Chapter 12. Curves and Curved Surfaces

金小刚 **Email:** *jin@cad.zju.edu.cn*

http://www.cad.zju.edu.cn/home/jin

*浙江大学**CAD&CG**国家重点实验室*

Where there is matter, there is geometry.

–Johannes Kepler

- 本章的目的是介绍曲线曲面在实时绘制中的作用。特别是许多曲面类型即将变成或者已经变成了图形**APIs**的一部分，并且有直接的加速器支持。

- 三角形是一种基本的绘制图元。图形硬件支持三角形的直接绘制。然而曲线曲面可以使用方程简要地描述几何物体。这些方程可以求值，然后生成三角形集合并且送入流水线进行绘制。

使用曲线曲面的优点

- 它们有比多边形集合更加紧凑的表达式；
- 它们提供可以缩放的几何图形；
- 它们提供比直线和平面多边形更加光滑更加连续的图形；
- 动画和碰撞检测更加简单更加快速。

- 对于具体的应用，使用一个紧凑的曲线曲面表示的优点：

  - 对于模型的保存，可以节省存储空间。
  - 对于一个函数曲面的变换比对网格表示的曲面的变换矩阵的复杂度更小。
  - 使用少量的控制多边形能得到比较真实的效果。
  - 可以很容易生成模型的**LOD结构**。因而在实时绘制时根据实际情况提高绘制质量或者提高帧速率。
  - 动画中，曲面只有少量的点需要去运动。这些点然后去形成一张曲面，生成一张光滑的曲面。
  - 对于碰撞检测，趋向于更加有效更加精确。
  - 可以使实时计算机图形应用更快速，编码更快速，生存期更长。

  ……

参数曲线(Parametric Curves)

- 参数曲线应用在许多不同的领域，可以使用许多不同的方法实现。
- 对于实时绘制图形学，参数曲线经常用作去沿着一条预定义的路径去移动视点和物体。这将包括改变位置和方向。这里我们将仅仅考虑位置路径。

- 一条参数曲线可以使用一些公式如参数*t*的一个函数来表示。数学上，我们写作**p(*t*)**，意思是对于每个*t*值，通过这个函数可以得到一个点。这个参数t可能属于某些区间，也称作定义域，如t∈[a,b]。这样生成的点是连续的，即当 $\varepsilon \to 0$，那么**p(*t*, ε) → p(*t*)**。也就是说，如果 ε 是一个非常小的数，那么**p(*t*, ε)**与**p(*t*)**是两个位置非常接近的点

- 我们将要介绍**Bézier曲线**，然后讨论怎样使用分段**Bézier曲线**，并引入曲线连续性的概念，最后介绍两种其他有用的曲线:三次**Hermite样条**和**Kochanek-Bartels样条**。

Bézier曲线

- 两点p_0与p_1之间的线性插值的轨迹是一条直线。

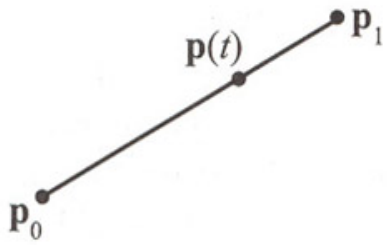
$$p(t) = p_0 + t(p_1 - p_0) = (1-t)p_0 + tp_1 \qquad t \in [0,1]$$

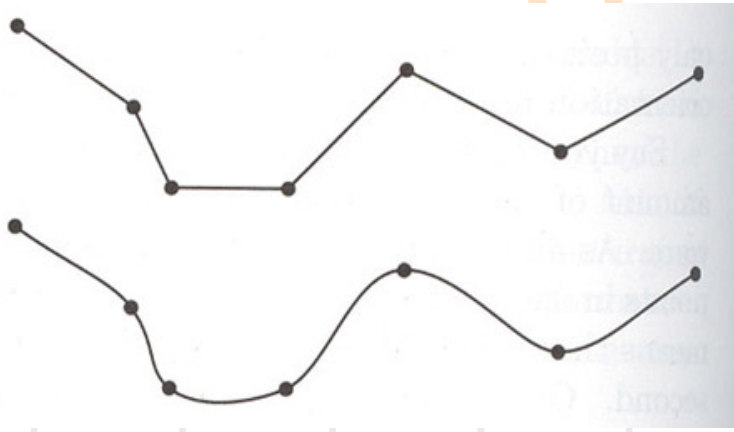
- 由 $p(0)=p_0$， $p(1)=p_1$,$0<t<1$,我们可以得到p_0与p_1之间的任意一点。如果我们要在p_0与p_1之间移动摄像机，一秒中移动二十步，那么我们可以设定

$$t_i = i/(20-1)$$

其中i是帧的序号，从0开始。

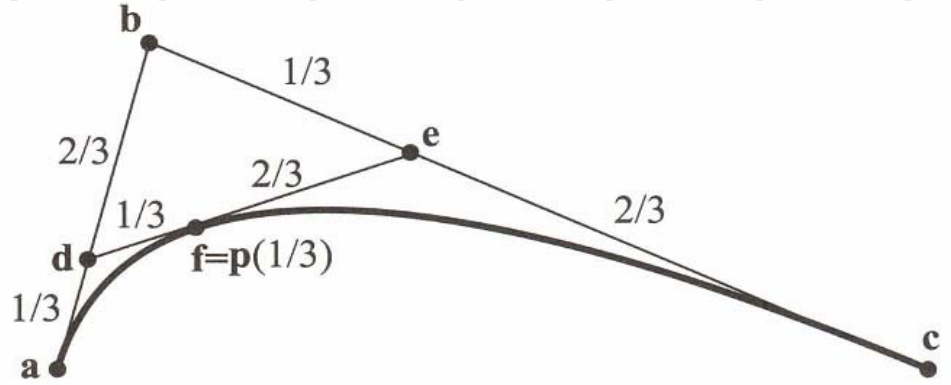
Bézier曲线





- 从上图可知，如果在两个点之间进行插值，使用线性插值的方法就足够了。但是如果用于多个点间的插值，在每两段曲线的连接处会产生令人无法忍受的尖点。
- 我们通过递归线性插值（**de Casteljau** 算法）来解决这个问题。

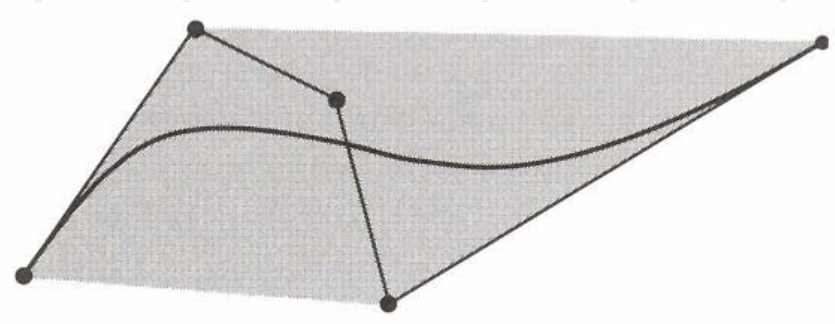
Example：通过递归线性插值构造二次Bézier曲线



如图定义a，b，c为控制顶点（控制顶点数目为曲线次数＋1）。为了找到对应t＝1/3的曲线上一点f，我们分别在a和b之间插值得到d，b和c之间插值得到e，那么d和e之间线性插值得到的f点即为所求曲线上一点。

$$
\begin{aligned}
\mathrm{p}(t) &= (1-t)\mathrm{d} + t\mathrm{e} \\
&= (1-t)[(1-t)\mathrm{a} + t\mathrm{b}] + t[(1-t)\mathrm{b} + t\mathrm{c}] \\
&= (1-t)^2\mathrm{a} + 2(1-t)t\mathrm{b} + t^2\mathrm{c}
\end{aligned}
$$

Bezier曲线



Example：线性插值构造的四次Bézier曲线

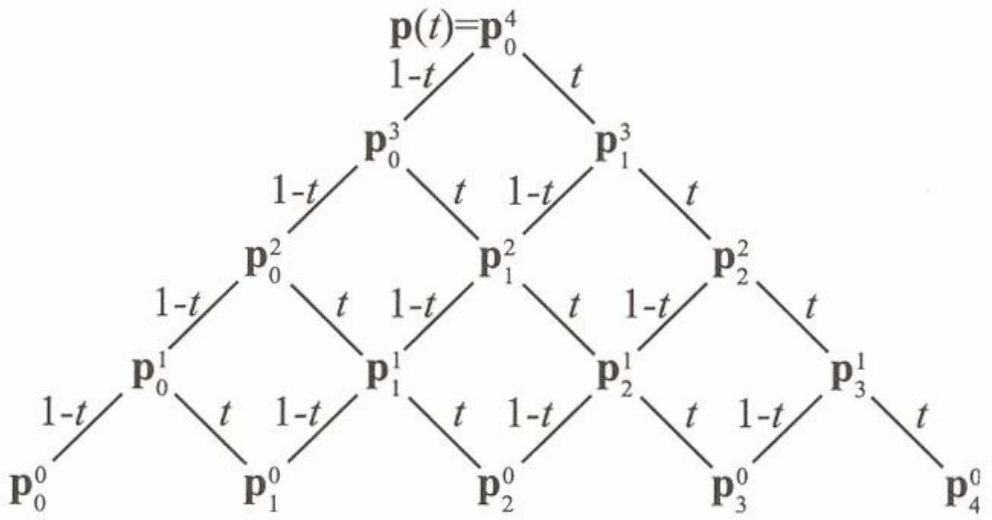
- 曲线位于控制顶点的凸包内；
- 在第一个控制顶点，曲线的切线与第一个和第二个控制顶点的连线平行。对于最后一个控制顶点，情况类似。

对de Casteljau 算法的一点总结

使用以下符号表示插值过程：

$$p_i^k(t) = (1-t)\,p_i^{k-1}(t) + t\,p_{i+1}^{k-1}(t), \begin{cases} k = 1 \ldots n \\ i = 0 \ldots n-k \end{cases}$$

对$n=4$，即4次Bézier曲线进行插值过程见下图：



使用Bernstein多项式构造Bézier曲线

前面我们介绍了通过递归插值构造Bézier曲线，

现在，我们使用Bernstein基给出构造Bézier曲线的代数公式。

$$\mathrm{p}(t)=\sum_{i=0}^{n}B_i^n(t)\mathrm{p}_i$$

其中，Berstein多项式定义如下：

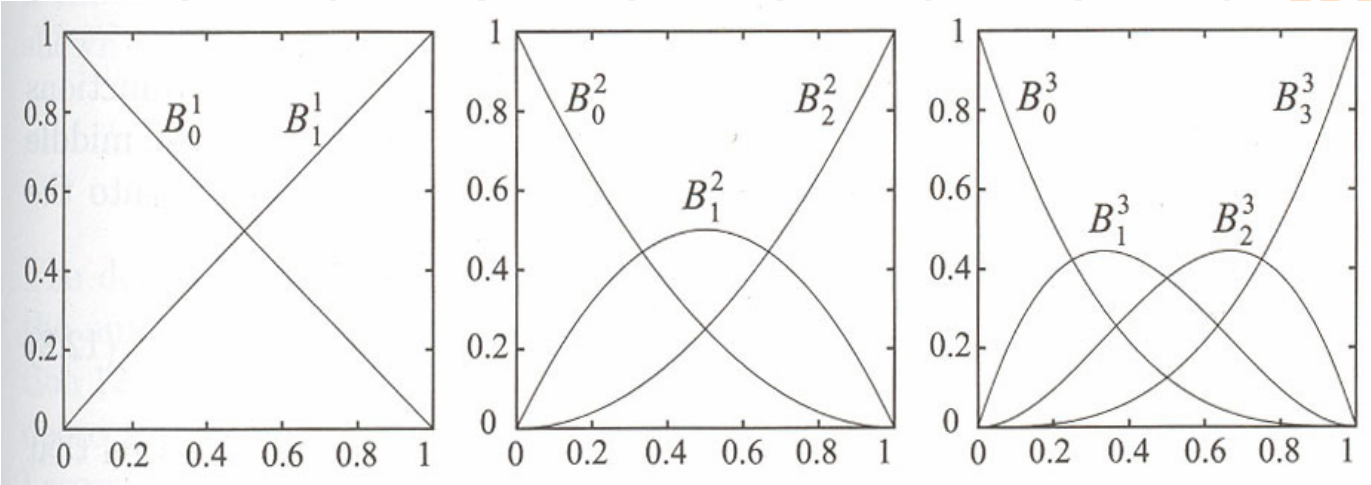
$$B_i^n(t)=\binom{n}{i}t^i(1-t)^{n-i}=\frac{n!}{i!(n-i)!}t^i(1-t)^{n-i}$$

Bernstein的两个基本性质：

$$B_i^n(t)\in[0,1],\ when\ t\in[0,1],$$

$$\sum_{i=0}^{n}B_i^n(t)=1$$

使用Bernstein多项式构造Bézier曲线



***n*＝1，2，3时的Berstein多项式/blending function**

从上图我们得到以下信息：

- **$p(0)=p_1$, $p(1)=p_n$, $p'(0)//p_2p_1$, $p'(1)//p_{n-1}p_n$,**即端点插值，端边相切；
- 混合函数的对称性，$B_i^n(t) = B_{n-i}^n(1-t)$
- 我们可以通过修改控制顶点来修改曲线。

使用Bernstein多项式构造Bézier曲线

- **Bézier曲线的旋转不变性：**

  通过旋转控制顶点可以实现对曲线的旋转。而不需要首先计算出Bezier曲线上的点，然后再进行曲线的旋转。

  参数曲线和曲面都具有这一非常好的性质，说明程序可以先旋转控制顶点（数量很少），然后再生成旋转曲线上的点，这比按照相反的顺序计算要快得多。

使用Bernstein多项式构造Bézier曲线

- 二次Bézier曲线的Berstein基形式表示为：

$$\begin{aligned} \mathrm{p}(t) &= B_0^2\mathrm{p}_0 + B_1^2\mathrm{p}_1 + B_2^2\mathrm{p}_2 \\ &= \binom{2}{0} t^0(1-t)^2\mathrm{p}_0 + \binom{2}{1} t^1(1-t)^1\mathrm{p}_1 + \binom{2}{2} t^2(1-t)^0\mathrm{p}_2 \\ &= (1-t)^2\mathrm{p}_0 + 2t(1-t)\mathrm{p}_1 + t^2\mathrm{p}_2 \end{aligned}$$

- 同样，可将三次Bézier曲线表示为：

$$\mathrm{p}(t) = (1-t)^3\mathrm{p}_0 + 3t(1-t)^2\mathrm{p}_1 + 3t^2(1-t)\mathrm{p}_2 + t^3\mathrm{p}_3$$

使用Bernstein多项式构造Bézier曲线

- 可以将**Bézier**曲线用幂积形式表示：

$$\mathrm{p}(t)=\sum_{i=0}^{n} t^{i} c_{i}$$

幂积形式的**Bézier**曲线对曲线快速生成很有用。

- 对**p(t)**求导，我们得到一个新的、次数低于原曲线的**Bézier**曲线：

$$\frac{d}{dt}\mathrm{p}(t)=n\sum_{i=0}^{n-1} B_{i}^{n-1}(t)(\mathrm{p}_{i+1}-\mathrm{p}_{i})$$

使用Bernstein多项式构造Bézier曲线

- **Bézier曲线的缺点：**

—除端点外，不经过所有控制顶点；

—随着曲线次数的升高，控制顶点的数目随之增加。

- 解决办法：

每对控制顶点之间的曲线用低阶曲线进行代替。这种分段插值提供了足够的连续性。

有理Bézier曲线

引入有理Bézier曲线的原因：

- Bézier曲线缺乏灵活性——只能调节控制顶点；
- Bézier曲线不能表示全部曲线——例如不能表示圆弧。

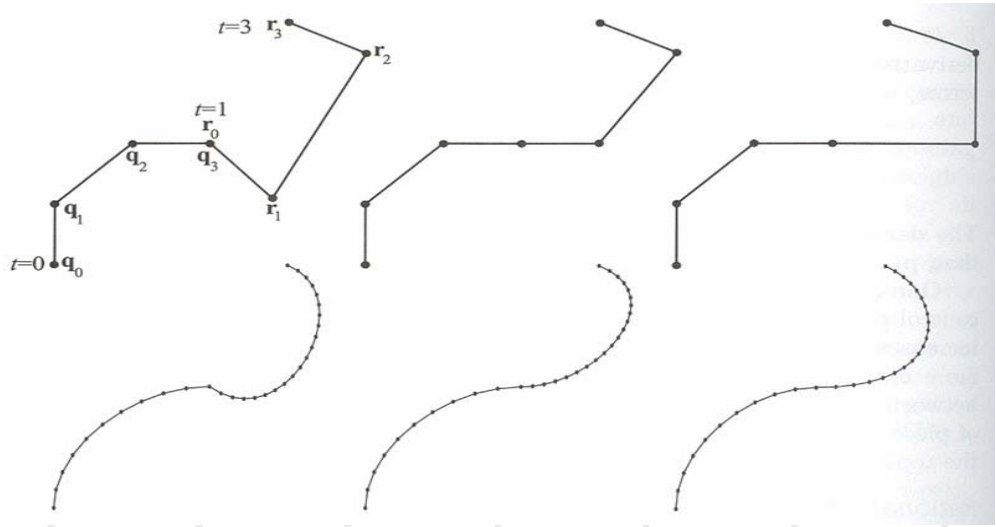
有理Bézier曲线的定义：

$$\mathrm{p}(t)=\frac{\sum_{i=0}^{n}\omega_i B_i^n(t)\mathrm{p}_i}{\sum_{i=0}^{n}\omega_i B_i^n(t)}$$

这里，我们通过增加权因子w_i来提高自由度。

连续性与分段Bézier曲线

- 假设我们需要连接两条三次Bézier曲线q_i和r_i，(i=0,1,2,3)。但是简单的使用$q_3=r_0$作为连接点(joint)，曲线不够光滑。



C^0, G^1, C^1连续的三次Bézier曲线

- 分段Bézier曲线: 多段Bézier曲线的合成

连续性与分段Bézier曲线

■基本概念: C^n, G^n 连续

—C^n连续:曲线有直到n阶的连续非零导函数;

—G^n连续:以G^1为例, G^1指两端子曲线在接合点处导数同向,但是对长度不作要求,比C^1稍弱。

■假设 $p(0)=q_0, p(1)=q_3=r_0, p(3)=r_3$以及 $t_0=0.0, t_1=1.0, t_2=3.0$

对于上页图中的$\mathbf{q}_i$段的曲线, $t\in[0,1]$。但是对于$\mathbf{r}_i$段的曲线,需要对t做变换;

将参数$t\in[t_1,t_2]$变换到$t\in[0,1]$,需要使用等式:

$$t'=\frac{t-t_1}{t_2-t_1}$$

连续性与分段Bézier曲线

- 运用Bézier曲线端点切线的性质, 接合点G^1连续的条件为:

$$(q_3 - q_2) = c(r_1 - r_0) \quad \text{for } c > 0$$

- 进一步, 取 $c = \dfrac{t_2 - t_1}{t_1 - t_0}$

能够达到C^1连续.

分段Bézier曲线的优点

- 可以使用低阶Bézier曲线生成高阶分段Bézier曲线类似的曲线;
- 分段Bézier曲线通过更多控制顶点;
- 常常使用三次Bézier曲线构造分段Bézier曲线，因为它是可描述S-型曲线的最低次曲线。

三次Hermite插值样条

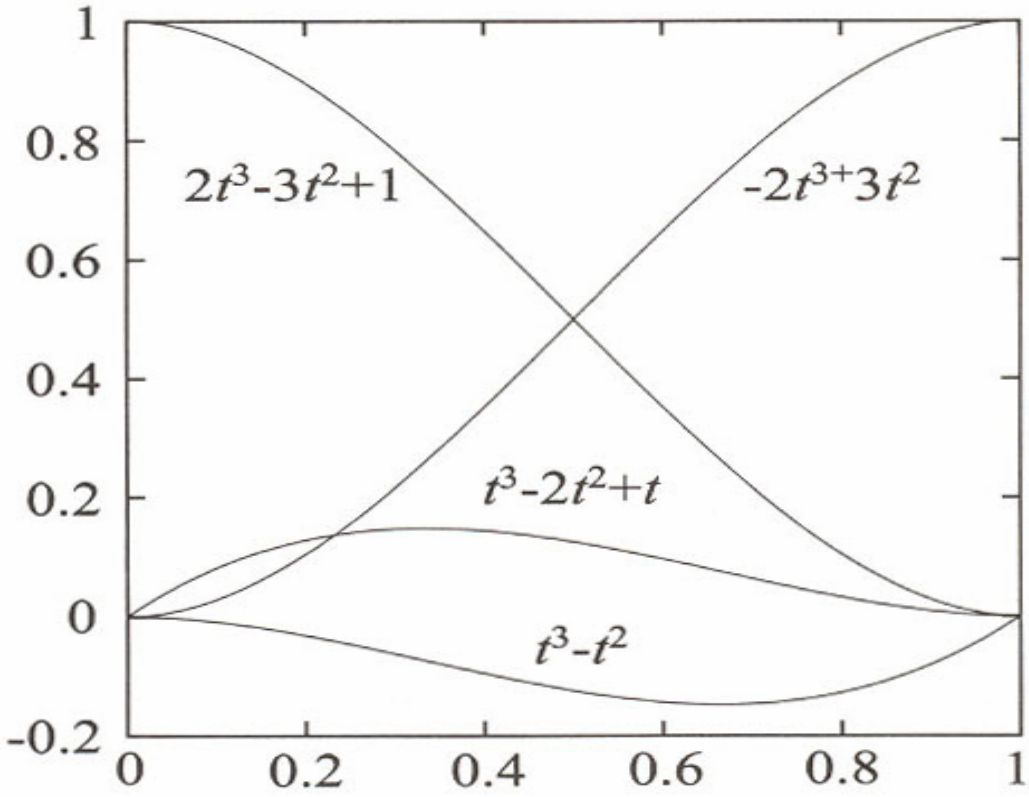
- 三次Hermite曲线由起始点$\mathbf{p}_0$、终点$\mathbf{p}_1$、起始切线$\mathbf{m}_0$、终点切线$\mathbf{m}_1$来定义：$t \in [0,1]$

$$\mathrm{p}(t) = (2t^3 - 3t^2 + 1)\mathrm{p}_0 + (t^3 - 2t^2 + t)\mathrm{m}_0$$
$$+ (t^3 - t^2)\mathrm{m}_1 + (-2t^3 + 3t^2)\mathrm{p}_1$$

- 性质：

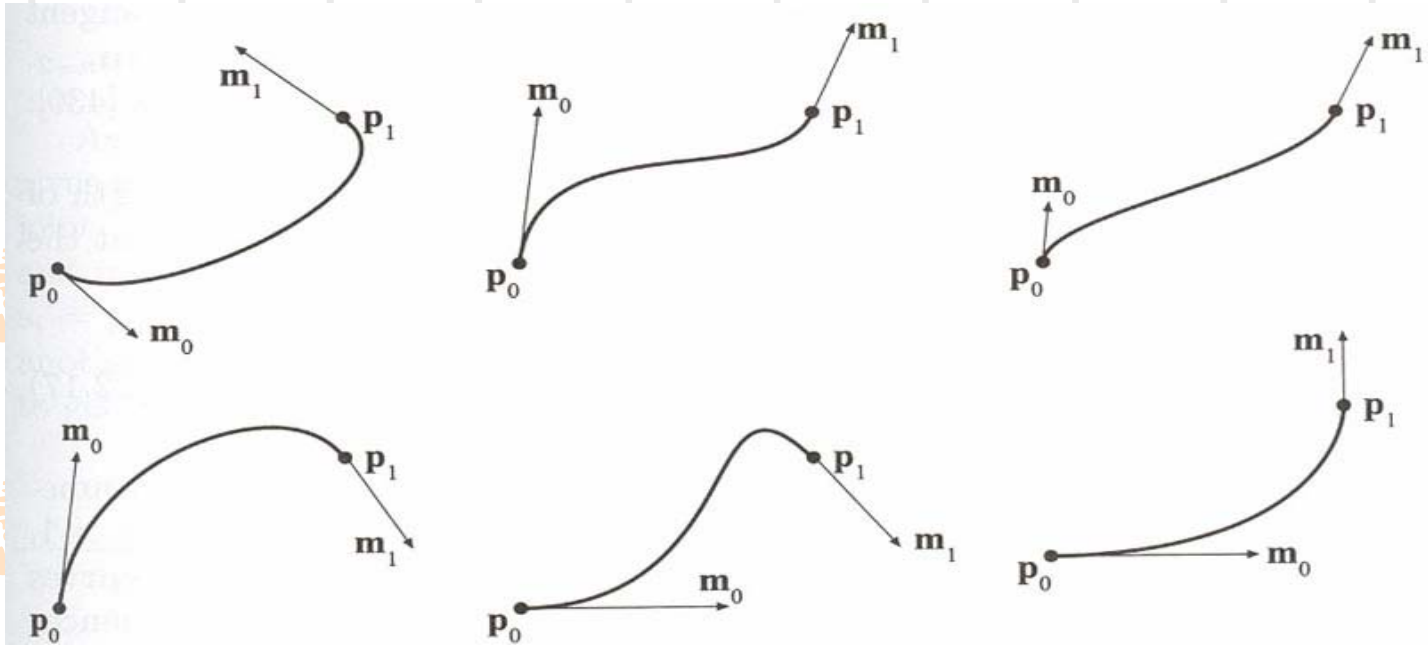
$$\mathrm{p}(0) = \mathrm{p}_0,\ \mathrm{p}(1) = \mathrm{p}_1,\ \frac{\partial \mathrm{p}}{\partial t}(0) = \mathrm{m}_0,\ \frac{\partial \mathrm{p}}{\partial t}(1) = \mathrm{m}_1$$

三次Hermite基函数图



三次Hermite插值例子

Example： 图中各曲线插值点相同，但始末切线方向不同。



三次Hermite插值例子

Kochanek-Bartels曲线

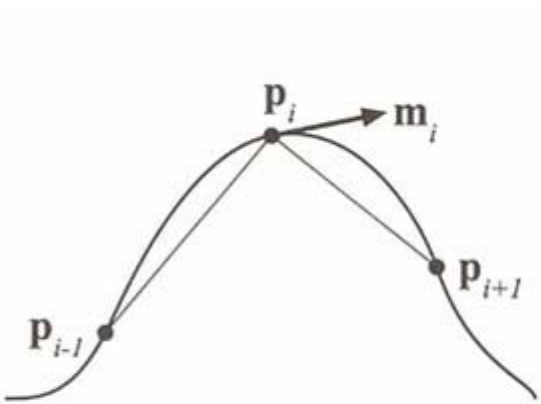
- 当使用多个点进行Hermite插值时，需要考虑插值点处切线的控制。

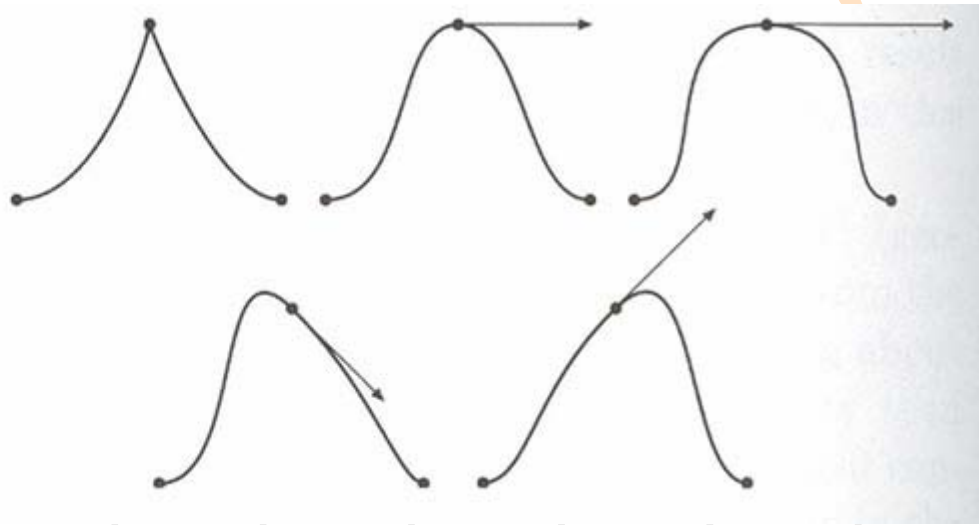
- 我们将介绍使用Kochanek-Bartels曲线计算切线的方法。

—假设，使用$n-1$段Hermite曲线对n个点$\mathbf{p}_0$，……，$\mathbf{p}_n$进行插值；

—如下图左所示，$\mathbf{p}_i$的切线值可由两段弦$\mathbf{p}_i-\mathbf{p}_{i-1}$和$\mathbf{p}_{i+1}-\mathbf{p}_i$计算。

Kochanek-Bartels曲线





左图为使用弦p_i-p_{i-1}和p_{i+1}-p_i计算p_i处切线m_i的示范。

右上图分别为不同张力参数$a\approx1$，$a\approx0$，$a\approx-1$情况下的曲线。

右下图分别为偏移参数b为负，b为正情况下的曲线。

Kochanek-Bartels曲线

- 切线计算过程：
  - 引入张力参数(tension parameter) *a*控制插值点切向长度，以达到控制曲线在该点弯曲程度的目的。

切线计算公式：
$$m_i = \frac{1-a}{2}((p_i - p_{i-1}) + (p_{i+1} - p_i))$$

上页图右上展示不同张力系数对曲线的影响， ***a***值越大，曲线的弯曲程度越大(***a*>1**时，会产生环)。

  - 引入偏移参数(bias parameter) *b*控制切线方向。

切线计算公式(假设***a*=0**)：
$$\mathrm{m}_i = \frac{1+b}{2}(\mathrm{p}_i - \mathrm{p}_{i-1}) + \frac{1-b}{2}(\mathrm{p}_{i+1} - \mathrm{p}_i)$$

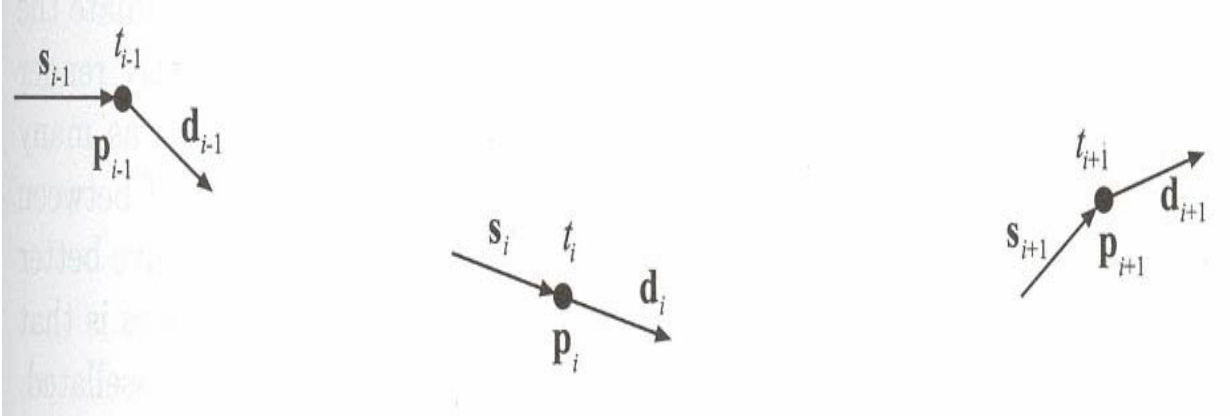
上页图右下展示不同偏移系数对曲线的影响，***b*>0**时，弦$\mathbf{p}_i$-$\mathbf{p}_{i-1}$对切线方向影响更大，***b*<0**，弦$\mathbf{p}_{i+1}$-$\mathbf{p}_i$对切线方向影响更大。

Kochanek-Bartels曲线

- 结合张力参数和偏移参数，得到切线计算公式如下：

$$m_i = \frac{(1-a)(1+b)}{2}(p_i - p_{i-1}) + \frac{(1-a)(1-b)}{2}(p_{i+1} - p_i)$$

连续性参数c



- 在每个点$\mathbf{p}_i$处定处义两个切向，入向切线$\mathbf{s}_i$和出向切线$\mathbf{d}_i$。并且在每段曲线$\mathbf{p}_i\mathbf{p}_{i+1}$中使用的是切线$\mathbf{d}_i$和$\mathbf{s}_{i+1}$。
- 切线$\mathbf{d}i$和$\mathbf{s}_{i+1}$计算如下:

$$\mathrm{s}_i = \frac{1-c}{2}(\mathrm{p}_i - \mathrm{p}_{i-1}) + \frac{1+c}{2}(\mathrm{p}_{i+1} - \mathrm{p}_i)$$

$$\mathrm{d}_i = \frac{1+c}{2}(\mathrm{p}_i - \mathrm{p}_{i-1}) + \frac{1-c}{2}(\mathrm{p}_{i+1} - \mathrm{p}_i)$$

Kochanek-Bartels曲线

- 结合张力参数、偏移参数、连续性参数，得到切线计算公式如下：

$$s_i = \frac{(1-a)(1+b)(1-c)}{2}(p_i - p_{i-1}) + \frac{(1-a)(1-b)(1+c)}{2}(p_{i+1} - p_i)$$

$$d_i = \frac{(1-a)(1+b)(1+c)}{2}(p_i - p_{i-1}) + \frac{(1-a)(1-b)(1-c)}{2}(p_{i+1} - p_i)$$

- 以上都是针对所有曲线使用相同时间t分布的讨论，为了使用不同t分布，做如下变换：($\Delta_i = t_{i+1} - t_i$)

$$s_i' = s_i \frac{2\Delta_i}{\Delta_{i-1} + \Delta_i},$$

$$d_i' = d_i \frac{2\Delta_i}{\Delta_{i-1} + \Delta_i}$$

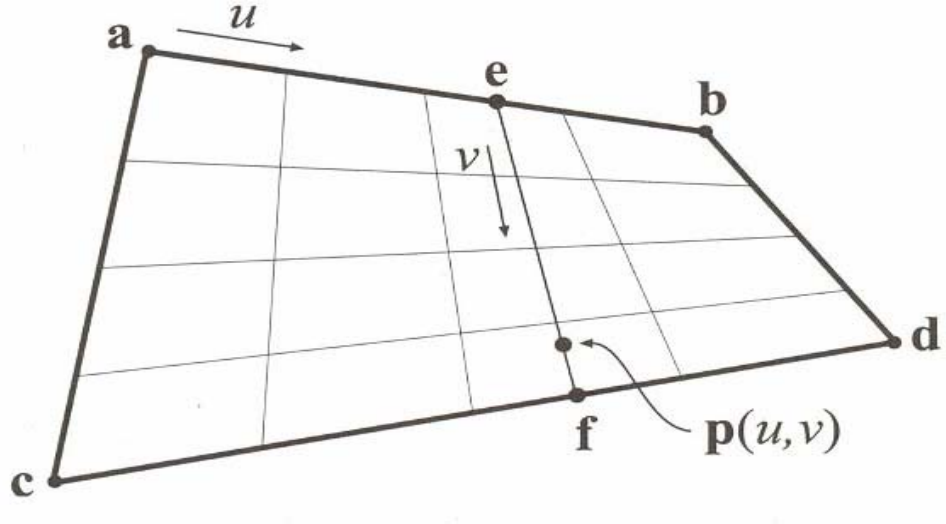
参数曲面

- 主要介绍**Bézier**曲面、三角域**Bezier**曲面、**Normal-Patches**等内容。

- 参数曲面由一系列控制顶点所定义，可用于雕塑曲面的建模。

Bézier曲面片

- 将前面阐述的Bézier曲线的一个参数扩展为两个参数，就得到Bézier曲面。
- 首先，我们讨论将线性插值扩展到双线性线形插值。



Example:双线性插值点a，b，c，d得到p(u，v)

Bézier曲面片

- 计算上图所示的曲面**P(u,v):**

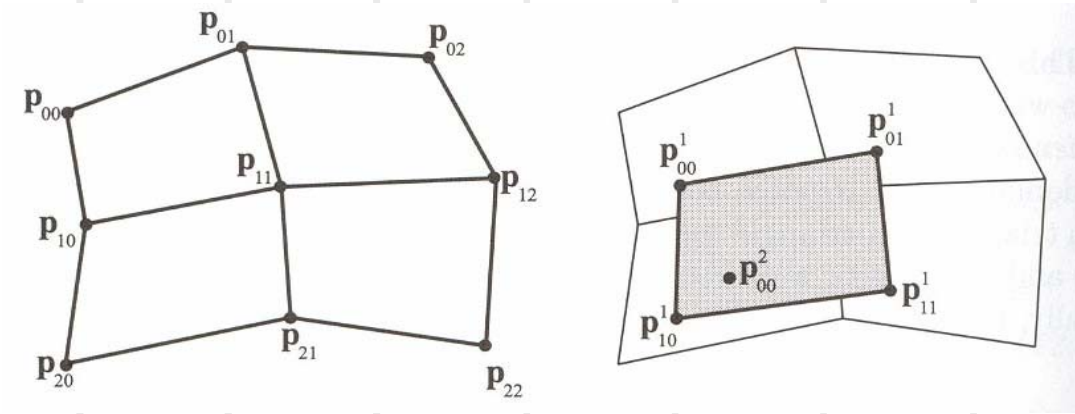
$$e = (1-u)a + ub$$
$$f = (1-u)c + ud \qquad (12.23) \qquad (u,v) \in [0,1] \times [0,1]$$

$$\begin{aligned} p(u,v) &= (1-v)e + vf \\ &= (1-u)(1-v)a + u(1-v)b + (1-u)vc + uvd \end{aligned} \qquad (12.24)$$

- 式**12.24**所表示的是最简单的空间参数曲面，曲面上的点由不同的参数**(u,v)**定义。
- 当参数域为矩形的时候，曲面被称为面片**(patch)**。

Bézier曲面片

例子：我们通过3×3的网格构造双二次 Bézier曲面片。



- 双线性插值相邻的每4个点计算四个中间点p^1_{ij}；
- 在中间点再次插值，得到曲面上一点p^2_{00}。

Bézier曲面片

- de Casteljau方法构造Bézier曲面片就是对反复双线性插值方法的扩展.
  - 对于n次曲面，需要$(n+1)^2$个控制顶点。插值得到的点$\mathbf{p}^n_{00}(u,v)$就是曲面片上的点(u,v)。
  - de Casteljau [patch]:

$$\mathrm{p}^k_{i,j}(u,v) = (1-u)(1-v)\mathrm{p}^{k-1}_{i,j} + u(1-v)\mathrm{p}^{k-1}_{i,j+1} + (1-u)v\mathrm{p}^{k-1}_{i+1,j} + uv\mathrm{p}^{k-1}_{i+1,j+1}$$

$$k = 1...n, i = 0...n-k, j = 0...n-k$$

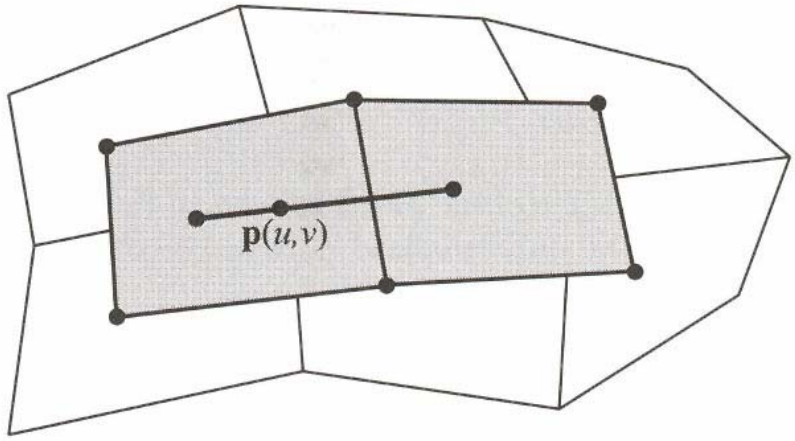
Bézier曲面片（Patches）

- **Berstein多项式表示的Bézier曲面片：**

$$\mathrm{p}(u,v)=\sum_{i=0}^{m}B_i^m(u)\sum_{j=0}^{n}B_j^n(v)\mathrm{p}_{i,j}=\sum_{i=0}^{m}\sum_{j=0}^{n}B_i^m(u)B_j^n(v)\mathrm{p}_{i,j}$$

$$=\sum_{i=0}^{m}\sum_{j=0}^{n}\binom{m}{i}\binom{n}{j}u^i(1-u)^{m-i}v^j(1-v)^{n-j}\mathrm{p}_{i,j}$$

如下图为曲面两个方向的次数**m**和**n**不同的情况：



Bézier曲面片

- 上式也可以表示为如下形式：

$$\begin{aligned} \mathrm{p}(u,v) &= \sum_{i=0}^{m} B_i^m(u) \sum_{j=0}^{n} B_j^n(v)\mathrm{p}_{i,j} \\ &= \sum_{i=0}^{m} B_i^m(u)\mathrm{q}_i(v) \end{aligned} \qquad (12.27)$$

其中，$\mathrm{q}_i(v) = \sum_{j=0}^{n} B_j^n(v)\mathrm{p}_{i,j}$ 即固定***v***值的**Bézier**曲线。

Bézier曲面的一些性质

- 取$(u,v)=(0,0)$, $(u,v)=(0,1)$, $(u,v)=(1,0)$, $(u,v)=(1,1)$,可以证明Bézier曲面片插值点$\mathbf{p}_{0,0},\mathbf{p}_{0,n},\mathbf{p}_{n,0},\mathbf{p}_{n,n}$;
- 曲面片的边界是由边界上的控制顶点确定的n次Bézier曲线;
- 曲面片角点的控制顶点的切线由边界上的Bézier曲线定义，且角点有两条切线;
- 曲面片在控制顶点的凸包内，且

$$\sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(u)B_j^n(v)=1$$

Bézier曲面的一些性质

- 旋转控制顶点再生成曲面片上的点，与先生成曲面片上的点再旋转控制顶点效果相同；
- 求导性质：

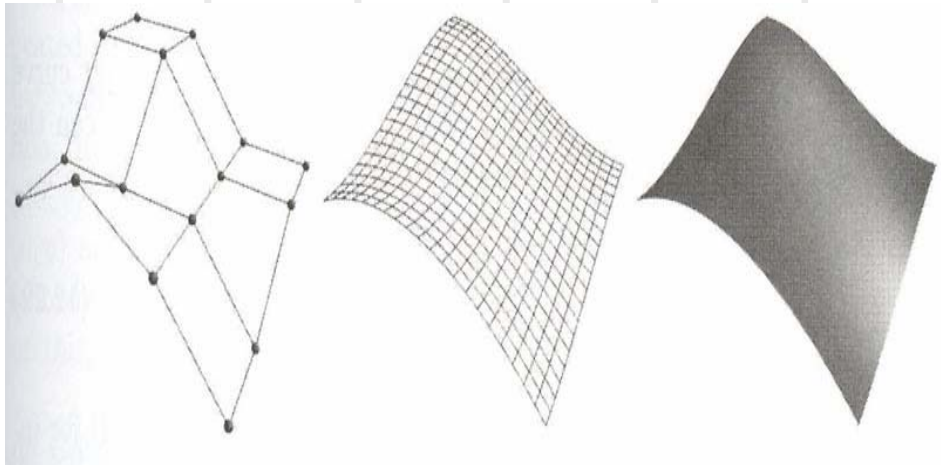
$$\frac{\partial \mathrm{p}(u,v)}{\partial u} = m\sum_{j=0}^{n}\sum_{i=0}^{m-1} B_i^{m-1}(u)B_j^n(v)[\mathrm{p}_{i+1,j} - \mathrm{p}_{i,j}]$$

$$\frac{\partial \mathrm{p}(u,v)}{\partial v} = n\sum_{i=0}^{m}\sum_{j=0}^{n-1} B_i^{m}(u)B_j^{n-1}(v)[\mathrm{p}_{i,j+1} - \mathrm{p}_{i,j}]$$

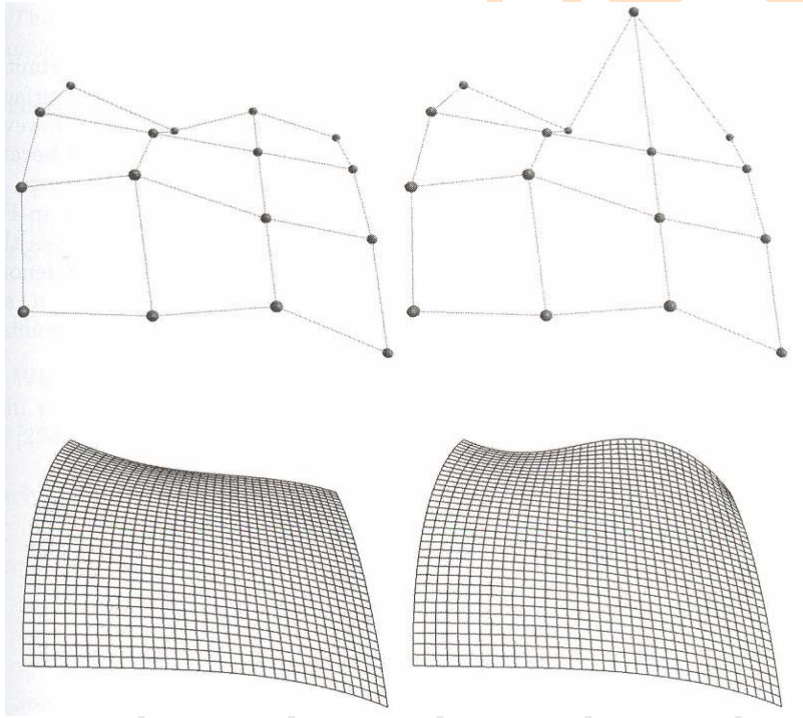
- 未单位化的法向为：

$$\mathrm{n}(u,v) = \frac{\partial \mathrm{p}(u,v)}{\partial u} \times \frac{\partial \mathrm{p}(u,v)}{\partial v}$$

Bézier曲面例子



4×4的Bézier曲面片示例



移动控制顶点产生的效果

有理Bézier曲面片

- 同Bézier曲线扩展到有理Bézier曲线一样，引入更大的自由度， Bézier曲面片可以扩展到有理Bézier曲面片:

$$\mathrm{p}(u,v)=\frac{\sum_{i=0}^{m}\sum_{j=0}^{n}\omega_{i,j}B_i^{\,m}(u)B_j^{\,n}(v)\mathrm{p}_{i,j}}{\sum_{i=0}^{m}\sum_{j=0}^{n}\omega_{i,j}B_i^{\,m}(u)B_j^{\,n}(v)} \qquad (12.29)$$

同理，下节讲到的三角域上的Bézier曲面同样可以扩展到有理Bézier三角曲面。

三角域上的Bézier曲面

- 虽然三角形通常被认为是比矩形更简单的几何单元，但是对于Bézier曲面却不是这样: Bézier曲面片比三角域上的Bézier曲面更简单。然而，因为三角形对于图形硬件来说是基本的单元，所以三角域上的Bézier曲面是非常有用的。

- 在三角化的网格中，三角域上的Bézier曲面的控制顶点如图12.16所示。如果三角域上的Bézier曲面是n次的，那么控制网格的每条边就有$n+1$个控制点。这些控制点用$\mathbf{p}^0_{i,j,k}$表示，有时可以简写为$\mathbf{p}_{ijk}$。对所有的控制点$i+j+k = n$，$i,j,k \geqslant 0$，控制顶点的总数为:

$$\sum_{x=1}^{n+1} x = \frac{(n+1)(n+2)}{2}$$

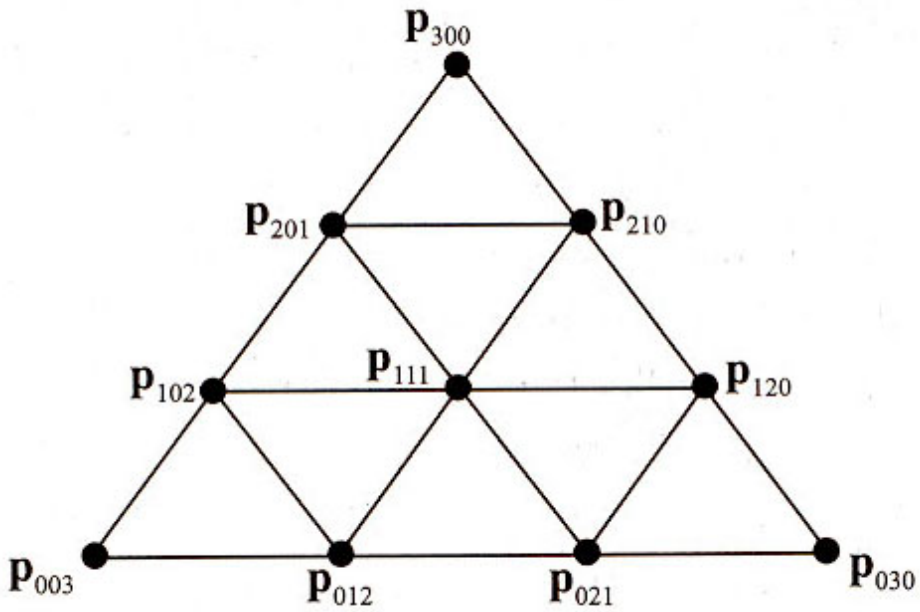


图12.16 三次三角域上的Bézier曲面的控制顶点

- 三角域上的Bézier曲面也是基于反复插值得到的。因为参数域是三角形，所以插值时用到重心（面积）坐标。
- 重心坐标: 如果三角形$\triangle \mathbf{p}_0\mathbf{p}_1\mathbf{p}_2$中有一个点，那么这个点可以这样描述:$\mathbf{p}(u,v)=\mathbf{p}_0+u(\mathbf{p}_1-\mathbf{p}_0)+v(\mathbf{p}_2-\mathbf{p}_0)=(1-u-v)\mathbf{p}_0+u\mathbf{p}_1+v\mathbf{p}_2$。这里$(u,v)$是重心坐标。
- 对于三角形中的任意点满足:$u\geqslant 0, v\geqslant 0, 1-u+v\geqslant 0 <=> u+v\leqslant 1$。

- 三角域**Bézier**曲面的**De Casteljau**算法

$$\mathrm{p}_{i,j,k}^{l}(u,v)=u\mathrm{p}_{i+1,j,k}^{l-1}(u,v)+v\mathrm{p}_{i,j+1,k}^{l-1}(u,v)+(1-u-v)\mathrm{p}_{i,j,k+1}^{l-1}(u,v),$$
$$l=1...n, i+j+k=n-l$$

- 使用**Bernstein**多项式形式表示的三角域**Bézier**曲面

$$\mathrm{p}(u,v)=\sum_{i+j+k=n} B_{ijk}^{n}(u,v)\mathrm{p}_{ijk}$$

现在**Bernstein**多项式依赖于u和v两个变量，其表达式如下：

$$B_{ijk}^{n}(u,v)=\frac{n!}{i!\,j!k!}u^{i}v^{j}(1-u-v)^{k},\quad i+j+k=n$$

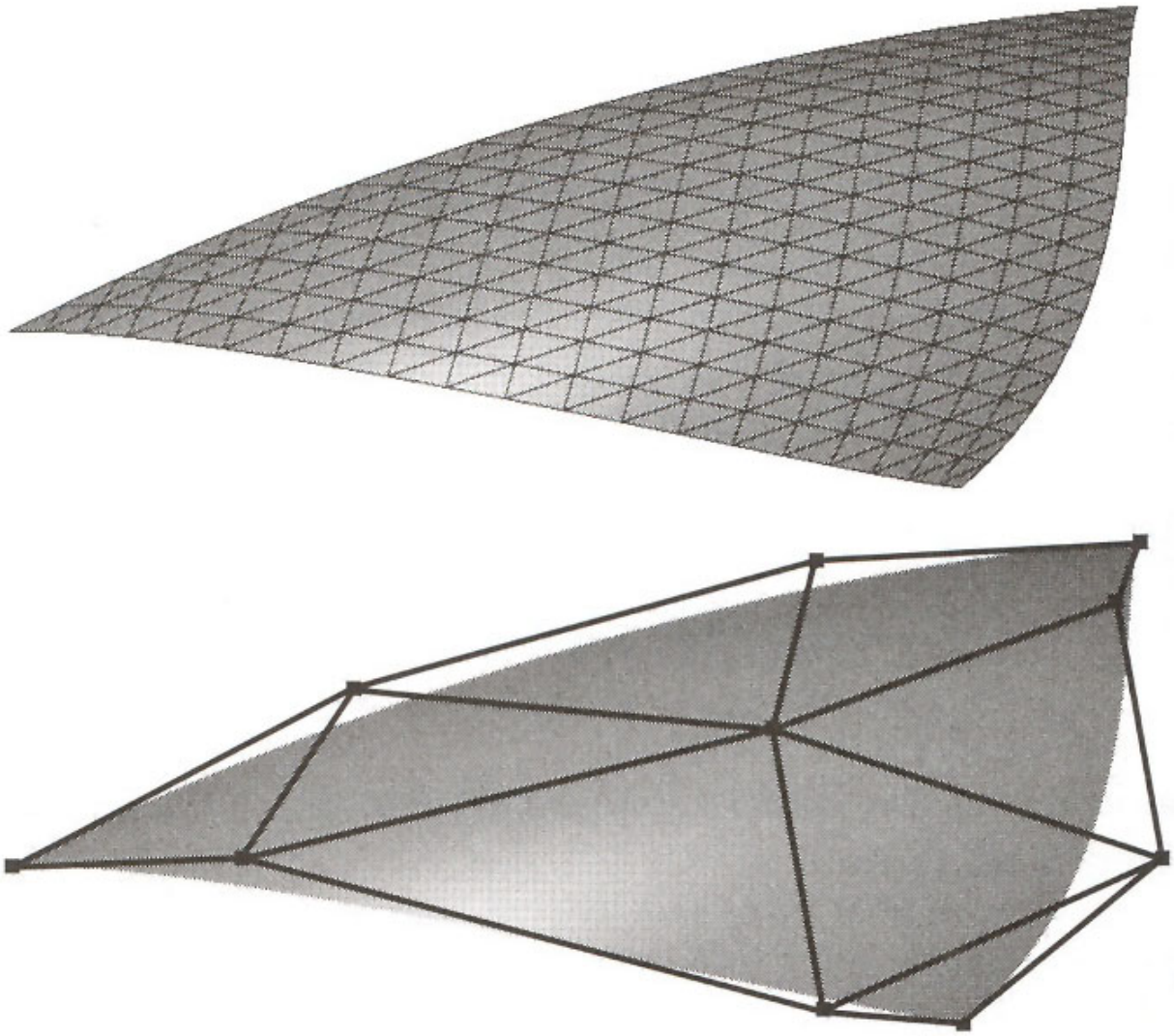
当$i,j,k<0$或$i,j,k>n$，**Bernstein**多项式为零，即$B^{n}_{ijk}(u,v)=0$。

- 三角域**Bézier**曲面的偏导数如下：

$$\frac{\partial \mathrm{p}(u,v)}{\partial u} = \sum_{i+j+k=n-1} B_{ijk}^{n-1}(u,v)\mathrm{p}_{i+1,j,k}$$

$$\frac{\partial \mathrm{p}(u,v)}{\partial u} = \sum_{i+j+k=n-1} B_{ijk}^{n-1}(u,v)\mathrm{p}_{i,j+1,k}$$

- 三角域**Bézier**曲面的性质：
  - 插值三个控制角点；
  - 曲面边界是边界控制点描述的**Bézier**曲线；
  - 曲面在控制点的凸包内；
  - 旋转控制点然后生成**Bézier**三角曲面上的点与先生成点，然后再旋转它们结果相同。

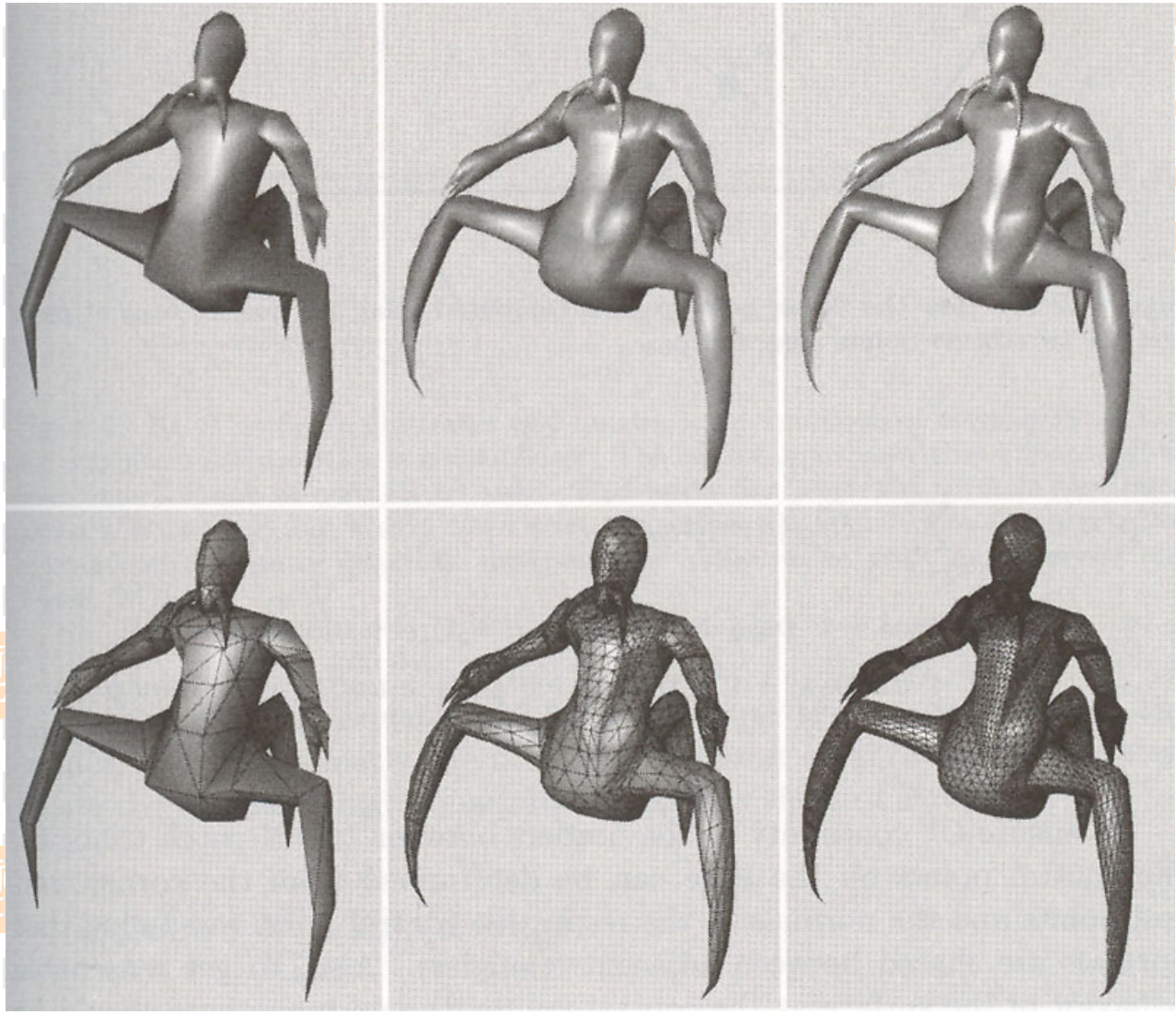


上图： **Bézier**三角曲面的网格图；

下图：着色后的曲面与控制网格。

N-Patches

- 我们将讨论三角域Bezier曲面的一个应用。
- 给定一个在每个顶点带有法向的三角形网格，N-Patches方法的目的是构造一个基于三角形的具有良好外观的曲面。
- 这里N-Patches是“Normal-Patches”的简写，也称为*PN*三角片。这个方法试图通过生成一个三角域Bezier曲面代替每个三角形来改善三角网格的绘制和轮廓。
- 图形硬件之所以能快速生成这些表面的原因：多边形化是通过每个三角形的顶点和法向完成的，而不需要邻域信息。只要将API做一点修改，所需要的是一个判断是否要生成N-patches的标志和多边形化层数。如下图所示。



每一列显示同一模型的不同层次细节。左边显示初始模型，由414个三角片组成；中间的模型有3726个三角片；右边有20286个三角片。

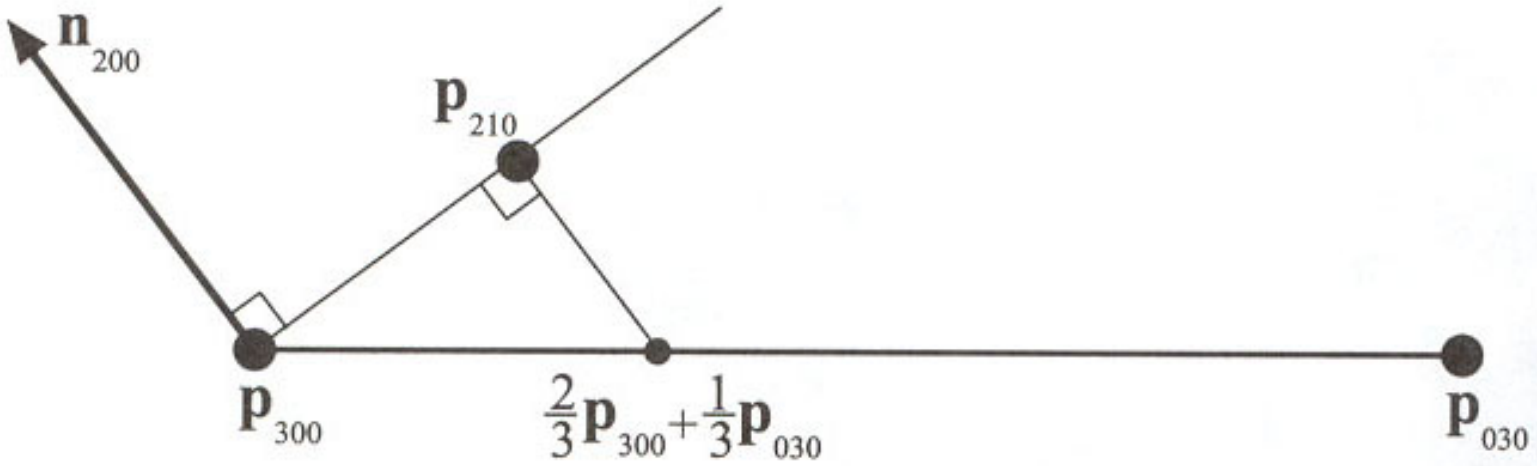
N-Patches算法描述

- 假设我们有一个三角形，其顶点为p_{300},p_{030},p_{003}，其法向为n_{200},n_{020},n_{002}。基本思想是对每个初始三角形，使用这些信息生成一个三次三角域**Bézier**曲面，然后从这个三角域**Bézier**曲面生成我们所想要的任意数目的三角片。

$$p(u,v)=\sum_{i+j+k=3} B_{ijk}^{3}(u,v)p_{ijk}$$
$$=u^3p_{300}+v^3p_{030}+w^3p_{003}+3u^2vp_{210}+3u^2wp_{201}$$
$$+3uv^2p_{120}+3v^2wp_{021}+3vw^2p_{012}+3uw^2p_{102}+6uvwp_{111}$$

这里$w=1-u-v$。

点P_{210}的计算



点p_{210}使用p_{300}处的法向n_{200}和角点p_{300}和p_{030}计算得到。

控制点的计算

- 为了保证两个**N-patch**三角片在边界上C^0连续，边上的控制点可以由控制角点及其各自控制点处的法向确定(相邻的三角片共享法向)。
- 为了能在控制点处能反映曲面的走向。我们采用以下策略计算边界上六个不同的控制点：

点$\mathbf{p}_{210}$的计算：我们使用控制点$\mathbf{p}_{300}$，$\mathbf{p}_{030}$和点$\mathbf{p}_{300}$处的法向$\mathbf{n}_{200}$。取点$\mathbf{2/3p}_{300}+\mathbf{1/3p}_{030}$，并且沿着法向$\mathbf{n}_{200}$的方向将它投影到由$\mathbf{p}_{300}$与$\mathbf{n}_{200}$定义的切平面上。设法向量已经单位化，那么点$\mathbf{p}_{210}$可以计算如下：

$$p_{210}=\frac{1}{3}(2p_{300}+p_{030}-(n_{200}\cdot(p_{030}-p_{300}))n_{200})$$

其它的边界点可以类似计算。内部控制点$\mathbf{p}_{111}$计算如下：

$$p_{111}=\frac{1}{4}(p_{210}+p_{120}+p_{102}+p_{201}+p_{021}+p_{012})-\frac{1}{6}(p_{300}+p_{030}+p_{003})$$

法向的计算

- **Vlachos et al.**使用如下二次插值来计算法向（而不是直接用三角域**Bezier**曲面的法向计算公式）：

$$\mathrm{n}(u,v) = \sum_{i+j+k=2} B_{ijk}^{2}(u,v)\mathrm{n}_{ijk}$$
$$= u^2\mathrm{n}_{200} + v^2\mathrm{n}_{020} + w^2\mathrm{n}_{002} + uv\mathrm{n}_{110} + uw\mathrm{n}_{101} + vw\mathrm{n}_{011}$$

这个式子可以看作为一个二次的**Bézier**三角曲面，这里控制点是六个不同的法向量。

- 在上述公式中，次数(二次)的选择是非常有道理的，因为导数比真实三角域**Bézier**曲面次数低一次，而法向的线性插值又不能描述变形情况，如下图所示。

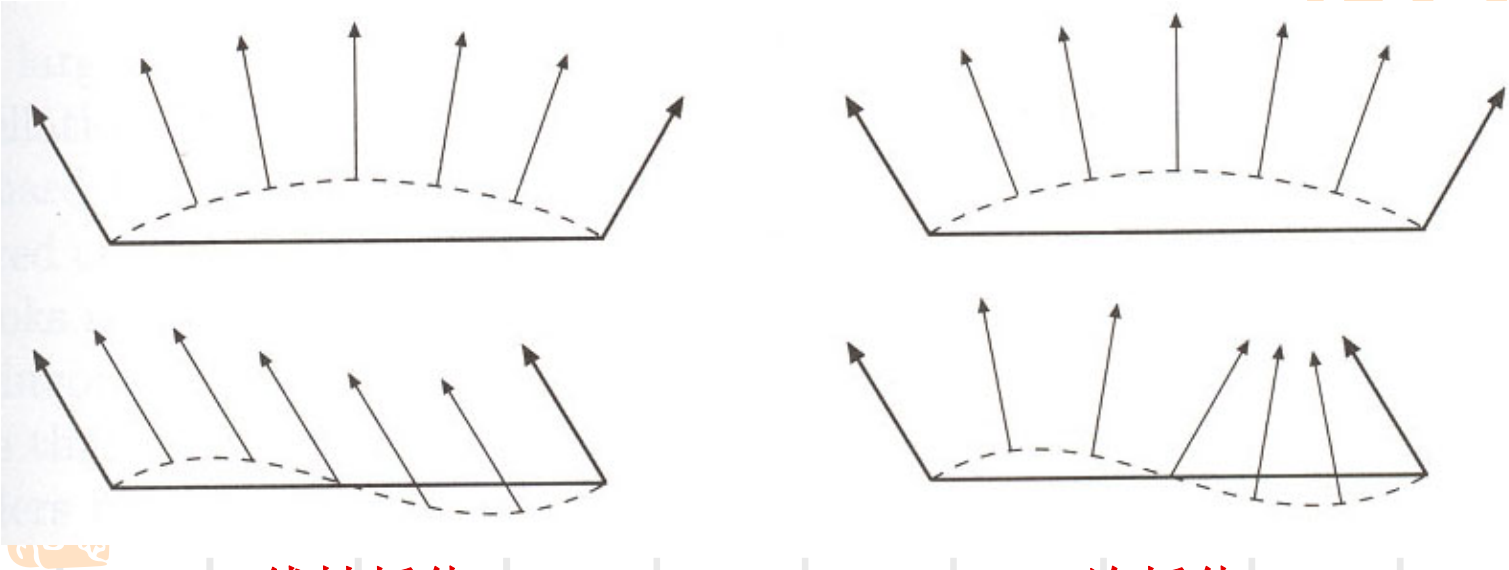
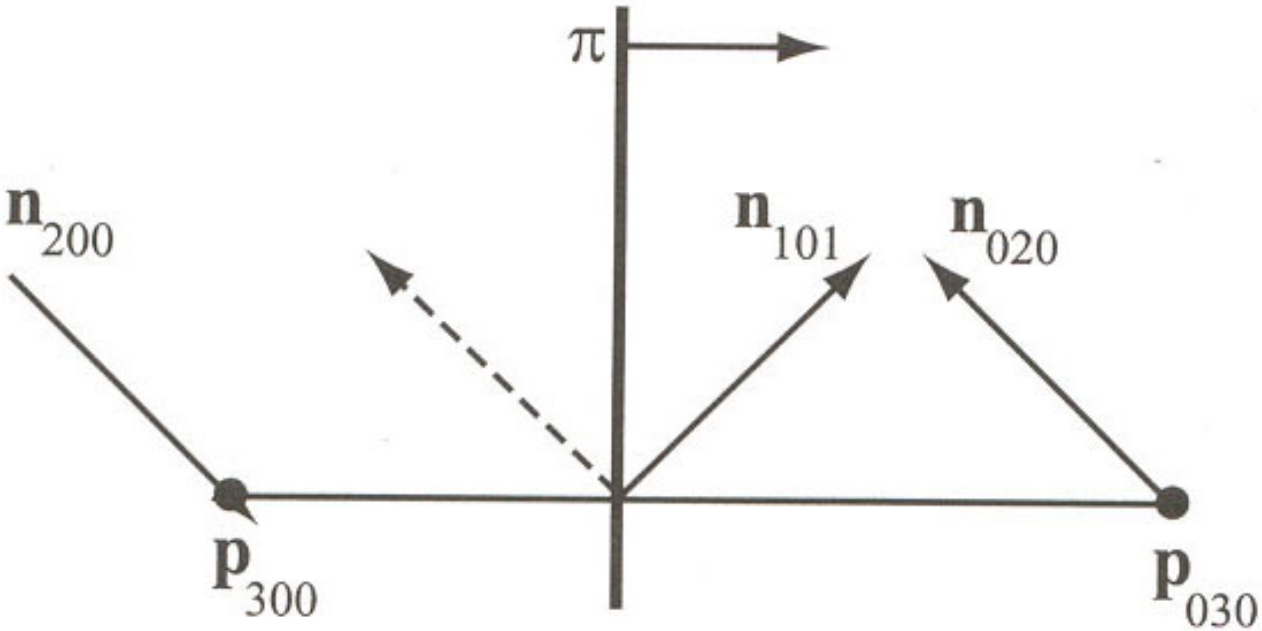


图12.20 为什么需要法向的二次插值而线性插值是不够的？

左列所示为法向线性插值的情况。当用法向来描述一个凸表面（上图）时，这种方法就比较好。

但是，当曲面有变形时，这种方法就会失效。



N-patches法向n_{101}的构造。虚线表示的是n_{200}与n_{020}的平均，n_{101}是这个法向量在π平面的反射向量。π平面的法向与p_{030}-p_{300}平行。

n_{110},n_{101},n_{011}的计算

- 我们可以直接使用n_{200},n_{020}的平均值去计算n_{110}。但是当n_{200}=n_{020}时，会出现图12.20左下图的问题。
- 假设所有法向都已单位化，n_{110}的没有单位化的计算公式如下（见上页图）：

$$n'_{110} = n_{200} + n_{020} - 2\frac{(p_{030} - p_{300})\cdot(p_{300} + p_{030})}{(p_{030} - p_{300})\cdot(p_{300} - p_{030})}(p_{030} - p_{300}) \quad (12.38)$$

- van Overveld和Wyvill最开始使用系数3/2而不是2。至于哪个值更好，很难从图像中判断。但是使用2给出了平面中真实反射方向的一个很好的插值。

这个方法的优点是曲面可以得到一个比较好的轮廓和形状，而计算量相对较少。而且完成这项工作只要在已有的代码中做小小的修改即可。所有要做的工作就是多边形化，得到原始模型的一些LOD。而这些工作使用硬件可以很简单地实现。

构造LOD (Level of Detail)

- 初始三角形数据是**LOD** 0。随着**LOD**层数增加，在三角形每条边上的点数增加。因而，**LOD** 1在三角形的每边引入一个新的点，从而在**Bezier**三角片上生成四个子三角形；**LOD** 2在每条边引入两个点，生成9个三角形。

- 一般来讲，**LOD** n生成$(n+1)^2$个三角形。为了避免**Bézier**三角片之间出现裂缝，网格中的每个三角形必须分割成相同的**LOD**。这是这个方法的一个缺点，因为小的三角形将同大的三角形进行相同的多边形化。

- 尽管**Bézier**三角片之间是C^0连续的，但是在多数情况下，这是可以接受的。这是因为三角片上的法向是连续的，所以一个**N-patches**集可以模拟一个G^1连续的曲面。

API与硬件支持

- OpenGL API可以支持Bézier曲面片和有理Bézier曲面片的显示。然而，这并不意味着所有的硬件支持在硬件中进行多边形化的过程。

- NVIDIA’s GeForce3包含了OpenGL扩展，使用他们自己的求值程序可以处理Bézier曲面片和三角域Bézier曲面(包括有理和非有理的)。他们还可以处理B样条。

- DirectX 8 API和OpenGL扩展支持对N-patch的处理。

- DirectX 8支持法向插值，但仅限于线性插值，DirectX 8.1允许法向的二次插值。但是法向插值计算量非常大，而且二次插值比线性插值更大。

- 除了标准的N-patch插值(三次三角域Bézier曲面)，DirectX 8.1还允许顶点位置的线性插值。这意味着，一个三角形可以被多边形化成许多法向经过插值的共面的更小的三角形。

曲面连续性

- 当我们使用**Bézier**曲面生成物体时，经常希望能将几个不同的**Bézier**曲面拼接在一起形成一张复杂的曲面。为了得到一个比较好看的结果，我们必须保证整个曲面要有一定的连续性。这与曲线是相同的思想。
- 假设两张双三次**Bézier**曲面片将被拼接在一起。每张曲面片有**4×4**个控制顶点，如图**12.22**所示，左图$\mathbf{a}_{ij}$，右图$\mathbf{b}_{ij}$，这里$0 \leqslant i,j \leqslant 3$。
  - 为了保证C^0连续，两张曲面片在边界上必须共享相同的控制顶点，也就是$\mathbf{a}_{3j}=\mathbf{b}_{0j}$。
  - 为了保证C^1连续，除了上面的条件，还要求对于每个j，点$\mathbf{a}_{2j}$，$\mathbf{b}_{0j}$，$\mathbf{b}_{1j}$要共线，而且要满足$\|\mathbf{a}_{2j}-\mathbf{b}_{0j}\|=k\|\mathbf{b}_{0j}-\mathbf{b}_{1j}\|$，这里$k$为一个常数，且对所有的$j$都相同。

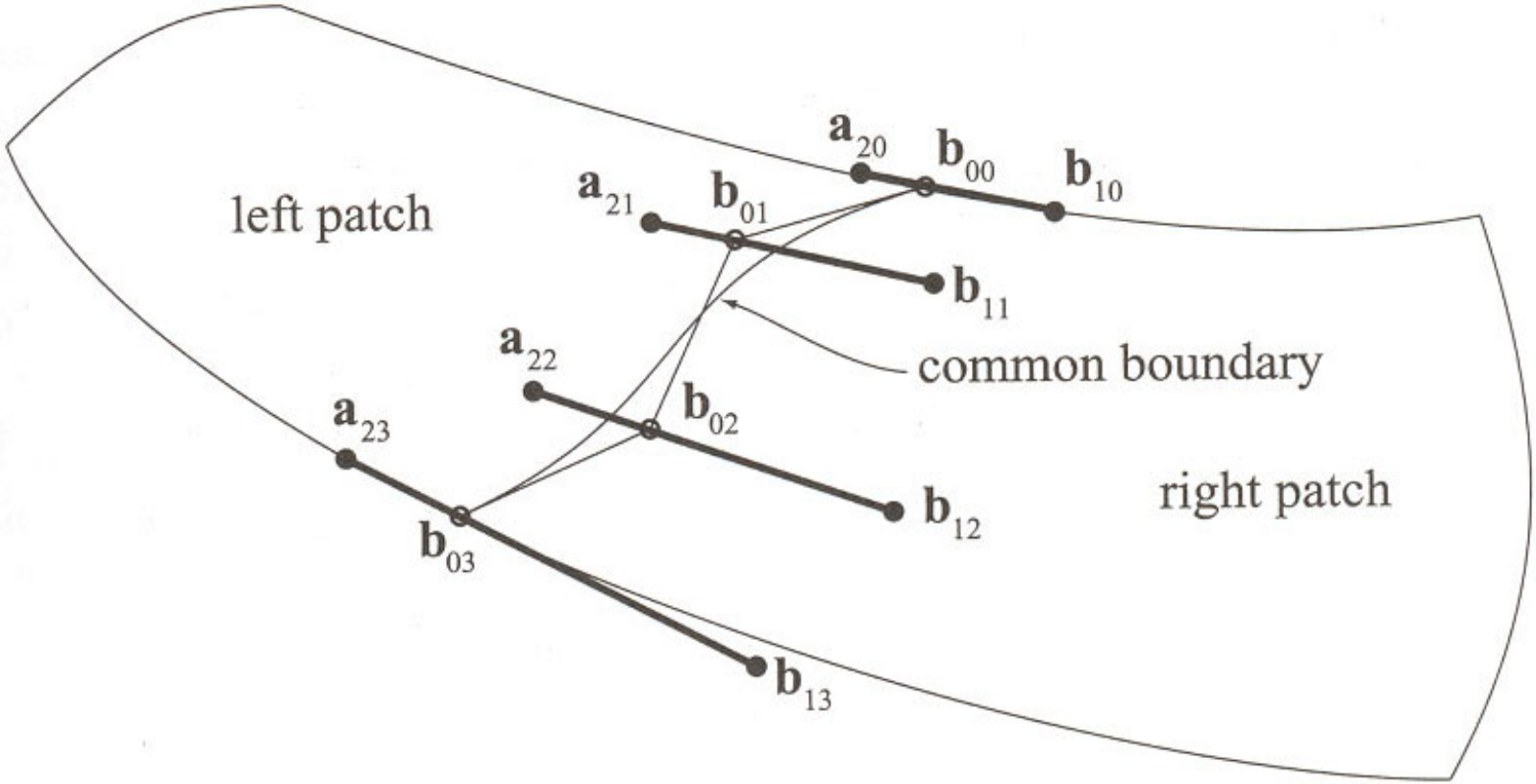


图12.22 怎样拼接两张Bézier曲面片，使得具有C^1连续

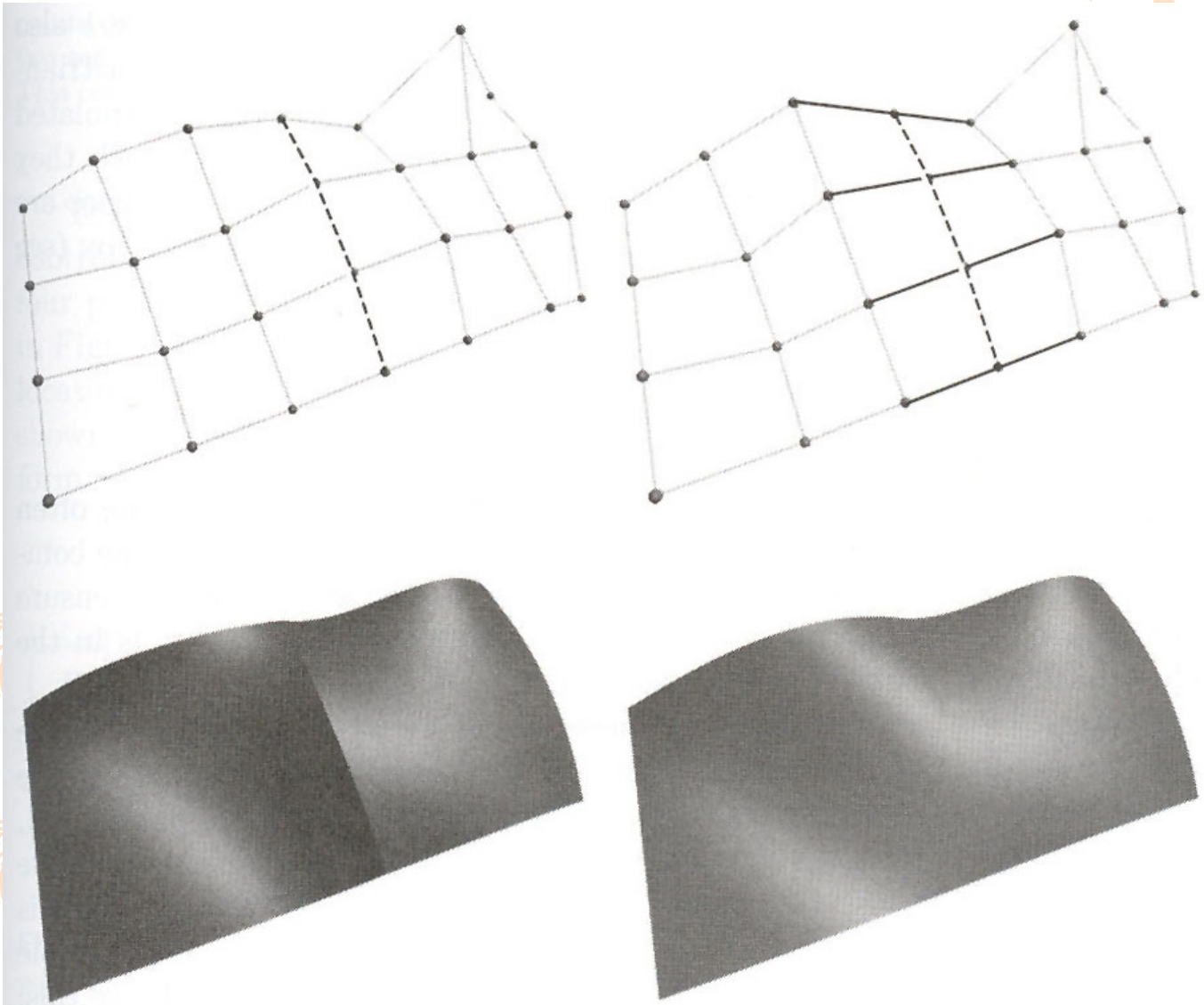


图12.23 左图是C^0连续拼接，右图是C^1连续拼接。

- 前面讲述的构造光滑曲面的方法可能会用完设置控制顶点的自由度。

当拼接四片曲面片时，共享一个顶点，如图12.24所示。右图是结果曲面，九个控制顶点处在同一个平面上，也即他们必须形成一个双线性的曲面片。结果显示如果要在角点处满足G^1连续，它必须要使得九个点共面。

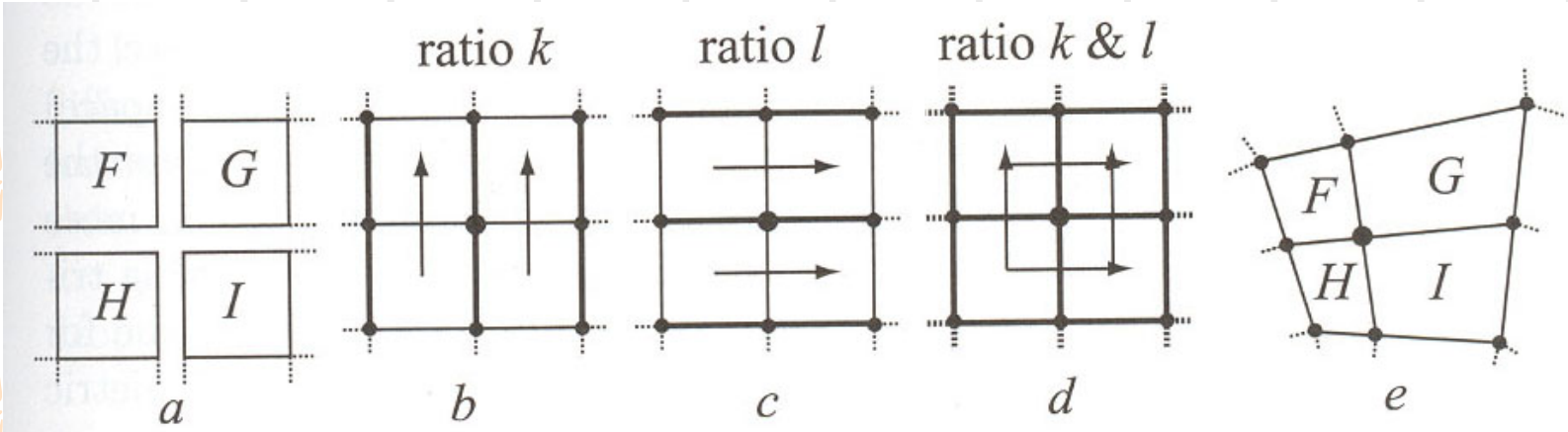


图12.24　拼接四块Bézier曲面片

- 三角域Bézier曲面的连续性更加复杂。当使用许多Bézier曲面构造一个复杂的物体时，我们很难确定在整个边界上能获得怎样的连续性。一个方法是使用后面介绍的细分曲面。

注意:要在边界上获得比较好的纹理必须要有C^1连续。对于反射和渲染计算，得到好的结果需要G^1连续。C^1或者更高阶的连续性能得到更好的结果。

有效多边形化 (Efficient Tessellation)

- 为了在实时绘制环境下使用参数曲面，我们需要把曲面转化为四边形或三角形。这个过程一般称作为多边形化。均匀多边形化是一种最简单的多边形化方式。

Example:假设Bézier参数曲面片$\mathbf{p}(u,v)$，我们想通过对曲面片的每边计算11个点来对曲面片进行多边形化，这样可以得到10×10×2＝200个三角形。最简单的办法是在参数空间uv平面进行均匀采样。这样，我们对所有$(u_k, v_l)= (0.1k,0.1l)$计算函数值$\mathbf{p}(u,v)$，这里k和l是0到10之间的任意整数。曲面上的四个点$\mathbf{p}(u_k,v_l)$,$\mathbf{p}(u_{k+1},v_l)$,$\mathbf{p}(u_{k+1},v_{l+1})$, $\mathbf{p}(u_k,v_{l+1})$可以构成两个三角形。

Tessellation主要方法

- 差分方法：在曲面上均匀生成三角形时能减少计算量；

- 自适应多边形化方法：多边形化时三角形的大小将自适应于曲面的形状，i.e.，在曲面越弯曲的地方三角形越多，而在不需要的地方将避免浪费三角形；

- 非整型多边形化方法：它在参数曲面的边界上使用独立的非整型多边形化因子。这使得曲面具有更光滑的层次细节。

差分方法(Differencing Schemes)

- 主要介绍两个方法:
  - 向前差分(**Forward Differencing**)
  - 中心差分(**Central Differencing**)

两个方法的主要目的是为了减少多项式计算的计算量。

向前差分(Forward Differencing)

- 一条**Bezier**曲线可以写成如下形式：

$$p(t)=\sum_{i=0}^{n} t^i c_i \qquad (12.39)$$

实际上，每一条多项式曲线都可以写成这种形式。

- 上式可以看作是三个独立的多项式(分别是**x,y,z**) .
这样我们可以仅仅讨论一个标量多项式：

$$f(t)=\sum_{i=0}^{n} t^i c_i = c_0 + tc_1 + t^2 c_2 + \cdots + t^n c_n \qquad (12.40)$$

- 向前差分是一种在均匀*t*值空间中快速计算多项式的值的技术。
- 一般来讲，我们想计算曲线在*t*=*sk*处的值，这里*s* 是一个正实数，*k*是一个非负整数。为了简化表示方式，我们引入$f_k=f(sk)$，而且有如下向前差分：

$$\begin{aligned}
\delta_k^1 &= f_{k+1} - f_k && \text{(第一次向前差分)} \\
\delta_k^2 &= \delta_{k+1}^1 - \delta_k^1 && \text{(第二次向前差分)} \\
\delta_k^3 &= \delta_{k+1}^2 - \delta_k^2 && \text{(第三次向前差分)} \qquad (12.41) \\
&\cdots \\
\delta_k^n &= \delta_{k+1}^{n-1} - \delta_k^{n-1} && \text{(第n次向前差分)}
\end{aligned}$$

- 由(12.41)我们知道，对于n次多项式，所有的向前差分δ^{n+1}为零。
- 多项式的计算

在开始计算多项式之前，向前差分法要计算一些初值信息：

$$f_0, \delta_0^1, \delta_0^2, \delta_0^3, \cdots, \delta_0^n$$

算法的伪代码如下：

1: compute $f, \delta^1, \delta^2, \delta^3, \cdots, \delta^n$
2: for $i = 1$ to l
3: $\quad f = f + \delta^1$
4: $\quad \delta^1 = \delta^1 + \delta^2$
5: $\quad \delta^2 = \delta^2 + \delta^3$
6: $\quad \cdots$
7: $\quad \delta^{n-1} = \delta^{n-1} + \delta^n$
8: end

- 性能分析

从伪代码中，我们知道它计算了$l+1$次f的值；每一次循环迭代生成一个新的f值。值得注意的是在初始信息被计算出之后，计算f值只用到了加法。开始之后，每个曲线上的新点以$O(n)$的代价计算出来，这里n为多项式的次数。因为Bezier曲线的de Casteljau和Bernstein求值每个点要花费$O(n^2)$的代价，一般来讲，向前差分更快。

注意：在这个方法中，由于误差积累，对于大的循环舍入误差可能会被引入。

同样，这种方法可以用于曲面片的计算。

- 例子：

假设下面的多项式要在**t=0.1k，k=0,1,2,…,10**处求值：

$$f(t) = 2 + 5t + t^2 \qquad (12.42)$$

为了使用向前差分方法，计算出了初始的信息：

$$f_0 = 2 \qquad f_1 = 2.51 \qquad f_2 = 3.04$$

$$\delta_0^1 = f_1 - f_0 = 0.51 \qquad \delta_1^1 = f_2 - f_1 = 0.53 \qquad (12.43)$$

$$\delta_0^2 = \delta_1^1 - \delta_0^1 = 0.02$$

现在带着f=2，δ^1=0.51，δ^2=0.02进入循环。在第一次迭代，我们可以得到f=2+0.51=2.51，δ^1=0.51+0.02=0.53；在第二次迭代，f=2.51+0.53=3.04，δ^1=0.53+0.02=0.55;第三次迭代，f=3.04+0.55=3.59，δ^1=0.55+0.02=0.57；一直计算下去……

中心差分(Central Differencing)

- 中心差分也是计算曲线曲面的一种有效方法。
- 标量函数$f(x)$可以通过泰勒展开如下：

$$f(x+\Delta)=\sum_{k=0}^{\infty}\frac{\Delta^k}{k!}f^{(k)}(x) \qquad (12.44)$$

在上式中，$f^{(k)}$是f的k阶导数。上面的公式的意思也就是如果所有的导数在某一点x处可以计算求值，那么函数f在$(x+\triangle)$也可以有效计算出来。因为一条曲线可以描述为$\mathbf{p}(t)=(x(t),y(t),z(t))$，所以对于一条曲线，需要三次使用上面的公式。

- 假设一条三次**Bezier**曲线：
  这里向量**u,v,w,x**是合并同类项之后**ti(i=0123)**的系数。

$$\begin{aligned} \mathrm{p}(t) &= B_0^3\mathrm{p}_0 + B_1^3\mathrm{p}_1 + B_2^3\mathrm{p}_2 + B_3^3\mathrm{p}_3 \\ &= (1-t)^3\mathrm{p}_0 + 3t(1-t)^2\mathrm{p}_1 + 3t^2(1-t)\mathrm{p}_2 + t^3\mathrm{p}_3 \\ &= t^3\mathrm{u} + t^2\mathrm{v} + t\mathrm{w} + \mathrm{x} \end{aligned} \qquad (12.45)$$

曲线函数的导数如下：

$$\begin{aligned} \mathrm{p}'(t) &= 3t2\mathrm{u} + 2t\mathrm{v} + \mathrm{w}, \\ \mathrm{p}''(t) &= 6t\mathrm{u} + 2\mathrm{v}, \\ \mathrm{p}^{(3)}(t) &= 6\mathrm{u}, \\ \mathrm{p}^{(k)}(t) &= 0, \quad k \geq 4. \end{aligned} \qquad (12.46)$$

- 由于函数的四阶以上导数都为零，所以方程**(12.44)**可以非常有效地进行计算。因为**p(***t*+△**)**和**p(***t*-△**)**的奇数项符号相反，所以加上他们可以简化计算：

$$p(t-\Delta)+p(t+\Delta)=2p(t)+\Delta^2 p''(t)$$

$$\Leftrightarrow \qquad (12.47)$$

$$p(t)=\frac{p(t-\Delta)+p(t+\Delta)-\Delta^2 p''(t)}{2}$$

- p"可以使用相似的方法进行计算：

$$p''(t)=\frac{p''(t-\Delta)+p''(t+\Delta)}{2} \qquad (12.48)$$

- 为了能够使用以上公式，初值**p(**0**),p(**1**),** p"(0), p"(1) 需要预先计算。然后可很快算出**p(**0.5**),p"(0.5)** 的值，进而算出**p(**0.25**)**和**p(**0.75**)**等一系列值。

中心差分性能分析

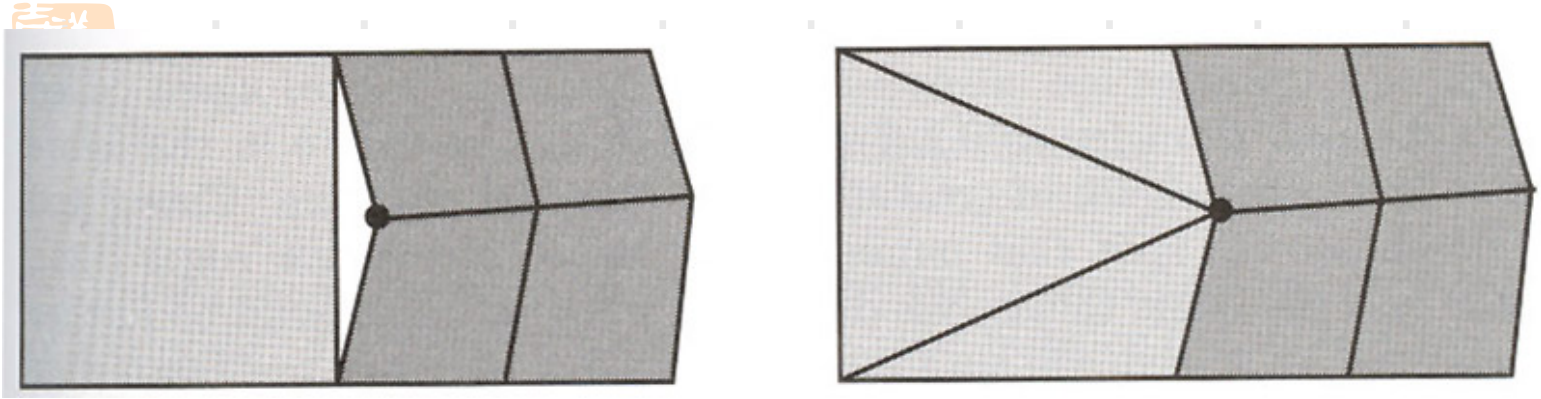
- 中心差分技术可以应用到矩形和三角形的曲面片。
- **DeLoura展示了怎样在双三次Bezier曲面片中使用该方法。为了计算这种曲面上9×9个点，直接使用方程(12.26)描述的Bernstein-Bezier公式，要使用大概12,500次运算。相反，中心差分只需要大约3,000次运算。**

Why自适应多边形化？

- 如果采样率足够高，均匀细分能够得到很好的结果。
- 然而，在一张曲面上，一些区域也许并不需要象另外一些区域一样需要那么多的三角片。例如：曲面弯曲很厉害的地方需要比较密的三角片，而在曲面很平坦的地方，只要数量很少的三角片就可以近似逼近它。
- 对于这个问题，均匀细分是无法解决的，自适应多边形化可以很好地解决这个问题。

自适应多边形化
(Adaptive Tessellation)

- 自适应多边形化方法在进行多边形化时，多边形的个数可以依赖于一些测量标准(如曲率，三角形边的长度或者屏幕大小)。注意，要避免不同的多边形化区域之间出现裂缝。



左图，在两个区域之间出现裂缝，因为右边的多边形化率高于左边。右图解决了问题。

参数曲面的自适应多边形化算法

- 对于矩形参数域的参数曲面，首先在参数域的角点上生成新的点(0,0),(0,1),(1,0),(1,1)。这四个点可以建立起两个三角形，这两个三角形分别用他们的点和各自的参数值表示，然后调用递归的多边形化算法。
- 对于参数域不是矩形域(如三角形域)的曲面也可以使用这个算法。

算法的伪代码描述

```
        AdaptiveTessellate(p,q,r)
1:      tessPQ = not curveTessEnough(p,q);
2:      tessQR = not curveTessEnough(q,r);
3:      tessRP  = not curveTessEnough(r,p);
4:      if(tessPQ and tessQR and tessTP)
5:              AdaptiveTessellate(p,(p+q)/2,(p+r)/2);
6:              AdaptiveTessellate(q,(q+r)/2,(q+p)/2);
7:              AdaptiveTessellate(r,(r+p)/2,(r+q)/2);
8:              AdaptiveTessellate((p+q)/2,(q+r)/2,(r+p)/2);
9:      else if(tessPQ and tessQR)
10:             AdaptiveTessellate(p,(p+q)/2,r);
11:             AdaptiveTessellate((p+q)/2,(q+r)/2,r);
12:             AdaptiveTessellate((p+q)/2,q,(q+r)/2);
13:     else if(tessPQ)
14:             AdaptiveTessellate(p,(p+q)/2,r);
15:             AdaptiveTessellate(q,r,(p+q)/2);
16:     else if(not triTessEnough(p,q,r))
17:             AdaptiveTessellate((p+q+r)/3,p,q);
18:             AdaptiveTessellate((p+q+r)/3,q,r);
19:             AdaptiveTessellate((p+q+r)/3,r,p);
20:             end;
```

算法说明

- 设初始三角形△**pqr**(这里假设点的信息包含它的三维位置以及参数坐标)。

  curveTessEnough(a,b):其任务是判断两点**a,b**之间的边是否被足够分割使得得到的线段足够逼近真实曲线。

  triTessEnough(a,b,c):判断整个三角形是否被足够分割。

- 最初输入一个三角形，调用函数**curveTessEnough(a,b)**，判断三角形的每条边是否被足够分割。如果都足够分割，则调用**triTessEnough(a,b,c)**，判断整个三角形是否足够分割。如果足够分割，那么退出过程。

- 如果有一条边或者多条边或者三角形内部没有被足够分割，则需要更进一步的分割。这时会出现四种不同的情况，如图**12.26**所示。

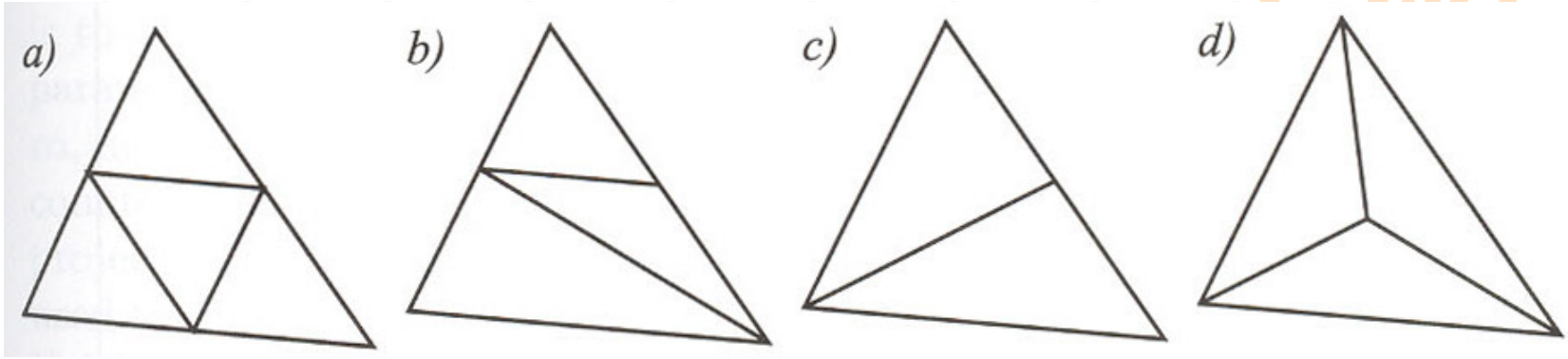


图12.26　进一步划分的四种情况。a)三角形的三条边都没有足够分割，所以整个三角形被分成四个新的三角形。b)只有两条边需要被分割。c)只有一条边需要被分割。d)所有的边都足够划分，但是三角形内部需要进一步分割。

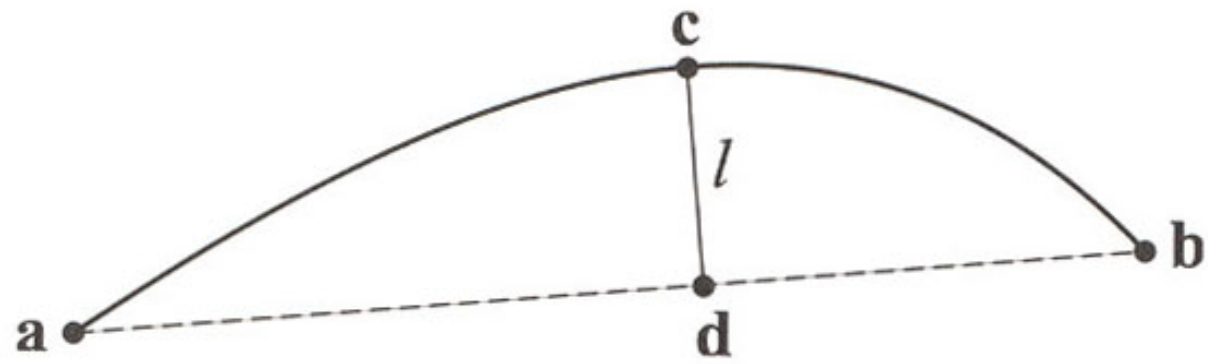


图12.27　a,b为曲面上的点。c为参数域上a,b参数值的平均值对应的曲面上的点，d为c在ab上的投影。*l*为c与d的距离。

- 因为只有边自身决定是否要进一步分割，所以每条边都以同样的方式进行分割，避免了裂缝。

- 对于三角形内部的测试，由于它不在边上生成新的点，不会出现裂缝。

- 如果过程**curveTessEnough**和**triTessEnough**构造不好的话，可能出现死循环(例如，两个过程一直返回**False)**。为了避免出现这种情况，使用一个退出判定条件值d。

$$d = \frac{\| (p-q) \times (p-r) \|}{\| p-r \|^2 + \| p-q \|^2 + \| q-r \|^2} \quad (12.49)$$

*d*值的说明

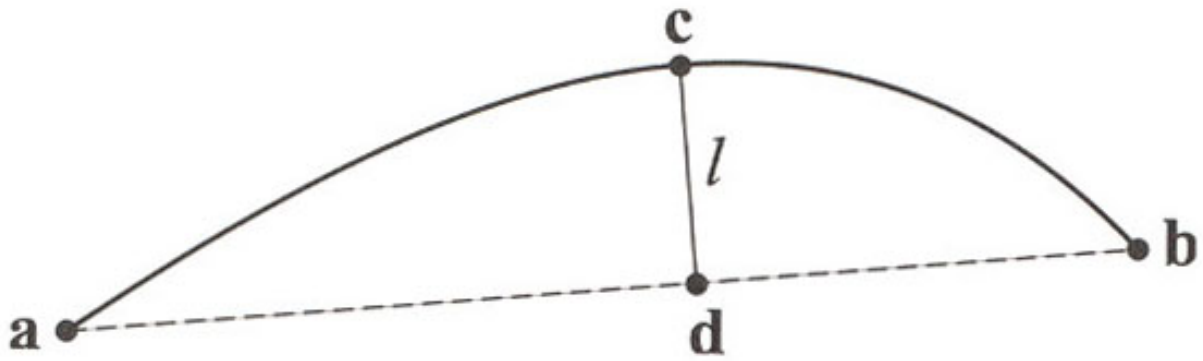
- 在公式(12.49)中，p,q和r都是三角形的二维*uv*坐标。这个公式计算的是在参数空间中三角形的面积的两倍与边的平方和的比率。

- 如果*d*值接近于零，三角形就趋近于退化，就可能出现无限循环。算法可以设置一个阈值，当*d*小于该值时，多边形化过程以如下方式终止：首先，三角形的每条边被分割直到它的所有部分都被分割得足够逼近原始形状。然后在三角形的中心生成一个新的点，然后由这个中点与三角形各边上的点生成三角形扇形片。

- **Chung与Field建议阈值的选取一般最好在0.005和0.05之间，当然也要视实际情况需要。**

curveTessEnough与triTessEnough的构造

- **curveTessEnough(p,q)**一个比较好的实现：如图12.27所示，使用参数**p,q**计算三维空间中的点**a,b**。得到参数空间中**p,q**之间的中点**m**，并且计算其对应的三维空间的点**c**。最后，计算点*c*与它在线段**ab**上的投影**d**之间的长度*l*。通过这个长度*l*可以判断该边上的曲线段是否足够扁平。如果*l*足够小，则认为是扁平的。为了保证算法能够终止，一般都会设置一个细分的上限值。当达到这个上限，分割终止。



影响Tessellation的其它因素

除了曲面的形状外，还有许多因素：

- 一个点的局部邻域是否在视域四棱锥内；
- 一个点的局部邻域是否朝向正面；
- 一个点的局部邻域是否在屏幕空间占据一大块区域；
- 一个点的局部邻域是否靠近物体的轮廓线；
- 一个点的局部邻域是否受到大量的镜面光的照射。

- 对于位于视域四棱锥外部边的剔除。使用一个球包围一条边。然后测试这个球与视域四棱台的关系。如果球在外面，那么**curveTessEnough**返回**true**，我们不再分割这条边。
- 背面剔除。在**a,b,c**点处的法向可以通过曲面的描述计算得到。这些法向可以确定了三个平面，如果这三个平面都是背向的，那么就不需要再对边进行分割了。
- 对屏幕空间的覆盖。投影采样物体到屏幕上，并且估计长度和面积。如果面积很大或者长度很大意味着要进一步分割。
- 提高物体轮廓处的多边形化率对于提高物体的绘制质量非常重要。通过测试**a**点的法向与视点到**a**的向量的点积是否接近于零可以找到轮廓边处的三角形。如果成立的话，需进一步进行分割。
- 由于曲面上镜面高光改变快速，经常会出现绘制走形现象。如果测试到沿着某点处法向的反射光线同该点的视线向量的点积接近于**1**，那么在这一点处会出现很强的高光，需要继续分割。

线段ab在屏幕空间中投影的快速估计

- 首先，将线段进行变换，使得它的中点在视线方向上。

- 然后，假设线段与投影平面*n*平行，从而可以计算出在屏幕空间中的投影*s*。如图**12.28**所示。

  使用线段**a'b'**代替**ab**，则屏幕空间中的投影**s**为

$$s = \frac{n\sqrt{(\mathbf{a}'-\mathbf{b}')\bullet(\mathbf{a}'-\mathbf{b}')}}{\mathbf{v}\bullet(\mathbf{a}'-\mathbf{e})} \qquad (12.50)$$

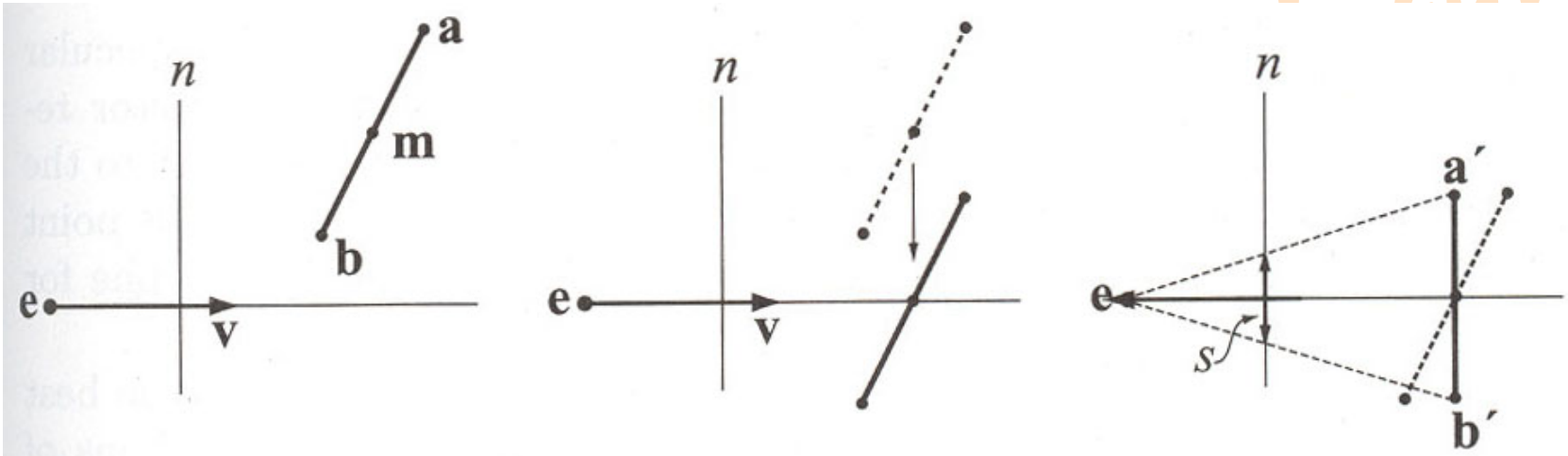


图12.28 线段的屏幕空间投影估计s

公式12.50中，分子是投影平面距离n乘以线段的长度，分母是线段中点到视点e的距离。将计算出的投影s同阈值t(表示屏幕空间边的最大长度)比较。为了避免计算平方根，可以重写式子12.50。如果下面的条件成立，则要继续分割：

$$s > t \Leftrightarrow (\mathbf{a}'-\mathbf{b}')\cdot(\mathbf{a}'-\mathbf{b}') > \frac{t^2}{n^2}(\mathbf{v}\cdot(\mathbf{a}'-\mathbf{e})) \quad (12.51)$$

自适应细分评价

- 优点：简单，不需要保留复杂的数据结构，也不需要保存三角形。
- 缺点：对于每条边，过程**curveTessEnough**被调用两次，也就是说邻域之间没有共享计算结果。一个解决办法是把完成的测试信息保存在一个哈希表中，只要在哈希表中发现一个元素，这个元素就可以被删除。
- 一般来讲，很难说某种方法能够适用于所有的情况。最好是能够启发式地测试多种方法，并且将它们混合起来。

四叉树算法

- 假设使用一个矩形片。传入整个参数域(从(0,0)到(1,1)的正方形)开始递归算法。使用前面描述的细分准则，测试曲面是否是足够分割的。如果是，则算法终止。否则，将参数域分成四个不完全相同的四边形，并且分别在这四个子区域上递归调用算法。继续递归调用算法直到曲面被足够分割，或者达到了预先设定的递归层数。这个算法的性质使得在多边形化过程中递归生成了一棵四叉树。

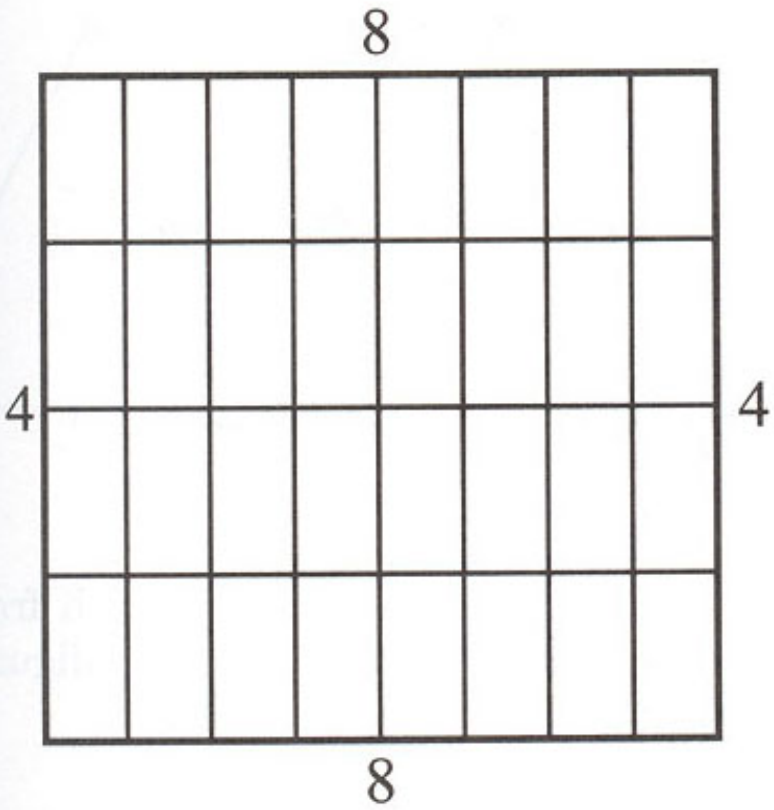
- 注意：如果相邻的子区域分割层数不同，就会出现裂缝。解决的办法是使相邻子区域的分割层数最多相差1。然后使用图12.25所示的方法来填充裂缝。

- 这个方法最大的缺点是需要记录的相关的信息太多。

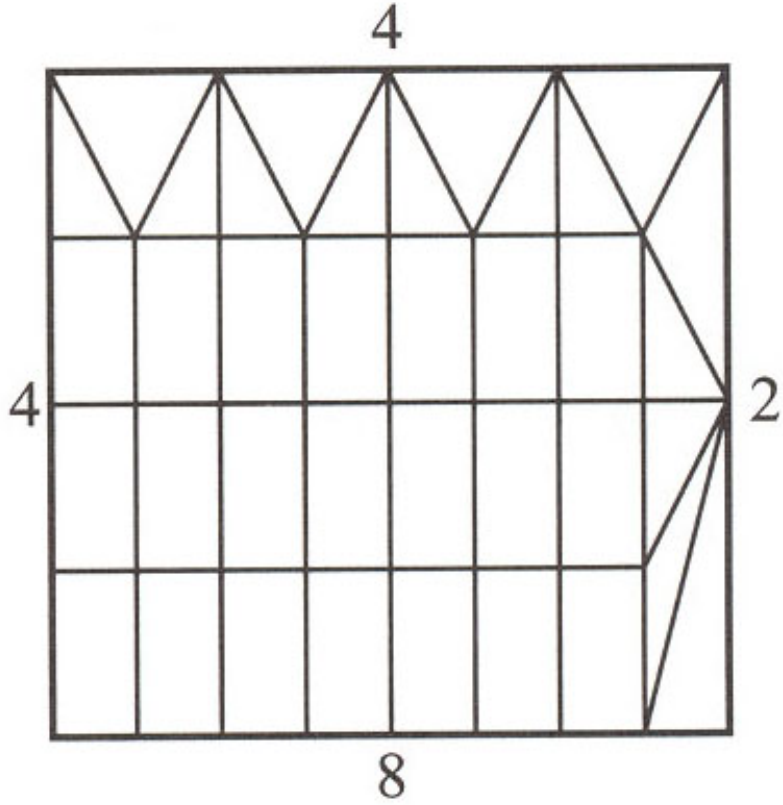
非整型多边形化
(Fractional Tessellation)

- 为了得到更加光滑的参数曲面，**Moreton**引入了非整型多边形化因子。通过在参数曲面上不同的边界上使用不同的多边形化因子就成了带条件约束的自适应多边形化方法。
- 下面我们给出使用非整型多边形化因子进行多边形化能够达到的效果。

- 一条边的多边形化因子是在这条边上生成的点的个数减1.

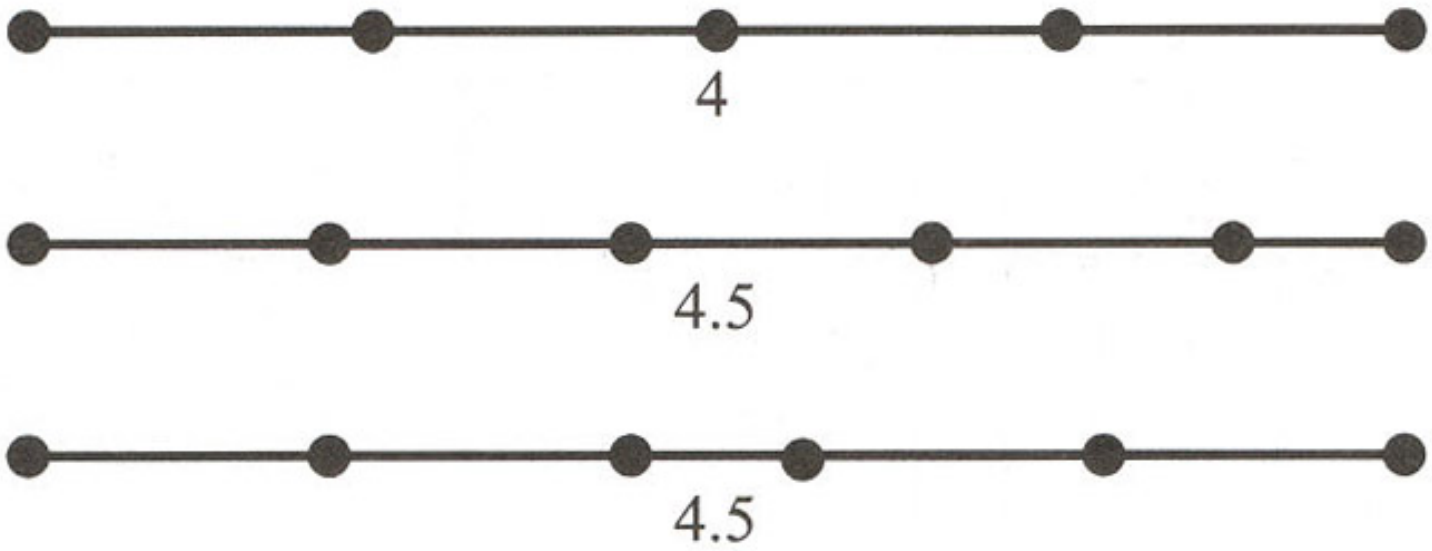


规整多边形化: 每一行使用一个因子，每一列使用一个因子。



四条边界使用独立的多边形化因子。

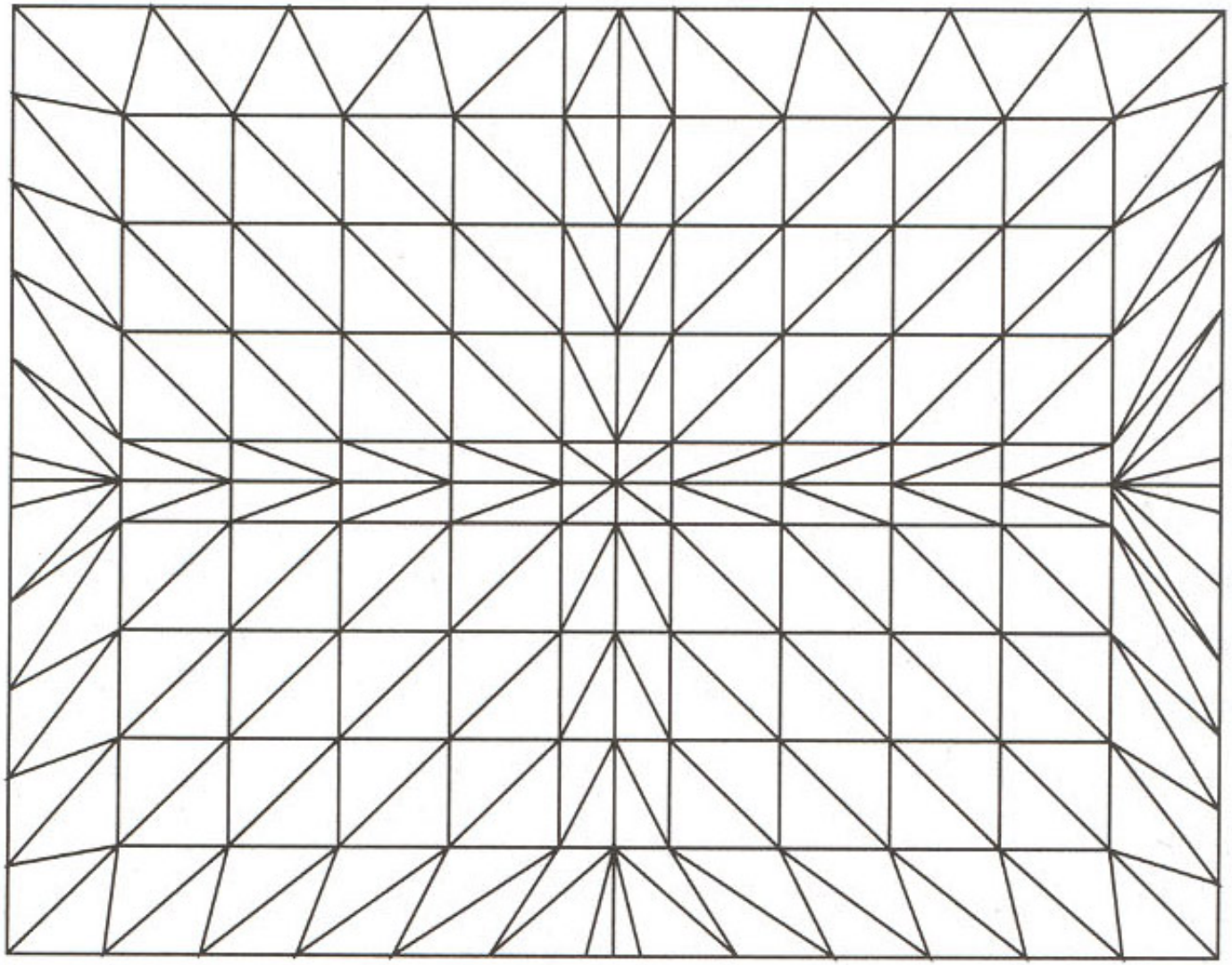
- 一条边的非整型多边形化因子的概念如下图。对于一个整型因子n，n＋1个点在k/n处生成(k=0,1,…,n)。对于一个非整型的多边形化因子r，$\lceil r \rceil$个点在k/r处生成(k=0,…,$\lfloor r \rfloor$).



上图：整型多边形化。

中图：非整型多边形化，右边带有小数部分。

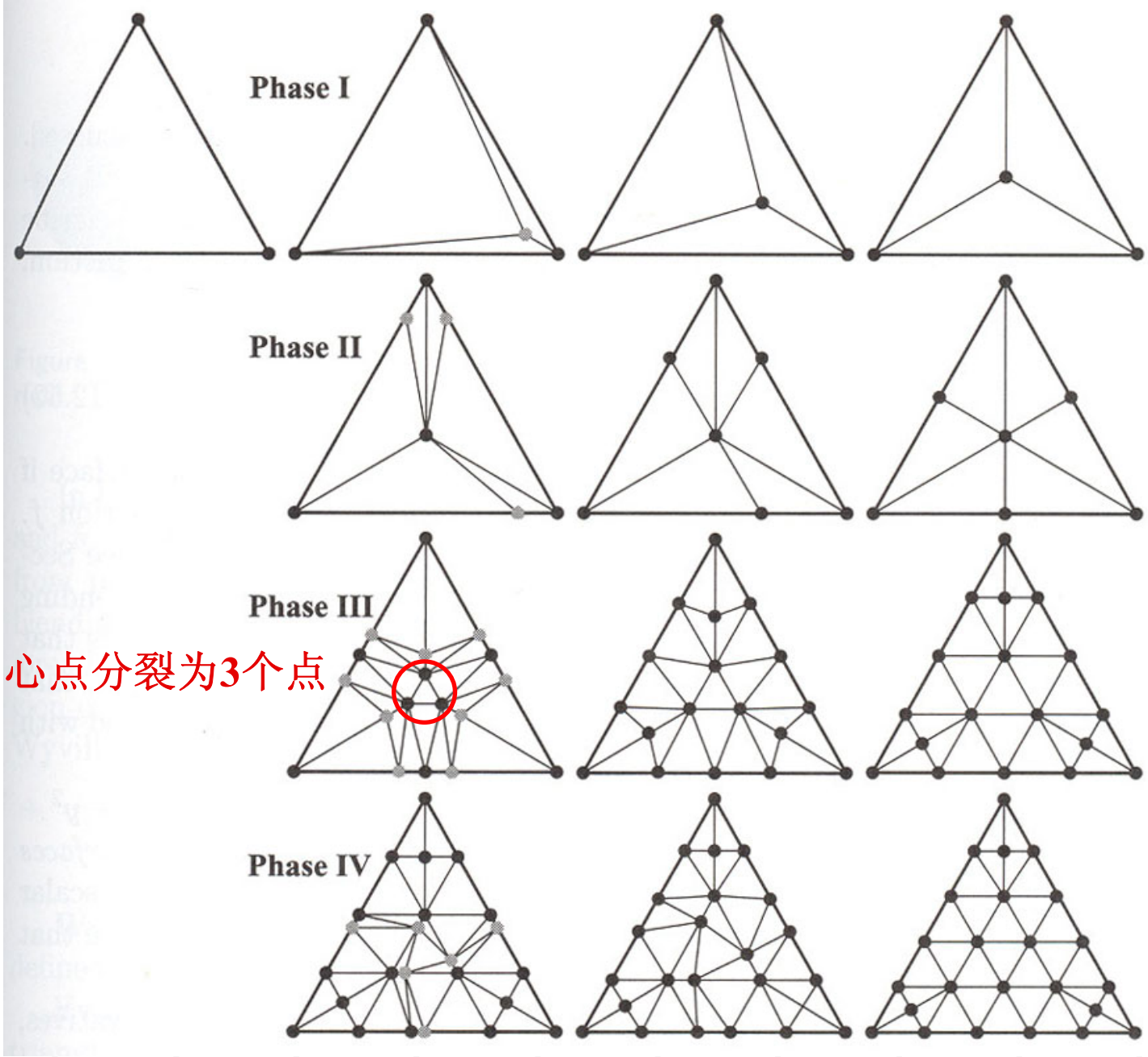
下图：中间带有小数部分的非整型多边形化。这种设置避免了相邻面片之间出现裂缝。



在一个矩形域上对一个面片进行非整型多边形化

面向三角片的非整型多边形化

- 在初始三角形的任意一个顶点处生成一个新的点，然后将它移动到三角形的中心（见下页图）。
- 在三角形的每条边上各生成一个新的点。然后沿着三角形的边移动，最后停留在边的中点位置。我们称这些点为边中点点。
- 对于每个边中点点，生成两个新的点，同时三角形的中心点转化成如图**12.32**所示的三个点。然后移动新生成的点。最后我们可以得到均匀的内点。
- 每个边中点点分裂成两个，而且原先三角形中心分裂的三个点分别分裂成两个。然后进行移动，最终可以得到均匀的内点。而且这个阶段可以重复使用，最后依然能保初始的结构。



中心点分裂为3个点

图12.32 三角形的非整型多边形化。每个灰色的点是新生成的点。

隐式曲面
(Implicit Surfaces)

- 与参数曲面使用参数(***u,v***)显式描述曲面上的点不同，隐式曲面使用下面的被称作为隐式函数的形式来描述曲面:

$$f(x,y,z) = f(\mathrm{p}) = 0 \qquad (12.52)$$

上式的意思是将点**p**代入函数***f***，如果结果为零，点**p**在隐式曲面上。

- 优点
  - 与光线求交测试非常简单。
  - 可以很容易应用**CSG**算法，而且易于混合和变形。

- **等值面(Isosurfaces)**

一个隐式函数的等值面可以表示为**f(x,y,z)=c**，这里**c**是一个标量函数。对于单位球函数，**f(x,y,z)=2**描述了一个以球心为中心，半径为**2**的等值面。

- **曲面法向**

隐式曲面的法向可以使用偏导数描述如下：

$$\nabla f(x,y,z)=\left(\frac{\partial f}{\partial x},\frac{\partial f}{\partial y},\frac{\partial f}{\partial z}\right) \qquad (12.53)$$

- **隐式曲面混合(Blending)**
  - **Blobby模型，软物体(soft objects)，元球模型(meta-ball)。**
  - 基本思想：使用几个非常简单的图元，如球或椭球，然后光滑地混合它们。数学上可以表示如下：

$$f(p)=\sum_{i=0}^{n-1}h(r_i) \qquad (12.54)$$

其中核函数**h(r)**可以有多种选择，如高斯函数，分段多项式函数。

$$h(r)=(1-\frac{r^2}{R^2})^3,\ h(r)=0, r\geq R \qquad (12.55)$$

隐式曲面的绘制

➢ 隐式曲面多边形化

隐式曲面多边形化有很多算法，最著名的是**Marching Cubes**算法。这个算法是将一个三维网格放置在整个空间中，然后在每个网格点上对隐式函数进行采样。每个点要么在隐式曲面内部要不么在外部。因为一个**Cube**有**8**个网格点，这样有**256**中不同的组合。每种组合在**Cube**内部生成**0**－**4**个三角形来表示隐式曲面。

***Wyvill*与*Bloomenthal*的多边形化的源码可以在他们的主页上得到。**

➢ 光线跟踪

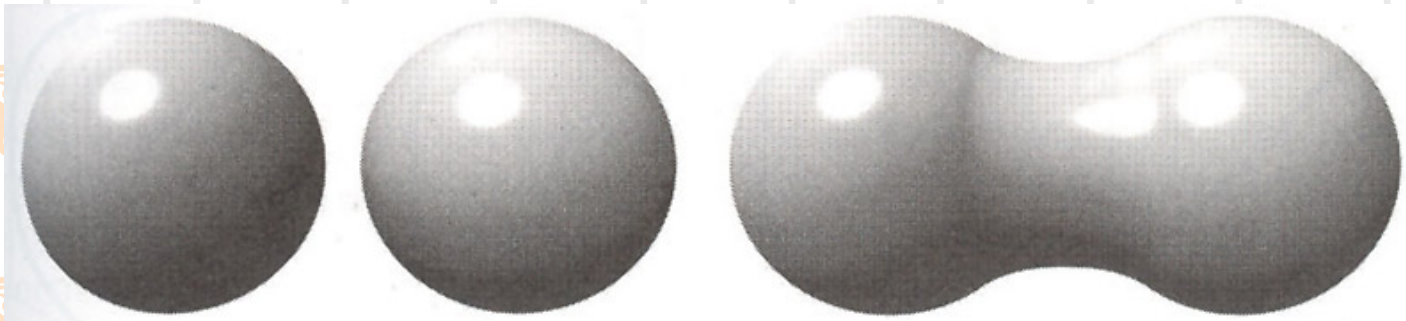


图**12.33** 左边两个球通过混合成右边的形状

细分曲线
(Subdivision Curves)

- 细分技术是一种生成曲线曲面的相对较新的方法。因为它在离散曲面(三角形网格)和连续曲面(如Bezier曲面片)之间架起了一座桥梁。

- 我们先简要介绍一下曲线细分方法，然后再讲曲面细分。

- 曲线细分方法分类

细分过程可以使用不同的方式来进行，因而产生了不同类的细分方法。

  - 逼近型细分方法：生成的极限细分曲线不经过顶点，而且每一次细分后，前一次细分的顶点被舍弃或者被更新。
  - 插值型细分方法：生成的极限细分曲线经过初始的顶点，它保留每一次细分后的所有顶点。

- 曲线细分方法

  - 割角法:是一个很有代表性的方法，可以很好解释细分方法的思想。如图**12.34**所示。它是**Chaikin**提出的方法。它是一个逼近型细分方法，按照以下方式进行细分的：

$$\mathrm{p}_{2i}^{k+1}=\frac{3}{4}\mathrm{p}_{i}^{k}+\frac{1}{4}\mathrm{p}_{i+1}^{k}$$

$$\mathrm{p}_{2i+1}^{k+1}=\frac{1}{4}\mathrm{p}_{i}^{k}+\frac{1}{4}\mathrm{p}_{i+1}^{k} \qquad (12.56)$$

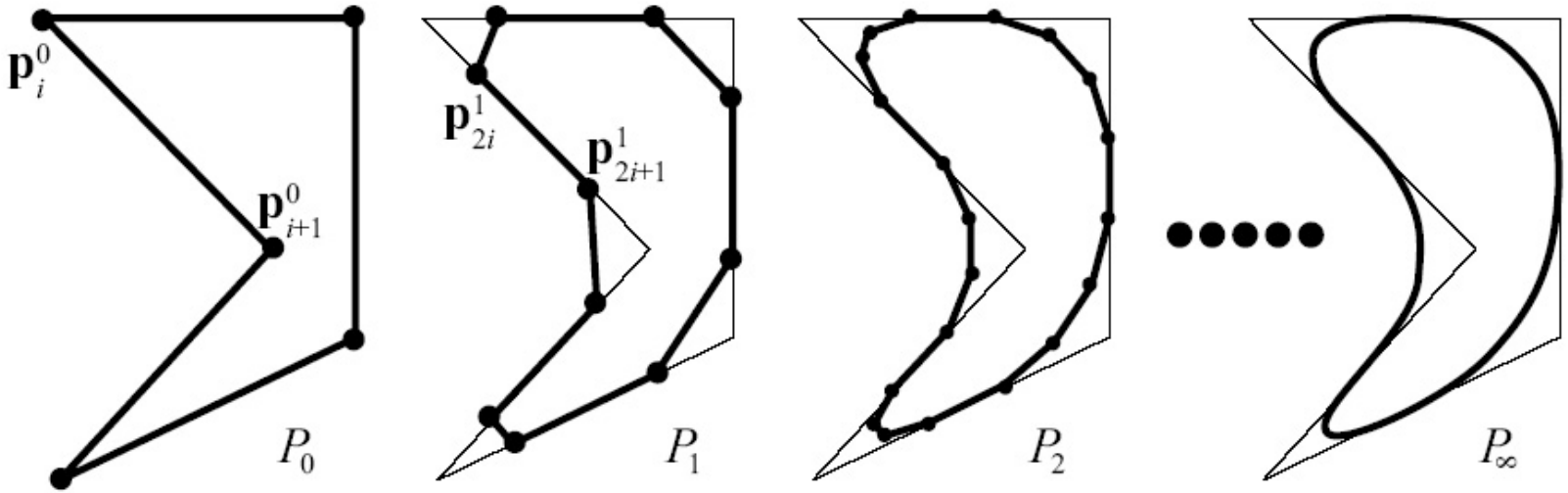


图12.34 Chaikin细分方法。生成的曲线不经过初始顶点。

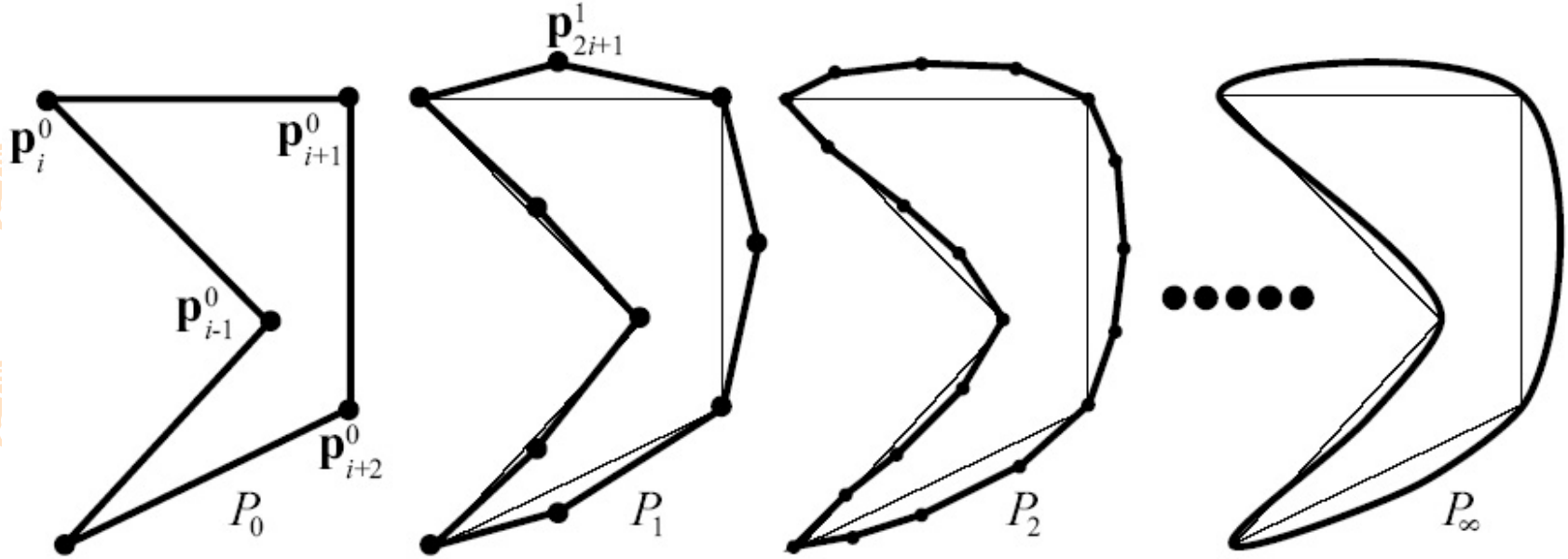


图12.35 4-点细分方法，插值细分方法。经过初始顶点。

- **4-点细分方法**

这个方法使用四个最近的点生成一个新的点。它是一个插值型细分方法，按照一下方式进行细分：

$$\begin{aligned} p_{2i}^{k+1} &= p_i^k \\ p_{2i+1}^{k+1} &= (\frac{1}{2}+\omega)(p_i^k + p_{i+1}^k) - \omega(p_{i-1}^k + p_{i+2}^k) \end{aligned} \qquad (12.57)$$

由上式可以看出，这个方法保留前一次细分生成的点而不改变它。w是一个张量参数。**w＝0**，得到线性插值，**w＝1/16**，得到图**12.35**所示的曲线。

- 另外一个逼近型细分方法

它按照下面的规则进行细分：

$$\begin{aligned} p_{2i}^{k+1} &= \frac{3}{4}p_i^k + \frac{1}{8}(p_{i-1}^k + p_{i+1}^k) \\ p_{2i+1}^{k+1} &= \frac{1}{2}(p_i^k + p_{i+1}^k) \end{aligned} \qquad (12.58)$$

第一个式子更新已经存在的点，第二个式子计算两个相邻点线段的中点。这个方法生成一条三次**B**样条曲线。

- 对于开曲线，在端点处出现一个问题，因为每个新生成的点两边各需要两个点，而现在只有一个。解决办法是通过端点反射与端点相邻的一个点，这样生成了一个新的控制点。如下图所示：

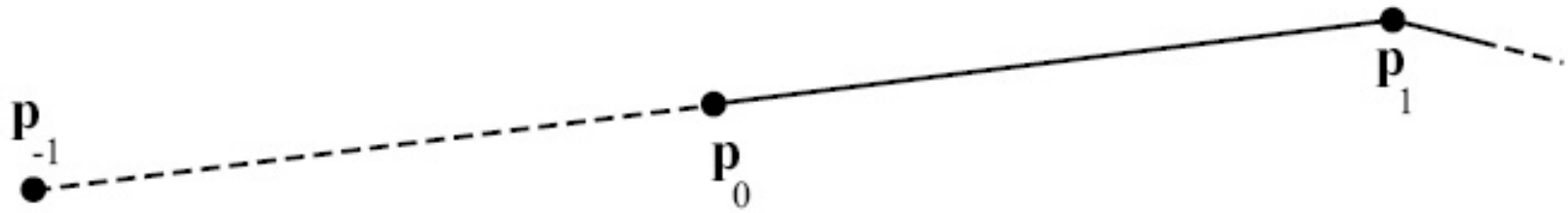


图12.36 对于开曲线，反射点p_{-1}的生成。反射点可以这样计算：$p_{-1}=p_0-(p_1-p_0)=2p_0-p_1$。

细分曲面
(Subdivision Surfaces)

- 优点
  - 细分曲面是一个功能强大的曲面定义方法，它能在任意拓扑的网格上定义光滑连续没有裂缝的曲面。
  - 细分曲面可以得到曲面的无限的层次细节模型。也就是可以生成任意数量的三角形或多边形。而原始曲面的表示是简洁的。
  - 细分规则简单而且容易实现。
- 缺点

曲面连续性的分析涉及到非常多的数学知识。

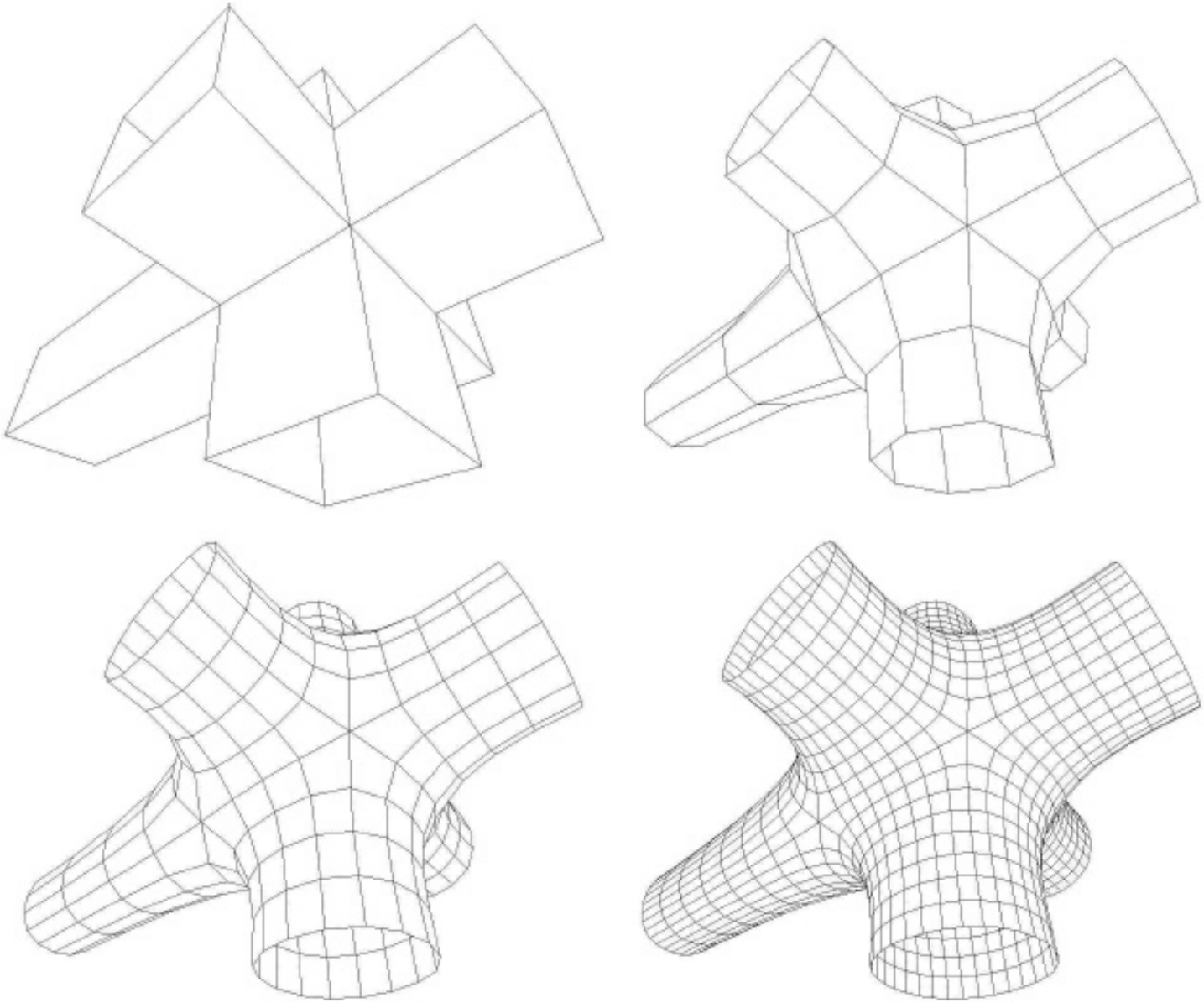


图12.37 上左图是控制网格，后面依次是细分1，2，3次的曲面。

- 曲面细分包括两个阶段，这两个阶段的实现细节就刻画了一个细分方法。
  - **Refinement phase**

    在这个阶段，由初始的控制网格生成新的顶点，并且将它们连接起来得到新的更小的三角形。多边形可以以不同的方式分裂。
  - **Smoothing phase**

    在这个阶段，重新计算一些或者全部顶点的位置。可以选择不同的细分规则，从而得到不同的特点，如连续阶、曲面是逼近还是插值。

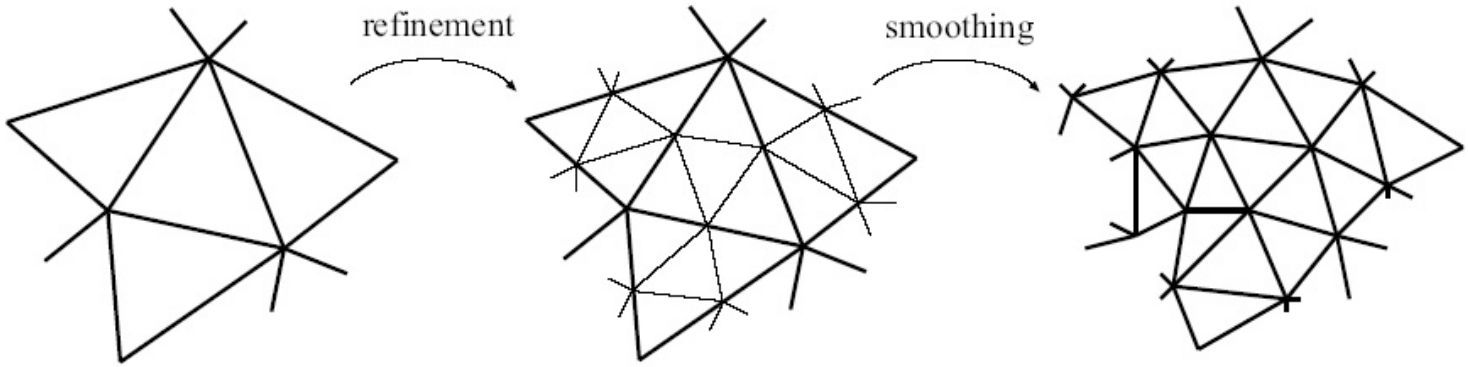


图12.38 Refinement 和 Smoothing

- 细分方法使用不同的分类标准可以分成不同的类
  - 细分规则是否固定(**Stationary**)

    一个固定的细分方法在每一次细分过程中使用同样的细分规则。反之则为非固定的细分方法。
  - 是否是均匀细分

    一个均匀细分方法对每个顶点或每条边都使用相同的细分规则。反之则为非均匀的细分方法。
  - 是基于三角形还是基于多边形

    一个基于三角形的细分方法仅仅在三角形上进行操作，这样也仅仅生成三角形
- 在后面我们将介绍几种不同的细分方法，接着给出细分曲面方法的两个扩展使用，最后介绍一些实用的细分和绘制算法。

Loop细分

- Loop细分方法是第一个三角形的细分方法。它与12.5节最后一种细分方法相似，是一种逼近方法。它更新每一个存在的点并且在每一条边上生成一个新的点。其连接关系如图12.39所示。每一个三角形被一分为四，在n次细分后，一个三角形被细分成4^n个三角形。

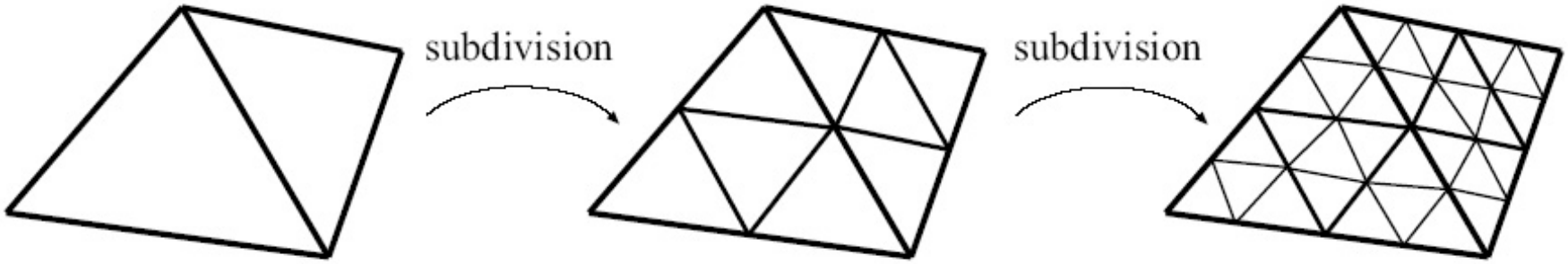


图12.39 Loop细分（或者改进的Butterfly细分方法）的两次细分步骤的顶点的连接关系。每个三角形生成四个新的三角形。

- 假设顶点为$\mathbf{p}^k$，其中k为细分次数。这意味着$\mathbf{p}^0$，是控制网格的顶点。一次细分之后，$\mathbf{p}^0$变为$\mathbf{p}^1$。一次次细分之后，$\mathbf{p}^0 \rightarrow \mathbf{p}^1 \rightarrow \mathbf{p}^2 \rightarrow \ldots \rightarrow \mathbf{p}^\infty$，$\mathbf{p}^\infty$是极限点。
- 如果点$\mathbf{p}^k$的度数为n，那么它有n个邻点，如图12.40所示。而且当$n=6$时，点$\mathbf{p}^k$称作规则点或者平凡点，否则称作不规则点或者非凡点。

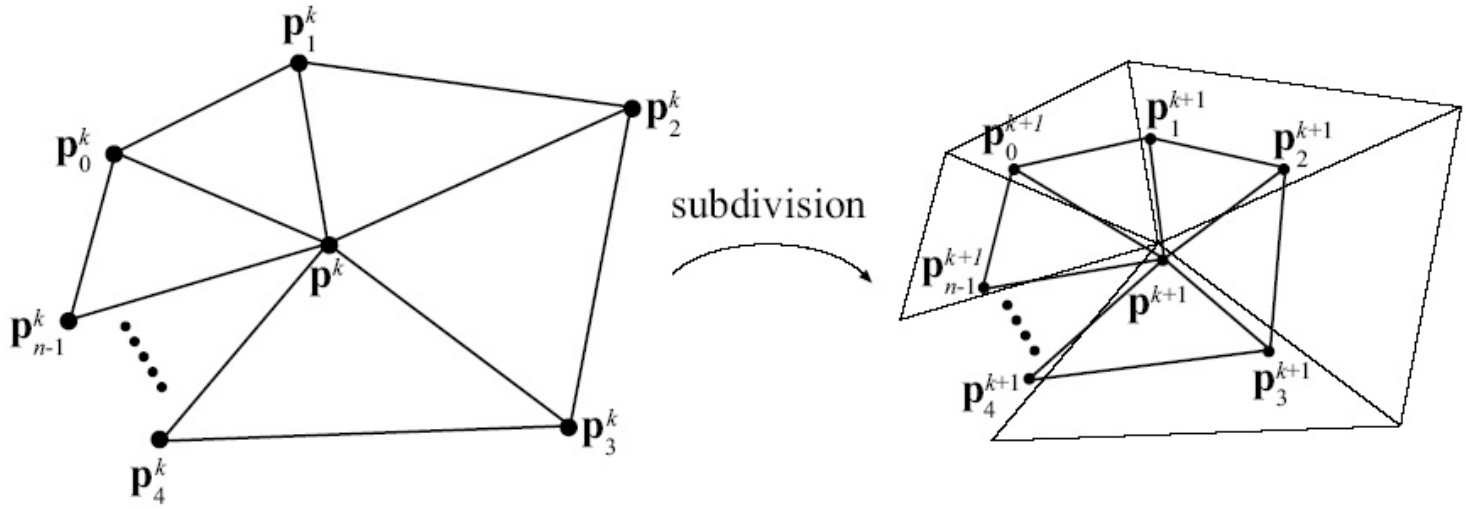


图12.40 Loop细分方法。左边的邻域细分成右边的邻域。中心点$\mathbf{p}^k$被更新而且被$\mathbf{p}^{k+1}$取代，对$\mathbf{p}^k$与$\mathbf{p}_i^k$之间的每条边，生成一个新的点($\mathbf{p}_i^{k+1}$，$i=1,\ldots,n$)

- **Loop细分规则**

$$\mathrm{p}^{k+1} = (1-n\beta)\mathrm{p}^{k} + \beta(\mathrm{p}_0^k + \cdots + \mathrm{p}_{n-1}^k),$$
$$\mathrm{p}_i^{k+1} = \frac{3\mathrm{p}^k + 3\mathrm{p}_i^k + \mathrm{p}_{i-1}^k + \mathrm{p}_{i+1}^k}{8}, i = 0 \ldots n-1. \qquad (12.59)$$

第一个公式是更新已经存在的点的规则，第二个公式是生成新点的公式。***n***为$\mathbf{p}^k$的度数。从上面两个式子可以看到，所有权系数的和都为**1**。常数β是***n***的函数：

$$\beta(n) = \frac{1}{n}\left(\frac{5}{8} - \frac{(3+2\cos(2\pi/n))^2}{64}\right) \qquad (12.60)$$

下式是**12.60**的一个变体，它避免了使用三角函数**(Warren,Weimer)**

$$\beta(n) = \frac{3}{n(n+2)} \qquad (12.61)$$

- **Loop细分规则可以形成一个细分模板如下图所示：**

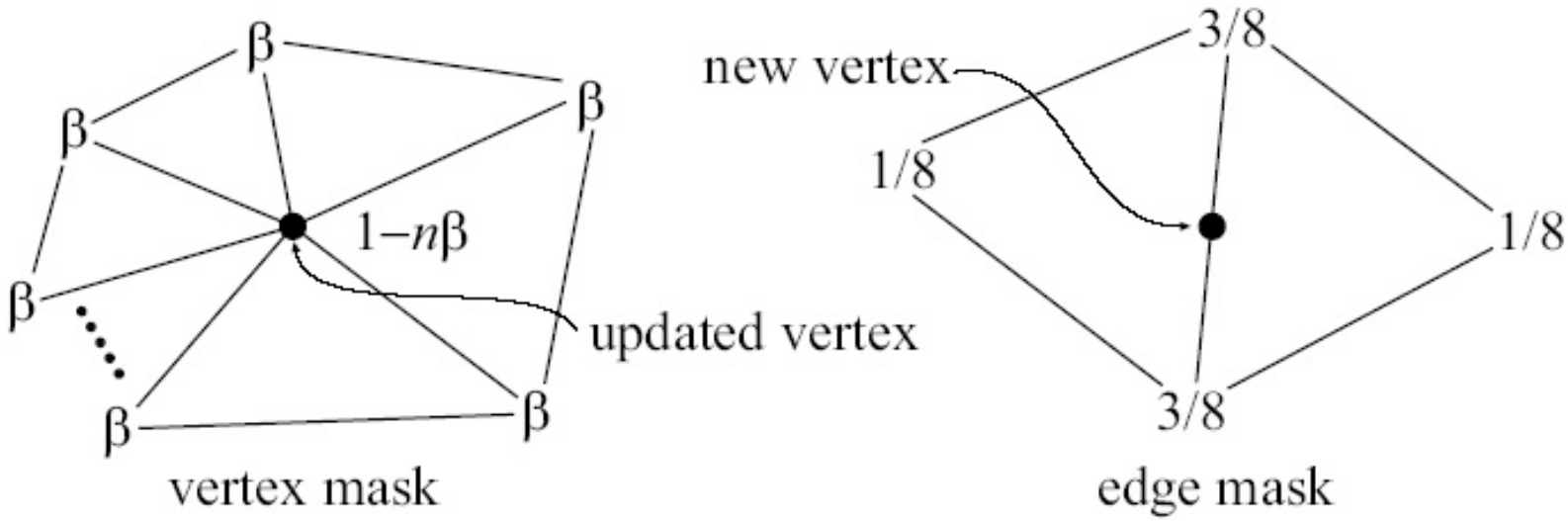


图12.41 Loop细分模板(黑箭头所指的点是被更新或者新生成的点)。一个模板指出了每个相关顶点的权系数。例如：当更新一个已经存在的点，权$1-n\beta$是已存在的点的系数，而权β所有邻点的系数。

- Loop细分方法可以生成一张在规则点处C^2连续，在不规则点处C^1连续的曲面。下图是Loop细分曲面的例子。

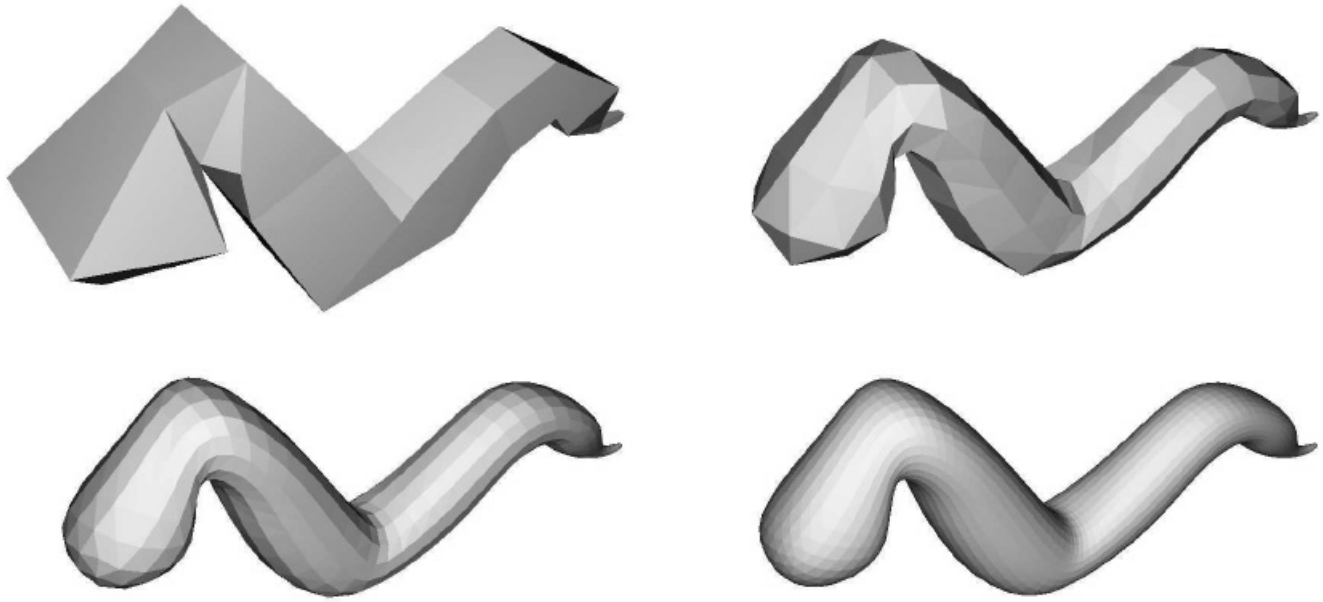


图12.42 使用Loop细分方法细分三次后得到一条虫子的曲面

注意，对于不封闭的网格，我们不能使用前面提供的细分规则，在边界部分我们必须使用特殊的规则。对于Loop细分方法，可以使用公式12.58的反射规则。

- 在无限次细分之后得到的曲面称为极限曲面。极限曲面上的点与切向可以使用解析表达表达式来计算。一个顶点的极限位置可以使用公式**12.59**的第一个公式来计算。

- 通过使用下式来替代 β **(n)**：

$$\gamma(n) = \frac{1}{n + \frac{3}{8\beta(n)}} \qquad (12.62)$$

点$\mathbf{p}^k$的两个极限切向可以通过加权其邻点(也称为**1-**环，或者**1-**邻域)来计算：

$$t_u = \sum_{i=0}^{n-1} \cos(2\pi i / n) p_i^k, \quad t_v = \sum_{i=0}^{n-1} \sin(2\pi i / n) p_i^k \qquad (12.63)$$

- 法向可以计算如下，这样计算给出了点$\mathbf{p}^k$的确切法向：

$$n = t_u \times t_v$$

- 使用逼近的细分方法的优点：
  - 光顺性好。
  - 逼近方法比插值方法收敛速度快。不过这是以牺牲物体的形状为代价的。
  - 极限曲面包含在在原始控制顶点的凸包内。

- 缺点：
  - 使用逼近方法生成的曲面形状就如同使用了低频滤波器过滤得到的形状，也就是说形状通常是过度收缩的。对于少量的网格，影响更加明显。如图12.43所示。
  - 解决办法：减少这种影响的一种方法是在控制网格上使用更多的点。Maillot和Stam提出了一种混合细分方法可以控制收缩的程度。

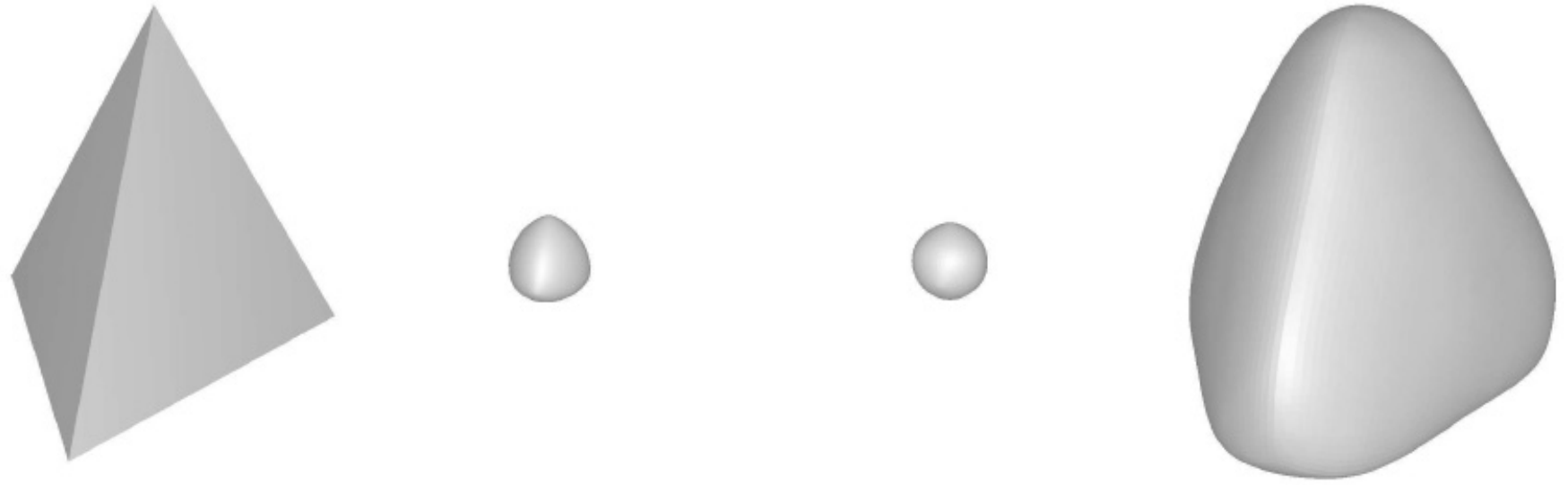


图**12.43** 一个四面体被**Loop**，$\sqrt{3}$ 和改进的**Butterfly(MB)**细分方法细分五次后得到的曲面。**Loop**和$\sqrt{3}$方法都是逼近的方法，而**MB**是插值方法。逼近方法就像是一个低频滤波器，这意味着会发生收缩。

注：***Loop**细分方法可以生成一般的三维二次箱样条。所以，对于一个完全由规则点构成的网格，我们最终可以将曲面描述为一种样条曲面。但是，对于非规则的点是不能用样条曲面描述的。*

改进的Butterfly细分

- 主要参考文献:

1. Dyn Nira, David Levin, John Gregory, "A butterfly subdivision scheme for surface interpolation with tension control", *ACM Transactions on Graphics*, 1990, 9(2):160-169.

2. Zorin Denis, Peter Schroder and Wim Sweldens, "Interpolating subdivision for meshes with arbitrary topology", *Proc. Siggraph'1996*, pp.189-192.

- 因为改进的**Butterfly**细分方法在边界上使用不同的规则，而且在不同度数的顶点处也使用不同的细分规则，所以这个方法是非均匀的。

- 与Loop细分方法(都是三角形细分方法)最大的不同在于，它不是逼近的方法，而是插值方法。插值意味着一个点一旦存在于网格上，那它的位置就不会改变。所以这个方法不会改变已经存在的点的位置，只在边上生成新的点。连接关系与Loop细分方法相同。

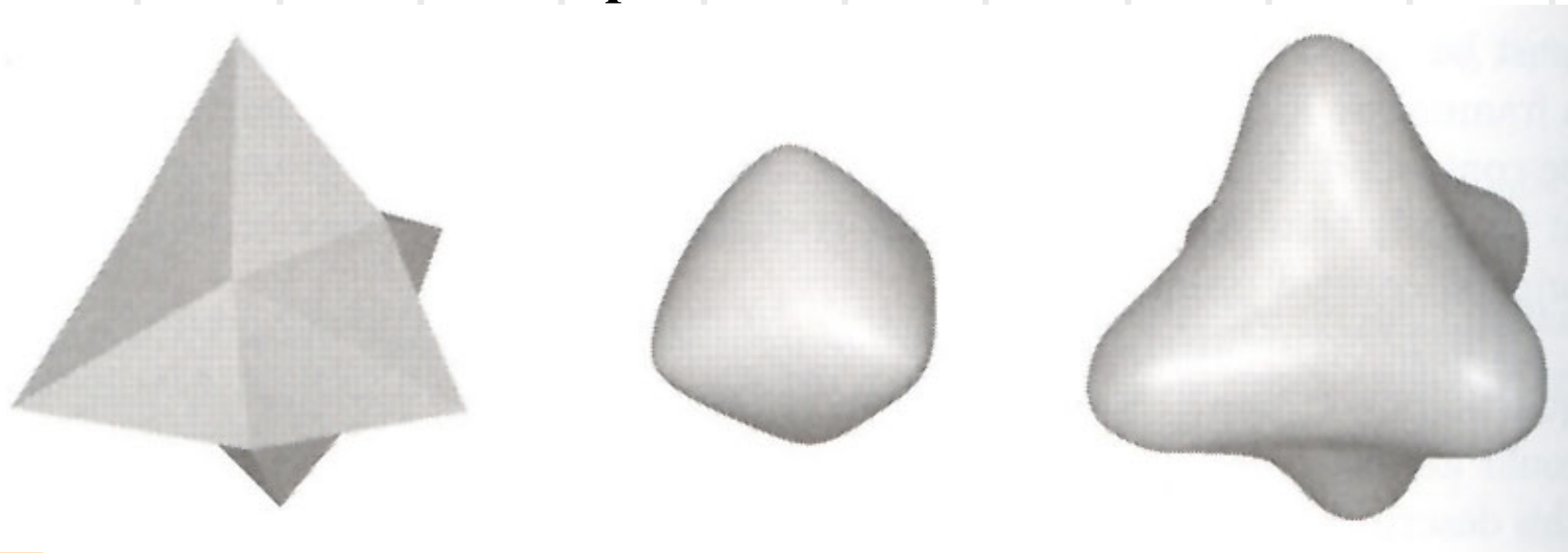


图12.44 左边的图是一个三维星形物体的网格。中间是使用Loop细分方法得到的曲面，它是逼近的。右边是使用改进的Butterfly方法得到的曲面，它是插值的。使用插值方法的一个优点是生成的曲面经常比逼近的方法得到的曲面更加接近控制网格。但是，对于精细网格，效果没有这么明显。

- **MB方法为在两个已存在的点之间生成新的点使用了四种不同的细分规则：**
  - **A、规则情况**

    在两个已存在点之间生成新的点，而且这两个点的度数为6。其模板如图12.45中左图所示。

  - **B、半规则情况**

    两个点中一个点的度数为6，另外一个点的度数不是6。这种情况下，新点的计算公式如下：

$$
\begin{aligned}
&n=3: w_0=5/12, w_1=-1/12, w_2=-1/12, \\
&n=4: w_0=3/8, w_1=0, w_2=-1/8, w_3=0, \qquad (12.64)\\
&n\geq 5: w_j=\frac{0.25+\cos(2\pi j/n)+0.5\cos(4\pi j/n)}{n}
\end{aligned}
$$

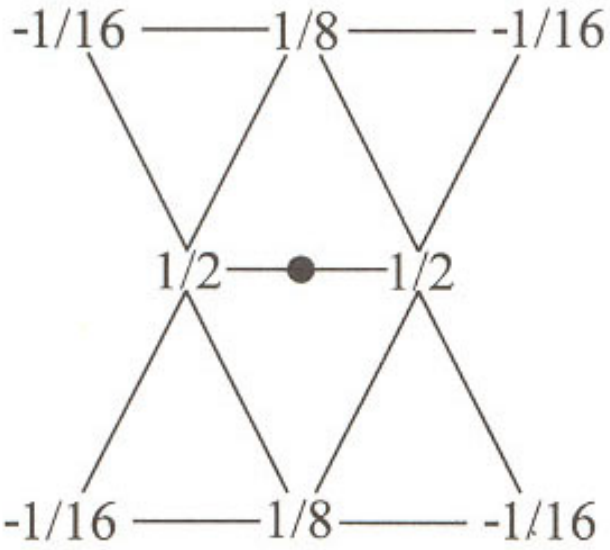
- **C、不规则情况**

  当一条边的两个端点的度数都不等于6，我们先使用情况B的公式计算这两个点的新点，然后将这两个点的平均作为新生成的点。这*仅仅在第一次细分时发生*，因为细分一次后，网格中只有A、B两种情况的点。因而这样选择的连续性不会影响到极限曲面。**Zorin et al.[840]**指出这种规则生成的形状具有更好的光顺性。

- **D、边界情况**

  在边界上，三角形网格上的一条边只有一个三角形同它相关联。这时可以使用**12.5**节描述的插值方法（计算反射点）**[197]**，这里$w=1/16$。那么图**12.45**中的权值的取值模板如下：

$$w_{-1}=-1/16, w_0=9/16, w_1=9/16, w_2=-1/16. \qquad (12.65)$$



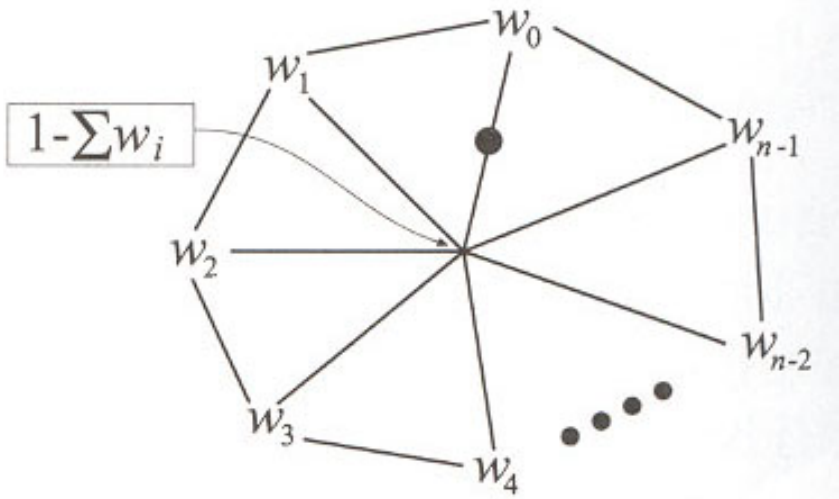


图**12.45** 左边的模板是**Butterfly**的模板，在两个规则点之间生成新的点时使用。右边的模板是在两个点中一个点是规则的一个点是不规则的情况下使用。注意黑色的点表示新生成的点。

- 极限点与极限切向量的计算
  - 因为是插值方法，顶点的极限位置就是顶点本身。
  - 极限切向量的计算比较复杂。对于不规则点($n \neq 6$)，切向可以使用方程12.63来计算。对于规则点($n=6$)，要使用到2-环(2-邻域)。规则点p的1-环与2-环如图12.46所示。

    切向量$\mathbf{t}_u$和$\mathbf{t}_v$可以计算如下(法向$\mathbf{n}=\mathbf{t}_u \times \mathbf{t}_v$):

    $$\mathbf{t}_u = \mathbf{u} \bullet \mathbf{r}, \quad \mathbf{t}_v = \mathbf{v} \bullet \mathbf{r}. \qquad (12.66)$$

    这里r是整个2-环的差分向量$\mathbf{p}_i$-p中的一个向量，u和v是由一些标量构成的向量。

    $$\mathbf{r} = (\mathbf{p}_0 - \mathbf{p}, \mathbf{p}_1 - \mathbf{p}, \mathbf{p}_2 - \mathbf{p}, \cdots, \mathbf{p}_{16} - \mathbf{p}, \mathbf{p}_{17} - \mathbf{p}),$$

    $$\mathbf{u} = (16, -8, -8, 16, -8, -8, -\frac{8}{\sqrt{3}}, \frac{4}{\sqrt{3}}, \frac{4}{\sqrt{3}}, -\frac{8}{\sqrt{3}}, \frac{4}{\sqrt{3}}, \frac{4}{\sqrt{3}},$$

    $$1, -\frac{1}{2}, -\frac{1}{2}, 1, -\frac{1}{2}, -\frac{1}{2}), \qquad (12.67)$$

    $$\mathbf{v} = (0, 8, -8, 0, 8, -8, 0, -\frac{4}{\sqrt{3}}, \frac{4}{\sqrt{3}}, 0, -\frac{4}{\sqrt{3}}, \frac{4}{\sqrt{3}},$$

    $$0, \frac{1}{2}, -\frac{1}{2}, 0, \frac{1}{2}, -\frac{1}{2}).$$

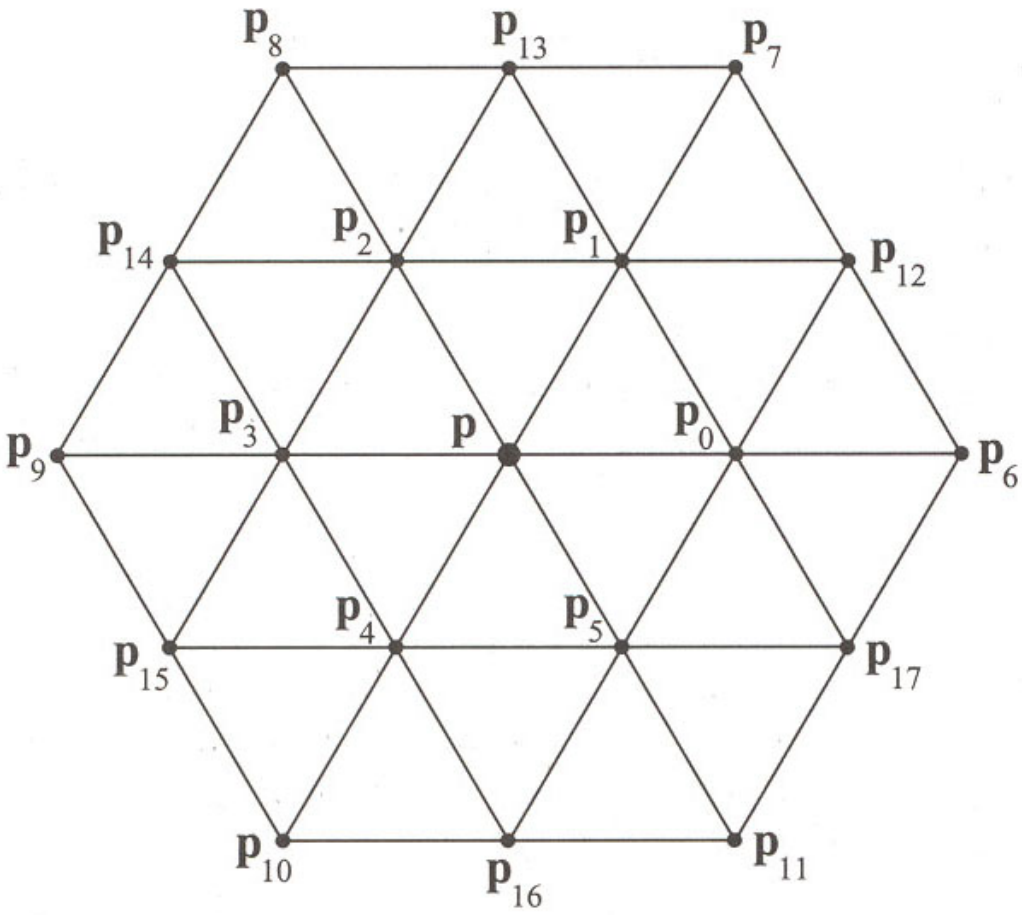


图12.46 规则点p的1-环(p_0,…,p_5)与2-环(p_6,…p_{17})

改进的Butterfly细分的评价

如果需要插值，MB方法是一个好的选择。

- 缺点：

  A、由于不自然的波动，可能生成奇怪的形状，不光顺，如图12.47。

  B、模板复杂，计算量大。

- 优点：

  A、能生成非常接近控制网格形状的曲面。

  B、适合于实时绘制。实时绘制中的网格一般不是非常精细的网格，因而一张插值曲面更加合适（若用逼近型，则会收缩），它更加接近控制网格的位置。尽管产生不光顺的问题，但是在大多数情况下，只要对网格进行微调就可以去掉那些褶皱。

  C、使用MB方法生成的整张曲面是C^1连续的，即使在不规则点处。

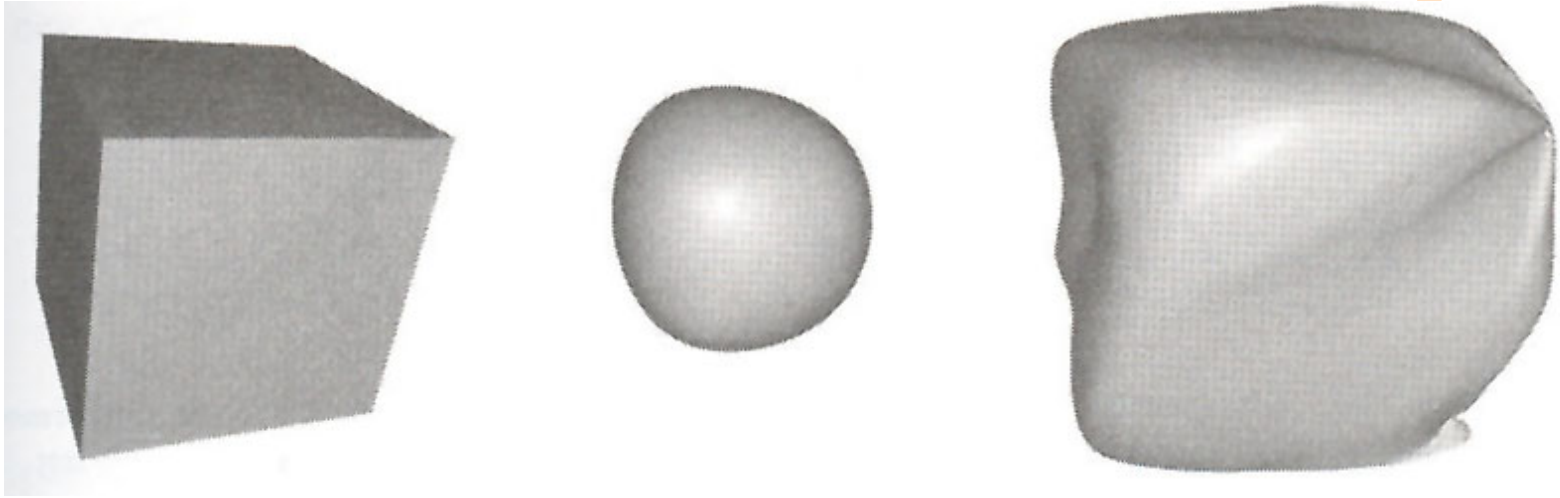


图12.47 左边是立方体网格，中间是使用Loop细分得到的曲面，右边是使用改进的Butterfly方法。每个面由两个三角形组成。可以看到右边的曲面上有不自然的变形。这是因为很难插值给定的点集。

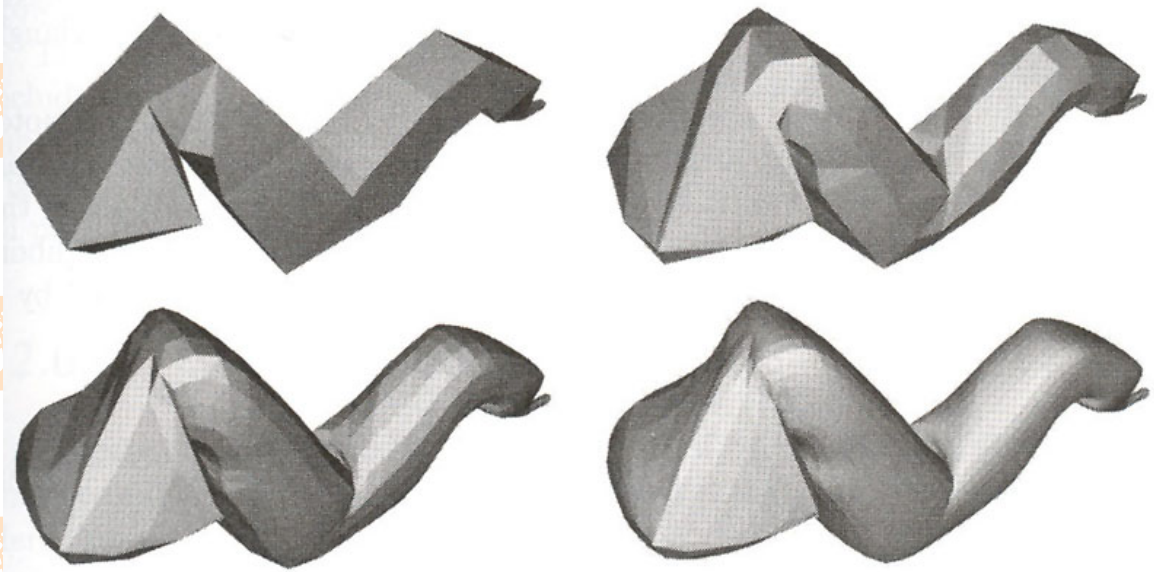


图12.48 使用改进的Butterfly方法细分三次后得到的虫子的模型曲面。在每一个细分步骤，都插值顶点。

$\sqrt{3}$-细分

- **Loop方法与MB方法都是将一个三角形一分为四，所以它们以$4^n m$的速度生成新的三角形，这里m是控制网格上的三角形数量，n是细分操作的次数。**
- **Kobbelt的$\sqrt{3}$-细分方法的一个特点是每次细分仅仅将一个三角形分成3个三角形。思想就是在每个三角形中心生成一个新的点(这里称为中点)，而不是在每条边上生成新的点。如图12.49所示。为了得到更加均匀的三角形形状，每条老边将去掉，然后连接与这条边相关联的两个三角形的中点。经过这个步骤后，新的三角形的特征非常相似于初始三角形。**

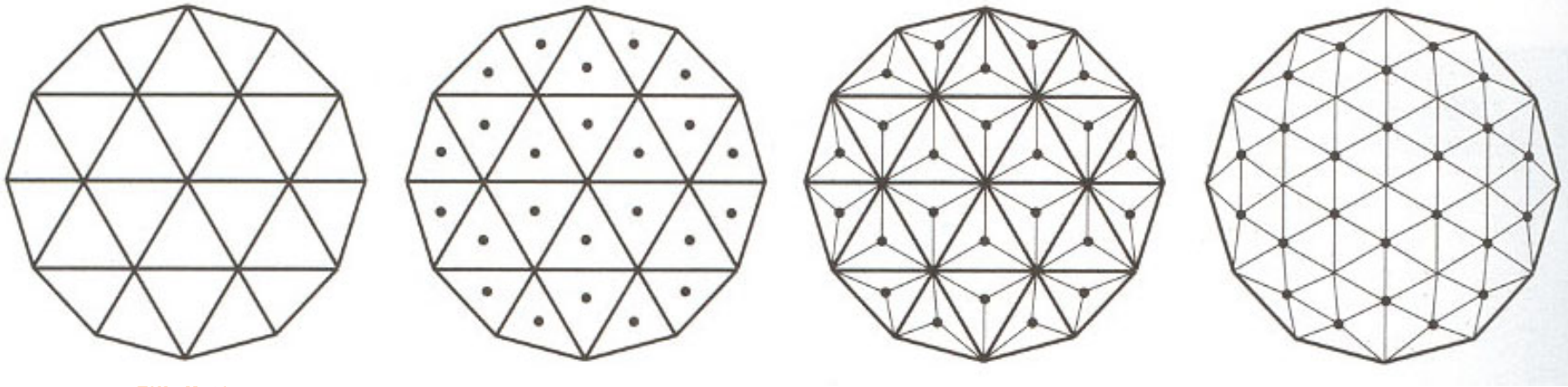


图12.49 $\sqrt{3}$ 一细分方法。首先在每个三角形的中心生成一个新的点；然后，连接中心点与三角形的三个顶点；最后，去掉旧的边，连接相邻三角形的中心点。

- 细分规则

$$\mathrm{p}_m^{k+1} = (\mathrm{p}_a^k + \mathrm{p}_b^k + \mathrm{p}_c^k)/3$$
$$\mathrm{p}^{k+1} = (1-n\beta)\mathrm{p}^{k+1} + \beta\sum_{i=0}^{n-1}\mathrm{p}_i^k \qquad (12.69)$$

这里$\mathbf{p}_m$表示中点，是三角形三个顶点$\mathbf{p}_a$,$\mathbf{p}_b$,$\mathbf{p}_c$的平均。每个老的顶点$\mathbf{p}^k$使用第二个公式进行更新，$\mathbf{p}_i^k(i=0\ldots n-1)$表示$\mathbf{p}_k$的最近的相邻点，$n$是$\mathbf{p}_k$的度数，$k$是细分次数。

β是度数n的函数，如果选择下面的关于β的函数$\beta(n)$，生成的曲面除了不规则点，其他任何地方都是C^2连续的，在不规则点处至少是C^1连续的。

$$\beta(n) = \frac{4-2\cos(2\pi/n)}{9n} \qquad (12.70)$$

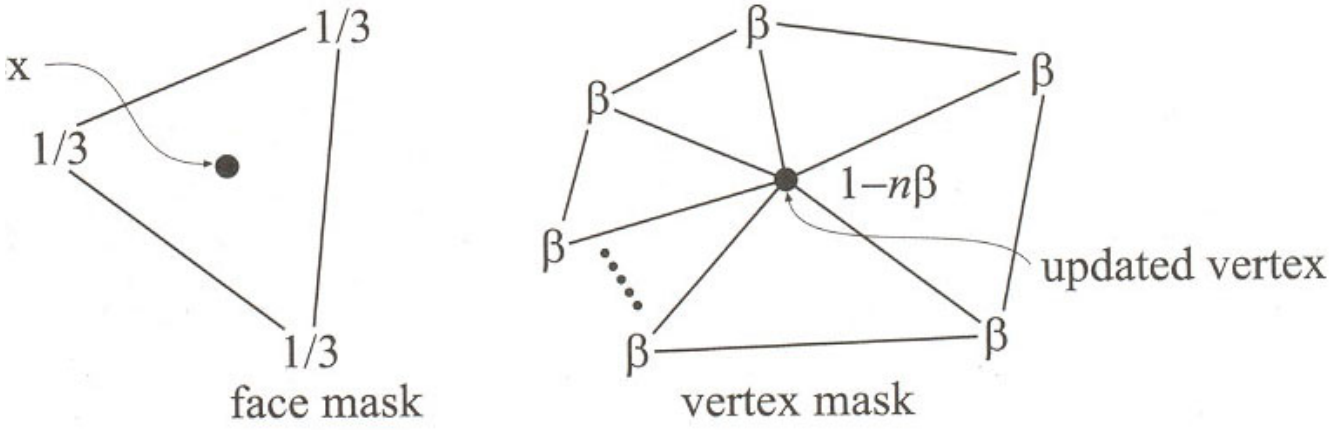


图12.50 细分模板

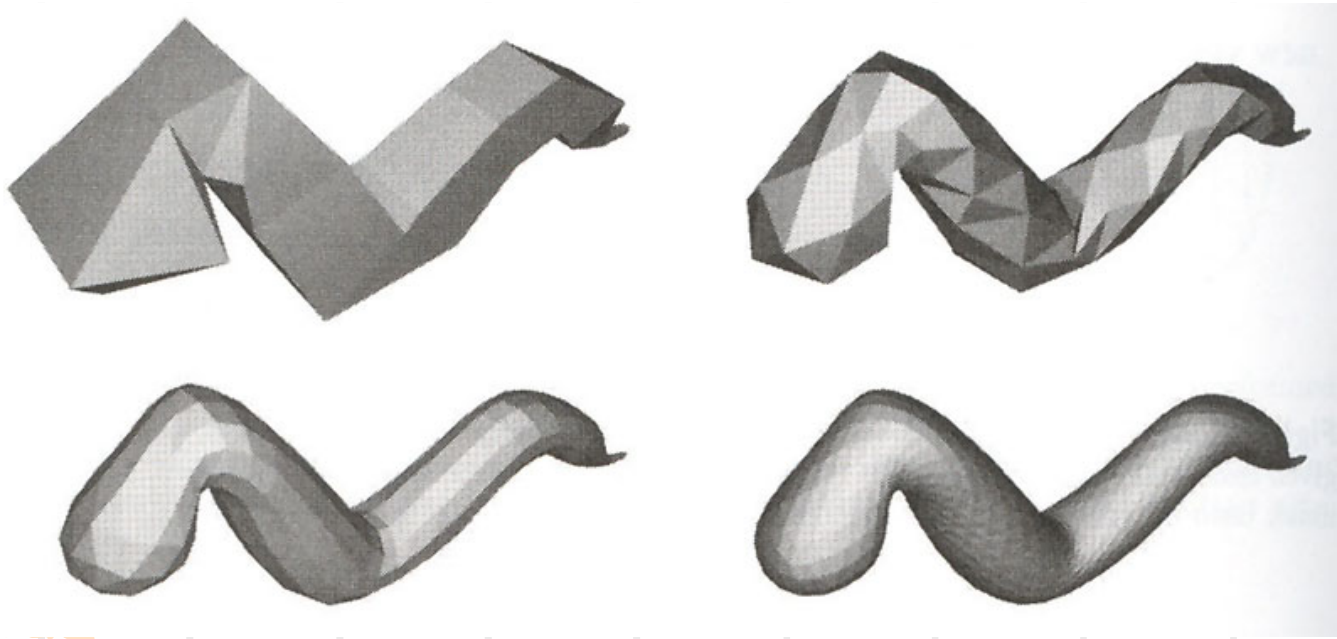


图12.51 使用√3细分方法细分三次得到的虫子曲面

$\sqrt{3}$ -细分评价

- 优点

  A、以一种更自然的方式支持自适应细分(详见[438]);

  B、与Loop和MB方法相比，模板更小，生成三角形的速度更慢。

  C、连续性与Loop方法相同。

- 缺点

  由于要去掉老边，连接相邻三角形的中心点，增加了复杂度; 而且因为这样的原因，在第一次细分之后，有时会得到一些意想不到的形状。

Catmull-Clark细分

- 有两个最著名的细分算法：Catmull-Clark[117]和Doo-Sabin[183]。它们能够处理多边形网格，而不仅仅是三角形网格。
- 这一节简要介绍Catmull-Clark细分算法。使用这个算法生成的曲面已经在Pixar的影片Geri's Game[171]和Toy Story 2中使用。
- 如DeRose et al.[171]指出的，Catmull-Clark曲面比较适合于生成对称性比较强的曲面。如一个椭球形的盒子可以得到一个对称的类似于椭球体的曲面。

- 基本思想

Catmull-Clark细分曲面的基本思想如下图12.52所示。这个方法生成的面包含四个顶点。事实上，在第一次细分后，仅仅生成带有四个顶点的面，所以在这里度数为4的点被称为规则点。

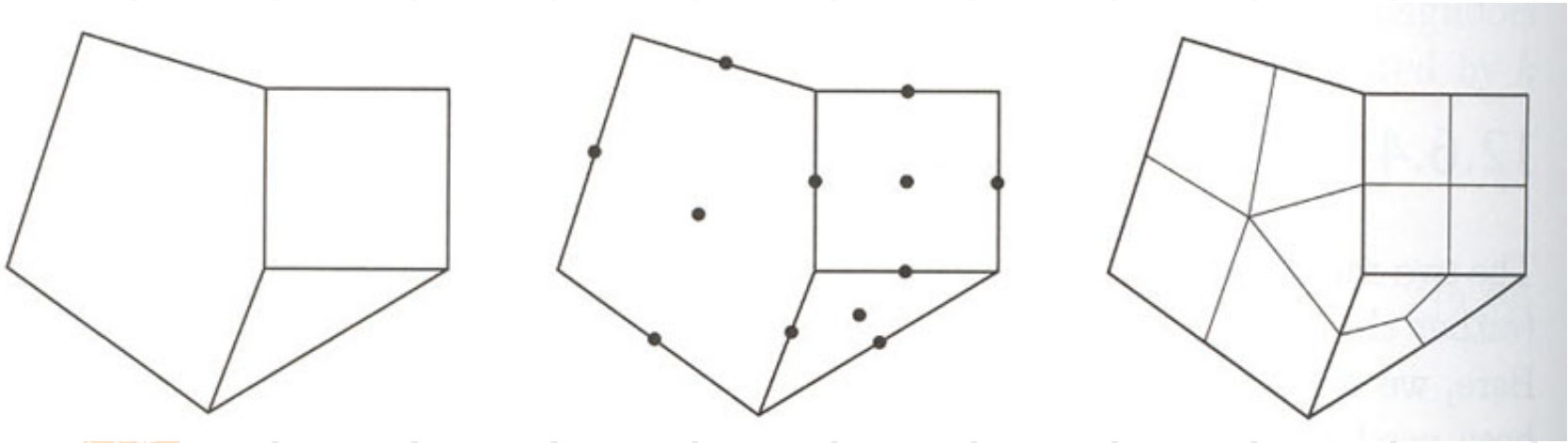


图12.52 Catmull-Clark细分的基本思想。每个多边形生成一个新的点，每条边生成一个新的点。然后如右图进行连接。

- 细分规则

A、对于每个面，计算面的质心生成一个面点；

B、对于边点和点点，可以通过下面的公式计算得到。

$$e_j^{k+1} = \frac{v^k + e_j^k + f_{j-1}^{k+1} + f_j^{k+1}}{4},$$

$$v^{k+1} = \frac{n-2}{n} v^k + \frac{1}{n^2} \sum_{j=1}^{n-1} e_j^k + \frac{1}{n^2} \sum_{j=1}^{n-1} f_j^{k+1} \qquad (12.71)$$

- **Catmull-Clark**细分曲面描述了一张普通的双三次**B**样条曲面(这也是**Catmull-Clark**细分方法的一个优点)。所以，对于仅仅由规则点组成的网格，我们可以将曲面表示成一**B**样条曲面。但是对于有不规则点的情况是不行的。

分片光滑细分
(Piecewise Smooth Subdivision)

- 在某些情况下，使用曲面并不能让人满意，因为它们缺少细节。通过使用**Bump**映射和**Displacement**映射可以改善这种曲面。还有一种方法，就是使用这一节要介绍的分片光滑细分的方法。

- 分片光滑细分方法的基本思想是：通过改变细分规则，可以使用尖刺（**darts**），角点，褶皱。这样扩大了可以建模或者表示的不同曲面的范围。

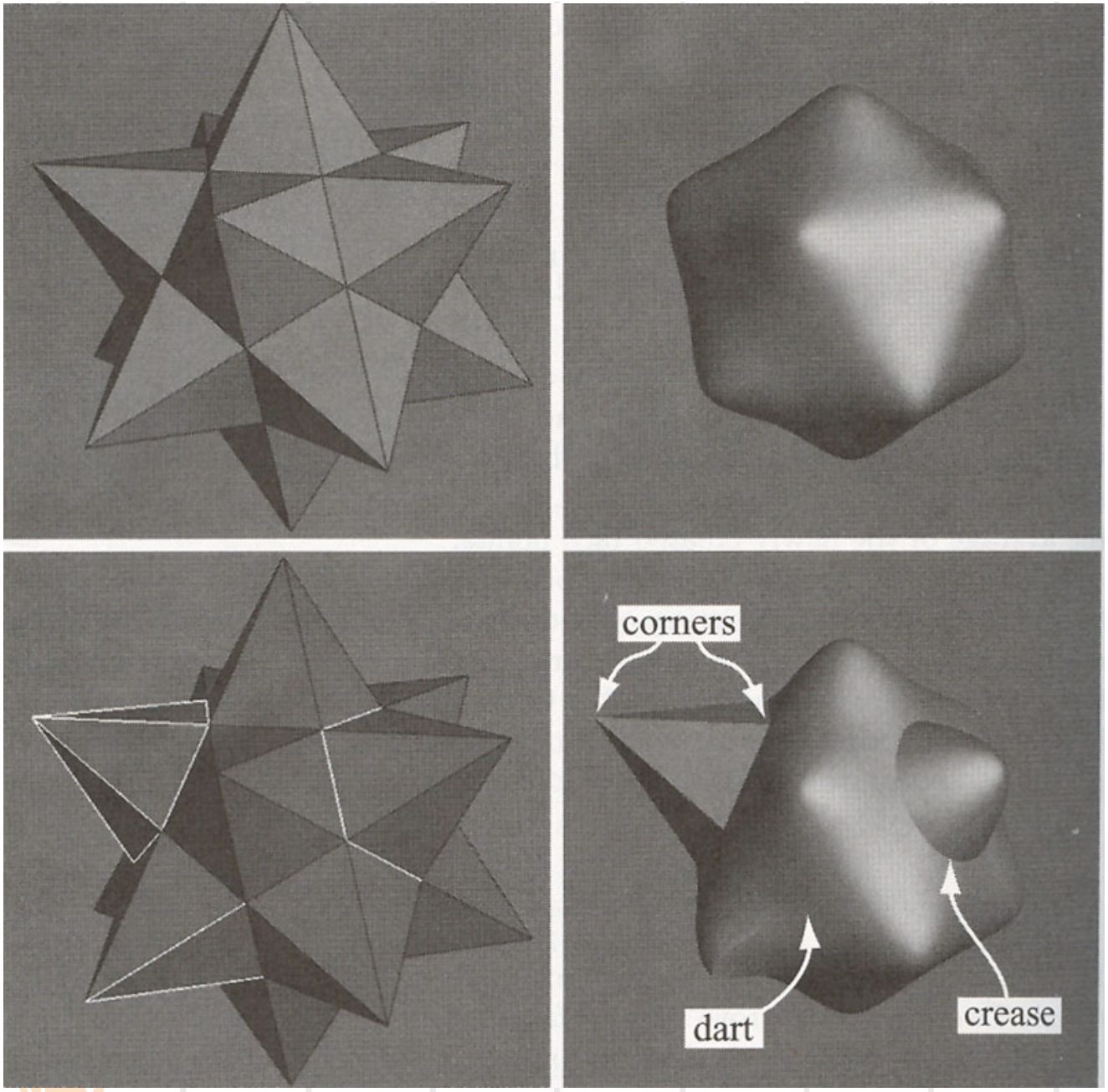


图12.53 上面一行给出一个控制网格和使用标准Loop细分方法得到的极限曲面。

下面一行是使用分片光滑Loop细分方法。左边的控制网格上使用一个亮灰色标出了要显示的尖锐边；右边是结果曲面。

位移细分(Displaced Subdivision)

- 一般的参数曲面与细分曲面可以非常好地描述光滑曲面，但是有时它们缺少细节限制了它们的应用。一个解决方法是使用**Bump**映射，然而，这仅仅是一个光照的技巧，物体的轮廓没有改变。

- 位移细分曲面的基本思想

偏移细分是**Bump**映射的自然扩展。给定一个粗糙的控制网格，使用标准细分方法得到一光滑细分曲面，然后使用一个标量场沿着曲面的法向进行偏移。

- 偏移曲面上点的位置、切向、法向的计算
  - 设定**p**为曲面上的点，单位法向为**n**=**n**’/||**n**’||, 那么偏移后的曲面上的点为

$$\mathbf{s} = \mathbf{p} + d\mathbf{n} \quad (12.72)$$

$$\mathbf{n}' = \mathbf{p}_u \times \mathbf{p}_v,$$
$$\mathbf{n} = \frac{\mathbf{n}'}{\|\mathbf{n}'\|}. \quad (12.73)$$

这里d为点**p**处的偏移量。

$\mathbf{p}_u$,$\mathbf{p}_v$是细分曲面的一阶导数，他们描述了点**p**处的两个切向。如果对初始网格使用**Loop**细分方法，切向的计算可以使用公式**12.63**计算。

  - 结果曲面的切向和法向

$$\mathbf{s}_u = \frac{\partial \mathbf{s}}{\partial u} = \mathbf{p}_u + d_u\mathbf{n} + d\mathbf{n}_u,$$
$$\mathbf{s}_v = \frac{\partial \mathbf{s}}{\partial v} = \mathbf{p}_v + d_v\mathbf{n} + d\mathbf{n}_v, \quad (12.74)$$
$$\mathbf{n}_s = \mathbf{s}_u \times \mathbf{s}_v.$$

- 一阶导数与二阶导数的计算

对于规则点($n=6$)，一阶和二阶导数的计算非常简单，计算模板如图12.54所示。对于非规则点($n\neq6$)，公式12.74中的第一式和第二式右边第三项可以被舍弃。

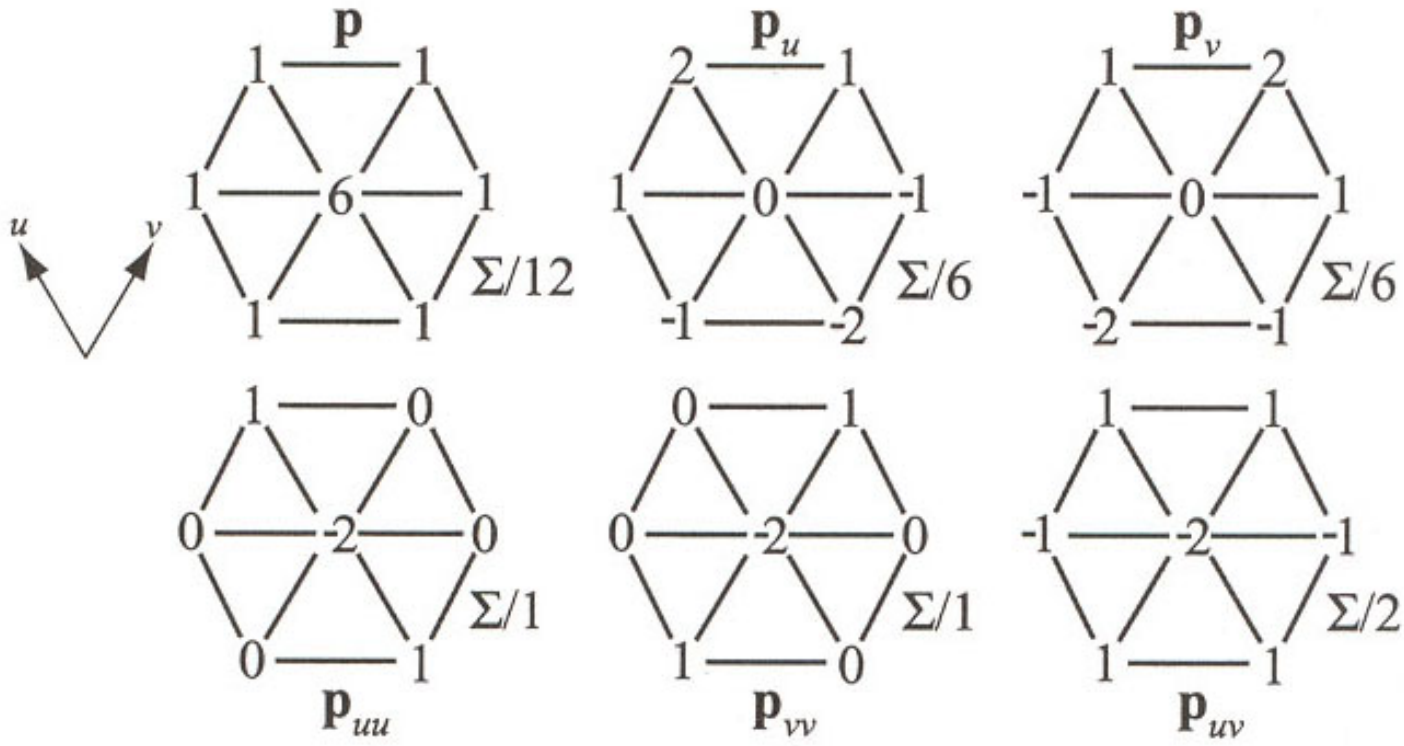


图12.54 Loop细分方法中规则点的计算模板

- 注意
  - 位移映射和细分网格使用相同的参数化。所以当一个三角形被细分后生成三个新的点，这三个点的偏移量可以由位移映射得到。
  - 如果细分次数大于存贮的最大细分层数（位移图），这时位移映射也使用相同的细分方法进行细分。
  - 当物体很远，位移映射将趋近于极限点，这时可以使用预处理时建立的一个偏移量的**mipmap**图。
  - 结果位移曲面除了非规则点处，其他地方都是C^1连续的，在非规则点处是C^0连续的。
  - 当一个位移曲面远离视点，可以使用标准的**Bump**映射给出偏移曲面的光照。这在当绘制瓶颈是几何层时非常有利。

图12.55 左边是初始网格，中间是使用Loop细分方法得到的曲面，右边是偏移细分曲面

细分曲面中的法向、纹理、颜色插值

- 法向

  A、Loop方法和MB方法的极限切向和法向可以显式计算，但是公式中的三角函数计算很费时。使用一个短的查找表可以有效完成这个工作。

  B、另外一个方法是计算控制网格的顶点处的极限法向，然后对法向使用与顶点相同的细分方法进行细分。然而这样在细分过程中，需要更大的存储空间，而且并不一定更快速。使用这个方法，最终并不能得到准确的法向。

- 纹理坐标和颜色

  假设网格上的每个点都有纹理坐标和颜色。为了能在细分曲面上反映这些信息，我们必须也要为每个新生成的点加入颜色和纹理坐标。

  解决方法：对颜色和纹理坐标使用与多边形网格相同的规则进行细分。

实际的细分和绘制方法

- 为了实现细分算法，必须要知道点和边的邻点信息。也就是说要知道连接关系信息。当一次细分所有点都生成了，就要更新连接关系信息，形成新的细分网格。所以实现一个细分方法最主要的部分是找到点、边、面的邻域和细分后的网格更新。

- 这一节我们介绍一些更加有效的细分方法和自适应细分技术。

均匀细分

对于均匀细分，每个三角形都被细分一定次数，可以使用一个二维数组来存放每次细分顶层的三角形[845]。假设要进行$k(k>1)$次细分，对于顶层三角形的每条边，生成了n个顶点。

$$n = 2 + \frac{k(k+1)}{2}$$

顶点的存放如下图所示

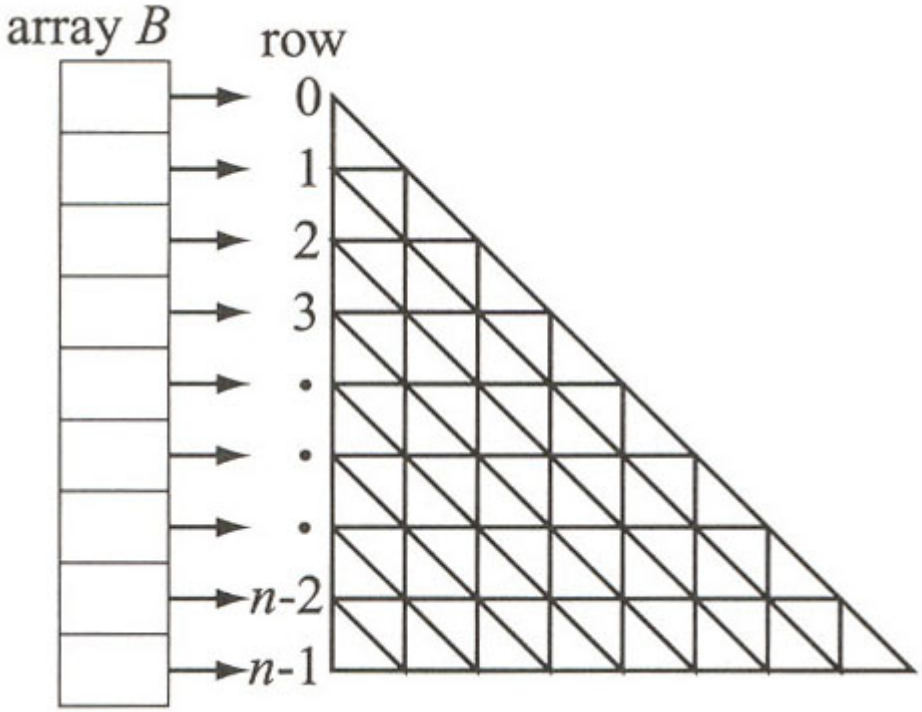


图12.56 数组B用来指向顶点的数组。每个指针指向每行的第一个顶点。这样排列使得邻域的查找变得简单。

由图12.56，很明显，如果要找第i行的顶点，我们只要找到指向这行第一个顶点的指针B[i]。然后，为了找到这行第j列的顶点，直接访问B[i][j]即可。邻域的查找更加简单:在每个点(i,j)其六个相邻点分别在(i±1,j),(i,j±1),(i-1,j-1),(i+1,j+1)。要注意边界的情况。

使用表格的快速细分方法

通过建立一个有限大小的表$\mathbf{B}^i$，它的内容仅仅依赖于网格中的连接关系。使得在生成新点过程中可以避免循环和邻域查找，这样加速了细分的速度。

自适应细分

随着细分层数的增加，三角片的数量成指数级增加。这样导致有些几何形状改变很慢的区域我们浪费使用许多三角片。这时，对网格进行自适应细分是非常有用的。

- 自适应细分的算法可以参考自适应多边形化的算法。而确定是否要进一步多边形化的测试同样适用于改变较少或者没有改变的细分曲面。
- 由于每个顶层三角形都是独立于其他三角形进行细分或者说多边形化的，因而在不同层细分的三角形生成速度不一样。可能出现一个问题：在两个顶层三角形之间会出现裂缝。一个简单的解决办法是生成一些三角形扇形片。如图**12.57**。

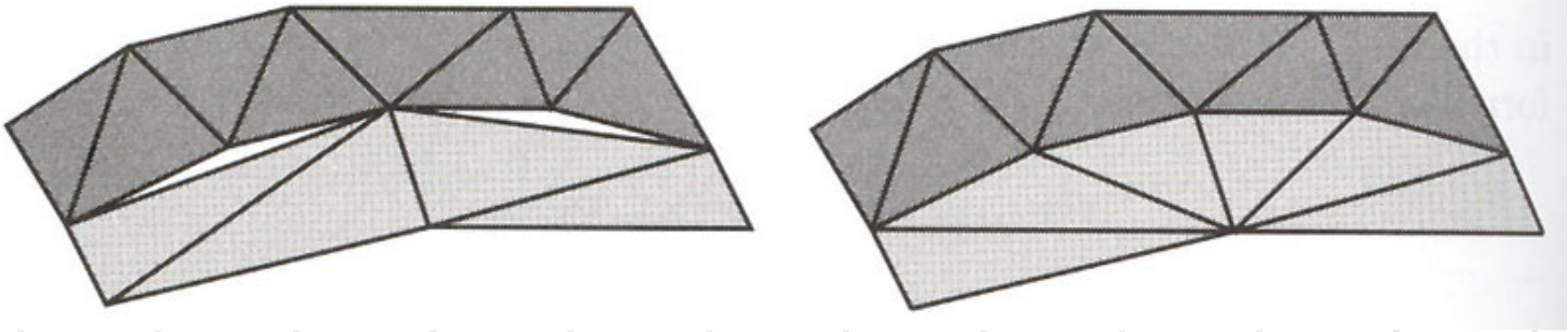


图12.57 左边，两个顶层的三角形（白色和灰色）多边形化不同的层数。他们不匹配了，出现了裂缝。右边，通过生成三角形扇形片消除裂缝。

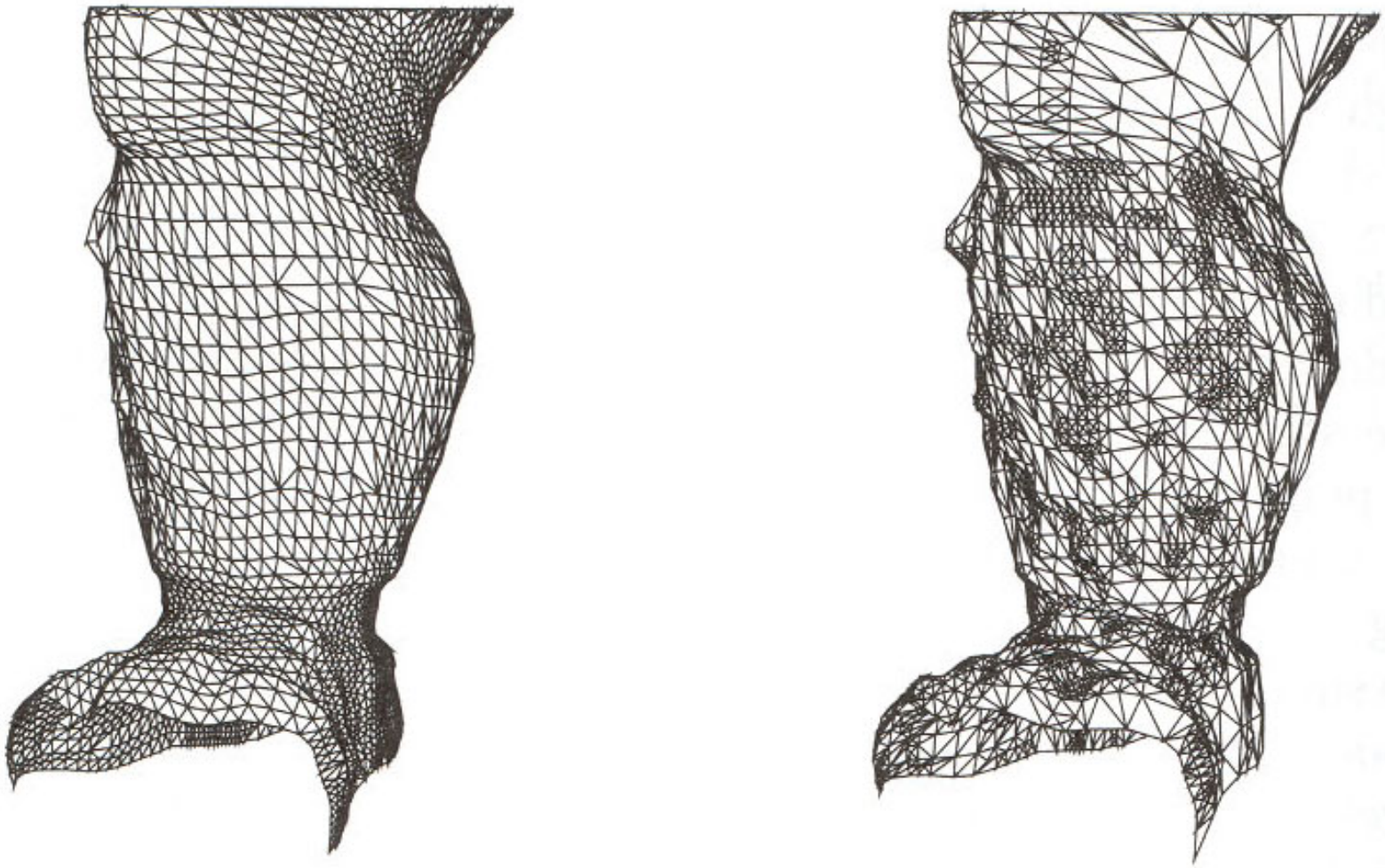


图12.58 左边：均匀细分，9000个三角形。右边：自适应细分，三角形数目基本相同。右边的阴影和轮廓看起来比左边的要好。

进一步阅读资料:

- 几何造型的介绍可以参阅**Mortenson**的书**[573];**
- 计算机辅助几何设计方面的内容可以参阅**Farin[224,227],** 和**Hoschek**与**Lasser[371];**
- 关于隐式曲面，**Bloomenthal et al.[89];**
- 关于细分曲面的内容，**[792],[845];**
- ……