



# Lecture 10: 几何介绍

SSE315: 计算机图形学  
Computer Graphics

---

陈壮彬

软件工程学院

chenzhb36@mail.sysu.edu.cn

# Today's topics

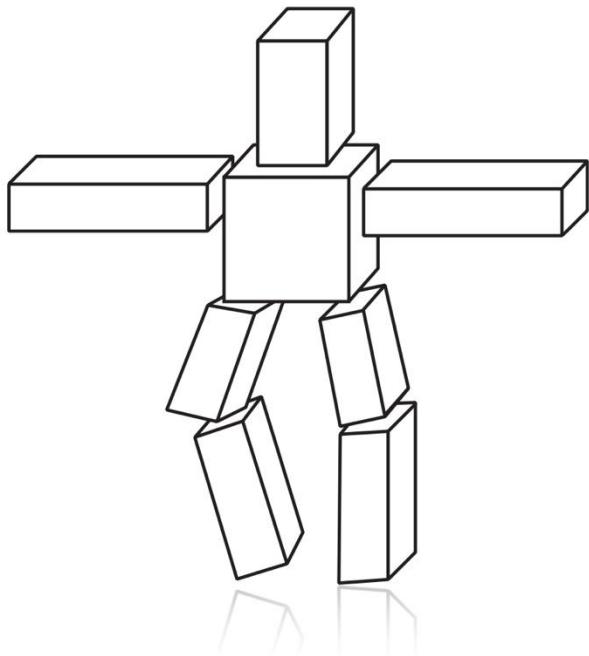
□ 几何基本介绍

□ 几何的隐式 (implicit) 表示

□ 几何的显式 (explicit) 表示

# 逐步提高模型的复杂度

Transformations



Geometry



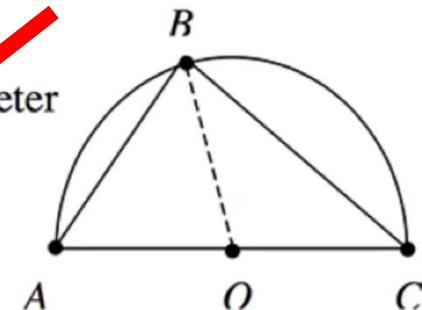
Materials, lighting



# 几何是什么?

**THEOREM 9.5.** Let  $\triangle ABC$  be inscribed in a semicircle with diameter  $\overline{AC}$ .

Then  $\angle ABC$  is a right angle.



*Proof:*

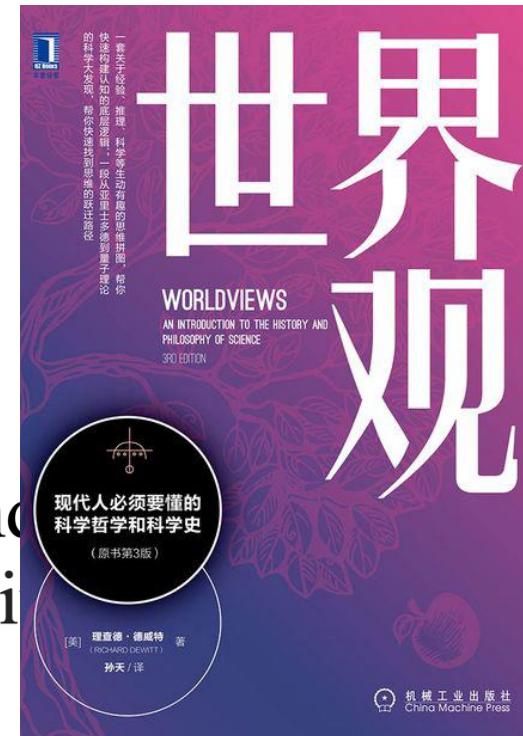
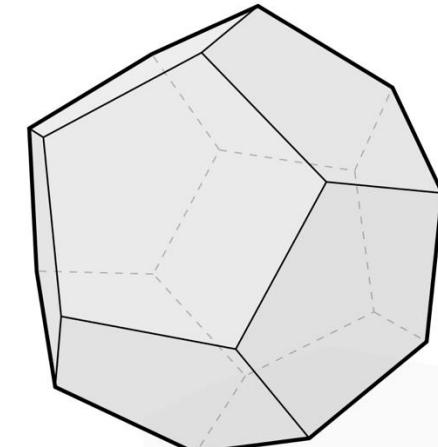
Statement	Reason
1. Draw radius $OB$ . Then $OB = OC = OA$	1. Given
2. $m\angle OBC = m\angle BCA$ $m\angle OBA = m\angle BAC$	2. Isosceles Triangle Theorem
3. $m\angle ABC = m\angle OBA + m\angle OBC$	3. Angle Sum Postulate
4. $m\angle ABC + m\angle BCA + m\angle BAC = 180$	4. The sum of the angles of a triangle is 180
5. $m\angle ABC + m\angle OBC + m\angle OBA = 180$	5. Substitution (line 2)
6. $2m\angle ABC = 180$	6. Substitution (line 3)
7. $m\angle ABC = 90$	7. Division Property of Equality
8. $\angle ABC$ is a right angle	8. Definition of Right Angle

# 几何是什么？

“Earth” “measure”

ge • o • m • et • ry /jē'ämətrē/ n.

1. The study of shapes, sizes, patterns, and positions of objects.
2. The study of spaces where some quantities (e.g., angles, etc.) can be *measured*.



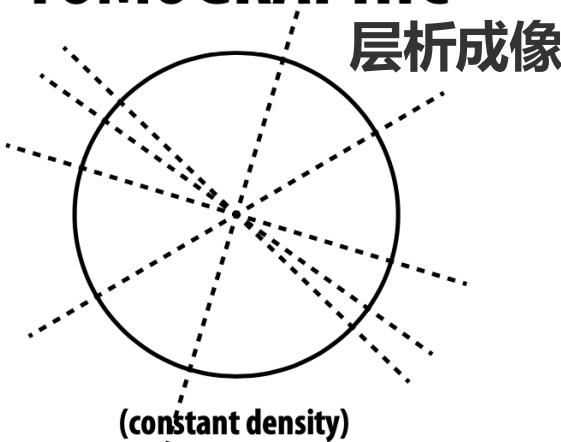
柏拉图：“.....地球的外观就像一个十二块皮革覆盖的球体.....”<sup>4</sup>

# 我们如何描述几何?

**IMPLICIT**

$$x^2 + y^2 = 1$$

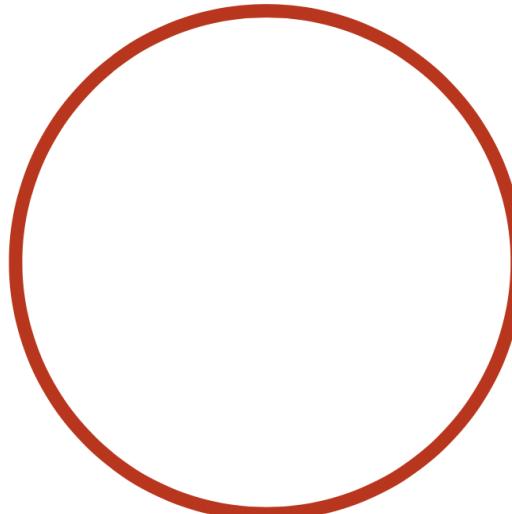
**TOMOGRAPHIC**



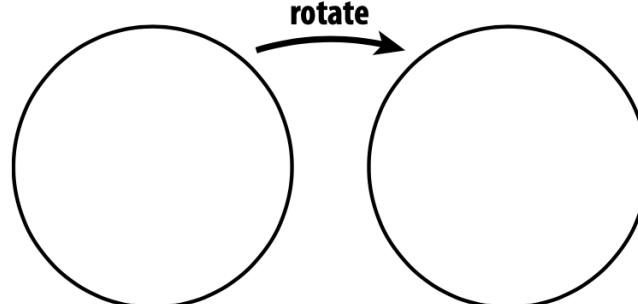
**CURVATURE**

$$\kappa = 1$$

**LINGUISTIC**  
“unit circle”



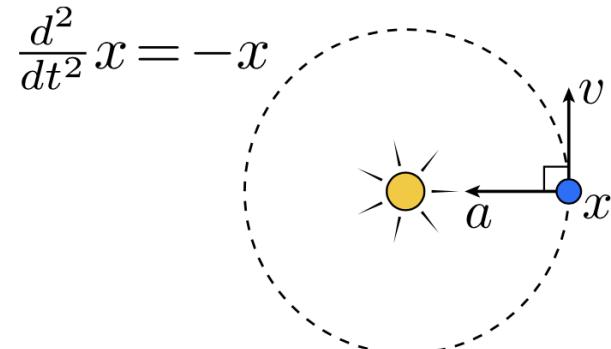
**SYMMETRIC**



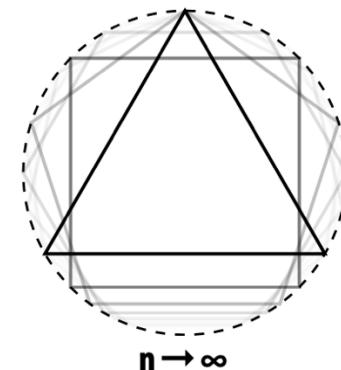
**EXPLICIT**

$$(\underbrace{\cos \theta}_{x}, \underbrace{\sin \theta}_{y})$$

**DYNAMIC**



**DISCRETE**

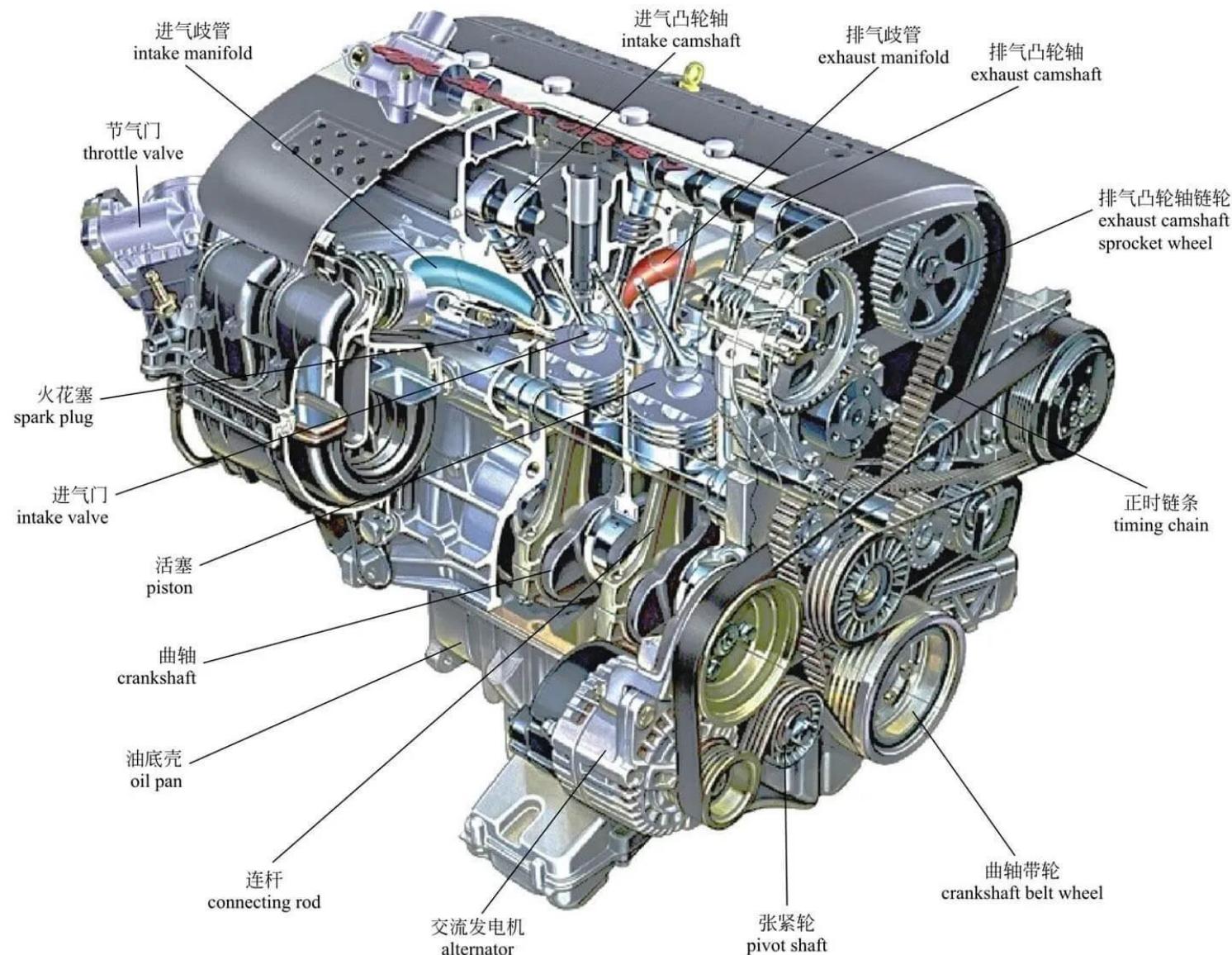


给定上述选项，在计算机中最  
佳的几何图形编码方式是什么？

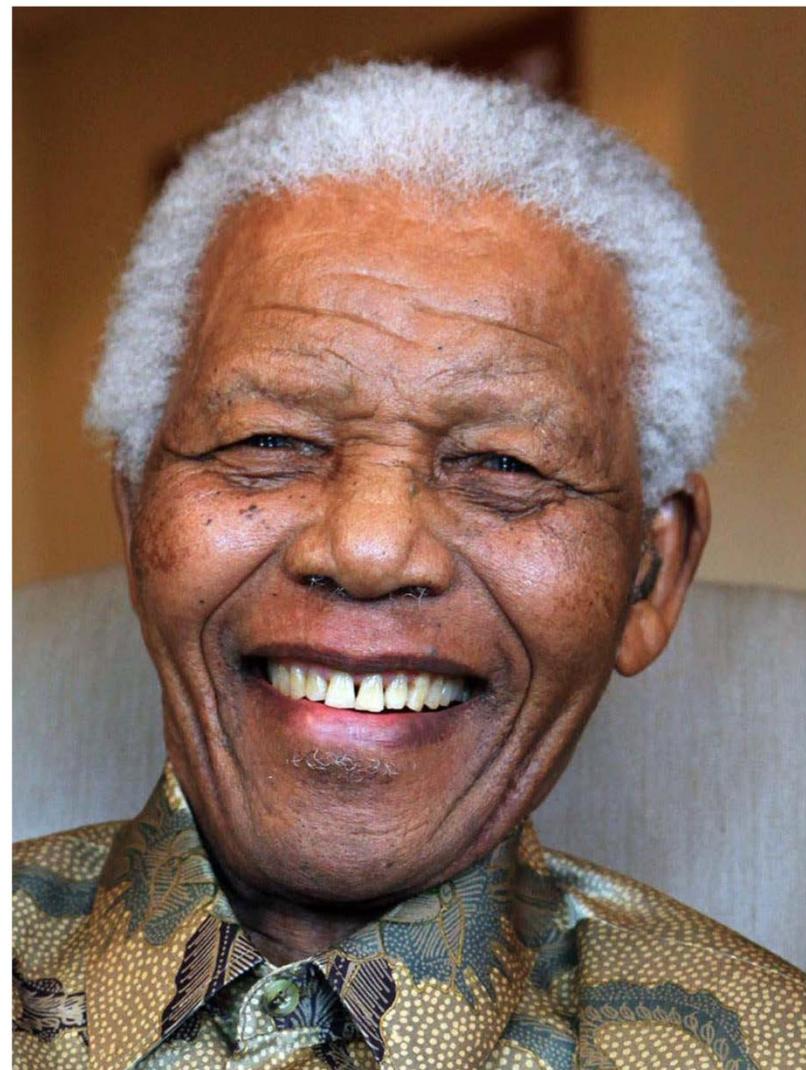
# 几何例子



# 几何例子



# 几何例子



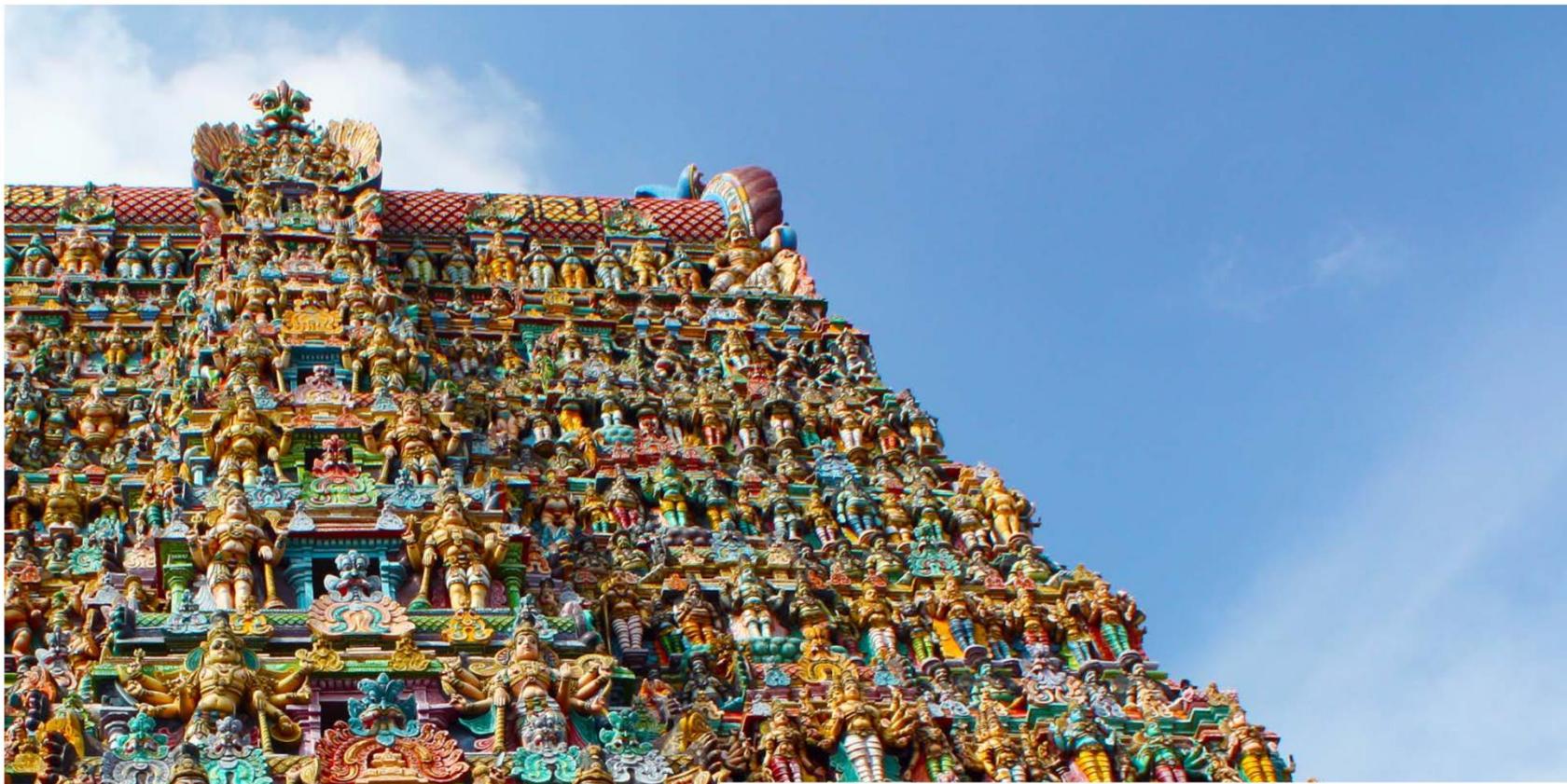
# 几何例子



# 几何例子



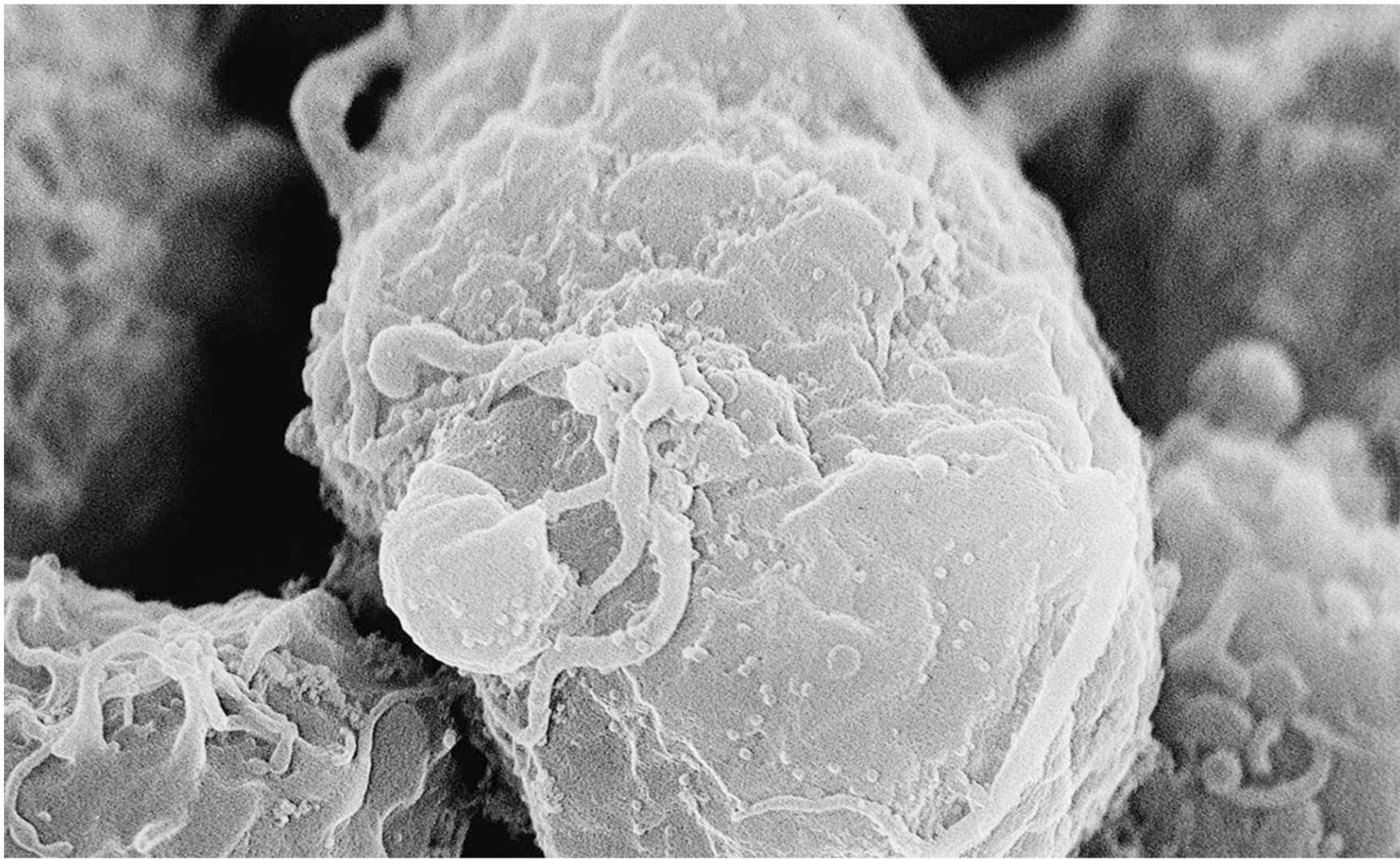
# 几何例子



# 几何例子



# 几何例子



# It's a jungle out there



# 没有最好的选择 – 几何是很复杂的

*“I hate meshes.  
I cannot believe how hard this is.  
Geometry is hard.”*

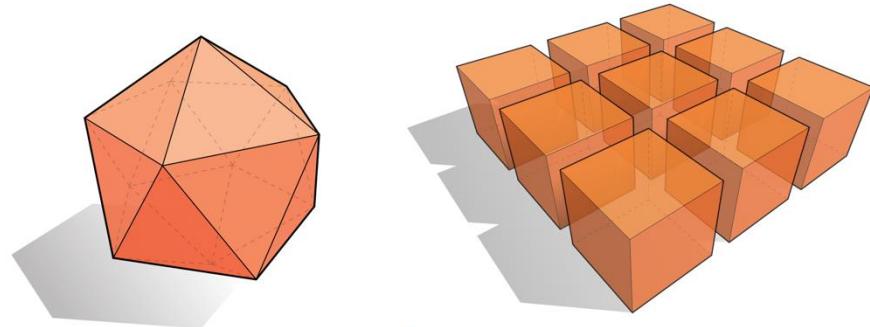
—David Baraff

Senior Research Scientist  
Pixar Animation Studios

# 很多数字化编码几何图形的方式

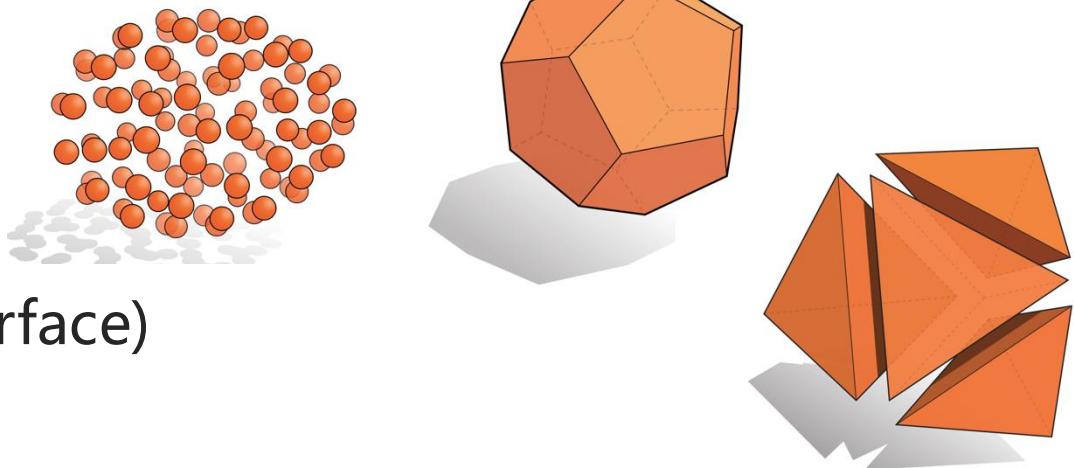
## □ 显式的 (explicit)

- 点云 (point cloud)
- 多边形网格 (polygon mesh)
- 细分 (subdivision), NURBS
- ...



## □ 隐式的 (implicit)

- 水平集 (level set)
- 代数曲面 (algebraic surface)
- L-系统 (L-systems)
- ...



## □ 每种选择适合不同的任务/几何体类型

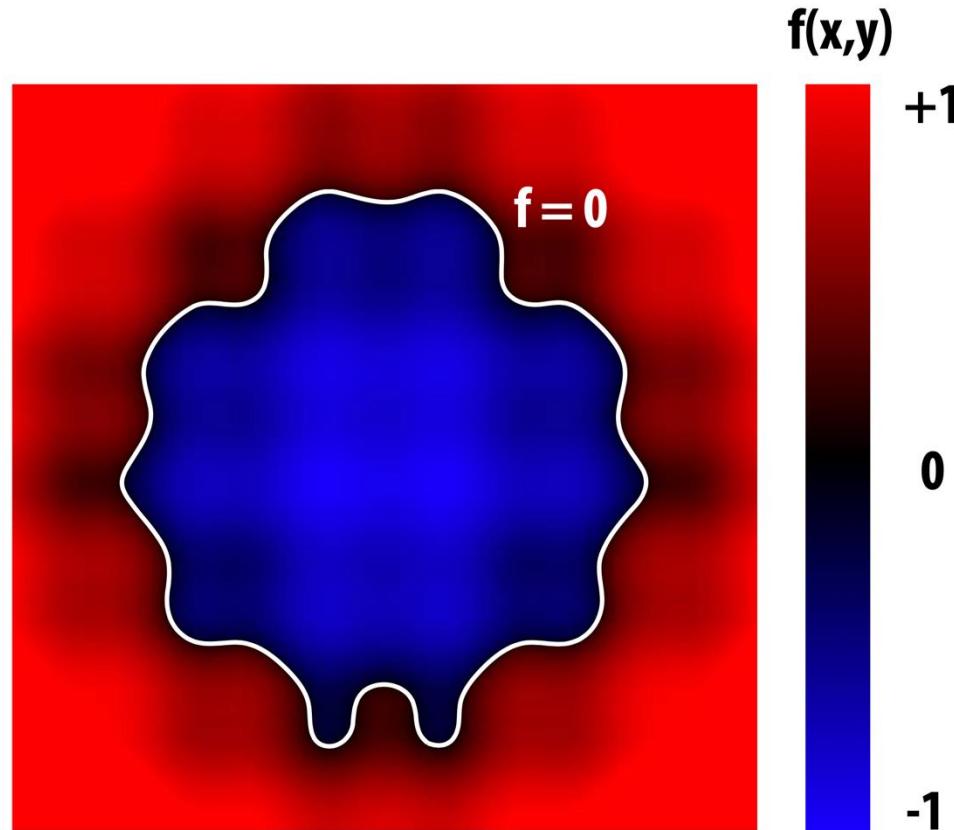
## □ 有时可能需要在不同表示方法之间来回转换

# 几何图形的隐式表示法

□不直接知道具体的点，但知道它们满足某些关系

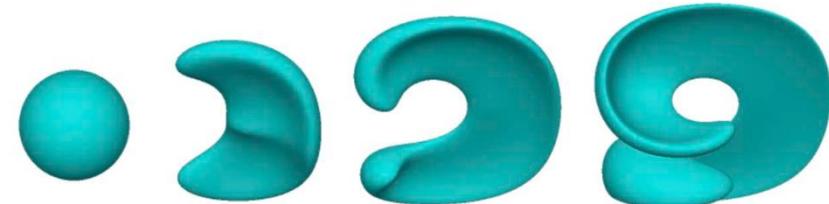
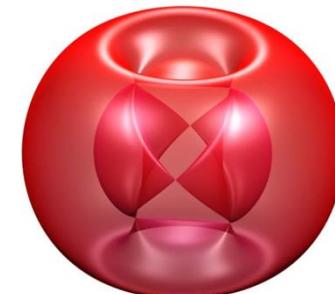
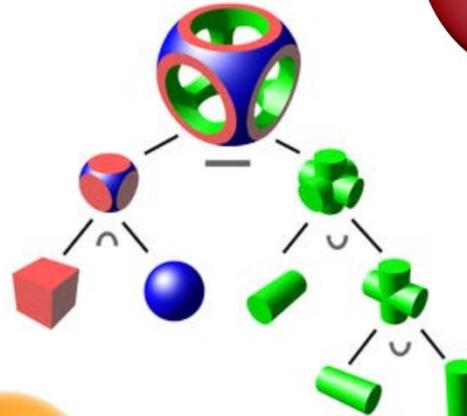
□比如说，单位球面上的所有点满足： $x^2 + y^2 + z^2 = 1$

□更一般的， $f(x, y, z) = 0$  (几何具体的位置)



# 图形学中更多的隐式表示法

- 代数曲面 (algebraic surfaces)
- 构造实体几何 (constructive solid geometry)
- 级别集方法 (level set methods)
- 滴状曲面 (blobby surfaces)
- 分形 (fractals)
- ...

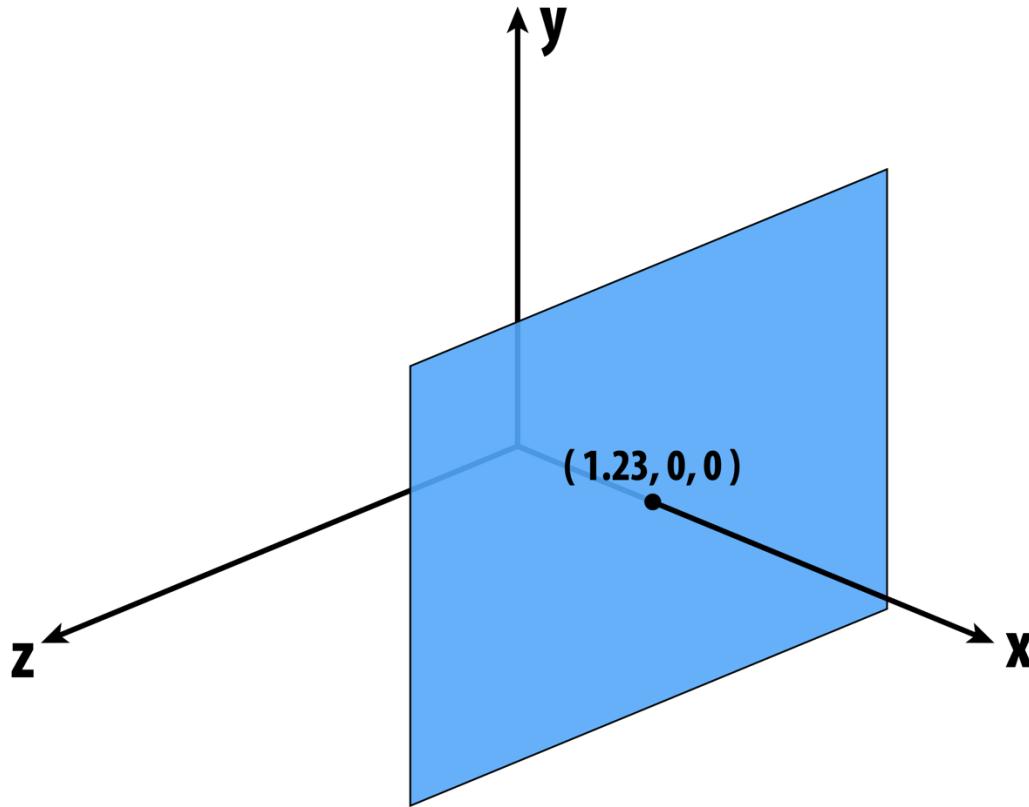


(Will see some of these a bit later.)

让我们设想一个满足  
 $f(x, y, z) = 0$  的平面，并  
找到平面上的任意一点

比如说,  $f(x, y, z) = x - 1.23$

口满足  $x = 1.23$  的点均在此平面上



口观察：隐式曲面会使一些任务变得困难，比如采样

假设我有一个新的平面  
 $f(x, y, z) = x^2 + y^2 + z^2 - 1$ ,  
我想知道某个点是否在其内部

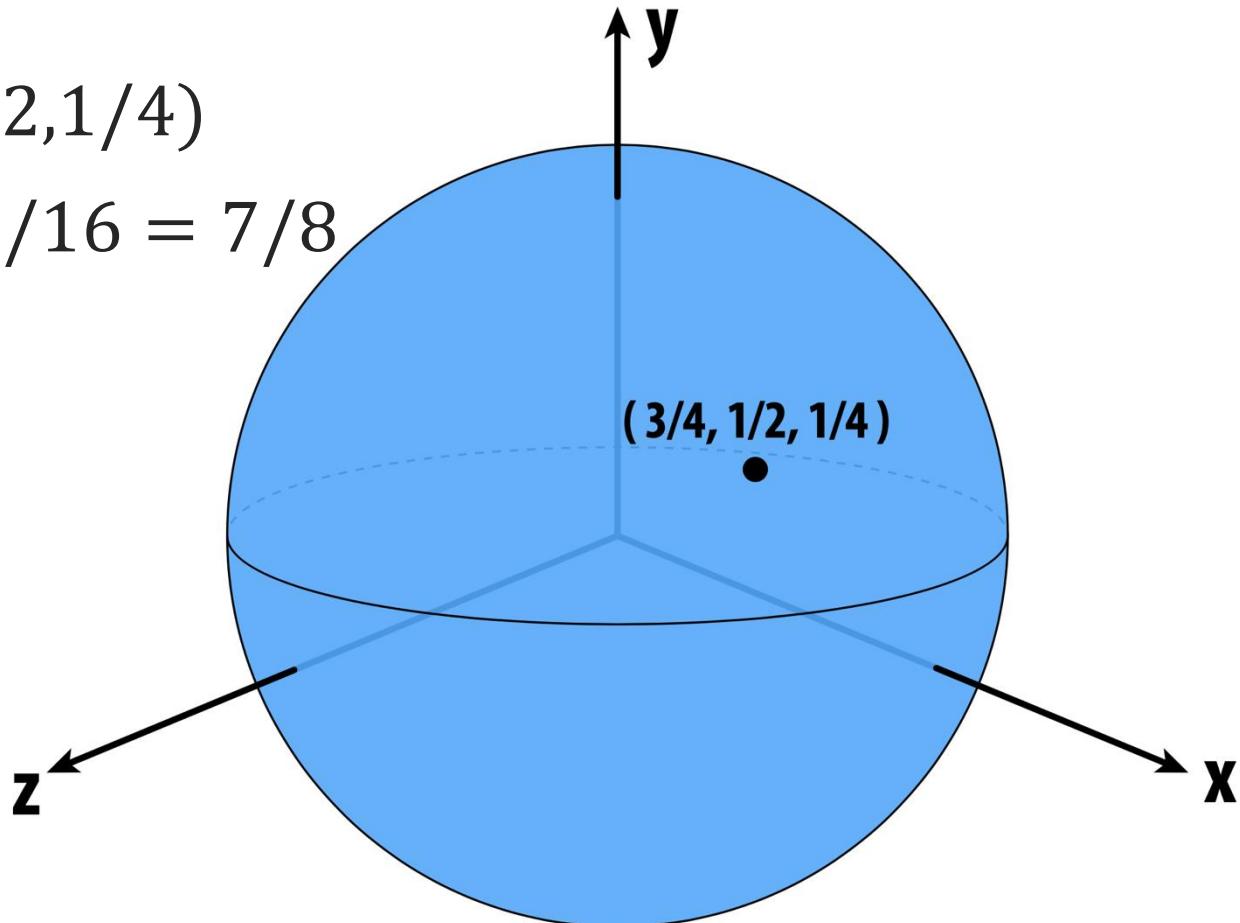
# 检查某点是否在单位球体内

□ 比如说点  $(3/4, 1/2, 1/4)$

□  $9/16 + 4/16 + 1/16 = 7/8$

□  $7/8 < 1$

□ Yes.



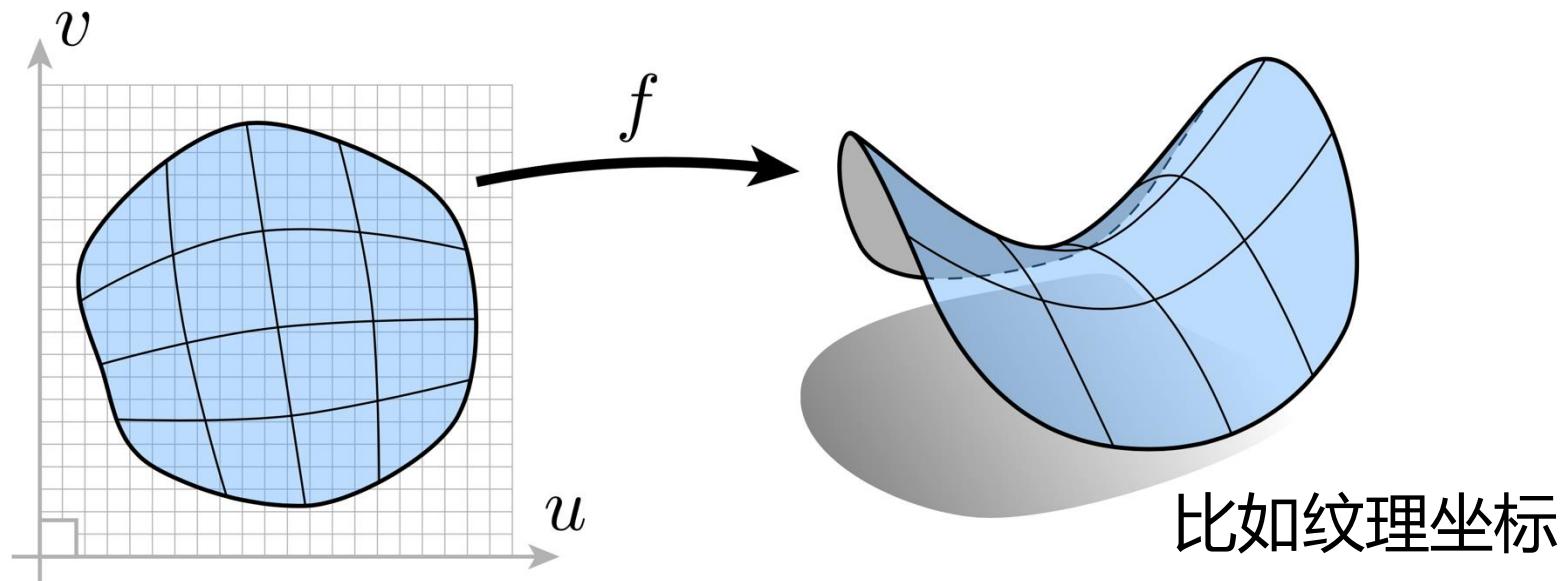
□ 隐式曲面让另一些任务变得简单，比如  
判断点是否在几何内部/外部

# 几何图形的显式表示法

所有点都直接给出

比如，球体上的点为  $(\cos(u) \sin(v), \sin(u) \sin(v), \cos(v))$ ,  
for  $0 \leq u < 2\pi$  and  $0 \leq v \leq \pi$

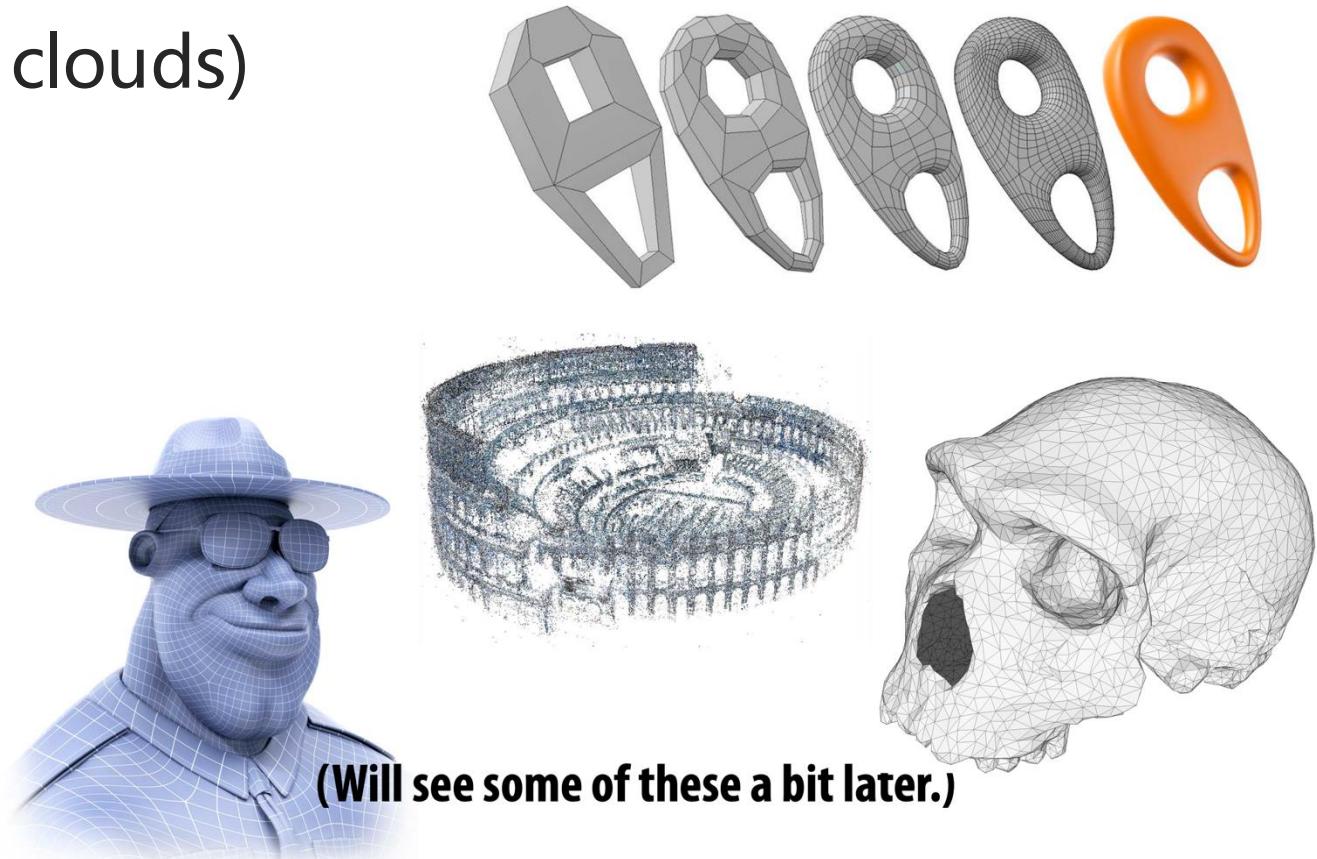
更一般地:  $f : \mathbb{R}^2 \rightarrow \mathbb{R}^3; (u, v) \mapsto (x, y, z)$



可能有很多类似的映射，比如每个三角形一个

# 图形学中更多的显式表示法

- 三角形网格 (triangle meshes)
- 多边形网格 (polygon meshes)
- 细分曲面 (subdivision surfaces)
- 点云 (point clouds)
- NURBS
- ...

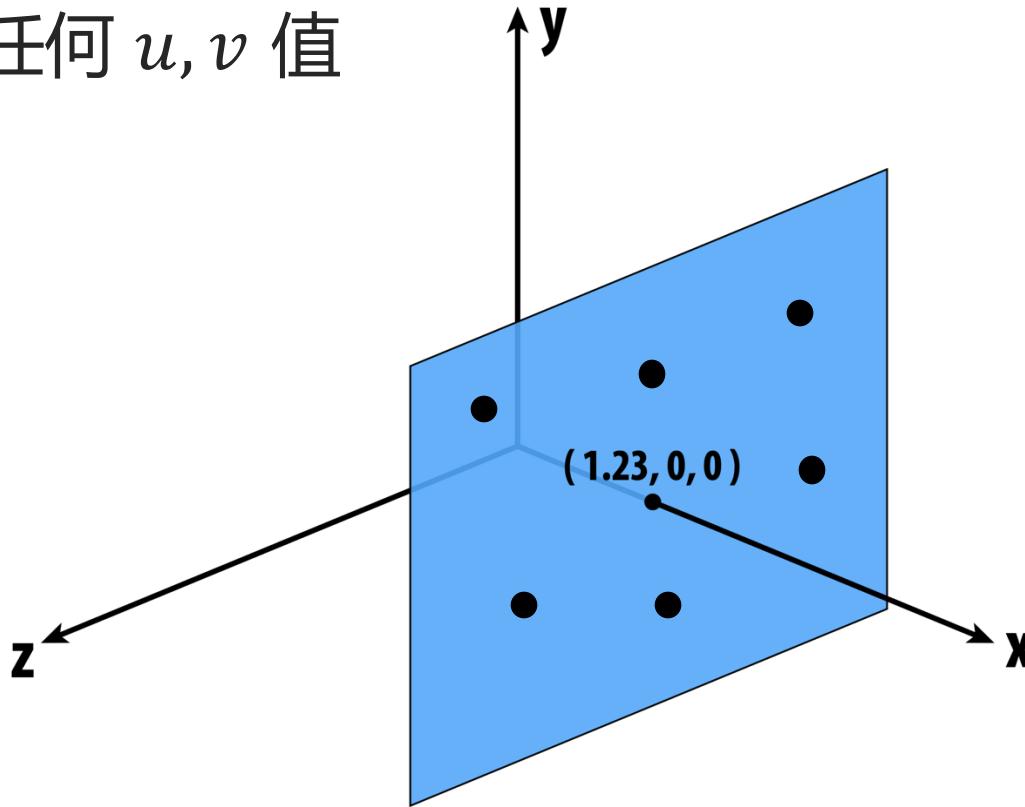


假设我们有一个表面的显式表示，如何找到表面上的某些点？

# 采样一个显式表面

□ 表面为:  $f(u, v) = (1.23, u, v)$

□ 只需要带入任何  $u, v$  值



□ 显式表面让一些任务变得容易, 比如采样

假设我有一个新的表面

$$f(u, v)$$

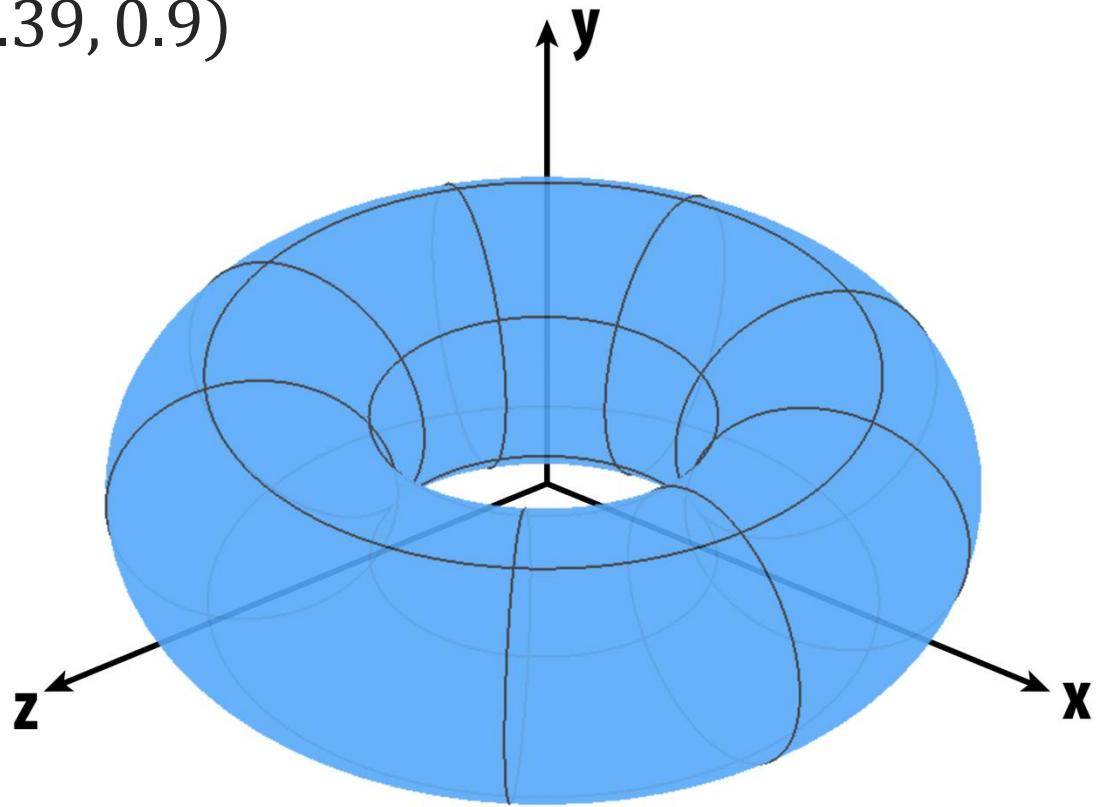
我想知道某个点是否在其内部

# 检查某点是否在环面内

□ 表面为  $f(u, v) = ((2 + \cos u) \cos v, (2 + \cos u) \sin v, \sin u)$

□ 比如说点  $(1.96, -0.39, 0.9)$

□ ...No



□ 显式表面会使另一些任务变得困难，比如内部/外部测试

结论：  
有些表示法比其他表示法效果  
更好 – 取决于具体任务！

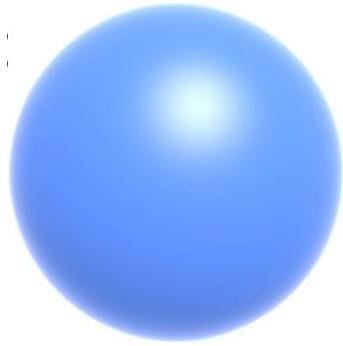
不同的表示法适用于不同类型  
的几何体

让我们来看看计算机图形学中  
使用的一些常见表示

# 代数曲面 (隐式表示)

□ 曲面是  $x, y, z$  多项式的零集 (zero set)

□ 例子



$$x^2 + y^2 + z^2 = 1$$



$$(R - \sqrt{x^2 + y^2})^2 + z^2 = r^2$$



$$(x^2 + \frac{9y^2}{4} + z^2 - 1)^3 =$$

□ 更复杂的形状呢？

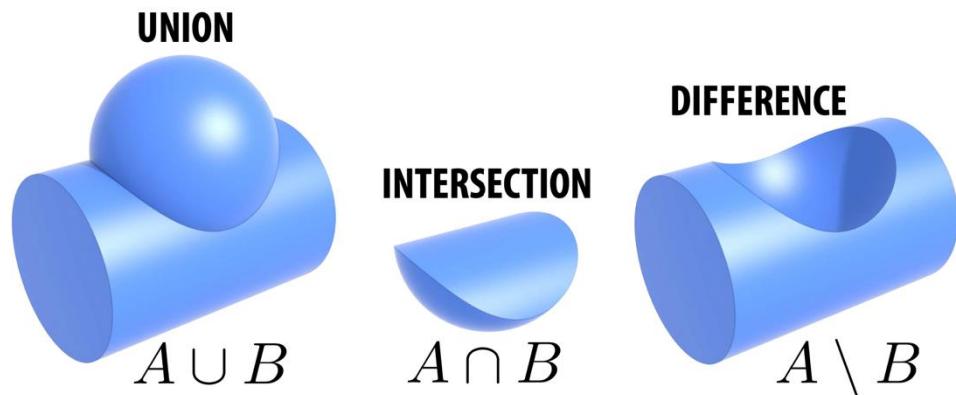
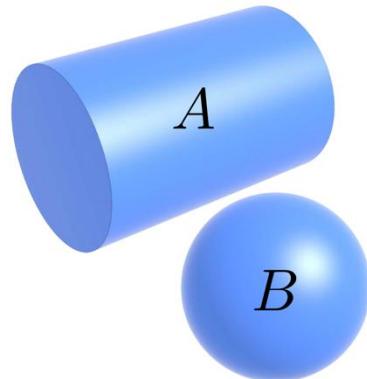


□ 很难找到对应的多项式

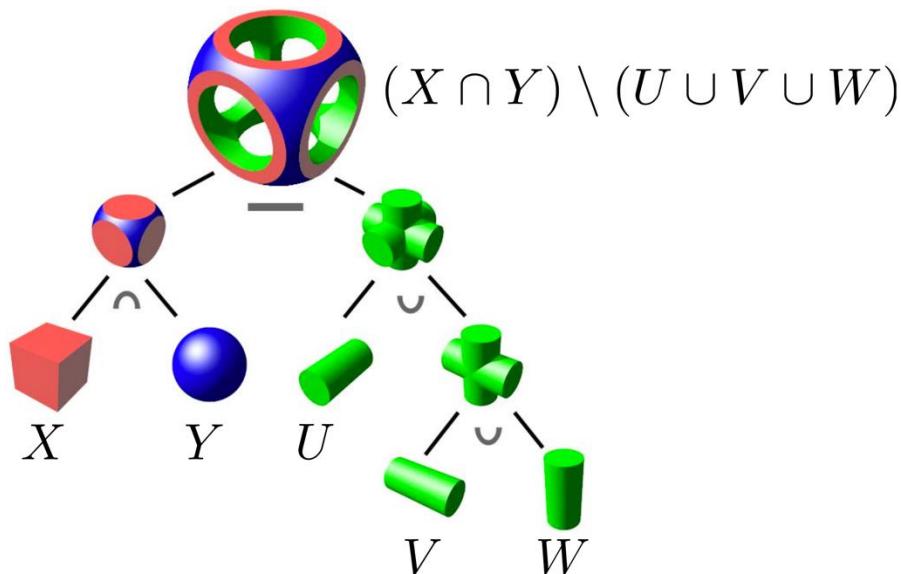
# 构造性立体几何 (隐式表示)

□ 通过布尔运算构建更复杂的形状

□ 基本运算

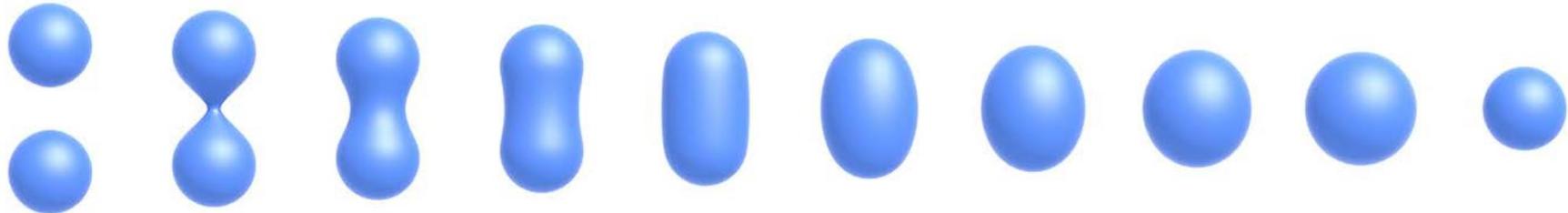


□ 将表达式链接起来



# 滴状曲面 (隐式表示)

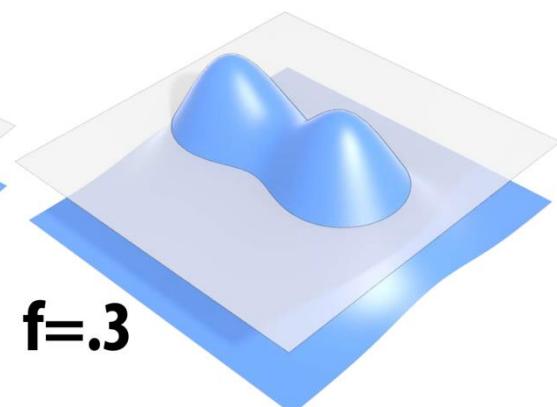
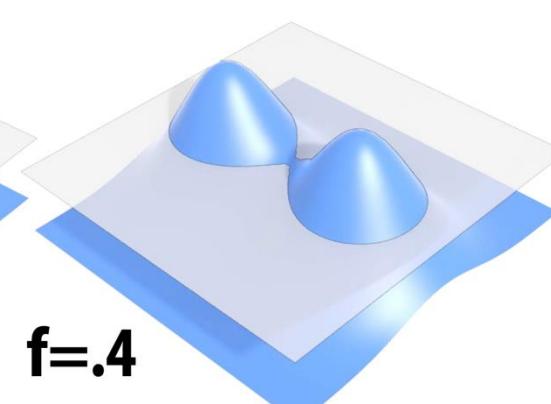
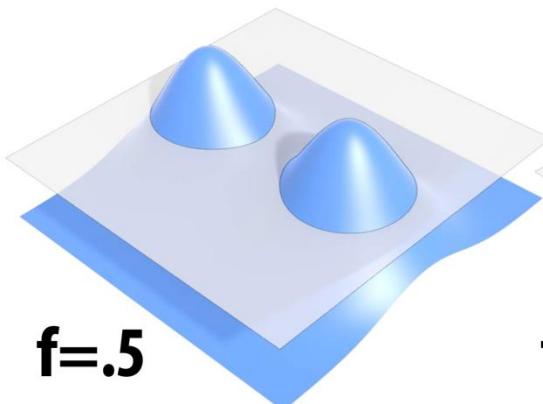
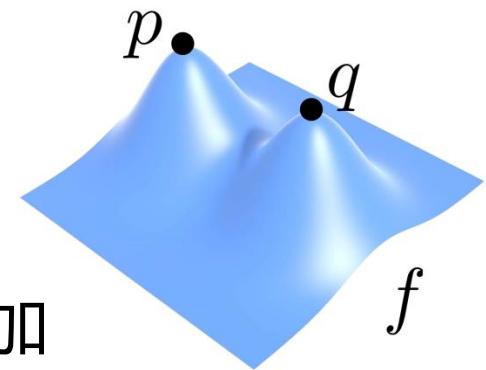
□不是通过布尔操作，而是逐渐将曲面混合在一起



□在 2D 中更好理解

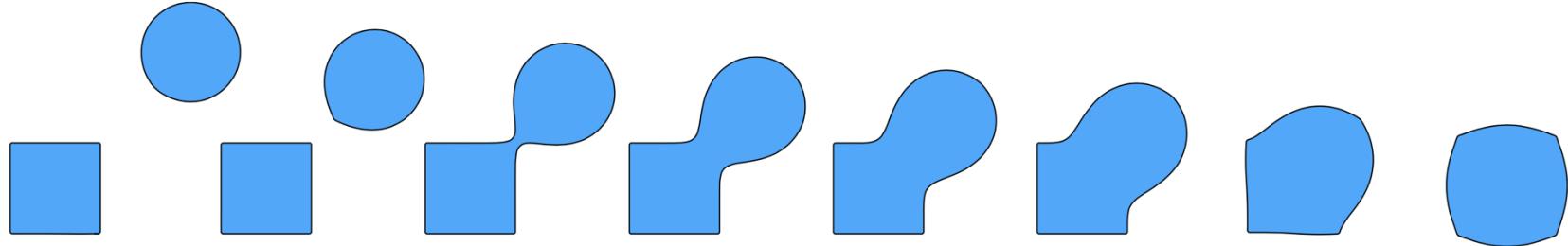
$\phi_p(x) := e^{-|x-p|^2}$  中心点为的  $p$  高斯函数

$f := \phi_p + \phi_q$  不同中心点的高斯函数相加



# 混合距离函数 (隐式表示)

- 距离函数(distance function)计算到对象上最近点的距离
- 可以混合任意两个距离函数  $d_1(x), d_2(x)$



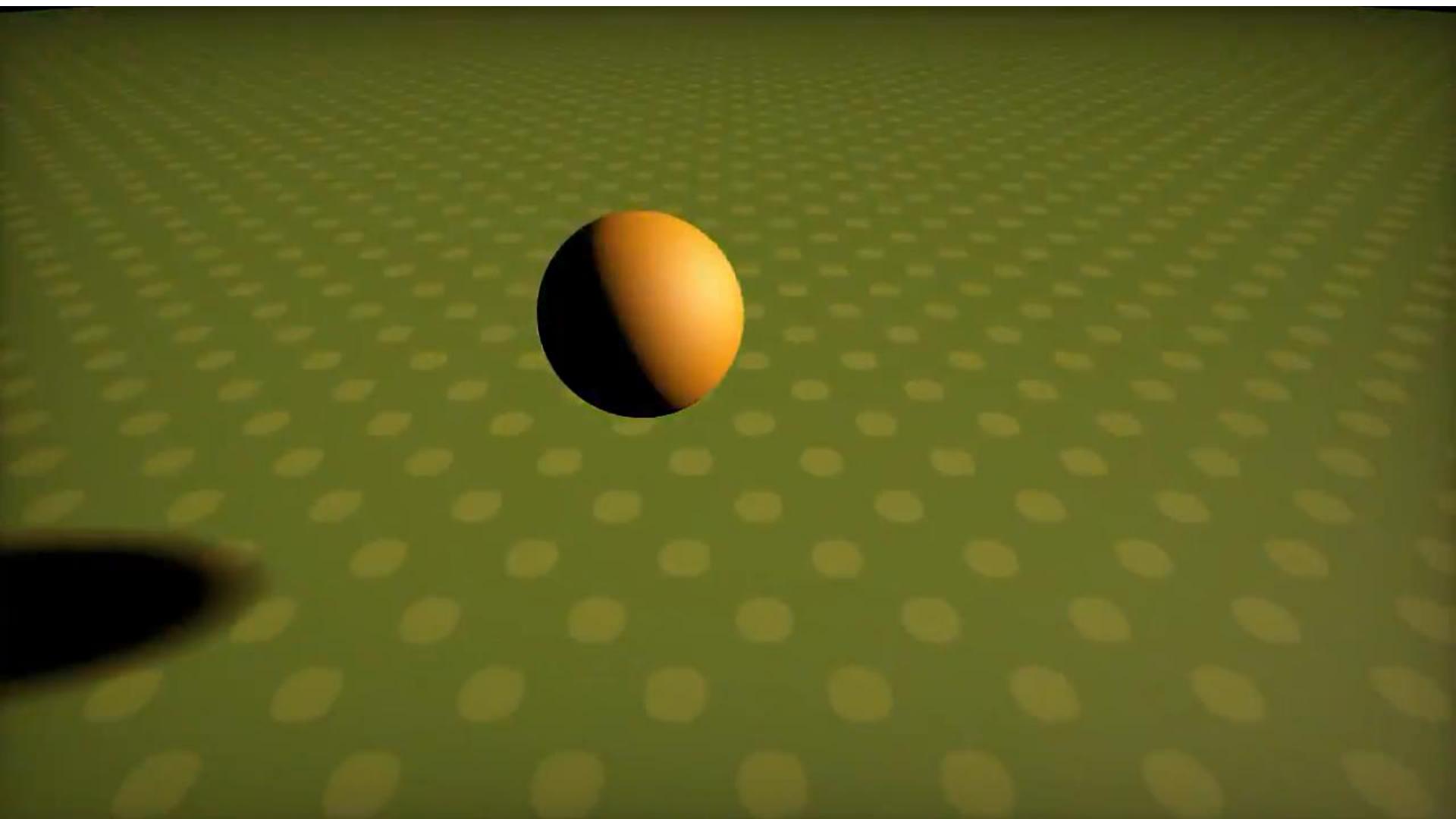
- 类似于前面混合两点的策略，有多种可能，比如

$$f(x) := e^{d_1(x)^2} + e^{d_2(x)^2} - \frac{1}{2}$$

- 外观取决于我们如何组合函数

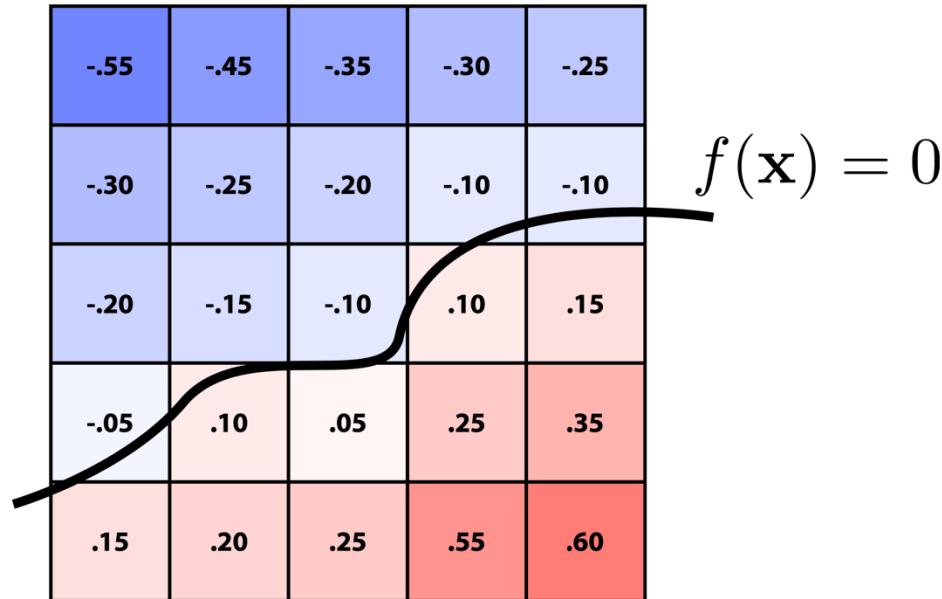
- Q: 如何实现  $d_1(x), d_2(x)$  的布尔联合 (Boolean union)
- A: 取最小值:  $f(x) = \min(d_1(x), d_2(x))$

# 纯距离函数场景 (专家级)



# 水平集方法 (隐式表示)

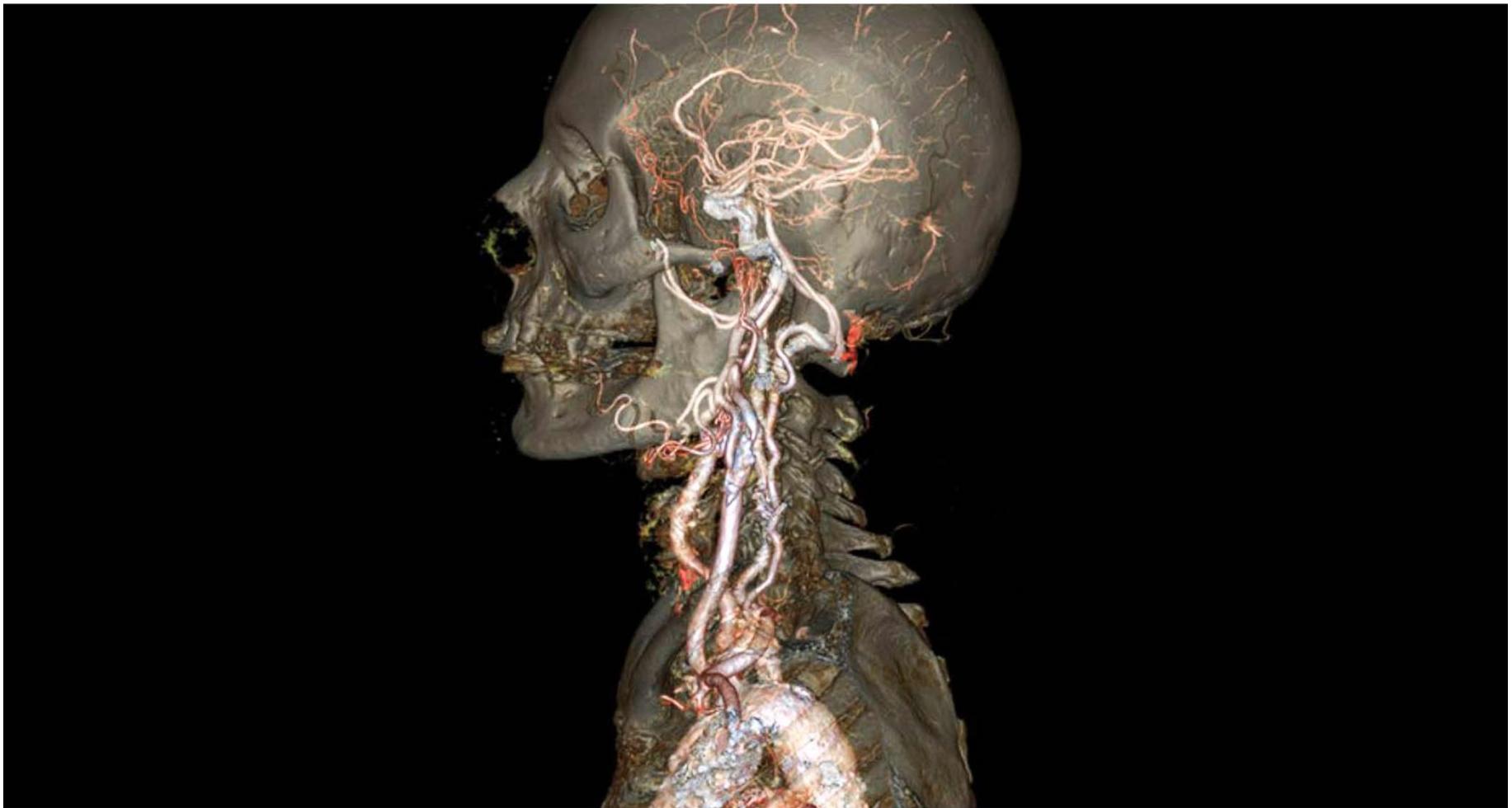
- 口 隐式曲面具有一些很有用的特性 (例如, 合并/拆分)
- 口 但是, 很难用具体的表达式来描述复杂的形状
- 口 另一个方案是, 存储网格值来近似函数



- 口 表面出现在插值为零的地方
- 口 对形状提供更明确的控制 (如纹理)
- 口 与闭式表达式 (closed-form expressions) 不同, 会有走样问题

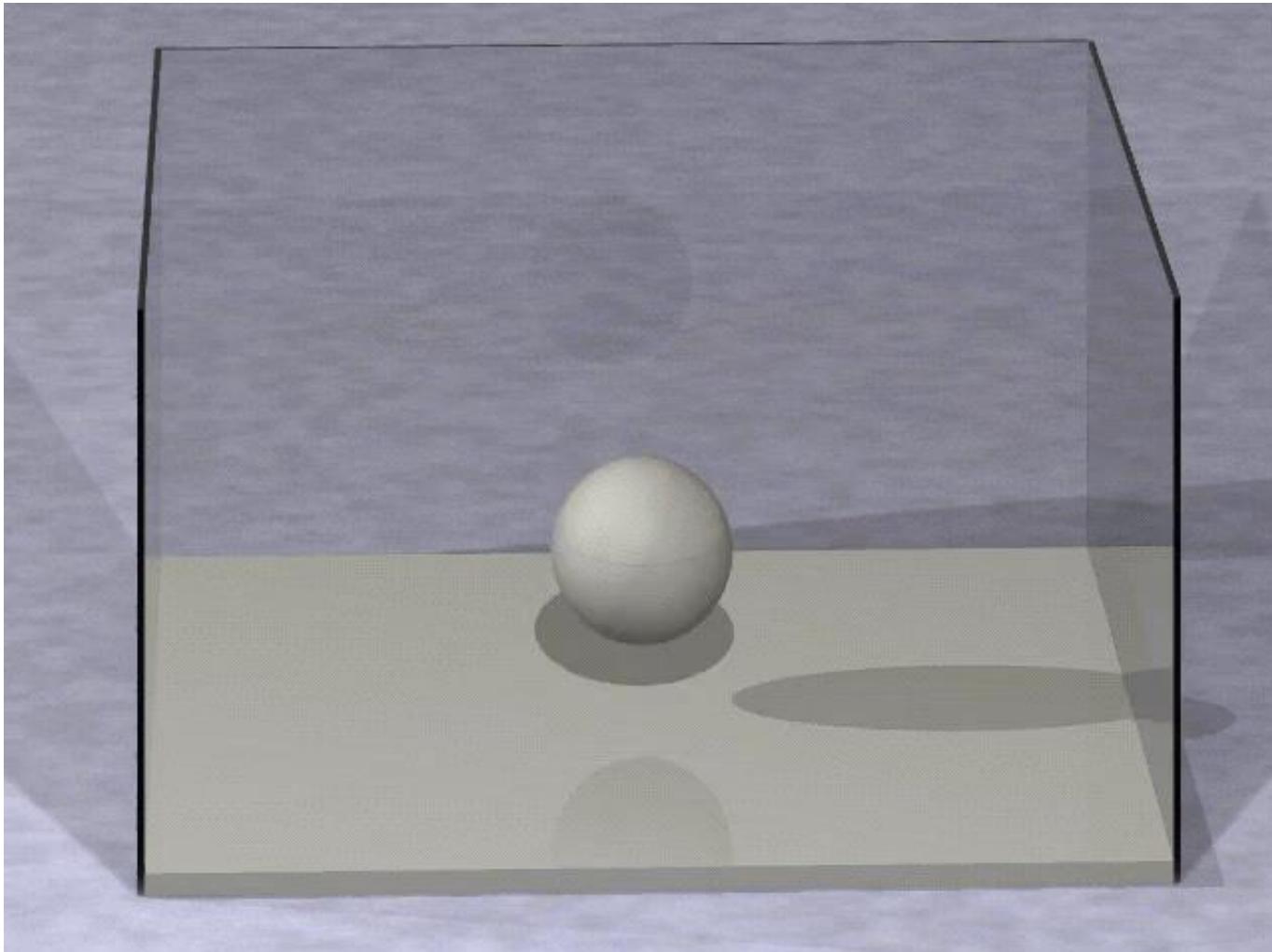
# 来自医学数据 (CT, MRI, etc.) 的水平集

□ 水平集编码恒定的组织密度等



# 物理模拟中的水平集

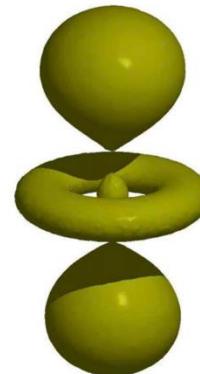
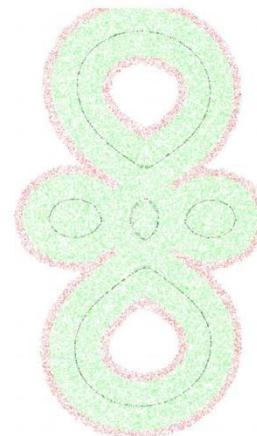
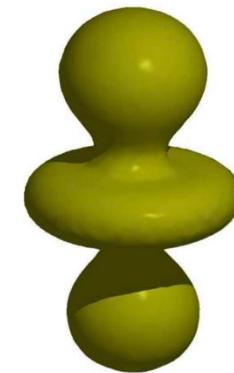
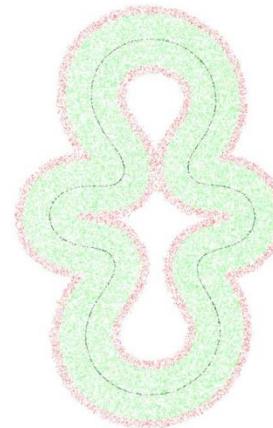
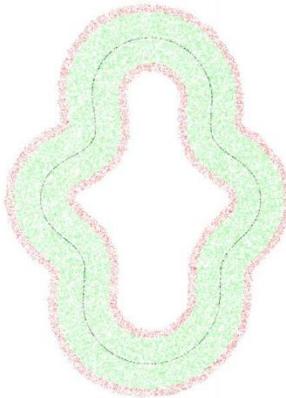
□ 水平集编码来到空气-液体边界的距离



# 水平集存储

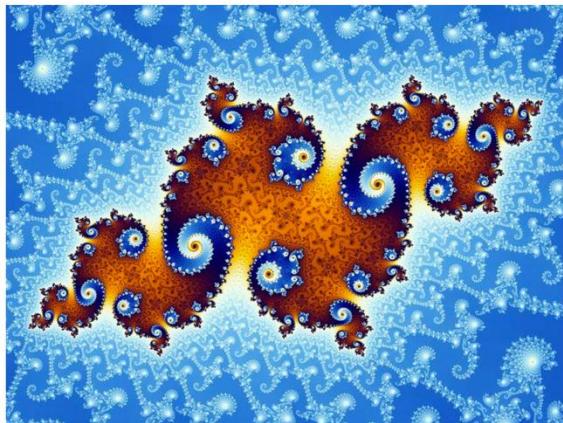
□ 缺点：2D 曲面的存储为  $O(n^3)$

□ 可以通过仅存储表面周围的窄带来降低成本

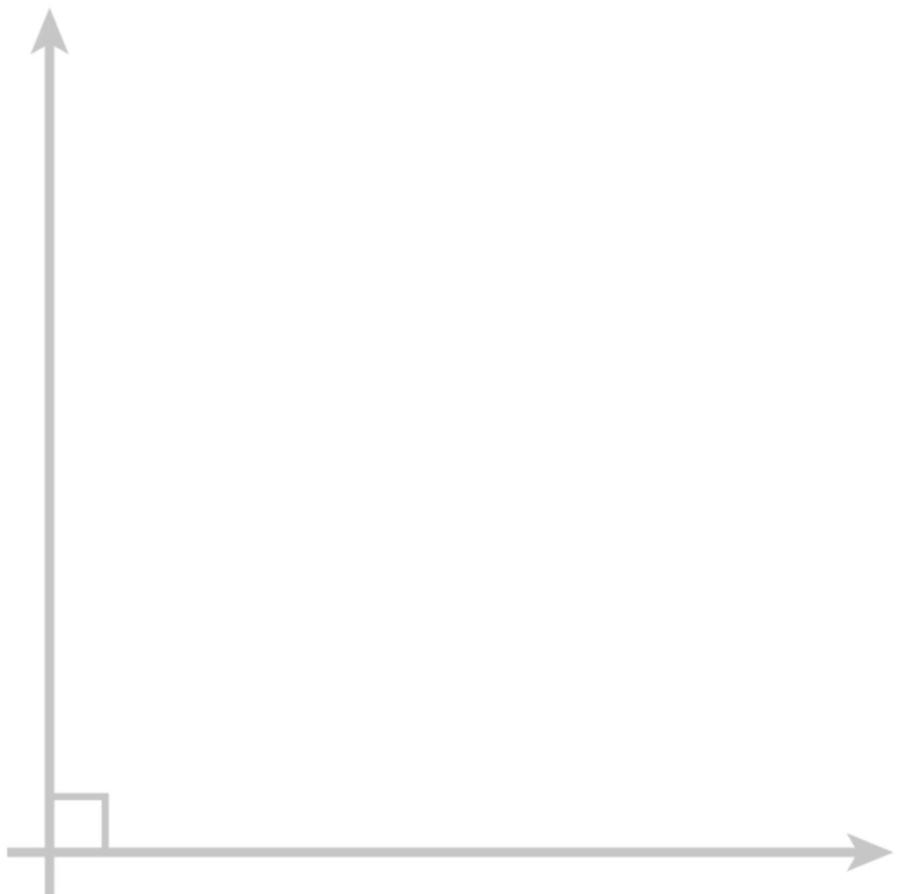


# 分形 (隐式表示)

- 无精确定义；呈现自相似性，在所有尺度上都表现出细节
- 描述自然现象的新“语言”
- 虽然有明确的数学表示，但是形状难以控制！

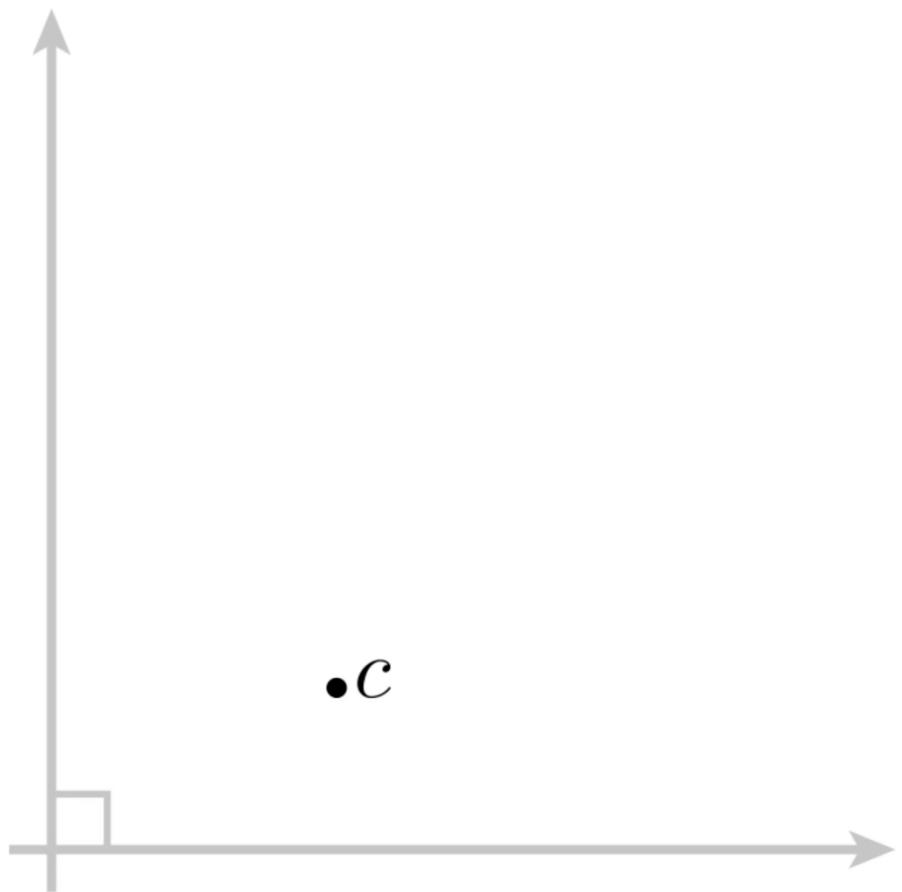


# 曼德布罗特集 (Mandelbrot Set) – 定义



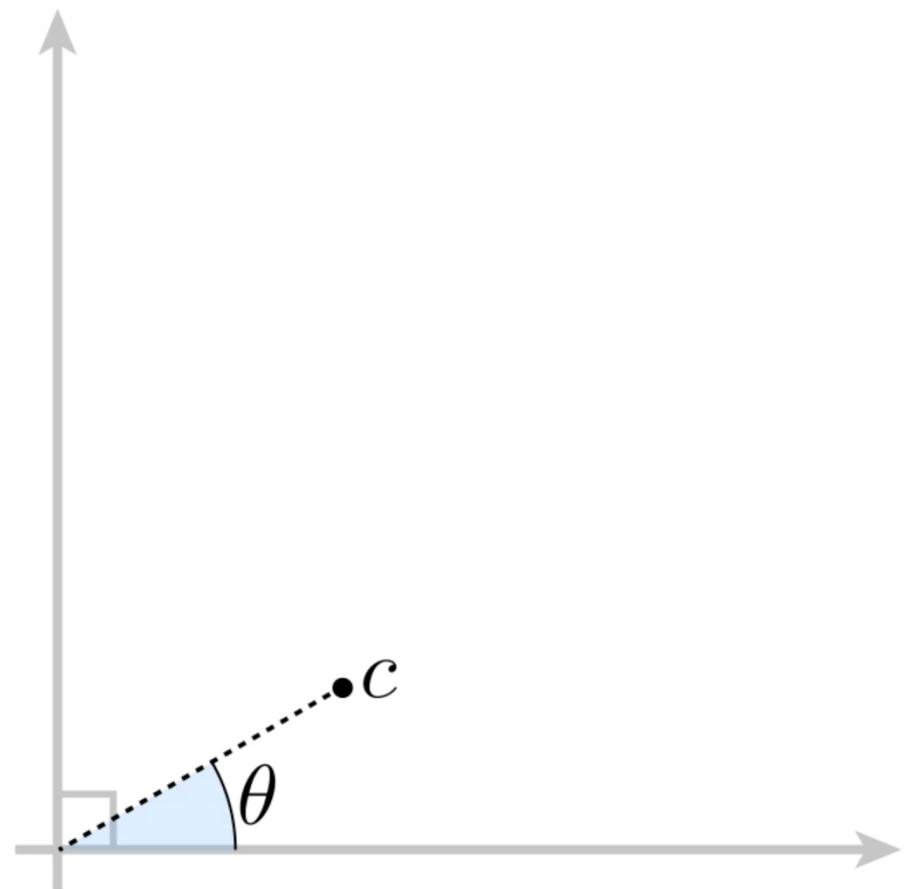
# 曼德布罗特集 (Mandelbrot Set) – 定义

□对于平面上的每个点  $c$



# 曼德布罗特集 (Mandelbrot Set) – 定义

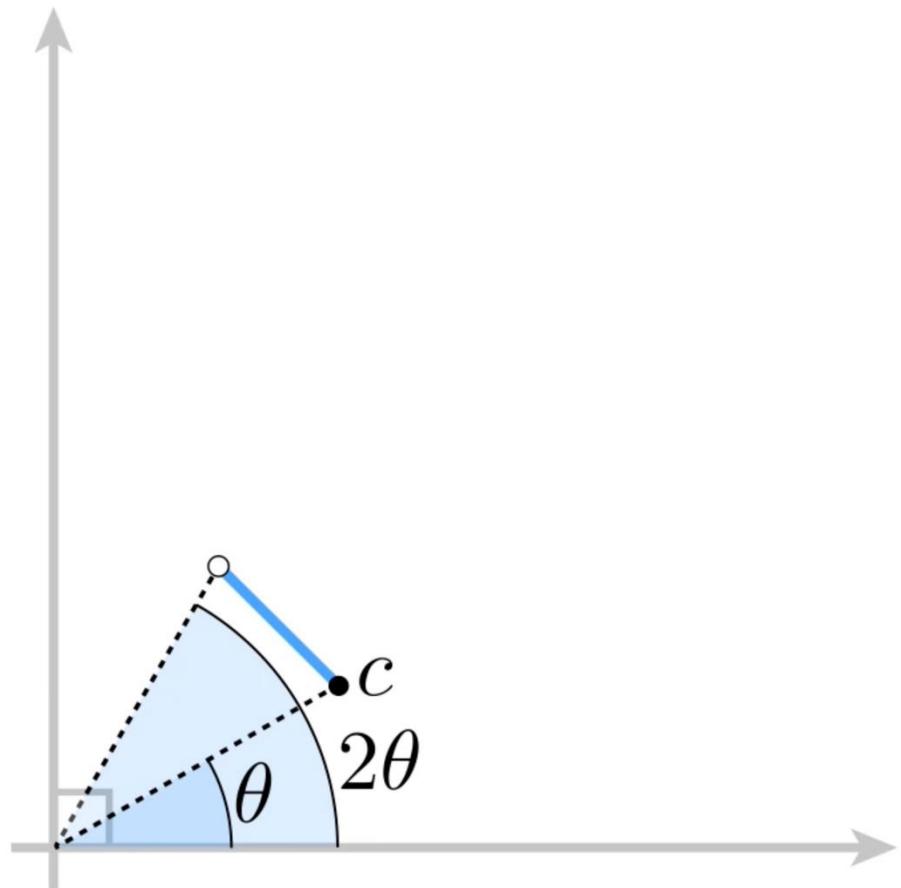
□ 对于平面上的每个点  $c$



# 曼德布罗特集 (Mandelbrot Set) – 定义

□ 对于平面上的每个点  $c$

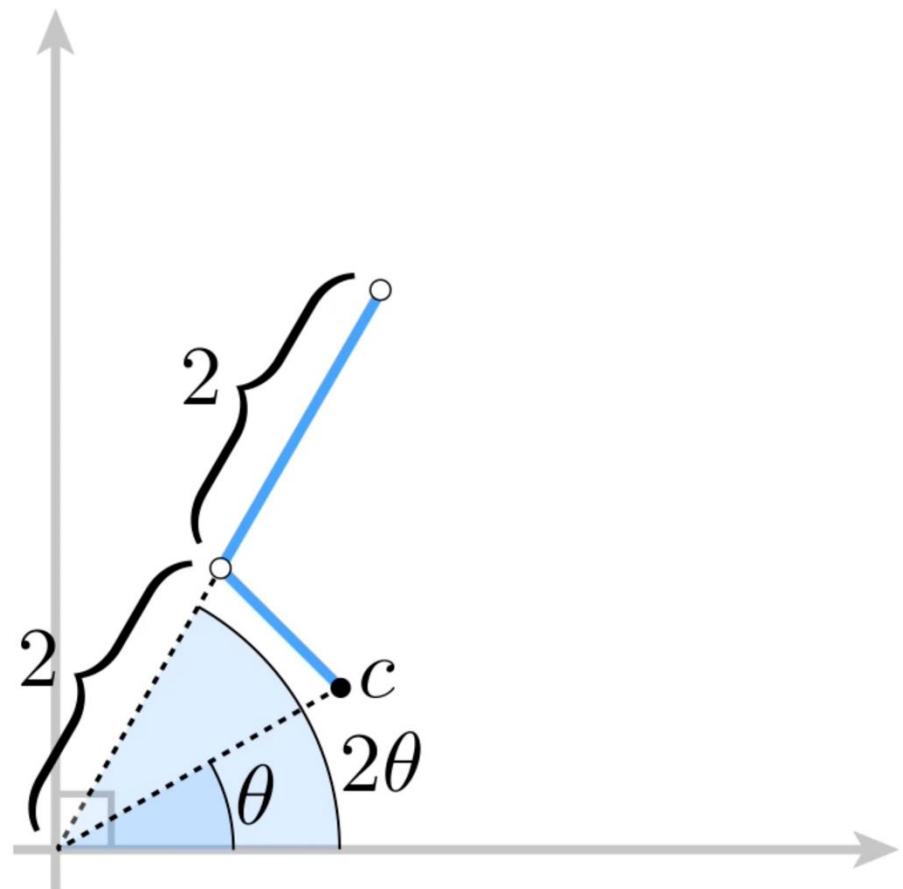
- double the angle



# 曼德布罗特集 (Mandelbrot Set) – 定义

□ 对于平面上的每个点  $c$

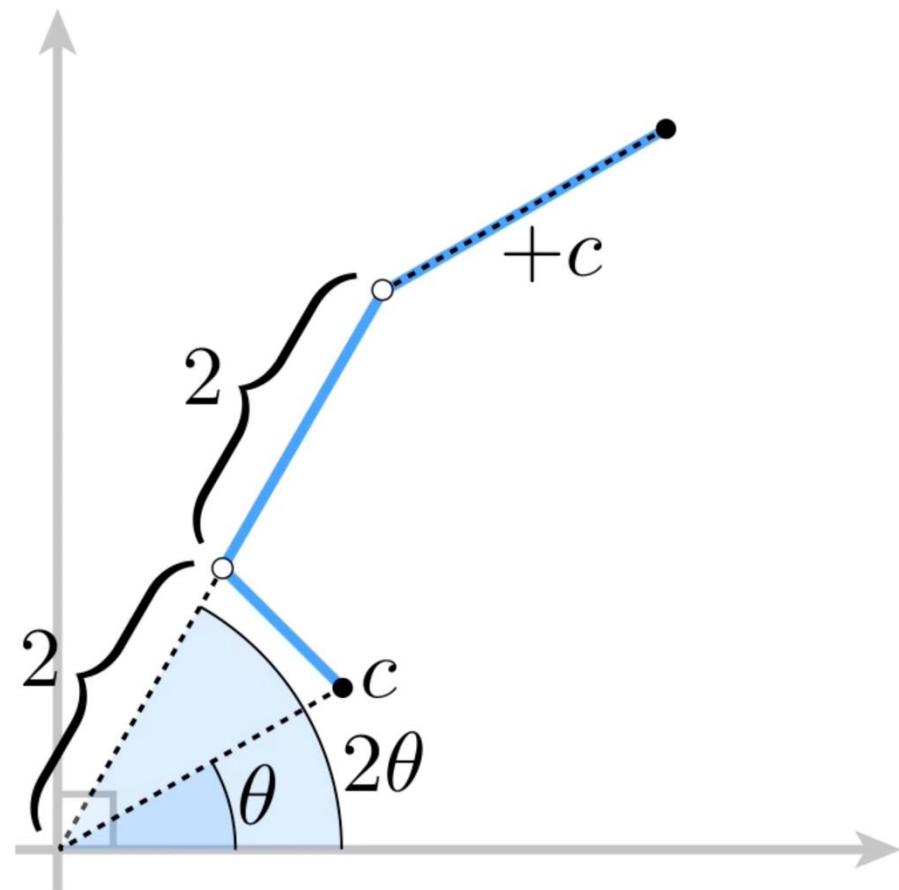
- double the angle
- square the magnitude



# 曼德布罗特集 (Mandelbrot Set) – 定义

对于平面上的每个点  $c$

- double the angle
- square the magnitude
- add the original point  $c$



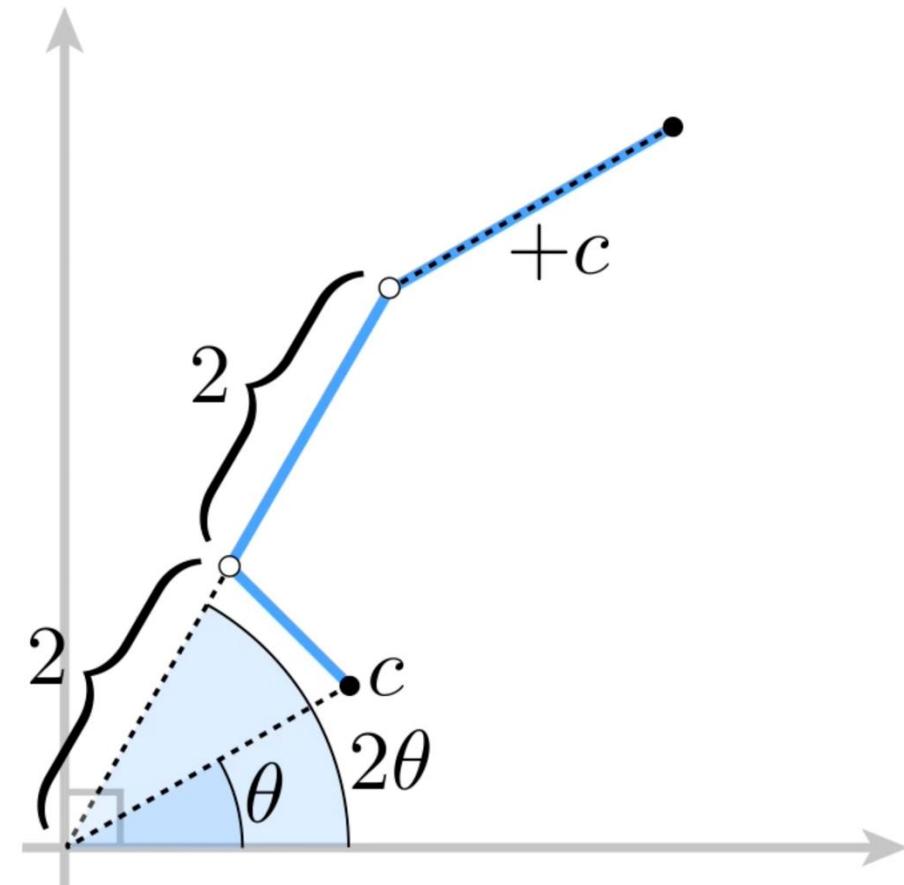
# 曼德布罗特集 (Mandelbrot Set) – 定义

□ 对于平面上的每个点  $c$

- double the angle
- square the magnitude
- add the original point  $c$
- repeat

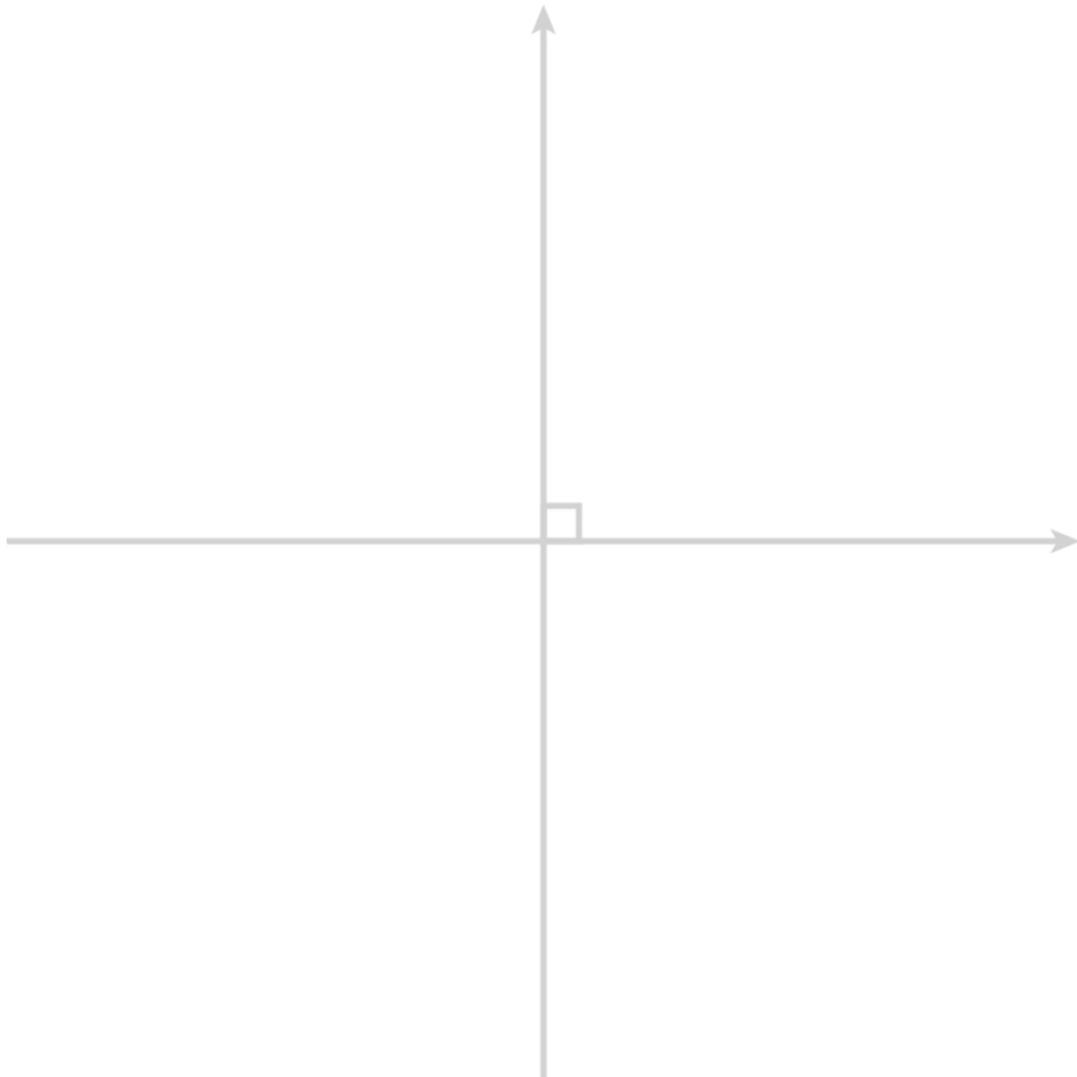
□ 复数表示

- replace  $z$  with  $z^2 + c$
- repeat

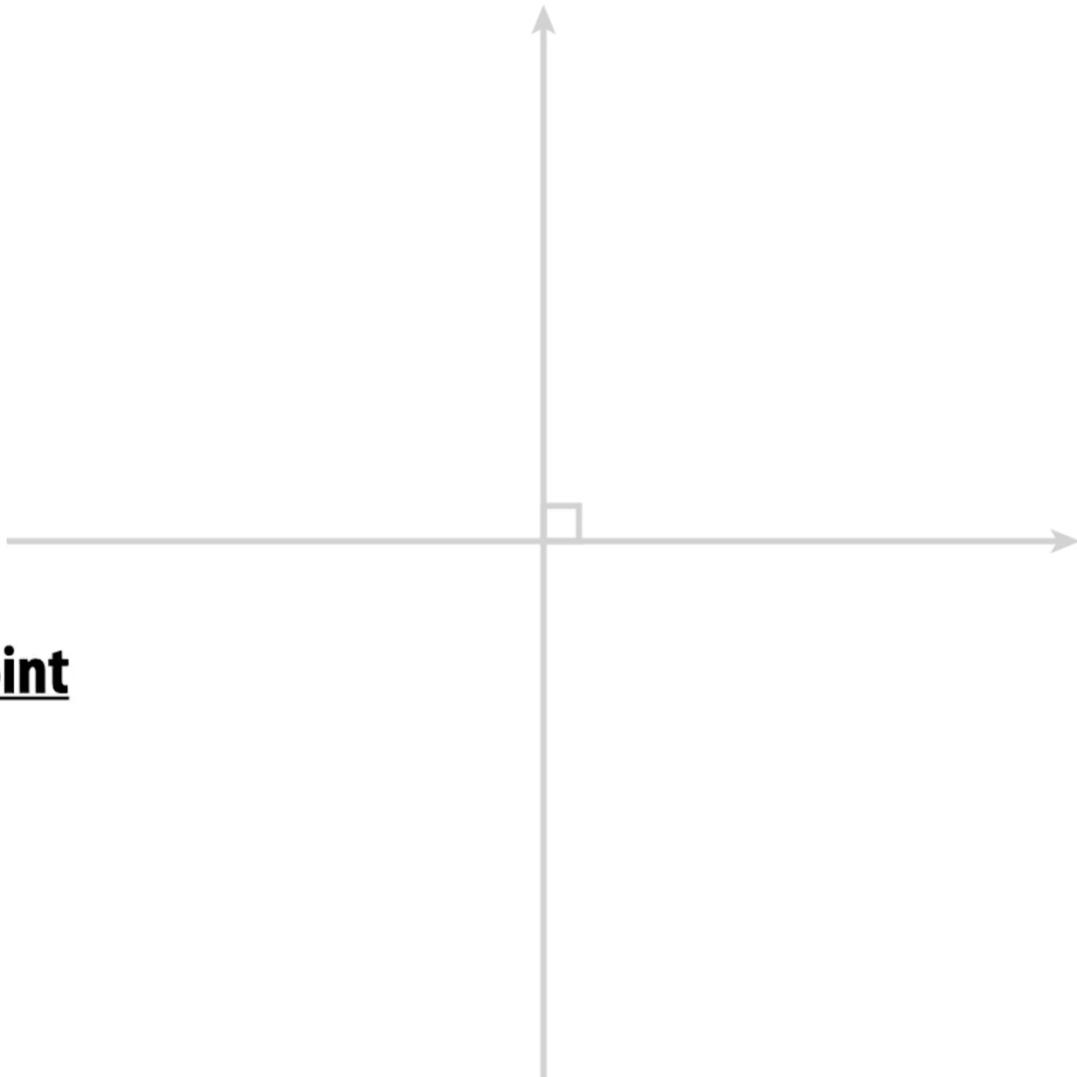


□ 如果 magnitude 保持有界 (不为  $\infty$ )，则它在 Mandelbrot 集合中

# Mandelbrot Set – 例子

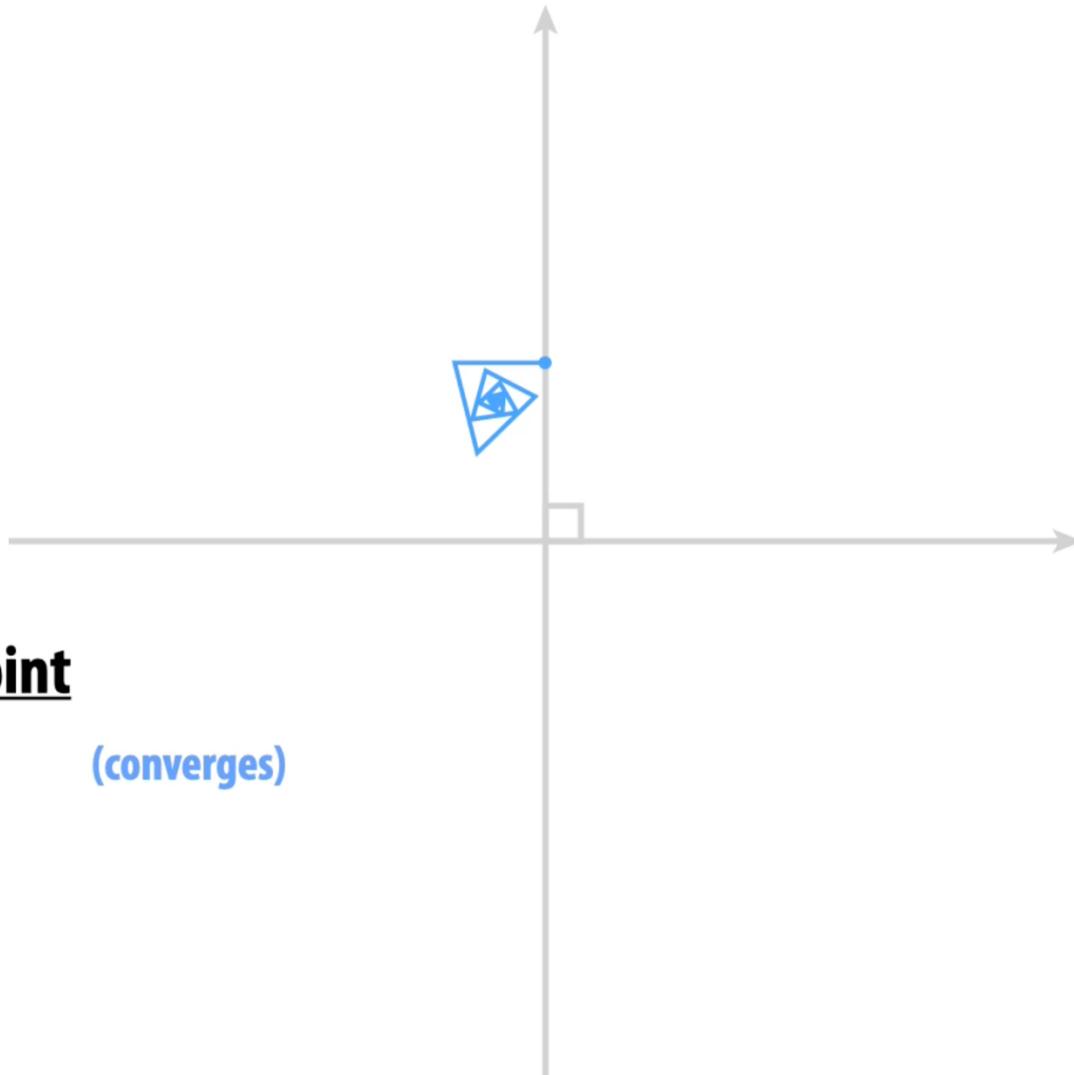


# Mandelbrot Set – 例子



**starting point**

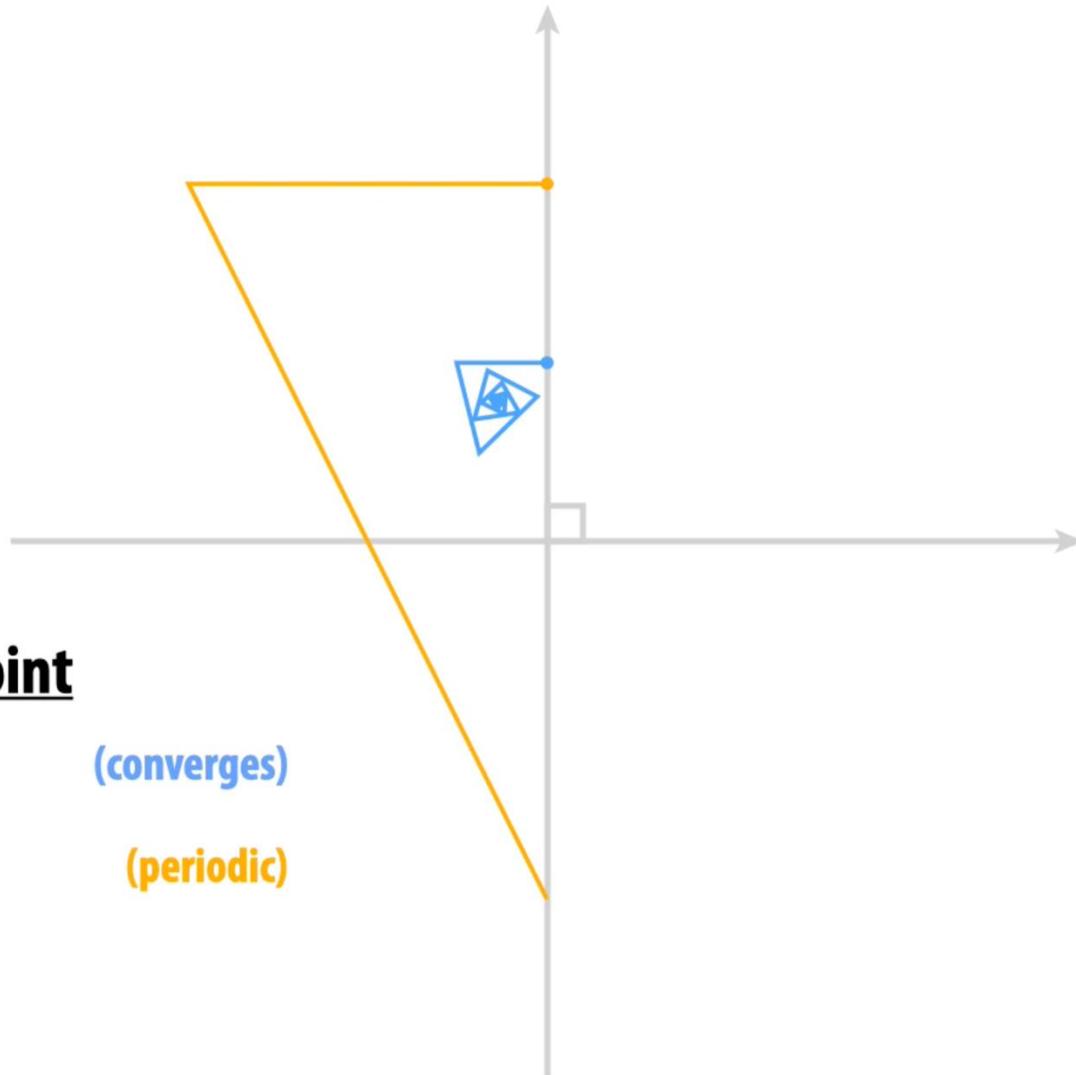
# Mandelbrot Set – 例子



**starting point**

■  $(0, 1/2)$  (converges)

# Mandelbrot Set – 例子

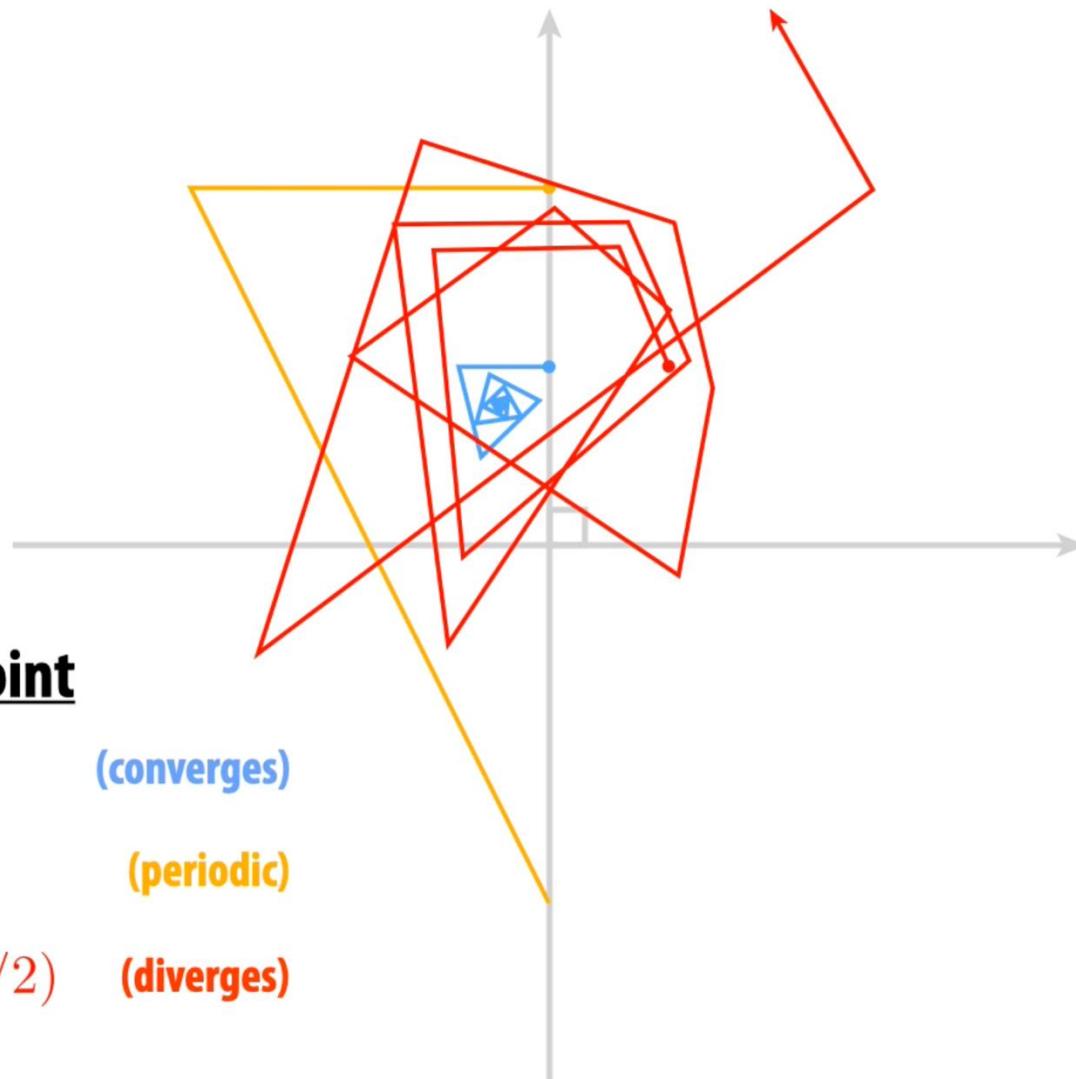


**starting point**

■  $(0, 1/2)$  (converges)

■  $(0, 1)$  (periodic)

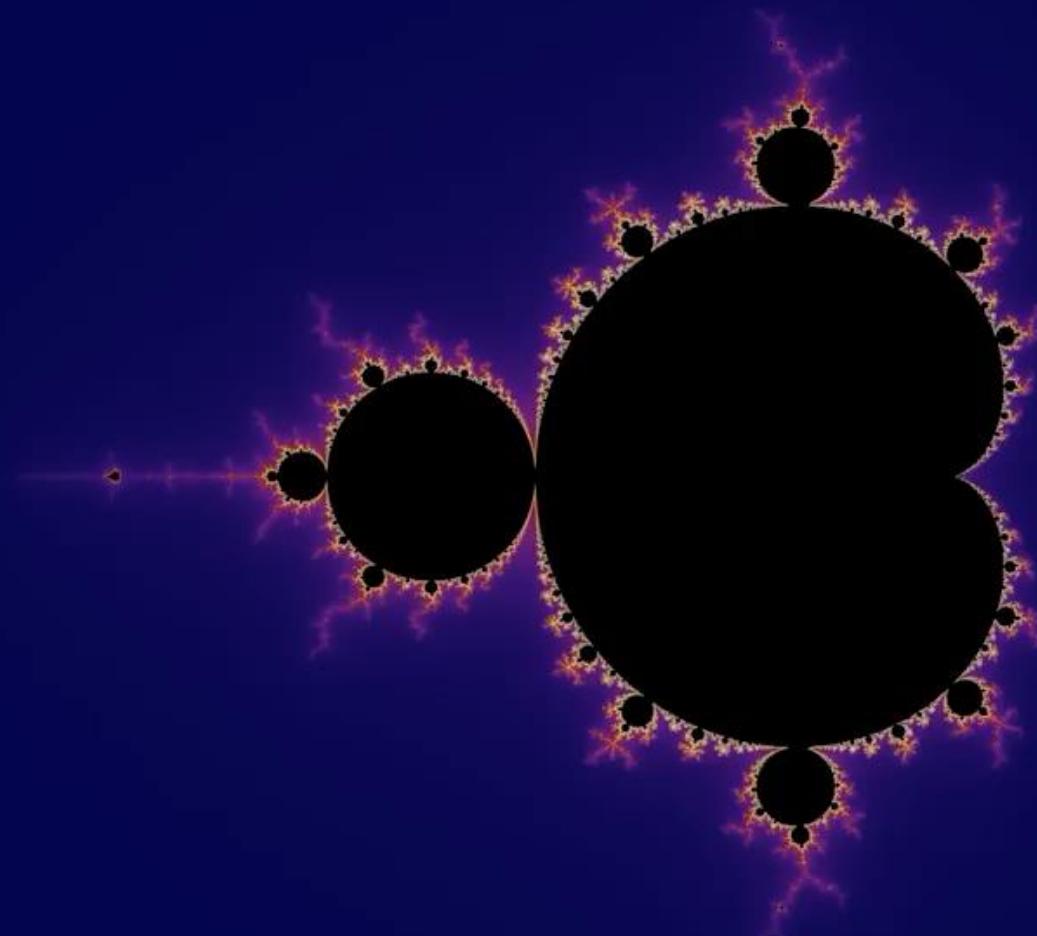
# Mandelbrot Set – 例子



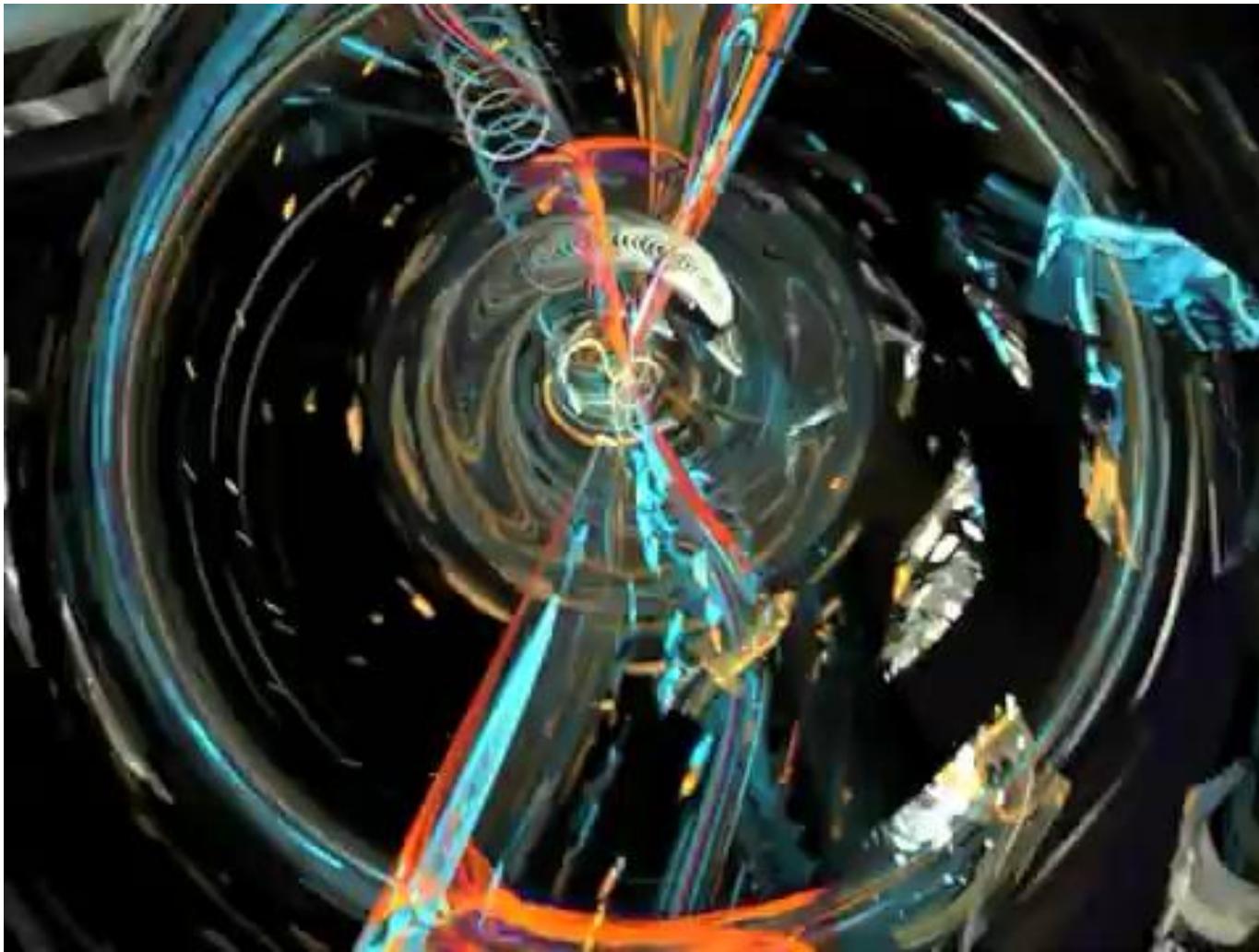
**starting point**

- $(0, 1/2)$  (converges)
- $(0, 1)$  (periodic)
- $(1/3, 1/2)$  (diverges)

# Mandelbrot Set – 放大



# 迭代函数系统 Iterated function systems



Scott Draves (CMU alumn) - see <http://electricsheep.org>

# 隐式表示法的优点和缺点

## 口优点

- 描述可以非常紧凑 (例如, 多项式)
- 很容易确定一个点是否处于集合表面 (只需把点代入)
- 其他查询也可能很容易 (例如, 到表面的距离)
- 对于简单形状, 精确描述/无采样误差
- 易于处理拓扑结构的变化 (例如, 流体)

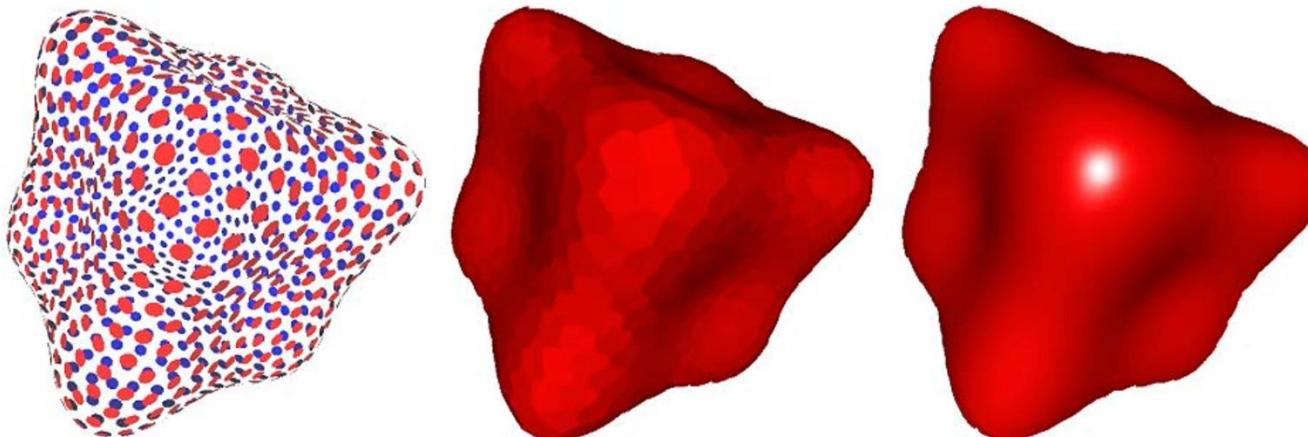
## 口缺点

- 查找形状中的所有点的成本很高 (例如, 用于绘图)
- 很难对复杂形状建模

那显式表示法呢？

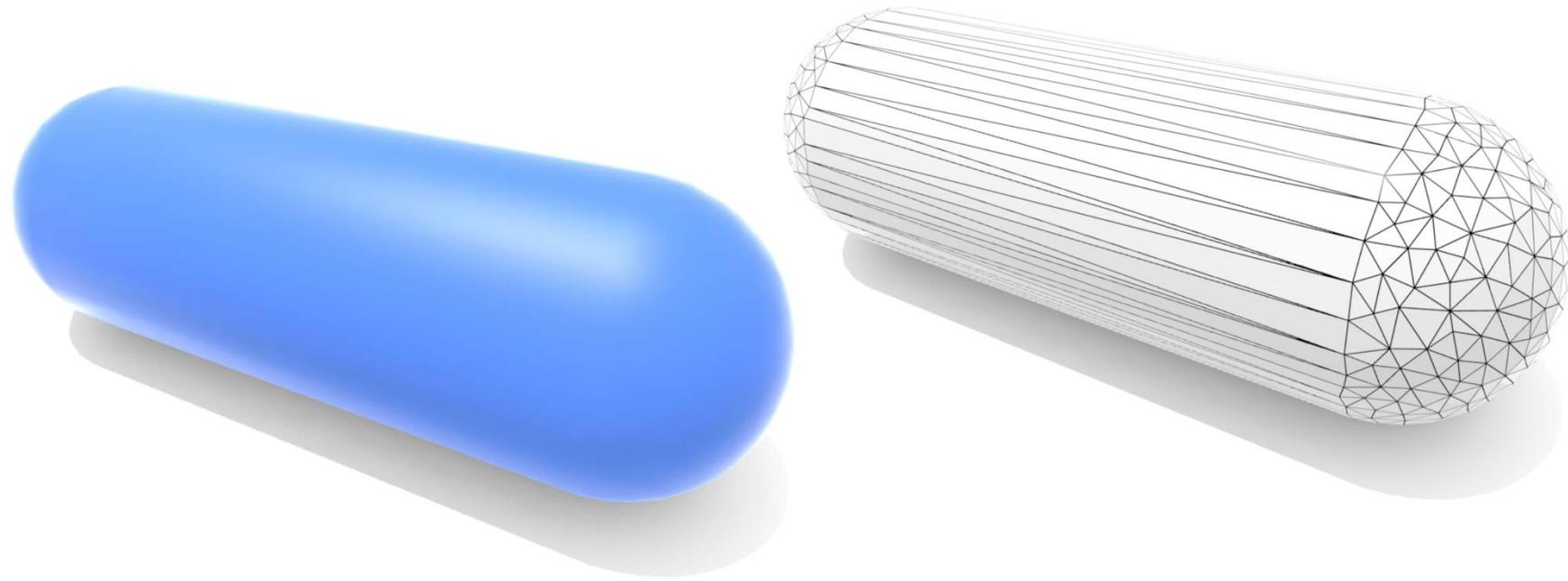
# 点云 Point cloud (显式表达)

- 最简单的表示：点列表  $(x, y, z)$
- 附加其他属性：法线、颜色
- 轻松表示任何类型的几何体
- 易于绘制密集云 ( $>>1$  point/pixel)
- 难以对采样不足的区域进行插值
- 难以处理/模拟/...



# 多边形网格 (显式表达)

- 存储顶点和多边形 (通常为三角形或四边形)
- 更容易进行处理/模拟，自适应采样 (多采样细节处)
- 数据结构更复杂：如何编码连通性，如何与邻居关联
- 不规则的邻居形状



# 三角形网格 (显式表达)

□ 将顶点存储为三元组坐标  $(x, y, z)$

□ 将三角形存储为三元组索引  $(i, j, k)$

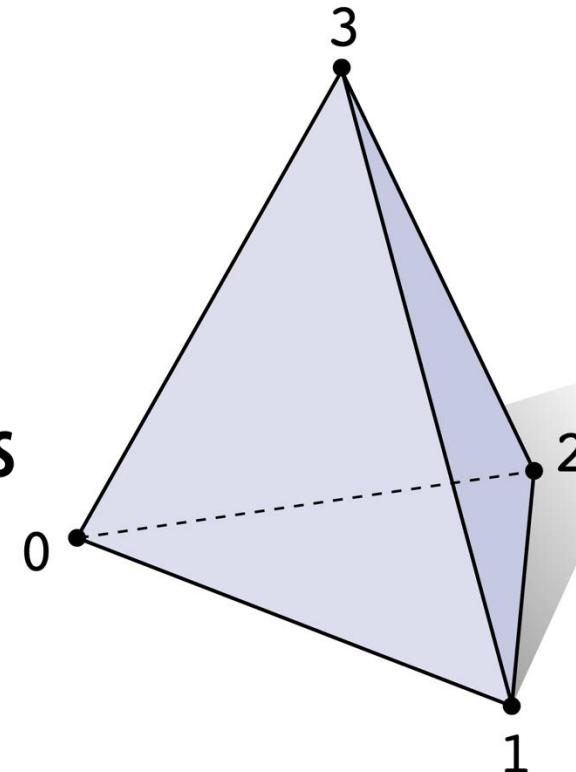
□ 例如四面体

## VERTICES

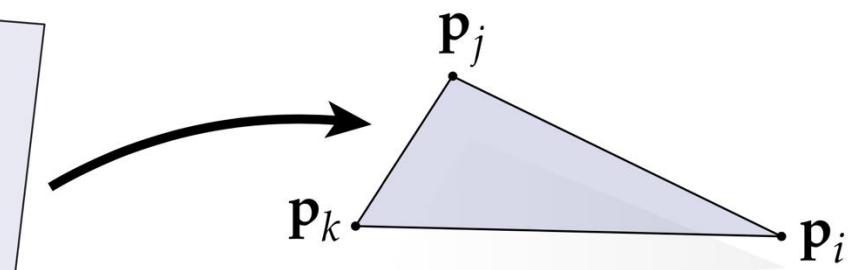
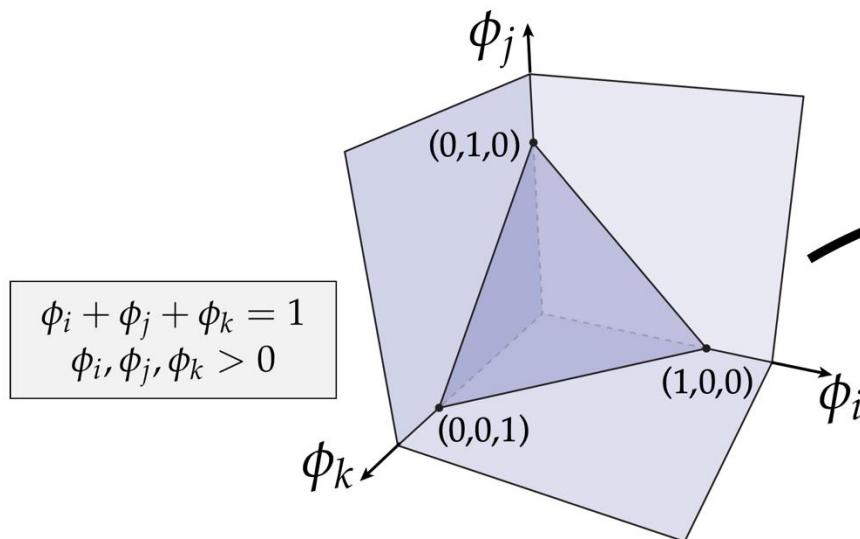
	x	y	z
0:	-1	-1	-1
1:	1	-1	1
2:	1	1	-1
3:	-1	1	1

## TRIANGLES

i	j	k
0	2	1
0	3	2
3	0	1
3	1	2



□ 使用重心插值定义三角形内的其他点



$$p = \phi_i p_i + \phi_j p_j + \phi_k p_k$$

# Wavefront Object File (.obj) 格式

□ 常用于图形学研究中

□ 一个文本文件，指定顶点、法线、纹理坐标及其连接关系

```
1 # This is a comment
2
3 v 1.000000 -1.000000 -1.000000
4 v 1.000000 -1.000000 1.000000
5 v -1.000000 -1.000000 1.000000
6 v -1.000000 -1.000000 -1.000000
7 v 1.000000 1.000000 -1.000000
8 v 0.999999 1.000000 1.000001
9 v -1.000000 1.000000 1.000000
10 v -1.000000 1.000000 -1.000000
11
12 vt 0.748573 0.750412
13 vt 0.749279 0.501284
14 vt 0.999110 0.501077
15 vt 0.999455 0.750380
16 vt 0.250471 0.500702
17 vt 0.249682 0.749677
18 vt 0.001085 0.750380
19 vt 0.001517 0.499994
20 vt 0.499422 0.500239
21 vt 0.500149 0.750166
22 vt 0.748355 0.998230
23 vt 0.500193 0.998728
24 vt 0.498993 0.250415
25 vt 0.748953 0.250920
26
```

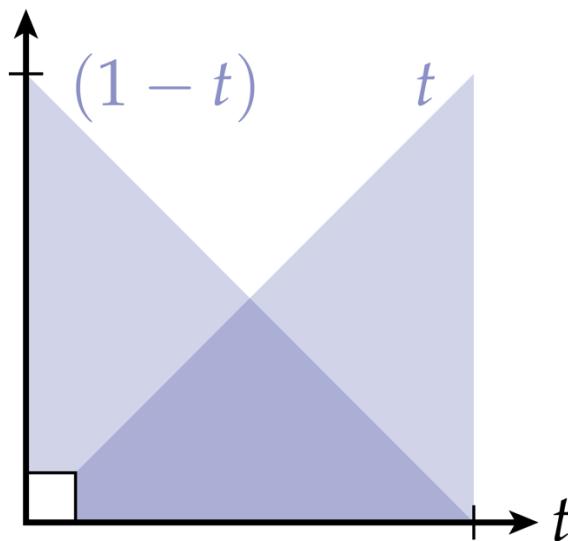
```
vn 0.000000 0.000000 -1.000000
vn -1.000000 -0.000000 -0.000000
vn -0.000000 -0.000000 1.000000
vn -0.00001 0.000000 1.000000
vn 1.000000 -0.000000 0.000000
vn 1.000000 0.000000 0.000001
vn 0.000000 1.000000 -0.000000
vn -0.000000 -1.000000 0.000000
35
36 f 5/1/1 1/2/1 4/3/1
37 f 5/1/1 4/3/1 8/4/1
38 f 3/5/2 7/6/2 8/7/2
39 f 3/5/2 8/7/2 4/8/2
40 f 2/9/3 6/10/3 3/5/3
41 f 6/10/4 7/6/4 3/5/4
42 f 1/2/5 5/1/5 2/9/5
43 f 5/1/6 6/10/6 2/9/6
44 f 5/1/7 8/11/7 6/10/7
45 f 8/11/7 7/12/7 6/10/7
46 f 1/2/8 2/9/8 3/13/8
47 f 1/2/8 3/13/8 4/14/8
```

# 回顾：线性插值 (1D)

□ 使用线性插值法进行插值；在 1D 中：

$$\hat{f}(t) = (1 - t)f_i + tf_j$$

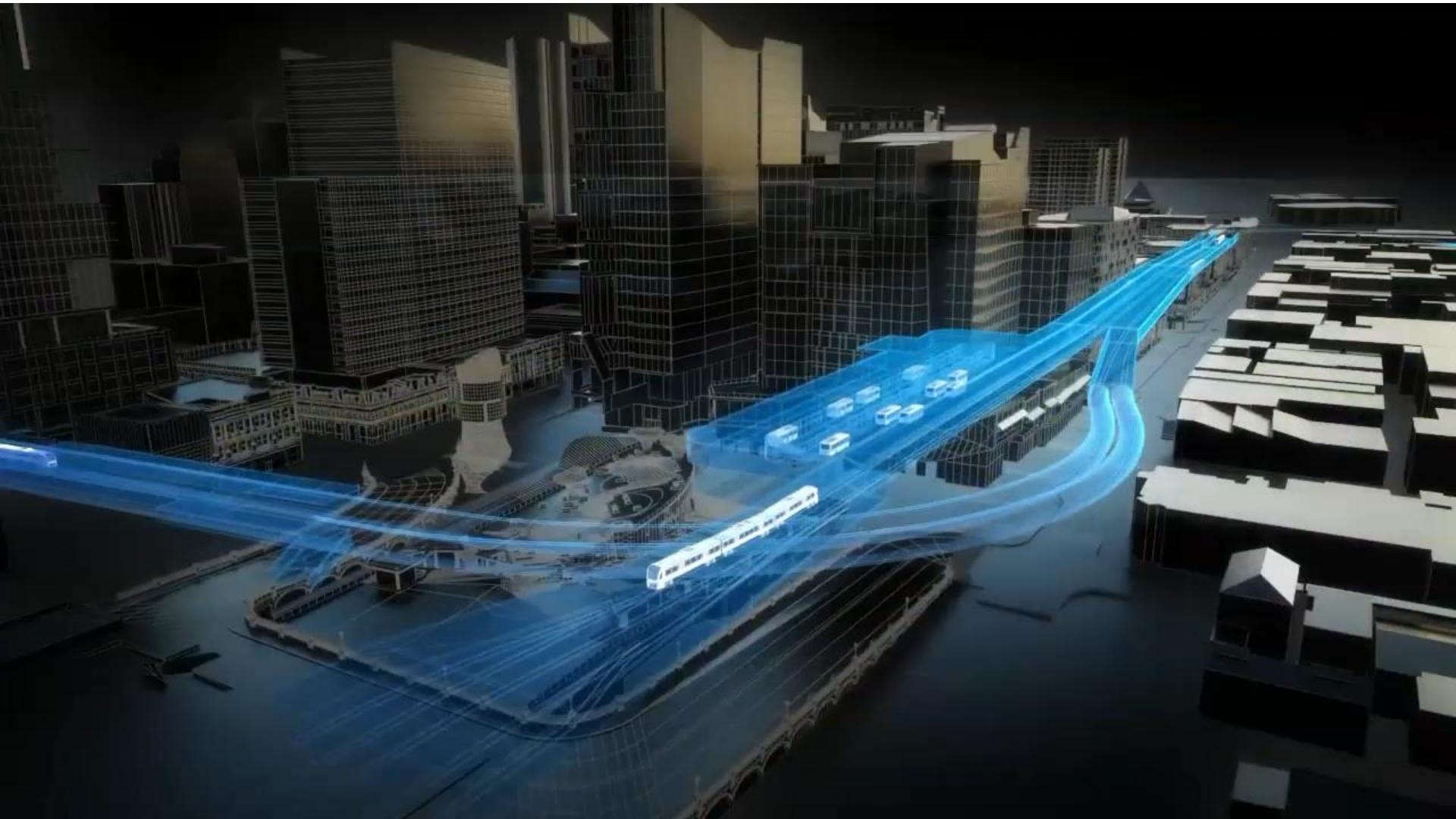
□ 可以将其视为两个函数的线性组合



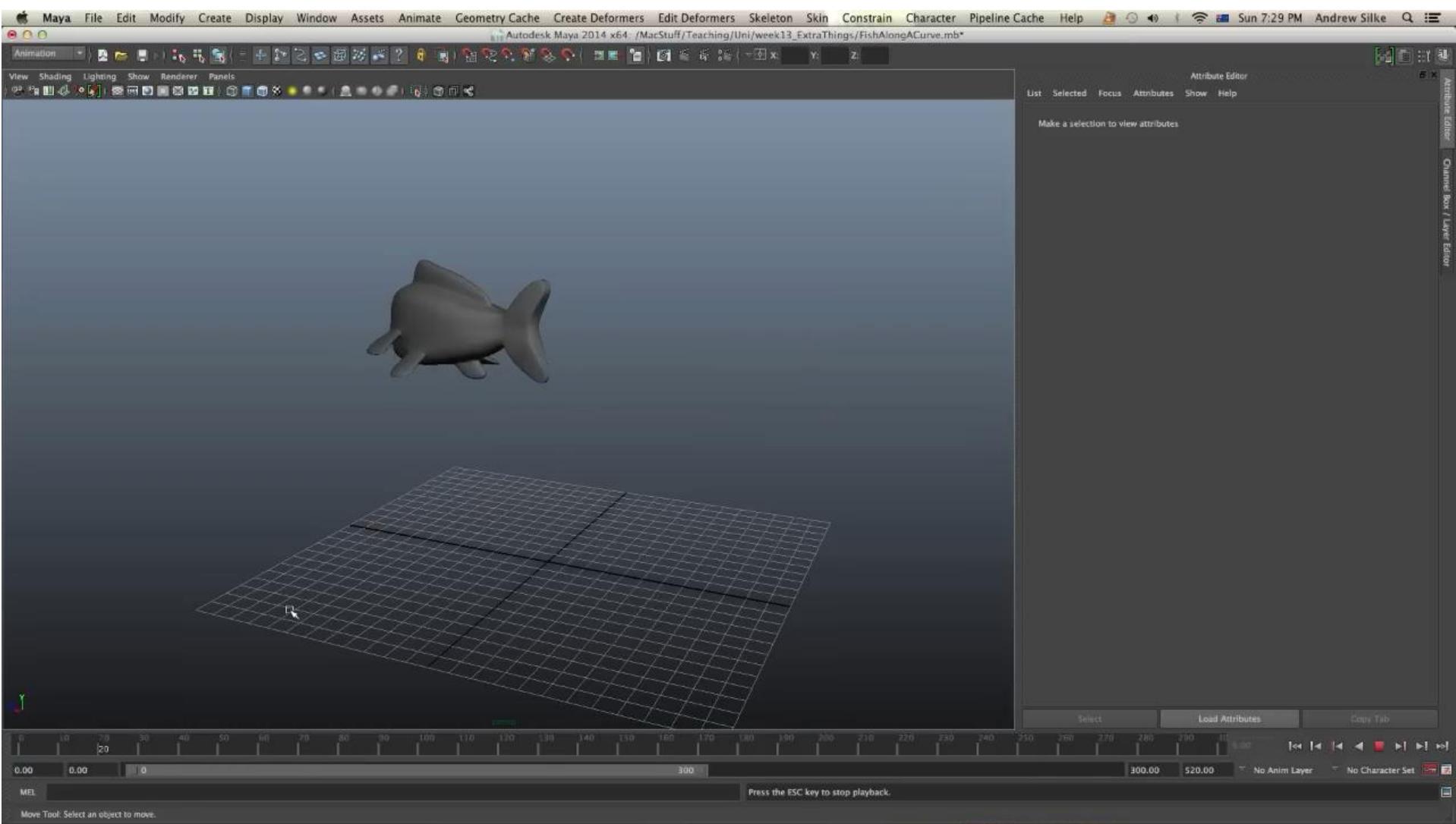
□ 为什么把自己局限于线性基函数？

□ 我们能用其他基函数得到更有趣的曲线 (curve) 吗？

# 相机路径

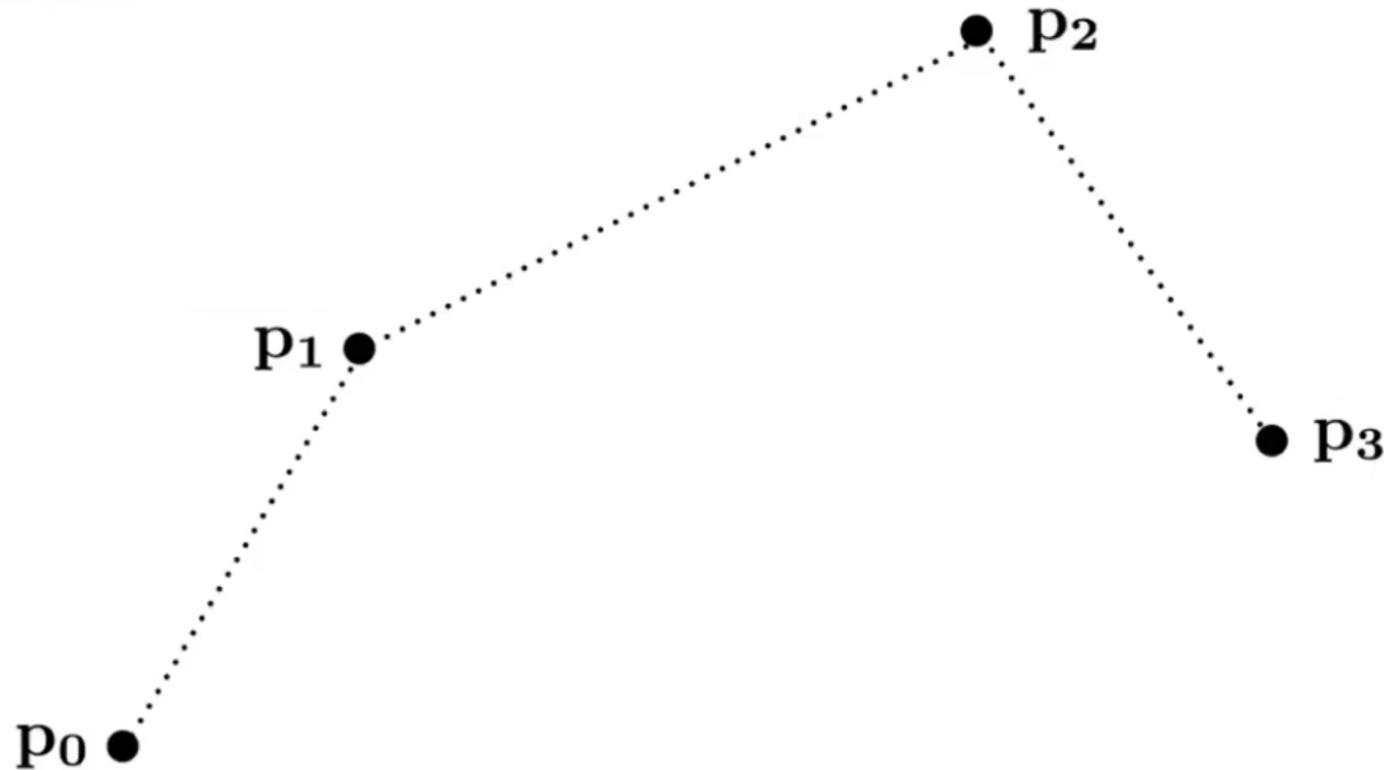


# 动画曲线



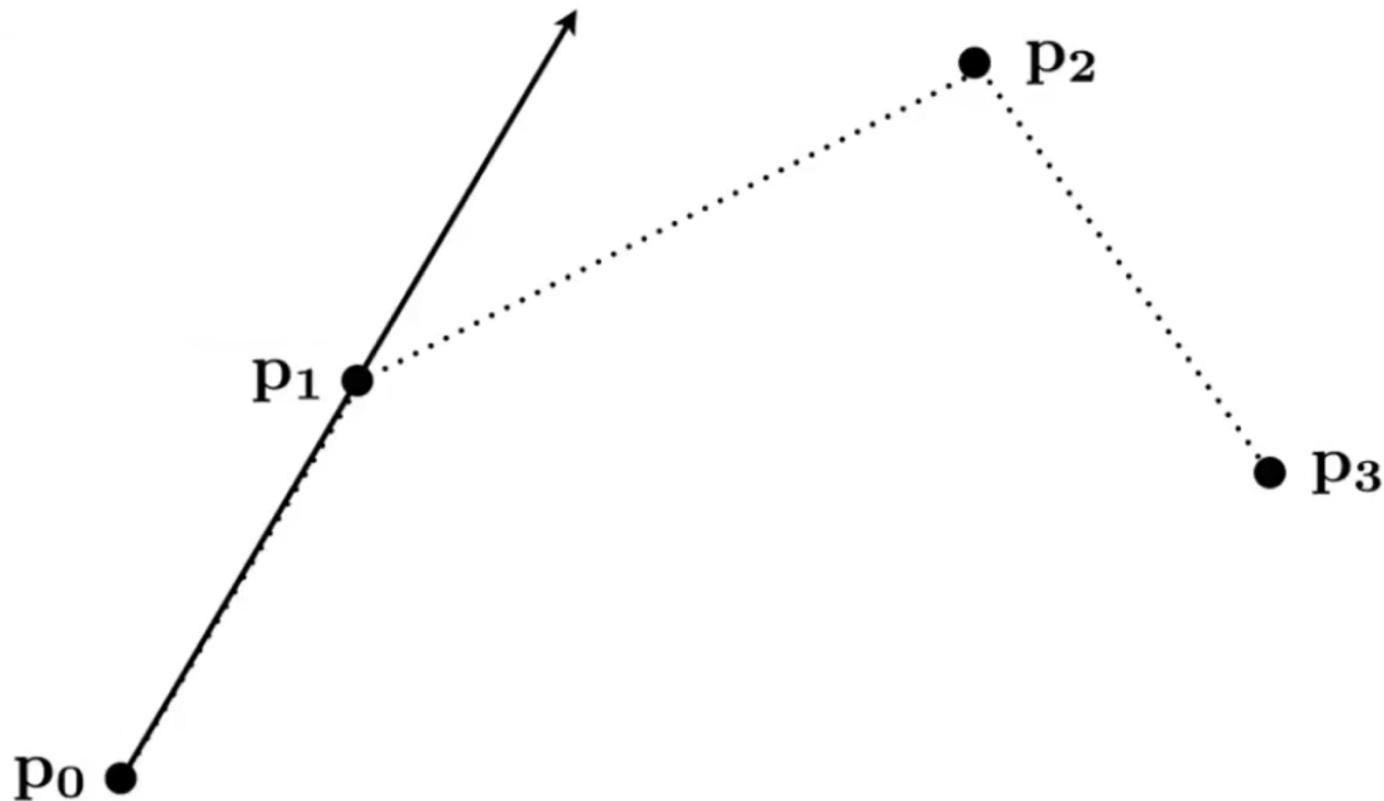
# 贝塞尔曲线

用一系列控制点去控制一条曲线



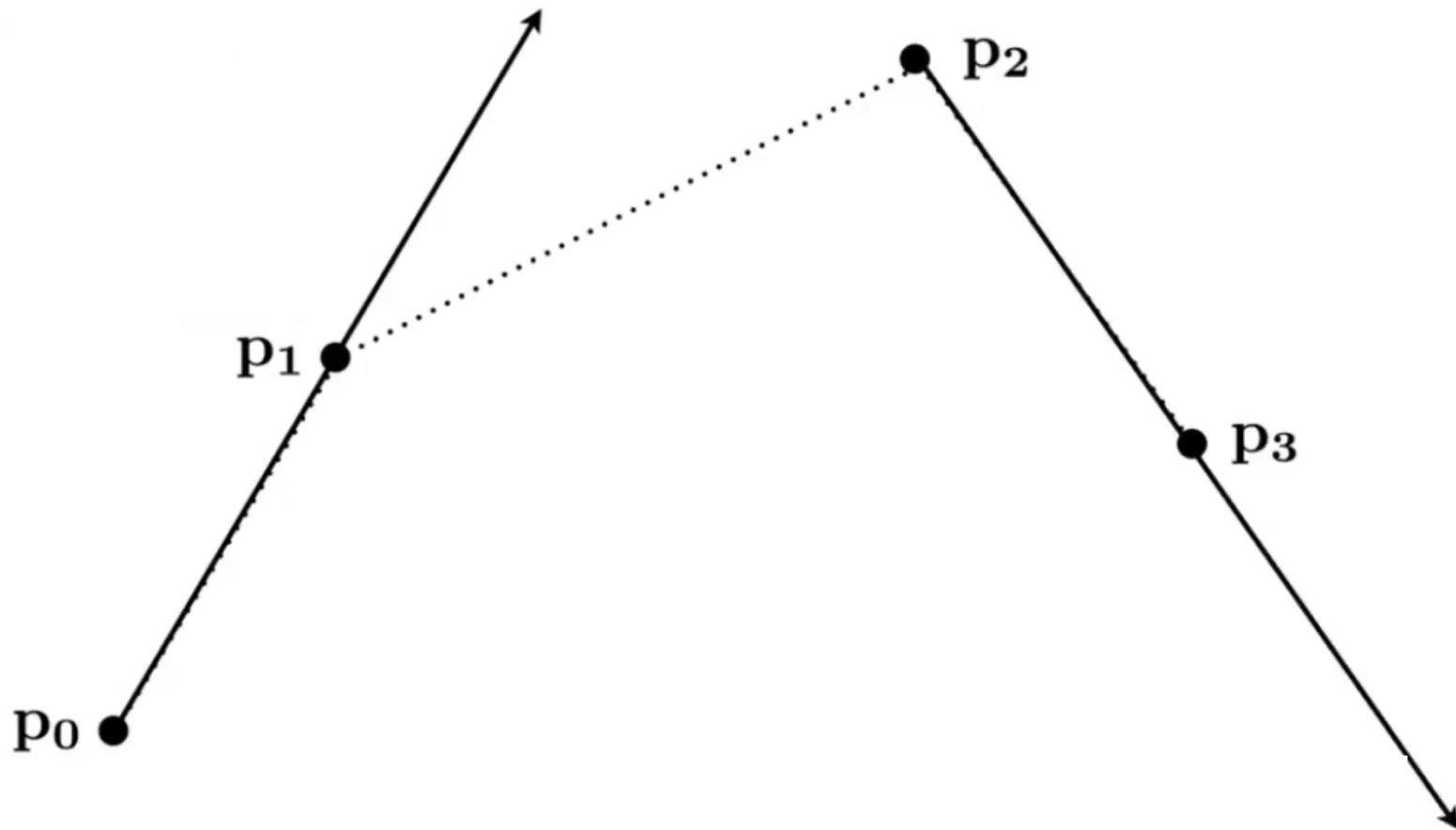
# 贝塞尔曲线

用一系列控制点去控制一条曲线



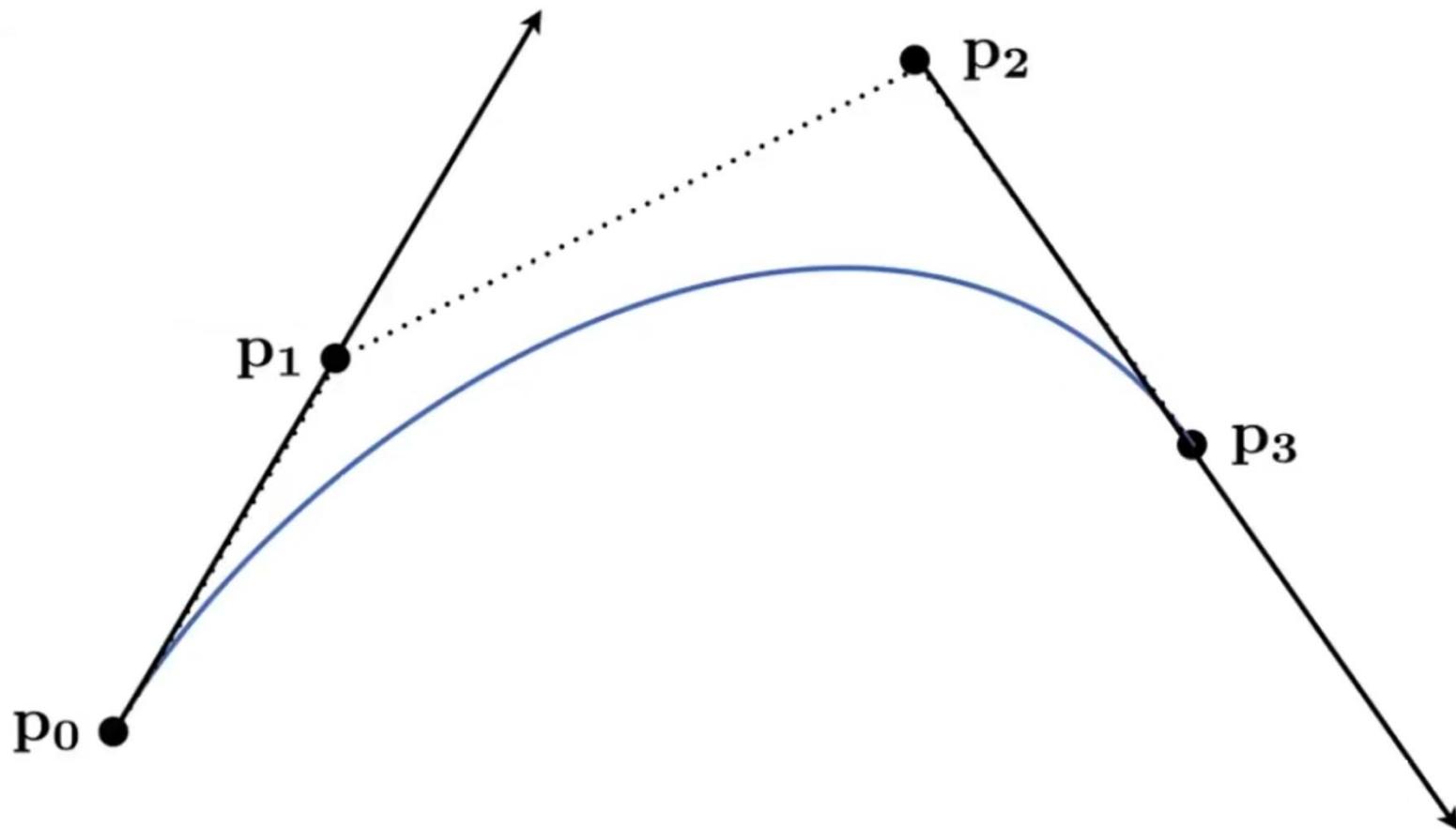
# 贝塞尔曲线

用一系列控制点去控制一条曲线



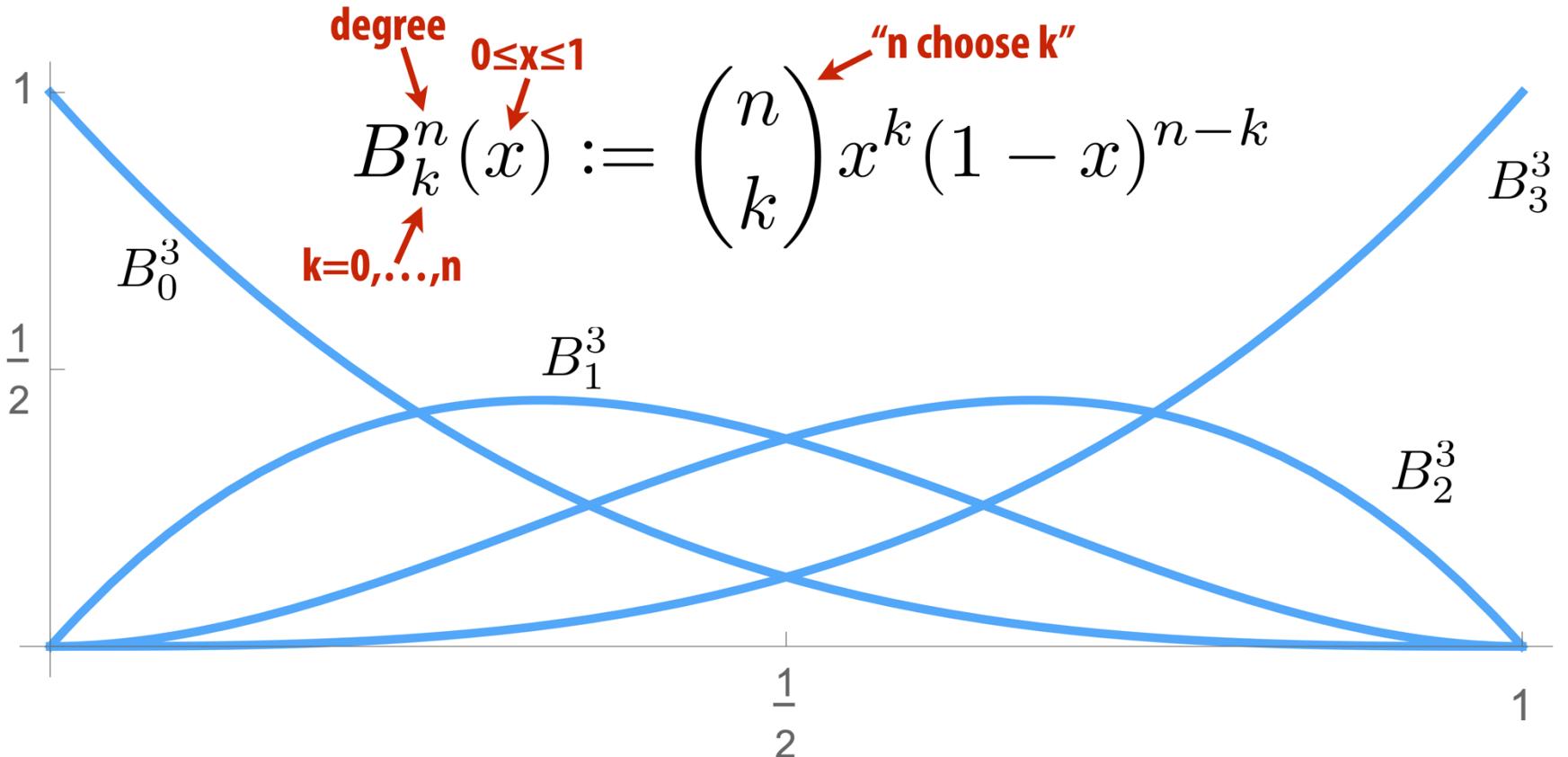
# 贝塞尔曲线

用一系列控制点去控制一条曲线



# 伯恩斯坦基 (Bernstein Basis)

- 线性插值本质上使用一阶多项式 (1<sup>st</sup>-order polynomials)
- 通过使用高阶多项式提供更大的灵活性
- 使用 Bernstein 基, 而不是通常的基 ( $1, x, x^2, x^3, \dots$ )



# 贝塞尔曲线 Bézier Curves (显式表达)

□ 贝塞尔曲线是利用 Bernstein 基表示的曲线

$$\gamma(s) := \sum_{k=0}^n B_{n,k}(s)p_k$$

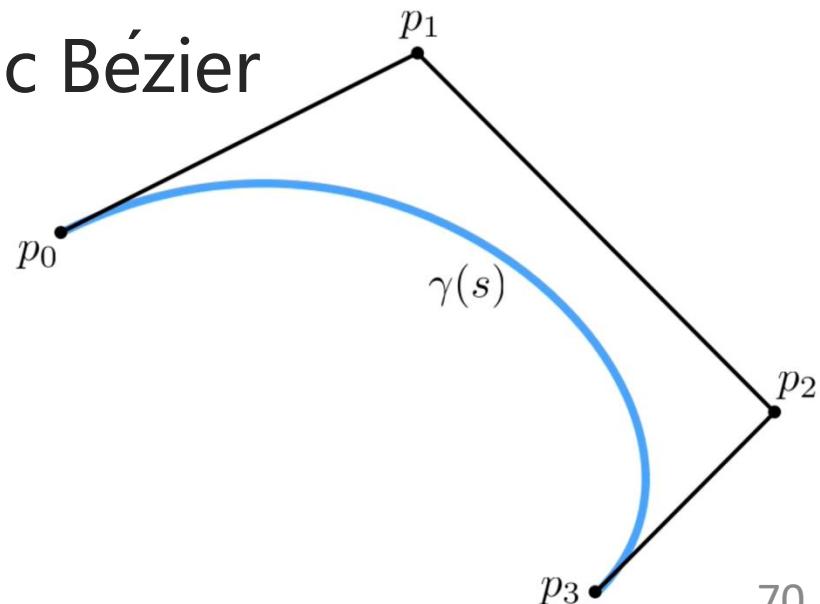
*control points*

□  $n = 1$ , 为一条线段

□  $n = 3$ , 为立方贝塞尔曲线 cubic Bézier

□ 重要特征:

- 1. 穿过端点
- 2. 与端段相切



# 贝塞尔曲线 Bézier Curves (显式表达)

□ 贝塞尔曲线是利用 Bernstein 基表示的曲线

$$\gamma(s) := \sum_{k=0}^n B_{n,k}(s)p_k$$

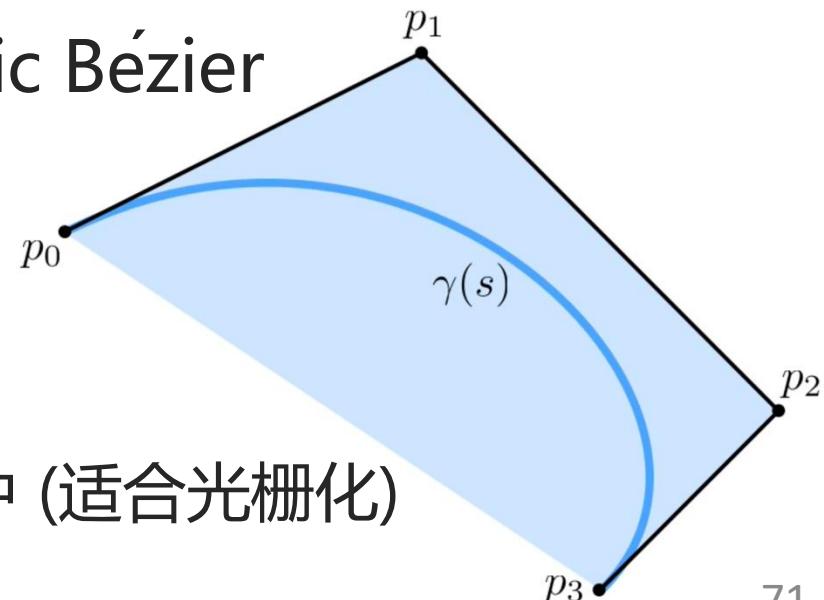
*control points*

□  $n = 1$ , 为一条线段

□  $n = 3$ , 为立方贝塞尔曲线 cubic Bézier

□ 重要特征:

- 1. 穿过端点
- 2. 与端段相切
- 3. 包含在凸包 (convex hull) 中 (适合光栅化)

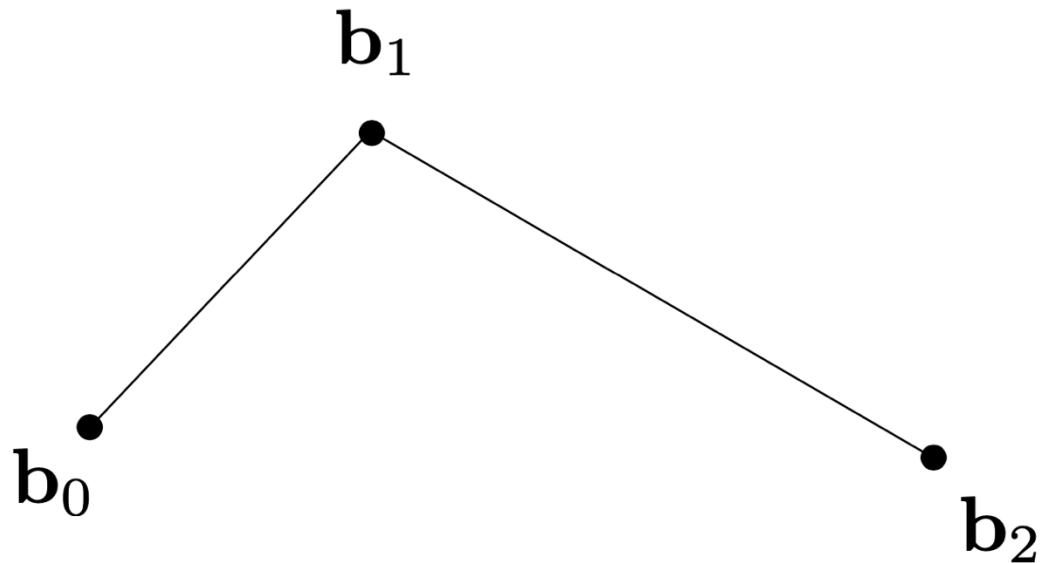


# 为什么要用点来控制曲线？

- 更灵活、精确地创建和修改曲线，不需要记录每一个点
- 更容易实现光滑、符合物理特性点曲线
- 更容易实现平滑的动画效果
- 更容易存储数据
- ...

# Bézier Curves – de Casteljau Algorithm

口考虑三个点的贝塞尔曲线 (quadratic Bezier)



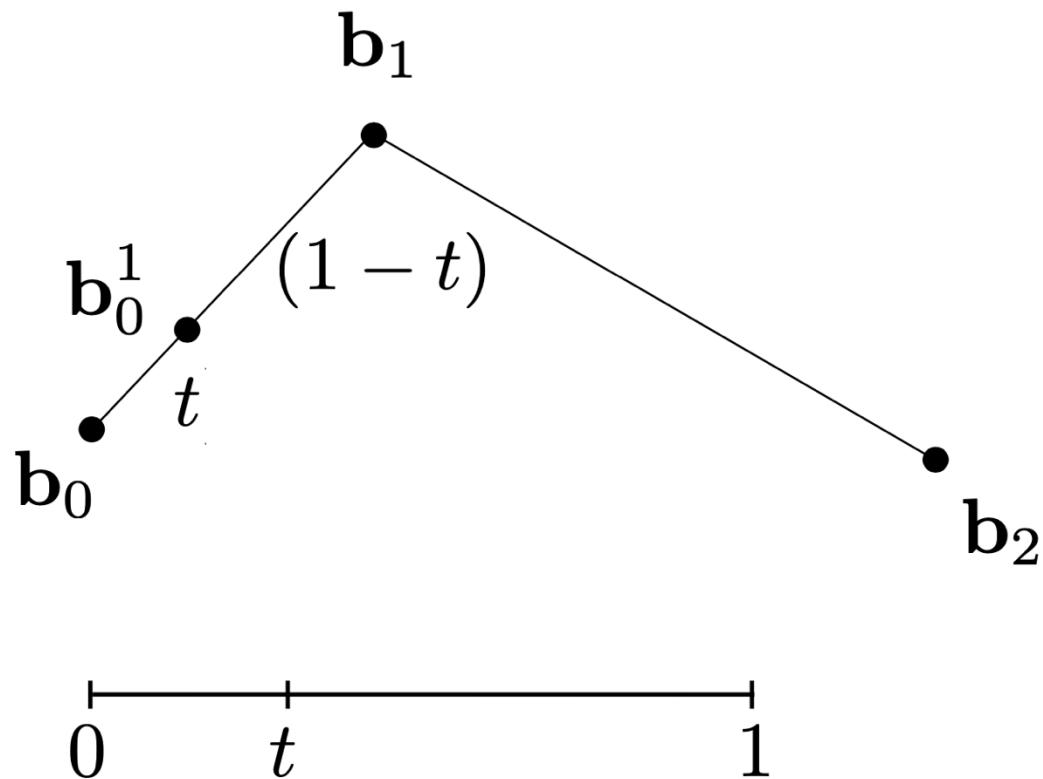
Pierre Bézier  
1910 – 1999



Paul de Casteljau  
b. 1930

# Bézier Curves – de Casteljau Algorithm

口在直线  $b_0, b_1$  中利用线性插值插入一个点  $b_0^1$



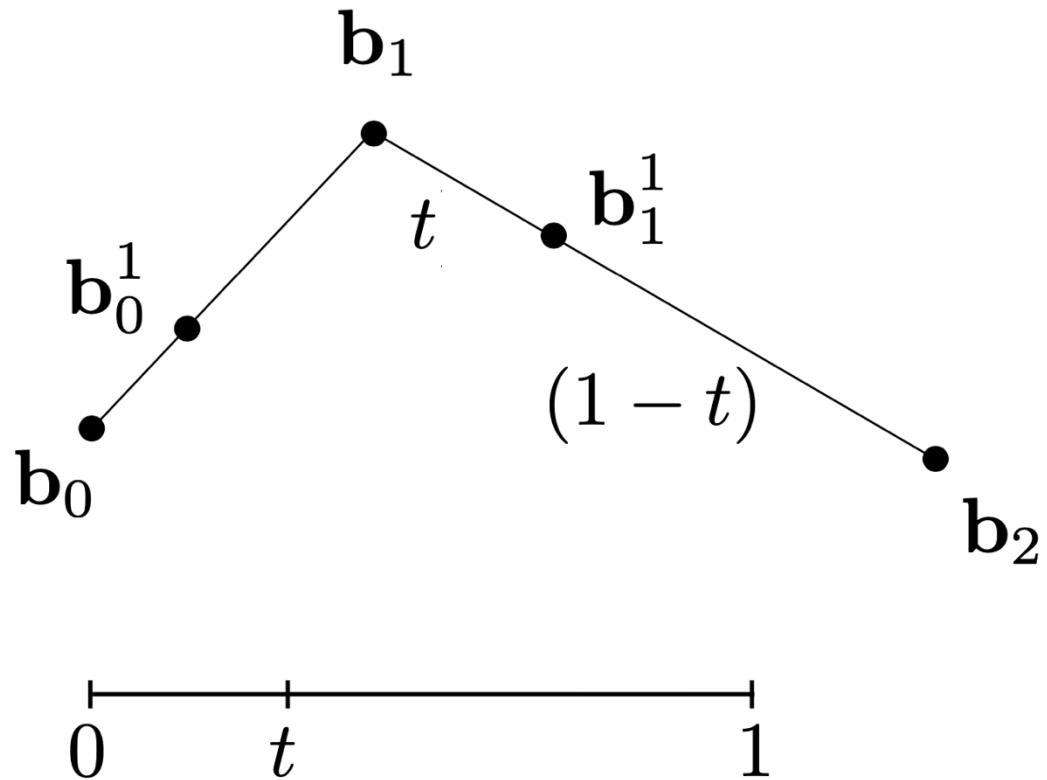
Pierre Bézier  
1910 – 1999



Paul de Casteljau  
b. 1930

# Bézier Curves – de Casteljau Algorithm

□ 在直线  $b_1, b_2$  中同样插入一个点  $b_1^1$



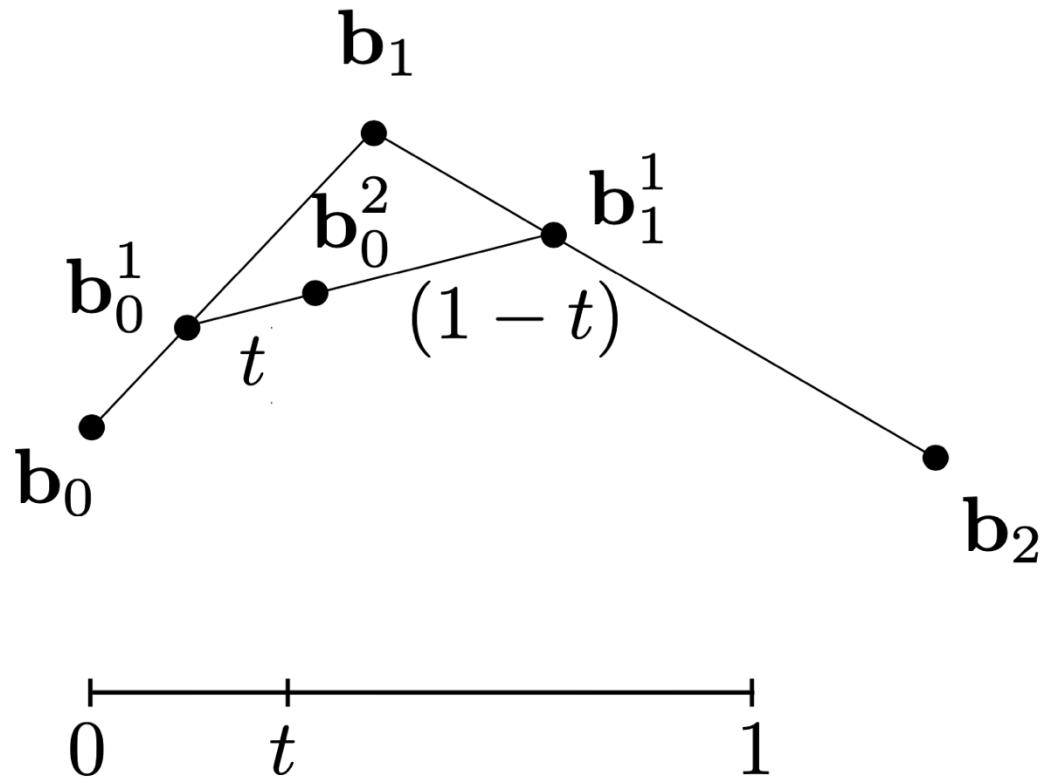
Pierre Bézier  
1910 – 1999



Paul de Casteljau  
b. 1930

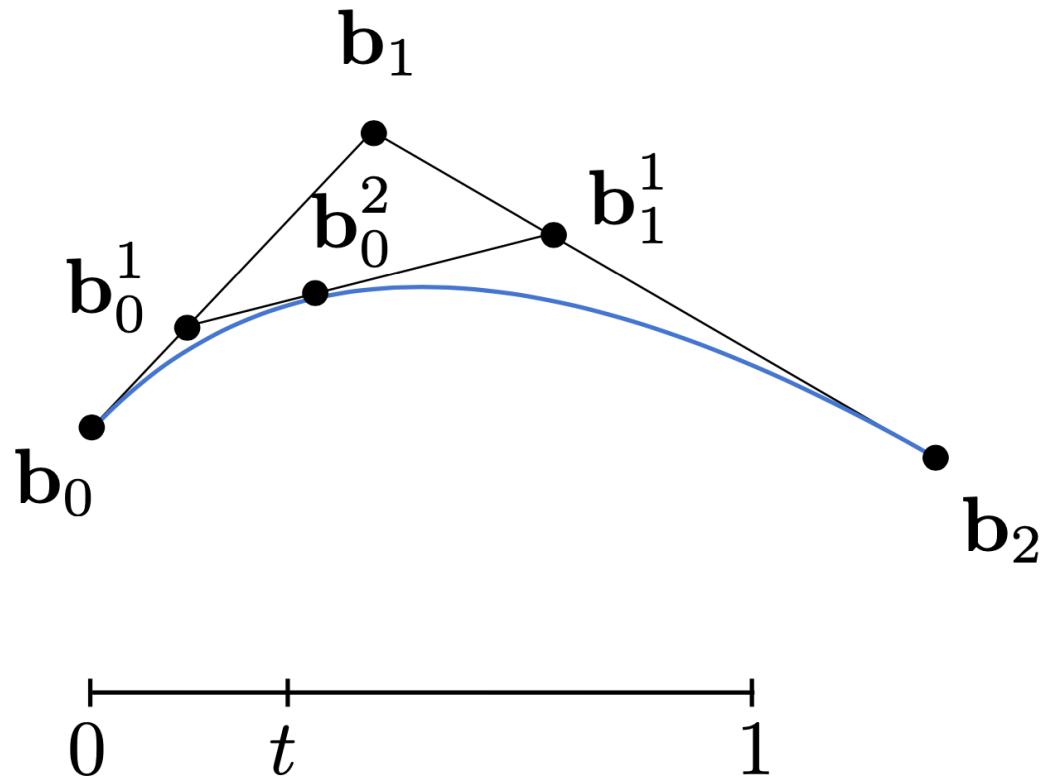
# Bézier Curves – de Casteljau Algorithm

□ 在新的直线  $b_0^1, b_2$  中插入一个点  $b_1^1$



# Bézier Curves – de Casteljau Algorithm

对  $[0,1]$  中的每个  $t$  运行相同的算法



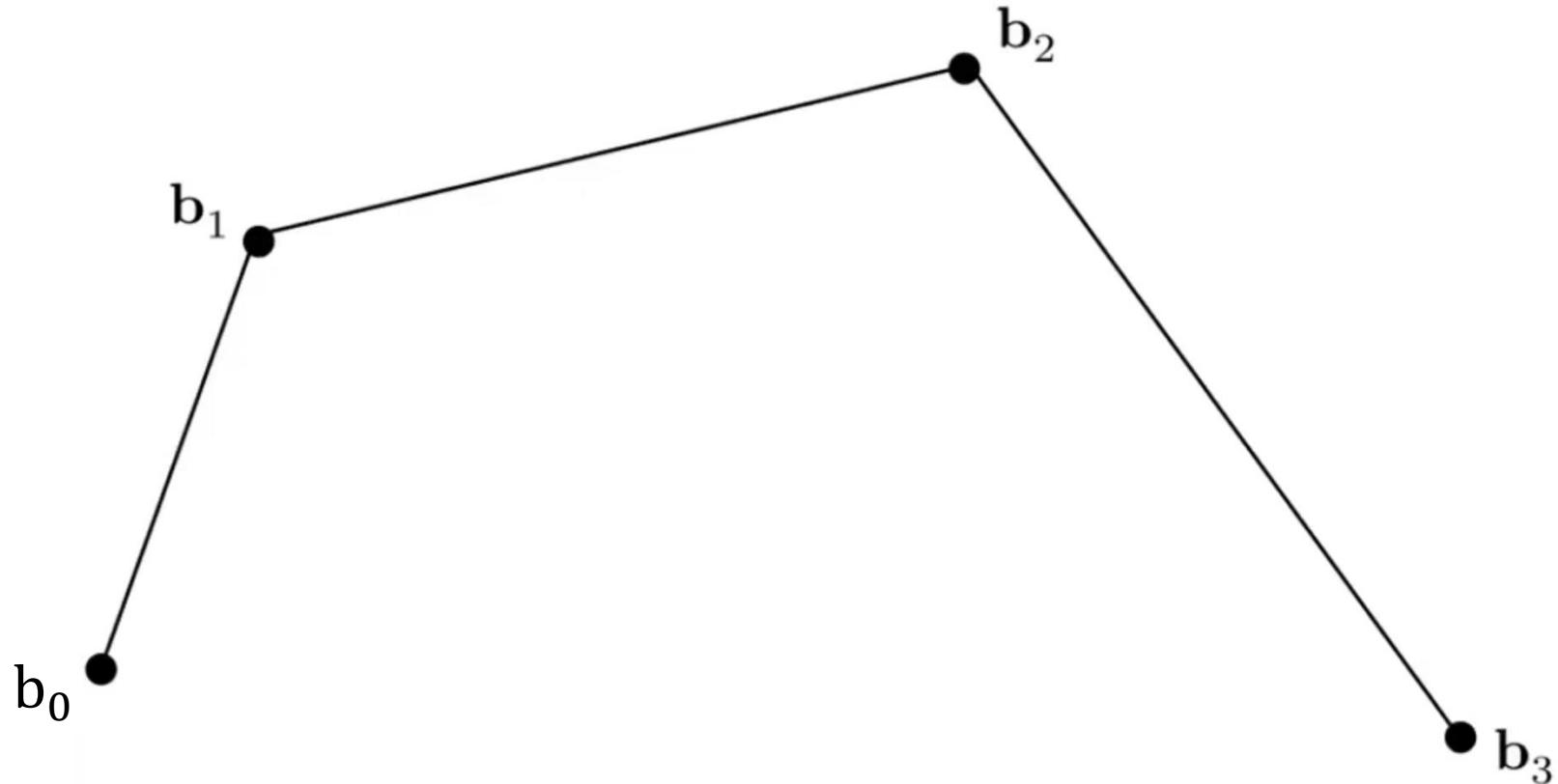
Pierre Bézier  
1910 – 1999



Paul de Casteljau  
b. 1930

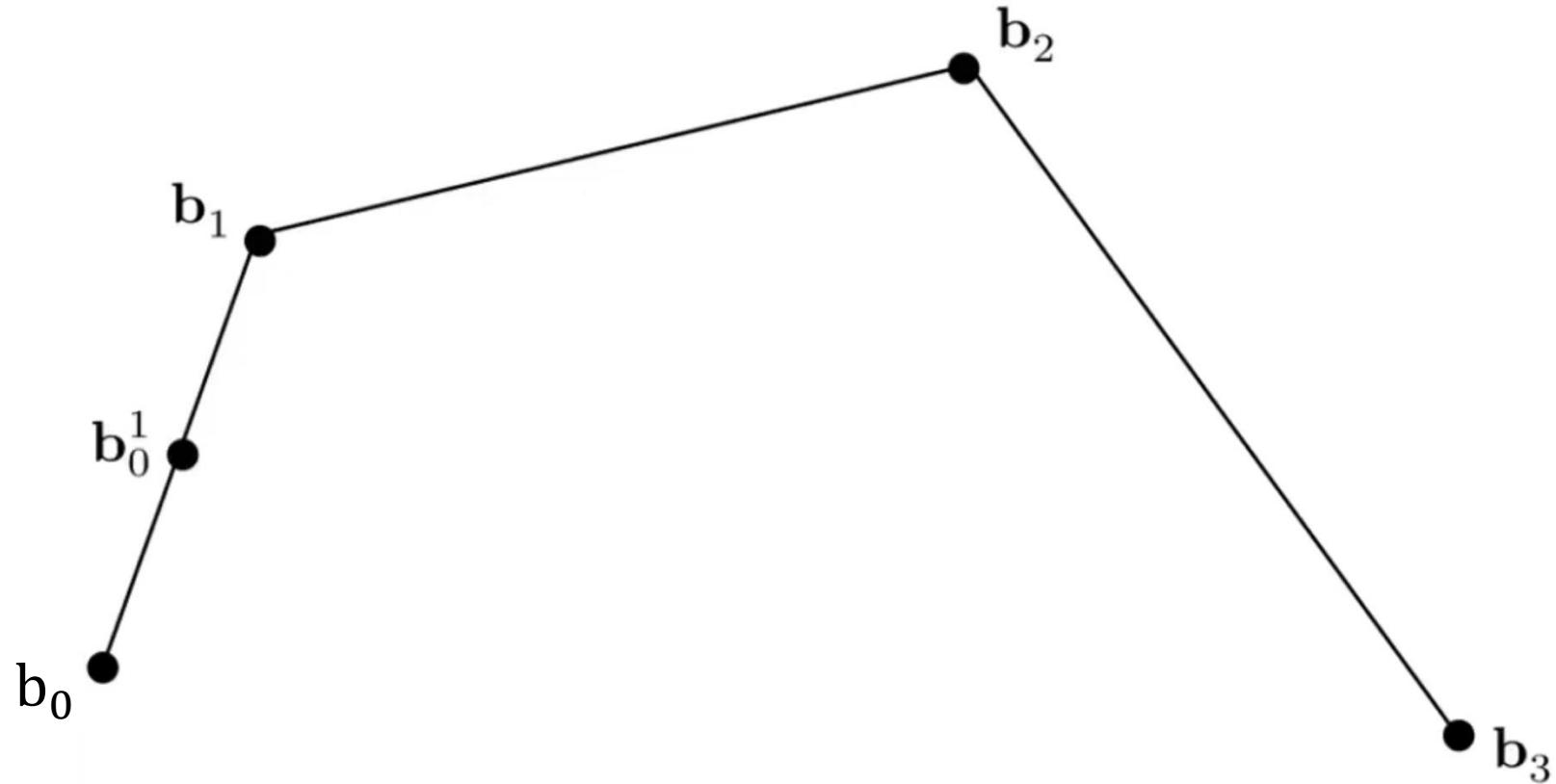
# Cubic Bézier Curve – de Casteljau

- 共四个输入点
- 相同的递归线性插值



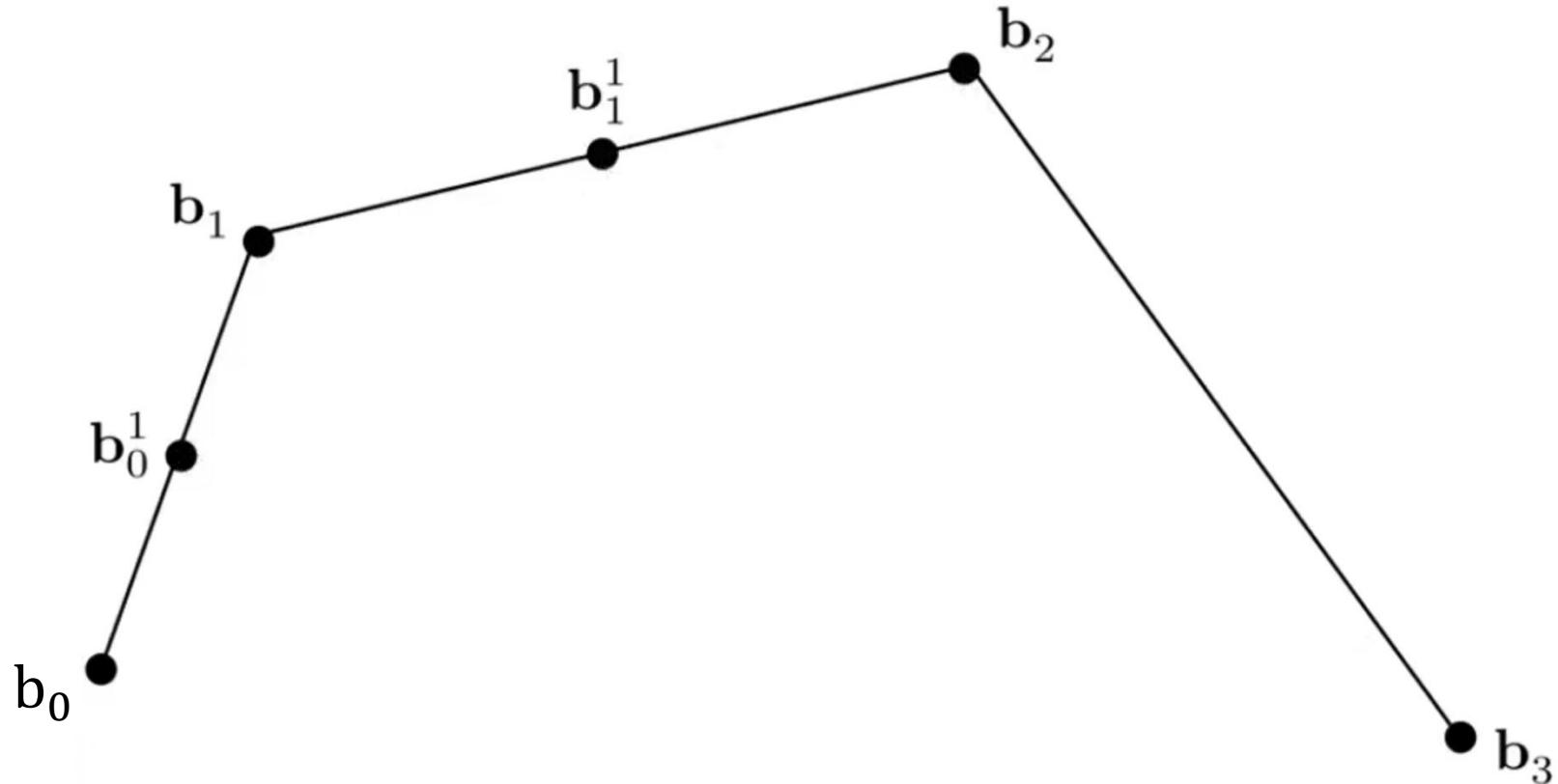
# Cubic Bézier Curve – de Casteljau

- 共四个输入点
- 相同的递归线性插值



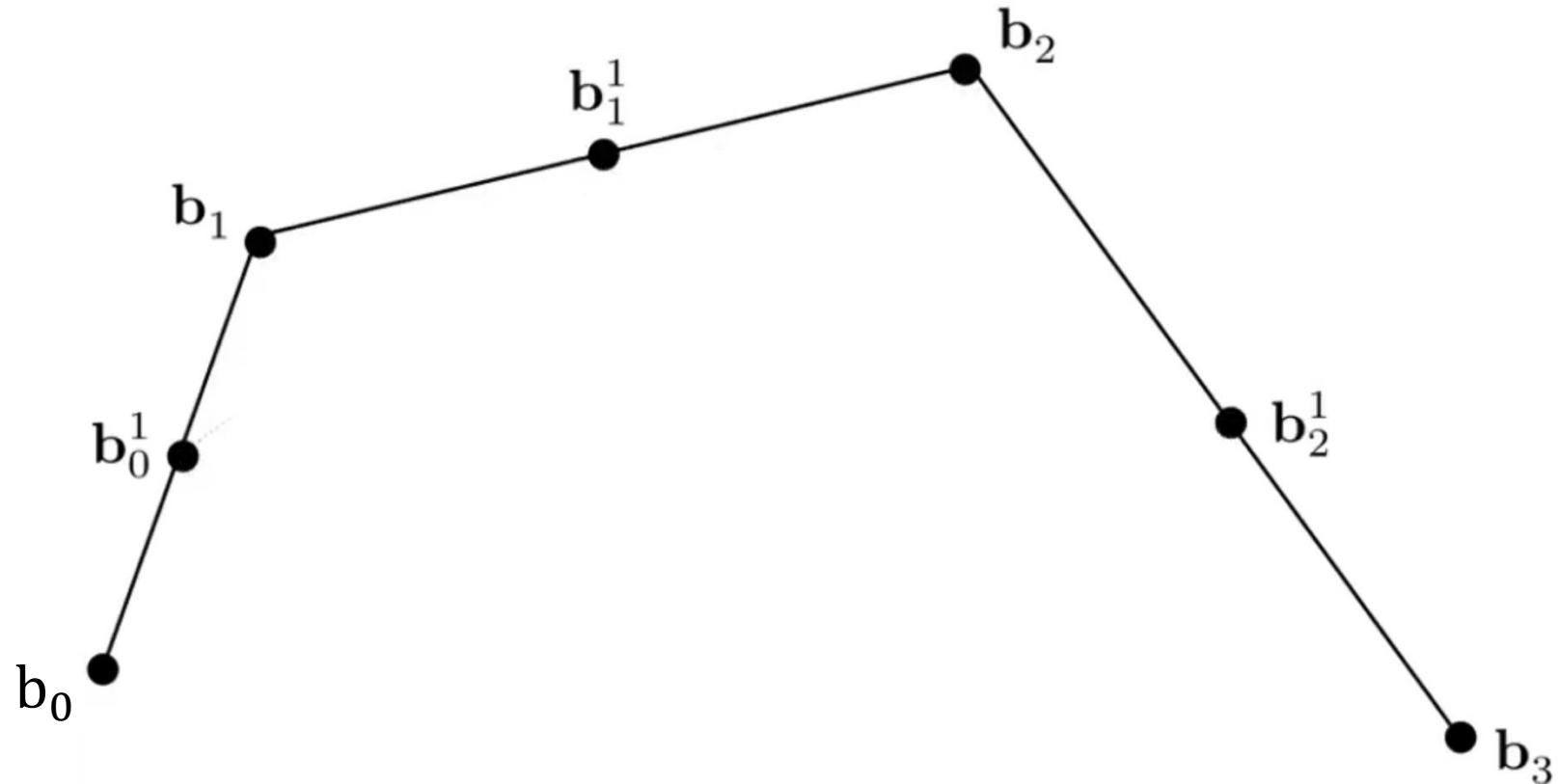
# Cubic Bézier Curve – de Casteljau

- 共四个输入点
- 相同的递归线性插值



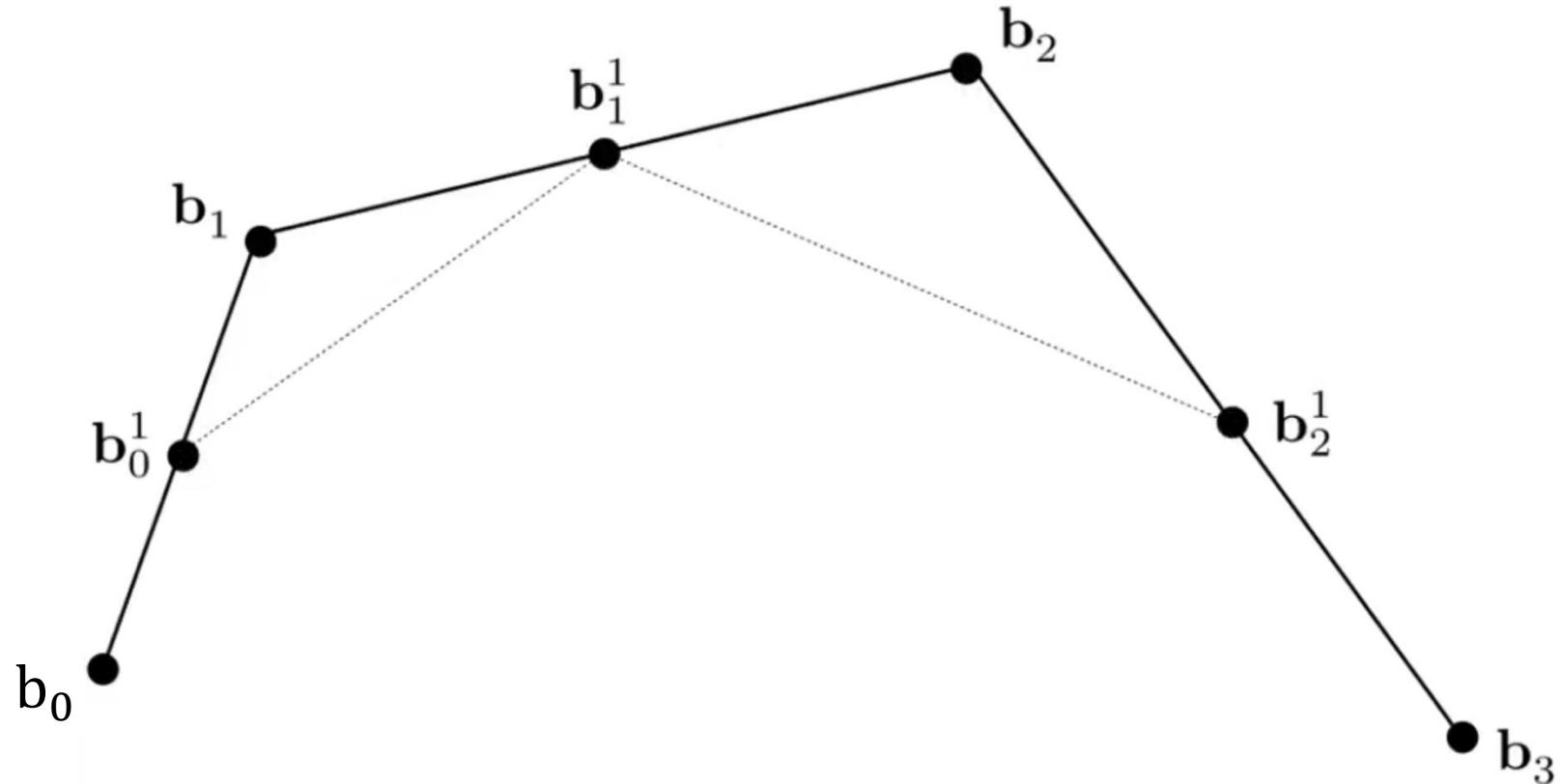
# Cubic Bézier Curve – de Casteljau

- 共四个输入点
- 相同的递归线性插值



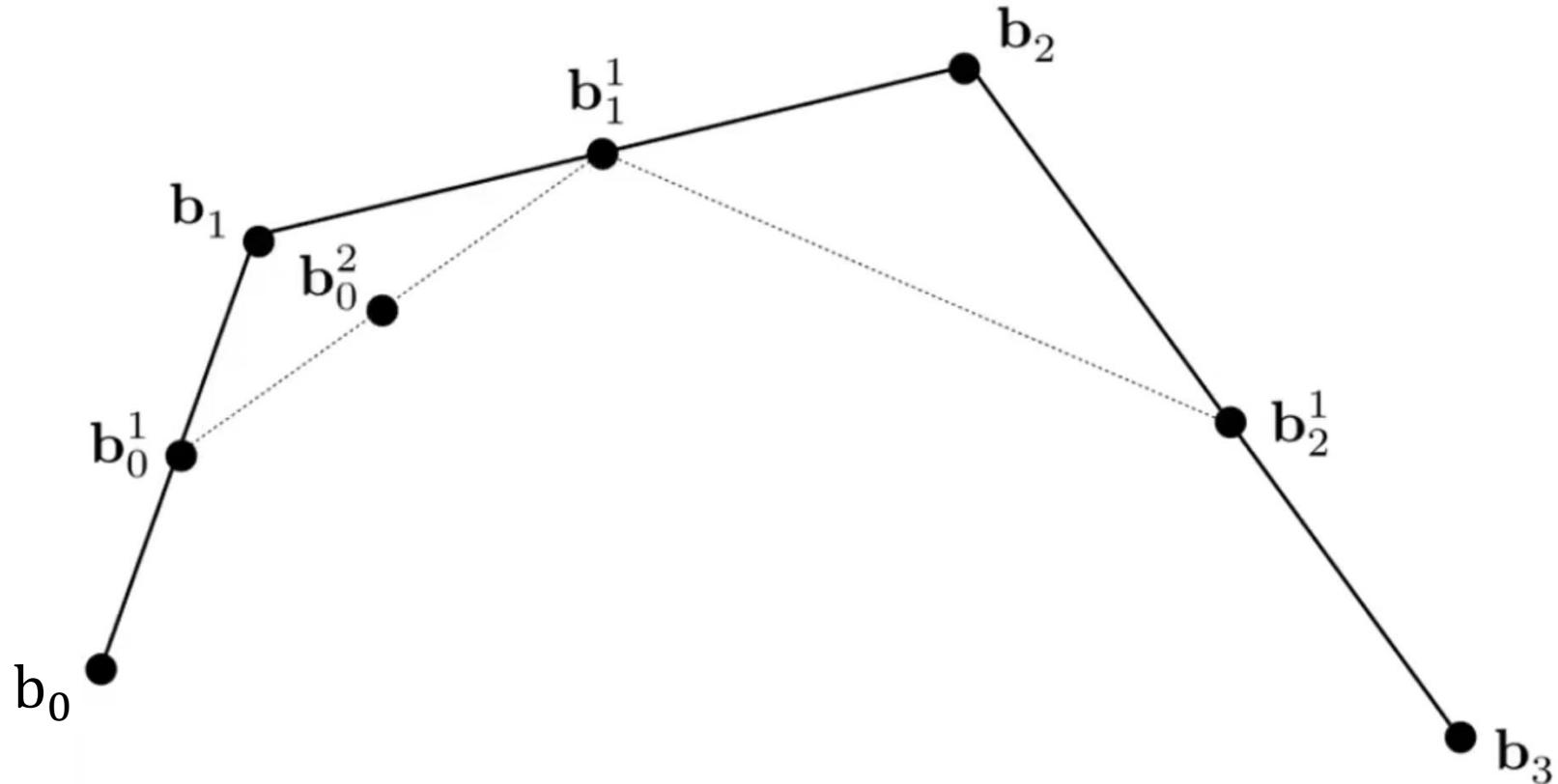
# Cubic Bézier Curve – de Casteljau

- 共四个输入点
- 相同的递归线性插值



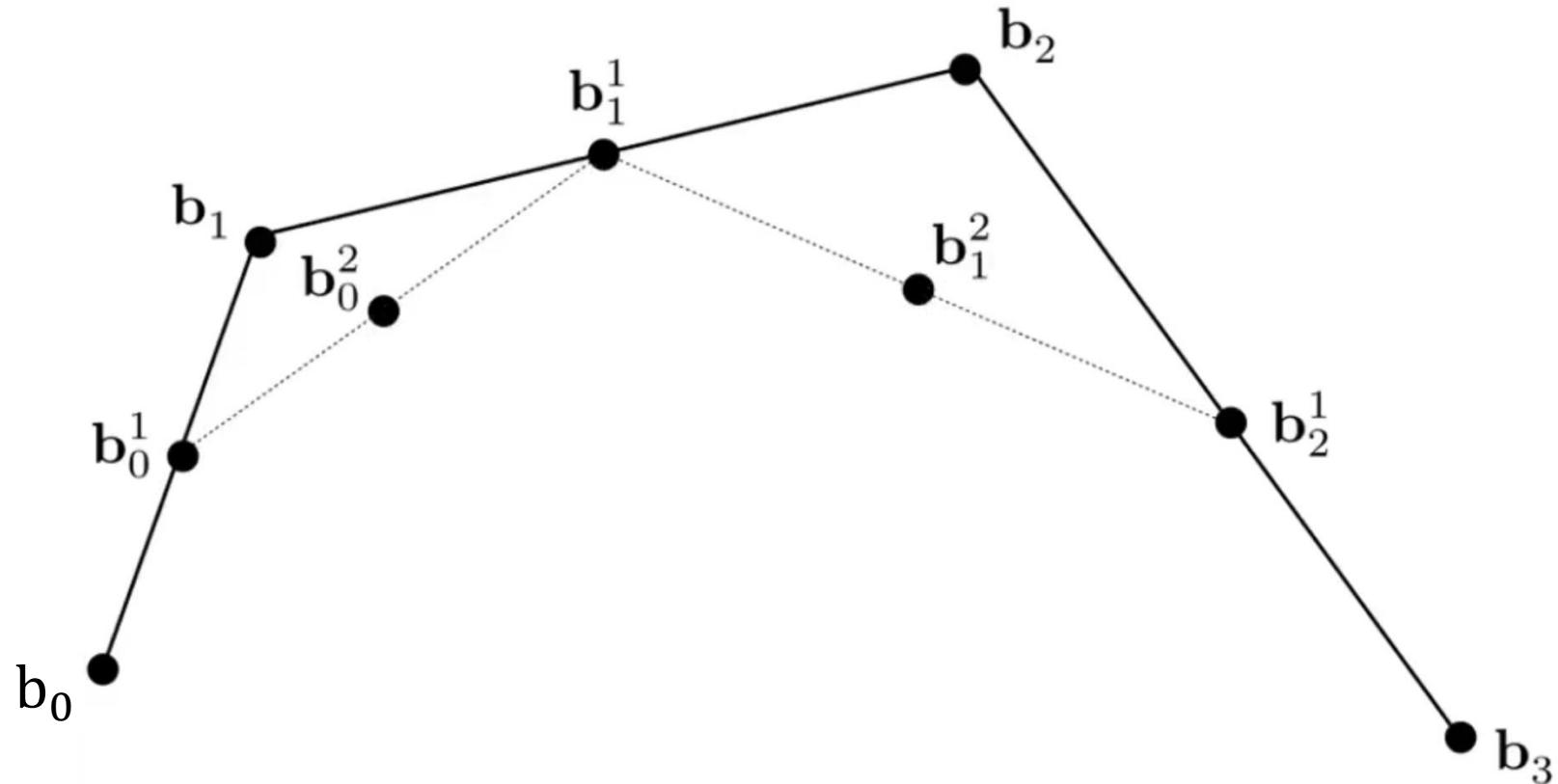
# Cubic Bézier Curve – de Casteljau

- 共四个输入点
- 相同的递归线性插值



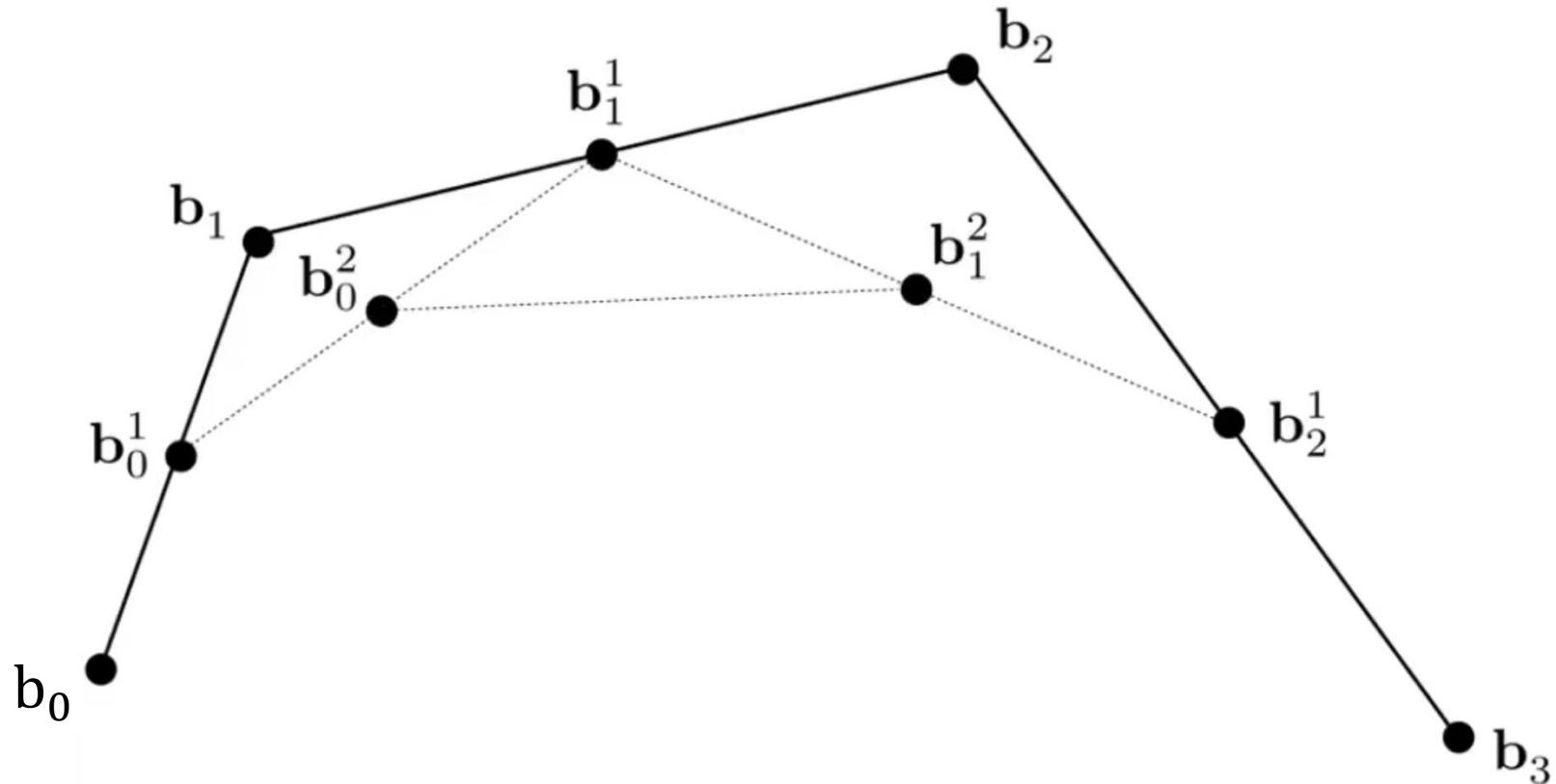
# Cubic Bézier Curve – de Casteljau

- 共四个输入点
- 相同的递归线性插值



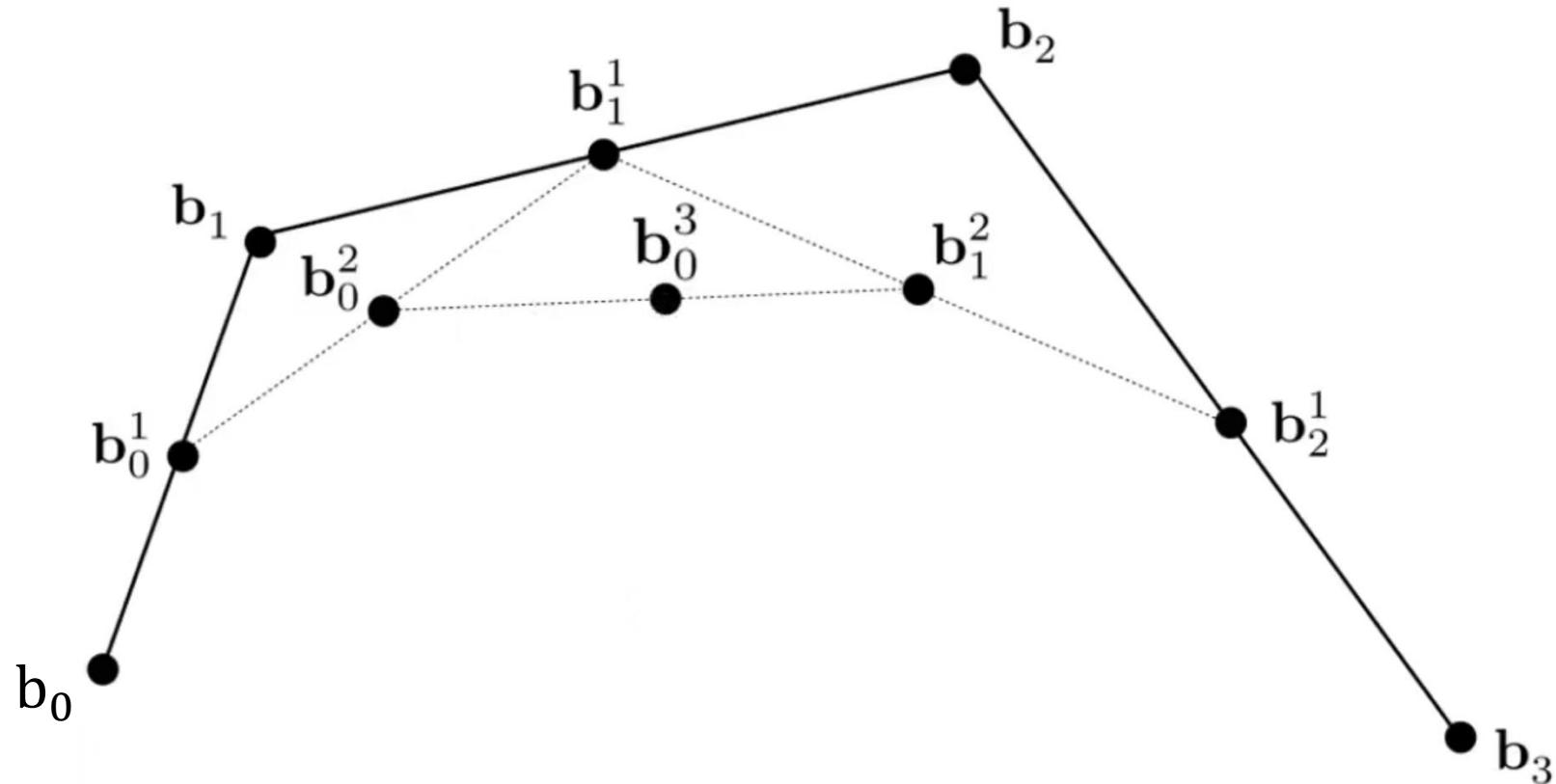
# Cubic Bézier Curve – de Casteljau

- 共四个输入点
- 相同的递归线性插值



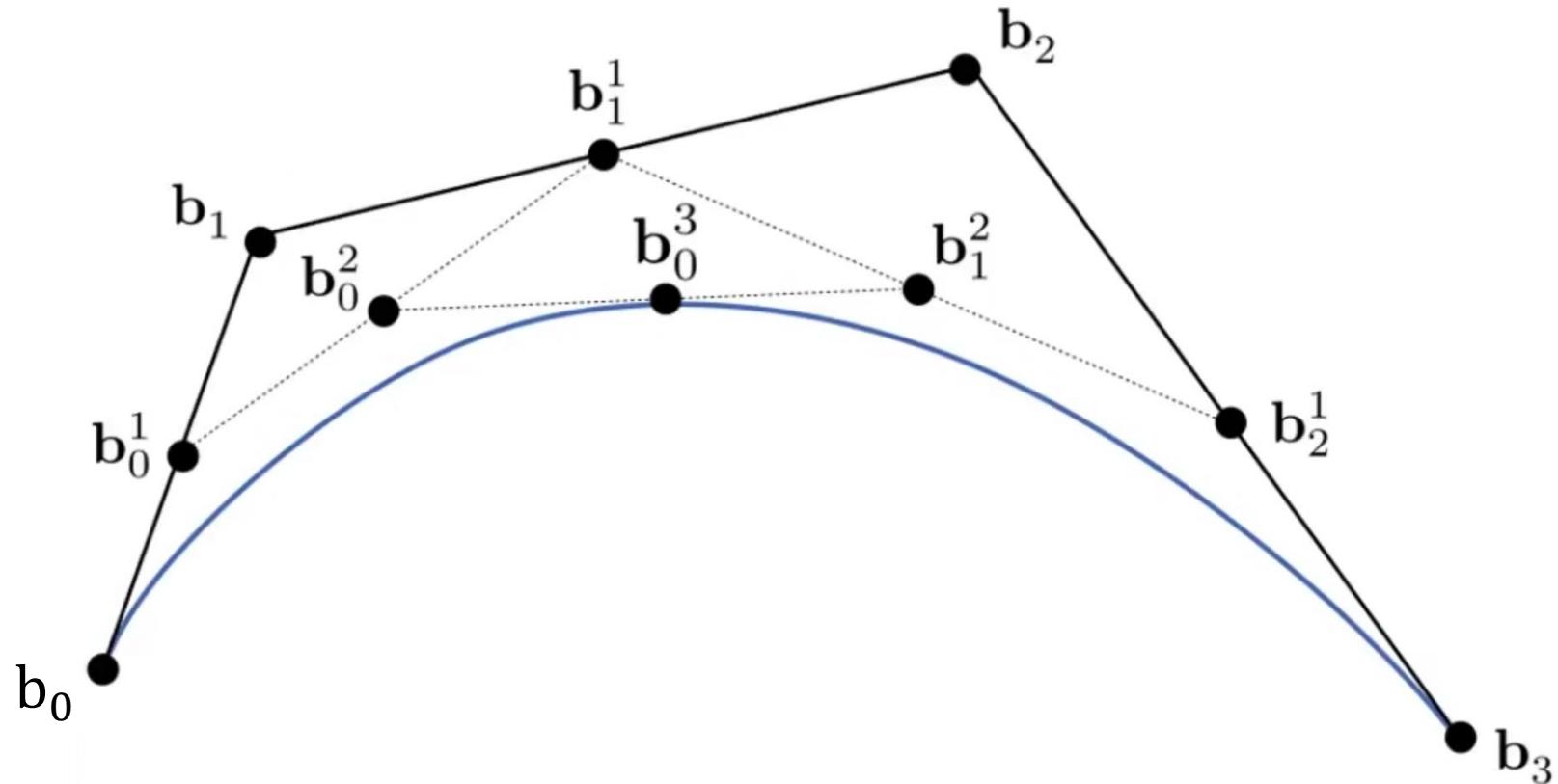
# Cubic Bézier Curve – de Casteljau

- 共四个输入点
- 相同的递归线性插值

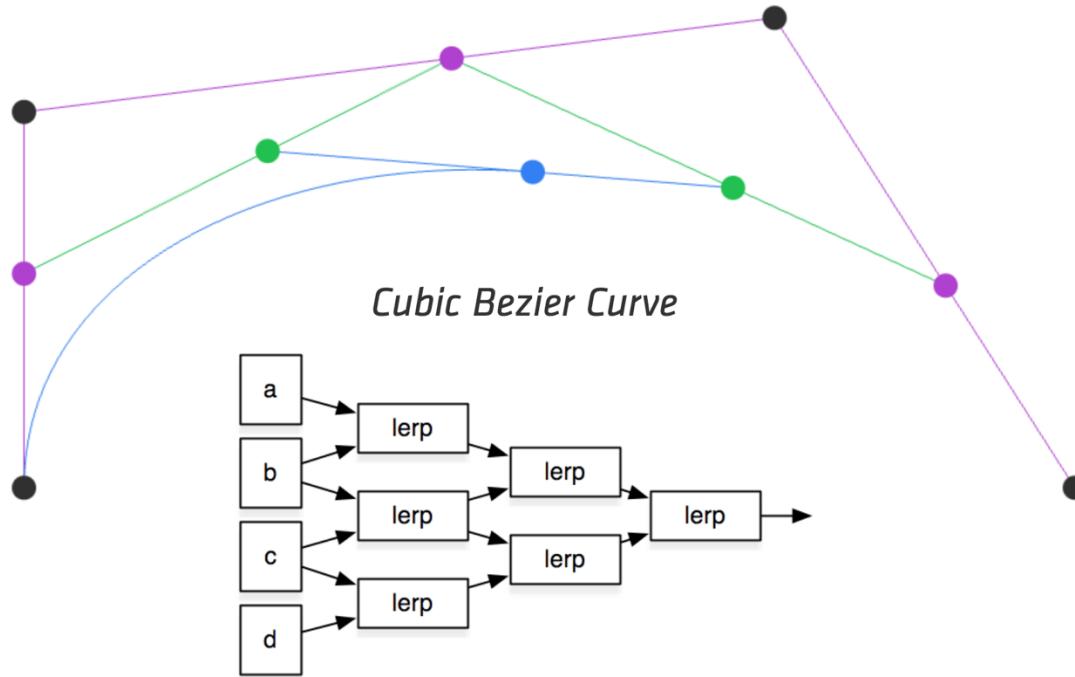


# Cubic Bézier Curve – de Casteljau

- 共四个输入点
- 相同的递归线性插值



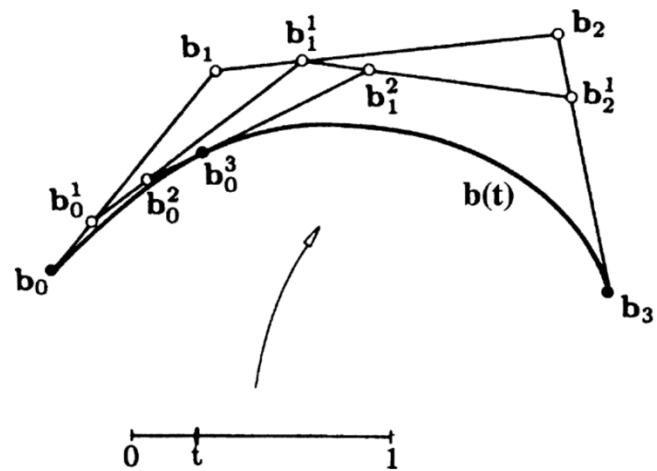
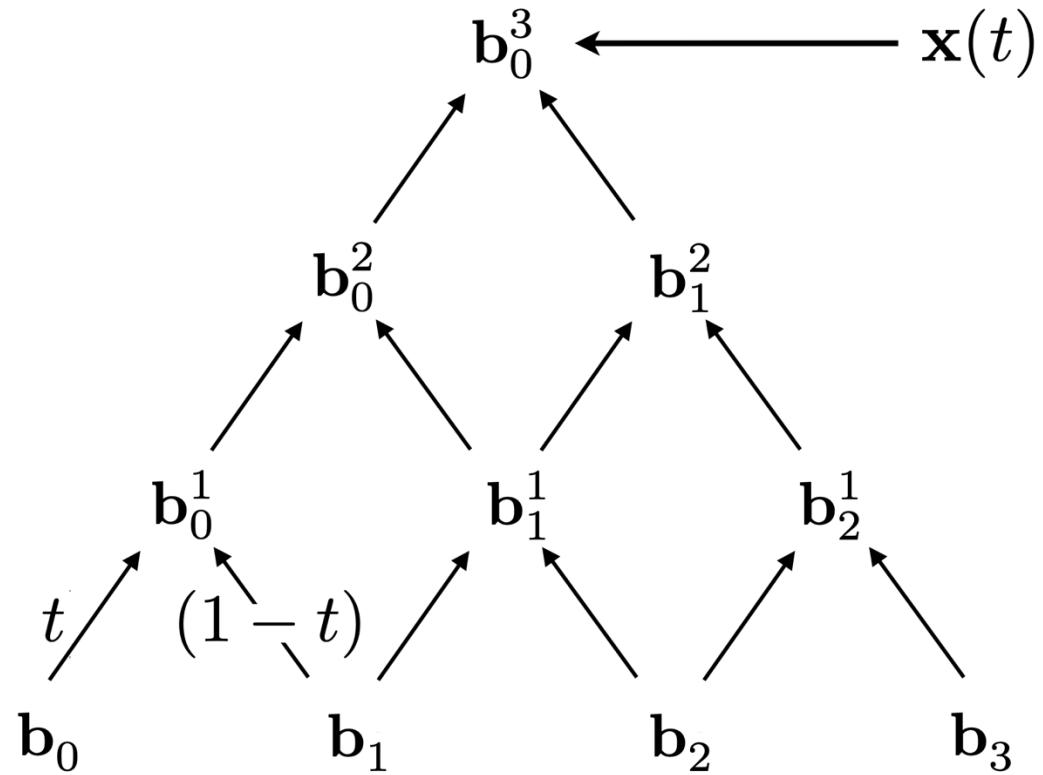
# 可视化 de Casteljau 算法



Animation: Steven Wittens, Making Things with Maths  
<https://acko.net/files/fullfrontal/fullfrontal/wdcode/online.html>

# 贝塞尔曲线 – 代数公式

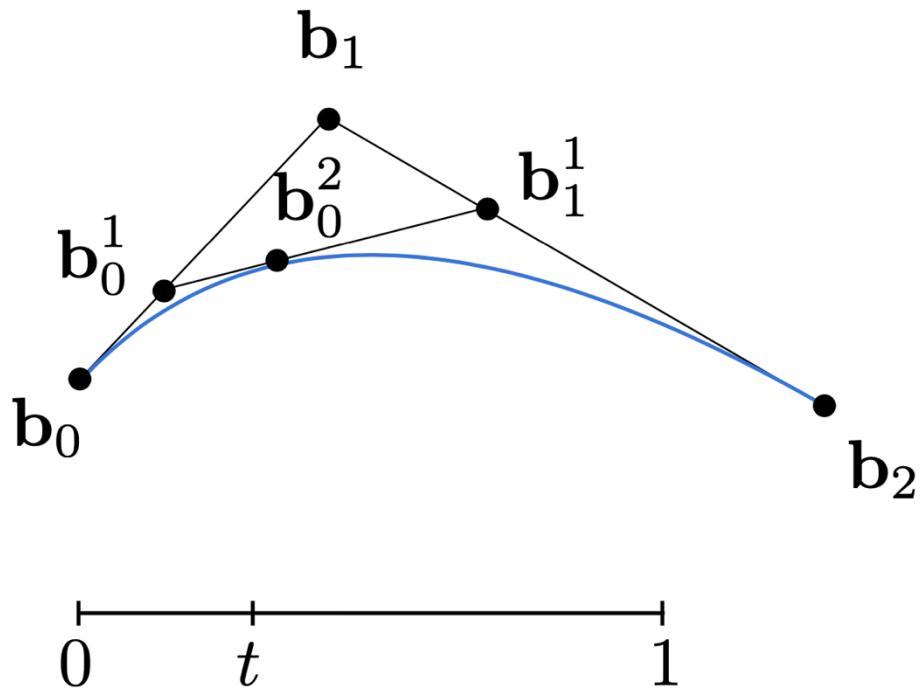
□ de Casteljau 算法给出了一个系数金字塔



每个向右箭头乘以  $t$ , 每个向左箭头乘以  $(1 - t)$

# 贝塞尔曲线 – 代数形式

口示例：三个点构成的二次贝塞尔曲线



$$b_0^1(t) = tb_0 + (1 - t)b_1$$

$$b_1^1(t) = tb_1 + (1 - t)b_2$$

$$b_0^2(t) = tb_0^1 + (1 - t)b_1^1$$

$$b_0^2(t) = \textcircled{t^2}b_0 + \textcircled{2t(1-t)}b_1 + \textcircled{(1-t)^2}b_2$$

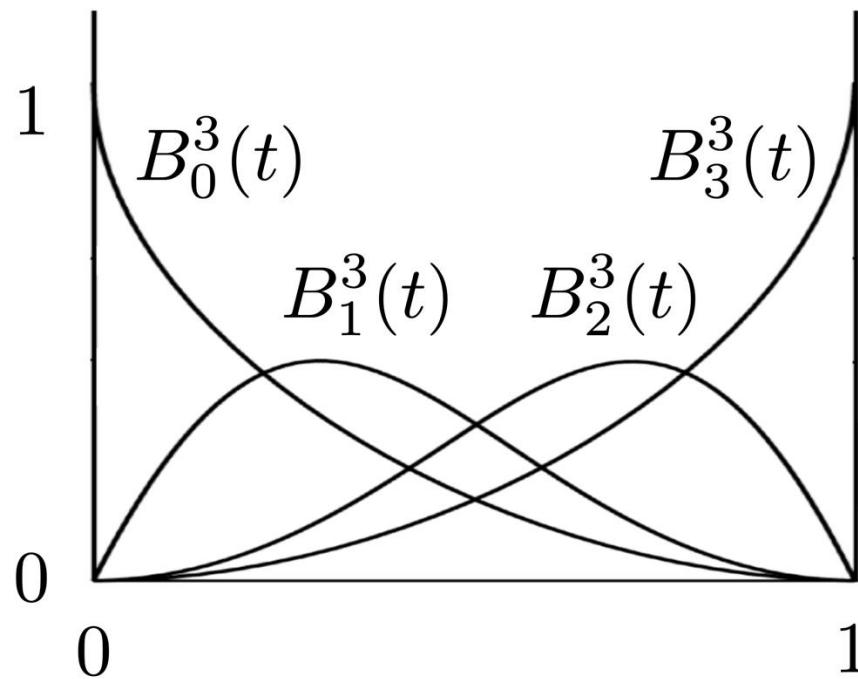
# 贝塞尔曲线基函数

□伯恩斯坦多项式

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



Sergei N. Bernstein  
1880 – 1968



# 贝塞尔曲线 – 一般代数形式

口伯恩斯坦 (Bernstein) 形式的  $n$  阶贝塞尔曲线

$n$  阶贝塞尔曲线  
( $n$  次向量多项式)

$$b^n(t) = b_0^n(t) = \sum_{j=0}^n b_j B_j^n(t)$$

塞爾曲线控制点  
( $\mathbb{R}^N$  中的向量)

伯恩斯坦多项式  
( $n$  次标量多项式)

# 贝塞尔曲线 – 一般代数形式：例子

□伯恩斯坦 (Bernstein) 形式的  $n$  阶贝塞尔曲线

$$b^n(t) = \sum_{j=0}^n b_j B_j^n(t)$$

□例子：假设  $n = 3$ , 且我们在  $\mathbb{R}^3$  空间中

□我们在 3D 中的控制点为

$$b_0 = (0, 2, 3), b_1 = (2, 3, 5), b_2 = (6, 7, 9), b_3 = (3, 4, 5)$$

□这些点定义了 3D 中的贝塞尔曲线，该曲线是  $t$  的三次多项式 (对于每一个  $t$ , 贝塞尔基均为标量)

$$b^n(t) = b_0(1-t)^3 + b_1 3t(1-t)^2 + b_2 3t^2(1-t) + b_3 t^3$$

# 贝塞尔曲线的特性

□ 插值端点 (Interpolates endpoints)

- 对于三阶贝塞尔曲线:  $b(0) = b_0$ ;  $b(1) = b_3$

□ 正切与端线段 (Tangent to end segments)

- 三阶的情况:  $b'(0) = 3(b_1 - b_0)$ ,  $b'(1) = 3(b_3 - b_2)$

□ 仿射变换性质 (Affine transformation property)

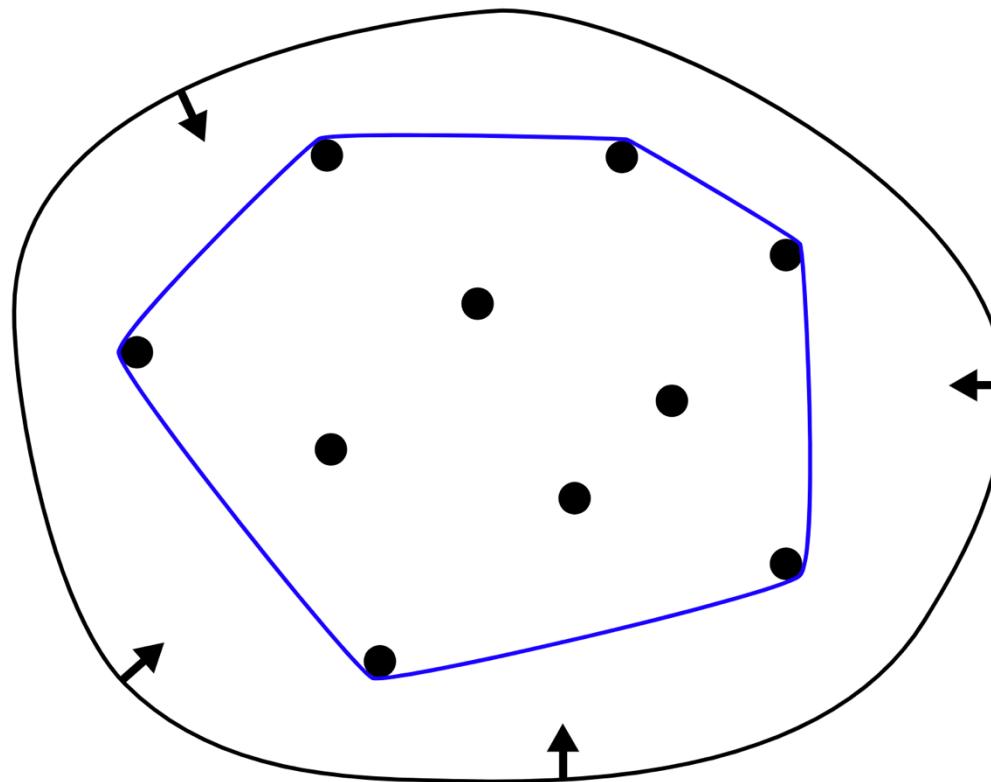
- 通过对控制点进行仿射变换来变换曲线

□ 凸包属性 (Convex hull property)

- 曲线在控制点的凸包内

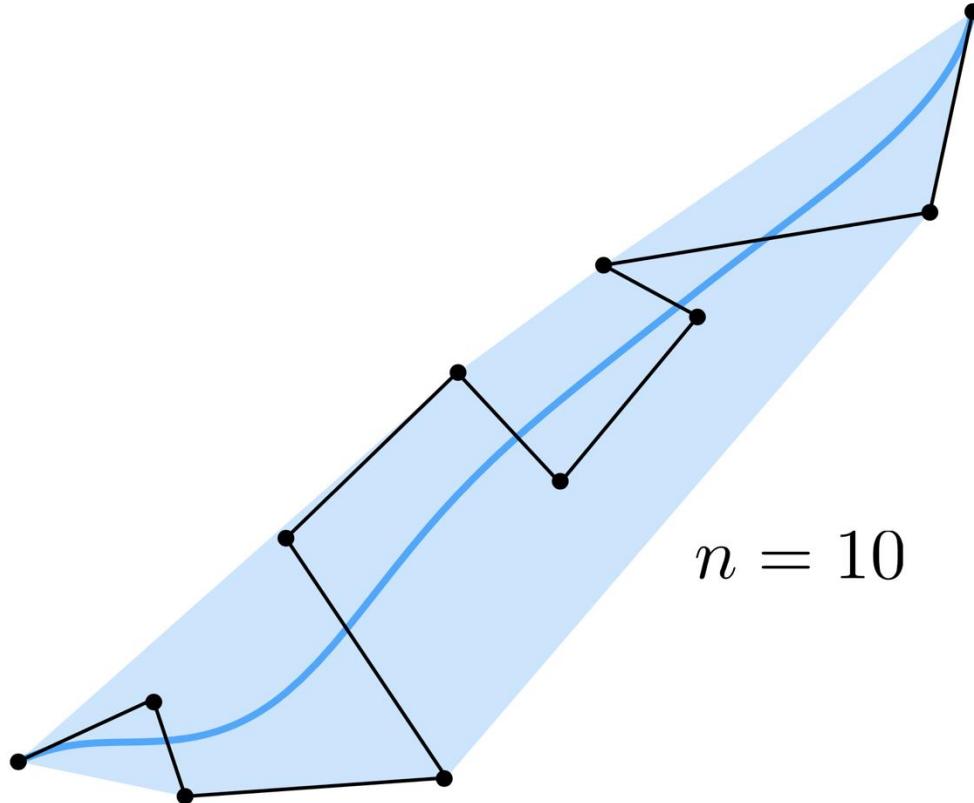
# 什么是凸包

口凸包就像是把一组点用橡皮筋包住，橡皮筋紧贴着最外围的点形成的形状就是凸包



# 高阶贝塞尔曲线

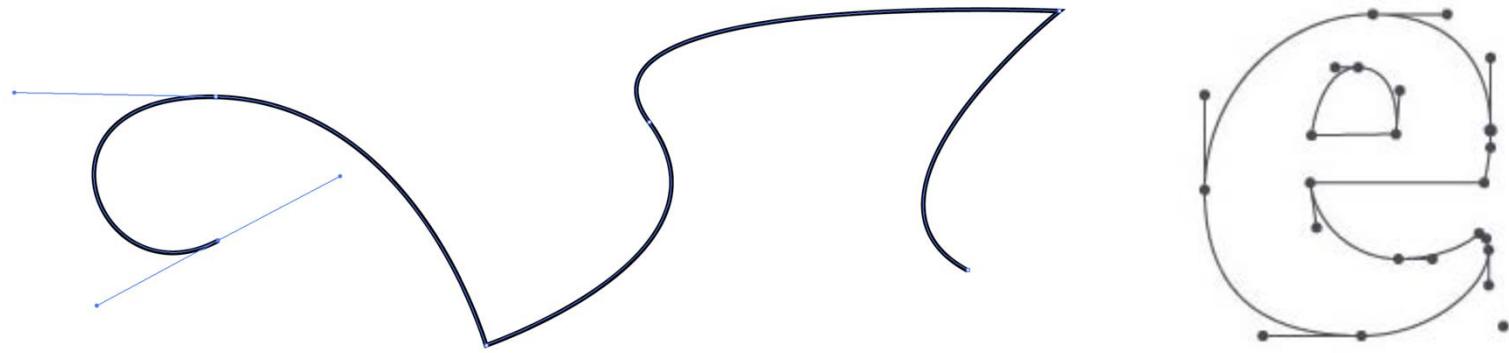
口高阶 Bernstein 多项式在插值上表现并不好



**Very hard to control!**

# 分段贝塞尔曲线 (显式表达)

- 另一种想法：将许多低阶 Bézier 曲线拼接在一起
- 广泛使用在字体、SVG 等



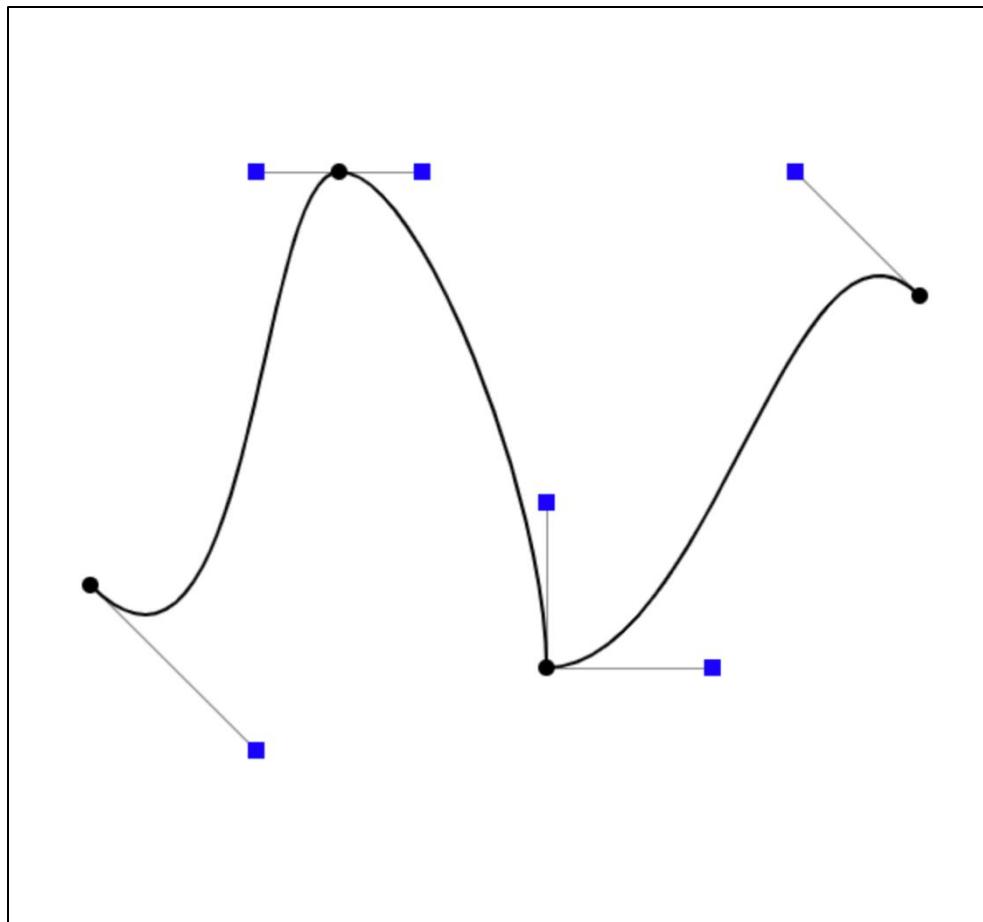
- 分段 Bézier 曲线：

**piecewise Bézier**

$$\gamma(u) := \gamma_i \left( \frac{u - u_i}{u_{i+1} - u_i} \right), \quad u_i \leq u < u_{i+1}$$

**single Bézier**

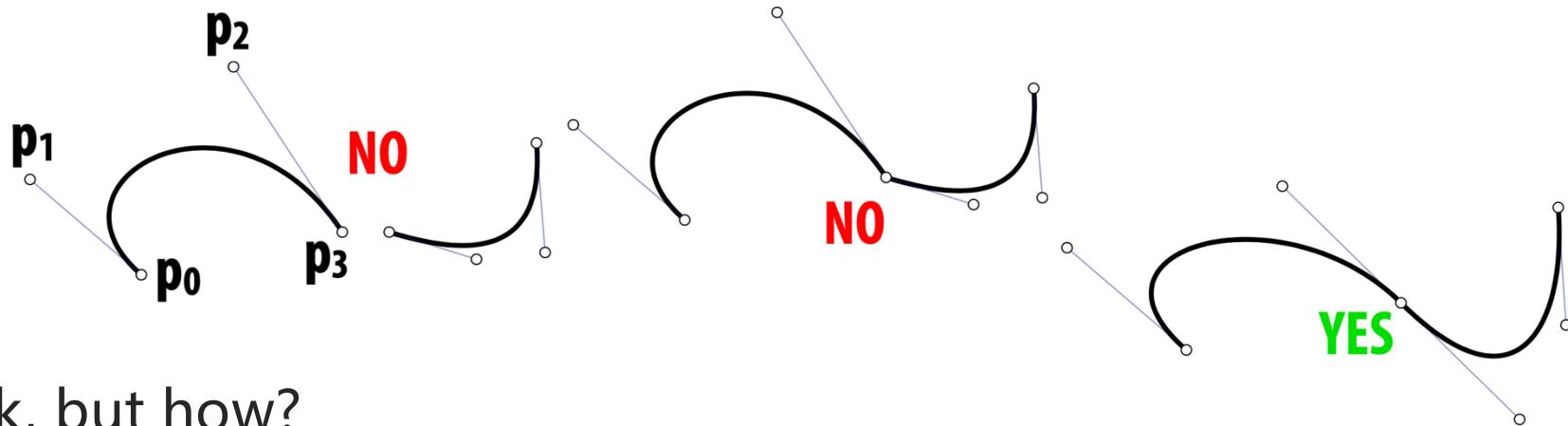
# Demo – 分段三阶贝塞尔曲线



David Eck, <http://math.hws.edu/eck/cs424/notes2013/canvas/bezier.html>

# 贝塞尔曲线 – 切线连续性 (tangent continuity)

□ 要获得无缝自然的曲线，需要点和切线对齐



□ Ok, but how?

□ 每条曲线是一个立方曲线

$$u^3 p_0 + 3u^2(1-u)p_1 + 3u(1-u)^2 p_2 + (1-u)^3 p_3$$

□ 希望每个分段的端点相交

□ 希望端点处的切线对齐

□ Q: 有多少约束 constraints 与自由度 degrees of freedom?

□ Q: 二次贝塞尔曲线呢？线性贝塞尔曲线呢？

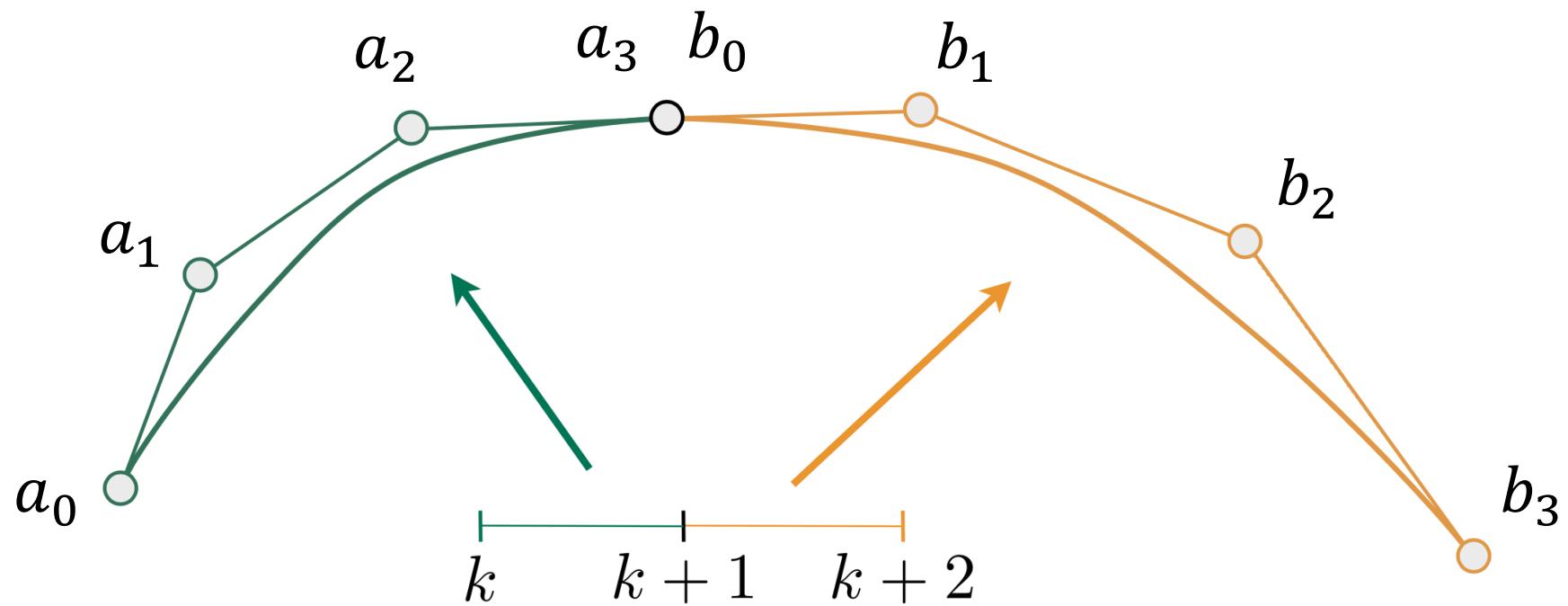
# 贝塞尔曲线 – 连续性 (Continuity)

两条贝塞尔曲线

$$\mathbf{a} : [k, k + 1] \rightarrow \mathbb{R}^N$$

$$\mathbf{b} : [k + 1, k + 2] \rightarrow \mathbb{R}^N$$

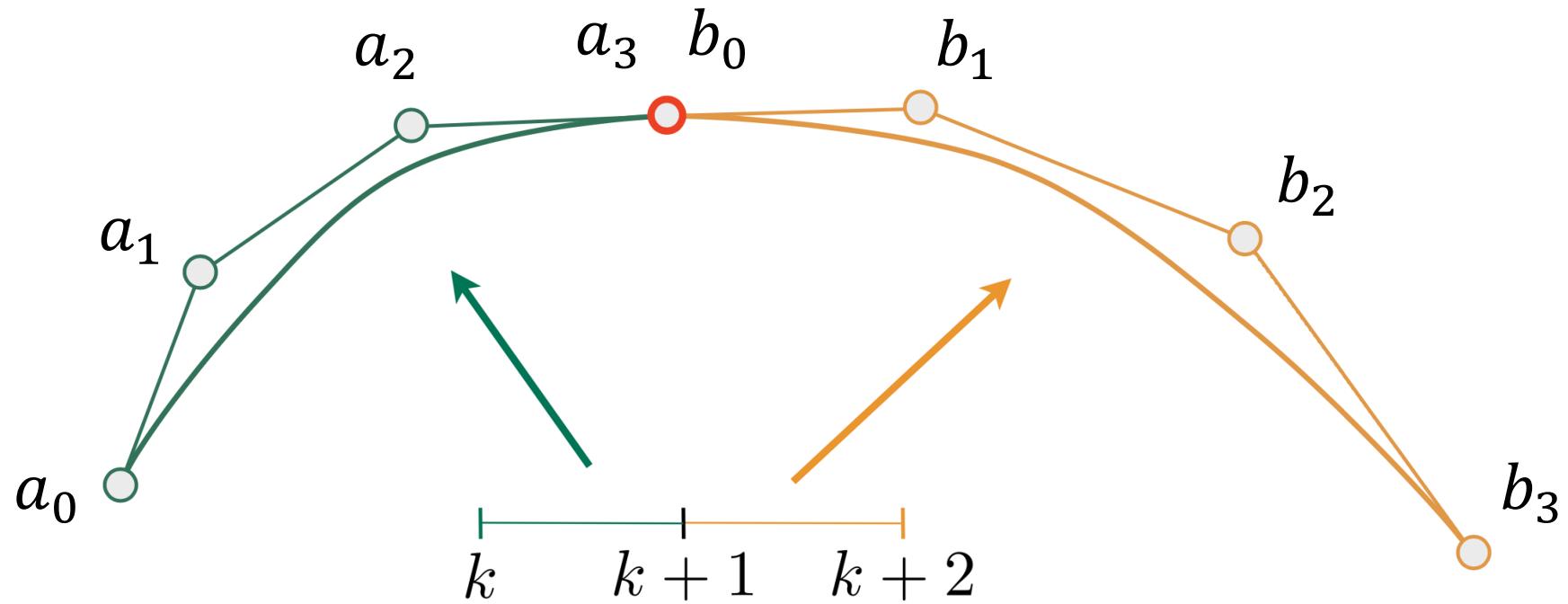
Assuming integer partitions here,  
can generalize



# 贝塞尔曲线 – 连续性 (Continuity)

□  $C^0$  连续性 (仅要求首位相连)

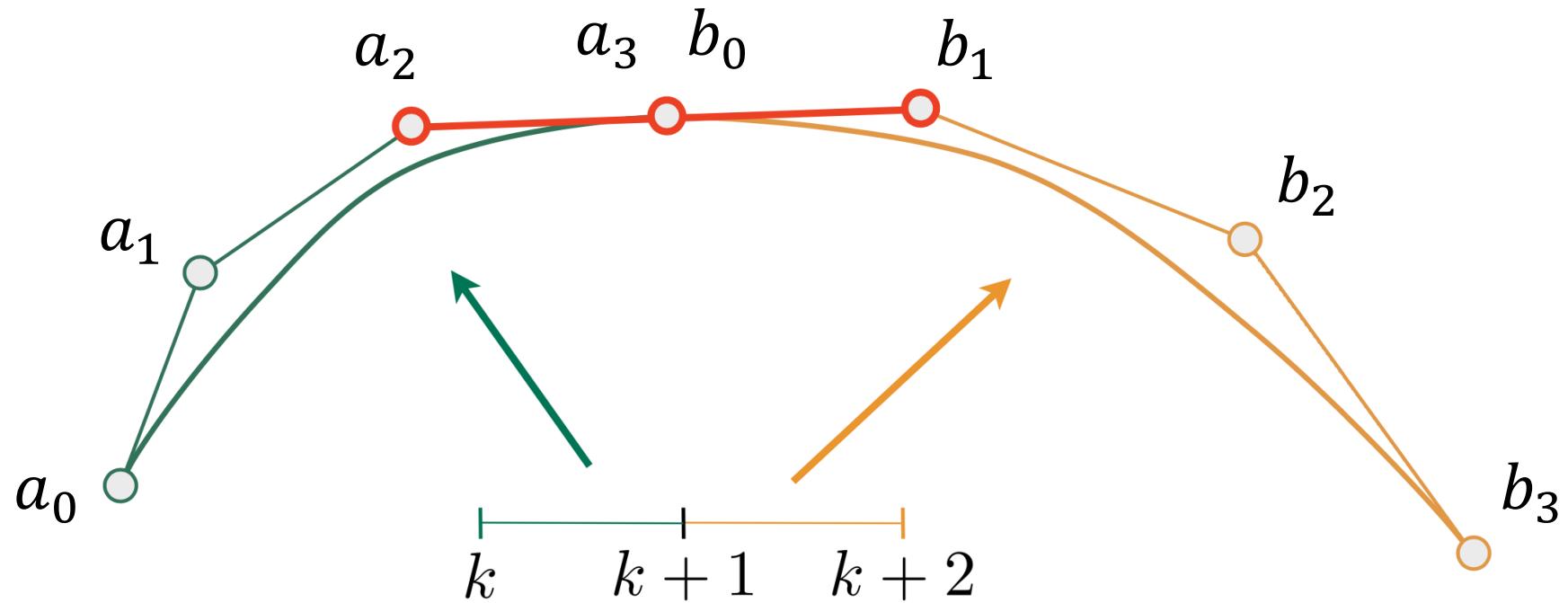
$$a_3 = b_0$$



# 贝塞尔曲线 – 连续性 (Continuity)

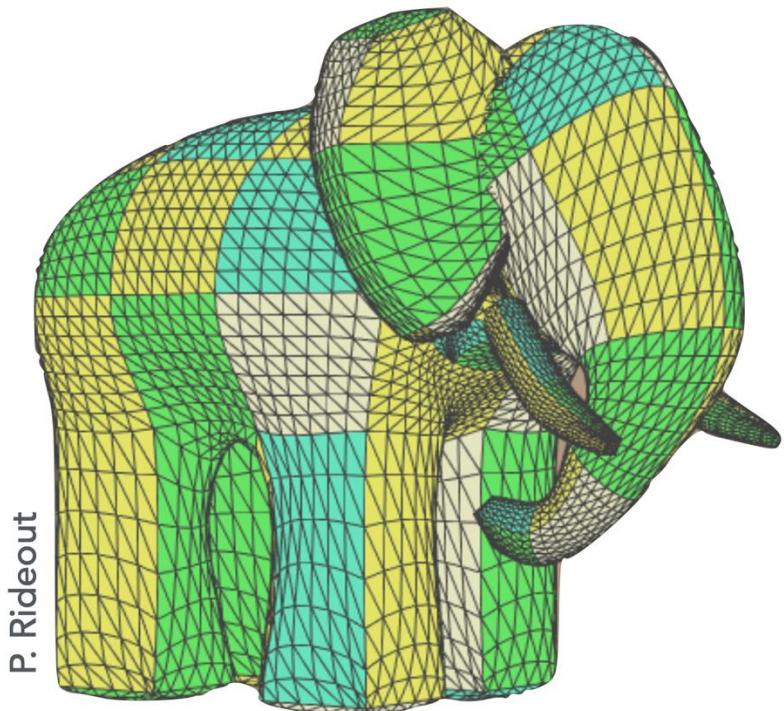
□  $C^1$  连续性 (仅要求首位相连)

$a_3 = b_0$  且线段  $(a_2, a_3)$  和  $(b_0, b_1)$  在一条直线上



# 贝塞尔曲面 (Bézier curve)

口将贝塞尔曲线延伸到曲面



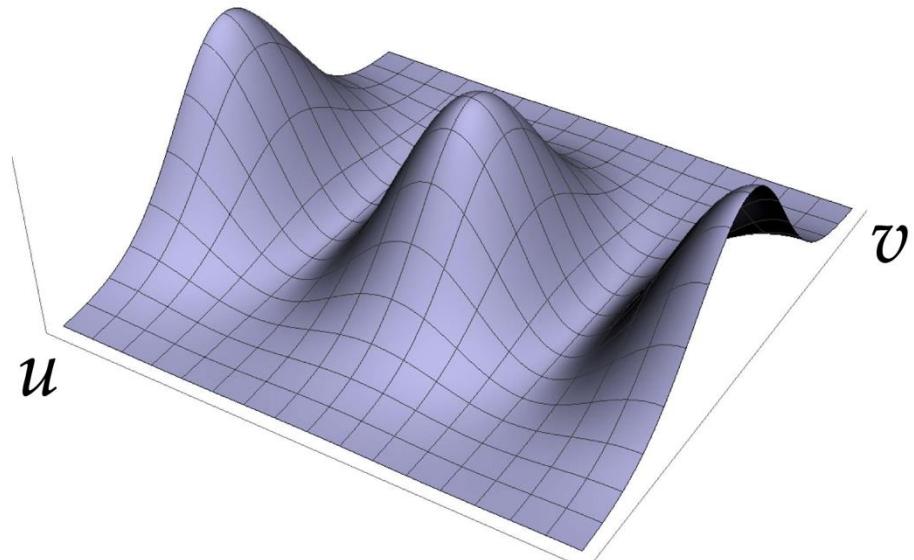
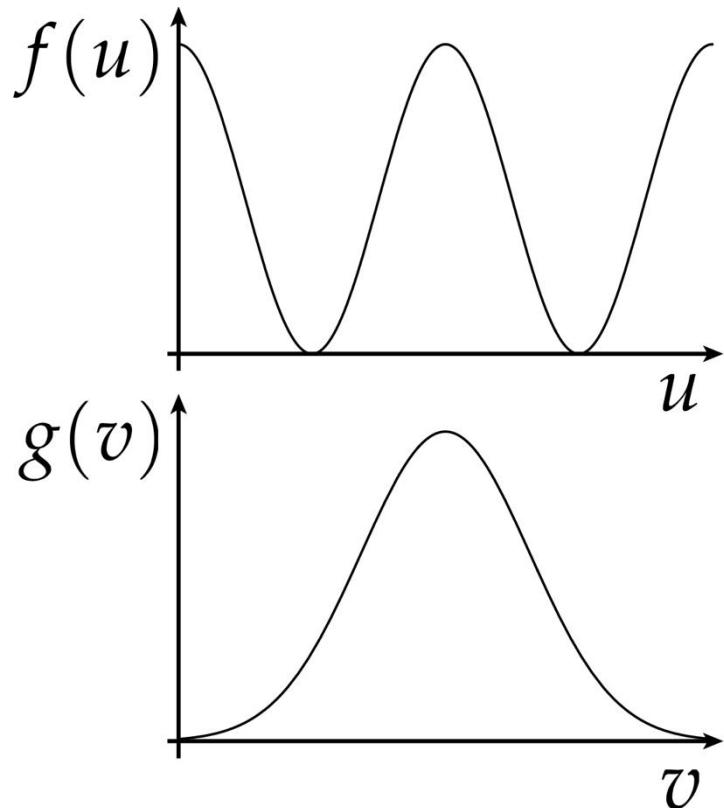
Ed Catmull's "Gumbo" model



Utah Teapot

# 张量积 Tensor product

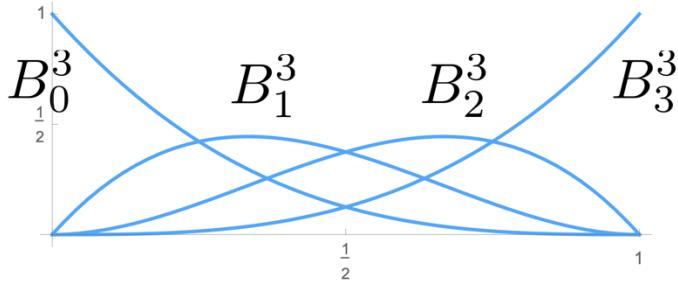
- 可以使用一对曲线来获得曲面 surfaces
- 由  $u$  处的曲线  $f$  和  $v$  处的曲线  $g$  的乘积给出的点  $(u, v)$  的值 (有时称为 “张量积” )：



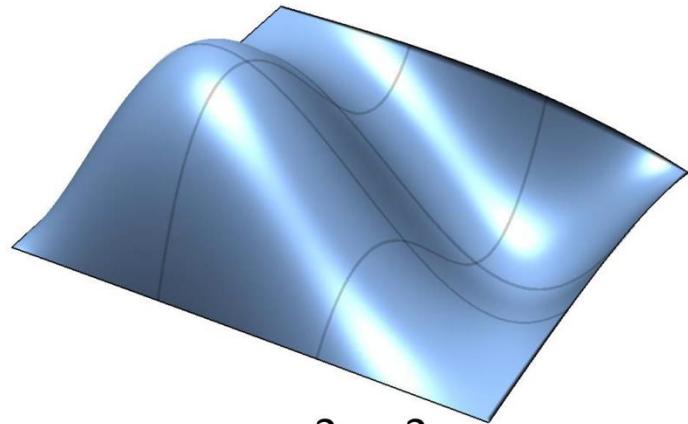
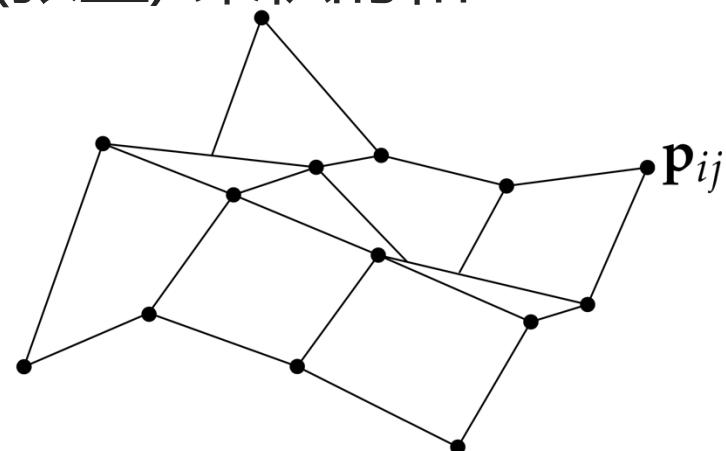
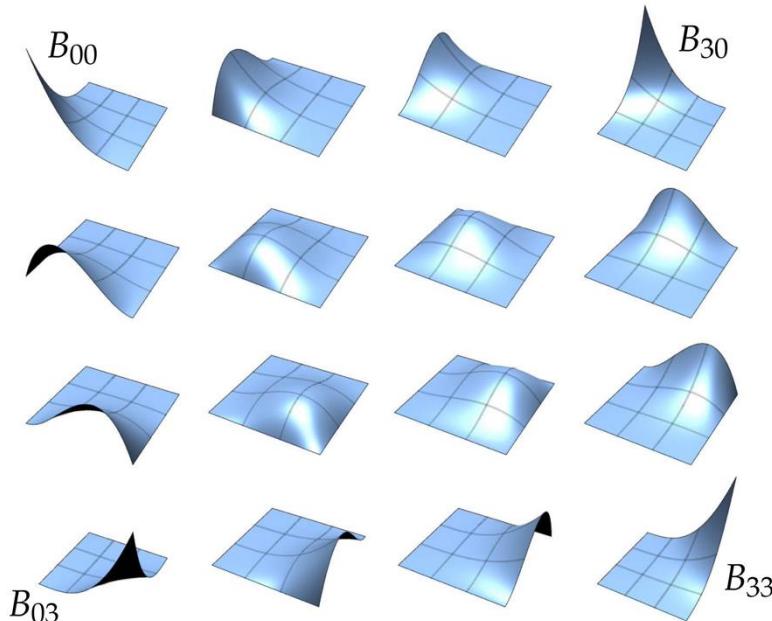
$$(f \otimes g)(u, v) := f(u)g(v)$$

# 贝塞尔片 (Bézier Patches)

口贝塞尔片是 Bernstein 基的 (张量) 乘积的和



$$B_{i,j}^3(u, v) := B_i^3(u) B_j^3(v)$$

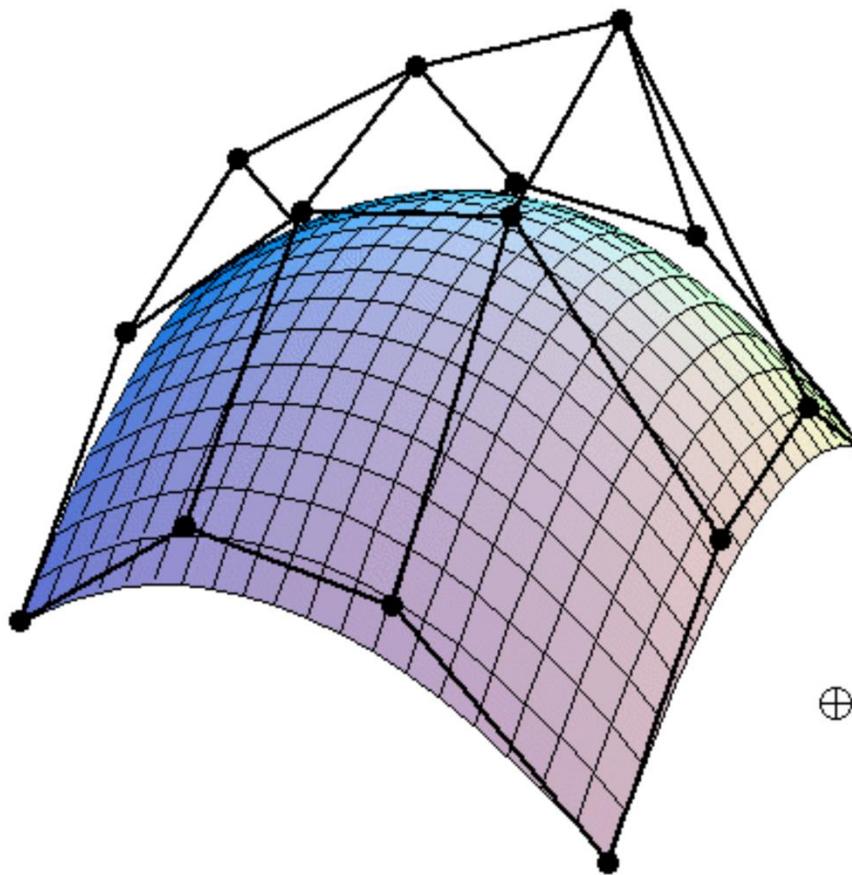


$$S(u, v) := \sum_{i=0}^3 \sum_{j=0}^3 B_{i,j}^3(u, v) \mathbf{p}_{ij}$$

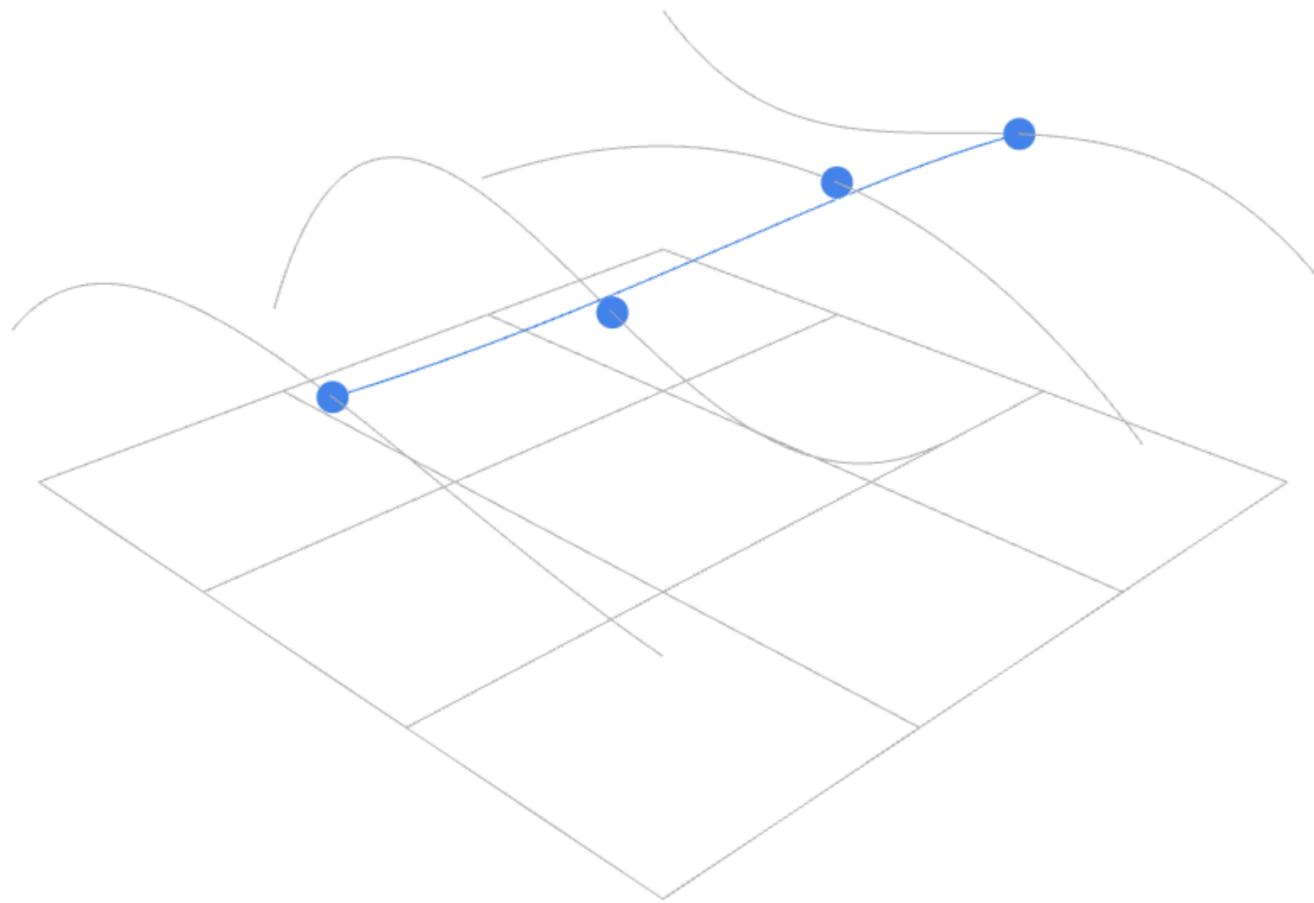
# 双三阶贝塞尔曲面片

## □ Bicubic Bézier Surface Patch

- 贝塞尔曲面及  $4 \times 4$  的矩阵控制点



# 双三阶贝塞尔曲面片 – 可视化



Animation: Steven Wittens, Making Things with Maths  
<https://acko.net/blog/making-mathbox/>

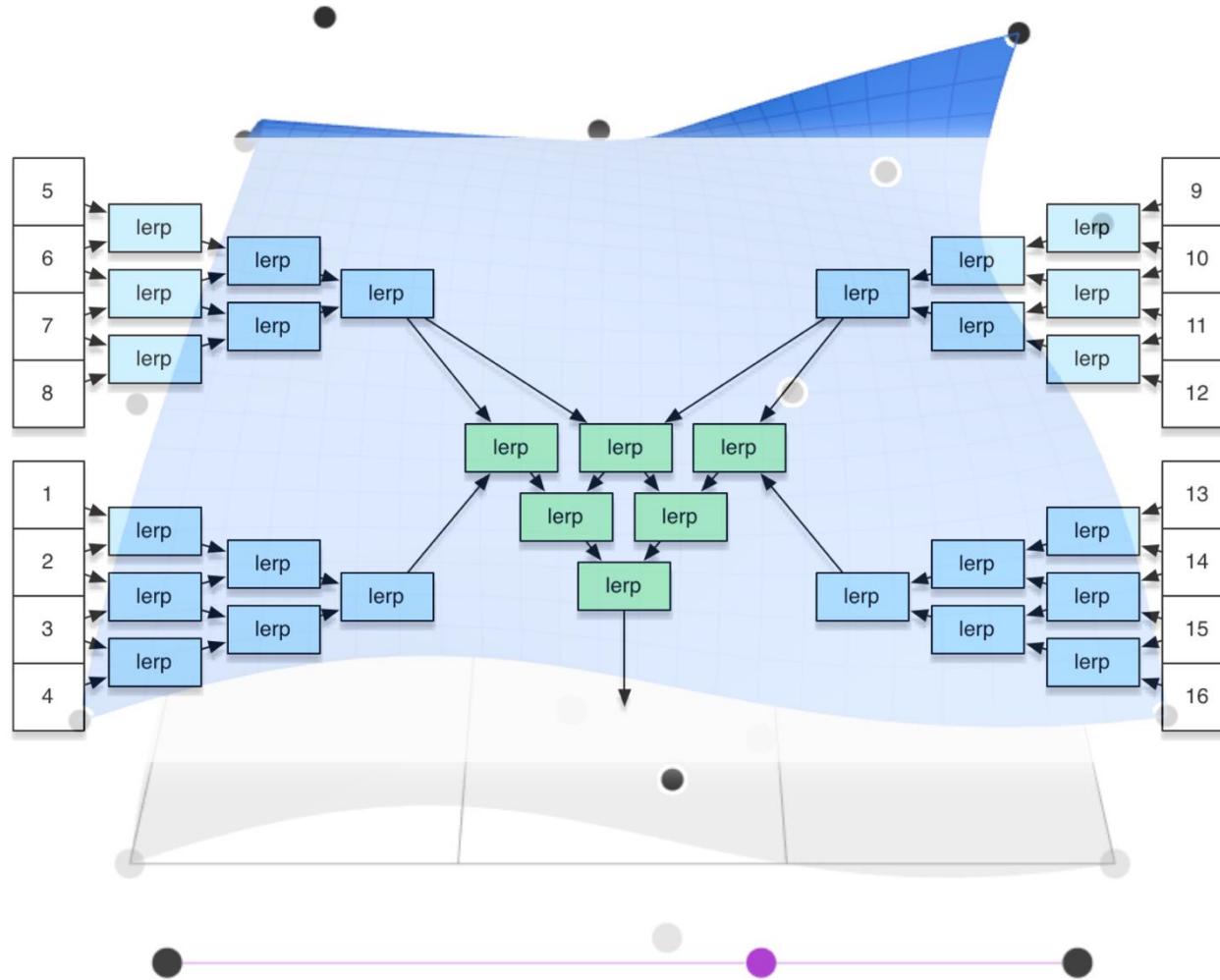
# 贝塞尔曲面

口贝塞尔曲面能构成复杂的几何图形



# 双三阶贝塞尔曲面

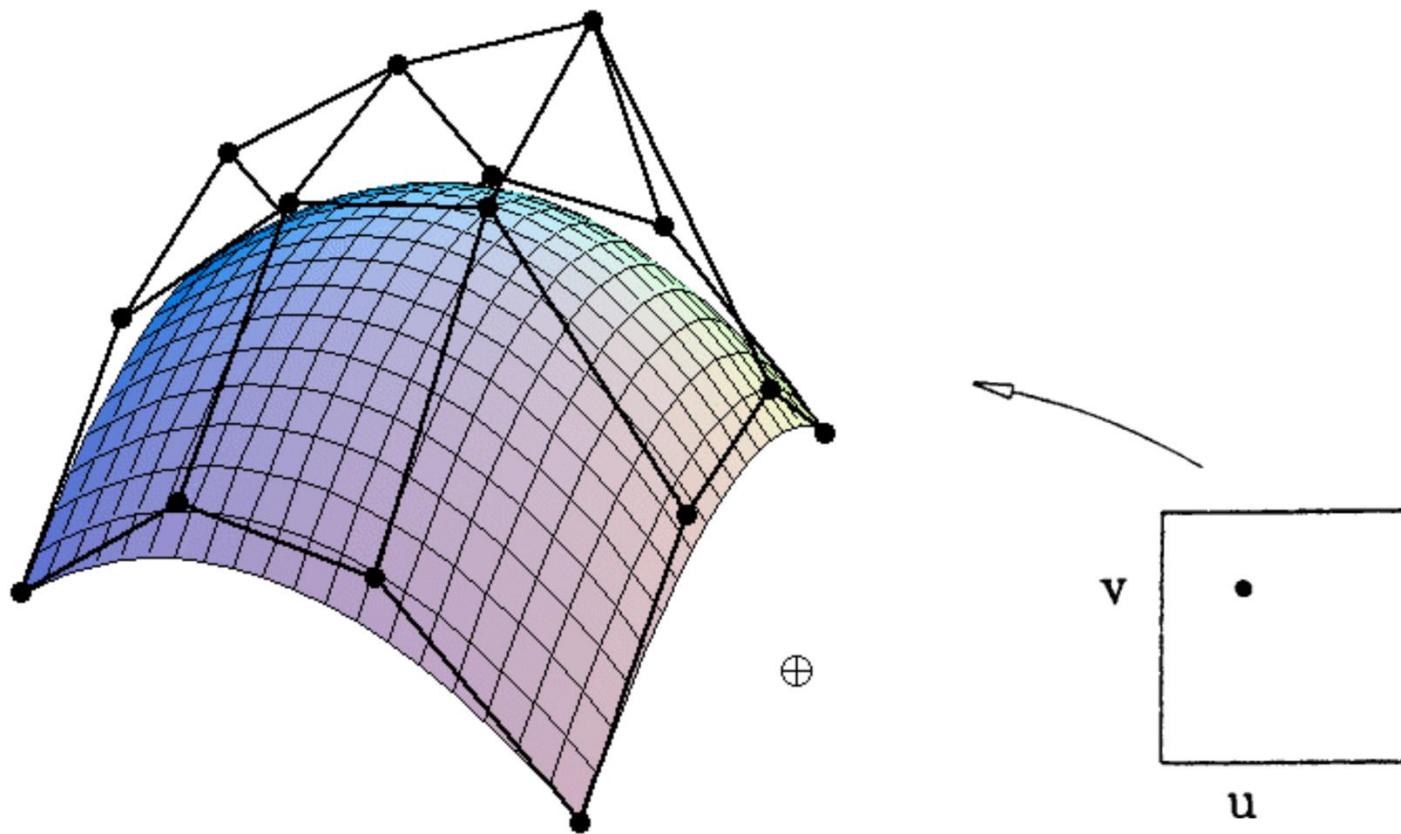
口基于控制点生成曲面上的点



# 基于参数 $(u, v)$ 计算曲面上的位置

对于双立方贝塞尔曲面片，

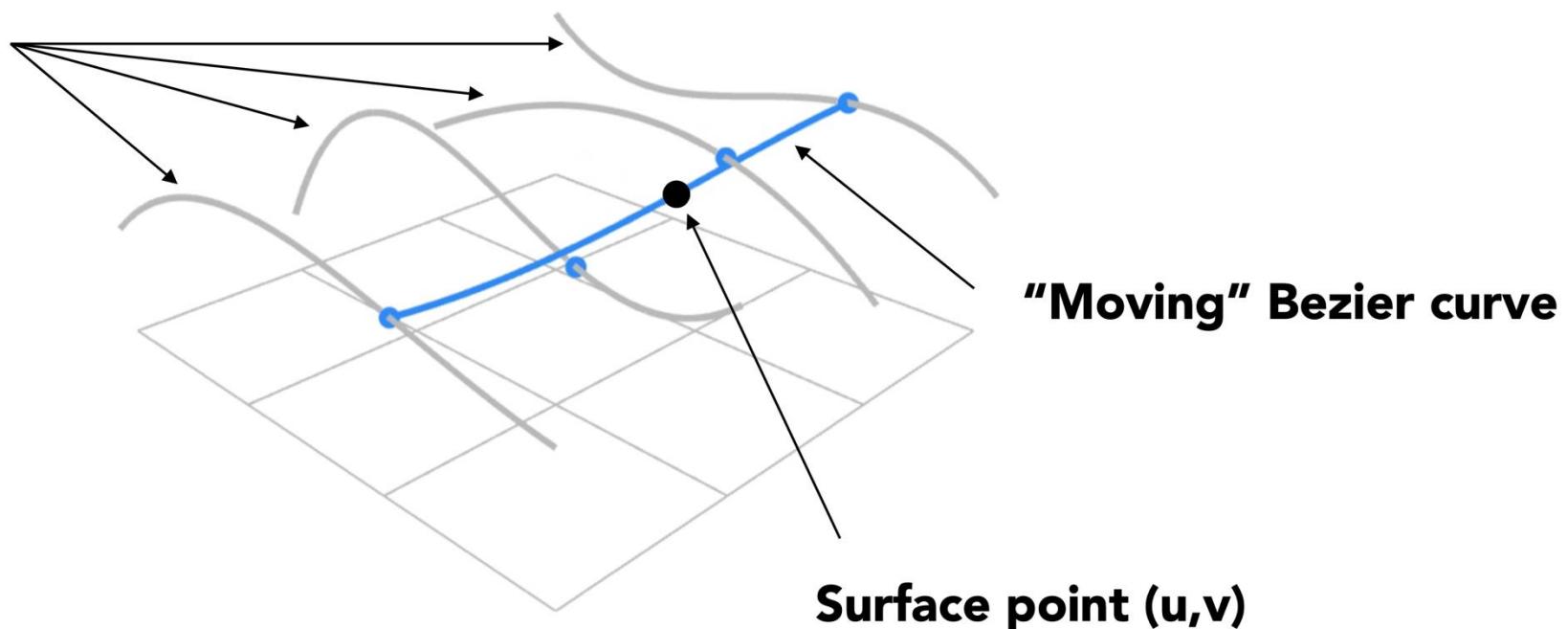
- 输入： $4 \times 4$  控制点
- 输出：由  $[0,1]^2$  中的  $(u, v)$  参数化的 2D 曲面



# 基于参数 $(u, v)$ 计算曲面上的位置

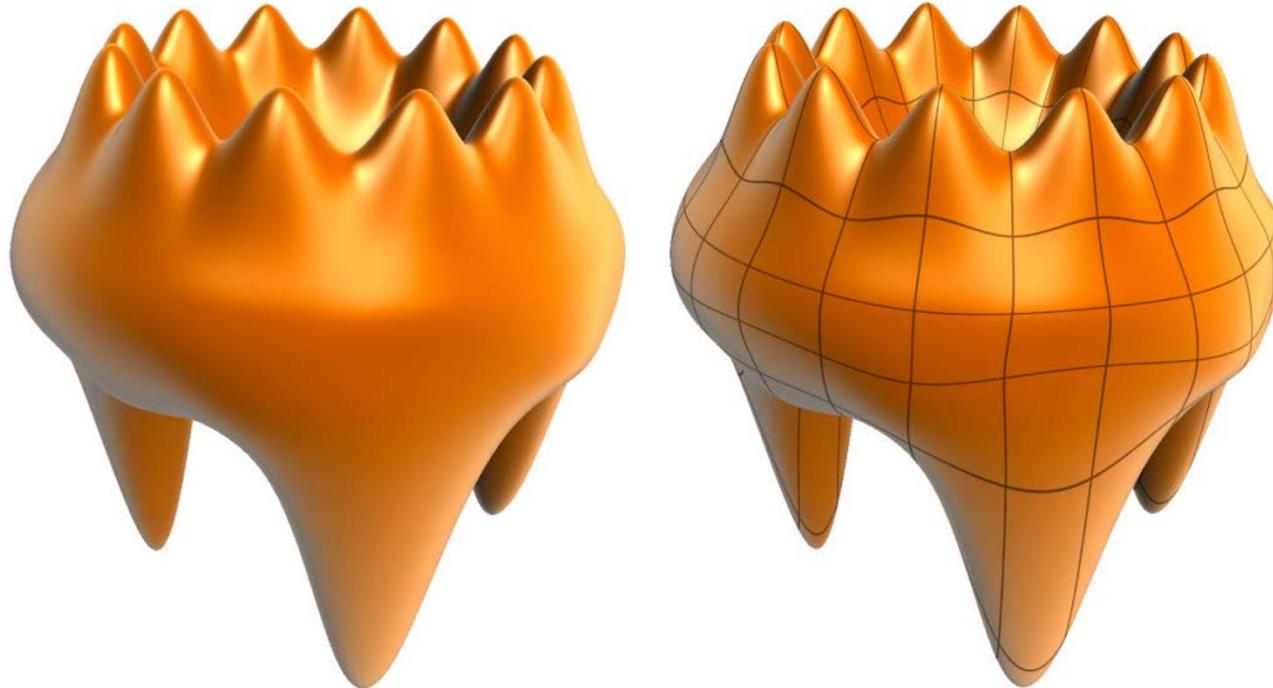
口目标：计算对应于  $(u, v)$  的曲面位置

- 使用 de Casteljau 算法计算  $u$  中 4 条贝塞尔曲线上的每一条上的点  $u$
- 这为“移动”的贝塞尔曲线提供了 4 个控制点
- 再使用 de Casteljau 评估“移动”曲线上的点  $v$



# 贝塞尔曲面 (Bézier Surface)

正如连接贝塞尔曲线，可以连接贝塞尔片来获得曲面：



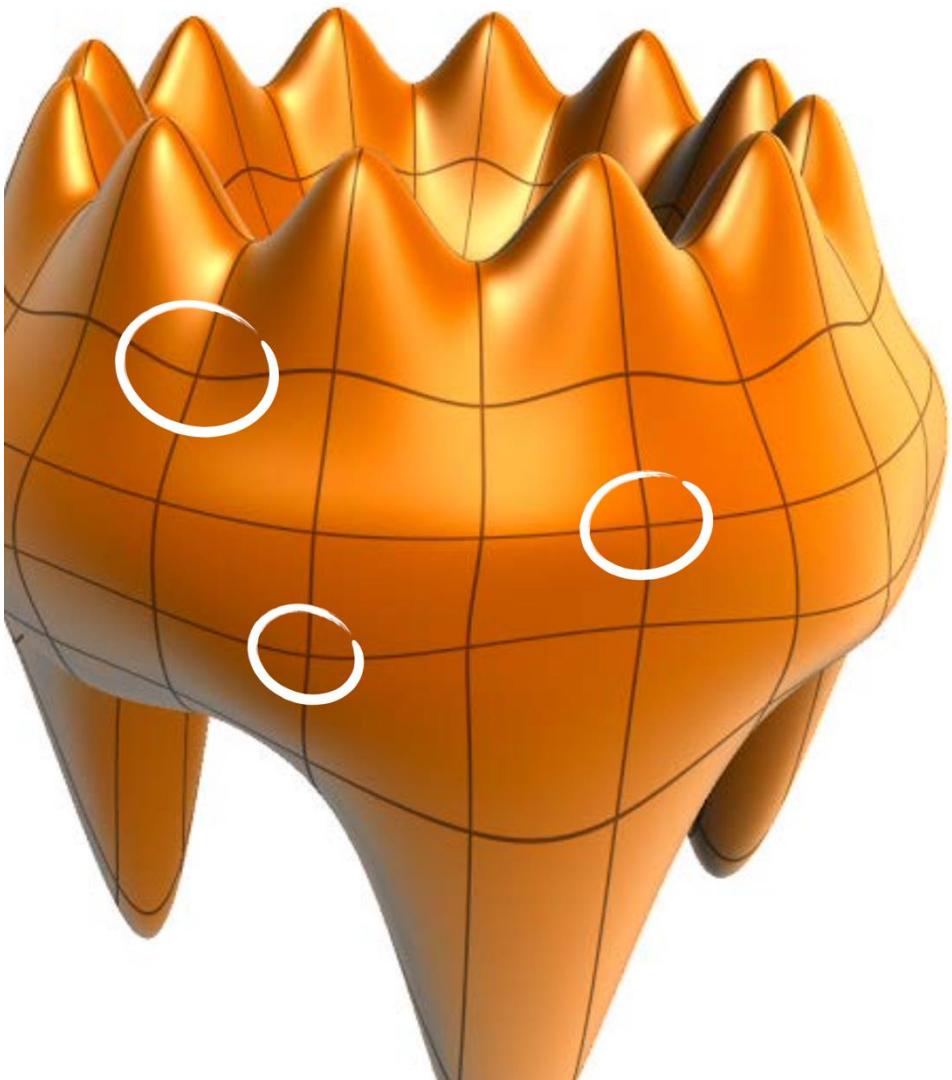
非常容易绘制：只需将每个片放进规则的  $(u, v)$  网格中

**Q：我们能否总得到切线连续性？**

**思考：有多少约束？有多少自由度？**

注意到最后一张图片有什么  
特别之处吗？

# 贝塞尔片太简单了



注意到每个顶点周围正好有四个片相交！

在实际中，这个条件约束性太强了

为了创造具有良好连续性的有趣形状，我们需要使用能提供更灵活连接方式的曲面片

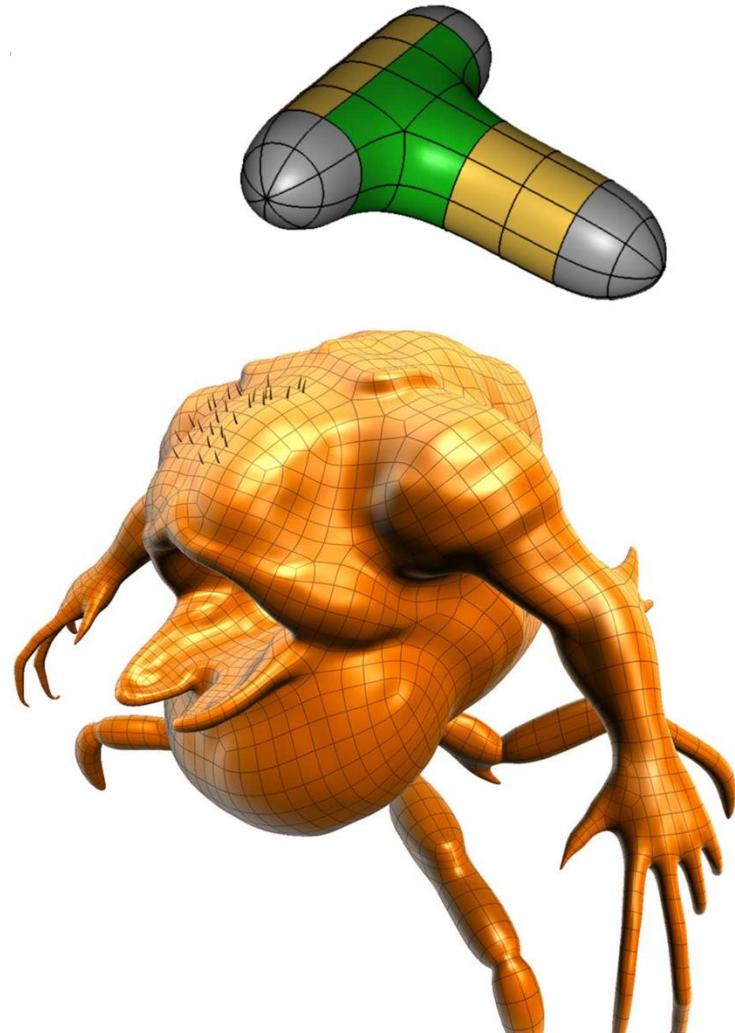
# 样条线面片方案 (Spline patch schemes)

- 有很多可替代的方案
- NURBS, Gregory, Pm, polar...

□折中 tradeoffs:

- 自由度 (degrees of freedom)
- 连续性 (continuity)
- 编辑难度 (difficulty of editing)
- 评估成本 (cost of evaluation)
- 一般性 (generality)
- ...

□As usual: pick the right tool for the job!

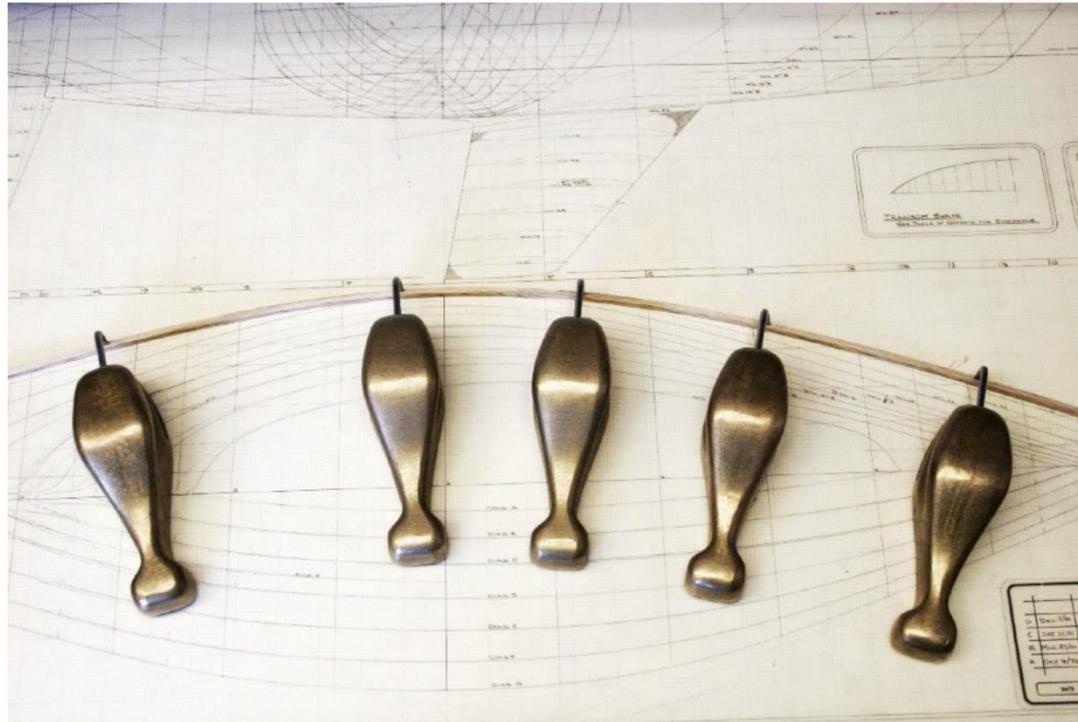


# Spline 样条曲线

口样条是一种连续曲线，它经过指定的一组点，并且在这些点上具有一定数量的连续变化率（导数）

口简单来说，样条就是受控制的曲线

口贝塞尔曲线就是一种典型的样条曲线



# Spline 样条曲线

□ 其他形式的样条曲线包括

- **B 样条曲线 (B-Splines)**

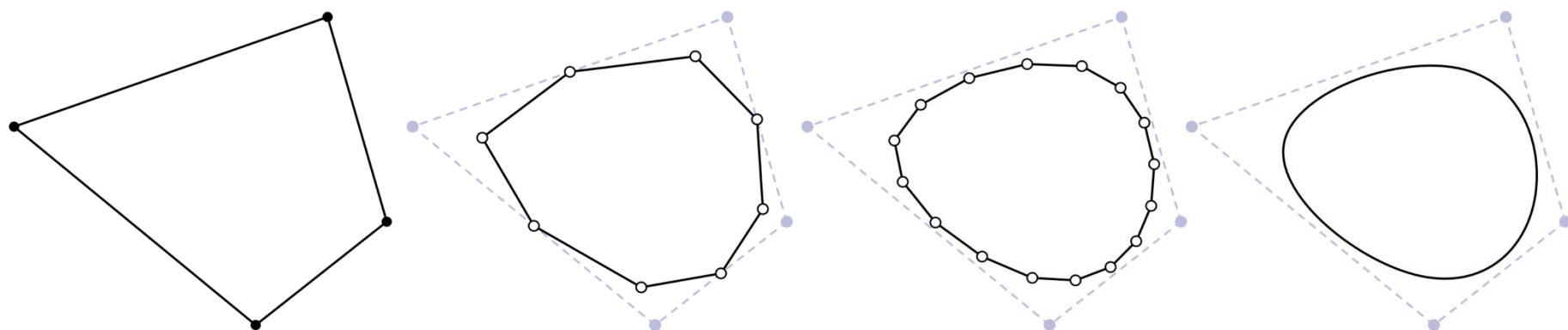
- Basis splines 的缩写
- 一种更为通用的样条曲线
- 相比于贝塞尔曲线， B 样条曲线能更好地控制曲线的**局部形状**

- **NURBS 曲线 (Non-uniform rational B-splines)**

- 一种特殊类型的 B 样条曲线
- 能表示更复杂的形状，如圆形和椭圆形

# 细分 Subdivision

- 另一种获得曲线/曲面的方法：细分
- 从“控制曲线 (control curve)”开始
- 反复拆分，取加权平均数获得新的点
- 仔细选择平均规则，可以得到漂亮的极限曲线
  - 能获得与某些样条方法一样的曲线



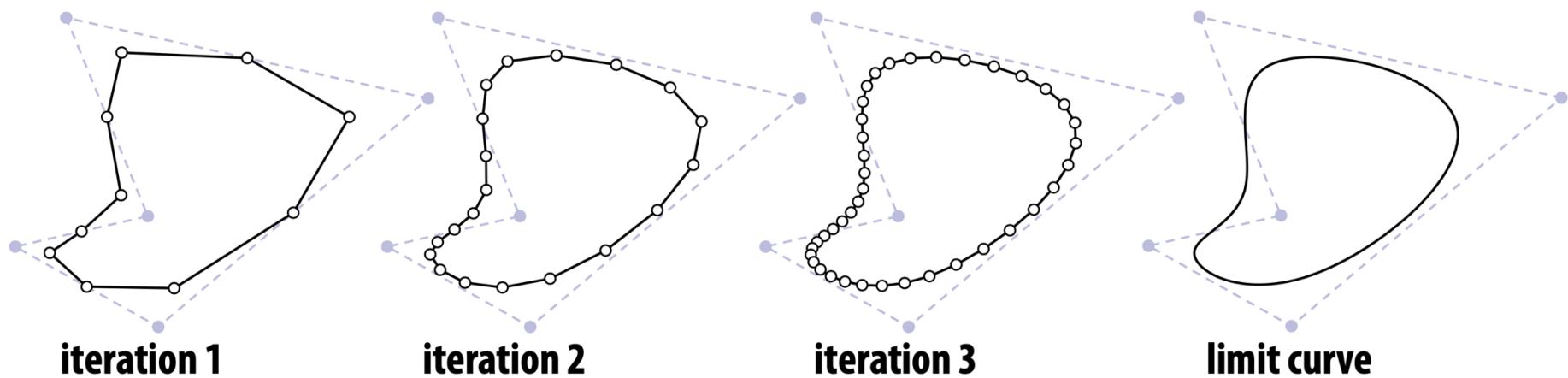
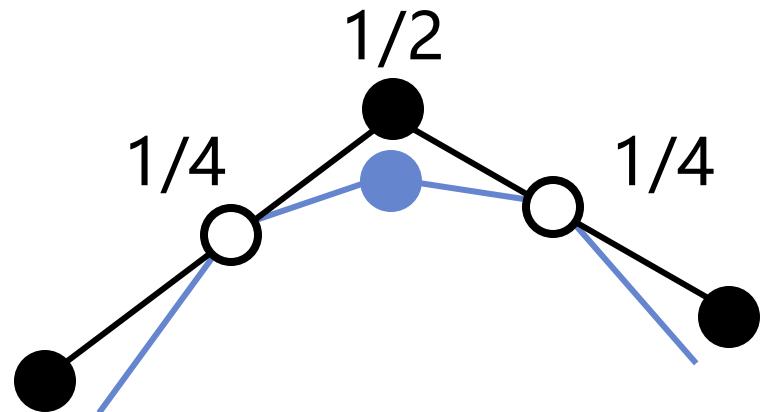
**Q: 细分是显式还是隐式的?**

# 细分 – 例子

□ 一种可能的方案：Lane-Riesenfeld

- 插入每条边的中点
- 使用 Pascal 三角形的第  $k$  行归一化的值作为邻居的权重
- 比如， $k = 2$  时的权重为  $(1/4, 1/2, 1/4)$
- 极限是  $k + 1$  次的 B 样条曲线

$k=0:$	1
$k=1:$	1 1
$k=2:$	1 2 1
$k=3:$	1 3 3 1



# 细分曲面 (显式的)

□ 从粗略多边形网格开始 (控制框架, control cage)

□ 细分每个元素

□ 通过局部平均更新顶点

□ 许多可能的规则:

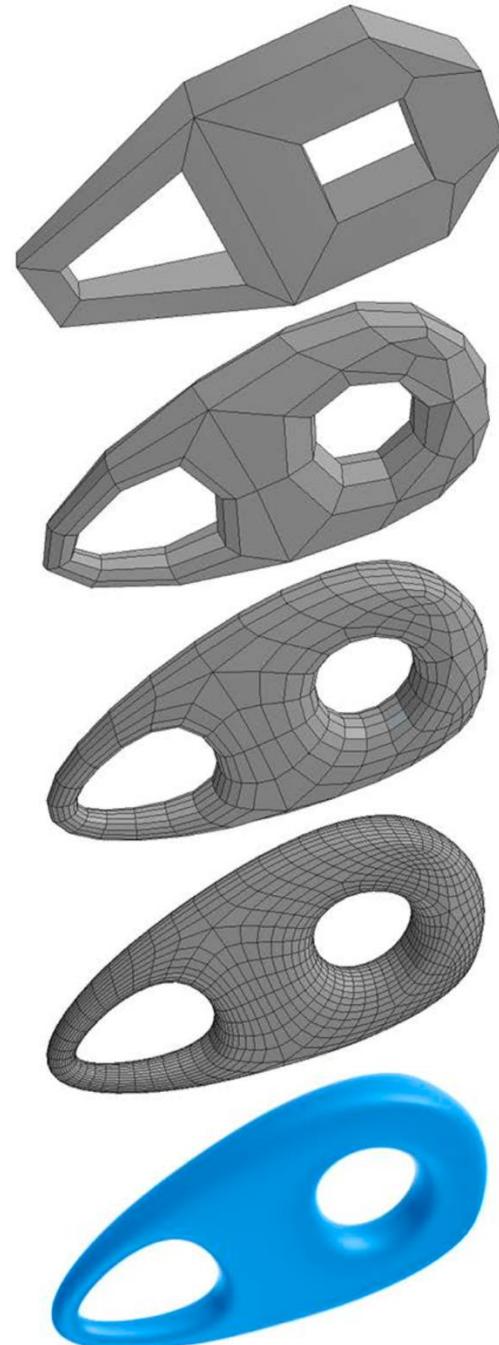
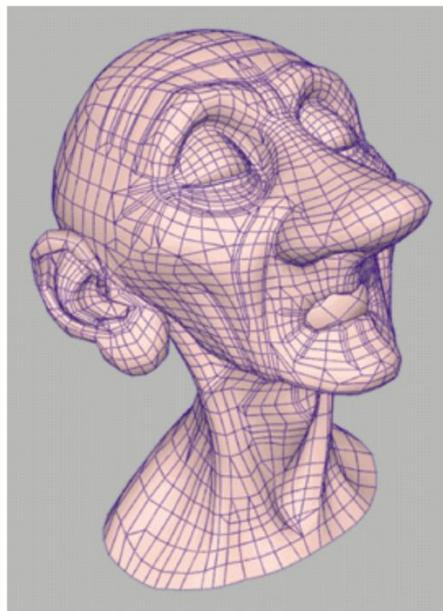
- Catmull-Clark (quads)
- Loop (triangles)
- ...

□ 常见的问题

- 插值还是近似?
- 顶点是否有连续性?

□ 建模比样条曲线更容易; 难以逐点评估

□ 在实践中广泛使用 (2019奥斯卡奖! )



# 电影中的细分 (皮克斯：格里的游戏)





中山大學 软件工程学院  
SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

谢谢

陈壮彬  
软件工程学院  
[chenzhb36@mail.sysu.edu.cn](mailto:chenzhb36@mail.sysu.edu.cn)