

计算设计的基础 数学

Rajaa Issa 著 ArchiBC 译

向设计专业人员介绍有效开发计算 3D 模型
的基础数学概念。

前言

《计算设计的基础数学》向设计专业人员介绍了有效开发 3D 建模和计算机图形学计算方法所必需的基础数学概念。这并不是一个完整而全面的资源，而是对基本和最常用概念的概述。该材料面向高中以上几乎没有数学背景的设计师。所有概念都使用 Grasshopper 可视化解释。有关 Grasshopper® (GH) 和 Rhinoceros® (Rhino) 的具体信息，请参阅如下链接。¹

所有内容分为三章。第一章讨论向量数学，包括向量的表示，向量的运算以及直线和平面方程。第二章回顾了矩阵运算和转换，第三章包含了参数曲线，特别是 NURBS 曲线以及连续性和曲率的概念，另外还简单回顾了 NURBS 曲线和曲面。

我要感谢 Robert McNeel & Associates 的 Dale Lear 博士出色而彻底的技术审查。他的宝贵意见对本版的制作起到了重要作用。我还要感谢 Robert McNeel & Associates 的 Margaret Becker 女士对技术写作和格式的审查。

Rajaa Issa

Robert McNeel & Associates

¹<https://discourse.mcneel.com/c/grasshopper/2> <https://www.rhino3d.com/>

译者序

时间来到了 2024 年这个节点,《Essential Mathematics for Computational Design》这本书已然出版到了第四版,译者在 Rhino 使用过程中深感基础数学知识的重要性。又见第四版相较第二版有了较多案例和行文的变化,故起重新翻译之心。

本译本以 Rhino 开发者网站上的版本为主,参考了第四版 PDF 和第二版中文翻译。本书版权显然仍归属原作者及 McNeel 公司所有。如有错漏及翻译问题,请联系译者。本书第一版译本为译者使用 markdown 手动翻译、感谢作者 Rajaa Issa 的帮助,我向作者询问了本书的一些问题,得到很多帮助和反馈。翻译过程中发现了原书版本的一些排版问题,这些问题已被原作者修复。

本版译本使用 Typst 重新编译。使用 ilm 模板。

——ArchiBC 中建五局设计技术科研院

版权信息:

Essential Mathematics for Computational Design, Fourth Edition, by Robert McNeel & Associates, 2019 is licensed under a [Creative Commons Attribution-Share Alike 3.0 United States License](<http://creativecommons.org/licenses/by-sa/3.0/us/>).

目录

前言	2
译者序	3
版权信息:	3
1. 向量数学	5
1.1. 向量的表示	5
1.2. 向量的运算	10
1.3. 直线的矢量方程	22
1.4. 平面的向量方程	25
1.5. 实例教程	26
2. 矩阵和变换	48
2.1. 矩阵运算	48
2.2. 变换操作	51
3. 参数曲线和曲面	60
3.1. 曲线参数	62
3.2. NURBS 曲线	68
3.3. 曲线的几何连续性	84
3.4. 曲线曲率	85
3.5. 参数曲面	86
3.6. 曲面的几何连续性	90
3.7. 曲面曲率	91
3.8. NURBS 曲面	94
3.9. 实例教程	102
4. 参考文献	116
4.1. 参考书目	116
4.2. 相关知识链接	116

1. 向量数学

向量表示一个有长度和方向的量，例如速度和力。三维坐标系下向量用三个有序实数表示，如下所示：

$$\vec{v} = \langle a_1, a_2, a_3 \rangle$$

1.1. 向量的表示

在本文档中，带箭头的小写粗体字母表示向量，向量分量用尖括号括起。大写字母表示点，点的坐标用圆括号括起。

使用坐标系和任何一个坐标系内的起始点，我们可以通过线段表示或者可视化这些向量。用箭头表示矢量方向。

例如，如果我们有一个向量在三维坐标系中平行于 x 轴，并且长度是 5 个单位我们可以写出这个向量：

$$\vec{v} = \langle 5, 0, 0 \rangle$$

为了表示这个向量，我们需要一个坐标系内的起始点。例如，下图中的所有箭头都是一个相同向量的等价表示，尽管它们位于不同的位置。

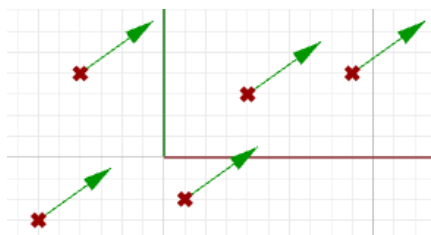


图 1 三维坐标系中的向量表示

给定一个 3D 向量 $\vec{v} = \langle a_1, a_2, a_3 \rangle$ ，向量的分量 a_1, a_2, a_3 是实数，所有的从点 $A(x, y, z)$ 到点 $B(x + a_1, y + a_2, z + a_3)$ 的线段都是向量 \vec{v} 的等价表示。

因此，我们如何定义代表给定向量的线段的结束点呢，我们先定义起始点 A ，令：

$$A = (1, 2, 3)$$

和一个向量：

$$\vec{v} = \langle 5, 6, 7 \rangle$$

那么结束点 B 对应的向量应该通过起始点和向量 \vec{v} 的分量加和得到。

$$\begin{aligned} B &= A + \vec{v} \\ &= (1 + 5, 2 + 6, 3 + 7) \\ &= (6, 8, 10) \end{aligned}$$

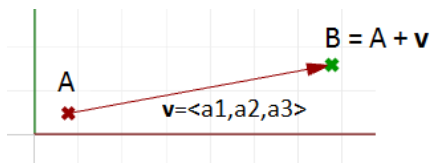


图 2 向量、向量起始点、向量结束位置所在点之间的关系

1.1.1. 位置向量

一种特殊的向量表示方法使用 **origin point**(0,0,0) 作为向量的起始点, 那么位置向量 $\vec{v} = \langle a_1, a_2, a_3 \rangle$ 能够被 **原点** 和结束点 B 构成的线段表示。由此得到:

$$\text{origin point} = (0, 0, 0)$$

$$B = (a_1, a_2, a_3)$$

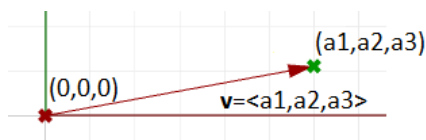


图 3 位置向量, 端点坐标等于向量坐标

一个给定向量的位置向量 $\vec{v} = \langle a_1, a_2, a_3 \rangle$ 能被从原点到点 (a_1, a_2, a_3) 的特殊线段表示。

1.1.2. 向量和点

不要混淆向量和点, 他们是非常不同的概念, 正如我们之前所说, 向量表示具有长度和方向的量, 但点表示位置。例如, 北方是一个向量, 北极是一个位置(点)。如果我们有一个向量和一个点具有相同的分量, 例如:

$$\vec{v} = \langle 3, 1, 0 \rangle$$

$$P = (3, 1, 0)$$

我们可以绘制如下向量和点:

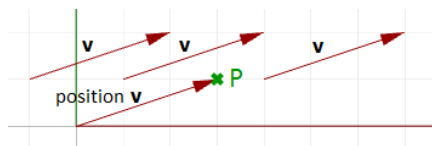


图 4 向量定义方向和长度，点定义位置

1.1.3. 向量长度

正如前面所说，向量有长度，我们使用 $|\vec{a}|$ 来表示给定向量 \vec{a} 的长度,例如：

$$\vec{a} = \langle 4, 3, 0 \rangle$$

$$|\vec{a}| = \sqrt{4^2 + 3^2 + 0^2}$$

$$|\vec{a}| = 5$$

一般来说，向量 $\vec{a} = \langle a_1, a_2, a_3 \rangle$ 的长度的计算如下：

$$|\vec{a}| = \sqrt{(a_1)^2 + (a_2)^2 + (a_3)^2}$$

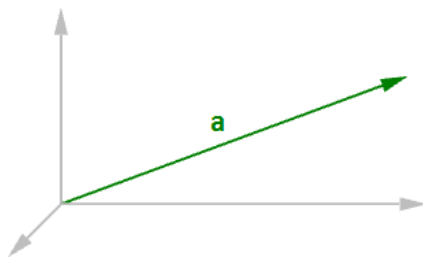


图 5 向量长度

1.1.4. 单位向量

单位向量是长度等于一个单位的向量，单位向量通常用于比较向量的方向。

单位向量是长度等于一个单位的向量。

要计算一个单位向量，需要找到需要计算向量的长度，并且将向量的分量除以向量长度。例如：

$$\vec{a} = \langle 4, 3, 0 \rangle$$

$$|\vec{a}| = \sqrt{4^2 + 3^2 + 0^2}$$

$$|\vec{a}| = 5 \text{ 单位长度}$$

如果 \vec{b} 是 \vec{a} 的单位向量，那么：

$$\vec{b} = \left\langle \frac{4}{5}, \frac{3}{5}, \frac{0}{5} \right\rangle$$

$$\vec{b} = \langle 0.8, 0.6, 0 \rangle$$

$$|\vec{b}| = \sqrt{0.8^2 + 0.6^2 + 0^2}$$

$$|\vec{b}| = \sqrt{0.64 + 0.36 + 0}$$

$$|\vec{b}| = 1 \text{ 单位长度}$$

一般的：

$$\vec{a} = \langle a_1, a_2, a_3 \rangle$$

$$\vec{a} \text{ 的单位向量} = \left\langle \frac{a_1}{|\vec{a}|}, \frac{a_2}{|\vec{a}|}, \frac{a_3}{|\vec{a}|} \right\rangle$$

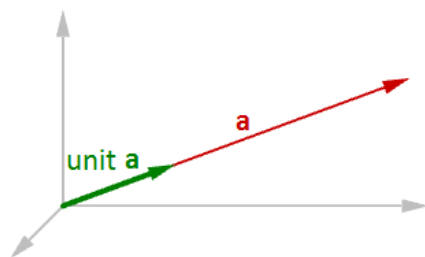


图 6 单位向量等于一个单位长度的向量

1.2. 向量的运算

1.2.1. 向量标量运算

向量的标量运算就是一个一个向量乘以一个实数。例如：

$$\vec{a} = \langle 4, 3, 0 \rangle$$

$$2 \cdot \vec{a} = \langle 2 \times 4, 2 \times 3, 0 \rangle$$

$$2 \cdot \vec{a} = \langle 8, 6, 0 \rangle$$

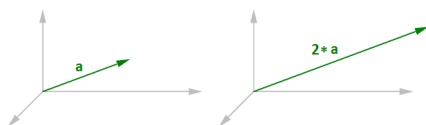


图 7 向量标量运算

一般的：给定向量 $\vec{a} = \langle a_1, a_2, a_3 \rangle$ 和一个实数 t

$$t \cdot \vec{a} = \langle t \cdot a_1, t \cdot a_2, t \cdot a_3 \rangle$$

1.2.2. 向量加法

向量加法是两个向量之间的运算并且得到第三个向量，我们通过加和向量的分量来实现向量相加。

向量加法通过分量相加实现

例如，我们有两个向量：

$$\vec{a} = \langle 1, 2, 0 \rangle$$

$$\vec{b} = \langle 4, 1, 3 \rangle$$

$$\vec{a} + \vec{b} = \langle 1 + 4, 2 + 1, 0 + 3 \rangle$$

$$\vec{a} + \vec{b} = \langle 5, 3, 3 \rangle$$

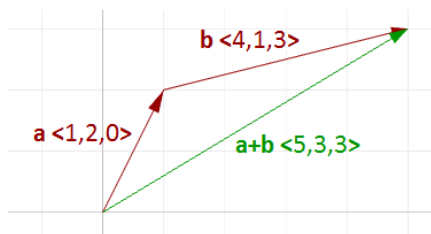


图 8 向量加法

一般来说，两个向量相加的运算如下：

$$\vec{a} = \langle a_1, a_2, a_3 \rangle$$

$$\vec{b} = \langle b_1, b_2, b_3 \rangle$$

$$\vec{a} + \vec{b} = \langle a_1 + b_1, a_2 + b_2, a_3 + b_3 \rangle$$

向量相加对于找到两个或者更多向量的平均方向非常有用。在这种情况下，我们通常使用相同长度的向量。下面是一个例子展示了使用相同长度向量和不同长度向量加和的区别。

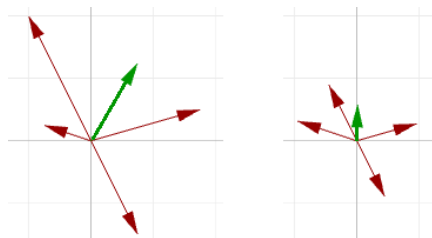


图 9 对不同长度向量求和（左）对相同长度向量求和（右）来获得平均方向

1.2.3. 向量减法

向量减法是两个向量的运算并且得到第三个向量，我们通过向量分量相减来实现向量的减法。例如，如果我们有二个向量 \vec{a} 和 \vec{b} ，并且我们令 \vec{a} 减去 \vec{b} ，那么：

$$\vec{a} = \langle 1, 2, 0 \rangle$$

$$\vec{b} = \langle 4, 1, 4 \rangle$$

$$\begin{aligned}\vec{a} - \vec{b} &= \langle 1 - 4, 2 - 1, 0 - 3 \rangle \\ &= \langle -3, 1, -4 \rangle\end{aligned}$$

如果我们令 \vec{b} 减去 \vec{a} ，我们会得到不同的结果：

$$\begin{aligned}\vec{b} - \vec{a} &= \langle 4 - 1, 1 - 2, 4 - 0 \rangle \\ &= \langle 3, -1, 4 \rangle\end{aligned}$$

注意向量 $\vec{b} - \vec{a}$ 和 $\vec{a} - \vec{b}$ 长度相同但是方向相反。

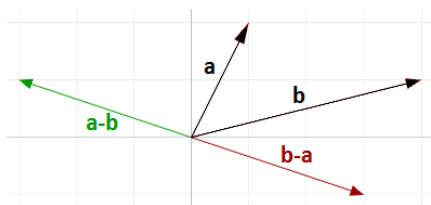


图 10 向量减法

一般来说，如果我们有二个向量 \vec{a} 和 \vec{b} ，那么 $\vec{a} - \vec{b}$ 向量的计算如下：

$$\vec{a} = \langle a_1, a_2, a_3 \rangle$$

$$\vec{b} = \langle b_1, b_2, b_3 \rangle$$

$$\vec{a} - \vec{b} = \langle a_1 - b_1, a_2 - b_2, a_3 - b_3 \rangle$$

向量减法通常用于找到两个点之间的向量，所以如果我们需要找到一个从位置向量 \vec{b} 结束点到位置向量 \vec{a} 结束点的向量，我们就会使用向量减法令 $\vec{a} - \vec{b}$ ，如图 11 所示：

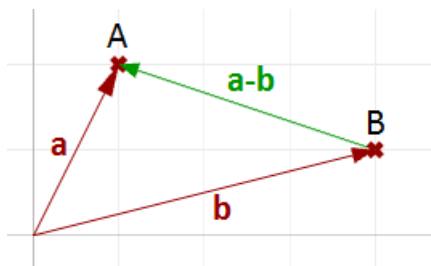


图 11 使用向量减法找到两点之间的向量

1.2.4. 向量的基本特性

这里有八个向量的基本特性，令 \vec{a} ， \vec{b} ， \vec{c} 为向量， s ， t 为实数，我们有这样几条性质：

$$\vec{a} + \vec{b} = \vec{b} + \vec{a}$$

$$\vec{a} + 0 = \vec{a}$$

$$s \cdot (\vec{a} + \vec{b}) = s \cdot \vec{a} + s \cdot \vec{b}$$

$$s \cdot t \cdot \vec{a} = s \cdot (t \cdot \vec{a})$$

$$\vec{a} + (\vec{b} + \vec{c}) = (\vec{a} + \vec{b}) + \vec{c}$$

$$\vec{a} + (-\vec{a}) = 0$$

$$(s + t) \cdot \vec{a} = s \cdot \vec{a} + t \cdot \vec{a}$$

$$1 \cdot \vec{a} = \vec{a}$$

1.2.5. 向量点积

向量点积是一个两个向量得到一个实数的运算。例如，如果我们有两个向量 \vec{a} 和 \vec{b}

$$\vec{a} = \langle 1, 2, 0 \rangle, \quad \vec{b} = \langle 4, 1, 3 \rangle$$

那么，点积是向量各分量乘积的和：

$$\vec{a} \cdot \vec{b} = 1 * 4 + 2 * 1 + 0 * 3$$

$$\vec{a} \cdot \vec{b} = 7$$

通常来说，给两个向量 \vec{a} 和 \vec{b} ：

$$\vec{a} = \langle a_1, a_2, a_3 \rangle, \quad \vec{b} = \langle b_1, b_2, b_3 \rangle$$

$$\vec{a} \cdot \vec{b} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

当我们做点积运算时，两个向量有相同的方向时我们能得到一个正实数。两个向量之间的点积为负意味着两个向量有相反的方向。

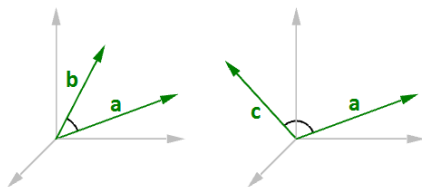


图 12 当两个向量同方向时，结果是正点积（左），当两个向量不同方向时，结果是负点积（右）

当计算两个单位向量的点积时，结果在-1 到 1 的闭区间上。例如：

$$\vec{a} = \langle 1, 0, 0 \rangle, \vec{b} = \langle 0.6, 0.8, 0 \rangle$$

$$\vec{a} \cdot \vec{b} = 1 \cdot 0.6 + 0 \cdot 0.8 + 0 \cdot 0 = 0.6$$

另外向量和向量自身的点积是向量长度的平方。例如：

$$\vec{a} = \langle 0, 3, 4 \rangle$$

$$\vec{a} \cdot \vec{a} = 0 \cdot 0 + 3 \cdot 3 + 4 \cdot 4 = 25$$

计算 \vec{a} 的长度的平方：

$$|\vec{a}| = \sqrt{4^2 + 3^2 + 0^2} = 5$$

$$|\vec{a}|^2 = 25$$

1.2.6. 向量点积、长度和角度

两个向量的点积和他们之间的夹角有关。

两个非零单位向量的点积等于两向量夹角点余弦值

一般的：

$$\vec{a} \cdot \vec{b} = |\vec{a}| \cdot |\vec{b}| \cdot \cos \theta$$

或者表示为：

$$\frac{\vec{a} \cdot \vec{b}}{|\vec{a}| \cdot |\vec{b}|} = \cos \theta$$

在这里： θ 是两向量之间的夹角。

如果两个向量均为单位向量，我们把公式简化为：

$$\vec{a} \cdot \vec{b} = \cos \theta$$

因为 90 度角的余弦值是 0 所以我们有结论：

当且仅当 $\vec{a} \cdot \vec{b} = 0$ 时， \vec{a} 垂直于 \vec{b} 。

例如我们计算两个正交向量——x 轴单位向量和 y 轴单位向量的点积，结果为 0。

$$\vec{x} = \langle 1, 0, 0 \rangle, \vec{y} = \langle 0, 1, 0 \rangle$$

$$\vec{x} \cdot \vec{y} = 1 \cdot 0 + 0 \cdot 1 + 0 \cdot 0 = 0$$

点积和一个向量到另一个向量的投影长度也有关系。例如：

$$\vec{a} = \langle 5, 2, 0 \rangle, \vec{b} = \langle 9, 0, 0 \rangle$$

$$\text{unit } \vec{b} = \langle 1, 0, 0 \rangle$$

$$\vec{a} \cdot \text{unit } \vec{b} = 5 \cdot 1 + 2 \cdot 0 + 0 \cdot 0 = 5$$

等于 \vec{a} 到 \vec{b} 的投影长度

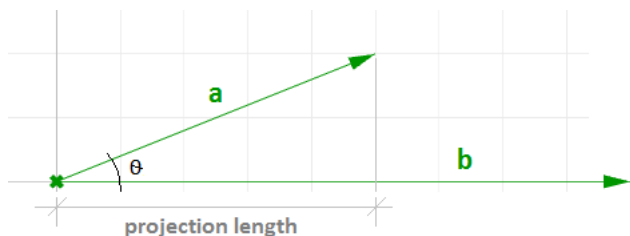


图 13 一个向量的投影长度是一个向量和另一个向量的单位向量的点积

一般来说，给定一个向量 \vec{a} 和一个非零向量 \vec{b} ，我们可以计算投影长度pL通过计算向量 \vec{a} 和向量 \vec{b} 的单位向量的点积。

$$pL = |\vec{a}| \cdot \cos \theta = \vec{a} \cdot \text{unit } \vec{b}$$

1.2.7. 点积的基本性质

如果 \vec{a} 、 \vec{b} 、 \vec{c} 是向量， s 是实数，我们有如下性质：

$$\vec{a} \cdot \vec{a} = |\vec{a}|^2$$

$$\vec{a} \cdot (\vec{b} + \vec{c}) = \vec{a} \cdot \vec{b} + \vec{a} \cdot \vec{c}$$

$$0 \cdot \vec{a} = 0$$

$$\vec{a} \cdot \vec{b} = \vec{b} \cdot \vec{a}$$

$$(s \cdot \vec{a}) \cdot \vec{b} = s \cdot (\vec{a} \cdot \vec{b}) = \vec{a} \cdot (s \cdot \vec{b})$$

1.2.8. 向量叉积

叉积是两个向量得到第三个垂直于两向量的向量的运算。

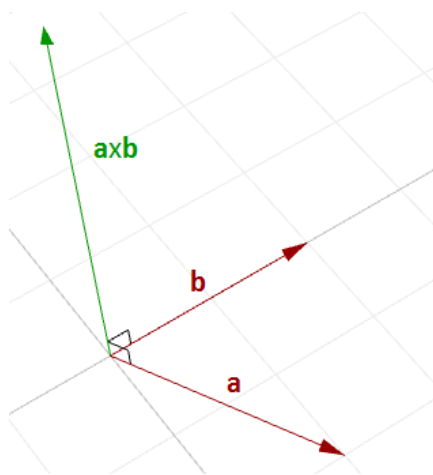


图 14 计算两个向量的叉积

例如，如果有两个在 XY 平面上的向量，则他们的叉积是一个垂直于 XY 平面的向量，方向为正或负的 Z 轴方向。

$$\vec{a} = \langle 3, 1, 0 \rangle, \vec{b} = \langle 1, 2, 0 \rangle$$

$$\begin{aligned}\vec{a} \times \vec{b} &= \langle (1 \cdot 0 - 0 \cdot 2), (0 \cdot 1 - 3 \cdot 0), (3 \cdot 2 - 1 \cdot 1) \rangle \\ &= \langle 0, 0, 5 \rangle\end{aligned}$$

向量 $\vec{a} \times \vec{b}$ 垂直于 \vec{a} 和 \vec{b} 。

你可能永远不需要手工计算两个向量的叉积，但如果你对如何计算有兴趣，请读下一节，否则可以跳过，叉积是用行列式进行定义的。以下是如何使用标准基向量进行行列式的简单示例。

$$i = \langle 1, 0, 0 \rangle, \quad j = \langle 0, 1, 0 \rangle, \quad k = \langle 0, 0, 1 \rangle$$

$$\vec{a} \times \vec{b} = \begin{vmatrix} i & j & k \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} = \begin{vmatrix} i & j \\ a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} + \begin{vmatrix} j & k \\ a_2 & a_3 \\ b_2 & b_3 \end{vmatrix} + \begin{vmatrix} k & i \\ a_3 & a_1 \\ b_3 & b_1 \end{vmatrix}$$

两个向量 $\vec{a} = \langle a_1, a_2, a_3 \rangle$ 和 $\vec{b} = \langle b_1, b_2, b_3 \rangle$ 的叉积使用上图计算如下²：

$$\begin{aligned}\vec{a} \times \vec{b} &= i(a_2 \cdot b_3) + j(a_3 \cdot b_1) + k(a_1 \cdot b_2) \\ &\quad - i(a_2 \cdot b_1) - j(a_3 \cdot b_2) - k(a_1 \cdot b_3) \\ &= i(a_2 \cdot b_3 - a_3 \cdot b_2) + j(a_3 \cdot b_1 - a_1 \cdot b_3) \\ &\quad + k(a_1 \cdot b_2 - a_2 \cdot b_1) \\ &= \langle a_2 \cdot b_3 - a_3 \cdot b_2, a_3 \cdot b_1 - a_1 \cdot b_3, a_1 \cdot b_2 - a_2 \cdot b_1 \rangle\end{aligned}$$

²译者注：这个解释过于抽象，建议观看 3Blue1Brown 的线性代数视频加以理解。或者看看英文版本附带的视频。

1.2.9. 叉积和两个向量的夹角

两个向量之间的夹角和向量叉积的模有关。角度越小（正弦值越小），叉积的模越短。在向量叉积运算中，运算次序很重要，例如：

$$\vec{a} = \langle 1, 0, 0 \rangle, \vec{b} = \langle 0, 1, 0 \rangle$$

$$\vec{a} \times \vec{b} = \langle 0, 0, 1 \rangle$$

$$\vec{b} \times \vec{a} = \langle 0, 0, -1 \rangle$$

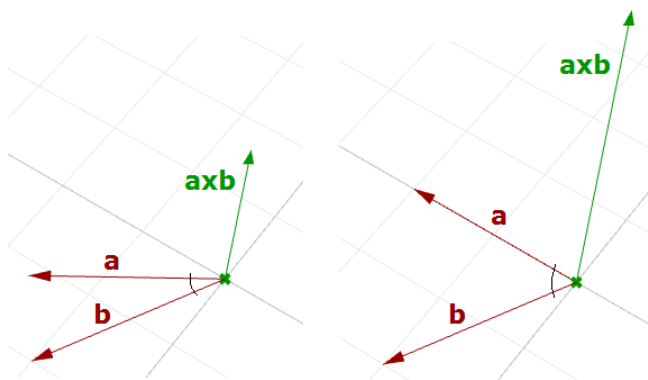


图 15 两个向量夹角的正弦和叉积的模之间的关系

在犀牛的右手系中， $\vec{a} \times \vec{b}$ 的方向由右手定则给出（其中 \vec{a} 是食指， \vec{b} 是中指， $\vec{a} \times \vec{b}$ 是拇指）

一般来说对任意的两个向量 \vec{a} 和 \vec{b}

$$|\vec{a} \times \vec{b}| = |\vec{a}| \cdot |\vec{b}| \cdot \sin \theta$$

其中：

θ 是向量 \vec{a} 和 \vec{b} 之间的夹角

如果 \vec{a} 和 \vec{b} 是单位向量, 那么我们可以说, 两向量叉积的模等于向量夹角的正弦。也就是:

$$|\vec{a} \times \vec{b}| = \sin \theta$$

两个向量叉积帮我们判断两个向量是否平行, 当平行时叉积为零向量。

向量 \vec{a} 和 \vec{b} 平行, 当且仅当 $\vec{a} \times \vec{b} = \vec{0}$ 。

1.2.10. 叉积性质

如果 \vec{a}, \vec{b} 和 \vec{c} 是向量, s 是一个实数, 我们有如下性质:

$$\vec{a} \times \vec{b} = -\vec{b} \times \vec{a}$$

$$(s \cdot \vec{a}) \times \vec{b} = s \cdot (\vec{a} \times \vec{b}) = \vec{a} \times (s \cdot \vec{b})$$

$$\vec{a} \times (\vec{b} + \vec{c}) = \vec{a} \times \vec{b} + \vec{a} \times \vec{c}$$

$$(\vec{a} + \vec{b}) \times \vec{c} = \vec{a} \times \vec{c} + \vec{b} \times \vec{c}$$

$$\vec{a} \cdot (\vec{b} \times \vec{c}) = (\vec{a} \times \vec{b}) \cdot \vec{c}$$

$$\vec{a} \times (\vec{b} \times \vec{c}) = (\vec{a} \cdot \vec{c}) \times \vec{b} - (\vec{a} \cdot \vec{b}) \times \vec{c}$$

1.3. 直线的矢量方程

直线的矢量方程适用与三维建模应用和计算机图形学。

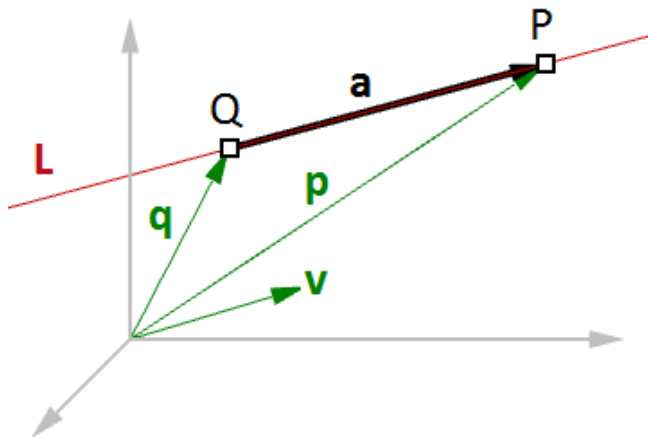


图 16 直线的矢量方程

例如，如果我们知道一条直线的方向和直线上的一个点，那么我们可以使用向量的方法，找到这条直线上的任意一点。如下所示：

$$\overline{L} = \text{line}$$

$$\vec{v} = \langle a, b, c \rangle \quad \text{表示直线方向的单位向量}$$

$$Q = (x_0, y_0, z_0) \quad \text{直线的基准点}$$

$$P = (x, y, z) \quad \text{直线上任意一点}$$

我们知道：

$$\vec{a} = t \cdot \vec{v} \quad (1)$$

$$\vec{p} = \vec{q} + \vec{a} \quad (2)$$

由 (1) 和 (2) 可知:

$$\vec{p} = \vec{q} + t \cdot \vec{v} \quad (3)$$

由此, 我们可以将 (3) 式改写为:

$$\langle x, y, z \rangle = \langle x_0, y_0, z_0 \rangle + \langle t \cdot a, t \cdot b, t \cdot c \rangle$$

$$\langle x, y, z \rangle = \langle x_0 + t \cdot a, y_0 + t \cdot b, z_0 + t \cdot c \rangle$$

也就是:

$$x = x_0 + t \cdot a$$

$$y = y_0 + t \cdot b$$

$$z = z_0 + t \cdot c$$

等价于:

$$P = Q + t \cdot \vec{v}$$

给定直线上一点 Q 和方向 \vec{v} , 直线上任意一点 P 都可以用方程 $P = Q + t \cdot \vec{v}$ 计算, t 是参数。

另一个常用的例子是找两点的中点, 下图显示了如何使用直线的向量方程找到中点:

\vec{q} 是点 Q 的位置向量, \vec{p} 是点 P 的位置向量, \vec{a} 是从 Q 到 P 的向量。

由向量减法可知:

$$\vec{a} = \vec{p} - \vec{q}$$

由向量的直线方程可知：

$$M = Q + t \cdot \vec{a}$$

既然我们需要找到中点，那么令： $t = 0.5$

因此我们得到：

$$M = Q + 0.5 \cdot \vec{a}$$

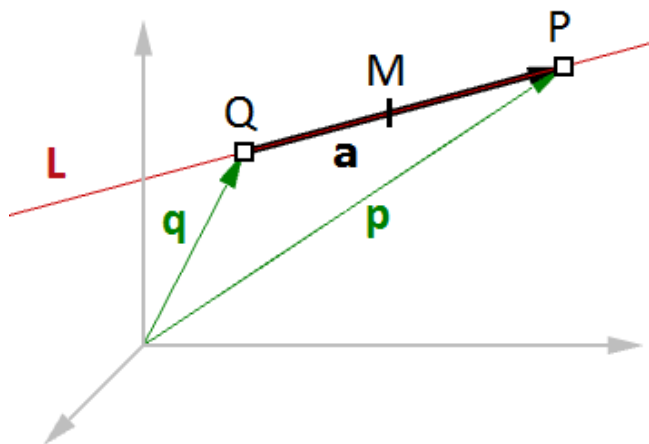


图 17 找到两点之间的中点

一般的，通过使用该方程将 t 跟改为 0 和 1 之间的值，可以找到 Q 和 P 之间任意点：

$$M = Q + t \cdot (P - Q)$$

给定两个点 P 和 Q ，两点之间任意一点 M 都能使用方程 $M = Q + t \cdot (P - Q)$ 计算，其中 t 在 0 和 1 之间。

1.4. 平面的向量方程

定义平面的一种方法是，通过一个点和一个垂直于平面的向量。这个向量被称作平面的法向量。法线方向指向平面正上方。

例如当我们知道平面三个不共线的点时，该如何计算平面的法向。

见图 18：令：

A = 平面上第一个点

B = 平面上第二个点

C = 平面上第三个点

并且：

\vec{a} = 点 A 的位置向量

\vec{b} = 点 B 的位置向量

\vec{c} = 点 C 的位置向量

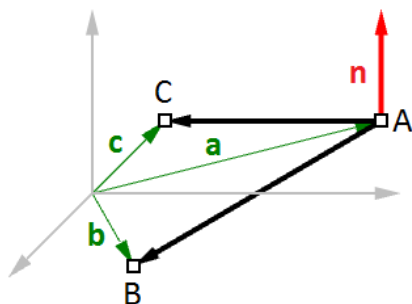


图 18 向量和平面

我们可以找到法向向量³ \vec{n} :

$$\vec{n} = (\vec{b} - \vec{a}) \times (\vec{c} - \vec{a})$$

我们还能利用向量点积推导平面的隐式方程:

$$\vec{n} \cdot (\vec{b} - \vec{a}) = 0$$

如果:

$$\vec{n} = \langle a, b, c \rangle$$

$$\vec{b} = \langle x, y, z \rangle$$

$$\vec{a} = \langle x_0, y_0, z_0 \rangle$$

代入上式可得:

$$\langle a, b, c \rangle \cdot \langle x - x_0, y - y_0, z - z_0 \rangle = 0$$

解这个点积方程可以得到平面的隐式方程:

$$a \cdot (x - x_0) + b \cdot (y - y_0) + c \cdot (z - z_0) = 0$$

1.5. 实例教程

我们在本章学习到的所有概念都可以用于解决建模时常见的几何问题，以下时使用本章学习到的概念使用犀牛和Grasshopper的具体教程。

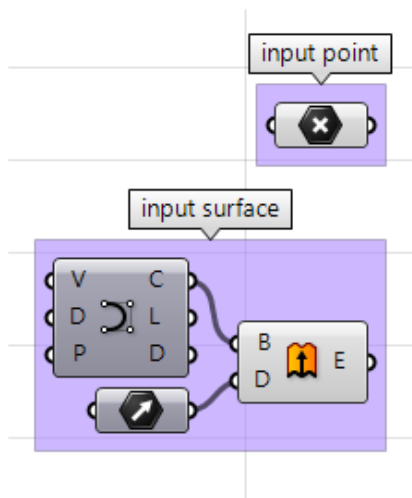
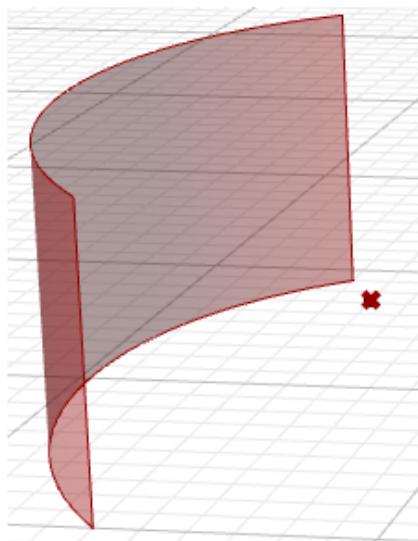
1.5.1. 面方向

给定一个点和一个曲面，我们该如何确定该点面向曲面的正面还是背面？

³译者注：a, b, c 任意两两组合相减做叉积都可以找到法向

输入:

- 一个曲面
- 一个点



参数:

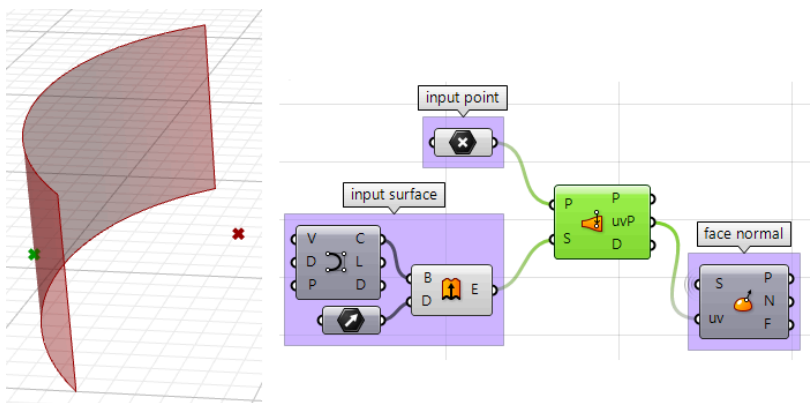
面方向由曲面法向定义, 我们需要以下信息:

- 最靠近输入点的曲面位置处的曲面法向向量。
- 从最近点到输入点的矢量。

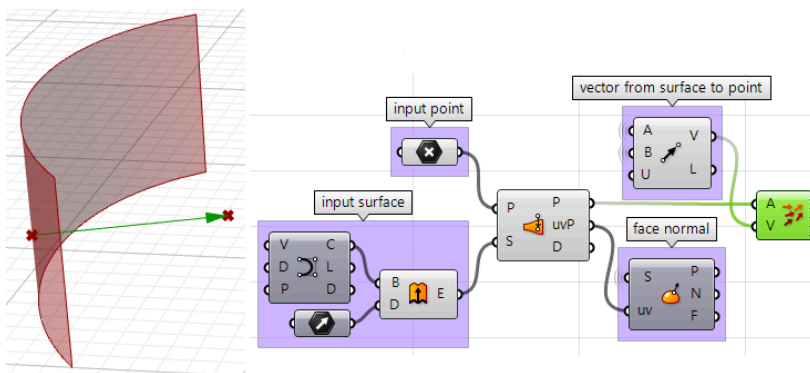
比较以上两个矢量, 如果同向, 则点在正面, 否则点在反面。

解决方案:

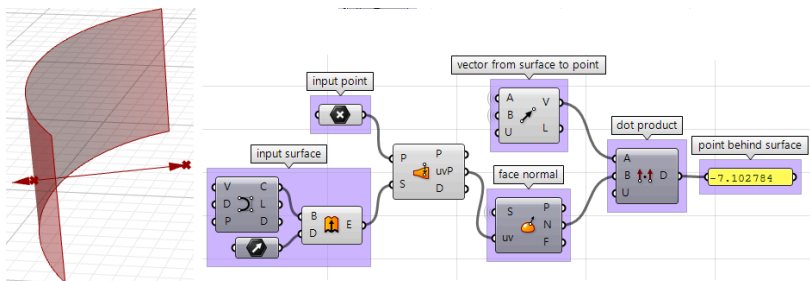
1. 使用 **pull** 组件找到曲面上输入点的最近点, 这将找到最近点的 **uv** 坐标, 之后我们可以使用这个坐标来确定曲面的法线方向。



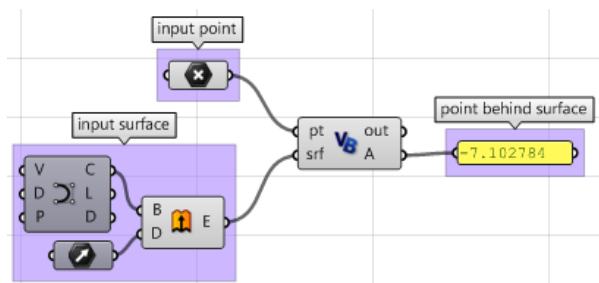
2. 利用最近点绘制指向输入点的向量，如图：



3. 现在我们可以利用点积来比较两个向量，如果结果为正，点位于曲面前，结果为负，点位于曲面后。



上述步骤也可以使用其他语言解决，如使用 GH 的 VB 组件：



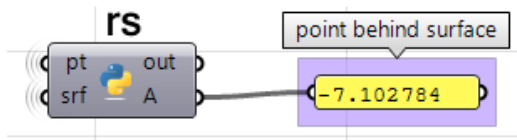
```
Private Sub RunScript(ByVal pt As
1 Point3d, ByVal srf As Surface, ByRef
A As Object)
2
3 ' 声明变量
4 Dim u, v As Double
5 Dim closest_pt As Point3d
6
7 ' 获取最近点uv
8 srf.ClosestPoint(pt, u, v)
9
10 ' 获取最近点
11 closest_pt = srf.PointAt(u, v)
12
13 ' 计算从最近点到输入点的方向
14 Dim dir As New Vector3d(pt - closest_pt)
15
16 ' 计算曲面法向
17 Dim normal = srf.NormalAt(u, v)
18
19 ' 计算两向量点积
```

```

20   A = dir * normal
21 End Sub

```

使用 GH 的 Python 组件（利用 RhinoScript 库）：

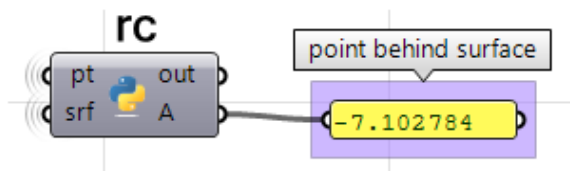


```

1  import rhinoscriptsyntax as rs #导入
   RhinoScript库
2
3  #找最近点uv
4  u, v = rs.SurfaceClosestPoint(srf, pt)
5
6  #获得最近点
7  closest_pt = rs.EvaluateSurface(srf, u, v)
8
9  #计算最近点到输入点向量
10 dir = rs.PointCoordinates(pt) - closest_pt
11
12 #计算曲面法向
13 normal = rs.SurfaceNormal(srf, [u, v])
14
15 #计算两向量点积
16 A = dir * normal

```

使用 GH 的 Python 组件（只使用 rhinocommon）：

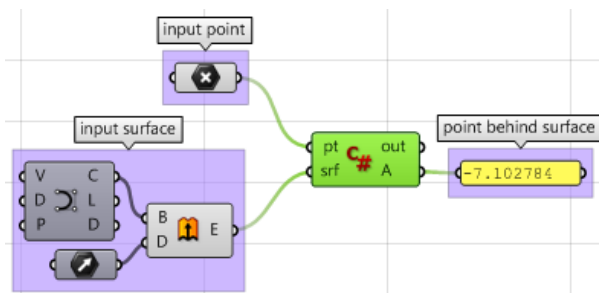


```

1  #找最近点
2  found, u, v = srf.ClosestPoint(pt)
3
4  if found:
5
6      #获取最近点
7      closest_pt = srf.PointAt(u, v)
8
9      #计算输入点到最近点向量
10     dir = pt - closest_pt
11
12     #计算曲面法向
13     normal = srf.NormalAt(u, v)
14
15     #计算两向量点积
16     A = dir * normal

```

使用 GH 的 C# 组件:



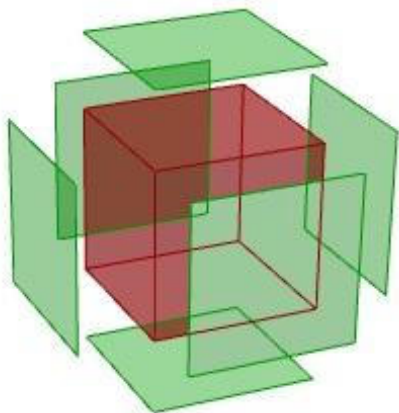
```

1 private void RunScript(Point3d pt,
2   Surface srf, ref object A)
3   {
4       // 声明变量
5       double u, v;
6       Point3d closest_pt;
7
8       // 获取最近点uv
9       srf.ClosestPoint(pt, out u, out v);
10
11      // 获取最近点
12      closest_pt = srf.PointAt(u, v);
13
14      // 计算输入点和最近点间向量
15      Vector3d dir = pt - closest_pt;
16
17      // 计算曲面法向
18      Vector3d normal = srf.NormalAt(u, v);
19
20      // 计算两向量点积
21      A = dir * normal;
22  }

```


1.5.2. 分解方盒

下面的案例演示如何分解多重曲面。如图是最终分解的立方体的样子：



输入：

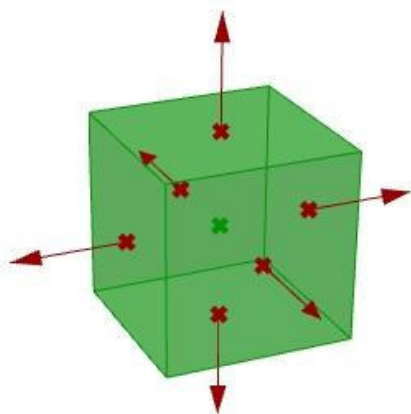
我们在 GH 中使用 **box** 组件输入：



参数：

思考我们需要知道的所有参数，以便解决本教程的问题

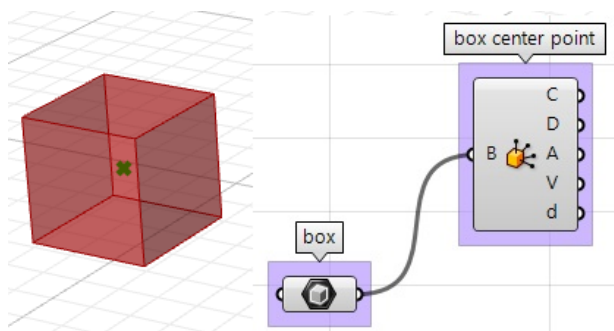
- 分解的中心
- 需要分解的立方体面
- 每个面移动的方向



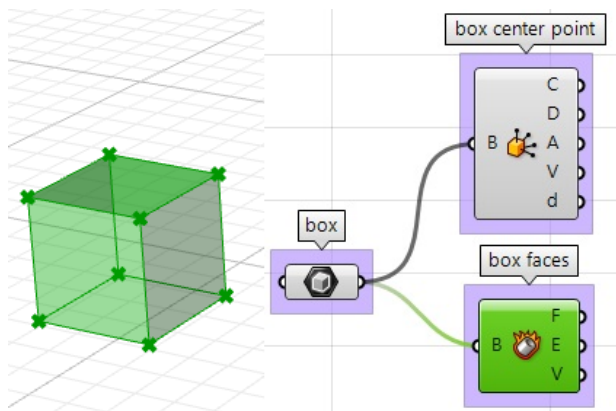
当我们确定了需要的参数，就要通过拼凑逻辑步骤得到结果，并且整合到解决方案中。

解决方案：

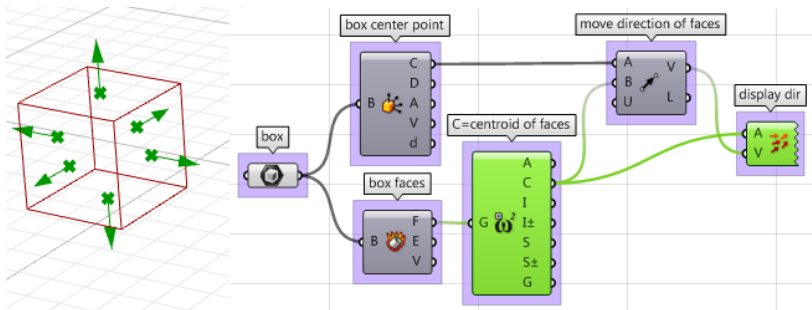
1. 使用 **Box Properties** 组件找到立方体的中心点：



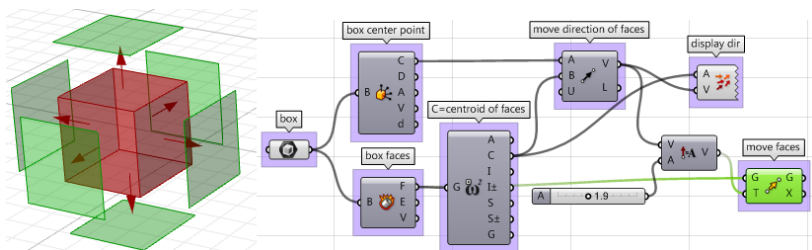
2. 使用 **Deconstruct Brep** 组件分解立方体的面：



3. 一个麻烦的部分是确定移动面的方向。我们需要先找到每个面的中心，然后定义从立方体中心岛每个面中心的方向，如图：

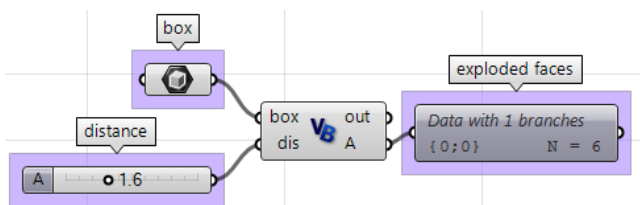


4. 一旦我们编写了所有参数的组件,我们就可以使用 **move** 组件以适当的方向移动面, 只要确保将移动向量设置为需要的长度:



上述步骤也可以使用 VB、C#、Python 语言解决。下面是解决方案。

使用 GH 的 VB 组件：



```
Private Sub RunScript(ByVal box As
1 Brep, ByVal dis As Double, ByRef A
As Object)
2
3 ' 获得实体的中心点
4 Dim area As
Rhino.Geometry.AreaMassProperties
5 area = Rhino.Geometry.AreaMassProperties.Compu
6
7 Dim box_center As Point3d
8 box_center = area.Centroid
9
10 ' 获得面列表
```

```

11      Dim faces As
      Rhino.Geometry.Collections.BrepFaceList =
      box.Faces
12
13      '声明变量
14      Dim center As Point3d
15      Dim dir As Vector3d
16      Dim exploded_faces As New List( Of
      Rhino.Geometry.Brep )
17      Dim i As Int32
18      '遍历所有面
19
20      For i = 0 To faces.Count() - 1
21          '提取每个面
22          Dim extracted_face As Rhino.Geometry.Brep
          = box.Faces.ExtractFace(i)
23
24          '获得每个面的中心
25          area = Rhino.Geometry.AreaMassProperties.Comp
26          center = area.Centroid
27
28          '计算每个面移动的方向(从立方体中心到面中
          心)
29          dir = center - box_center
30          dir.Unitize()
31          dir *= dis
32
33          '移动提取的面
34          extracted_face.Transform(Transform.Translati
35

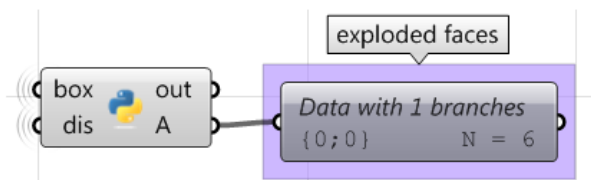
```

```

36         '加入面列表
37         exploded_faces.Add(extracted_face)
38     Next
39
40     '制定输出的面列表
41     A = exploded_faces
42 End Sub

```

使用 GH 的 Python 组件（利用 RhinoCommon 库）：



```

1  import Rhino
2
3  #获得立方体中心点
4  area =
5  Rhino.Geometry.AreaMassProperties.Compute(box)
6  box_center = area.Centroid
7
8  #获得面列表
9  faces = box.Faces
10
11 #声明变量
12 exploded_faces = []
13
14 #遍历所有面
15 for i, face in enumerate(faces):

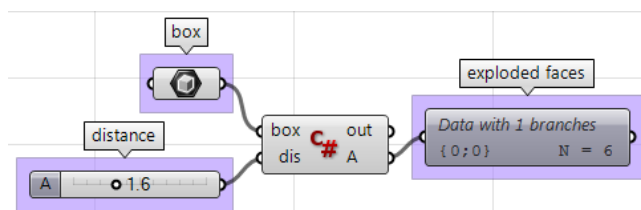
```

```

15
16 #或则一个面的复制
17 extracted_face = faces.ExtractFace(i)
18
19 #获得每个面的中心
20 area = Rhino.Geometry.AreaMassProperties.Compute(
21 center = area.Centroid
22
23 #计算每个面移动的方向(从立方体中心到面中心)
24 dir = center - box_center
25 dir.Unitize()
26 dir *= dis
27
28 #移动提取的面
29 move
    = Rhino.Geometry.Transform.Translation(dir)
30 extracted_face.Transform(move)
31
32 #加入到面列表
33 exploded_faces.append(extracted_face)
34
35 #指定到要输出的面列表
36 A = exploded_faces

```

使用 GH 的 C# 组件：



```

1 private void RunScript(Brep box, double
   dis, ref object A)
2 {
3
4     // 获得实体中心
5     Rhino.Geometry.AreaMassProperties area =
6     Point3d box_center = area.Centroid;
7
8     // 获得面列表
9     Rhino.Geometry.Collections.BrepFaceList faces
    = box.Faces;
10
11    // 声明变量
12    Point3d center;    Vector3d dir;
13    List<Rhino.Geometry.Brep> exploded_faces =
    new List<Rhino.Geometry.Brep>();
14
15    // 遍历所有面
16    for( int i = 0; i < faces.Count(); i++ )
17    {
18        // 提取每个面
19        Rhino.Geometry.Brep extracted_face =
        box.Faces.ExtractFace(i);
20
21        // 获得每个面的中心
22        area = Rhino.Geometry.AreaMassProperties.Compu
23        center = area.Centroid;
24
25        // 计算每个面移动的方向(从立方体中心到面
        中心)

```



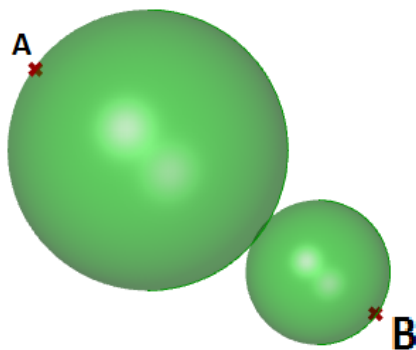
```

26     dir = center - box_center;
27     dir.Unitize();
28     dir *= dis;
29
30     // 移动提取的面
31     extracted_face.Transform(Transform.Translation)
32
33     // 加入到面列表
34     exploded_faces.Add(extracted_face);
35 }
36
37 // 指定到要输出的面列表
38 A = exploded_faces;
39 }

```

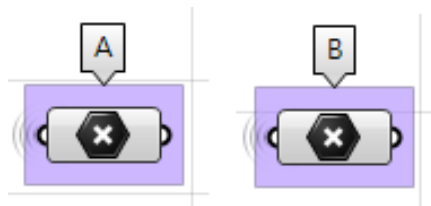
1.5.3. 相切球体

本实例将演示如何在两个输入点之间创建两个相切球体, 结果如图:



输入:

两个点A和B在三维坐标系。



参数:

以下是解决问题需要的参数，两个球体间的切点D，由一个在(0, 1)的开区间内的参数t，表示A和B之间的点。

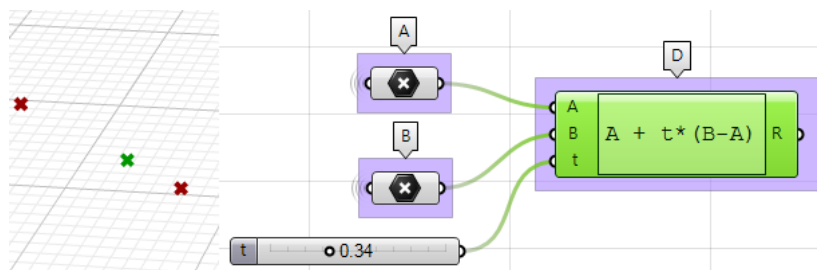
- 第一个球体的中心既AD的中点 C_1
- 第二个球体的中心既BD的中点 C_2
- 第一个球体的半径 r_1 既 AC_1 的长度
- 第二个球体的半径 r_2 既 BC_2 的长度

解决方案:

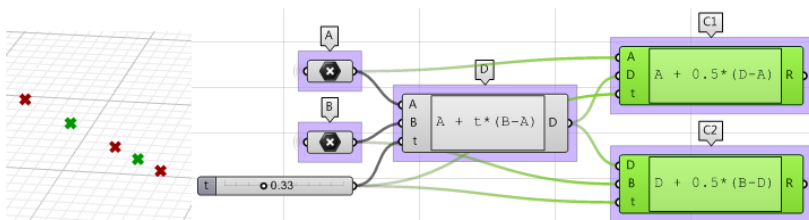
1. 使用 **Expression** 组件在参数t的定义下定义AB之间点D，我们使用的表达式基于直线的向量方程：

$$D = A + t \cdot (B - A)$$

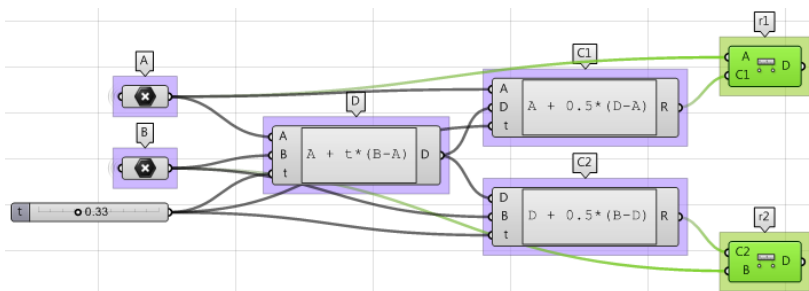
获得AB之间的点，我们的表达式就是： $A + t \cdot (B - A)$ ；



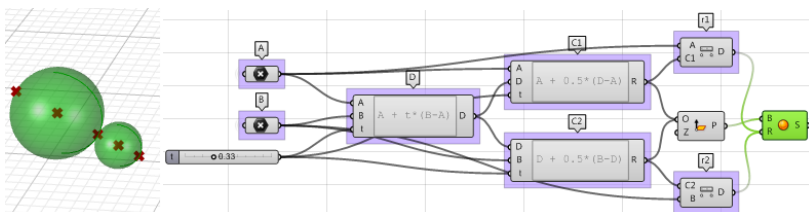
2. 使用 **Expression** 组件利用相同的表达式得到 C_1 和 C_2 ;



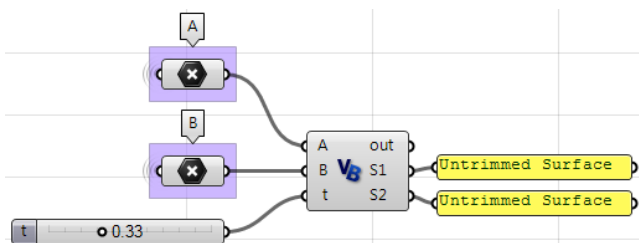
3. 使用 **Distance** 组件得到两个球体的半径 r_1 和 r_2 ;



4. 最后一步我们从 **Plane** 和半径创建球体。



使用 GH 的 VB 组件:



```

Private Sub RunScript(ByVal A As
1 Point3d, ByVal B As Point3d, ByVal t
As Double, ByRef S1 As Object, ByRef
2 S2 As Object)
3
4 ' 声明变量
5 Dim D, C1, C2 As Rhino.Geometry.Point3d
6 Dim r1, r2 As Double
7
8 ' 获得AB间的点
9 D = A + t * (B - A)
10
11 ' 获得AD中点
12 C1 = A + 0.5 * (D - A)
13
14 ' 获得DB中点
15 C2 = D + 0.5 * (B - D)
16
17 ' 获得球体半径
18 r1 = A.DistanceTo(C1)
19 r2 = B.DistanceTo(C2)
20
21 ' 创建球体并输出
22 S1 = New Rhino.Geometry.Sphere(C1, r1)

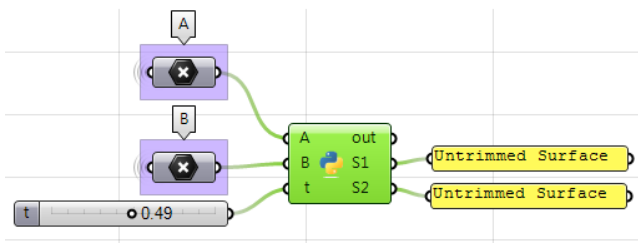
```

```

21     S2 = New Rhino.Geometry.Sphere(C2, r2)
22
23 End Sub

```

使用 GH 的 Python 组件：

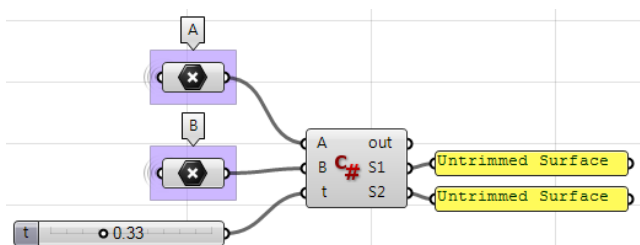


```

1  import Rhino
2
3  #find a point between A and B
4  D = A + t * (B - A)
5
6  #find mid point between A and D
7  C1 = A + 0.5 * (D - A)
8
9  #find mid point between D and B
10 C2 = D + 0.5 * (B - D)
11
12 #find spheres radius
13 r1 = A.DistanceTo(C1)
14 r2 = B.DistanceTo(C2)
15
16 #create spheres and assign to output
17 S1 = Rhino.Geometry.Sphere(C1, r1)
18 S2 = Rhino.Geometry.Sphere(C2, r2)

```

使用 GH 的 C# 组件:



```
private void RunScript(Point3d A, Point3d
1 B, double t, ref object S1, ref object
   S2)
2 {
3     //declare variables
4     Rhino.Geometry.Point3d D, C1, C2;
5     double r1, r2;
6
7     //find a point between A and B
8     D = A + t * (B - A);
9
10    //find mid point between A and D
11    C1 = A + 0.5 * (D - A);
12
13    //find mid point between D and B
14    C2 = D + 0.5 * (B - D);
15
16    //find spheres radius
17    r1 = A.DistanceTo(C1);
18    r2 = B.DistanceTo(C2);
19
20    //create spheres and assign to output
```

C#

```
21    S1 = new Rhino.Geometry.Sphere(C1, r1);  
22    S2 = new Rhino.Geometry.Sphere(C2, r2);  
23 }
```

2. 矩阵和变换

变换是指移动（也称为平移）、旋转和缩放对象等操作。它们使用矩阵存储在 3D 程序中，矩阵仅是数字组成的矩形数组。使用矩阵可以非常快速地执行多个转换。事实上，[4x4] 矩阵可以表示所有变换。为所有转换提供统一的矩阵维度可以节省计算时间。⁴

<i>matrix</i>	c1	c2	c3	c4
<i>row</i> (1)	+	+	+	+
<i>row</i> (2)	+	+	+	+
<i>row</i> (3)	+	+	+	+
<i>row</i> (4)	+	+	+	+

2.1. 矩阵运算

在计算机图形学中，与之最有关的矩阵运算就是矩阵乘法，我们会对矩阵乘法做详细解释。

2.1.1. 矩阵乘法

矩阵乘法用于将变换应用于几何图形。例如我们有一个点，我们令他绕某个轴旋转，我们用一个旋转矩阵乘该点，得到新的旋转后的位置。

⁴译者注：显然这些变换不包含扭转等复杂变换，矩阵维度指矩阵的行数和列数。

$$\begin{array}{c} \text{旋转矩阵} \\ \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \cdot \begin{array}{c} \text{输入点} \\ \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \end{array} = \begin{array}{c} \text{输出点} \\ \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} \end{array}$$

大多时候，我们需要对同一个几何物体进行多次变换，例如我们需要移动和旋转 1000 个点，我们可以用以下方法之一。

方法 1

1. 将移动矩阵乘 1000 个点来移动这些点。
2. 将旋转矩阵乘产生的 1000 个点来旋转上一步移动的点。⁵

操作次数 = **2000**。

方法 2

1. 将旋转矩阵乘移动矩阵得到组合变换矩阵。
2. 将组合矩阵乘 1000 个点在一步之内得到移动和旋转后的点。

操作次数 = **1001**。

注意，方法一需要几乎两次的操作数才能得到相同的结果。虽然方法 2 非常有效，但当且仅当移动和旋转矩阵都是 $[4 \times 4]$ 矩阵时，运算才成为可能，这就是为什么在计算机图形学中使用 $[4 \times 4]$ 矩阵来表示所有的变换，并且用 $[4 \times 1]$ 的矩阵表示点。

⁵译者注：对于矩阵乘法来说，交换律是失效的，也就是 $A \times B$ 和 $B \times A$ 并不表示相同的意义，所以翻译过程中的乘法顺序会尽量和书写顺序保持一致，但仍需读者时时注意乘法顺序，防止混淆和错误。

三维建模程序提供了工具来进行变换和矩阵乘法, 但如果你对如何使用数学方法对矩阵进行乘法感到好奇, 我们提供了一个简单示例, 为了令两矩阵相乘, 他们必须有匹配的维数. 这也意味着第一个矩阵的列数必须等于第二个矩阵的行数. 相乘的结果矩阵的行数和列数分别为第一个矩阵的行数和第二个矩阵的列数. 例如我们有两个矩阵 M 和 P , 维数分别为 $[4 \times 4]$ 和 $[4 \times 1]$, 得到的矩阵 $M \cdot P$ 结果的维数为 $[4 \times 1]$ 如下所示:

$$\begin{matrix} & M & & P \\ \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} & \cdot & \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} & = & \begin{matrix} P \\ \begin{bmatrix} x' = a \cdot x + b \cdot y + c \cdot z + d \cdot 1 \\ y' = e \cdot x + f \cdot y + g \cdot z + h \cdot 1 \\ z' = i \cdot x + j \cdot y + k \cdot z + l \cdot 1 \\ 1 = 0 \cdot x + 0 \cdot y + 0 \cdot z + 1 \cdot 1 \end{bmatrix} \end{matrix} \end{matrix}$$

2.1.2. 单位矩阵

单位矩阵是一个特殊的矩阵, 除了对角线上的分量为 1 外, 所有的分量都为 0。

1.0	0.0	0.0	0.0
0.0	1.0	0.0	0.0
0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0

单位矩阵的主要性质是单位矩阵乘任意矩阵或 0 的结果依然是该矩阵或 0。

$$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \times \begin{bmatrix} 2.0 \\ 3.0 \\ 1.0 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 1.0 \times 2.0 + 0.0 \times 3.0 + 0.0 \times 1.0 + 0.0 \times 1.0 \\ 0.0 \times 2.0 + 1.0 \times 3.0 + 0.0 \times 1.0 + 0.0 \times 1.0 \\ 0.0 \times 2.0 + 0.0 \times 3.0 + 1.0 \times 1.0 + 0.0 \times 1.0 \\ 0.0 \times 2.0 + 0.0 \times 3.0 + 0.0 \times 1.0 + 1.0 \times 1.0 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 3.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

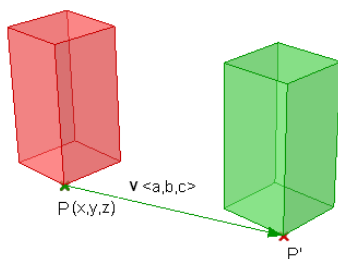
2.2. 变换操作

大多数变换都保留了几何物件各部分之间的平行关系。例如，同一直线上点在变换后仍保持共线。此外，一个平面上的点在变换后保持共面。这种类型的变换称为**仿射变换**。

2.2.1. 移动（平移）变换

将点从起始位置移动某个向量的计算如下：

$$P' = P + \vec{v}$$



假设：

- $P(x, y, z)$ 是一个给定的点
- $\vec{v} = \langle a, b, c \rangle$ 是一个变换向量

那么我们有：

$$P'(x) = x + a$$

$$P'(y) = y + b$$

$$P'(z) = z + c$$

点以 $[4 \times 1]$ 矩阵格式表示, 最后一行插入 1。例如, 点 $P(x, y, z)$ 表示如下:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

使用 $[4 \times 4]$ 矩阵进行变换 (这被称作齐次坐标系下的变换), 而不是 $[3 \times 3]$ 矩阵, 可以表示包括移动在内的所有变换。平移矩阵的一般格式为:

$$\begin{bmatrix} 1 & 0 & 0 & a_1 \\ 0 & 1 & 0 & a_2 \\ 0 & 0 & 1 & a_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

例如, 为了使用向量 $\vec{v}\langle 2, 2, 2 \rangle$ 移动点 $P(2, 3, 1)$, 得到的点位置为:

$$P' = P + \vec{v} = (2 + 2, 3 + 2, 1 + 2) = (4, 5, 3)$$

如果我们使用矩阵形式, 并将平移矩阵输入点, 我们得到的点位置如下所示:

$$\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} (1 \cdot 2 + 0 \cdot 3 + 0 \cdot 1 + 2 \cdot 1) \\ (0 \cdot 2 + 1 \cdot 3 + 0 \cdot 1 + 2 \cdot 1) \\ (0 \cdot 2 + 0 \cdot 3 + 1 \cdot 1 + 2 \cdot 1) \\ (0 \cdot 2 + 0 \cdot 3 + 0 \cdot 1 + 1 \cdot 1) \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \\ 3 \\ 1 \end{bmatrix}$$

类似的，任意几何物件都是通过构造平移矩阵乘点来平移的。例如我们有一个八个角点定义的立方体，我们想在 x 方向移动 4 个单位， y 方向移动 5 单位， z 方向移动 3 单位，我们必须用平移矩阵乘所有八个角点得到新的角点以此得到新的平移后的立方体。

$$\begin{bmatrix} 1 & 0 & 0 & 4 \\ 0 & 1 & 0 & 5 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

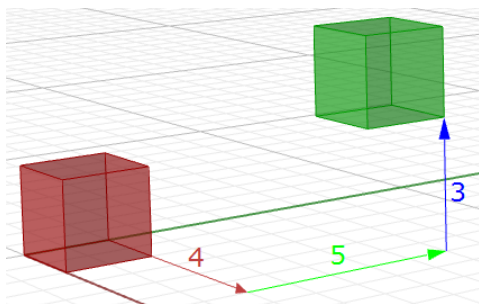
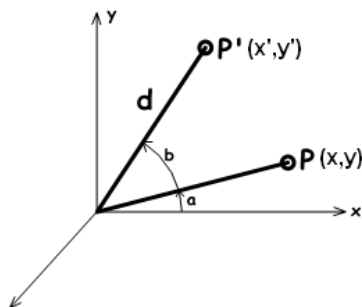


图 19 平移所有方框角点

2.2.2. 旋转变换

本节介绍如何使用三角函数计算绕 z 轴和原点的旋转，然后推导旋转变换的一般矩阵格式。



在 xy 平面取一点 $P(x, y)$ 并且旋转一个角度 b 。我们由图易得：

$$x = d \cdot \cos(a) \quad (1)$$

$$y = d \cdot \sin(a) \quad (2)$$

$$x' = d \cdot \cos(b + a) \quad (3)$$

$$y' = d \cdot \sin(b + a) \quad (4)$$

用三角恒等式(和角公式)展开 x' 和 y' 得到角的正弦和余弦值:

$$x' = d \cdot \cos(a) \cos(b) - d \cdot \sin(a) \sin(b) \quad (5)$$

$$y' = d \cdot \cos(a) \sin(b) + d \cdot \sin(a) \cos(b) \quad (6)$$

由(1)和(2)得:

$$x' = x \cdot \cos(b) - y \cdot \sin(b)$$

$$y' = x \cdot \sin(b) + y \cdot \cos(b)$$

沿世界坐标 **z** 轴旋转 b 弧度的矩阵如下:

$$\begin{bmatrix} \cos b & -\sin b & 0 & 0 \\ \sin b & \cos b & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

沿世界坐标 **x** 轴旋转 b 弧度的矩阵如下:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos b & -\sin b & 0 \\ 0 & \sin b & \cos b & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

沿世界坐标 **y** 轴旋转 b 弧度的矩阵如下:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos b & -\sin b & 0 \\ 0 & \sin b & \cos b & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

已一个立方体为例, 我们将其旋转 30 度, 我们需要做如下操作:

1. 构造一个 30 度的旋转矩阵, 使用沿 Z 轴旋转的通用形式, 旋转矩阵如下:

$$\begin{bmatrix} 0.87 & -0.5 & 0 & 0 \\ 0.5 & 0.87 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. 将旋转矩阵乘输入的几何物件, 如果针对立方体, 乘每个角点, 找到立方体的新位置。

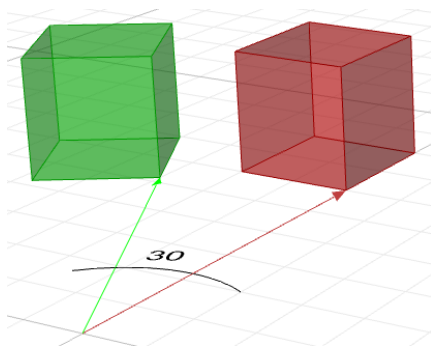


图 20 旋转几何

2.2.3. 缩放变换

为了缩放几何物件, 我们需要一个比例因子和一个中心。比例因子可以在 x、y 和 z 方向上均匀缩放, 也可以在每个维度上是独立的。

缩放点可以使用以下公式:

$$P' = \text{ScaleFactor}(S) \cdot P$$

或:

$$P'.x = S_x \cdot P.x$$

$$P'.y = S_y \cdot P.y$$

$$P'.z = S_z \cdot P.z$$

这是缩放变换的矩阵形式，假设缩放中心是世界坐标原点 $(0,0,0)$ 。

$$\begin{bmatrix} \text{Scale} - x & 0 & 0 & 0 \\ 0 & \text{Scale} - y & 0 & 0 \\ 0 & 0 & \text{Scale} - z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

例如我们为了将一个立方体相对世界坐标原点缩放 0.25，那么缩放矩阵如下：

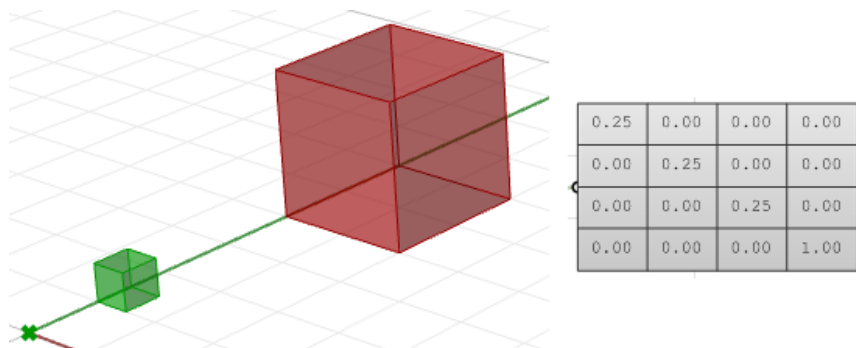
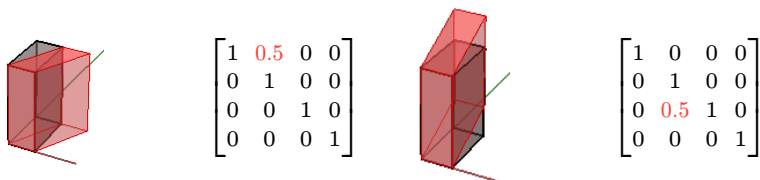


图 21 缩放几何

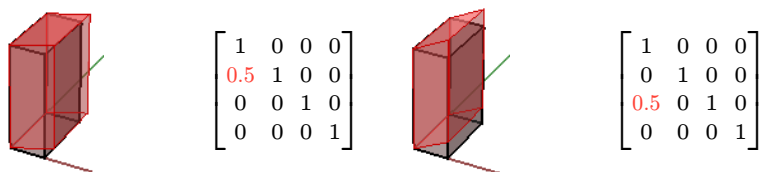
2.2.4. 剪切变换

三维的剪切是沿着某一个轴不变另外垂直的另外一个轴方向变化的变换。例如，沿 z 轴的剪切变换不会改变 z 轴的尺寸，但会改变 x 和 y 轴方向的尺寸。下面是一些例子：

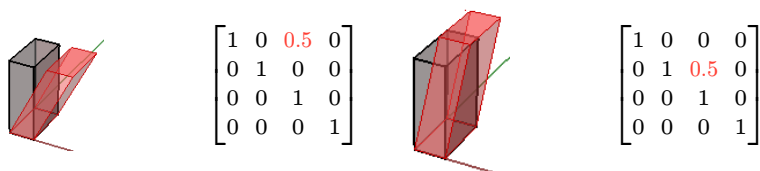
1. 保持 y 轴方向不变， x 、 z 方向剪切



2. 保持 x 轴方向不变, y 、 z 方向剪切:



3. 保持 z 轴方向不变, x 、 y 方向剪切:



2.2.5. 镜像或翻转变换

镜像变换在直线或平面上创建对象的镜像。二维物体是通过一条直线镜像的, 而三维物体是通过一个平面镜像的。请记住, 镜像变换翻转了几何面的法线方向。

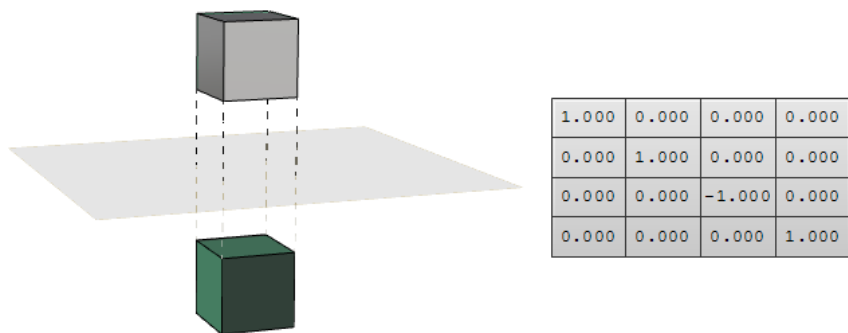


图 22 使用世界坐标 xy 平面镜像变换的矩阵，面法向被翻转

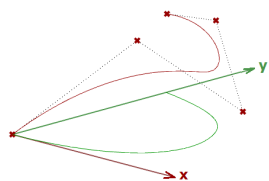
2.2.6. 平面投影变换

直观说，一个空间点 $P(x, y, z)$ 在 xy 平面上的投影点 $P_{xy}(x, y, 0)$ 只需设置 z 为 0 即可。同理 P 在 xz 平面的投影点是 $P_{xz}(x, 0, z)$ 。当投影到 yz 平面时， $P_{yz} = (0, y, z)$ 。我们称这些为正交投影。

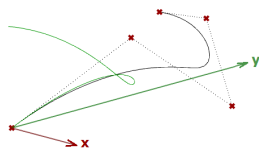
如果我们有一条曲线作为输入，我们应用平面投影变换，将得到一条曲线投影到那个平面上。下面是一个用矩阵格式投影到 xy 平面的曲线的例子。

注：NURBS 曲线（下一章解释）使用控制点来定义曲线。投射一条曲线等于投射它的控制点。⁶

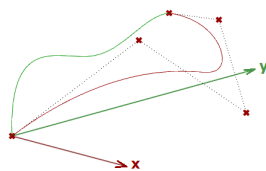
⁶译者注：有关 Nurbs 投影不变性的证明的扩展阐述可参考如下链接：<https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/NURBS/NURBS-property.html>



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. 参数曲线和曲面

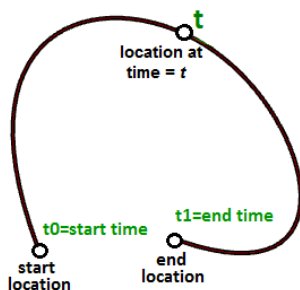
假设您每个工作日都从家里到工作地点。您早上 8:00 出发, 9:00 到达。在 8:00 到 9:00 之间的每个时间点, 您都会在沿途的某个位置。如果您在旅途中每分钟绘制一次位置, 则可以通过连接绘制的 60 个点来定义家庭和工作之间的路径。假设您每天以完全相同的速度行驶, 则在 8:00 您将在家 (起始位置), 在 9:00 您将在工作 (结束位置), 在 8:40 您将在与第 40 个绘图点完全相同的路径上的位置。恭喜, 您刚刚定义了第一条参数曲线! 您已使用时间作为参数来定义路径, 因此您可以将路径曲线称为参数化曲线。您从开始到结束 (8 到 9) 花费的时间间隔称为曲线区间或曲线域。

我们可以用一些参数如 t 来描述参数曲线上点的位置的 x 、 y 、 z , 如下所示:

$$x = x(t)$$

$$y = y(t)$$

$$z = z(t)$$



其中: t 是一个区间内的实数

我们之前学习过, 直线的参数方程使用 t 定义如下:

$$x = x' + t \cdot a$$

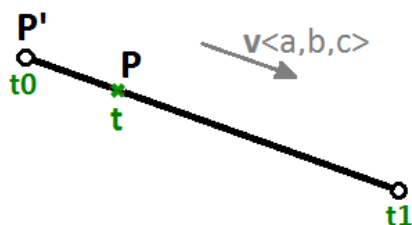
$$y = y' + t \cdot b$$

$$z = z' + t \cdot c$$

其中:

x 、 y 、 z 是自变量为区间内实数 t 的函数, x' 、 y' 、 z' 是线段上某一点的坐标, 因为 $\vec{v}\langle a, b, c \rangle$ 平行与直线, a 、 b 、 c 实际上定义了直线的方向,

由此我们可以使用一个介于实数 t_0 和 t_1 之间的参数 t 和一个沿线段方向的单位向量 \vec{v} 来描述线段的参数方程:



$$P = P' + t \cdot \vec{v}$$

另一个例子是圆。xy 平面上圆的参数方程: 中心位于原点 $(0,0)$, 角度参数 t 范围在 0 和 2π 弧度之间。

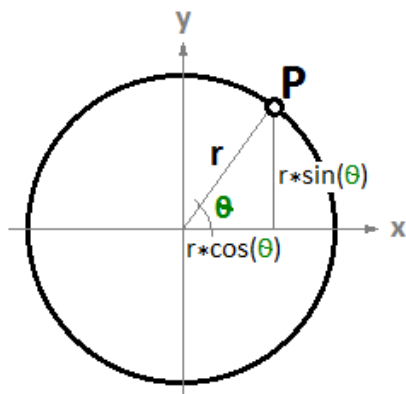
$$x = r \cdot \cos(t)$$

$$y = r \cdot \sin(t)$$

我们可以得到圆的参数方程的一般形式为:

$$x/r = \cos(t)$$

$$y/r = \sin(t)$$



由于:

$$\cos(t)^2 + \sin(t)^2 = 1 \quad (\text{毕达哥拉斯恒等式})$$

我们有:

$$\left(\frac{x}{r}\right)^2 + \left(\frac{y}{r}\right)^2 = 1, \text{ or } x^2 + y^2 = r^2$$

故该参数方程表示一个圆。

3.1. 曲线参数

曲线上的参数表示该曲线上某个点的地址。如前所述，您可以将参数曲线视为在一定时间内以固定或可变速度在两点之间行进的路径。如果行进需要 T 时间，则参数 t 表示曲线上的位置对应的 T 时间段内的时间。

如果在 A 、 B 两点之间有一直线路径（线段），且 \vec{v} 是从 A 到 B 的向量（ $\vec{v} = B - A$ ），则可以用参数方程来表示 A 、 B 两点之间的所有点 M ，如下所示：

$$M = A + t \cdot (B - A)$$

其中：

t 是一个介于 0 和 1 之间的实数。

在这种情况下， t 值的范围（0 到 1）被称为曲线区间或曲线域。如果 t 是区间外的值（小于 0 或大于 1），则结果点 M 将位于线段 \overline{AB} 之外。

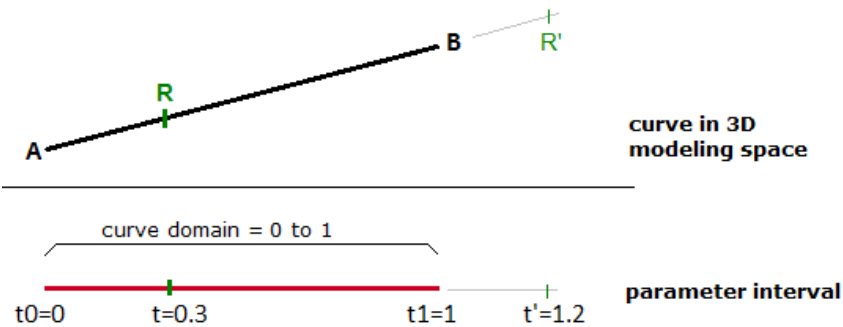


图 23 三维空间和参数区间中的线段的参数曲线

同样的原理适用于任意的参数曲线。曲线上的任意一点都可以在定义曲线的参数区间内使用 t 值进行计算。区间的起始位置的值通常称为 t_0 ，结束位置的值称为 t_1 。

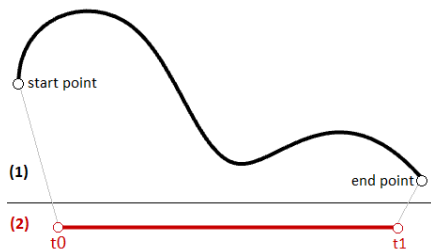


图 24 三维空间 (1) 和参数区间 (2) 中的曲线

3.1.1. 曲线区间

曲线区间的定义为计算该曲线内点的参数范围。区间通常用两个实数来描述区间限制，以 (min to max) 或 (min, max) 的形式表示。区间限制可以是任意两个值，这些值可能与曲线的实际长度相关，也可能与曲线的实际长度无关。在递增区间中，区间 min 参数的计算结果为曲线的起点，区间 max 的计算结果为曲线的终点。⁷

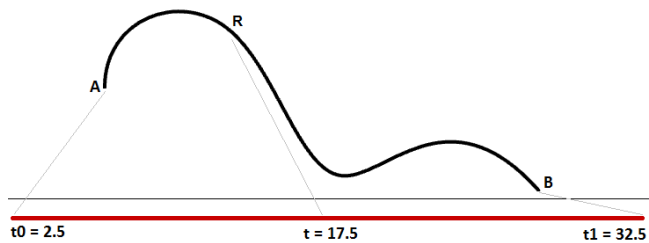


图 25 曲线区间介于任意两数之间

⁷作者在本章中大量的重复使用和混用区间和域 (domain and interval) 在意思相同的情况下，译者行文将统一使用区间一词。

更改曲线区间被称为曲线的重新参数化。比如，一个常见的操作是将曲线区间更改为(0 to 1)。曲线的重新参数化不会影响到曲线形状。就如通过跑步代替走路改变路径上的时间并不会改变路径形状本身一样。

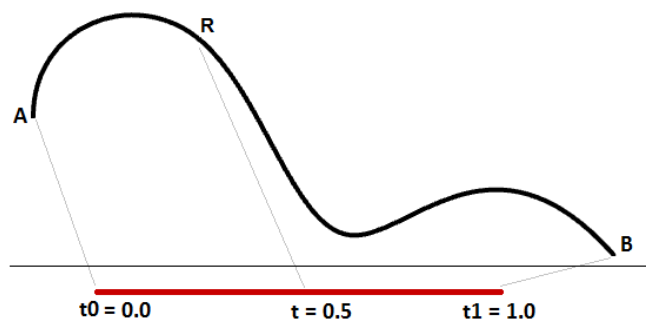


图 26 归一化曲线区间到 0 到 1

递增区间意味着区间的最小值指向曲线起点，区间通常是递增的，但也不绝对。

3.1.2. 曲线计算

我们能知道到曲线区间是三维曲线内所有点对应参数值的范围，但是区间中值并不能保证计算得到曲线的中点。如图 28 所示。

我们可以将曲线的均匀参数化视为以恒定速度行进的路径。两点连成的一阶直线就是一个示例，其中相等的参数间隔转换为曲线上相等的长度间隔。这是一种特殊情况，其中相等的参数间隔在三维曲线上计算为相等的间隔。

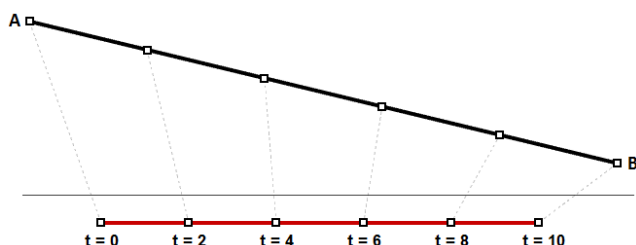


图 27 一阶直线相等的参数间隔计算为相等的曲线长度

但速度更可能随着路径变化。假设走一条路需要 30 分钟，那么很难恰巧在第 15 分钟走完路程的一半。图 30 展示了这种情况，相等的参数间隔在三维曲线上有不一致的长度。

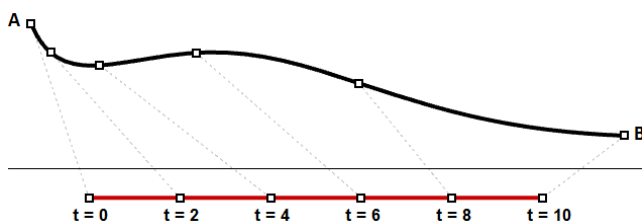


图 28 相等的参数间隔通常不会转换为曲线上的相等距离

你可能需要计算三维曲线上根据曲线长度定义比例的点，如你可能需要将曲线划分为等长的曲线段，一般来说建模软件会提供根据曲线长度计算曲线上点的工具。

3.1.3. 曲线的切向量

切向量是曲线在任何参数（或曲线上的点）处于在该点接触曲线但不相交的向量。切向量的斜率方向等价于同一点处曲线的斜率方向。下面是曲线在两个参数上该点的切线。⁸

⁸译者注：在日常和学术交流中，切向量和切线经常混用，严格意义上切向量仅是曲线上该点的方向，但是切线还隐含了该向量从曲线上当前点出发的含义。在本书中两者概念并不做严格区分。

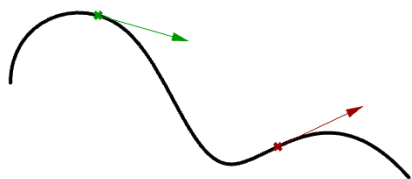


图 29 曲线的切线

3.1.4. 三次多项式曲线

Hermite 和 Bezier 曲线是由四个参数确定的三次多项式曲线的两个示例。Hermite 曲线由的两个端点和两个切向量确定，而 Bezier 曲线由四个点定义。虽然它们在数学上有所不同，但它们具有相似的特征和局限性。

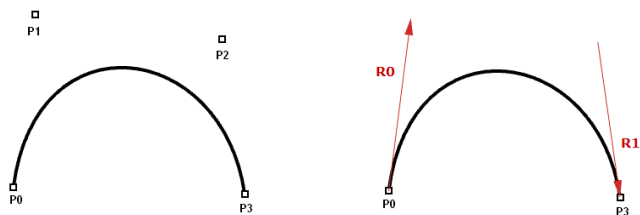


图 30 三次多项式曲线: Bezier 曲线 (左) 和 Hermite 曲线 (右)

大多数情况下，曲线由多个曲线段组成，这就要求我们制作分段的三次曲线。下面是分段 Bezier 曲线的图示，该曲线使用七个点创建了两端三次曲线。请注意，曲线仅是连接到一起，并不平滑连续。

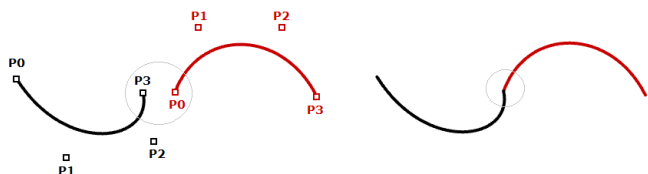


图 31 两个 Bezier 曲线段共用端点

尽管 Hermite 曲线使用同 Bezier 曲线相同数量的参数 (四个参数定义一条曲线), 但它们提供了曲线切线的附加信息, 这些信息也可以与下一条曲线共享, 以创建更平滑的曲线, 同时减少总存储空间, 如下所示。

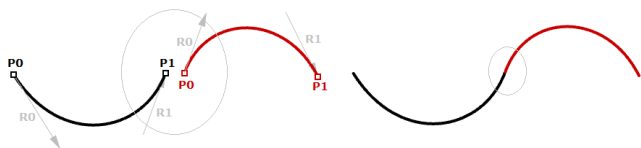


图 32 两个 Hermite 曲线段共用一个点和一个切线

非均匀有理 B 样条曲线 (NURBS) 是一种强大的曲线表示, 可保持更平滑、更连续的曲线。曲线段之间共享更多控制点, 以更少的存储空间实现更平滑的曲线。

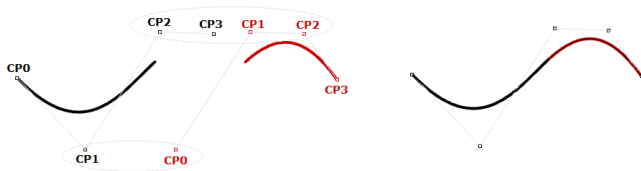


图 33 两个 3 次 NURBS 曲线段共享三个控制点。

NURBS 曲线和曲面是 Rhino 用来表示几何形状的主要数学表示。本章稍后将详细介绍 NURBS 曲线特性和组成。

3.1.1.5. 三阶 Bezier 曲线计算

de Casteljau 算法以其发明者 Paul de Casteljau 的名字命名, 使用递归方法计算贝塞尔曲线。算法步骤可以总结如下:

输入:

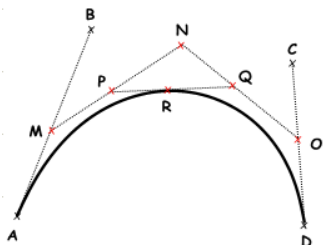
- 定义曲线的四个点 A, B, C, D 和曲线区间内的参数 t 。

输出:

- 和参数 t 对应的曲线上的点 R 。

解决方案：

1. 在线段 \overline{AB} 上找到 t 对应的点 M 。
2. 在线段 \overline{BC} 上找到 t 对应的点 N 。
3. 在线段 \overline{CD} 上找到 t 对应的点 O 。
4. 在线段 \overline{MN} 上找到 t 对应的点 P 。
5. 在线段 \overline{NO} 上找到 t 对应的点 Q 。
6. 在线段 \overline{PQ} 上找到 t 对应的点 R 。



3.2. NURBS 曲线

NURBS 是一种数学上精确表示曲线的方式，具有直观的可编辑性。使用 NURBS 表示自由曲线曲面很容易并且其控制结构使得编辑变得容易并可控。

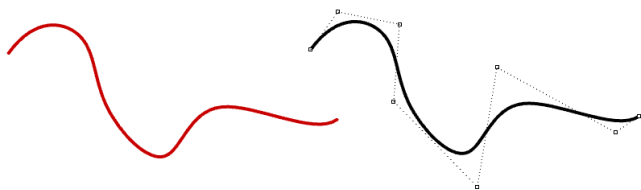


图 34 非均匀有理 B 样条及其控制结构

对使用 NURBS 建模软件的人来说，对 NURBS 的基础了解是必要的。NURBS 曲线有四个主要特性：阶数、控制点、节点列表和计算规则。对于有兴趣更深入研究 NURBS 的人来说，下附一些书籍和参考资料。

1. Wikipedia: De Boor's algorithm⁹

2. Michigan Tech, Department of Computer Science, De Boor's algorithm¹⁰

3.2.1. 阶数

曲线的阶数¹¹是正整数, Rhino 允许任何大于等于 1 阶的曲线。1、2、3 和 5 阶曲线是最常用的, 但是 4 阶和 5 阶以上的阶数在实际作业中并不常用, 下面列出一些常见曲线及度数:

直线和多段线是 1 阶的 NURBS 曲线。



圆和椭圆是 2 阶的 NURBS 曲线。



自由曲线通常表示为 3 阶或 5 阶曲线。



3.2.2. 控制点

NURBS 曲线的控制点是至少 (阶数+1) 个点的列表。改变 NURBS 曲线形状的最直观方法是移动其控制点。

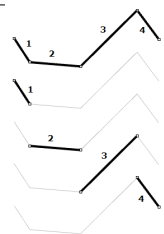
⁹http://en.wikipedia.org/wiki/De_Boor's_algorithm

¹⁰<http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/de-Boor.html>

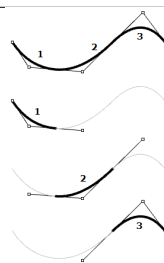
¹¹译者注: 曲线的阶数本质是曲线对应的有理多项式的次数。故在翻译时当形容多项式时会翻译成三次多项式曲线, 在仅涉及曲线时会翻译成三阶 Bezier 曲线。在部分中文资料里会把 Nurbs 曲线的 Order 翻译为阶数, Rhino 官方的资料里 Order 翻译为次数, 其值为 Degree+1。但又会和多项式的次数混淆。译者在这里不持立场, 统一只使用 Degree 作为阶数而尽量不出现次数的概念 (除非明确的涉及多项式的描述)。

影响 NURBS 曲线中每个子曲线段的控制点数由曲线的阶数定义。例如，阶数为 1 的曲线中的每个子曲线段仅受两个端控制点的影响。在 2 阶曲线中，每个子曲线段受三个控制点影响，依此类推。¹²

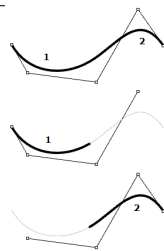
1 阶曲线的所有控制点都在曲线上。在 1 阶 NURBS 曲线中，两个（阶数+1）控制点定义一个子曲线段。使用五个控制点，曲线有四个子曲线段。



圆形和椭圆是二阶曲线的示例。在 2 阶 NURBS 曲线中，三个（阶数+1）控制点定义一个子曲线段。使用五个控制点，曲线有三个子曲线段。



3 阶曲线的控制点通常不接触曲线，但开放曲线的端点除外。在 3 阶 NURBS 曲线中，四个（阶数+1）控制点定义一个子曲线段。使用五个控制点，曲线有三个子曲线段。



3.2.3. 控制点权重

每个控制点都有一个关联的数字，称为权重。除少数例外，权重应为正数。当所有控制点的权重相同时，通常为 1，则该曲

¹²译者注，在大多数文献中，对 span 的翻译是跨度，但是相对不太直观，译者将其翻译为子曲线段。

线称为非有理曲线。¹³直观地说，您可以将权重视为每个控制点的引力。控制点的相对权重越高，曲线越接近该控制点。

值得注意的是，最好避免改变曲线权重。改变权重很少能给出期望的结果，同时它在求曲线交点等操作中引入了许多计算难题。使用有理曲线的唯一充分理由是表示无法以其他方式绘制的曲线，例如圆和椭圆。¹⁴

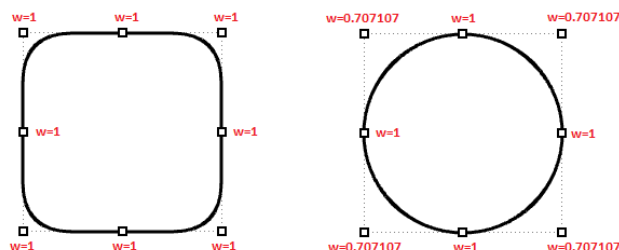


图 35 不同权重的控制点对结果曲线的影响。左曲线是非有理曲线，控制点权重一致。右边的圆圈是一条有理曲线，角控制点的权重小于 1。

3.2.4. 节点列表

每一条 NURBS 曲线都有一个与之关联的数字列表，我们称之为节点列表（有时也称为节点向量）。节点解释起来有些抽象，使用建模软件时，我们无需手动设置节点列表，下面附一些对了解节点列表有意义的内容。¹⁵

¹³译者注：非有理是指曲线的表达式是多项式而非有理式，不建议在使用中翻译为无理。

¹⁴译者注：90 度圆弧控制点中间点的权重正好为 $\sqrt{2}/2 \approx 0.707107$ 。圆弧控制点的中间点的权重和控制点连线的夹角有关，具体证明可自行深入了解。

¹⁵译者注：Knot 翻译为节点，Knots 一般指节点列表，但二者经常混用。节点本身只是一个节点列表中的数字，使用节点一词有轻微误导之嫌，但暂未找到更好的处理方式，故沿用。部分资料 Knot 翻译为结，把 Knot Vector 翻译为结点向量，本翻译版本和 Rhino 官方中文资料保持一致，不采用这种方式。

3.2.5. 节点是参数值

节点列表是位于曲线区间内的参数值的非递减列表。在 Rhino 中，节点列表长度比控制点个数多阶数-1。即节点列表长度等于控制点数量加上曲线阶数减 1。¹⁶

- $|\text{knots}| = |\text{CVs}| + \text{Degree} - 1$
- 节点列表长度=控制点数量+阶数-1

一般来说，对于非周期曲线，节点向量的第一个值等于曲线区间最小值，节点向量第二个值等于区间最大值。

例如一个具有七个控制点且曲线区间为 0 到 4 的开放的三阶 NURBS 曲线的节点列表类似于 $\langle 0, 0, 0, 1, 2, 3, 4, 4, 4 \rangle$

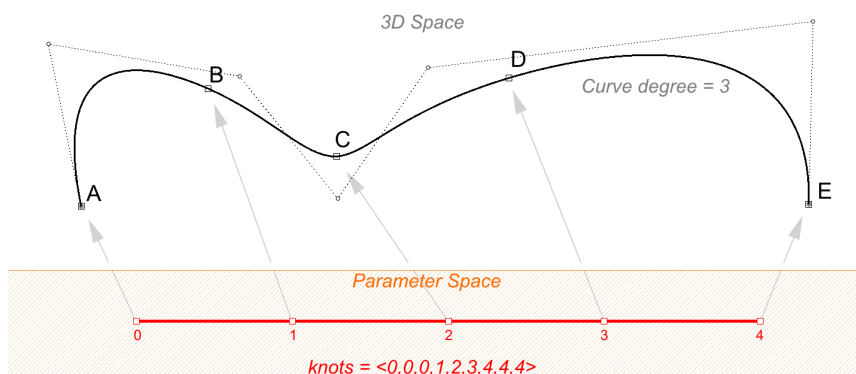


图 36 控制点、节点列表长度和阶数的关系

缩放节点列表中的值不会影响曲线形状，如果将曲线区间从 0-4 修改为 0-1，节点列表会缩放，但曲线形状不会改变。

¹⁶译者注：根据算法不同，不同软件对节点数量有不同的实现。具体参考：<https://www.rhino3d.com/features/nurbs/>

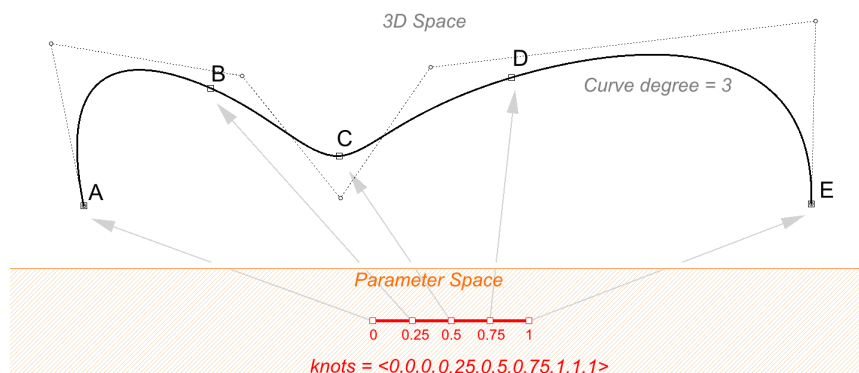


图 37 缩放节点列表后的曲线

3.2.6. 重复节点

节点的重复数量是指相同的值在节点列表中出现的次数，节点重复数量不能大于曲线阶数，重复数量可以控制节点对应参数位置曲线的连续性。

3.2.7. 完全重复节点

完全重复节点的重数等于曲线阶数，在该节点对应的参数值处，曲线和控制点重合。

例如，夹紧的开放曲线在曲线末端有完全重复节点。这正是末端控制点和曲线端点重合的原因，非端点处的重节点使曲线在对应参数位置控制点和曲线重合。¹⁷

下面两条线具有相同的控制点位置和数量，但有不同的节点列表和曲线形状。

¹⁷译者注：clamped curve 指用端点的重节点将曲线端点“夹”到控制点的曲线。虽然感觉不太恰当，但也没有更合适的翻译了。

阶数=3 控制点数量 =7 节点列表= $\langle 0, 0, 0, 1, 2, 3, 4, 4, 4 \rangle$ 节点列表长度=9 曲线区间 (0 到 4)	
阶数=3 控制点数量 =7 节点列表= $\langle 0, 0, 0, 1, 1, 1, 4, 4, 4 \rangle$ 节点列表长度=9 曲线区间 (0 到 4)	

)

中间的完全重复节点会产生组结, 曲线经过了关联的控制点。

3.2.8. 单一节点

单一节点是只在节点列表中只出现一次的节点, 一般出现在节点列表的内部, 也就是非端点的位置。

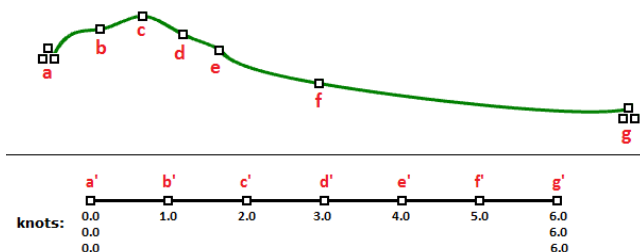


图 38 夹紧曲线在起点和终点具有完整重复节点, 内部的节点都是单一节点

3.2.9. 均匀节点列表

均匀节点列表满足以下条件。

节点列表已完全重复节点开始，然后是单一节点，最后又是完全重复节点，这些节点的值满足等差序列并且递增。典型的是上一节说的钳位的开放曲线。稍后我们将看到周期曲线，他有不同的节点形式。

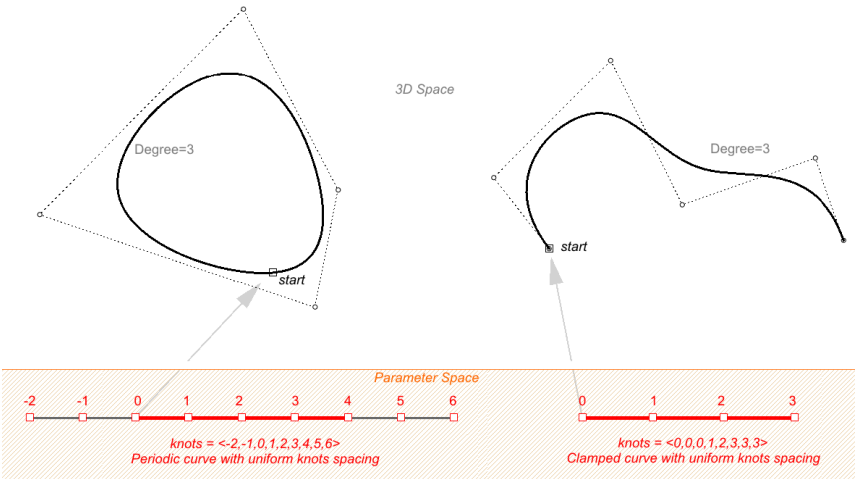


图 39 均匀节点列表意味着结之间的间距是恒定的，但夹紧曲线除外，它们在开始和结束时可以有完整的完全重复节点，并且仍然被认为是均匀的。左边的曲线是周期性的（闭合，没有扭结），右边的曲线是夹紧的（打开的）。

3.2.10. 非均匀节点列表

允许 NURBS 曲线在节点间具有不同的间距有助于控制曲线曲率，已创建更顺滑的曲线，已下图为例，左侧使用非均匀节点列表构造曲线，右侧使用均匀节点列表构造曲线，一般来说如果 NURBS 曲线的节点值之间的间距和控制点之间的距离成正比，曲线会更顺滑。

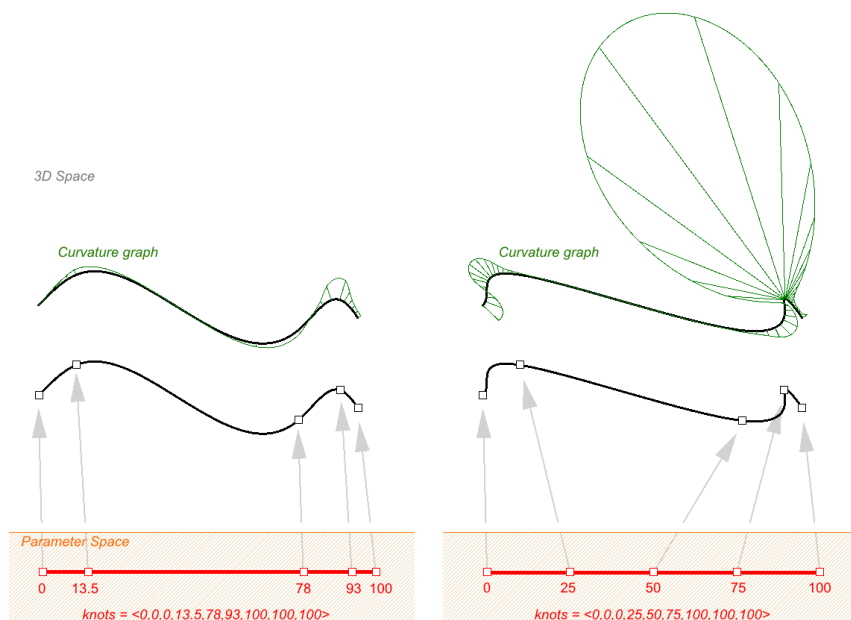


图 40 非均匀节点列表可以帮助平滑曲线

一个既非均匀又有理的曲线示例是 NURBS 圆，下图是具有九个控制点节点列表长度为 10 的 2 阶曲线，曲线区间为 0 到 4，节点间距在 0 和 1 之间来回横跳。

- 节点列表 = $\langle 0, 0, 1, 1, 2, 2, 3, 3, 4, 4 \rangle$ — (内部的完全重复节点)
- 节点间距 = $[0, 1, 0, 1, 0, 1, 0, 1, 0]$ — (非均匀)

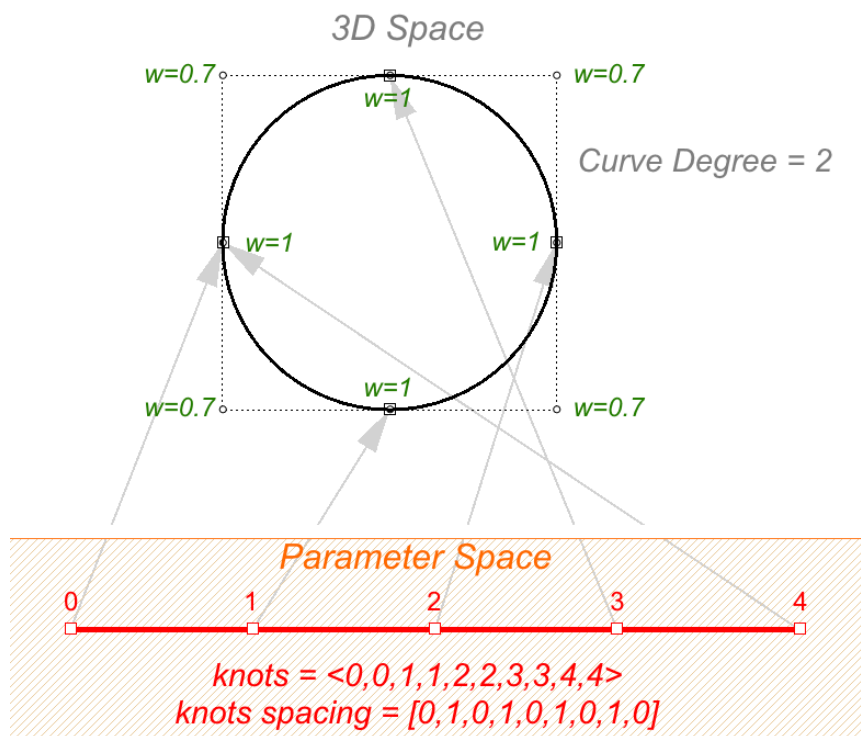


图 41 非均匀有理曲线示例 NURBS 圆¹⁸

3.2.11. 计算规则

计算规则使用了一个考虑阶数, 控制点, 节点向量的数学公式, 该函数输入曲线区间内的一个数返回一个点。

使用该公式, 函数可以计算曲线参数并且生成对应的点。参数是曲线区间内的值。区间一般来说是递增区间, 由曲线起点的区间最小值 t_0 和曲线终点的区间最大值 t_1 组成。

¹⁸译者注, 此处 $w=0.7$ 为约数, 准确值应为 $\frac{\sqrt{2}}{2}$

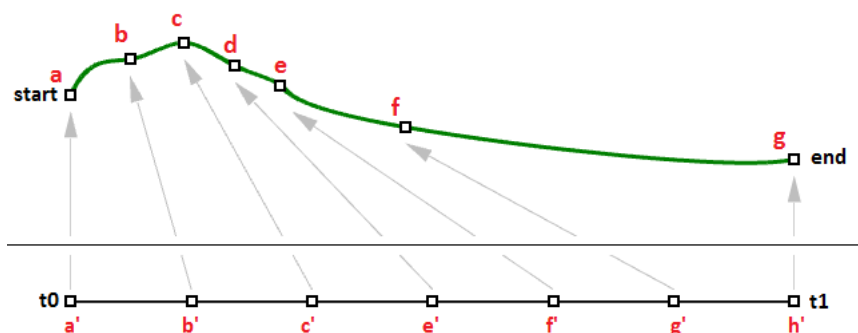


图 42 计算曲线上参数到点

3.2.12. NURBS 曲线的特性

为了创建一条 NURBS 曲线，通常来说需要提供如下信息：

- 维度，通常为 3
- 阶数（有时候使用 Order，等于阶数+1）
- 控制点列表
- 控制点权重列表
- 节点列表

创建曲线时，一般至少需要提供控制点的位置和阶数，其他的信息可以根据阶数和控制点列表自动生成。选择与起点重合的终点通常会创建一条闭合的周期曲线。下表是一些开放和封闭曲线的示例。

一阶开放曲线，曲线经过所有控制点	
------------------	--

<p>三阶开放曲线, 曲线端点均与控制点重合</p>	
<p>三阶闭合周期曲线, 曲线接缝 (起点终点重合位置) 不经过控制点</p>	
<p>移动周期曲线的控制点不会影响曲线连续性</p>	
<p>当其曲线被迫通过某个控制点时, 会产生纽结</p>	
<p>移动非周期曲线的控制点并不能保证曲线的连续性, 但可以更好的控制结果。</p>	

3.2.13. 夹紧和周期 NURBS 曲线

闭合夹紧曲线的端点和控制点重合，周期曲线时平滑的闭合曲线，二者差异可以通过比较控制点和节点快速了解。

下面是一个开放的钳位非有理 NURBS 曲线的示例。该曲线有四个控制点，端点上有完全重复节点，且控制点权重均为 1。

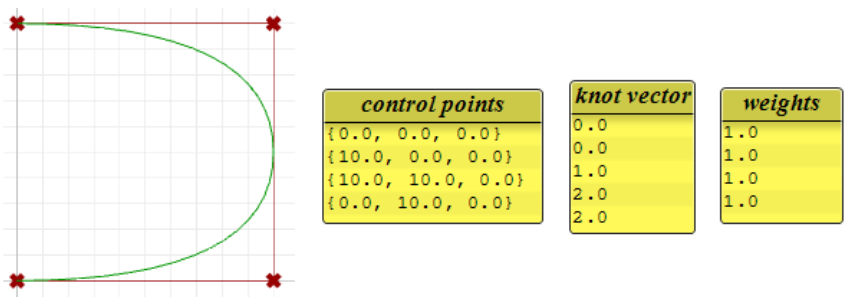


图 43 分析三阶开放非有理 NURBS 曲线

下面的圆形曲线¹⁹是三阶闭合周期曲线的例子，他也是非有理即等权重的。注意周期曲线需要更多的控制点，并且有几个点重叠了。节点列表全部是单一节点。

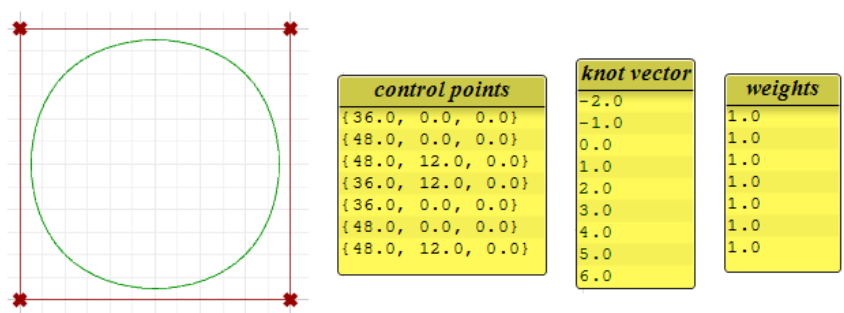


图 44 分析三阶闭合周期 NURBS 曲线

¹⁹译者注：不是圆

注意周期曲线将四个控制点变成了七个（4+阶数）控制点，而钳位曲线仅仅使用四个控制点。周期曲线只使用单一节点，钳位曲线的端点使用完全重复节点，使曲线端点经过控制点。

如果我们将前面案例的阶数设置为二阶而不是3阶，那么节点列表长度会变小，并且周期曲线控制点数量会发生变化。

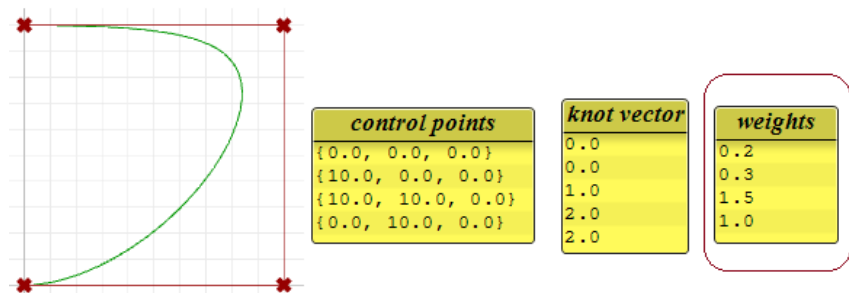


图 45 分析二阶开放 NURBS 曲线

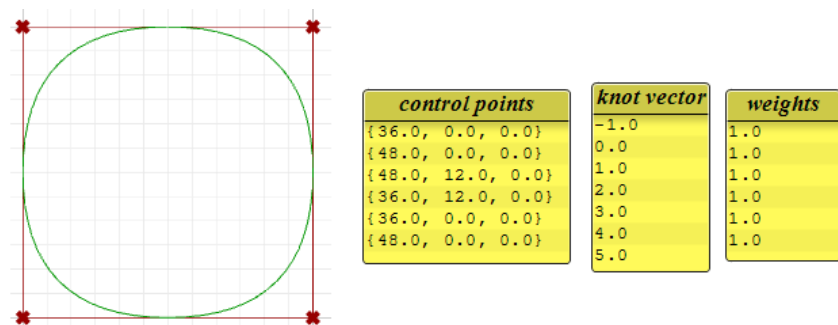


图 46 分析二阶闭合周期 NURBS 曲线

3.2.14. 权重

非有理均匀 NURBS 曲线的控制点权重均为 1，但在有理曲线中可能会不一致，下面展示了权重变化的效果。

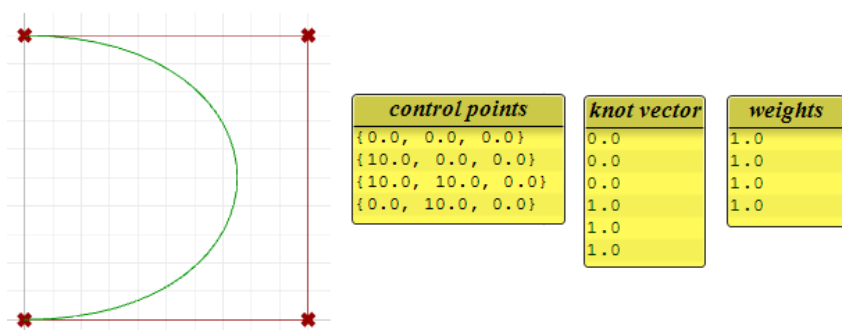


图 47 分析开放 NURBS 曲线的权重

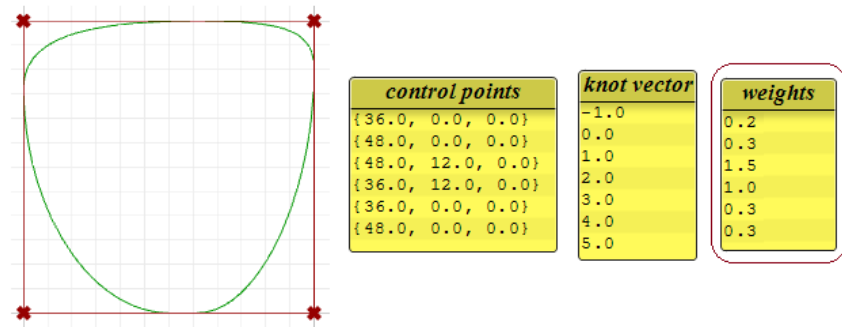


图 48 分析闭合 NURBS 曲线的权重

3.2.15. NURBS 曲线的计算

de Boor 算法以其发明者 Carl de Boor 命名，是 Bezier 曲线 de Casteljau 算法的推广。它具有数值稳定性，广泛用于评估 3D 应用中 NURBS 曲线上的点。以下是使用 de Boor 算法评估 3 阶 NURBS 曲线上的点的示例。

输入:

- 七个控制点 P_0 到 P_6
- 节点列表:

$$u_0 = 0.0$$

$$u_1 = 0.0$$

$$u_2 = 0.0$$

$$u_3 = 0.25$$

$$u_4 = 0.5$$

$$u_5 = 0.75$$

$$u_6 = 1.0$$

$$u_7 = 1.0$$

$$u_8 = 1.0$$

输出:

- 曲线上 $u = 0.4$ 的点

解决步骤:

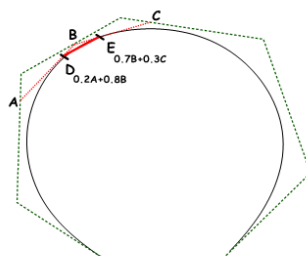
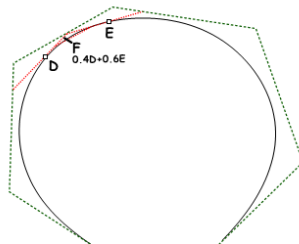
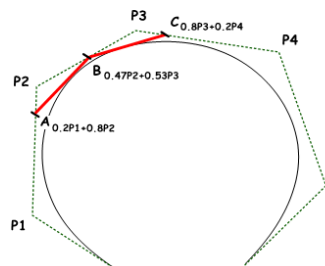
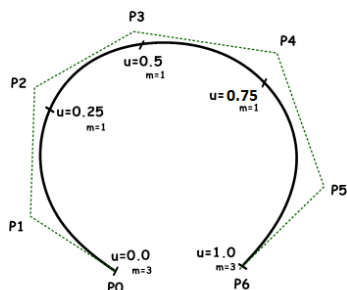
1. 计算第一次迭代的系数

$$A_c = \frac{u - u_1}{u_{1+3} - u_1} = 0.8$$

$$B_c = \frac{u - u_2}{u_{2+3} - u_2} = 0.53$$

$$C_c = \frac{u - u_3}{u_{3+3} - u_3} = 0.2$$

2. 使用系数数据计算点:



$$A = 0.2P_1 + 0.8P_2$$

$$B = 0.47P_2 + 0.53P_3$$

$$C = 0.8P_3 + 0.2P_4$$

3. 计算第二次迭代的系数:

$$D_c = \frac{u - u_2}{u_{2+3-1} - u_2} = 0.8$$

$$E_c = \frac{u - u_3}{u_{3+3-1} - u_3} = 0.3$$

4. 使用系数计算点:

$$D = 0.2A + 0.8B$$

$$E = 0.7B + 0.3C$$

5. 计算最后一个系数:

$$F_c = \frac{u - u_3}{u_{3+3-2} - u_3} = 0.6$$

6. 找到 $u = 0.4$ 参数处的点:

$$F = 0.4D + 0.6E$$

3.3. 曲线的几何连续性

连续性是 3D 建模中的一个重要概念。连续性对于实现视觉平滑度以及获得平滑的光线反射和空气阻力非常重要。下表显示了各种连续性及其定义:

- **G0** (位置连续): 两条曲线段仅连接在一起。
- **G1** (切向连续): 两条曲线段在连接点处切向相同。

- **G2** (曲率连续): 两条曲线段在连接点处的曲率和切向一致。
- **GN**: 更高的连续性...

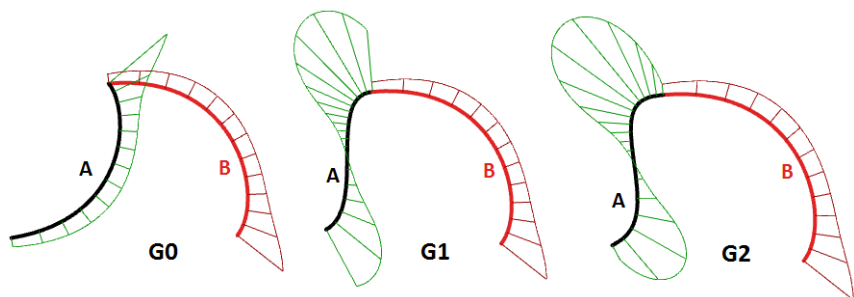


图 49 使用曲率图检查曲线连续性

3.4. 曲线曲率

曲率是 3D 曲线和曲面建模中广泛使用的概念。曲率定义为单位弧长度上曲线切线的角度变化。对于圆或球体，它是半径的倒数，并且在整个区间中是恒定的。

在平面中曲线上的任意一点，最接近通过的曲线该点的直线是切线。我们还可以找到经过该点并与曲线相切的最接近圆既曲率圆。该圆的半径的倒数就是该点的曲线曲率。

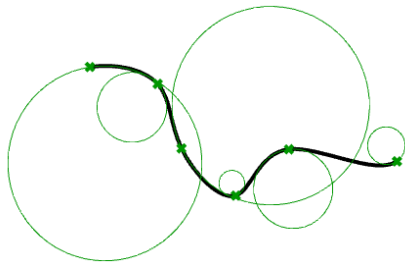


图 50 检查不同点处的曲线曲率

曲率圆可以位于曲线的左侧或右侧。如果我们关心这一点，我们会建立一个约定，例如如果圆位于曲线的左侧则给出曲率正号，如果圆位于曲线的右侧则给出负号。这称为有符号曲率。连接曲线的曲率值表示这些曲线之间的连续性。

3.5. 参数曲面

3.5.1. 曲面参数

参数曲面是二维区间上有两个独立参数（通常表示为 u 和 v ）的函数。已平面为例，如果我们在平面上有一点 P ，并且平面上有两个不平行的向量 \vec{a} 和 \vec{b} ，那么我们可以用 u 和 v 定义平面的参数方程如下：

$$P = P' + u \cdot \vec{a} + v \cdot \vec{b}$$

其中：

- P' : 是平面上的已知点
- \vec{a} : 是平面上的第一个向量
- \vec{b} : 是平面上的第二个向量
- u : 第一个参数
- v : 第二个参数

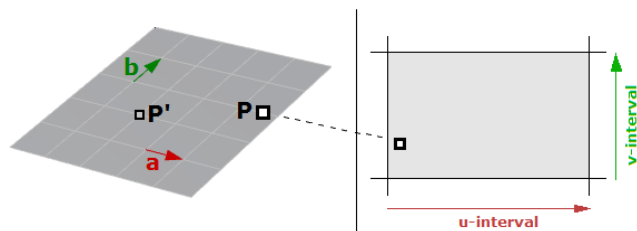


图 51 参数区间矩形平面

另一个例子是球体。已原点为圆心半径位 R 的球体的笛卡尔方程为：

$$x^2 + y^2 + z^2 = R^2$$

这意味着三个变量才能确定一个点, 这对于需要两个变量的参数曲面来说没有意义。但在球面坐标系中, 可以使用如下三个值进行球面上点的定位:

- r : 点到原点 (球心) 的距离
- θ : 在 xy 平面上与 x 轴的夹角
- φ : 和 z 轴的夹角

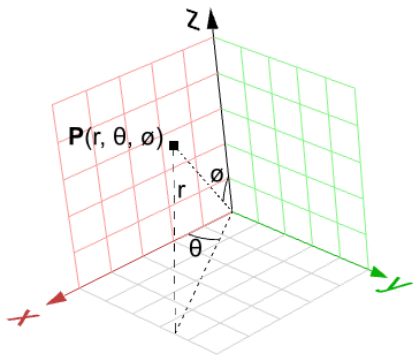


图 52 球面坐标系

从球面坐标到笛卡尔坐标的点变换可通过如下公式计算:

$$x = r \cdot \sin(\varphi) \cdot \cos(\theta)$$

$$y = r \cdot \sin(\varphi) \cdot \sin(\theta)$$

$$z = r \cdot \cos(\varphi)$$

其中:

- r 是到原点的距离, 其值大于 0
- θ 的范围为从 0 到 2π
- φ 的范围为从 0 到 π

由于半径 r 是常数, 所以我们只剩下两个变量, 因此我们可用上述的变量来创建球面的参数表示:

$$u = \theta$$

$$v = \varphi$$

即:

$$x = r \cdot \sin(v) \cdot \cos(u)$$

$$y = r \cdot \sin(v) \cdot \sin(u)$$

$$z = r \cdot \cos(v)$$

其中 (u, v) 在区间 $(2\pi, \pi)$ 内

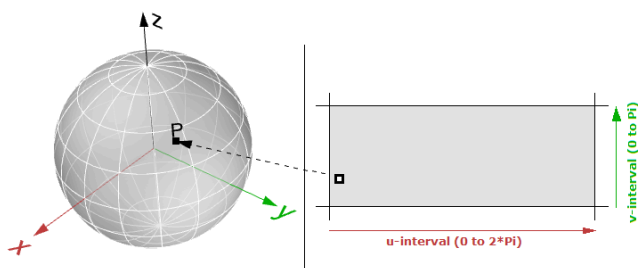


图 53 球面的参数区间矩形平面

参数曲面遵循如下的一般形式:

$$x = x(u, v)$$

$$y = y(u, v)$$

$$z = z(u, v)$$

其中: u 和 v 是曲面区间上的两个参数。

3.5.2. 曲面区间

曲面区间被定义为表示能曲面上点的 (u, v) 参数范围。每个方向(u 或 v)上参数的区间通常被描述为两个值组成的一维区间(u_{\min} 到 u_{\max})和(v_{\min} 到 v_{\max})。²⁰

更改曲面区间既重新参数化曲面。递增区间意味着区间的最小值表示曲面的最小点，和曲线相同，区间通常是递增的，但并非绝对。

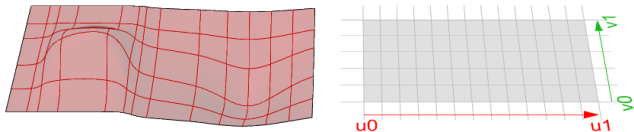


图 54 3D 空间中的 NURBS 曲面（左），曲面的参数区间矩形，区间在一个方向上从 u_0 到 u_1 ，第二个方向上从 v_0 到 v_1 （右）

3.5.3. 曲面计算

在曲面区间内的参数进行曲面计算会得到曲线上的点，同曲线一样曲面区间上的中心点不一定得到曲面的中心点。另外，曲面区间之外的 u 和 v 值不会得到有效的结果。

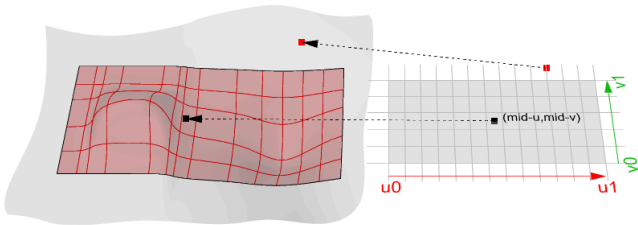


图 55 曲面计算

²⁰译者注：区间一词出现在曲面时除非特殊说明是一维区间，默认均指二维区间。

3.5.4. 曲面的切平面

给定点处曲线的切平面是在该点正好接触曲面的平面, 切平面的 z 方向表示该点的法线方向。

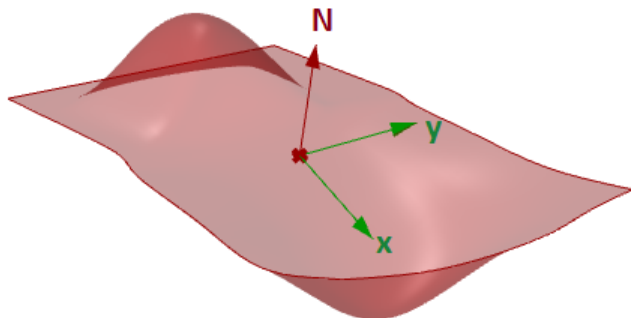


图 56 曲线的切向量和法向量

3.6. 曲面的几何连续性

许多模型不能仅由一个曲面建好, 曲面面片之间的连续性对于视觉上的平滑和光的反射以及空气阻力有重要意义。下面显示了各种连续性的定义。

- **G0** (位置连续): 两个曲面仅连接在一起。
- **G1** (切向连续): 两个曲面沿相接的边对应的切线在 u 和 v 方向均平行。
- **G2** (曲率连续): 两条曲面在连接线处的曲率和切向一致。
- **GN**: 更高的连续性...

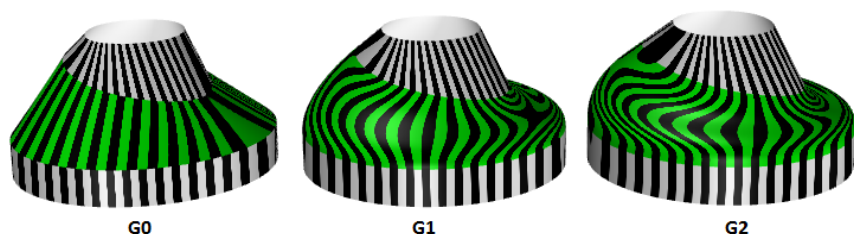


图 57 通过斑马纹检查曲面连续性

3.7. 曲面曲率

对曲面来说，法向曲率是曲线曲率在曲面上的推广。给定曲面上一个点和该点切平面上一个方向，法向曲率的计算方法是将曲面和由该点和法向和一个自定义的方向构造的平面和曲面相交形成的交线在该点的有符号曲率。

如果我们取切平面上该点所有的方向，并计算对应的所有法向曲率，将有一个最大值和最小值。

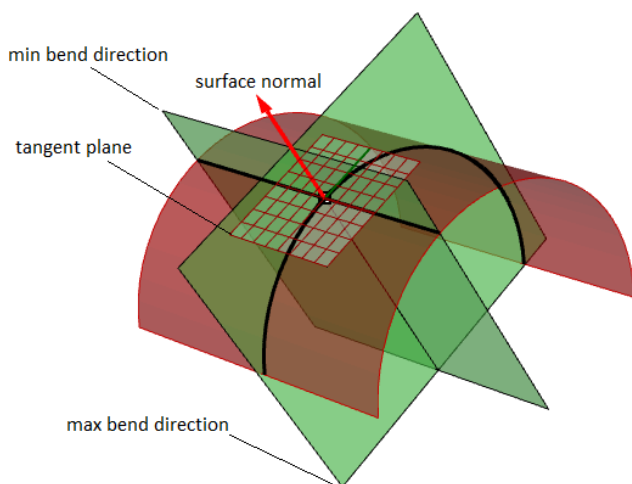


图 58 法向曲率

3.7.1. 主曲率

曲面上某点的主曲率是该点法向曲率的最大值和最小值, 他表征了曲面上该点的最大和最小弯曲。用主曲率计算曲面的高斯曲率和平均曲率。²¹

例如在圆柱面上, 沿轴向方向没有弯曲 (曲率为 0), 最大弯曲出现在平面和轴向垂直时 (曲率等于半径的倒数)。这两个曲率 (0 和半径的倒数) 构成了主曲率。

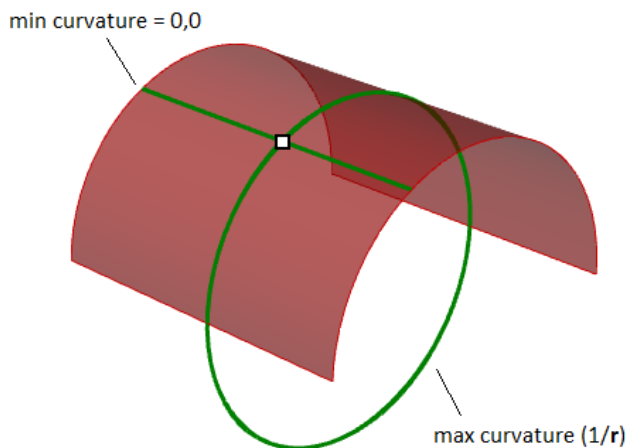
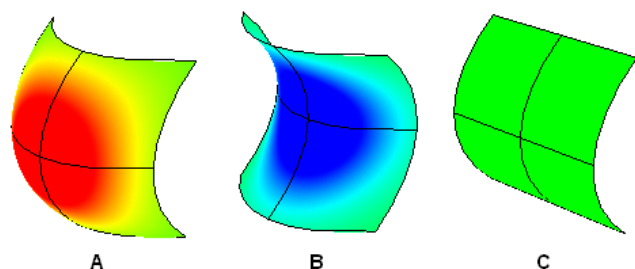


图 59 曲面上点的主曲率是该点的最大和最小曲率

3.7.2. 高斯曲率

曲面上某点的高斯曲率是该点的两个主曲率的乘积。曲面上有正高斯曲率的点切平面仅经过该点, 负高斯曲率的点切平面会切割表面。

²¹译者注: 光滑连续曲面某点的两个主曲率所在切割平面一定相互垂直。



- A: 碗状时高斯曲率为正。
- B: 马鞍状时高斯曲率为负。
- C: 当存在一个方向平坦时 (圆柱、平面等), 高斯曲率为 0。

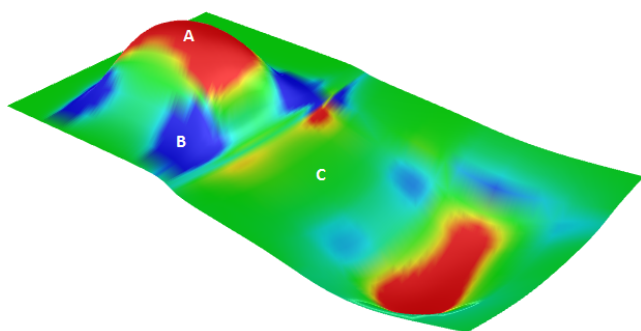


图 60 分析曲面的高斯曲率

3.7.3. 平均曲率

曲面上某点的平均曲率是主曲率的平均值。平均曲率为 0 那么高斯曲率一定小于等于 0。

处处平均曲率均为 0 的曲面被称为极小曲面。在固定物体 (如一个线环) 上维持稳态的肥皂膜是一个典型的物理上的极小曲面。肥皂膜在稳态状态下两边气压相等, 也没有形变, 此时其

面积也是最小化的。这与肥皂泡有极大区别，肥皂泡内部有空气，内外的压力并不一致。平均曲率可用于查找曲面曲率突变的区域。

处处平均曲率均相等的曲面被称为恒定曲率（CMC）曲面。CMC 曲面包括稳态后的肥皂泡，附着在物体上或漂浮的泡泡均可。和肥皂膜不同，肥皂泡内有气体，内外压力形成了平衡，中间的压力差由肥皂泡膜面积最小化趋势产生的力平衡。

3.8. NURBS 曲面

您可将 NURBS 曲面视为沿两个方向布置的 NURBS 曲线的网格。NURBS 曲面的形状由多个控制点以及该曲面在两个方向（ u 和 v 方向）中每个方向上的阶数定义。NURBS 曲面可高效存储和表示高精度的自由曲面。NURBS 曲面的数学方程和细节超出了本文的范围。我们将只关注对设计师最有用的特性。

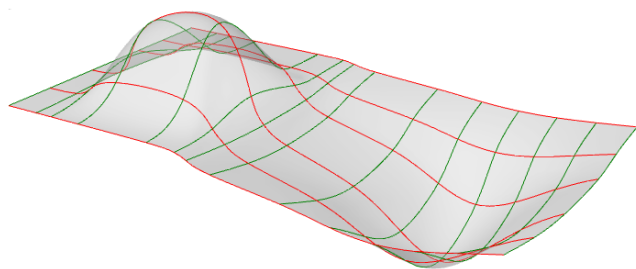


图 61 NURBS 曲面： u 方向为红色等值线， v 方向为绿色等值线

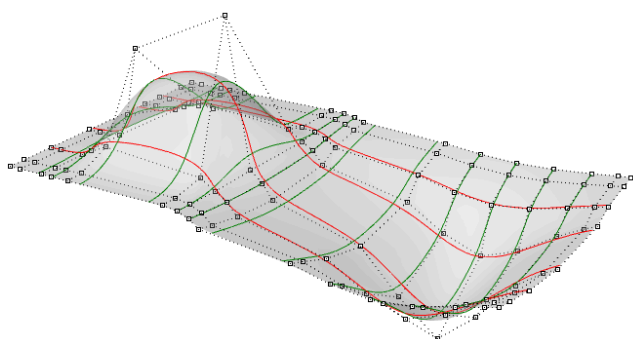


图 62 NURBS 曲面的控制结构

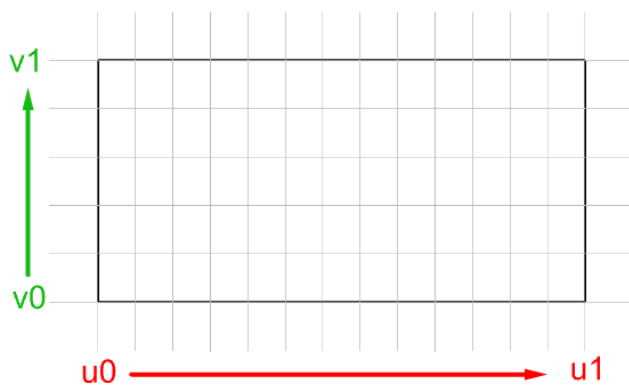


图 63 NURBS 曲面的参数区间矩形

在大多数情况下, 在参数区间矩形中相等间隔的参数不会转换为三维空间中的相等间隔。

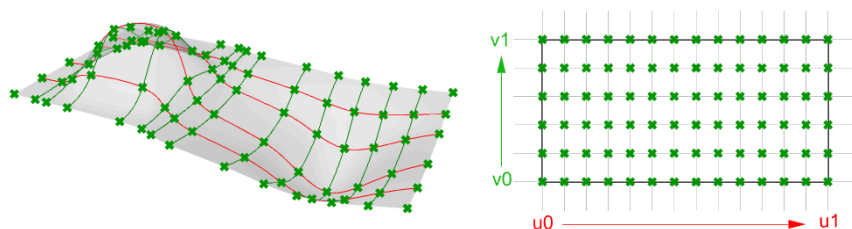


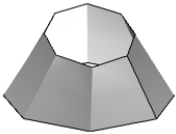
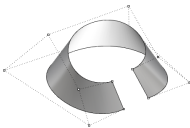
图 64 计算表面上的点

3.8.1. NURBS 曲面的特性

NURBS 曲面的特性和 NURBS 曲线非常相似，只是增加了一个维度。NURBS 曲面包含如下特性：

- 维度，通常为 3
- u 和 v 方向的阶数（有时候使用 Order，等于阶数+1）
- 控制点列表
- 控制点权重列表
- 节点列表

和 NURBS 曲线一样，我们不需要了解如何创建 NURBS 曲面的具体细节，因为建模软件通常会提供一组包装好的工具。我们始终可以将曲面（以及关联的曲线）重建为新的控制点数和阶数。曲面同样可以是开放的、闭合的又或者是周期的。下面是一些示例。²²

u 和 v 方向的 1 阶曲面。所有控制点都位于曲面上	
u 方向阶数为 3, v 方向阶数为 1 的开放曲面。曲面角点控制点角点重合	

²²译者注：下表中的闭合指一个方向的闭合而不是指曲面形成了封闭实体。

<p>u 方向阶数为 3, v 方向阶数为 1 的闭合 (非周期的) 曲面。控制点角点和曲面接缝重合</p>	
<p>移动闭合 (非周期的) 曲面会导致扭结, 切曲面看起来不光滑</p>	
<p>u 方向阶数为 3, v 方向阶数为 1 的闭合周期曲面。控制点和曲面接缝不重合</p>	
<p>移动周期曲面的控制点不会影响曲面连续性或产生扭结</p>	

3.8.2. NURBS 曲面的奇点

假如我们构造一个简单的一节矩形平面, 拖动两个端点控制点到中间重叠, 则获得了一个缩为单点的边。我们能注意到曲面的等值线在该点收敛。

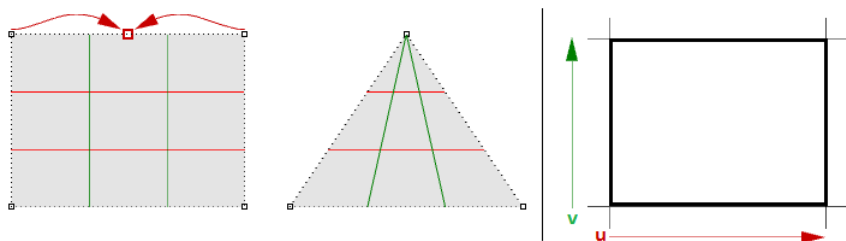


图 65 折叠曲面的两个角点，创建有奇点的三角形，参数区间矩形没有变化

上面的三角形也可以在不产生奇点的情况下构造，我们可以使用三角形线修剪曲面，当我们检查修剪曲面的底层 NURBS 结构时，我们会发现仍是矩形的。

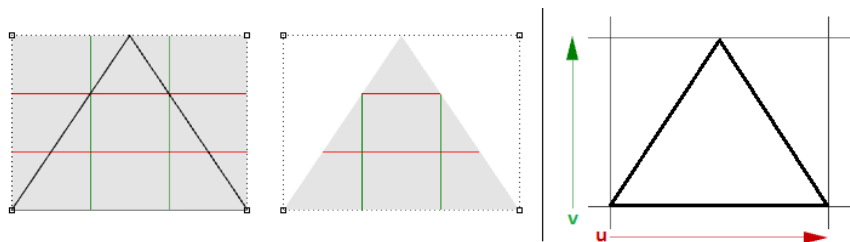


图 66 修剪矩形 NURBS 曲面创建三角形面

没有奇点就难以生成的曲面的其他常见例子是圆锥体和球体。圆锥体的顶部和球体的顶部和底部边缘折叠为一个点。无论是否存在奇点，参数区间矩形都或多或少地保持一个矩形区域。

3.8.3. 修剪的 NURBS 曲面

NURBS 曲面可以修剪或取消修剪。修剪曲面使用基础 NURBS 曲面和闭合曲线来修剪该曲面的一部分。每个曲面都有一条闭合曲线用于定义外边界(外环)，并且可以具有不相交的闭合内曲线来定义孔(内环)。具有与其底层 NURBS 曲面相同的外环且没有孔的曲面就是我们所说的未修剪曲面。

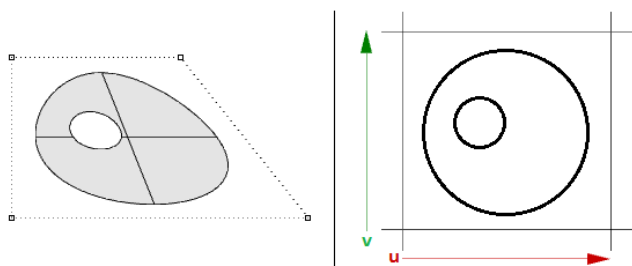


图 67 模型空间里和参数区间中的修剪曲面

3.8.4. 多重曲面

一个多重曲面由两个或多个 NURBS 曲面（可能修剪过）组合组成。每个曲面都有自己的结构, 参数区间, 结构线方向等, 并且多个曲面间无需一致。多重曲面使用边界表示法 (BRep) 表示。BRep 结构描述了曲面、边缘、顶点以及他们的修剪和连接关系。修剪后的曲面也可由 BRep 数据结构组织。

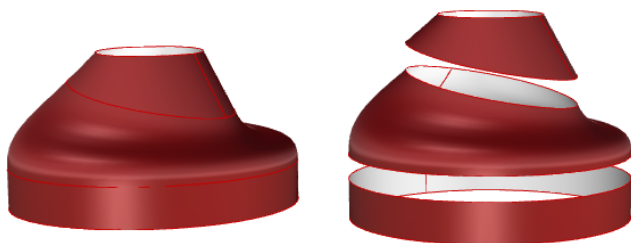


图 68 多重曲面有曲面组合而成, 其公共边缘在公差范围内一致

BRep 是一种数据结构, 它根据每个面的底层曲面、面的边缘、顶点、参数空间 2D 修剪以及每个相邻面之间的关系来描述多重曲面的数据结构。BRep 对象在封闭 (水密) 时也称为实体。

如下图多重曲面是一个简单的立方体, 由六个未修剪曲面连接在一起组成。

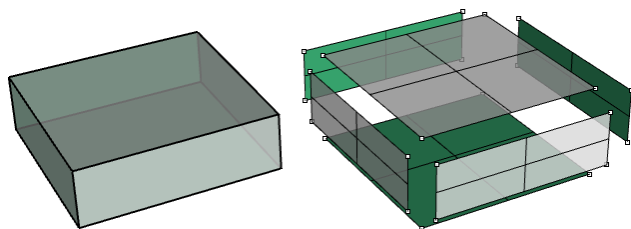


图 69 由六个未修剪曲面连接在一起组成的多重曲面立方体

也可以使用修剪的曲面组成相同的立方体, 如下图中顶部的曲面。

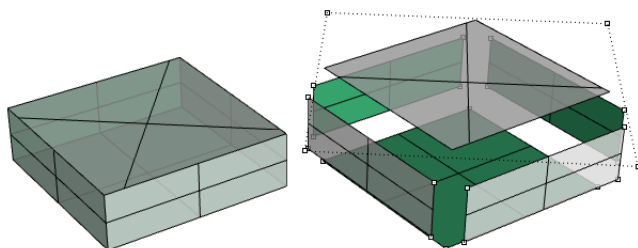


图 70 已修剪面组成的立方体

下面示例中的圆柱体的顶面和底面是一个修剪的平面。

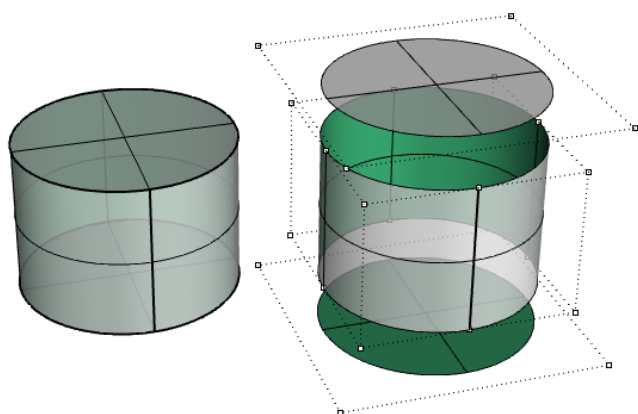


图 71 底面上控制点的显示

我们看到，编辑 NURBS 曲线和未修剪的曲面非常直观，并且可以通过移动控制点的方式交互完成。但是，编辑修剪曲面和多重曲面可能非常难。主要挑战是能够将不同面的连接边缘保持在所需的公差范围内。共享公共边的相邻面可以进行修剪，并且通常不具有匹配的 NURBS 结构，因此以使该公共边变形的方式修改对象可能会导致间隙。

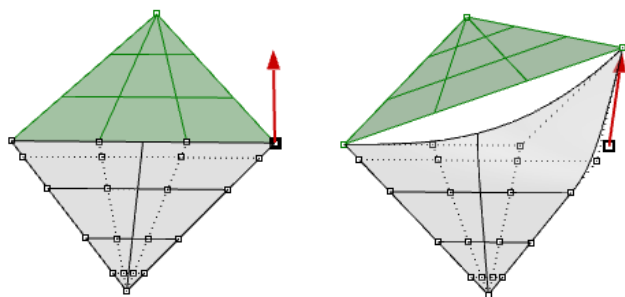


图 72 两个三角形面连接在一个多重曲面中，但没有匹配的连接边。移动其中一个角会创建一个孔

另一个挑战是，通常较难直接控制结果，尤其是在修改修剪几何图形时。

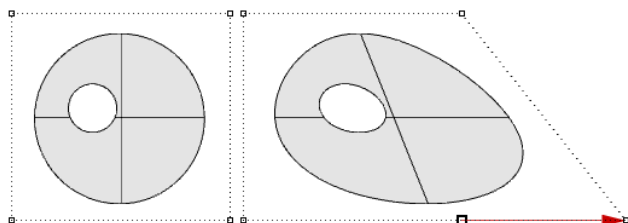


图 73 创建修剪曲面后，对编辑结果控制有限

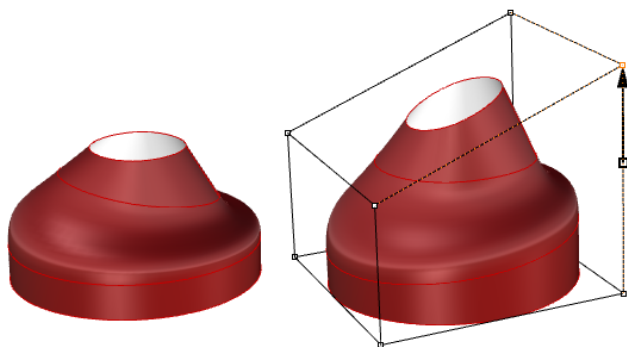


图 74 使用变形盒编辑多重曲面

在参数区间矩形中，使用未修剪的底层曲面与二维修剪曲线相结合来描述修剪曲面，这些曲线的计算结果为三维曲面内的三维边。

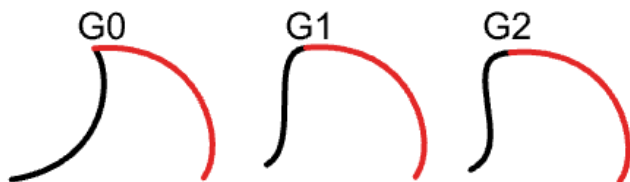
3.9. 实例教程

以下案例使用本章中学习的概念。他们使用 Rhinoceros 5 和 Grasshopper 0.9 创建。²³

²³译者注：经测试教程中的案例在 Rhino8 和 GH1 中可正常运行。

3.9.1. 曲线连续性

检查两条输入曲线之间的连续性。假定曲线在第一条曲线的末端和第二条曲线的起点相交。

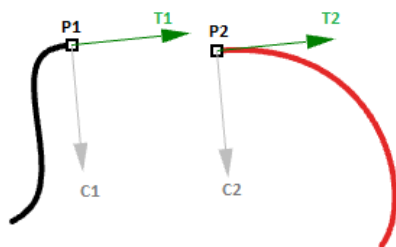


输入:

- 两条曲线

参数:

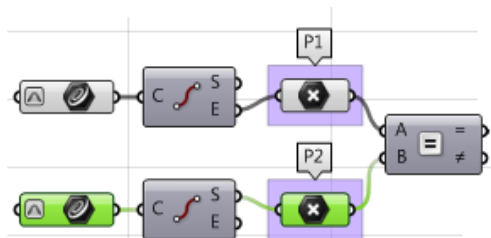
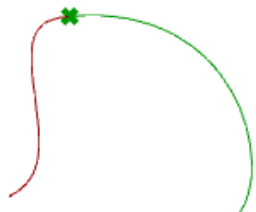
计算以下内容以确定连续性



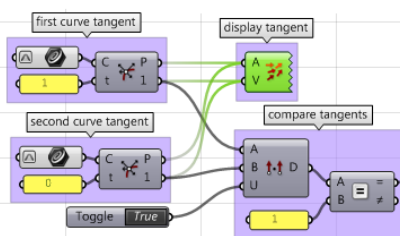
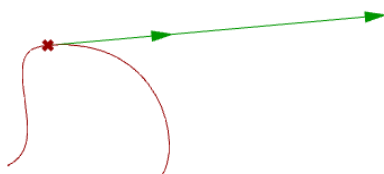
- 第一条曲线的终点 P1
- 第二条曲线的起点 P2
- 第一条曲线末端和第二条曲线起点处的切线: T1 和 T2
- 第一条曲线末端和第二条曲线起点处的曲率: C1 和 C2

解决方案:

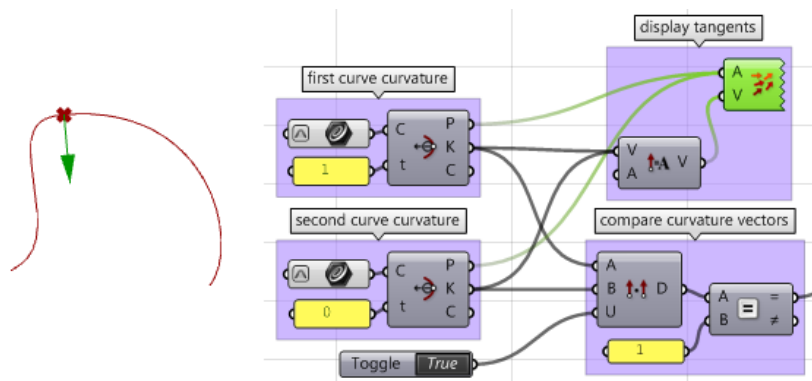
1. 重新参数化输入曲线。我们这样做是为了我们知道曲线的起点在 $t = 0$ 计算，终点在 $t = 1$ 计算。
2. 提取两条曲线的终点和起点，并检查它们是否重合。如果是，两条曲线至少是 G_0 连续。



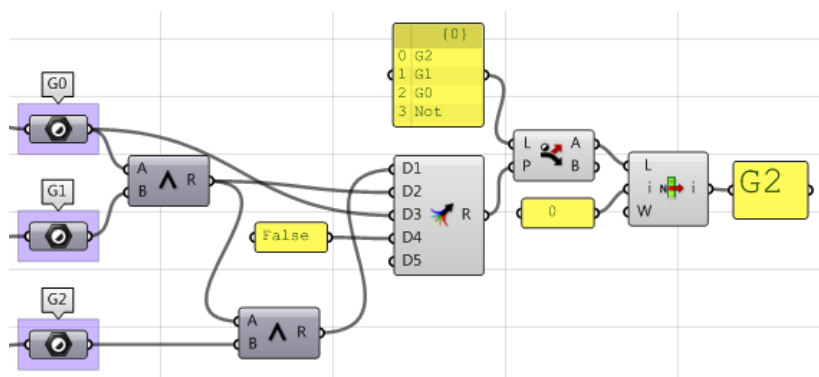
3. 计算切向量
4. 使用点积比较切线，确定做了向量单位化操作。如果向量平行，那么我们有 G_1 连续。



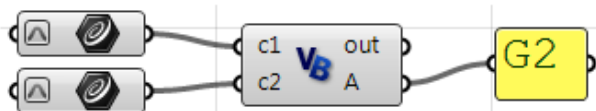
5. 计算曲率向量（法向）
6. 比较曲率向量，如果二者一致，则两曲线 G_2 连续。



7. 创建筛选三个结果（G1、G2 和 G3）的逻辑，并选择最高的连续性。



使用 VB 组件：



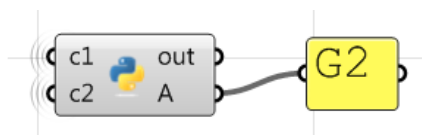
	Private Sub RunScript(ByVal c1 As	VB Visual Basic
1	Curve, ByVal c2 As Curve, ByRef A As Object)	
2		
3	' 定义变量	
4	Dim continuity As New String("")	
5	Dim t1, t2 As Double	
6	Dim v_c1, v_c2, c_c1, c_c2 As Vector3d	
7		
8	' 得到起点终点	
9	Dim end_c1 = c1.PointAtEnd	
10	Dim start_c2 = c2.PointAtStart	
11		
12	' G0连续检查	
13	If end_c1.DistanceTo(start_c2) = 0 Then	
14	continuity = "G0"	
15	End If	
16		
17	' G1连续检查	
18	If continuity = "G0" Then	
19	' 计算切向	
20	v_c1 = c1.TangentAtEnd	
21	v_c2 = c2.TangentAtStart	
22	' 单位化向量	
23	v_c1.Unitize	
24	v_c2.Unitize	
25	' 比较向量	
26	If v_c1 * v_c2 = 1.0 Then	
27	continuity = "G1"	
28	End If	


```

29 End If
30
31 'G2连续检查
32 If continuity = "G1" Then
33     '获得曲线区间最值
34     t1 = c1.Domain.Max
35     t2 = c2.Domain.Min
36     '计算曲率向量
37     c_c1 = c1.CurvatureAt(t1)
38     c_c2 = c2.CurvatureAt(t2)
39     '单位化
40     c_c1.Unitize
41     c_c2.Unitize
42     '向量比较
43     If c_c1 * c_c2 = 1.0 Then
44         continuity = "G2"
45     End If
46 End If
47
48 '结果输出
49 A = continuity
50
51 End Sub

```

使用 Python 组件：



 Python

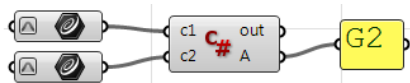
```
1  #定义变量
2  continuity = ""
3
4  #得到起点终点
5  end_c1 = c1.PointAtEnd
6  start_c2 = c2.PointAtStart
7
8  #G0连续检查
9  if end_c1.DistanceTo(start_c2) == 0:
10     continuity = "G0"
11
12  #G1连续检查
13  if continuity == "G0":
14     #计算切向
15     v_c1 = c1.TangentAtEnd
16     v_c2 = c2.TangentAtStart
17     #单位化向量
18     v_c1.Unitize()
19     v_c2.Unitize()
20     #比较向量
21     dot = v_c1 * v_c2
22     if dot == 1.0:
23         continuity = "G1"
24     else:
25         print("Failed G1")
26         print(dot)
27
28  #G2连续检查
29  if continuity == "G1":
30
```

```

31     #获得曲线区间最值
32     t1 = c1.Domain.Max
33     t2 = c2.Domain.Min
34     #计算曲率向量
35     c_c1 = c1.CurvatureAt(t1)
36     c_c2 = c2.CurvatureAt(t2)
37     #单位化
38     c_c1.Unitize()
39     c_c2.Unitize()
40     #向量比较
41     dot = c_c1 * c_c2
42     if dot == 1.0:
43         continuity = "G2"
44     else:
45         print("Failed G2")
46         print(dot)
47
48 #输出结果
49 A = continuity

```

使用 c# 组件:



```

1 private void RunScript(Curve c1, Curve
  c2, ref object A)
2 {
3     // 定义变量
4     string continuity = ("");

```



```

5      double t1, t2;
6      Vector3d v_c1, v_c2, c_c1, c_c2;
7
8      //得到起点终点
9      Point3d end_c1 = c1.PointAtEnd;
10     Point3d start_c2 = c2.PointAtStart;
11
12     //G0连续检查
13     if( end_c1.DistanceTo(start_c2) == 0){
14         continuity = "G0";
15     }
16
17     //G1连续检查
18     if( continuity == "G0")
19     {
20         //计算切向
21         v_c1 = c1.TangentAtEnd;
22         v_c2 = c2.TangentAtStart;
23         //单位化向量
24         v_c1.Unitize();
25         v_c2.Unitize();
26         //比较向量
27         if( v_c1 * v_c2 == 1.0 ){
28             continuity = "G1";
29         }
30     }
31
32     //G2连续检查
33     if( continuity == "G1" )
34     {

```

```

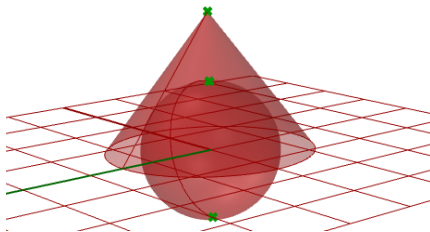
35      // 获得曲线区间最值
36      t1 = c1.Domain.Max;
37      t2 = c2.Domain.Min;
38      // 计算曲率向量
39      c_c1 = c1.CurvatureAt(t1);
40      c_c2 = c2.CurvatureAt(t2);
41      // 单位化
42      c_c1.Unitize();
43      c_c2.Unitize();
44      // 向量比较
45      if( c_c1 * c_c2 == 1.0 ){
46          continuity = "G2";
47      }
48  }
49
50      // 输出结果
51      A = continuity;
52  }

```

3.9.2. 有奇点的曲面

提取球体和圆锥中的奇点。

输入：



- 一个球体和一个圆锥体。

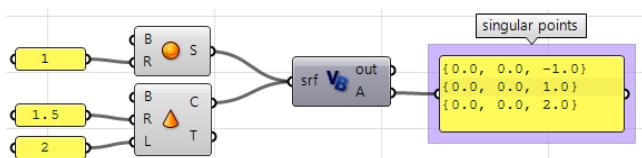
参数:

- 奇点可通过分析具有无效或 0 长度修剪边来检测。

解决方案:

1. 遍历所有边缘。
2. 检查是否有无效修剪边缘并标记。
3. 提取在空间中该点的位置。

使用 VB 组件:



```

1 Private Sub RunScript(ByVal srf As Brep, ByRef A As Object)
2
3     '定义点列表
4     Dim singular_points As New List(Of Point3d)
5
6     '获得所有BRep边缘
7     For Each trim As BrepTrim In srf.Trims
8
9         '标记为空的边缘
10        If trim.Edge Is Nothing Then
11
12            '找到对应参数区间的位置
13            Dim pt2d = New Point2d(trim.PointAtStart)

```

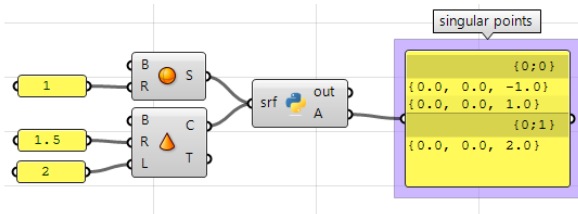


```

14
15     ' 计算模型空间中位置
16     Dim pt3d = trim.Face.PointAt(pt2d.x,
17                                     pt2d.y)
18
19     ' 添加到结果列表
20     singular_points.Add(pt3d)
21
22 End If
23
24 Next
25
26 ' 输出
27 A = singular_points
28
29 End Sub

```

使用 python 组件:



```

1  #定义点列表
2  singular_points = []
3
4  #获得所有BRep边缘
5  for trim in srf.Trims:
6

```

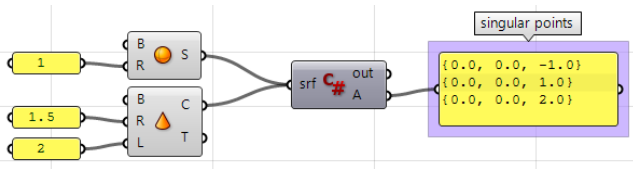
Python

```

7   #标记为空的边缘
8   if trim.Edge == None:
9       #找到对应参数区间的位置
10      pt2d = trim.PointAtStart
11
12      #计算模型空间中位置
13      pt3d = trim.Face.PointAt(pt2d.X, pt2d.Y)
14
15      #添加到结果列表
16      singular_points.append(pt3d)
17
18  #输出
19  A = singular_points

```

使用 c# 组件:



```

1  private void RunScript(Brep srf, ref
   object A)
2  {
3      // 定义点列表
4      List < Point3d > singular_points = new
       List<Point3d>();
5
6      // 获得所有BRep边缘
7      foreach( BrepTrim trim in srf.Trims)
8      {

```

C#

```

9      // 标记为空的边缘
10     if( trim.Edge == null)
11     {
12         // 找到对应参数区间的位置
13         Point2d pt2d = new
            Point2d(trim.PointAtStart);
14
15         // 计算模型空间中位置
16         Point3d pt3d = trim.Face.PointAt(pt2d.X,
            pt2d.Y);
17
18         // 添加到结果列表
19         singular_points.Add(pt3d);
20     }
21 }
22
23 // 输出
24 A = singular_points
25 }

```

4. 参考文献

4.1. 参考书目

1. Edward Angel, “Interactive Computer Graphics with OpenGL,” Addison Wesley Longman, Inc., 2000.
2. James D Foley, Steven K Feiner, John FHughes, “Introduction to Computer Graphics” Addison-Wesley Publishing Company, Inc., 1997.
3. James Stewart, “Calculus,” Wadsworth, Inc., 1991.
4. Kenneth Hoffman, Ray Kunze, “Linear Algebra”, Prentice-Hall, Inc., 1971
5. Rhinoceros® help document, Robert McNeel and Associates, 2009.

4.2. 相关知识链接

1. Wikipedia: Projection (linear algebra)²⁴
2. Wikipedia: Cubic Hermite spline.²⁵
3. Wikipedia: Bézier curve.²⁶
4. Wikipedia: Non-uniform rational B-spline.²⁷
5. Wikipedia: De Casteljau’s algorithm.²⁸
6. Wikipedia: NURBS.²⁹

²⁴[http://en.wikipedia.org/wiki/Projection_\(linear_algebra\)](http://en.wikipedia.org/wiki/Projection_(linear_algebra))

²⁵http://en.wikipedia.org/wiki/Cubic_Hermite_spline

²⁶http://en.wikipedia.org/wiki/B%C3%A9zier_curve

²⁷http://en.wikipedia.org/wiki/Non-uniform_rational_B-spline

²⁸http://en.wikipedia.org/wiki/De_Casteljau's_algorithm

7. Wikipedia: De Boor' s algorithm.³⁰
8. MichiganTech, Department of Computer Science, De Boor' s algorithm.³¹

²⁹<http://en.wikipedia.org/wiki/NURBS>

³⁰http://en.wikipedia.org/wiki/De_Boor's_algorithm

³¹<http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/de-Boor.html>