所形成的空间网格，称之为特征网格。

Bezier曲面的矩阵表示式是：

$$\bullet \quad P(u, v) = \begin{bmatrix} B_{0,n}(u), B_{1,n}(u), \dots, B_{m,n}(u) \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} & \cdots & P_{0m} \\ P_{10} & P_{11} & \cdots & P_{1m} \\ \cdots & \cdots & \cdots & \cdots \\ P_{n0} & P_{n1} & \cdots & P_{nm} \end{bmatrix} \begin{bmatrix} B_{0,m}(v) \\ B_{1,m}(v) \\ \cdots \\ B_{n,m}(v) \end{bmatrix}$$

上式的曲面称为 $m \times n$ 次的，在一般应用中， n, m 不大于4



角点位置：四个角点分别是其控制网格的四个点

$$P(0,0) = P_{0,0} \quad P(0,1) = P_{0,n}$$

$$P(1,0) = P_{m,0} \quad P(1,1) = P_{m,n}$$

- 四条边界线是Bezier曲线，固定一维不动即可求出四条边界线。

8.3.2 Bezier曲面性质

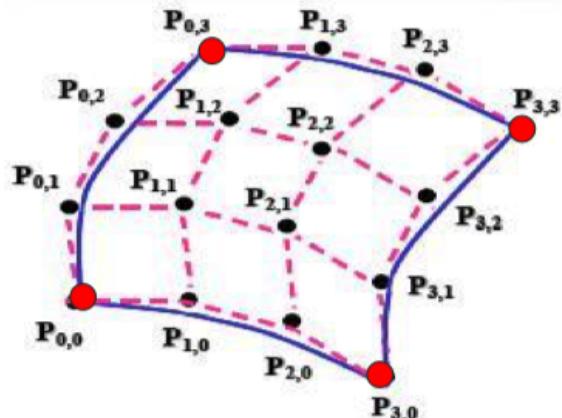
- Bezier曲面特征网格的四个角点正好是Bezier曲面的四个角点。

$$P(0,0) = P_{0,0}$$

$$P(1,0) = P_{m,0}$$

$$P(0,1) = P_{0,n}$$

$$P(1,1) = P_{m,n}$$



- Bezier曲面特征网格最外一圈顶点定义Bezier曲面的四条边界。
- 几何不变性
- 对称性
- 凸包性
- Bezier曲面片的拼接

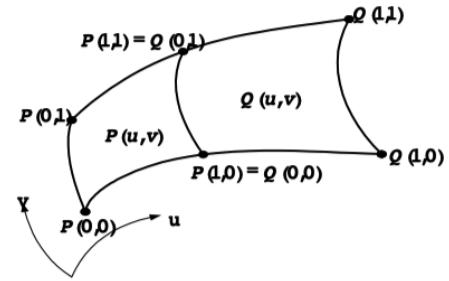
设两张 $m \times n$ 次Bezier曲面片：

$$P(u, v) = \sum_{i=0}^m \sum_{j=0}^n P_{ij} B_{i,m}(u) B_{j,n}(v)$$

$$Q(u, v) = \sum_{i=0}^m \sum_{j=0}^n Q_{ij} B_{i,m}(u) B_{j,n}(v)$$

○

分别由控制顶点 P_{ij} 和 Q_{ij} 定义。



Bezier曲面片的拼接

如果要求两曲面片达到 G^0 连续，则它们有公共的边界，即：

$$P(1, v) = Q(0, v)$$

于是有： $P_{ni} = Q_{0i}$ ， ($i = 0, 1, \dots, m$)

- 如果要求该公共边界达到 G^1 连续，则两曲面片在该边界上有公共的切平面，因此曲面的法向应当是跨界连续的，即： $Q_u(0, v) * Q_v(0, v) = a(v)P_u(1, v) * P_v(1, v)$ 。

8.3.3 递推算法 (曲面的求值)

**Bezier曲线的递推 (de Casteljau) 算法，可以推广到
Bezier曲面的情形**

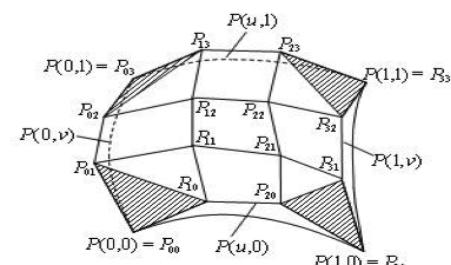
$$P_{ij} (i = 0, 1, \dots, m; j = 0, 1, \dots, n)$$

- $P(u, v) = \sum_{i=0}^{m-k} \sum_{j=0}^{n-l} P_{i,j}^{k,l} B_{i,m}(u) B_{j,n}(v) = L = P_{00}^{m,n}$ $u, v \in [0, 1]$

一条曲线可以表示成两条低一次曲线的组合，一张曲面可以表示成低一次的四张曲面的线性组合

$$P_{i,j}^{k,l} = \begin{cases} P_{ij} & (k = l = 0) \\ (1-u)P_{ij}^{k-1,0} + uP_{i+1,j}^{k-1,0} & (k = 1, 2, \dots, m; l = 0) \\ (1-v)P_{0,j}^{m,l-1} + vP_{0,j+1}^{m,l-1} & (k = m, l = 1, 2, \dots, n) \end{cases} \quad (1)$$

- 当按(1)式方案执行时，先以 u 参数值对控制网格 u 向的 $n+1$ 个
多边形执行曲线de Casteljau
算法， m 级递推后，得到沿 v 向
由 $n+1$ 个顶点 P_{0j}^{m0} 构成的多边形



再以 v 参数值对它执行曲线的de Casteljau算法， n 级递推以后，得到一个 P_{00}^{mn} ，即所求曲面上的点。一条曲线可以表示成2条低一次的曲线的线性组合，曲面可以表示成低一次的4张曲面的线性组合。

$$\begin{aligned}
 P(u, v) &= \sum_{i=0}^m \sum_{j=0}^n P_{ij} B_{i,m}(u) B_{j,n}(v) \\
 &= \sum_{i=0}^m \left(\sum_{j=0}^n P_{ij} B_{j,n}(v) \right) B_{i,m}(u)
 \end{aligned}$$

8.4 B样条曲线产生背景及定义

8.4.1 Bezier曲线缺点

- 一旦确定了特征多边形的顶点数 ($n+1$)，也就决定了曲线的阶次 (n 次)。如果阶数过高，则曲线中会有大量极值点，造成曲线波动现象明显。
- Bezier曲线或曲面的拼接比较复杂。
- Bezier曲线或曲面不能作局部修改，牵一发而动全身。因为每个Bernstein多项式在整个区间 $[0, 1]$ 上都有支撑，所以每个控制顶点对 0 到 1 之间的 t 值处的曲线都有影响。

8.4.2 B样条方法

- 采用分段连续多项式的方法。整条曲线用一个完整的表达形式，但内在的量是一段一段的。既克服了波动现象，曲线也是低次的，有统一的表达和统一的算法。
- 分段方法
 - $n + 1$ 个点，每两点之间构造一条多项式， $n + 1$ 个点有 n 个小区间。
 - 每个小区间构造一条三次多项式，变成了 n 段的三次多项式拼接在一起，段与段之间要达到 C^2 连续。

8.4.3 B样条的递推定义和性质

- B样条曲线的数学表达式
 - $P(u) = \sum_{i=0}^n P_i B_{i,k}(u)$, $u \in [u_{k-1}, u_{n+1}]$ 。 P_i ($i = 0, 1, \dots, n$) 是控制多边形的顶点。
 - $B_{i,k}(u)$ 称为 k 阶 ($k - 1$ 次) B 样条基函数， k 是刻画次数， $k \in [2, n + 1]$ 。对 Bezier 曲线来说，阶数 = 次数；对 B 样条来说，阶数是次数+1。
 - B 样条基函数是一个称为节点矢量的非递减的参数 u 的序列所决定的 k 阶分段多项式，这个序列称为节点向量。

8.4.4 de Boor-Cox 递推定义

- 原理：对于 k 阶 ($k - 1$) 次的 B 样条基函数，构造一种递推的公式，由 0 次构造 1 次，1 次构造 2 次，2 次构造 3 次 ... 依次类推。

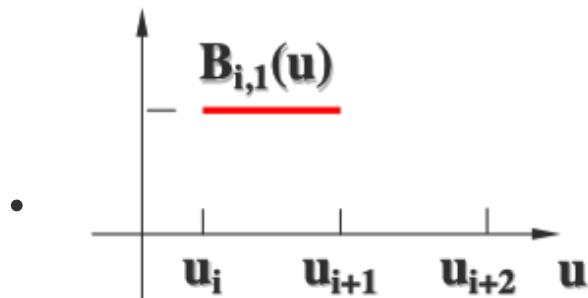
$$B_{i,1}(u) = \begin{cases} 1 & u_i < u < u_{i+1} \\ 0 & \text{Otherwise} \end{cases}$$

并约定: $\frac{0}{0} = 0$

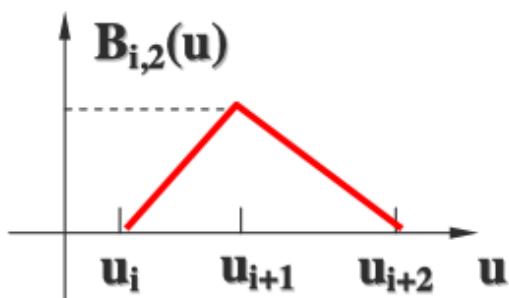
$$B_{i,k}(u) = \frac{u - u_i}{u_{i+k-1} - u_i} B_{i,k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} B_{i+1,k-1}(u)$$

- 该递推公式表明: 若确定第*i*个*k*阶B样条 $B_{i,k}(u)$, 需要用到 $u_i, u_{i+1}, \dots, u_{i+k}$ 共*k+1*个节点, 称区间 $[u_i, u_{i+k}]$ 为 $B_{i,k}(u)$ 的支承区间

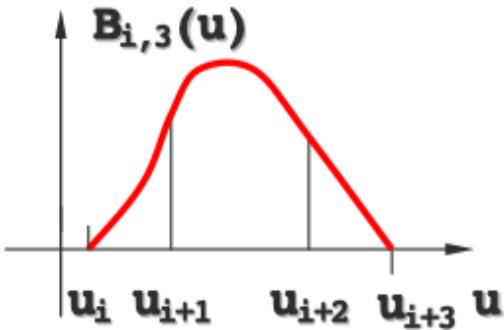
- 曲线方程中, $n+1$ 个控制顶点 P_i ($i = 0, 1, \dots, n$), 要用到 $n+1$ 个*k*阶B样条 $B_{i,k}(u)$ 。它们支撑区间并集定义了这一组B样条基的节点矢量 $U = [u_0, u_1, \dots, u_{n+k}]$ 。



(1阶B样条)



(2阶B样条)



$B_{i,3}(u)$ 的图像

8.4.5 B样条基函数定义区间及节点向量

- K 阶 B 样条对应节点向量数

$$\circ \quad B_{i,1}(u) = \begin{cases} 1 & u_i < u < u_{i+1} \\ 0 & \text{Otherwise} \end{cases} \quad B_{i,k}(u) = \frac{u - u_i}{u_{i+k-1} - u_i} B_{i,k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} B_{i+1,k-1}(u)$$

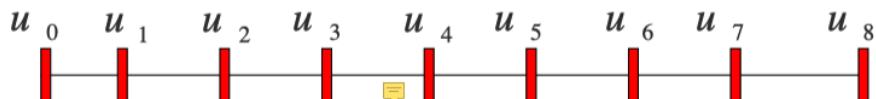
- 对于 $B_{i,1}$ (1阶0次基函数) 来说，涉及到 u_i 到 u_{i+1} 一个区间，即一阶的多项式涉及一个区间两个节点。
 $B_{i,2}$ 是由 $B_{i,1}$ 和 $B_{i+1,1}$ 组成，因此 $B_{i,2}$ 涉及 2 个区间 3 个节点； $B_{i,3}$ 涉及 3 个区间 4 个节点...， $B_{i,k}$ 涉及 k 个区间 $k+1$ 个节点。

- B 样条函数定义区间

$$P(u) = \sum_{i=0}^n P_i B_{i,k}(u) \quad B_{i,1}(u) = \begin{cases} 1 & u_i < u < u_{i+1} \\ 0 & \text{Otherwise} \end{cases}$$

$$\circ \quad u \in [u_{k-1}, u_{n+1}] \quad B_{i,k}(u) = \frac{u - u_i}{u_{i+k-1} - u_i} B_{i,k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} B_{i+1,k-1}(u)$$

$$U = \{u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8\}$$



$$\circ \quad n=4 \quad k=4 \quad B_{i,k}(u) = \frac{u - u_i}{u_{i+k-1} - u_i} B_{i,k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} B_{i+1,k-1}(u)$$

第一项是 $p_0 B_{0,4}(u)$ ，涉及哪些节点？ u_0 到 u_4

第二项是 $p_1 B_{1,4}(u)$ ，涉及哪些节点？ u_1 到 u_5

第五项是 $p_4 B_{4,4}(u)$ ，涉及哪些节点？ u_4 到 u_8

- “阶数+顶点”等于节点向量的个数。
 - **B** 样条曲线定义

$$P(u) = \sum_{i=0}^n P_i B_{i,k}(u) \quad B_{i,1}(u) = \begin{cases} 1 & u_i < x < u_{i+1} \\ 0 & \text{Otherwise} \end{cases}$$

$$u \in [u_{k-1}, u_{n+1}] \quad B_{i,k}(u) = \frac{u - u_i}{u_{i+k-1} - u_i} B_{i,k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} B_{i+1,k-1}(u)$$
 - **u_i**是节点值, **U= (u₀,u₁,…u_{n+k})** 构成了**k**阶 (**k-1**次) **B**样条函数的节点矢量
 - **B**样条曲线所对应的节点向量区间: $u \in [u_{k-1}, u_{n+1}]$
-

8.5 B样条曲线性质及类型划分

8.5.1 B样条基函数的主要性质

- 局部支承性
 - $B_{i,k}(u) \begin{cases} \geq 0 & u \in [u_i, u_{i+k}] \\ = 0 & \text{otherwise} \end{cases}$ 而Bezier在整个区间非0
 - 反过来, 对每个区间 (u_i, u_{i+k}) , 至多只有 k 个基函数在其上非零。
- 权性
 - $\sum_{i=0}^n B_{i,k}(u) = 1, u \in [u_{k-1}, u_{n+1}]$
- 连续性
 - $B_{i,k}(u)$ 在 r 重节点处的连续阶不低于 $k - 1 - r$
- 分段参数多项式
 - $B_{i,k}(u)$ 在每个长度非零的区间 $[u_i, u_{i+1}]$ 上都是次数不高于 $k - 1$ 的多项式, 它在整个参数轴上是分段多项式。

8.5.2 B样条函数的主要性质

- 局部性
 - k 阶B样条曲线上的一点至多与 k 个控制顶点有关, 与其它控制顶点无关。
 - 移动曲线的第*i*个控制顶点 P_i , 至多影响到定义在区间上那部分曲线的形状, 对曲线其余部分不发生影响。
- 变差缩减性
 - 与Bezier的变差缩减性一致
- 几何不变性
 - B样条曲线的形状和位置与坐标系的选择无关

- 凸包性

- B样条曲线落在 P_i 构成的凸包之中。其凸包区域小于或等于同一组控制顶点定义的Bezier曲线凸包区域。
- 该性质导致顺序k+1个顶点重合时，由这些顶点定义的k次B样条曲线段退化到这一个重合点。顺序k+1个顶点共线时，由这些顶点定义的k次B样条曲线为直线。

8.5.3 B样条曲线类型的划分

- 均匀B样条曲线

- 当节点沿参数轴均匀等距分布，即 $u_{i+1} - u_i = \text{常数} > 0$ 时，表示均匀 B 样条函数。
- 均匀B样条的基函数呈周期性。即给定n和k，所有的基函数有相同形状。每个后续基函数仅仅是前面基函数在新位置上的重复。
- $B_{i,k}(u) = B_{i+1,k}(u + \Delta u) = B_{i+2,k}(u + 2\Delta u)$
- 举例
 - 假定有四个控制点，取参数值 $n = 3, k = 3$ ，则 $n + k = 6$ ， $u = (0, 1, 2, 3, 4, 5, 6)$ 。

$$P(u) = \sum_{i=0}^n P_i B_{i,k}(u) \quad B_{i,1}(u) = \begin{cases} 1 & u_i < u < u_{i+1} \\ 0 & \text{Otherwise} \end{cases}$$

$$B_{i,k}(u) = \frac{u - u_i}{u_{i+k-1} - u_i} B_{i,k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} B_{i+1,k-1}(u)$$

- $B_{0,1}(u) = \begin{cases} 1 & 0 \leq u < 1 \\ 0 & \text{其它} \end{cases} \quad u = (0, 1, 2, 3, 4, 5, 6)$

$$B_{0,2}(u) = uB_{0,1}(u) + (2-u)B_{1,1}(u) = uB_{0,1}(u) + (2-u)B_{1,1}(u)$$

$$= \begin{cases} u & 0 \leq u < 1 \\ 2 - u & 1 \leq u < 2 \end{cases}$$

$$B_{0,3}(u) = \frac{u}{2} B_{0,2}(u) + \frac{3-u}{2} B_{1,2}(u-1)$$

$$= \begin{cases} \frac{1}{2}u^2 & 0 \leq u < 1 \\ \frac{1}{2}u(2-u) + \frac{1}{2}(u-1)(3-u) & 1 \leq u < 2 \\ \frac{1}{2}(3-u)^2 & 2 \leq u < 3 \end{cases}$$

$$B_{1,3}(u) = \begin{cases} \frac{1}{2}(u-1)^2 & 1 \leq u < 2 \\ \frac{1}{2}(u-1)(3-u) + \frac{1}{2}(u-2)(4-u) & 2 \leq u < 3 \\ \frac{1}{2}(4-u)^2 & 3 \leq u < 4 \end{cases}$$

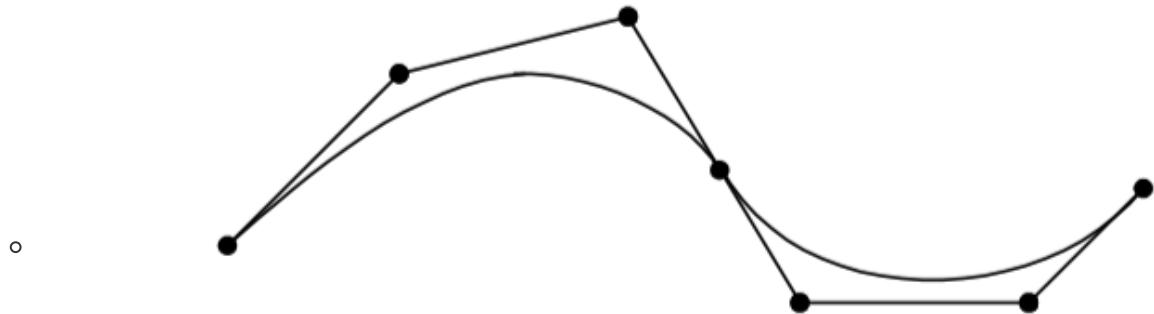
$$B_{3,3}(u) = \begin{cases} \frac{1}{2}(u - 3)^2 & 3 \leq u < 4 \\ \frac{1}{2}(u - 3)(5 - u) + \frac{1}{2}(u - 4)(6 - u) & 4 \leq u < 5 \\ \frac{1}{2}(6 - u)^2 & 5 \leq u < 6 \end{cases}$$

- 准均匀B样条曲线

- 与均匀B样条曲线的差别在于两端节点具有重复度k，这样的节点矢量定义了准均匀的B样条基函数。均匀: $u = (0, 1, 2, 3, 4, 5, 6)$; 准均匀: $u = (0, 0, 0, 1, 2, 3, 4, 5, 5, 5)$ 。
- 均匀B样条曲线没有保留Bezier曲线端点的几何性质，采用准均匀的B样条曲线可以解决这个问题。

- 分段Bezier曲线

- 节点矢量中两端节点具有重复度k，所有内节点重复度为k-1，这样的节点矢量定义了分段的Bernstein基函数。



三次分段Bezier曲线

- B样条曲线用分段Bezier曲线表示后，各曲线段就具有了相对的独立性。缺点是增加了定义曲线的数据，控制顶点数及节点数。
- 非均匀B样条曲线
 - 当节点沿参数轴的分布不等距，即 $u_{i+1} - u_i$ 不等于常数时，表示非均匀B样条函数。

8.6 B样条曲面

给定参数轴和v的节点矢量

$$U = [u_0, u_1, \dots, u_{m+p}]$$

$$V = [v_0, v_1, \dots, v_{n+q}]$$

p×q阶B样条曲面定义如下：

$$P(u, v) = \sum_{i=0}^m \sum_{j=0}^n P_{i,j} N_{i,p}(u) N_{j,q}(v)$$

$P_{i,j}$ 构成一张控制网格，称为B样条曲面的特征网格

- $N_{i,p}(u)$ 和 $N_{j,q}(v)$ 是B样条基，分别由节点矢量u和v按 deBoor-Cox递推公式决定

第九章 真实感图形学

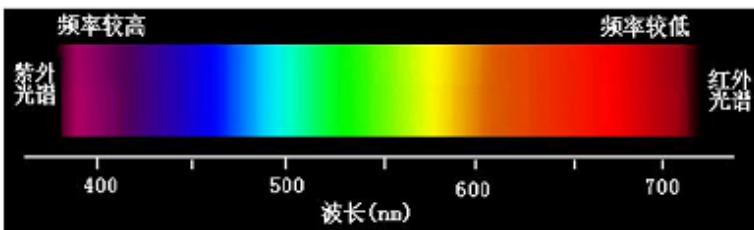
9.1 真实感图形学概述

- 真实感图形学的三部曲
 - 建模：建立三维场景
 - 消隐：消隐解决物体深度的显示及确认物体内的相互关系
 - 渲染：消隐后，在可见面上进行敏感光泽的处理，然后绘制
- 本章重点词汇
 - incident light, 入射光
 - diffuse, 扩散, 漫反射
 - specular, 缩写spec, 镜面的
 - refracted, 折射的
 - transparent, 透明的
 - Distribution, 分布
 - Intensity, 光强

9.2 颜色模型

9.2.1 颜色及颜色模型概述

- 颜色是人的视觉系统对可见光的感知结果，感知到的颜色由光波的波长决定。
- 人眼对于颜色的观察和处理是一种生理和心理现象，其机理还没有完全搞清楚。
- 视觉系统能感觉的波长范围为380~780nm。

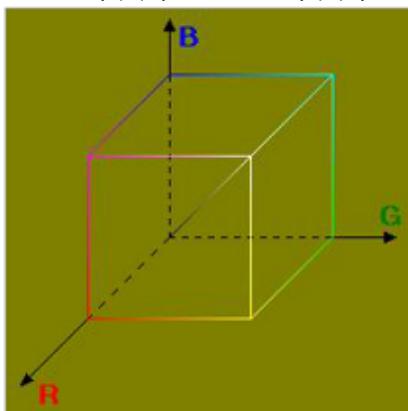


- 颜色模型（颜色空间），是表示颜色的一种数学方法，人们用它来标定颜色。通常用三个参数表示。
- 几乎所有的颜色模型都从RGB颜色模型导出。
- 目前现有颜色模型还没有一个完全符合人的视觉感知特性、颜色本身的物理特性或发光物体和光反射物体的特性。

9.2.2 RGB颜色工业模型

- 如图所示，单位立方体中的三个角对应红色(R)、绿色(G)、蓝色(B)三基色，而其余三个角分别对应于三基色的补色——青色(C)、黄色(Y)、品红色(M)
- 从RGB单位立方体的原点即黑色(0,0,0)到白色顶点(1,1,1)的主对角线被称为灰度线，线上所有的点具有相等的

分量，产生灰度色调。



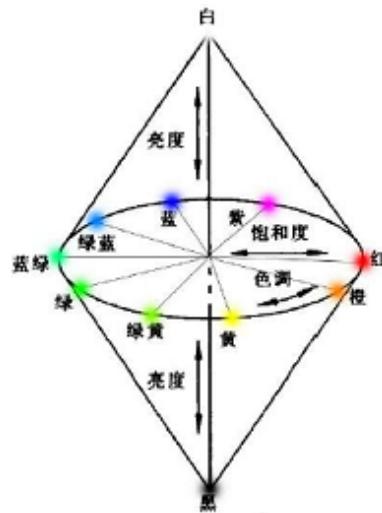
9.2.3 其他颜色工业模型

- 主要用于彩色电视信号传输标准，主要有YIQ、YUV、YC_bC_r彩色模型。
- 三种彩色模型中，Y分量均代表黑白亮度分量，其余分量用于显示彩色信息。只需利用Y分量进行图像显示，彩色图像就转为灰度图像。

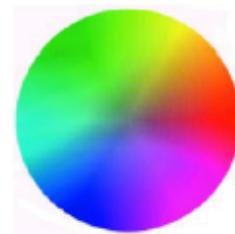
9.2.4 颜色视觉模型

- 以上彩色模型是从色度学或硬件实现的角度提出的
- 但用色调(Hue)、饱和度(Saturation, 也称彩度)、亮度(Illumination)三要素来描述彩色空间能更好地与人的视觉特性相匹配。
- 颜色的三个基本属性(也称人眼视觉三要素)
 - 色调(Hue): 由物体反射光线中占优势的波长决定的，是彩色互相区分的基本特性。
 - 饱和度(Saturation)或彩度: 彩色的深浅程度，它取决于彩色中白色的含量。饱和度越高，彩色越深，白色光越少。
 - 亮度(Illumination): 光波作用于感受器所发生的效应，它取决于物体的反射系数。反射系数越大，物体

亮度越大。



- HSI 彩色模型是截面为三角形或圆形的锥体模型。



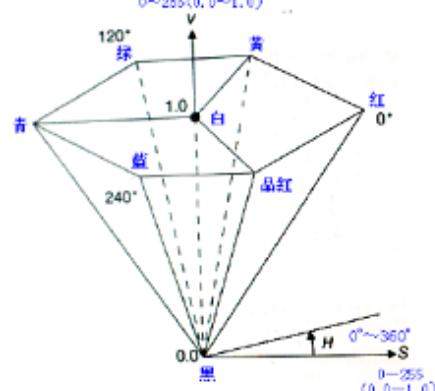
- H, 色调, 颜色的外观, 用角度表示: 如赤橙黄绿青蓝紫



- S, 饱和度, 分为 低(0%~20%), 不管色调如何而产生灰色 中(40%~60%), 产生柔和的色泽(pastel) 高(80%~100%), 产生鲜艳的颜色(vivid color)



- I, 光照, 颜色的量度, 取值范围从0%(黑)~100%(最亮)



- HSV彩色模型 (Hue, Saturation, Value)

- HSL采用亮度L(lightness)、HSV采用明度V(value)作为坐标。

9.3 简单光照模型

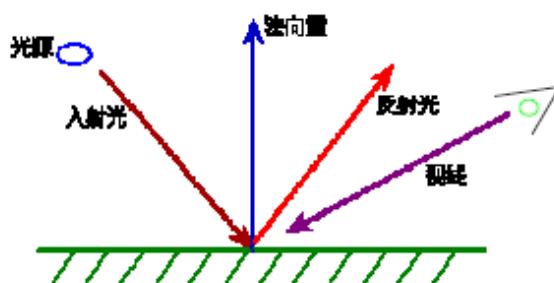
光照明模型：illumination model，模拟物体表面的光照明物理现象的数学模型。简单光照明模型只考虑光源对物体的直接光照。

9.3.1 光照模型的发展

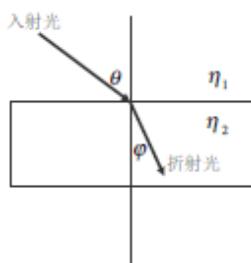
- 1967年，Wylie等人第一次在显示物体时加进光照效果，认为光强与距离成反比。
- 1970年，Bouknight提出第一个光反射模型：Lambert漫反射+环境光（第一个可用的光照模型）。
- 1971年，Gouraud提出漫反射模型加插值的思想（漫反射的意思是光强主要取决于入射光的强度和入射光与法线的夹角）。
- 1975年，Phong提出图形学中第一个最有影响的光照明模型。在漫反射模型的基础上加进了高光项。

9.3.2 光照背景物理知识

- 反射定律：入射角等于反射角，而且反射光线、入射光线与法向量在同一平面上。

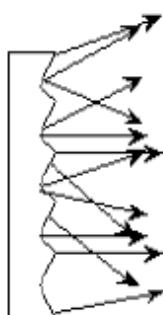


- 折射定律：折射线在入射线与法线构成的平面上，折射角与入射角满足： $\frac{\eta_1}{\eta_2} = \frac{\sin\varphi}{\sin\theta}$ ，参数是折射率和折射角



- 能量守恒： $I_i = I_d + I_s + I_t + I_v$ ，入射光强=漫反射光强+镜面反射光强+折射光强+吸收光强

- 漫反射光：光线射到物体表面上后（比如泥塑物体的表面，没有一点镜面效果），光线会沿着不同的方向等量的散射出去的现象。漫反射光均匀向各方向传播，与视点无关，它是由表面的粗糙不平引起的。



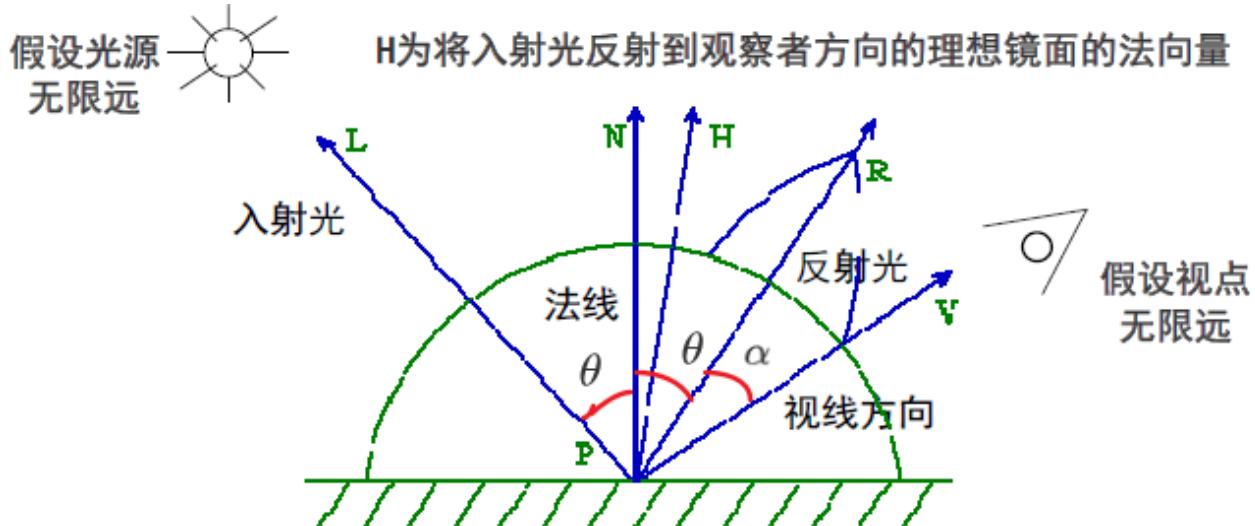
- 镜面反射光：一束光照射到一面镜子上或不锈钢的表面，光线会沿着反射光方向全部反射出去，这种叫

镜面反射光。

- 折射光：比如水晶、玻璃等，光线会穿过去一直往前走。
- 吸收光：比如冬天晒太阳会感觉到温暖，这是因为吸收了太阳光。

9.3.3 Phong光照模型

Phong光照模型 = 环境光+漫反射光+镜面反射光



单一光源照射下Phong光照模型： $I = I_a K_a + I_p K_d \cos\theta + I_p K_s \cos^n \alpha$

常用形式： $I = I_a K_a + I_p K_d (L \cdot N) + I_p K_s (R \cdot V)^n$ ， L是已知光源， V是视线的方向， R是反射光可计算出来， N是物体表面的法向可计算

- 环境光， $I_{ambient} = I_a K_a$ ， I_a 环境光强度， K_a 环境光反射系数 邻近各物体所产生的光的多次反射最终达到平衡时的一种光。可近似认为同一环境下的环境光，其光强分布是均匀的。
- 漫反射光， $I_{diffuse} = I_p K_d (L \cdot N)$ ， $\cos\theta = L \cdot N$ ， I_p 点光源光强， K_d 物体表面漫反射率 光照射到比较粗糙的物体表面，物体表面某点的明暗程度不随观测者的位置变化，这种等同地向各个方向散射的现象称为光的漫反射。漫反射光强近似服从Lambert定律
- 镜面反射光， $I_{spec} = I_p K_s (R \cdot V)^n$ ， $\cos\alpha = R \cdot V$ ， I_p 点光源光强， K_s 物体表面某点的高光亮稀疏， n 镜面反射指数，取值1-2000，反映光滑程度 光照射到相当光滑的物体表面，产生镜面反射光，其特点是在光滑表面会产生高光区域。

$$\text{结合RGB颜色模型后, } \begin{cases} I_r = I_{ar} K_{ar} + I_{pr} K_{dr} (L \cdot N) + I_{pr} K_{sr} (R \cdot V)^n \\ I_g = I_{ag} K_{ag} + I_{pg} K_{dg} (L \cdot N) + I_{pg} K_{sg} (R \cdot V)^n \\ I_b = I_{ab} K_{ab} + I_{pb} K_{db} (L \cdot N) + I_{pb} K_{sb} (R \cdot V)^n \end{cases}$$

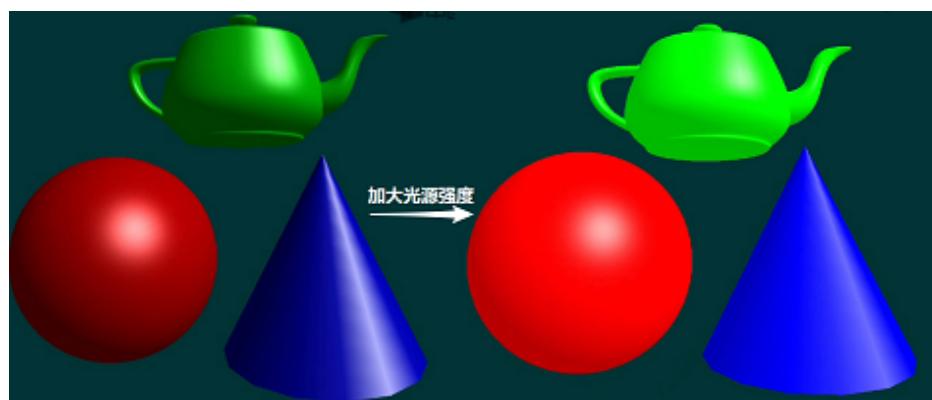
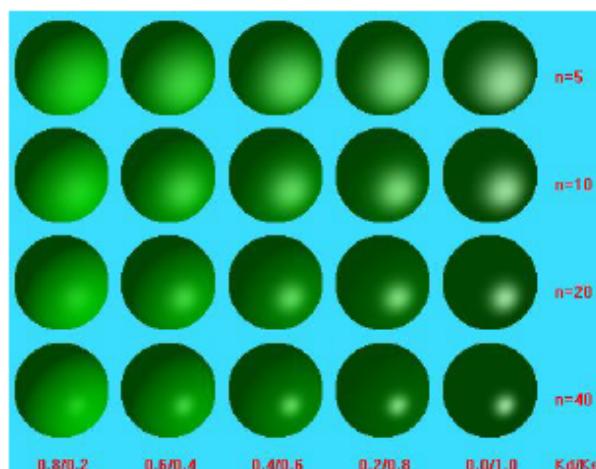
Phong模型扫描线算法：其中 $(r, g, b) = k_a(r_{pa}, g_{pa}, b_{pa}) + \sum(k_d(r_{pd}, g_{pd}, b_{pd}) \cos\theta + k_s(r_{ps}, g_{ps}, b_{ps}) \cos^n \alpha)$

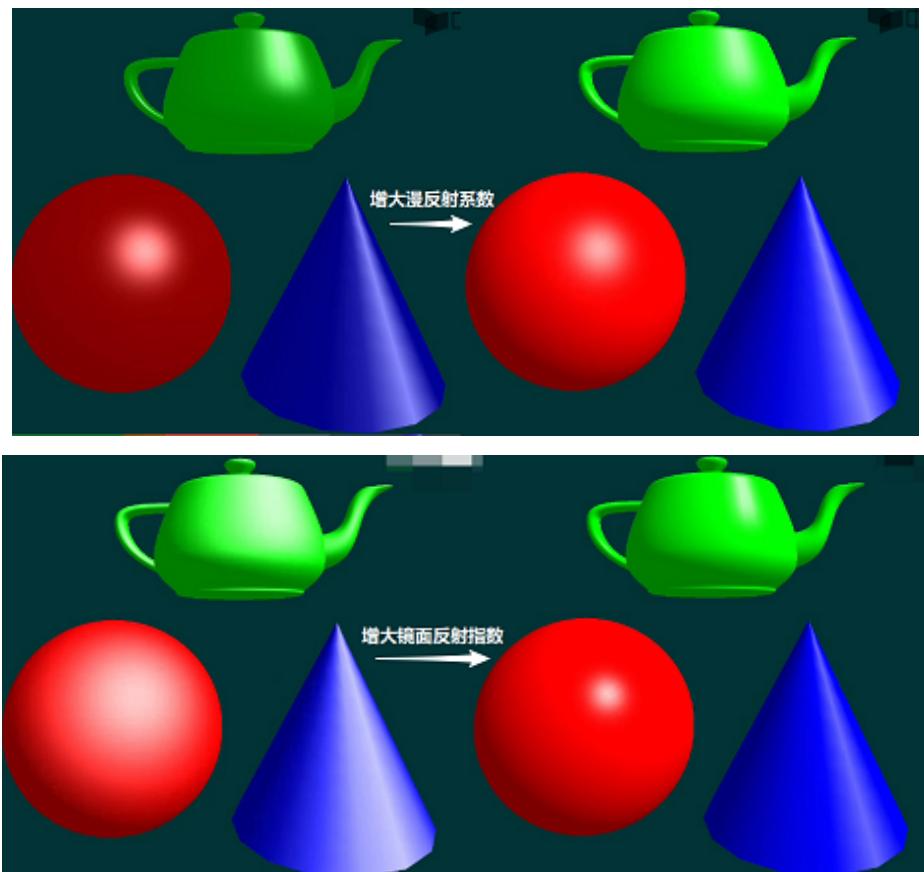
```

1   for 屏幕上每一条扫描线y do
2     begin
3       将数组Color初始化成为y扫描线的背景颜色值
4       for y扫描线上的每一可见区间段s中的每个点(x,y) do
5         begin
6           设(x,y)对应的空间可见点为P
7           求出P点处的单位法向量N、P点的单位入射光向量L、单位视线向量V
8           求出L在P点的单位镜面反射向量R
9           (r,g,b) = 代码块外的那个公式
10          置Color(x,y) = (r,g,b)
11        end
12      显示Color
13    end

```

Phong示例：





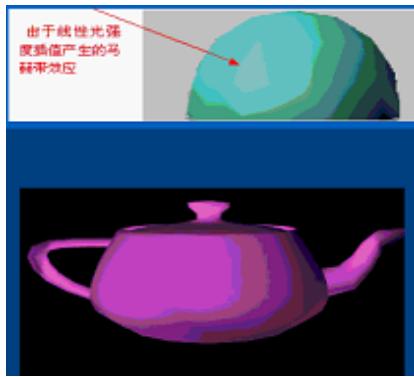
Phong总结：

- 它是真实感图形学中提出的第一个有影响的光照明模型，生成图象的真实度已经达到可以接受的程度。
- 其模拟光从物体表面到观察者眼睛的反射。尽管这种方法符合一些基本的物理法则，但它更多的是基于对现象的观察，所以被看成是一种经验式的方法。
- 实际应用中，这个经验模型还有以下问题
 - 显示出的物体像塑料，无质感变化
 - 没有考虑物体间相互反射光
 - 镜面反射颜色与材质无关
 - 镜面反射入射角大，会产生失真现象

9.4 增量式光照模型

9.4.1 光暗处理

- 光暗处理的必要性
 - 三维物体通常用多边形（三角形）来近似模拟。
 - 由于每一个多边形的法向一致，因而多边形内部的象素颜色相同，而且在不同法向的多边形邻接处，光强突变，使具
 - 有不同光强的两个相邻区域之间的光强不连续性(马赫带效应)。



用简单光照模型显示



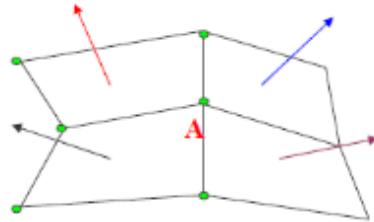
用增量式光照模型显示

- 明暗处理

- 思想：每一个多边形的顶点处计算出光照强度或参数，然后在各个多边形内部进行均匀插值
- 方法
 - Gouraud明暗处理（双线性光强插值算法）
 - Phong明暗处理（双线性法向插值算法）

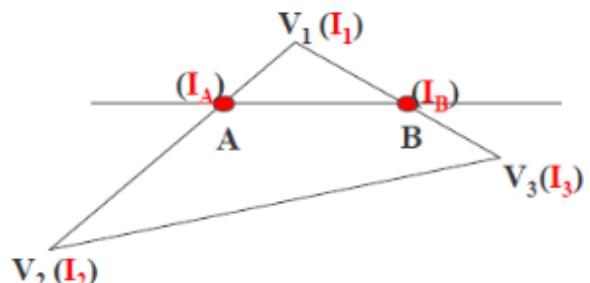
9.4.2 Gouraud明暗处理

- 核心：双线性光强插值
- 算法过程
 - 计算多边形顶点的平均法向。
 - 与某个顶点相邻的所有多边形的法向平均值近似作为该顶点的近似法向量，顶点A相邻的多边形有k个，它的法向量计算为： $N_\alpha = \frac{1}{k}(N_1 + N_2 + \dots + N_k)$
 - 用Phong光照模型计算顶点的光强
 - Phong光照模型出现前，采用如下光照模型计算： $I = I_a K_a + I_p K_d \frac{(L \cdot N_\alpha)}{r+k}$



- 插值计算离散边上点的光强

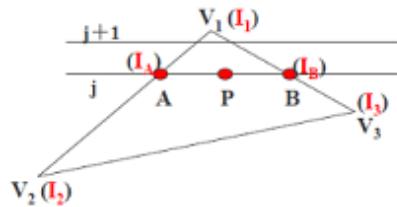
- $I_A = uI_1 + (1 - u)I_2, u = \frac{AV_2}{V_1V_2}$
- $I_B = vI_1 + (1 - v)I_3, v = \frac{BV_3}{V_1V_3}$



- 插值计算多边形内域中各点的光强

- $I_p = tI_A + (1 - t)I_B, t = \frac{PB}{AB}$ 求任一点的光强需要进行两次插值计算

- 扫描线增量思想的优化



- 离散边上个点的光强

$$I_{A,j+1} = I_{A,j} + \Delta I_A$$

$$I_{B,j+1} = I_{B,j} + \Delta I_B$$

$$\Delta I_A = \frac{I_1 - I_2}{y_1 - y_2}$$

$$\Delta I_B = \frac{I_1 - I_3}{y_1 - y_3}$$

- 扫描线内部

$$I_{i+1,p} = I_{i,p} + \Delta I_p$$

$$\Delta I_p = \frac{I_B - I_A}{x_B - x_A}$$

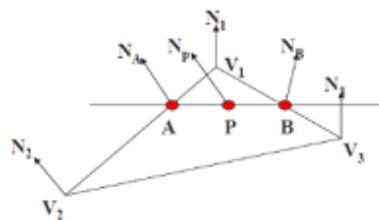
9.4.3 Phong明暗处理

- Gouraud明暗处理的不足

- 不能有镜面反射光（高光）
- 双线性插值是把能量往四周均匀，平均的结果就是光斑被扩大了，本来没光斑的地方插值后反而出现了光斑。
- Phong明暗处理以计算量、时间为代价，引入镜面反射，解决高光问题

- 算法过程

- 计算每个多边形顶点处的平均单位法矢量（与Gouraud明暗处理方法的第一步相同）
- 用双线性插值方法求得多边形内部各点的法矢量 N_A 由 N_1 和 N_2 线性插值而来 N_B 由 N_1 和 N_3 线性插值而来 N_P 由 N_A 和 N_B 线性插值而来
- 按光照模型确定多边形内部各点的光强



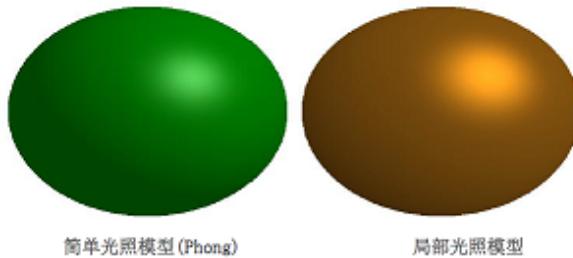
- 两种增量式光照模型比较

Phong方法	Gouraud方法
产生的效果高光明显	效果并不明显
高光多位于多边形内部	多边形内部无高光
明暗变化缺乏层次感	光强度变化均匀，与实际效果更接近
计算量大	计算量小

- 增量式光照模型的不足
 - 物体边缘轮廓是折线段而非光滑曲线
 - 等间距扫描线会产生不均匀效果
 - 插值结果取决于插值方向

局部光照模型

简单光照模型是一个比较粗糙的经验模型，不足之处在于镜面反射项与物体表面的材质无关



9.5 局部光照模型

9.5.1 局部光照模型概述

- 仅处理光源直接照射物体表面，不处理物体间反射的影响
- 相对于简单光照模型的优点
 - 基于入射光能量导出的光辐射模型
 - 反映表面的粗糙度对反射光强的影响
 - 高光颜色与材料的物理性质有关
 - 改进入射角很大时的失真现象
 - 考虑了物体材质的影响，可以模拟磨光的金属光泽

9.5.2 理论基础

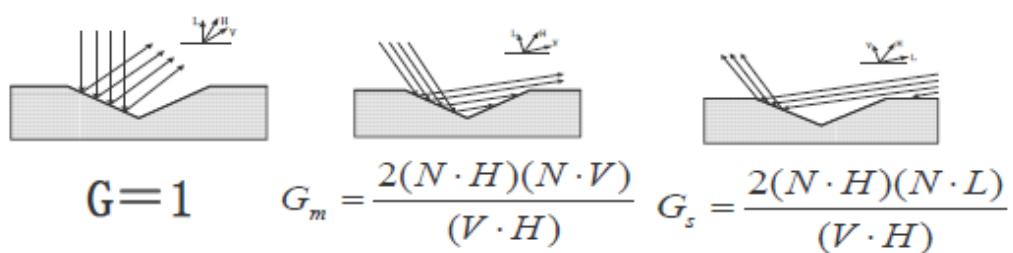
光的电磁理论

- 光波是电磁波的一种，自然光照射到物体表面的反射光，其反射率系数 ρ 可由 Fresnel 公式计算：

$$\rho = \frac{1}{2} \left(\frac{\sin^2(\theta - \psi)}{\sin^2(\theta + \psi)} + \frac{\sin^2(\theta - \psi)}{\sin^2(\theta + \psi)} \right)$$
- θ 是入射角， ψ 是折射角， η_1, η_2 是发生反射的物体表面两侧折射率，其中 $\sin\psi = \frac{\eta_1}{\eta_2} \sin\theta$
- 反射率 ρ 与折射率有关，是波长的函数 $\rho(\theta, \lambda)$

微平面理论

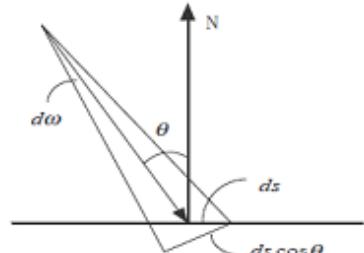
- 将粗糙物体表面看成是由无数个微小的理想镜面组成，这些镜面朝向各异，随机分布，所以可用分布函数去描述
- 综合各种原因后，物体的反射率可以这样计算：
 - $DG\rho(\theta, \lambda)$ D 为微平面法向的分布函数 G 为由于微平面的相互遮挡或屏蔽而使光产生的衰减因子
- Gauss分布函数模拟法向分布 (Torrance和Sparrow两人用Gauss高斯分布来模拟，也可用Berkmann分布)
 - $D = ke^{-(a/m)^2}$ k 为常系数 a 为微平面的法向与平均法向的夹角，即 $(N \cdot H) m$ 为微平面斜率的均方根，表示表面的粗糙程度， $m = \sqrt{\frac{m_1^2 + m_2^2 + \dots + m_n^2}{n}}$
- 衰减因子 G 也可反映物体表面的粗糙程度
 - $G = \min(1, G_m, G_s)$ 下图分别是无遮挡、反射光被遮挡、入射光被遮挡的情况



9.5.3 局部光照明模型

- 数学定义
 - I_r 反射光光强， E_i 单位时间内单位面积上的入射光能量
 - R_{bd} 物体表面对入射自然光的反射率系数， $R_{bd} = \frac{I_r}{E_i}$

- 入射光能量 E_i 可表示为 $E_i = I_i \cos\theta \cdot d\omega = I_i (N \cdot L) d\omega$



- 反射光光强： $I_r = R_{bd} I_i (N \cdot L) d\omega$

- 反射率系数

- 反射率系数可表示为漫反射率和镜面反射率的代数和： $R_{bd} = K_d R_d + K_s R_s$

$$K_d + K_s = 1$$

$$R_d = R_d(\lambda)$$

$$R_s = \frac{DG\rho(\theta, \lambda)}{\pi(N \cdot L)(N \cdot V)}$$

- 局部光照模型

- 局部光照模型表示为： $I_r = I_a K_a + I_i (N \cdot L) d\omega (K_d R_d + K_s R_s)$

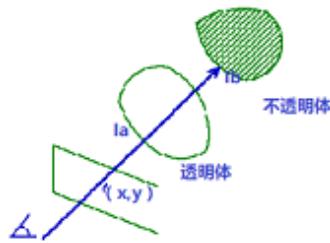
9.6 光透射模型

9.6.1 光透射模型概述

- 简单和局部光照模型没有考虑光的透射现象
- 其适用于场景中有透明或者半透明的物体的光照处理
- 早期用颜色调和法进行光透射模拟

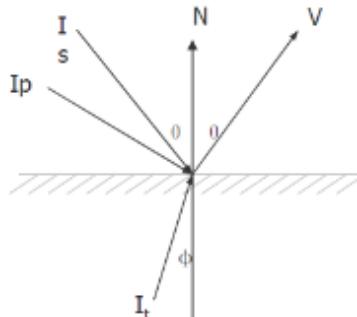
9.6.2 颜色调和法

- $I = t \times I_b + (1 - t) \times I_a$
- 不考虑透明体对光的折射以及透明物体本身的厚度，光通过物体表面是不会改变方向的，可以模拟平面玻璃。



9.6.3 Whitted 光透射模型

- 提出了第一个整体光照模型，并给出了一般光线跟踪算法的范例，综合考虑了光的反射、折射、透射和阴影等。被认为是计算机图形领域的一个里程碑。
- 即在简单光照明模型的基础上，加上透射光项、镜面反射光项
- $I = I_a \cdot K_a + I_p \cdot K_d \cdot (L \cdot N) + I_p \cdot K_s \cdot (R \cdot V)^n + I_t' \cdot K_t' + I_s' \cdot K_s'$



9.7 整体光照模型

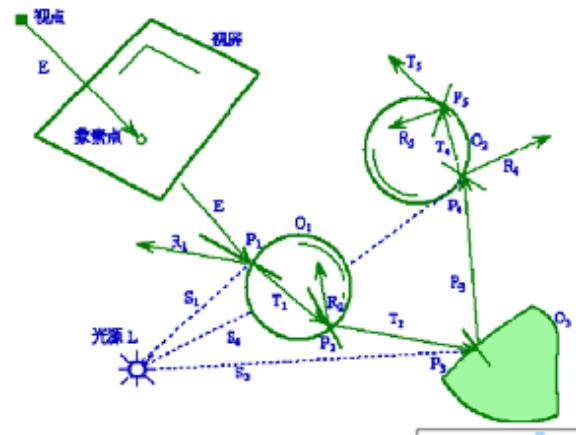
9.7.1 整体光照模型的诞生

- 简单和局部光照模型不能很好地模拟光的折射、反射和阴影等，也不能用来表示物体间的相互光照影响。
- 整体光照模型有光线跟踪、辐射度两种方法 光线跟踪效果图如下



9.7.2 光线跟踪基本过程

- 如图, 点光源 L , 透明体 O_1, O_2 , 不透明体 O_3
- 首先, 视线 E 从视点出发, 过视屏一个像素点, 到达球体 O_1 交于点 P_1
- 交点: 从 P_1 向光源 L 作一条阴影测试线 S_1 , 发现期间没有遮挡的物体, 用局部光照模型计算光源对 P_1 在其视线 E 方向上的光强作为局部光强
- 在交点产生衍生的光线
 - 反射光线 R_1 方向, 没有再与其他物体相交, 设该方向的光强为 0, 结束 R_1 的跟踪
 - 折射光线 T_1 方向, 与 O_1 交于点 P_2 , 由于点在物体内部, 假设它的局部光强为 0
 - 反射光线 R_2 , 可递归跟踪下去计算光强
 - 看折射光线 T_2 , 与 O_3 交于点 P_3 , 作阴影测试线 S_3 , 无遮挡, 计算该点局部光强。并且该点还产生了 R_3 可以继续跟踪下去...



9.7.3 光线跟踪的停止

- 该光线未碰到任何物体
- 该光线碰到了背景
- 光线在经过许多次反射和折射以后, 就会产生衰减, 光线对于视点的光强贡献很小
- 光线反射或折射次数即跟踪深度大于一定值

光线跟踪伪码:

```

1 RayTracing(start, direction, weight, ret_color)
2 {
3     if(weight < MinWeight) ret_color = BLACK;
4     else
5     {
6         计算光线与所有物体的交点中离start最近的点;
7         if(无交点) ret_color = BLACK;
8         else

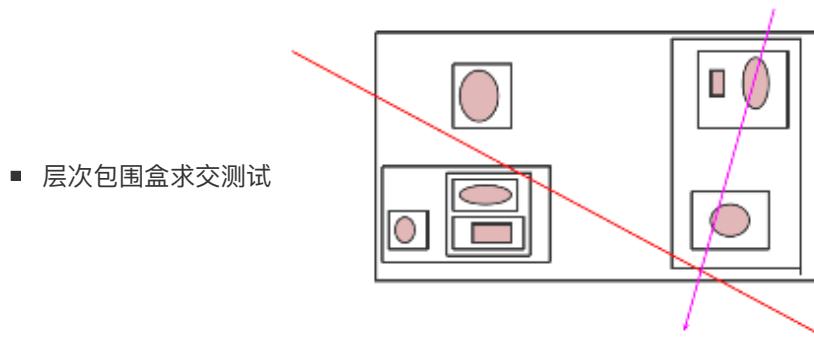
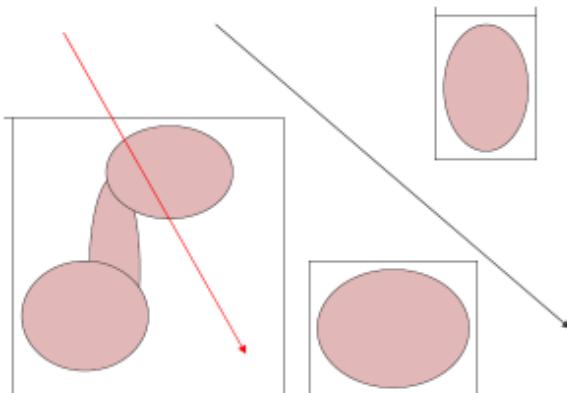
```

```

9   {
10    I_local = 在交点处用局部光照模型计算出的光强;
11    计算反射方向R;
12    RayTracing(最近的交点, R, weight*W_r, I_r);
13    计算折射方向T;
14    RayTracing(最近的交点, T, weight*W_t, I_t);
15    ret_color = I_local + K_r * I_r + K_t * I_t;
16  }
17 }
18 }
```

9.7.4 光线跟踪的加速

- 光线跟踪问题
 - 光线跟踪进行大量的求交运算，效率低
- 加速方案
 - 提高求交速度：针对性的几何算法、...
 - 减少求交次数：包围盒、空间索引、...
 - 包围盒求交测试 (先和包围盒求交，有交点再和内部图形求交)

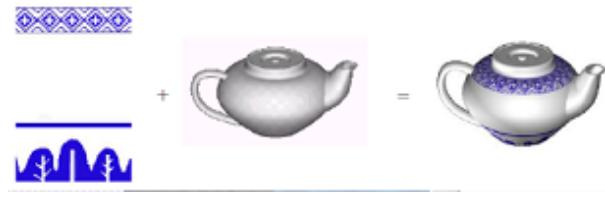


- 减少光线条数：颜色插值、自适应控制、...
- 采用广义光线和采用并行算法等

9.8 纹理映射

9.8.1 纹理映射作用

- 表面用纹理来代替，不用构造模型和材质细节，节省时间和资源
- 可用一个粗糙多边形和纹理来代替详细的几何构造模型，节省时间和资源
- 让用户做其他更重要的东西



9.8.2 纹理分类

- 颜色纹理：颜色或明暗度变化体现出来的表面细节，如刨光木材表面上的木纹。

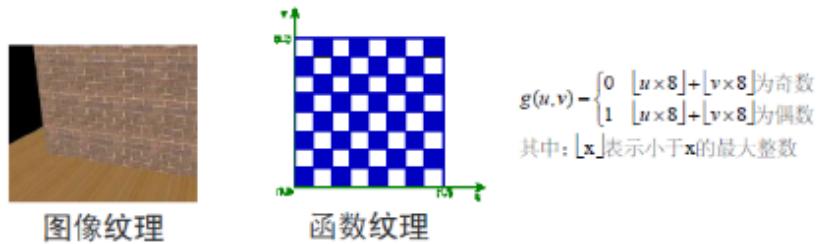


- 几何纹理：由不规则的细小凹凸体现出来的表面细节，如桔子皮表面的皱纹。



9.8.3 纹理定义

- 图象纹理：将二维纹理图案映射到三维物体表面，绘制物体表面上一点时，采用相应的纹理图案中相应点的颜色值
- 函数纹理：用数学函数定义简单的二维纹理图案，如方格地毯；或用数学函数定义随机高度场，生成表面粗糙纹理即几何纹理



9.8.4 纹理映射

纹理映射 (Texture Mapping)

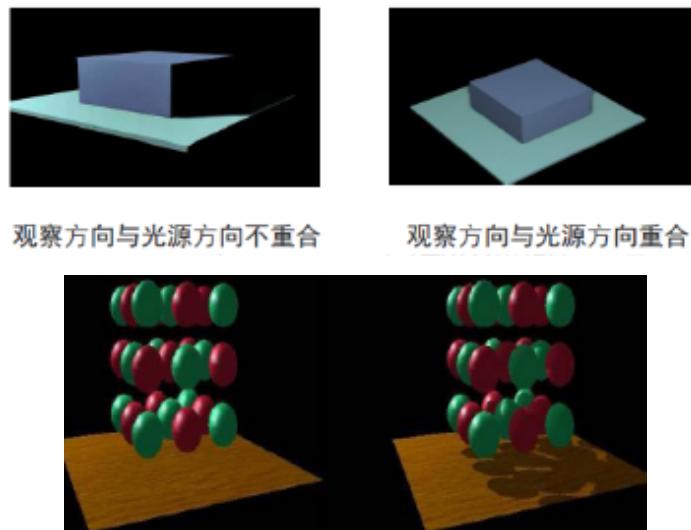
- 通过将数字化的纹理图像覆盖或投射到物体表面，而为物体表面增加表面细节的过程
- 寻找一种从纹理空间(u, v)到三维曲面(s, t)之间的映射关系，将点(u, v)对应的彩色参数值映射到相应的三维曲面(s, t)上，使三维曲面表面得到彩色图案
- 两种方法
 - 在绘制一个三角形时，为每个顶点指定纹理坐标，三角形内部点的纹理坐标由纹理三角形的对应点确定。即指定：
$$\begin{cases} (x_0, y_0, z_0) \rightarrow (u_0, v_0) \\ (x_1, y_1, z_1) \rightarrow (u_1, v_1) \\ (x_2, y_2, z_2) \rightarrow (u_2, v_2) \end{cases}$$
 - 指定映射关系，如
$$\begin{cases} u = a_0x + a_1y + a_2z + a_3 \\ v = b_0x + b_1y + b_2z + b_3 \end{cases}$$
- 扰动函数 $P(u, v)$

- 通过对景物表面各采样点的位置作微小扰动来改变表面的微观几何形状。
 - 几何纹理使用一个称为扰动函数的数学函数进行定义。
 - 设景物表面的参数方程 $Q = Q(u, v)$, 表面任一点的法线 $N = N(u, v) = \frac{Q_u(u, v) \times Q_v(u, v)}{|Q_u(u, v) \times Q_v(u, v)|}$
 - 则扰动后的表面为 $Q' = Q(u, v) + P(u, v) \cdot N$
-

9.9 阴影处理

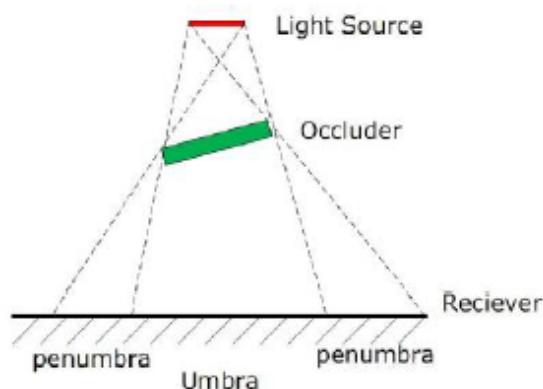
9.9.1 阴影定义

- 阴影是由于观察方向与光源方向不重合而造成的
- 阴影使人感到画面上景物的远近深浅，从而极大地增强画面的真实感



9.9.2 阴影分类

- 本影：不被任何光源所照到的区域Umbra。
- 半影：被部分光源所照到的区域Penumbra

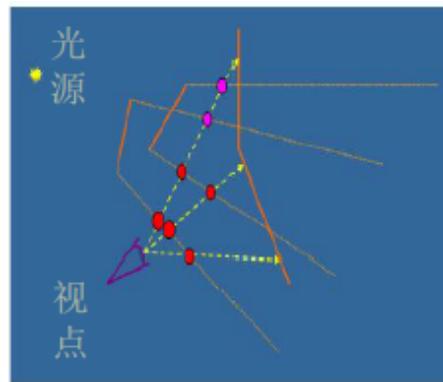


- 自身阴影：由于物体自身的遮挡而使光线照射不到它上面的某些面
- 投射阴影：由于物体遮挡光线，使场景中位于它后面的物体或区域受不到光照射而形成的。

9.9.3 阴影体法

阴影算法——阴影体法 (Shadow Volume)

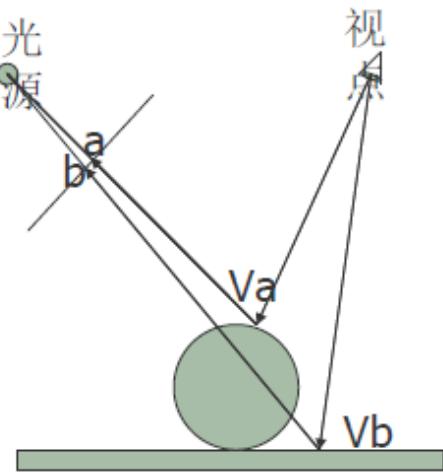
- 由一个点光源和一个三角形可以生成一个无限大的阴影体。落在这个阴影体中的物体，就处于阴影中。
- 在对光线进行跟踪的过程中，若射线穿过了阴影体的一个正面（朝向视点的面），则计数器加1。若射线穿过了阴影体的一个背面（背向视点的面），则计数器减1。最终计数器大于0，则说明这个像素处于阴影中，否则处于阴影之外。



9.9.4 阴影图法

阴影算法——阴影图法 (Shadow Mapping) :

- 思想是使用Z缓冲器算法，从投射阴影的光源位置对整个场景进行绘制。
- 对于Z缓冲器的每一个象素，它的z深度值包括了这个像素到距离光源最近点的物体的距离。一般将Z缓冲器中的整个内容称为阴影图 (Shadow Map)，有时候也称为阴影深度图。
- 从视点的角度来进行，对场景进行二次绘制。
- 在对每个图元进行绘制的时候，将它们的位置与阴影图进行比较，如果绘制点距离光源比阴影图中的数值还要远，那么这个点就在阴影中，否则就不在阴影中。



第十章 绘制技术

10.1 基于图像的绘制技术IBR

10.1.1 传统图像绘制与IBR比较

IBR, Image-Based Redering

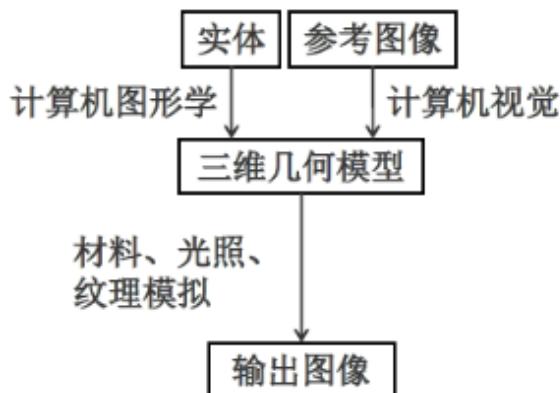


图1 传统的图像绘制过程

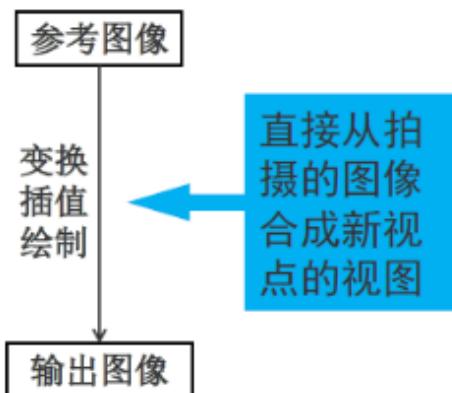
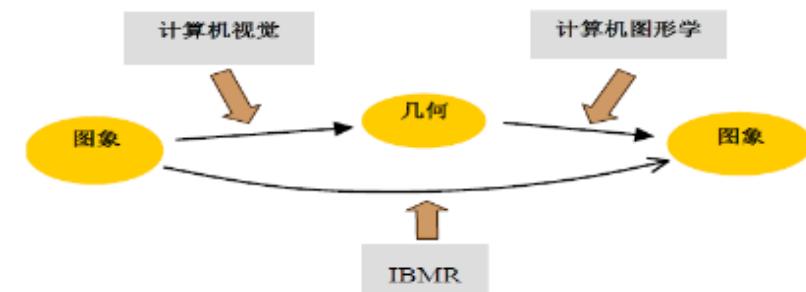


图2 IBR绘制过程

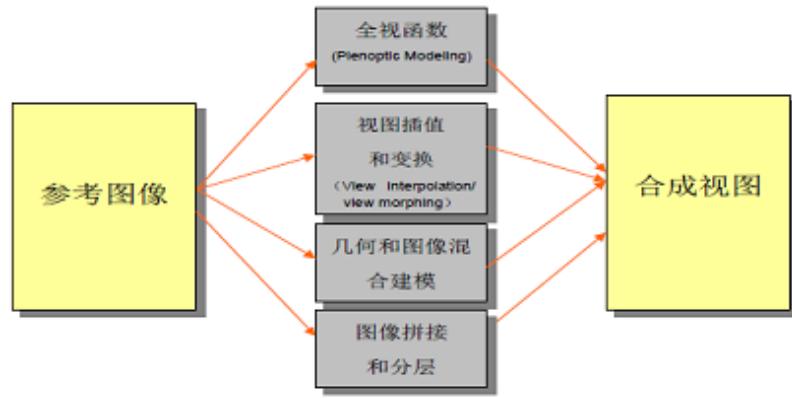
传统图像绘制	基于图像的绘制技术
建模复杂、困难	建模简单
计算和显示开销大，绘制速度慢	显示速度快
难以达到真实感效果	真实感强



10.1.2 IBR发展方向

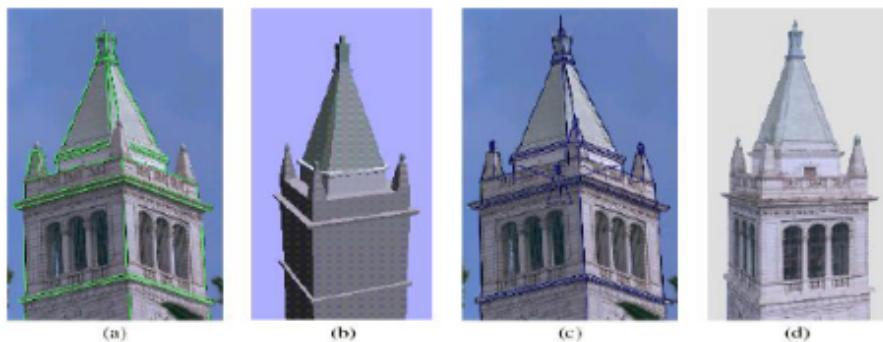
- 与传统集合绘制方式有效结合
- 算法固化提高图像生成速度，达到实时绘制
- 分层绘制
- 提高Morphing中特征提取的效率和质量
- 图像拼接中的快速配准

10.1.3 IBR的绘制过程



IBR绘制中的重要方法：

- 几何和图像混合建模(Hybrid Geometry- and Image-based Approach [DTM96])
 - 特点
 - 可以通过拍摄的几张照片合成逼真的新视图，简单快捷
 - 只能适用于普通建筑物等外形规整的景物
 - 过程
 - a. 拍摄相片，交互指定建筑物边缘
 - b. 生成建筑物粗模型
 - c. 利用基于模型的立体视觉算法精化模型
 - d. 利用基于视点的纹理映射合成新视图



- 视图插值和变形(View Interpolation[CW93] / View Morphing[SD96])

- 特点
 - 简单方便，只要求几幅参考图像
 - 漫游范围受限，只能在几幅参考图像的视点连线之间作有限运动
 - 常用于加速图形学中的绘制速度
- 视图插值方法(View Interpolation):
 - 要求新视点位于两参考图象视点所决定的直线(基线， baseline)上。由参考图线性插值产生新视图。
 - 一般情况下不能产生正确的透视投影结果，而只生成近似的中间视图。



- 视图变换方法(View Morphing):
 - 要求新视点位于两参考图象视点所决定的直线(基线， baseline)上。由参考图线性插值产生新视图。
 - 一般情况下不能产生正确的透视投影结果，而只生成近似的中间视图。

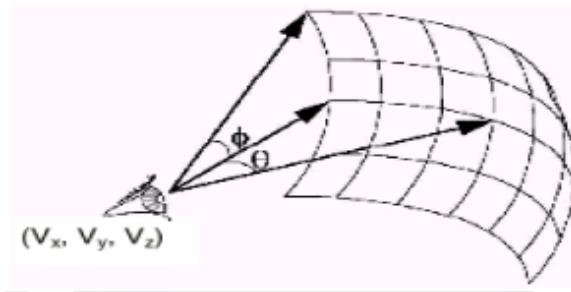


$$Warp_{01}(w, P) = (1 - w)P_0 + wF_0P_0$$

$$Warp_{10}(w, P) = (1 - w)F_1P_1 + wP_1$$

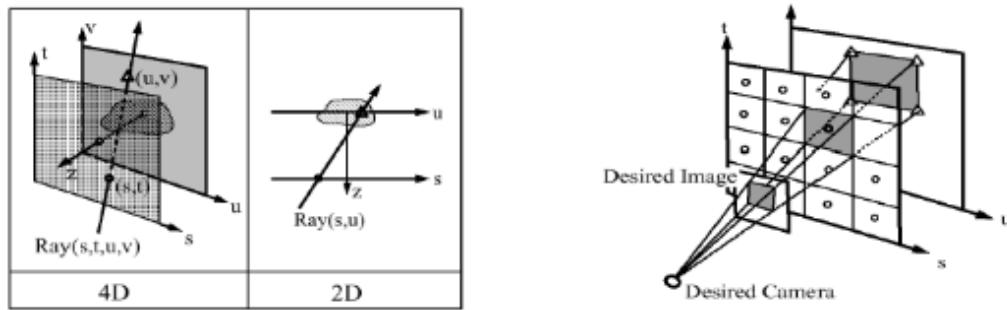
- 利用参考图像上像素点重投影生成新视图
 - 利用投影知识决定的变形位置
- 全视函数(Plenoptic Function-based)

- $\mu = Plenoptic(\theta, \phi, \lambda, V_x, V_y, V_z, t)$



- 基于光场的显示(Light Field Rendering[LH96] and Lumigraph [GGSC96])

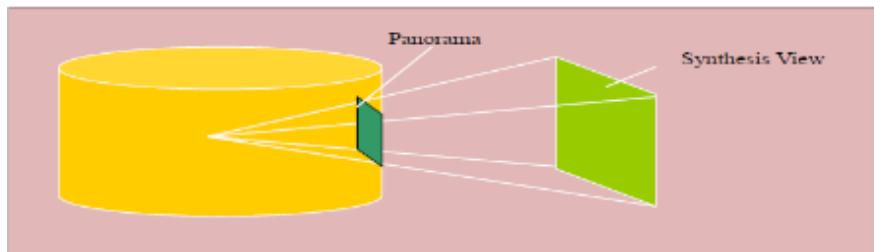
- 基于“自由空间中沿一条光线传递的辐射能不变”的假设，把全视函数简化为描述离开或进入一封闭自由空间(如空立方体)的完全光流分布
- 由于只考虑视流信息，因此不必对反射属性作假设，不需要立体对应关系
- 数据量大，采样困难



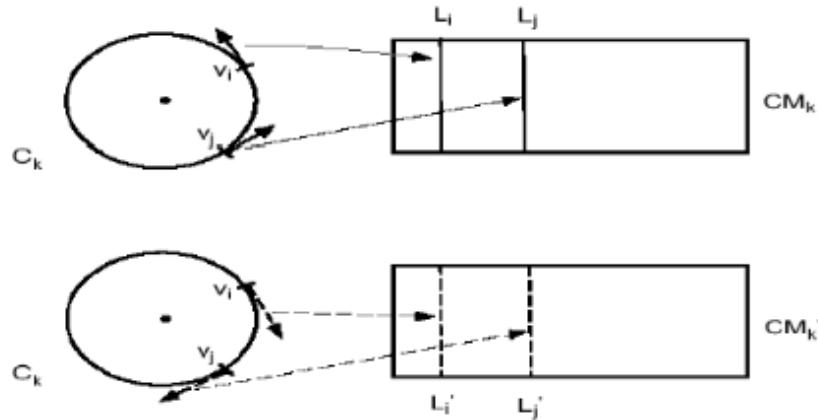
- 图象拼接(mosaic)

- 全景图(Panorama)

- 在一个视点拍摄的几幅图像，通过整合拼接成一个视点周围的场景视图，投影在圆柱面或者球面上成为全景图。
- 只需在一个视点拍摄的几张照片，数据量小，采样方便。
- 视点不能移动，但是可以转动观察方向，通过放大缩小(zoomin/zoom out)进行近似的前后运动

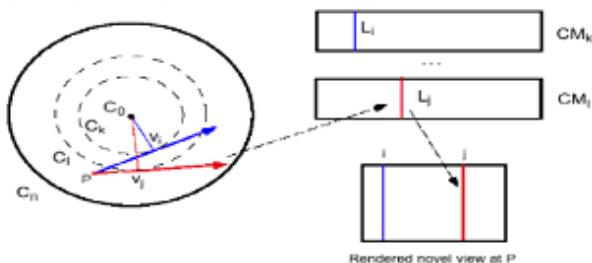


- 同心拼接(Concentric Mosaic)

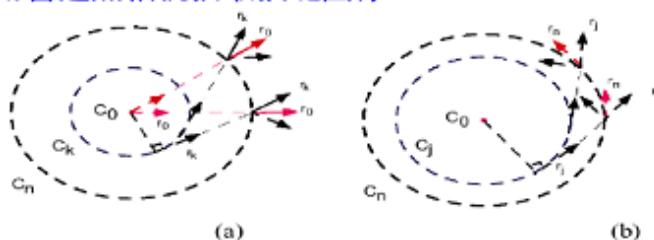


- 通过沿一系列同心圆切线方向拍摄的狭缝图像拼合成同心拼接

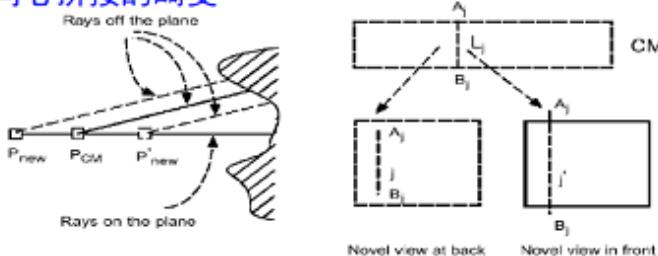
利用同心拼接合成视图原理



使用普通照相机获取狭缝图像



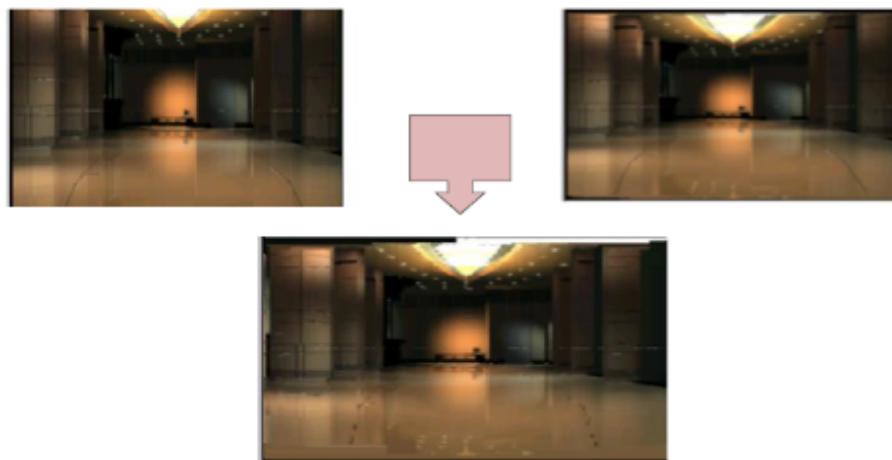
同心拼接的畸变



- 特点：只需在一个圆上拍摄的一系列图像，采样方便 视点可以在采样圆内做平面移动，生成场景真实感强 数据量较光场等全视函数方法为小 移动范围受限，基于狭缝图象的绘制：有场景畸变现象

- 基于狭缝图象段的绘制

- 将同心拼接中使用的狭缝图象进一步分为狭缝图象段
- 测定不同距离上对应狭缝图象段的伸缩比例
- 利用狭缝图象段的伸缩进行正确的绘制



- 立体视觉(Stereo Vision)

10.2 基于点的建模与绘制

10.2.1 点建模与绘制方法

Point based Rendering

随着模型多边形复杂度的剧增，点模型的优势越发明显。

点模型的数字几何处理流程：

- 点的获取
 - 3D扫描仪（深度照相机生成的深度图和激光三维扫描仪等设备得到的大量空间三维点位置）
 - 点模型的另一个来源是现有模型，大部分几何模型如多边形网格模型、隐式曲面等都能方便地转化为点模型
- 点的处理
 - 配准
 - 机器配准：标签法
 - 自动配准：特征提取法
 - 去噪
 - 移动最小二乘（MLS）曲面逼近原始点集模型
 - 基于三维Meanshift过程的各向异性点模型去噪算法
 - MLS曲面重建方法
 - 点模型的非局部去噪方法
 - 基于采样保真性的点模型去噪算法
 - 修补
 - 基于上下文的点模型修复



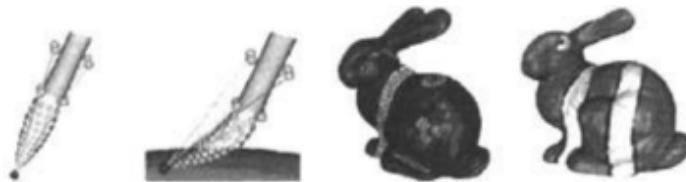
Original model

Marked area

Smooth Filling Context-based Filling

- 点的建模

- 目标：从原始点云中构造出一个连续的表面模型
- 涉及的技术较多，曲面重建、曲面简化、几何属性分析、特征提取、重采样 点模型的后期处理则是对点模型作进一步的造型处理：编辑、变形、布尔运算
- 曲面重建技术
 - 指根据离散扫描点数据重建三维模型的过程
 - 常用方法：RBF(Radial Basis Function)曲面重建方法 MLS(Moving Least-Square)曲面
- 曲面简化技术
 - 利用三维扫描仪获得的点模型通常具有很高的复杂度，为了使大规模数据模型符合几何处理和绘制，必须对数据模型进行简化
- 点模型编辑技术
 - 对点模型的颜色、纹理等外观属性以及法向量的处理。例如下图是点模型的画刷着色效果图。



基于点模型的画刷及看色效果

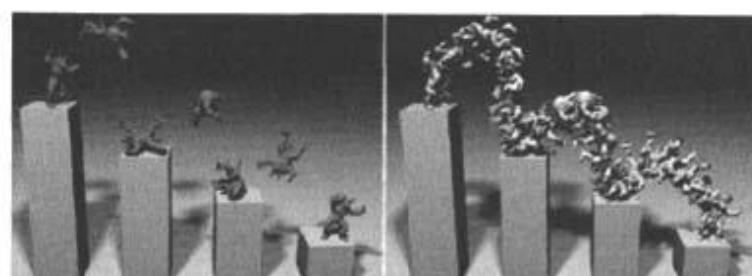
- 几何造型技术

- 实体几何造型(CSG: Constructive Solid Geometry) 一种基于简单实体的布尔运算构造复杂模型的技术，即通过对多个简单的点模型进行布尔运算后生成复杂的点模型。



利用布尔运算构造几何实体

- 自由变形 这种变形技术首先生成一个置换函数 $d: R^3 \rightarrow R^3$ ，然后对表面上的每个点 P_i ，实施变换 $P_i \rightarrow d(P_i)$
- 点模型动画(MFM: Mesh Free Method) 力学分析的新方法，近年来研究者将无网格方法和点模型相结合，提出了基于点的动画。例如：模型形变



(a)模型关键帧姿态

(b)算法插值获取中间动作

- 模型渐变(Morphing)技术

- 点模型自身的特点决定了在模型渐变中，它比基于网格模型的渐变有明显的优势，但由于点模型没有提供表面的解析表达式和参数化信息，从另一方面又增加了它渐变的难度。



点模型渐变

- 基于物理的渐变 需要建立相关物理模型，中间过渡点模型根据能量方程求解，这种渐变在一定程度

上模拟了真实的物理现象

- 基于几何的渐变 通过对源和目标模型进行几何变换来获得中间过渡模型，其计算量没有基于物理的渐变大
- 点模型的绘制，如Qsplat技术
 - 由斯坦福大学开发的具有代表性的点绘制技术。
 - 它利用树状层次包围球数据结构，树中每个结点包含球的位置和半径、每点处的法向量、法锥面的宽度、颜色值。
 - 在进行绘制时，层次树按深度优先方法递归遍历。对每个中间结点，首先判断该球是否完全在屏幕外或者是完全背向的，以进行可见性选择。如果该结点至少有一部分子结点是可见的，则将该结点在屏幕上投影大小同一个阈值进行比较。如果大于阈值，则继续向下递归；如果小于阈值或者已经到达叶结点，则按该结点的球位置及半径确定的屏幕上的位置和大小绘制一个小区域。

10.2.2 基于点的建模与绘制的发展趋势

- 基于点的自然场景建模
- 基于点模型的动画
- 点模型的简化、压缩和传输
- 基于GPU的大规模点模型绘制