几何表示

北京大学 前沿计算研究中心

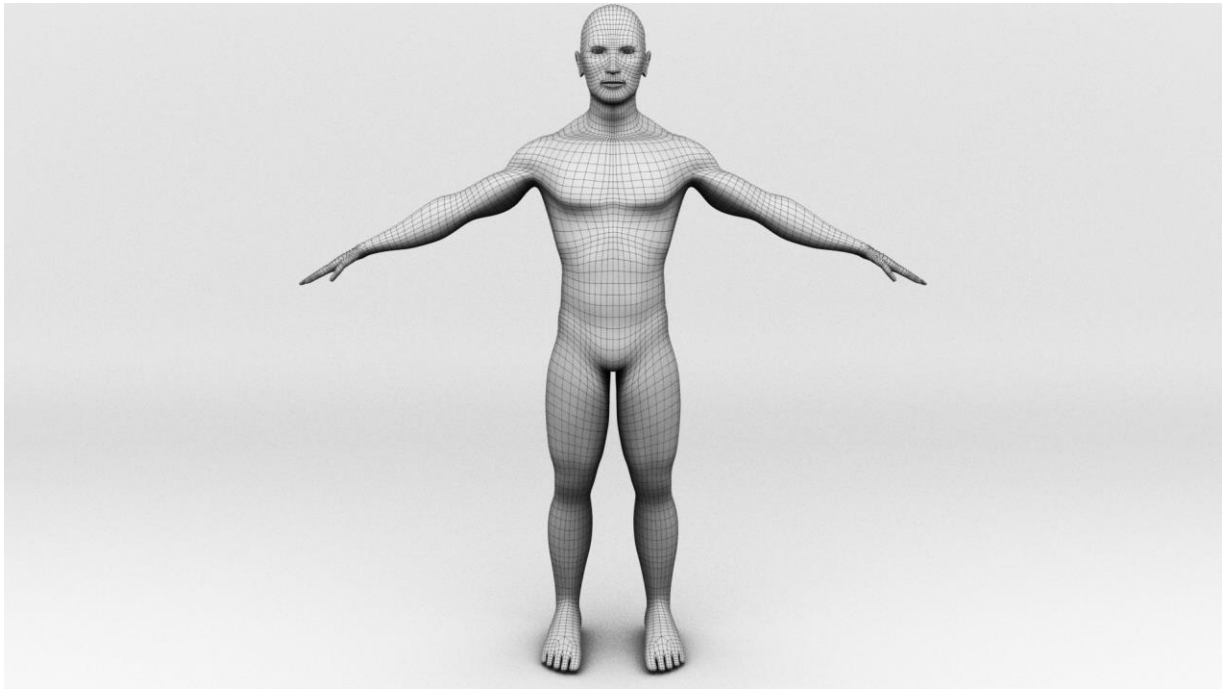
刘利斌

主要内容

- 几何的定义
- 几何表示
  - 点云 (point cloud)
  - 多边形网格 (polygon mesh)
  - 参数曲面
    - 贝塞尔曲线 Bézier Curve
    - B样条曲面与NURBS
  - 细分网格 (Subdivision Surface)
  - 隐式曲面
    - 绘制：体素/marching cube
- 参考：GAMES-102 (https://games-cn.org/games102/)

* Thank Prof. Baoquan Chen for many of the slides

What is geometry?



Geometry model of a human

What is geometry?

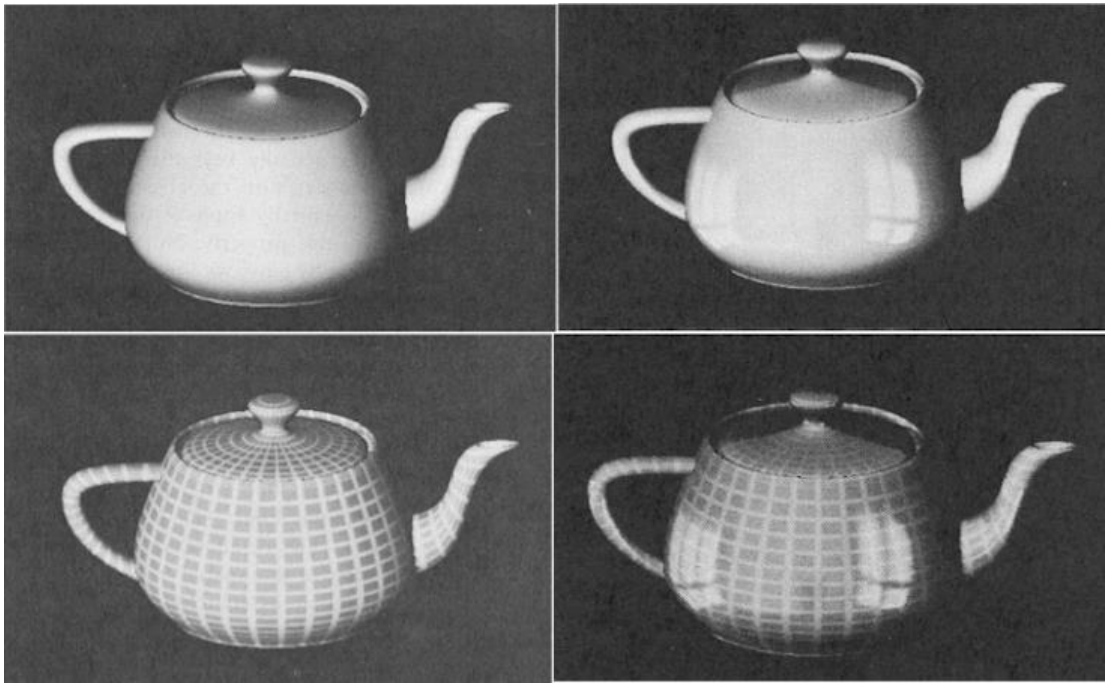
ge·om·e·try “measurement of earth”

1. The study of shapes, sizes, patterns and positions.

2. The study of spaces where some quantity (lengths, angles, etc.) can be measured.

Examples of geometry – the Utah Teapot

The actual teapot, now displayed at
the Computer History Museum
in Mountain View, California



Martin Newell's early teapot renderings
(Martin created teapot model in 1975 using Bézier curves)

Examples of geometry – The Stanford Bunny

Mesh created by reconstruction from laser scans

Photograph of the scanned statue (purchased by Greg Turk at a store on University Ave in 1994)

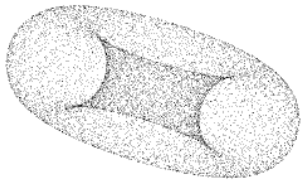
How to encode geometry on a computer?

- Explicit
  - Point cloud
  - Polygon mesh
  - Subdivision surface
  - …
- Implicit
  - Level set
  - Algebraic surface
  - …

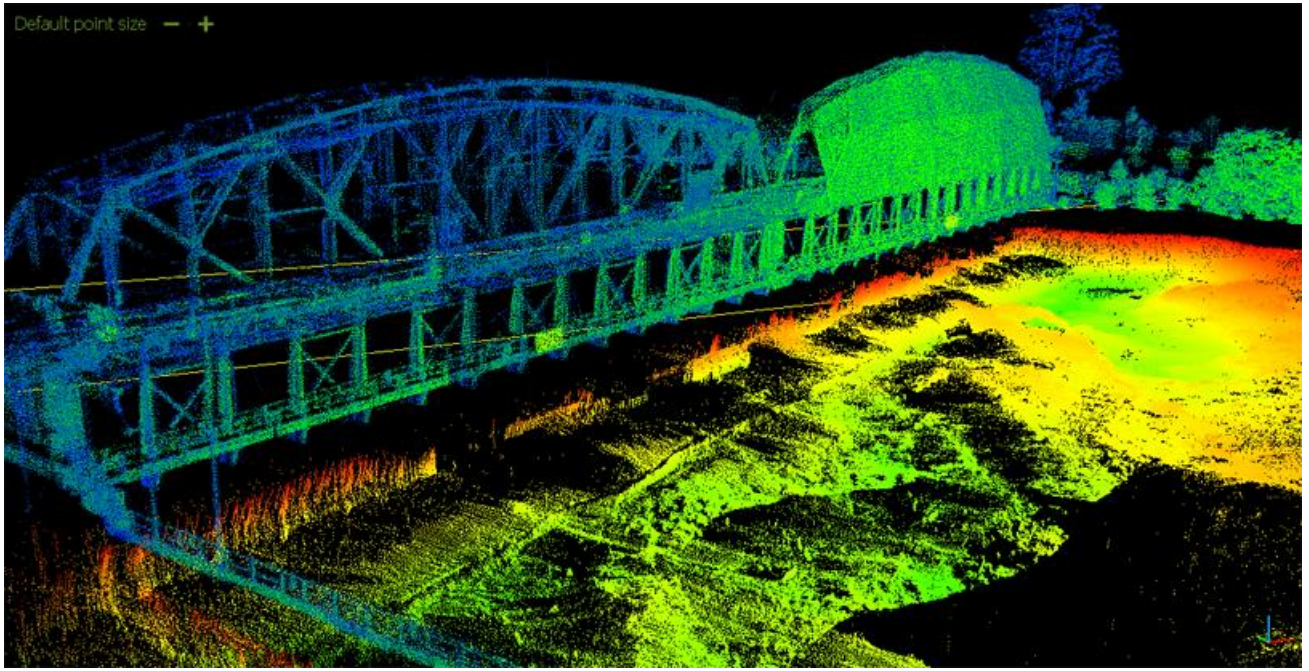
Each choice best suited to a different task/type of geometry

Point Cloud

Point cloud



- Scanned by LiDAR or other scanners
- Points on object surface



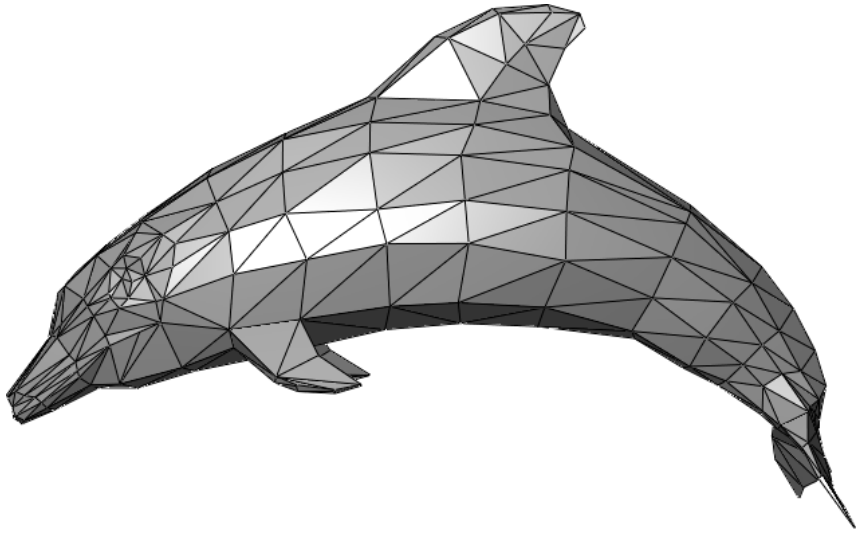
Point cloud

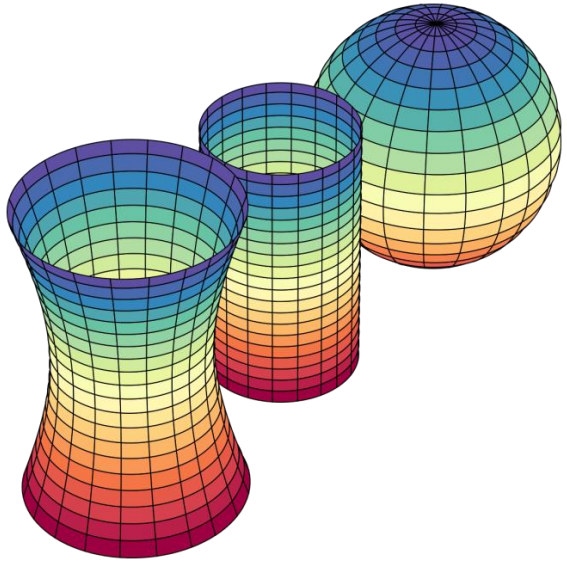
Point Cloud Scan Data into 3D AutoCad Model Animation - 3D Laser Scanning and 3D Modeling
https://www.youtube.com/watch?v=qv1SHm9qzag

Polygon Mesh

Polygon mesh

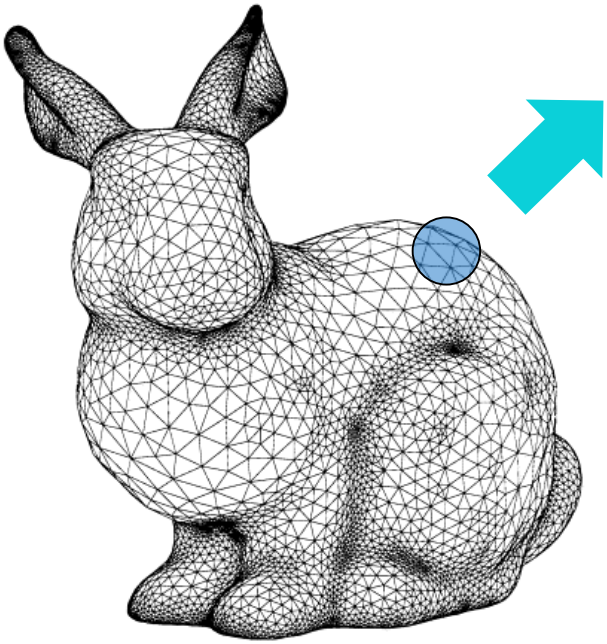
- Store vertices and polygons (triangles and quads most often)
- Perhaps most common representation in graphics
- Easy (?) to do processing, simulation and adaptive sampling
- More complicated data structures

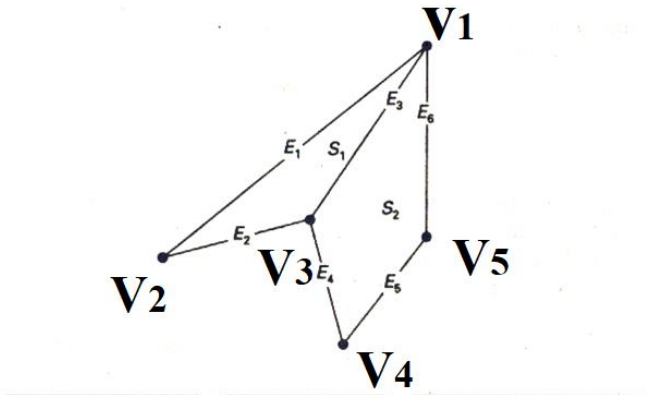




Polygon mesh

- Vertex table, edge table and polygon-surface table
- BTW, Euler's theorem on *polyhedrons*: $v - e + f = 2$



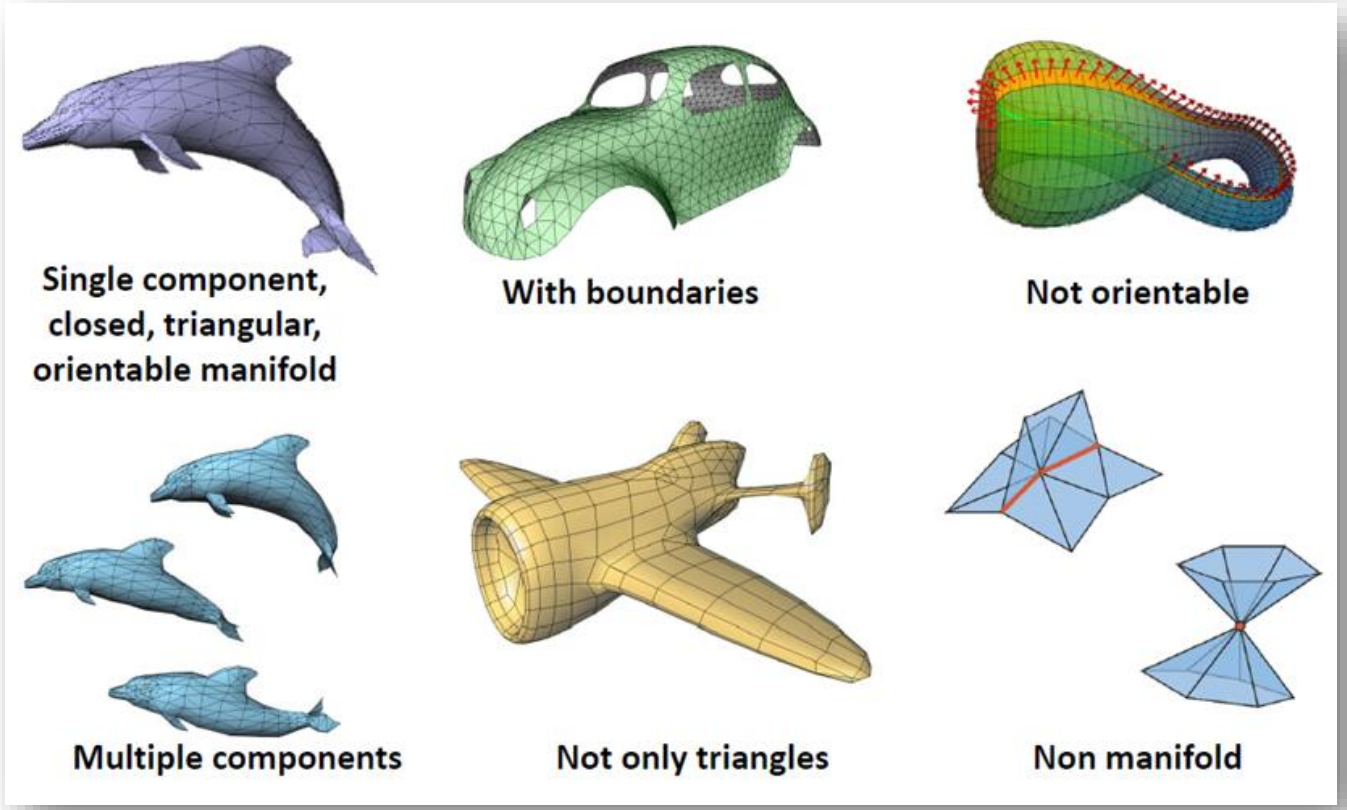


VERTEX TABLE	
V_1:	x_1, y_1, z_1
V_2:	x_2, y_2, z_2
V_3:	x_3, y_3, z_3
V_4:	x_4, y_4, z_4
V_5:	x_5, y_5, z_5

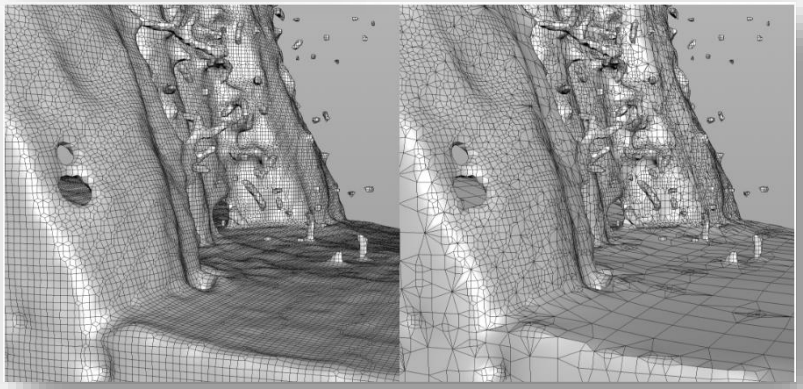
EDGE TABLE	
E_1:	V_1, V_2
E_2:	V_2, V_3
E_3:	V_3, V_1
E_4:	V_3, V_4
E_5:	V_4, V_5
E_6:	V_5, V_1

POLYGON-SURFACE TABLE	
S_1:	E_1, E_2, E_3
S_2:	E_3, E_4, E_5, E_6

Examples of polygon mesh



Examples of polygon mesh



Triangle mesh

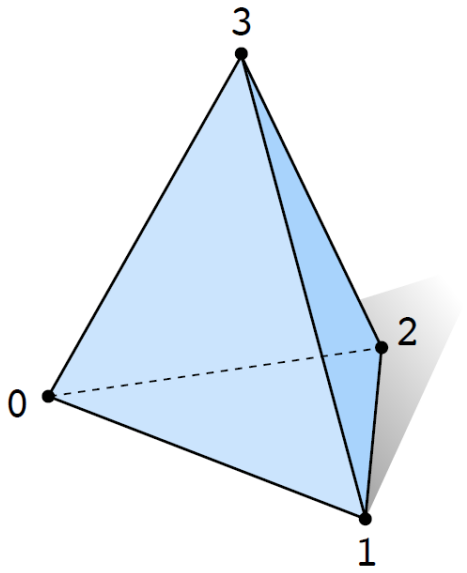
- Store vertices as triples of coordinates (x, y, z)
- Store triangles as triples of indices (I, j, k)
- E.g., for a tetrahedron

VERTICES

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

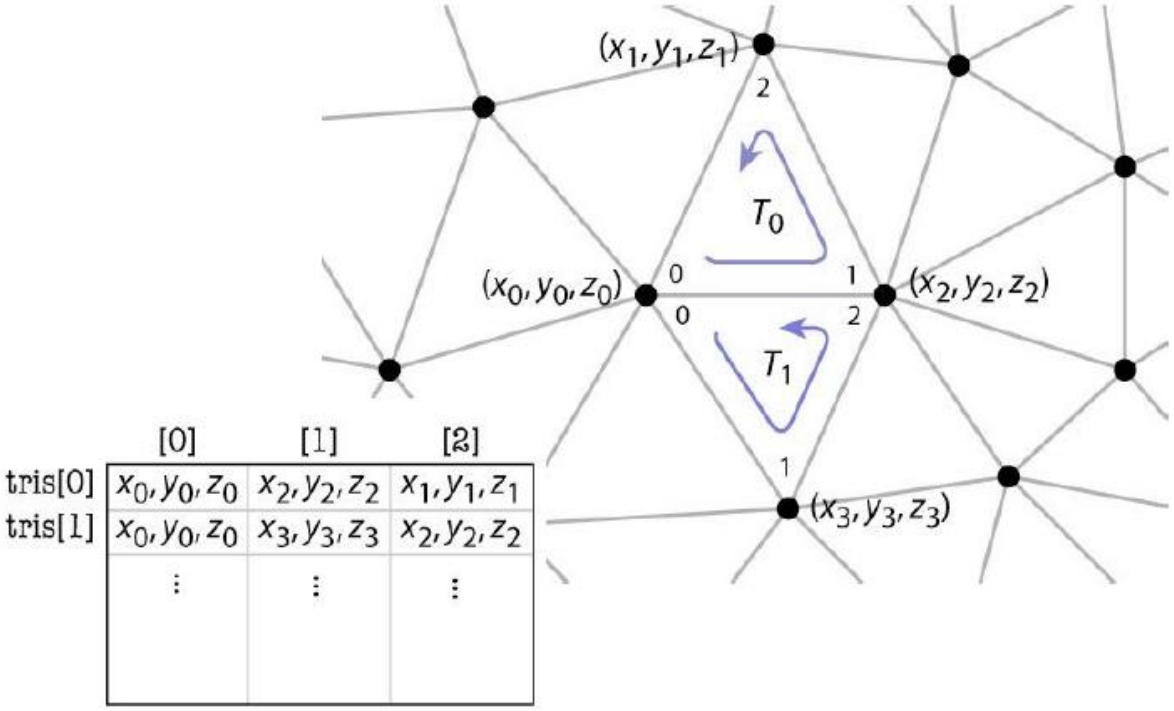
TRIANGLES

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



Basic mesh representation – Separate triangles

- List of triangles

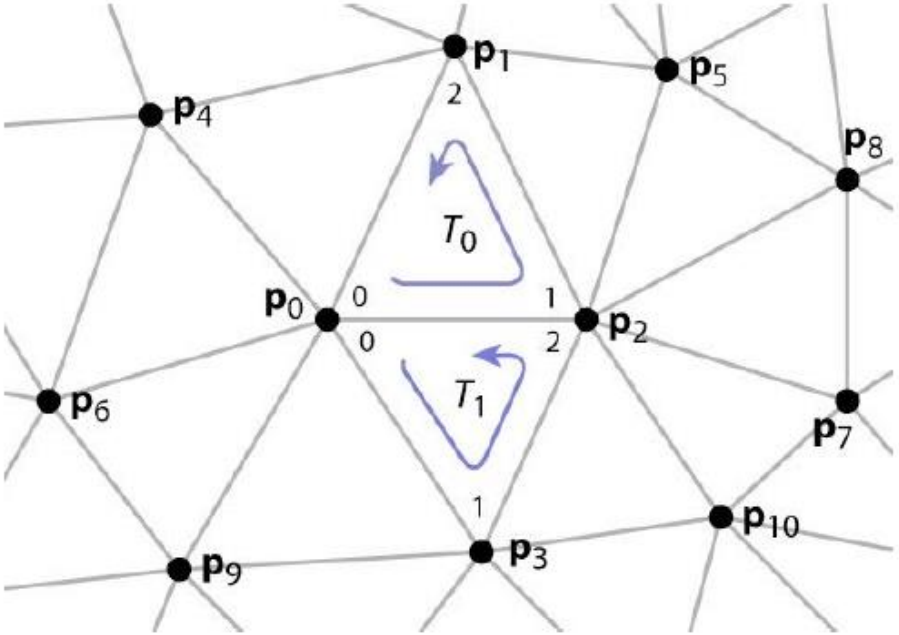


Basic mesh representation – Indexed triangle set

- List of vertices / indexed triangle

verts[0]	x_0, y_0, z_0
verts[1]	x_1, y_1, z_1
	x_2, y_2, z_2
	x_3, y_3, z_3
	⋮

tInd[0]	0, 2, 1
tInd[1]	0, 3, 2
	⋮



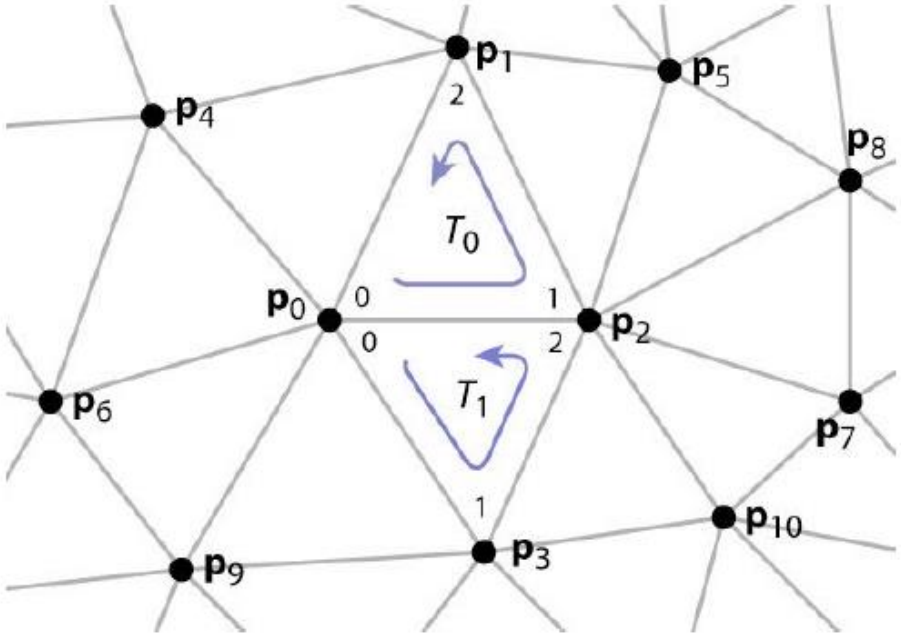
Comparison

- List of triangles
  - GOOD: simple
  - BAD: contains redundant vertex information
- List of vertices + List of indexed triangles
  - GOOD: sharing vertex position information, reduces memory usage
  - GOOD: ensure integrity of the mesh (changing a vertex's position in 3D space causes that vertex in all the polygons to move)
  - BAD: more complex

Other data structures and properties for triangles

- How to be more efficient when regularity, e.g., triangle strips
- How to find triangle neighbors?
- Normal
- Plane
- Simplification
- Multi-resolution – Level Of Details (LODs)
- Remeshing, smoothing ….

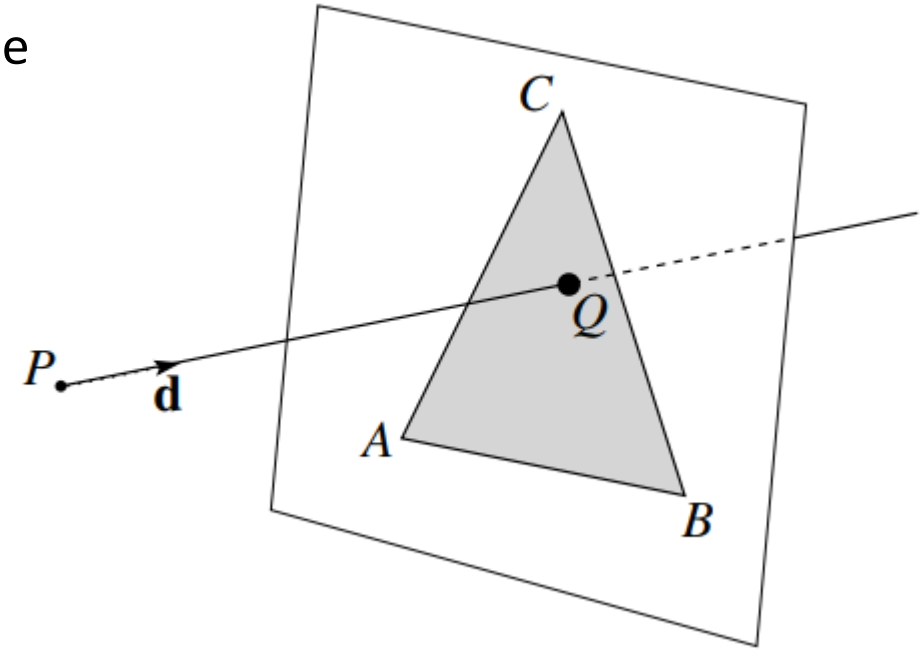
Basic operation: Normal of a triangle



$$n = \frac{(p_2 - p_0) \times (p_1 - p_0)}{\|(p_2 - p_0) \times (p_1 - p_0)\|}$$

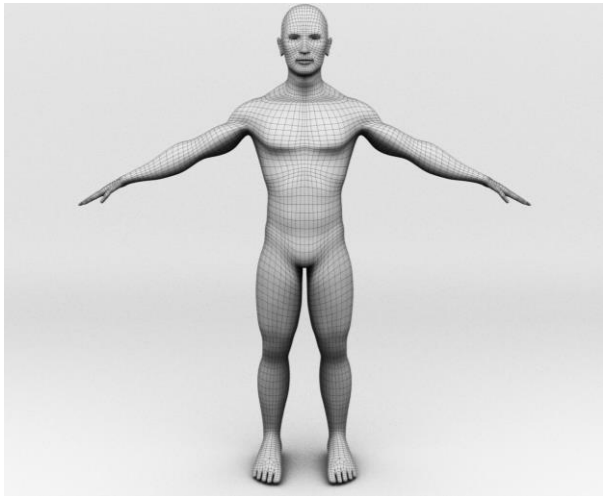
Basic operation: Intersection of line and triangle

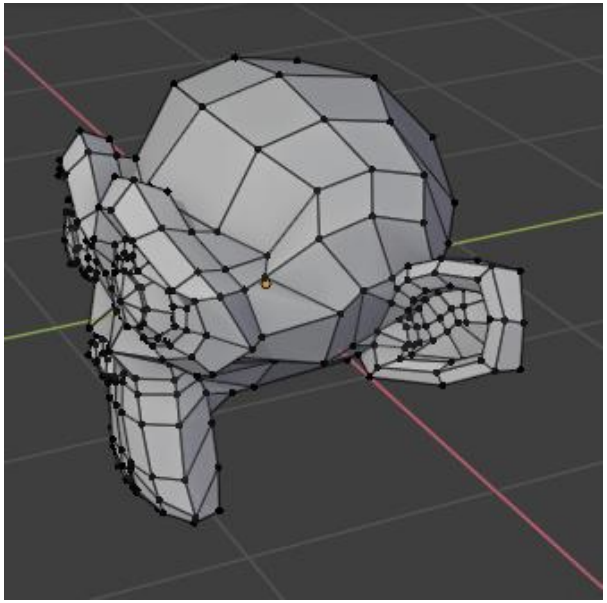
- Calculate the position Q
  - Get the equation of plane ABC
  - Calculate the intersection of place ABC and line
- Check if Q is inside the triangle



Quad mesh

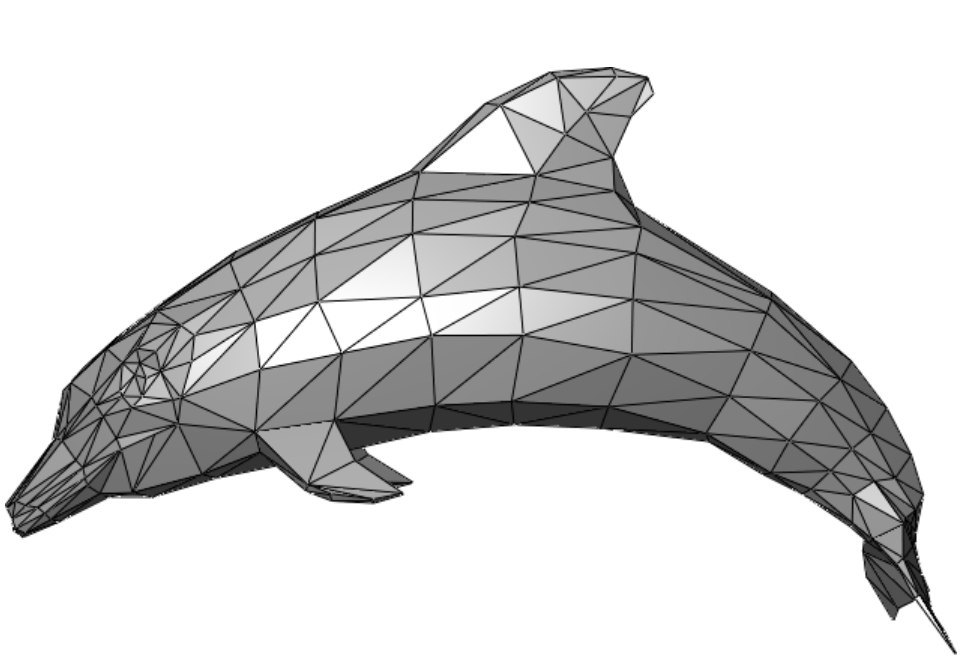
- Nice regularity, easy to store
- Easy to parameterize, good feature! (e.g., texture mapping)
- How to convert a general polygonal mesh to quad mesh is still a research topic

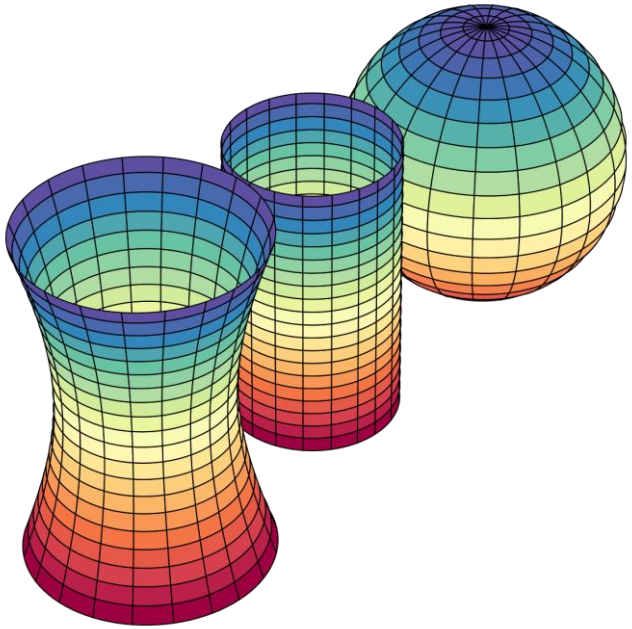




Pros and Cons of polygon surface

- GOOD: Simple, fast in rendering and processing
- BAD: Need much more surfaces to present smooth curved surfaces



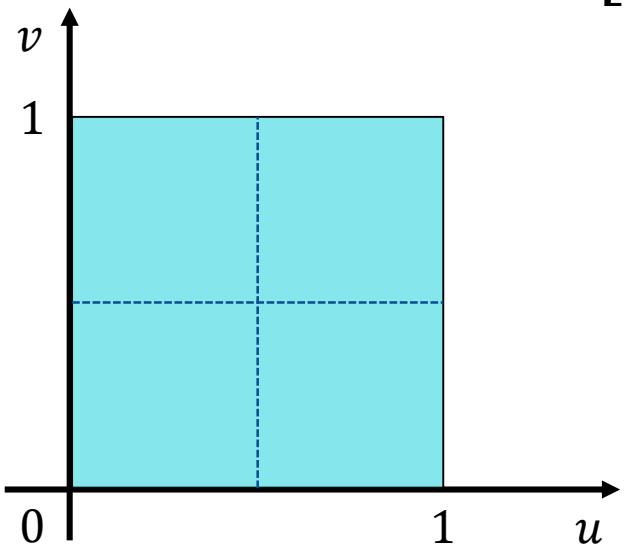


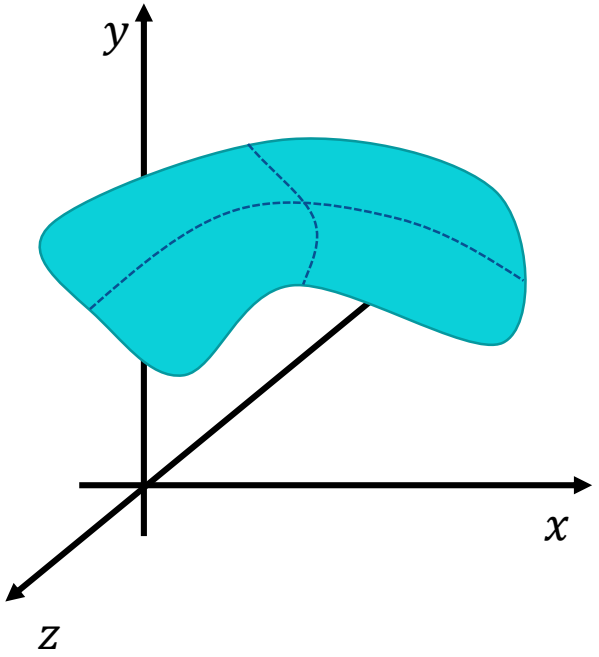
Parametric Surface

Parametric surface

$$f\colon \mathbb{R}^2 \mapsto \mathbb{R}^3$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = f\left(\begin{bmatrix} u \\ v \end{bmatrix}\right)$$





Parametric surface

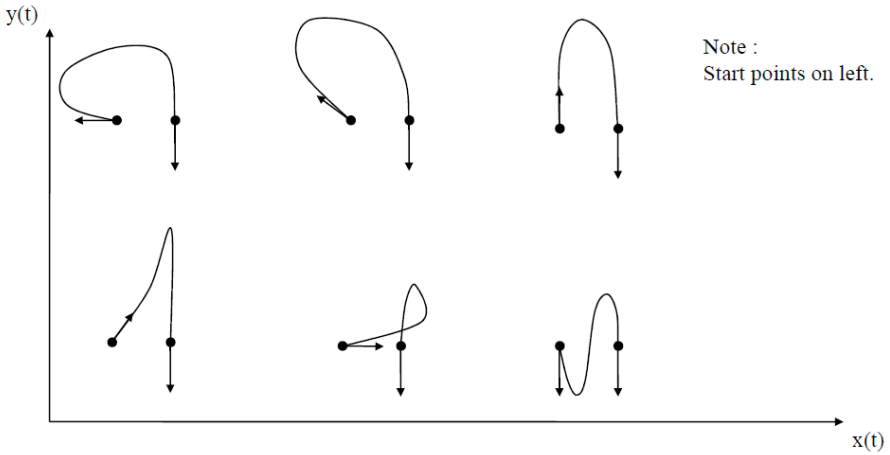
- What representation do we want?
  - Easy to compute
  - Easy to manipulate
  - Predictable
  - Universal
  - Smooth

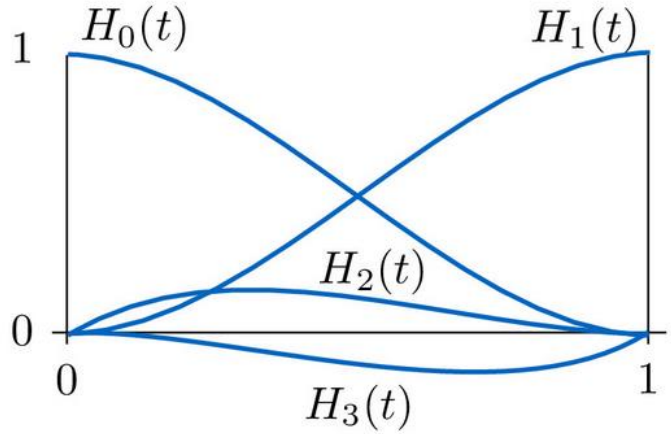
Recall: Hermit curve

Hermite Specification

$$S(t) = at^3 + bt^2 + ct + d$$

$$= [H_0(t) \quad H_1(t) \quad H_2(t) \quad H_3(t)] \begin{bmatrix} y_1 \\ y_2 \\ m_1 \\ m_2 \end{bmatrix}$$

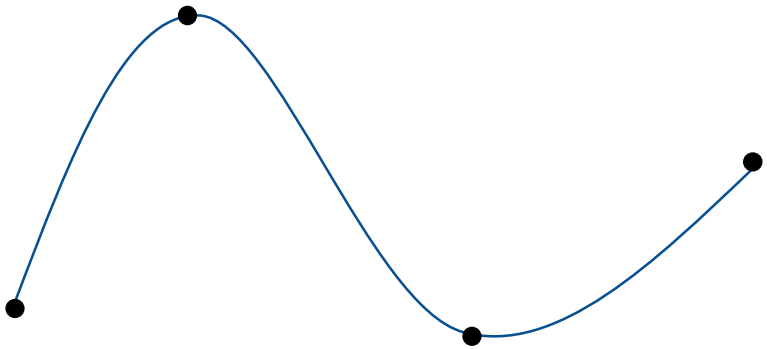




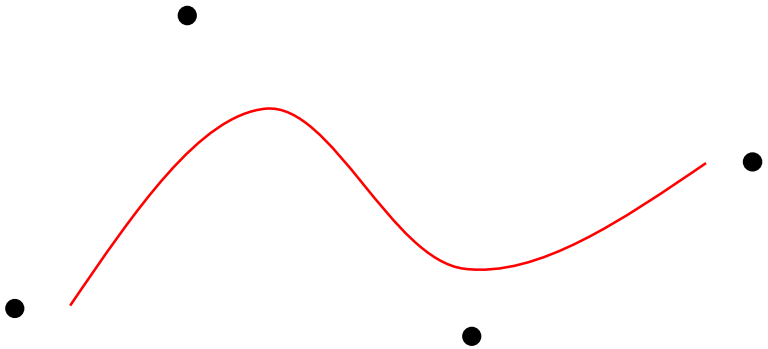
$$H_0(t) = 2t^3 - 3t^2 + 1 \qquad H_1(t) = -2t^3 + 3t^2$$

$$H_2(t) = t^3 - 2t^2 + t \qquad H_3(t) = t^3 - t^2$$

Interpolation v.s. Fitting



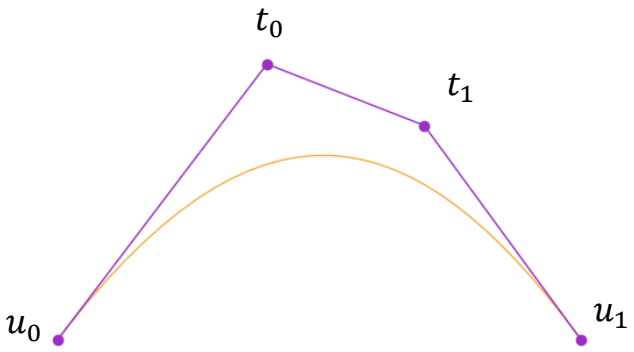
Interpolation



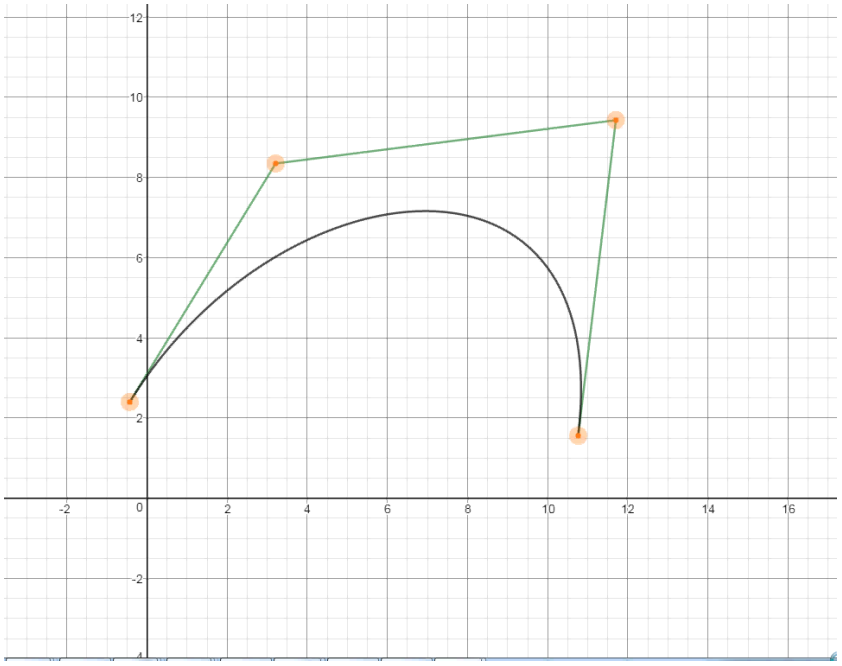
Fitting

Bézier Curve

1962年，法国雷诺汽车公司（Renault Car company）的工程师 Pierre Bézier, 提出的一种新的曲线表示方法



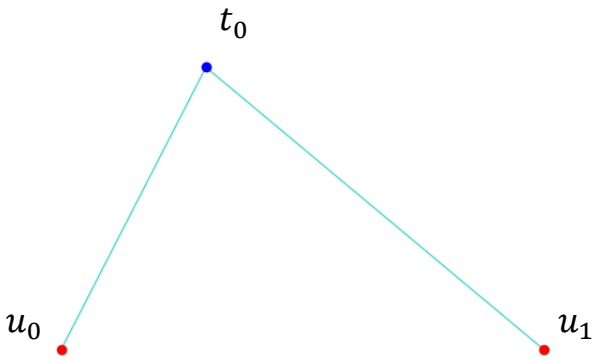
A smooth curve passing u_0 and u_1

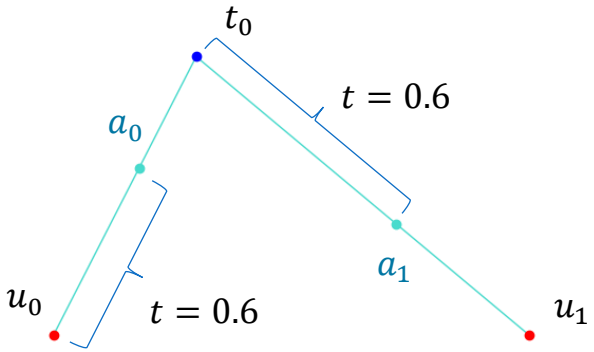


An extremely nice explanation of Bézier curve:
https://www.youtube.com/watch?v=aVwxzDHniEw

https://www.desmos.com/calculator/cahqdxeshd

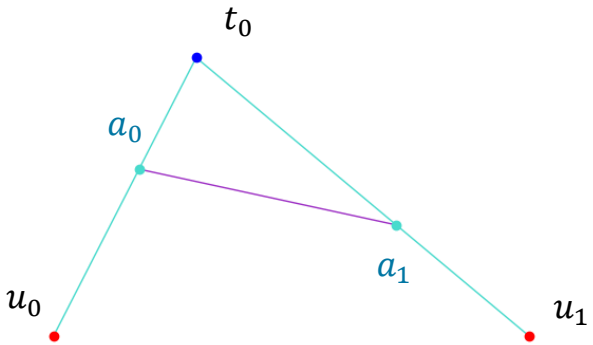
How to Get a Bézier Curve?

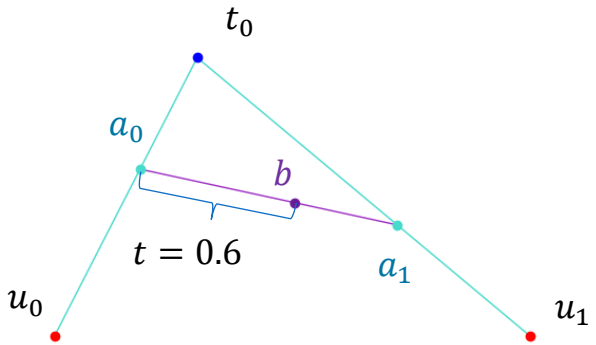




- Assume we have only one additional control point t_0
- We take two immediate points using *lerp*
- $a_0 = lerp(u_0, t_0, t)$ $a_1 = lerp(t_0, u_1, t)$

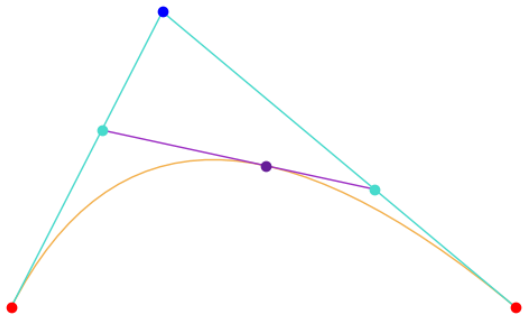
How to Get a Bézier Curve?

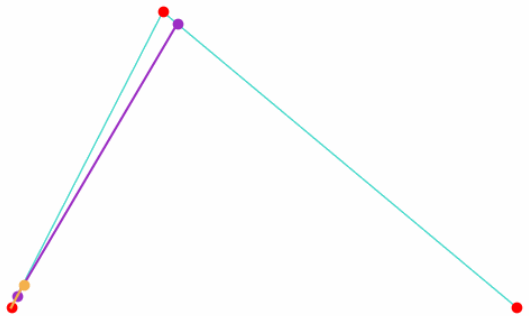




- $a_0 = lerp(u_0, t_0, t)$ $a_1 = lerp(t_0, u_1, t)$
- Connect a_0 and a_1, take a final lerp
- $b = lerp(a_0, a_1, t)$

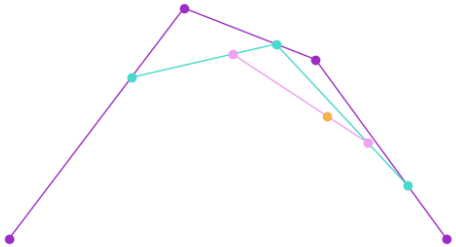
How to Get a Bézier Curve?

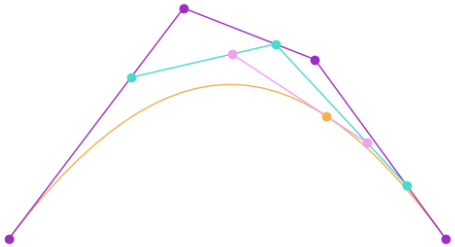




- Move t from 0 to 1, will get a smooth curve
- We call it quadratic Bézier curve

Higher Order Bézier Curve

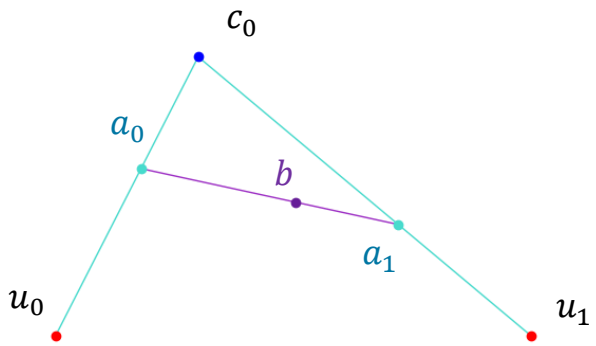




- Just more lines, and more lerps……
- A cubic Bézier has two control points, needs 3+2+1 lerps

Some Math in Bézier Curve

- The above method is called De Casteljau's algorithm (德卡斯特里奥)
- It's more efficient and numerically stable comparing with analytical expression

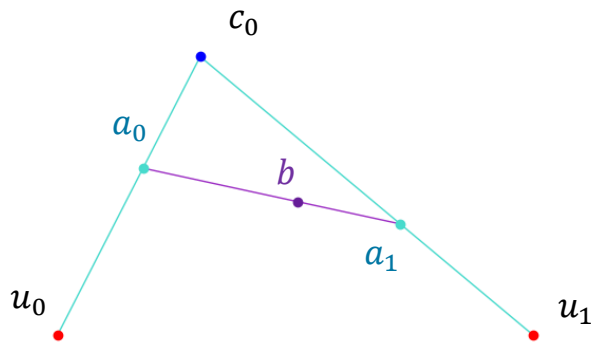


$$a_0 = (1-t)u_0 + tc_0$$

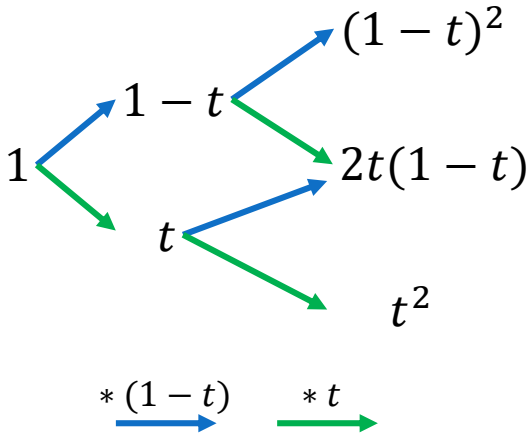
$$a_1 = (1-t)c_0 + tu_1$$

$$\begin{aligned} b &= (1-t)a_0 + ta_1 \\ &= (1-t)^2 u_0 + 2t(1-t)c_0 + t^2 u_1 \end{aligned}$$

Some Math in Bézier Curve



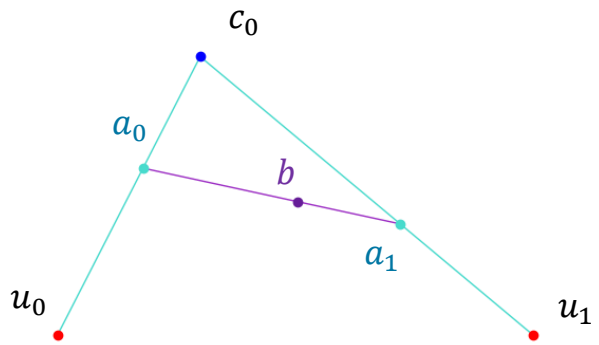
$$b = (1-t)a_0 + ta_1$$
$$= (1-t)^2 u_0 + 2t(1-t)c_0 + t^2 u_1$$



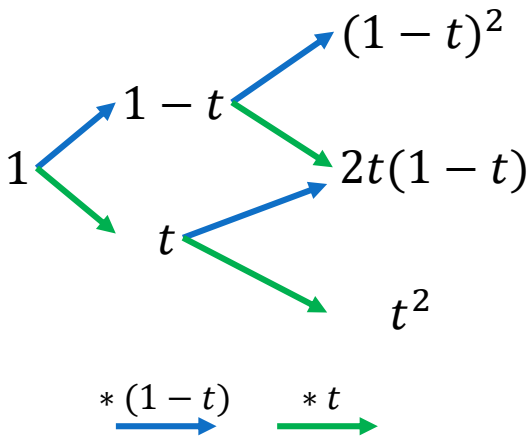
Exactly Pascal's triangle! (杨辉三角)

Also Bernstein polynomials

Some Math in Bézier Curve



$$b = (1-t)a_0 + ta_1$$
$$= (1-t)^2 u_0 + 2t(1-t)c_0 + t^2 u_1$$



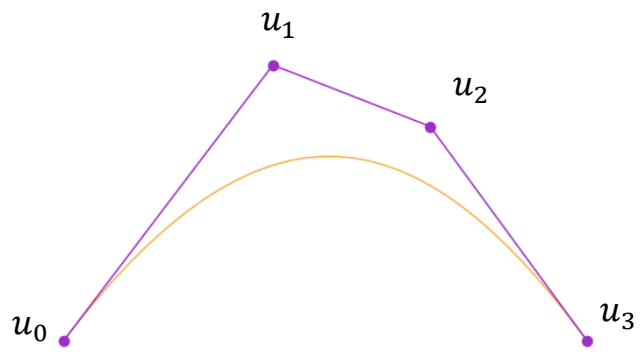
Exactly Pascal's triangle! (杨辉三角)
Also Bernstein polynomials

We can get analytical expression:

$$\boldsymbol{b}(t) = \sum_{k}^{n} \binom{n}{k} t^k (1-t)^{n-k} u_k$$

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

Some Math in Bézier Curve



Bernstein polynomials

$$\boldsymbol{b}(t) = \sum_{k}^{n} \binom{n}{k} t^k (1-t)^{n-k} u_k$$

$$= \sum_{k}^{n} B_{k,n}(t) u_k$$

$$\boldsymbol{b^T}(t) = \boldsymbol{t}^T M_B U = B_n(t)^T U$$

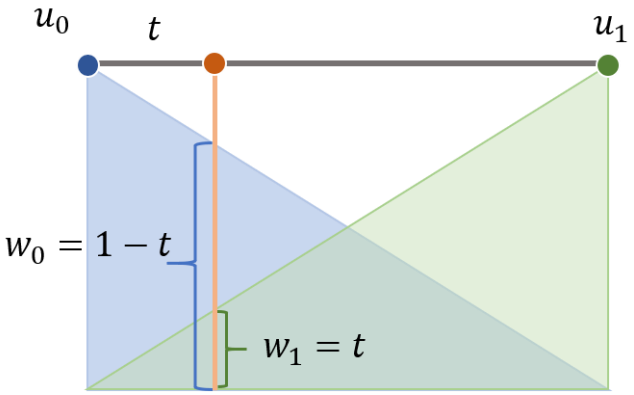
basis functions

$$\boldsymbol{t} = \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix} \qquad M_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} \boldsymbol{u}_0^T \\ \boldsymbol{u}_1^T \\ \boldsymbol{u}_2^T \\ \boldsymbol{u}_3^T \end{bmatrix} = \begin{bmatrix} u_{0,1} & u_{0,2} & u_{0,3} \\ u_{1,1} & u_{1,2} & u_{1,3} \\ u_{2,1} & u_{2,2} & u_{2,3} \\ u_{3,1} & u_{3,2} & u_{3,3} \end{bmatrix}$$

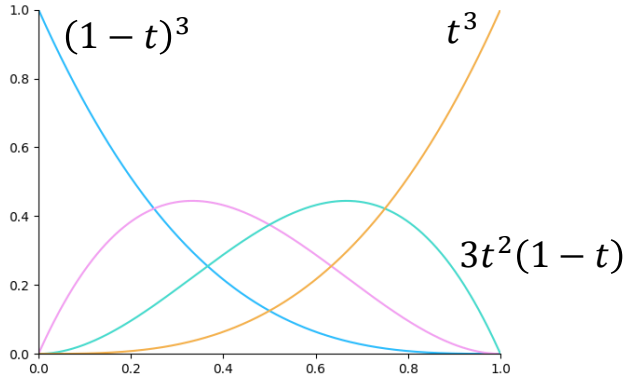
Some Math in Bézier Curve

Bernstein polynomials



Weight of lerp

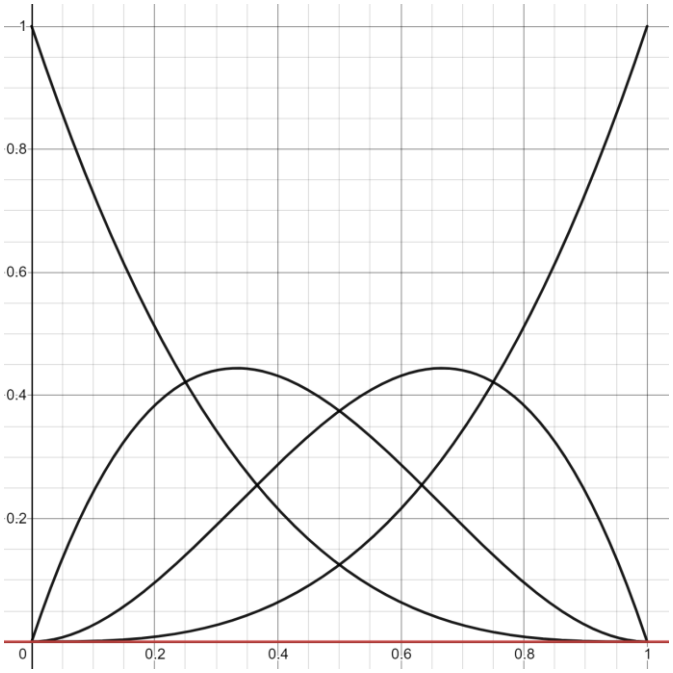
$$B_3(t) = \begin{bmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 3t^2(1-t) \\ t^3 \end{bmatrix}$$



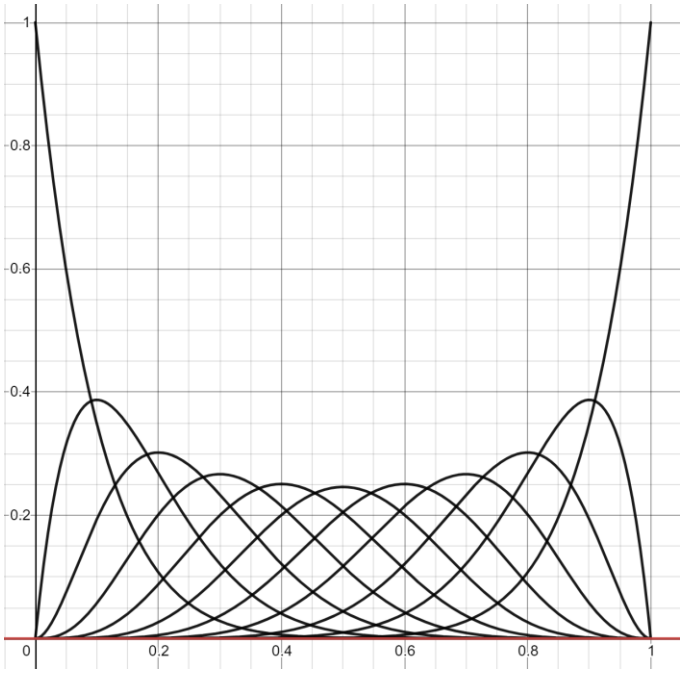
Weight of Cubic Bézier

- Both lerp and Bézier are weighted sum of control points
- Only the weight functions are different

More about Bernstein polynomials

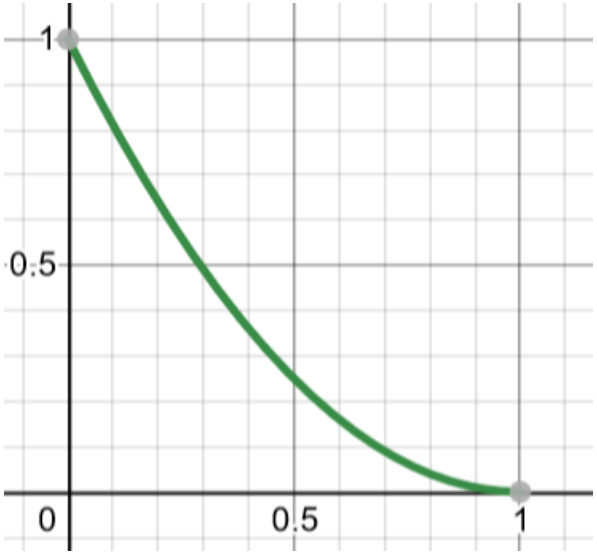


$n = 3$



$n = 10$

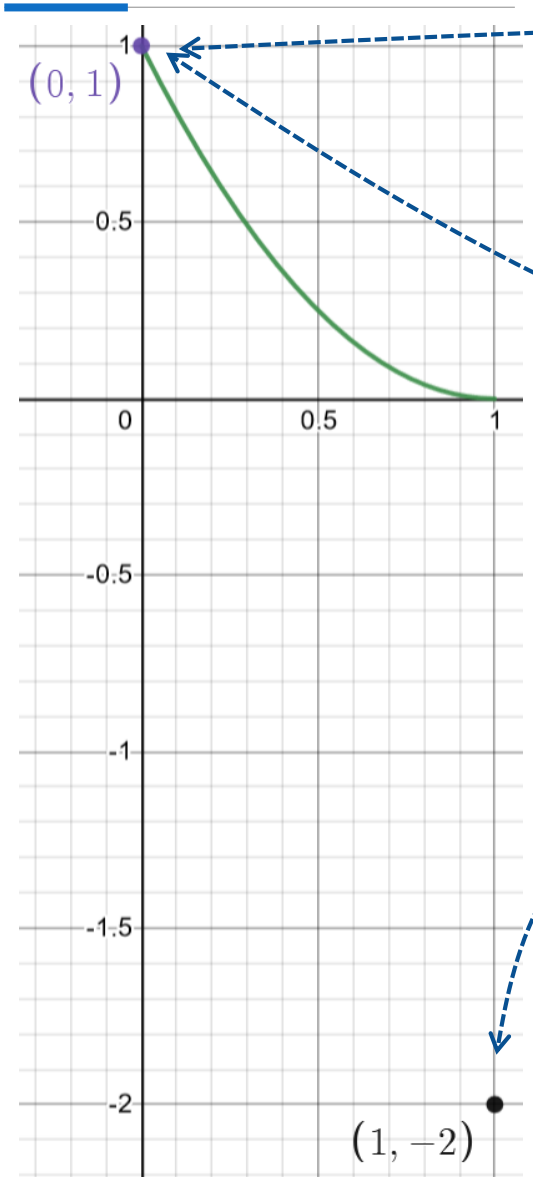
More about Bernstein polynomials



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

$$f(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} t^2 + \begin{bmatrix} 1 \\ -2 \end{bmatrix} t + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

More about Bernstein polynomials

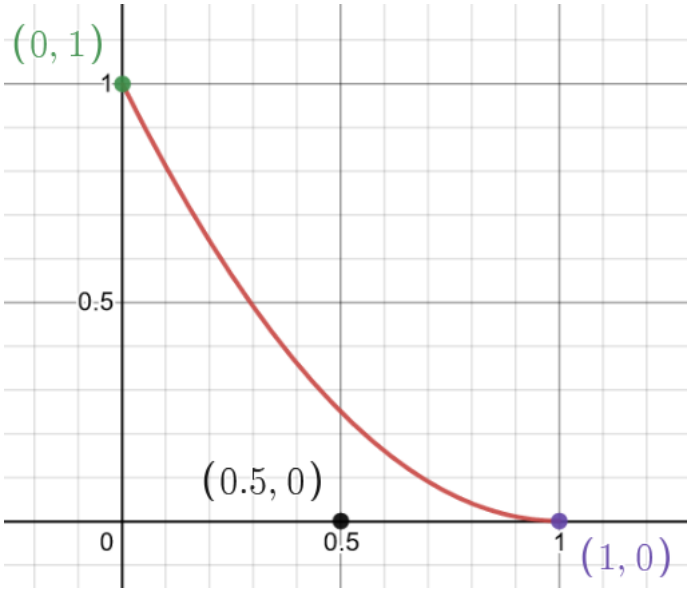


$$y = x^2 - 2x + 1$$

$$f(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix} t^2 + \begin{bmatrix} \mathbf{1} \\ \mathbf{-2} \end{bmatrix} t + \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}$$

Control points are hard to interpret

More about Bernstein polynomials



$$B_2(t) = \begin{bmatrix} (1-t)^2 \\ 2t(1-t) \\ t^2 \end{bmatrix}$$

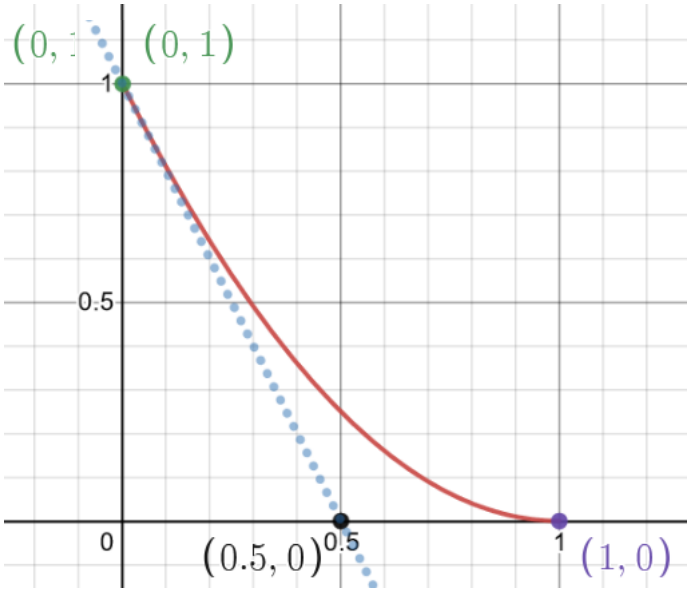
$$y = x^2 - 2x + 1$$

Convert to Bernstein polynomials

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

$$= \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix} (1-t)^2 + \begin{bmatrix} \mathbf{0.5} \\ \mathbf{0} \end{bmatrix} 2t(1-t) + \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \end{bmatrix} t^2$$

More about Bernstein polynomials



$$B_2(t) = \begin{bmatrix} (1-t)^2 \\ 2t(1-t) \\ t^2 \end{bmatrix}$$

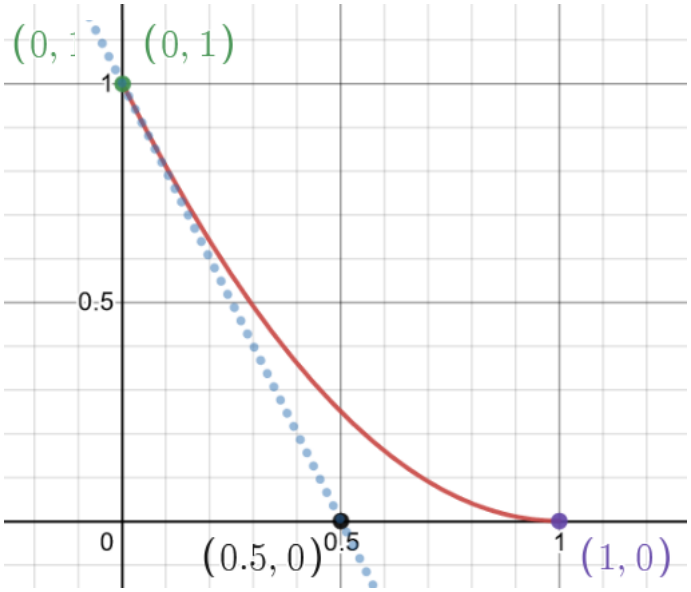
$$y = x^2 - 2x + 1$$

Convert to Bernstein polynomials

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

$$= \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix} (1-t)^2 + \begin{bmatrix} \mathbf{0.5} \\ \mathbf{0} \end{bmatrix} 2t(1-t) + \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \end{bmatrix} t^2$$

More about Bernstein polynomials



$$B_2(t) = \begin{bmatrix} (1-t)^2 \\ 2t(1-t) \\ t^2 \end{bmatrix}$$

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

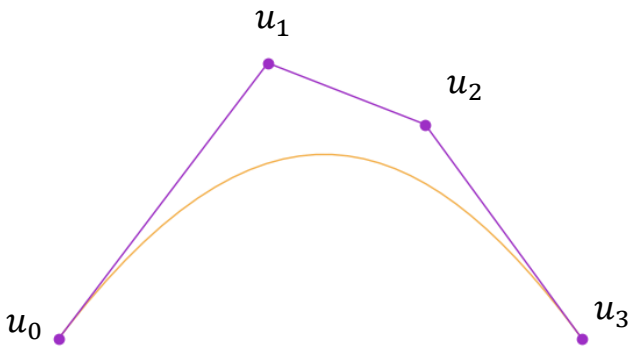
Convert to Bernstein polynomials

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

$$= \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix} (1-t)^2 + \begin{bmatrix} \mathbf{0.5} \\ \mathbf{0} \end{bmatrix} 2t(1-t) + \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \end{bmatrix} t^2$$

Control points are **meaningful**

Some Math in Bézier Curve



Tangential direction at u_0 is always $u_0 \rightarrow u_1$?

$$\boldsymbol{b}(t) = \sum_{k}^{n} \binom{n}{k} t^k (1-t)^{n-k} \, u_k$$

$$\boldsymbol{b}'(0) = n(u_1 - u_0)$$

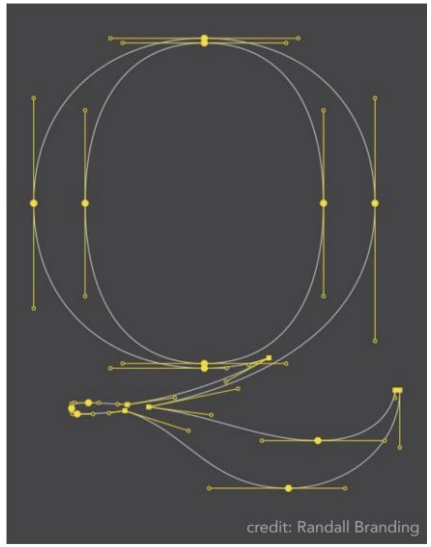
$$\boldsymbol{b}'(1) = n(u_n - u_{n-1})$$

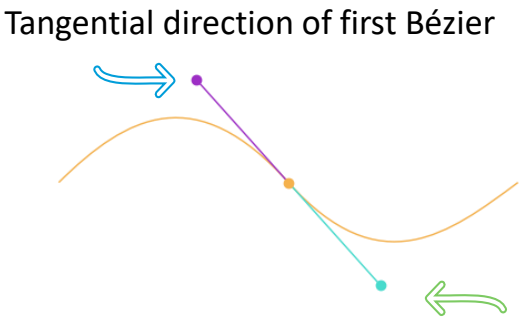
$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

Some Math in Bézier Curve

We can *concatenate* two Bézier curves *smoothly*

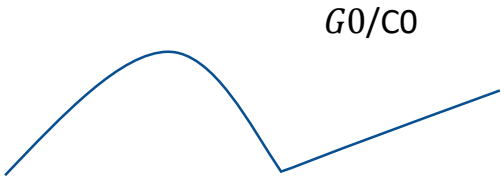
Just make a pair of central symmetry control points!





Geometry continuity

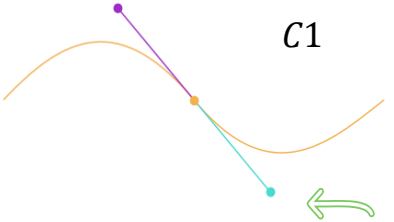
G0/C0 continuity: Same value



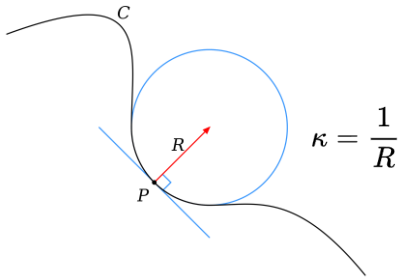
G1 continuity: Same tangential direction

C1 continuity: Continuous wrt parameter t

Same tangential direction and norm



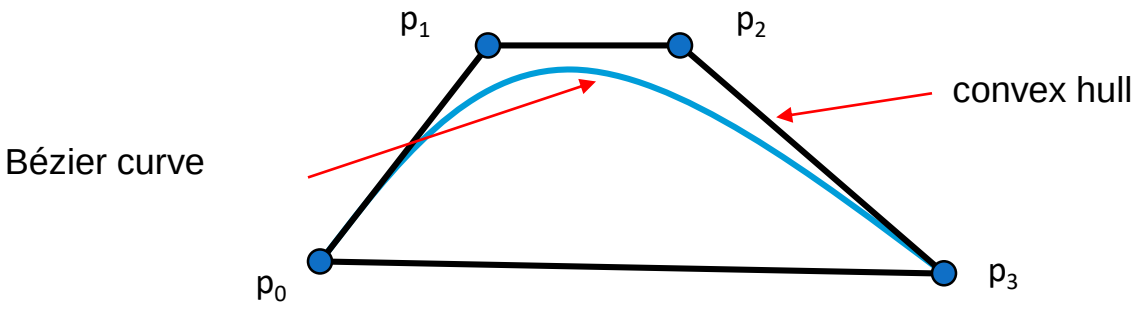
G2 continuity: G1 continuity + same curvature



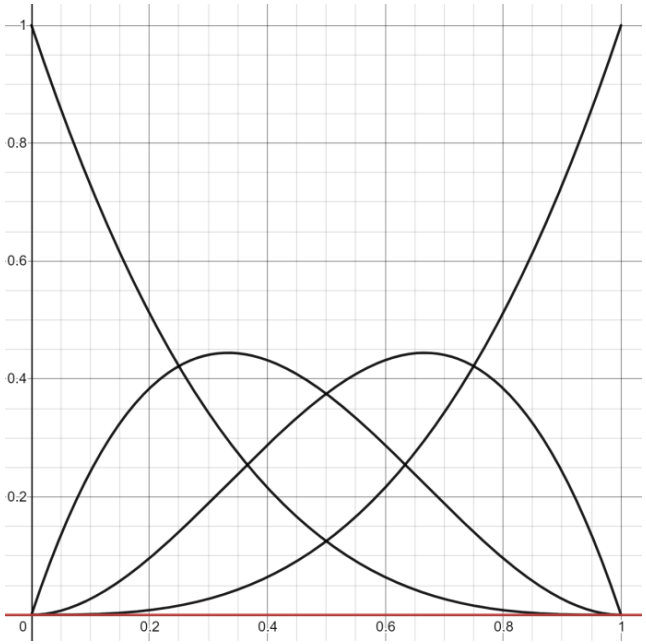
Analysis

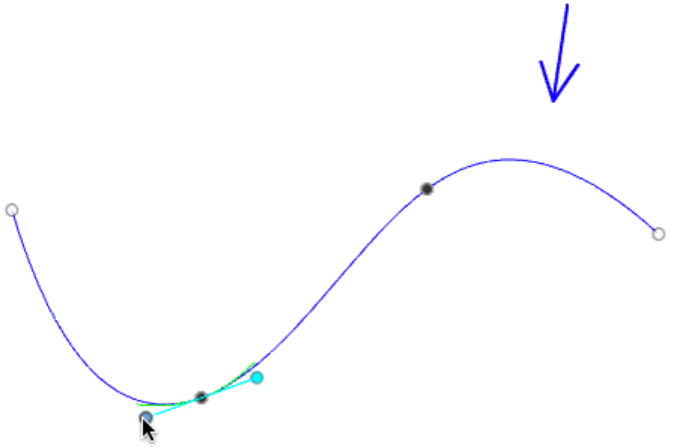
- The properties of the Bernstein polynomials ensure that all Bézier curves lie in the convex hull of their control points (although not the tightest bound!)

- Hence, even though we do not interpolate all the data, we cannot be too far away



Analysis

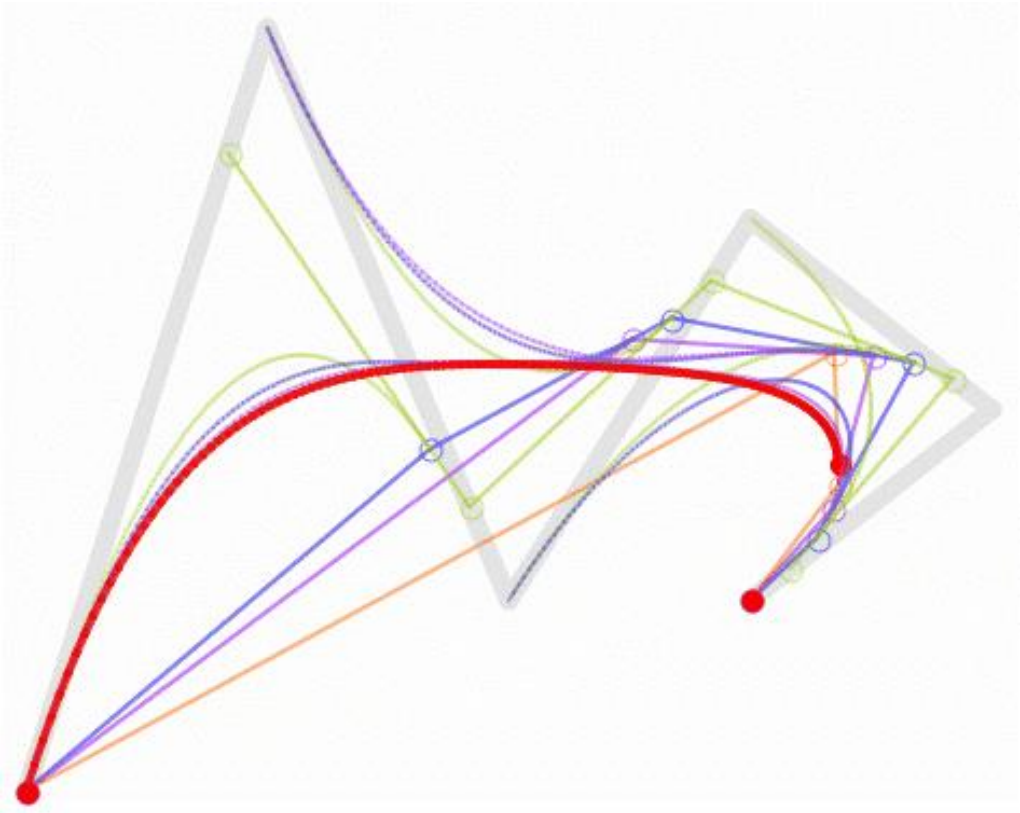




From autodesk

- Every control point effects the entire curve
- Add control points means to increase the order of polynomial
  - higher order means more sensitive

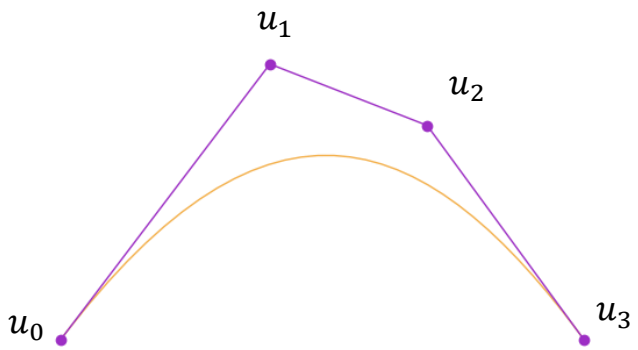
Higher-order Bézier Curve



…can be hard to control…

Demo: https://www.desmos.com/calculator/twyfn7nrac

1D - Bézier Curve



Bernstein polynomials

$$\boldsymbol{b}(t) = \sum_{k}^{n} \binom{n}{k} t^k (1-t)^{n-k}\, u_k$$

$$= \sum_{k}^{n} B_{k,n}(t) u_k$$

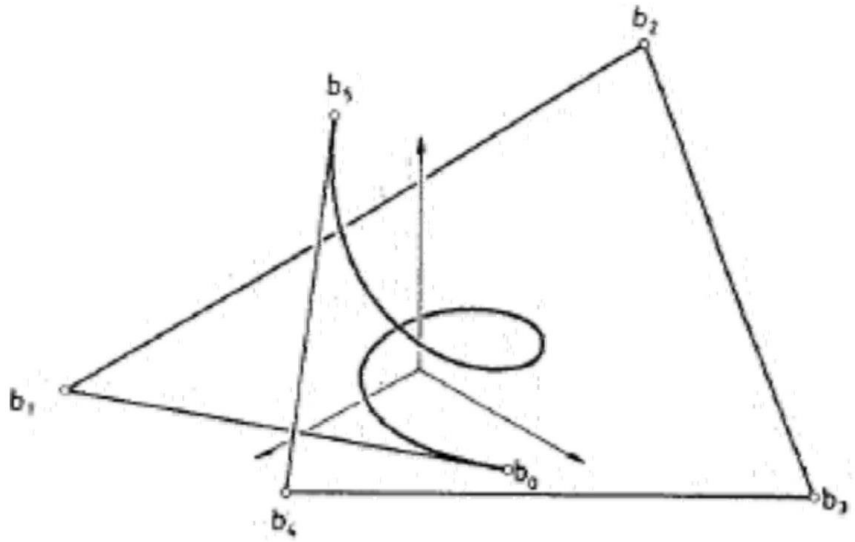
$$\boldsymbol{b^T}(t) = \boldsymbol{t}^T M_B U = B_n(t)^T U$$

basis functions

$$\boldsymbol{t} = \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix} \qquad M_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} \boldsymbol{u}_0^T \\ \boldsymbol{u}_1^T \\ \boldsymbol{u}_2^T \\ \boldsymbol{u}_3^T \end{bmatrix} = \begin{bmatrix} u_{0,1} & u_{0,2} & u_{0,3} \\ u_{1,1} & u_{1,2} & u_{1,3} \\ u_{2,1} & u_{2,2} & u_{2,3} \\ u_{3,1} & u_{3,2} & u_{3,3} \end{bmatrix}$$

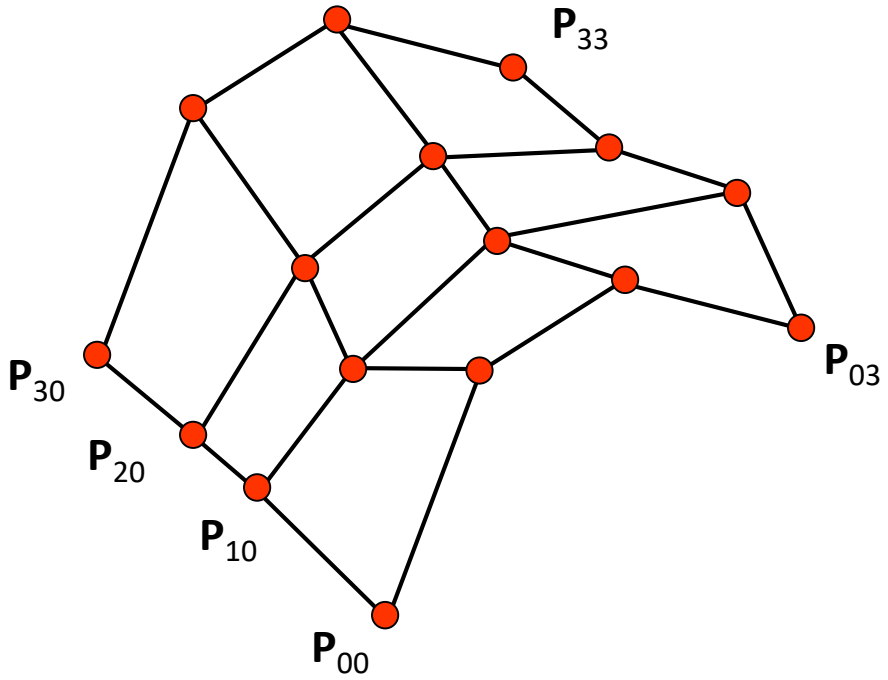
1D - Bézier Curve in 3D space



Spline surface

Control points: 4×4 matrix of points $\mathbf{P}_{ij}$

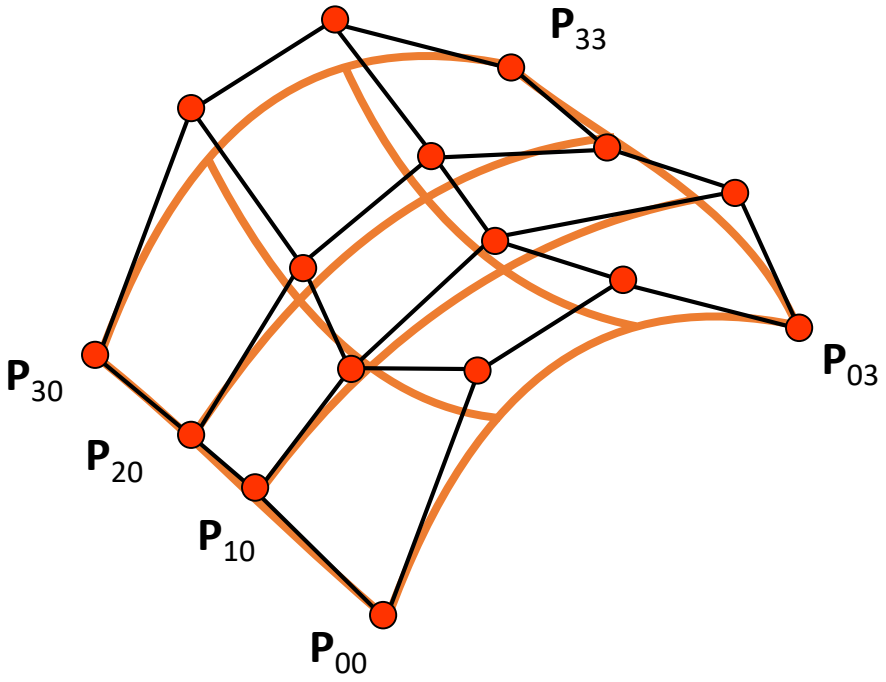
Define a smooth patch.



Spline surface

Control points: 4×4 matrix of points $\mathbf{P}_{ij}$

Define a smooth patch.

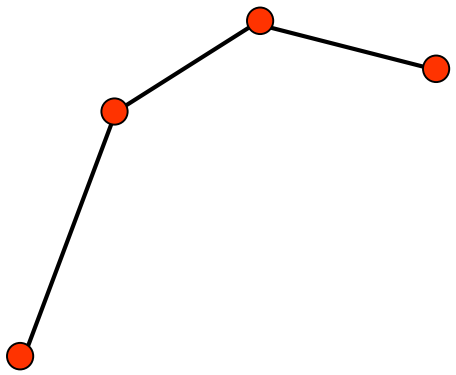


Spline surface

Control points: 4×4 matrix of points $\mathbf{P}_{ij}$

Define four curves.

Surface: Smooth these four curves.

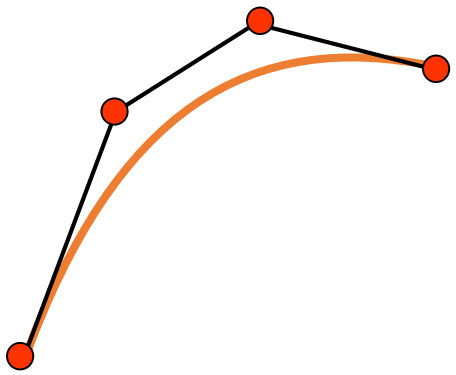


Spline surface

Control points: 4×4 matrix of points $\mathbf{P}_{ij}$

Define four curves.

Surface: Smooth these four curves.

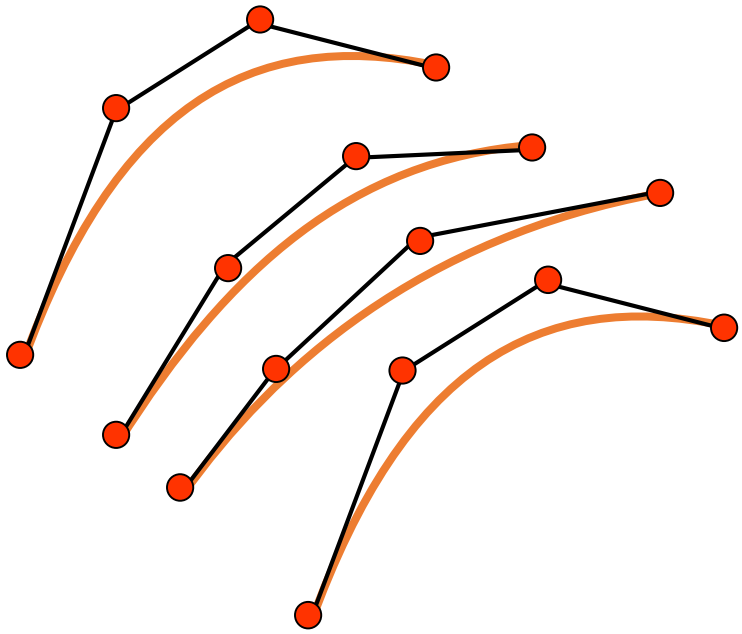


Spline surface

Control points: 4×4 matrix of points $\mathbf{P}_{ij}$

Define four curves.

Surface: Smooth these four curves.

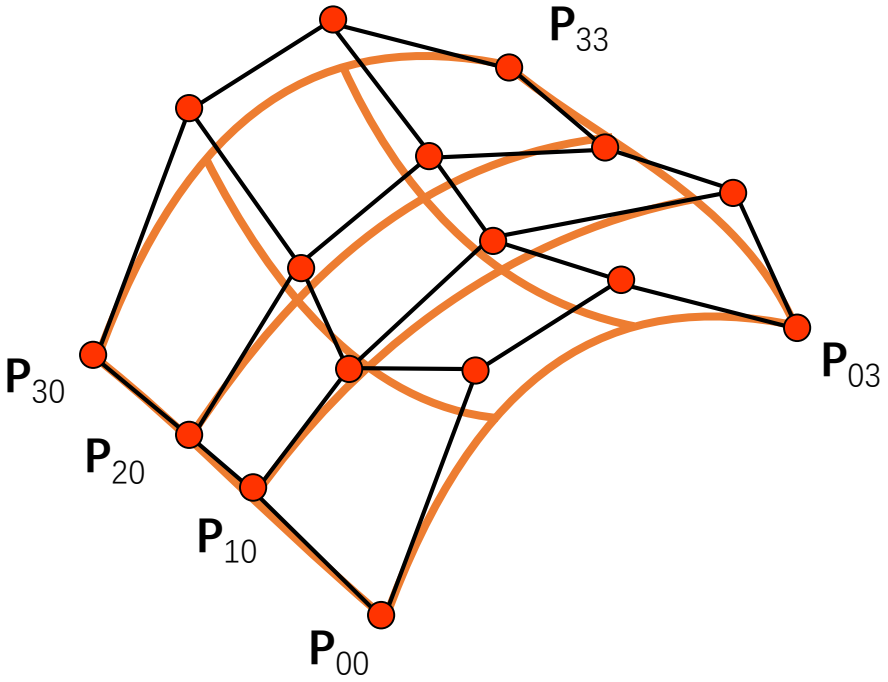


Spline surface

Control points: 4×4 matrix of points $\mathbf{P}_{ij}$

Define four curves.

Surface: Smooth these four curves.



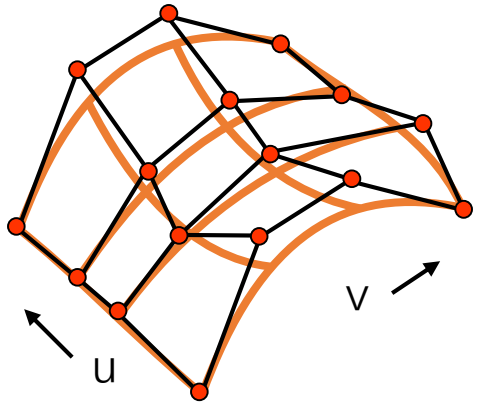
Bézier surface

Surface:
Control points: 4×4 matrix of points $\mathbf{P}_{i,j}$

Curve: $\mathbf{P}(u) = \sum_{k=0}^{3} B_k(u)\mathbf{P}_k$

Surface: $\mathbf{P}(u,v) = \sum_{i=0}^{3} B_i(u)\left(\sum_{j=0}^{3} B_j(v)\mathbf{P}_{ij}\right)$

$$= \sum_{i=0}^{3}\sum_{j=0}^{3} B_i(u)\,B_j(v)\mathbf{P}_{ij}$$



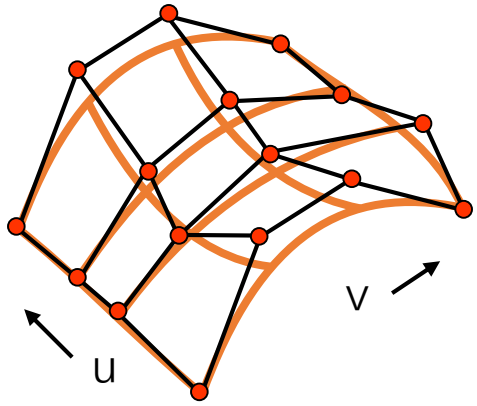
Bézier surface

Surface:
Control points: 4×4 matrix of points $\mathbf{P}_{i,j}$

$$\text{Curve: } \mathbf{P}(u) = \sum_{k=0}^{3} B_k(u)\mathbf{P}_k$$

$$\text{Surface: } \mathbf{P}(u,v) = \sum_{i=0}^{3} B_i(u)\left(\sum_{j=0}^{3} B_j(v)\mathbf{P}_{ij}\right)$$

$$= \sum_{i=0}^{3}\sum_{j=0}^{3} B_i(u)\,B_j(v)\mathbf{P}_{ij}$$



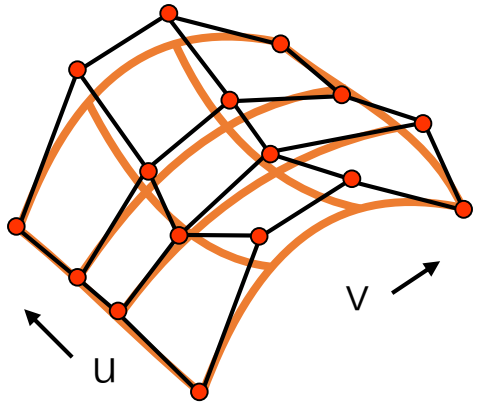
Bézier surface

Surface:
Control points: 4×4 matrix of points $\mathbf{P}_{i,j}$

Curve: $\mathbf{P}(u) = \sum_{k=0}^{3} B_k(u)\mathbf{P}_k$

Surface: $\mathbf{P}(u,v) = \sum_{i=0}^{3} B_i(u)\left(\sum_{j=0}^{3} B_j(v)\mathbf{P}_{ij}\right)$

$= \sum_{i=0}^{3}\sum_{j=0}^{3} B_i(u)\,B_j(v)\mathbf{P}_{ij}$



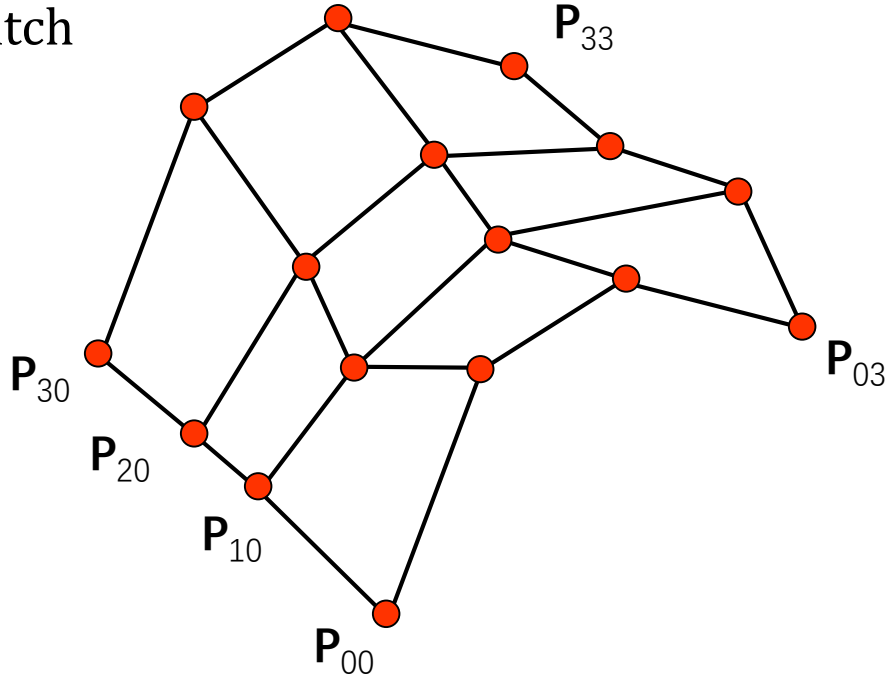
Tensor product surface

Bézier surface

Generic: $\mathbf{P}(u,v) = \sum_{j=0}^{m}\sum_{k=0}^{n} B_j(v)B_k(u)\mathbf{P}_{j,k}$

with $B_{k,n}(u) = \frac{n!}{k!\,(n-k)!}u^k(1-u)^{n-k}$

Often $m = n = 4$: ⇄ bicubic patch

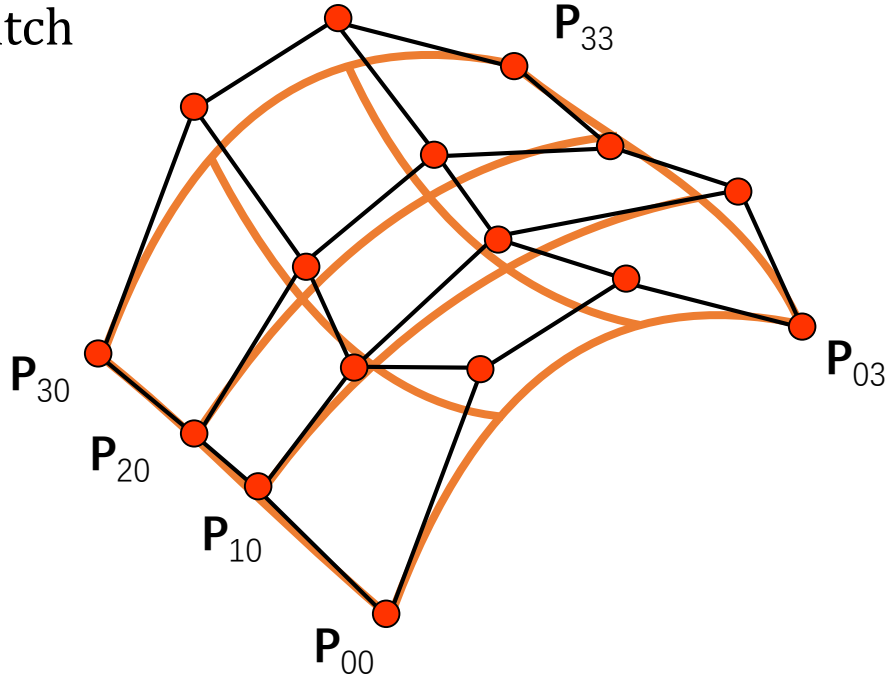


Bézier surface

Generic: $\mathbf{P}(u,v) = \sum_{j=0}^{m}\sum_{k=0}^{n} B_j(v)B_k(u)\mathbf{P}_{j,k}$

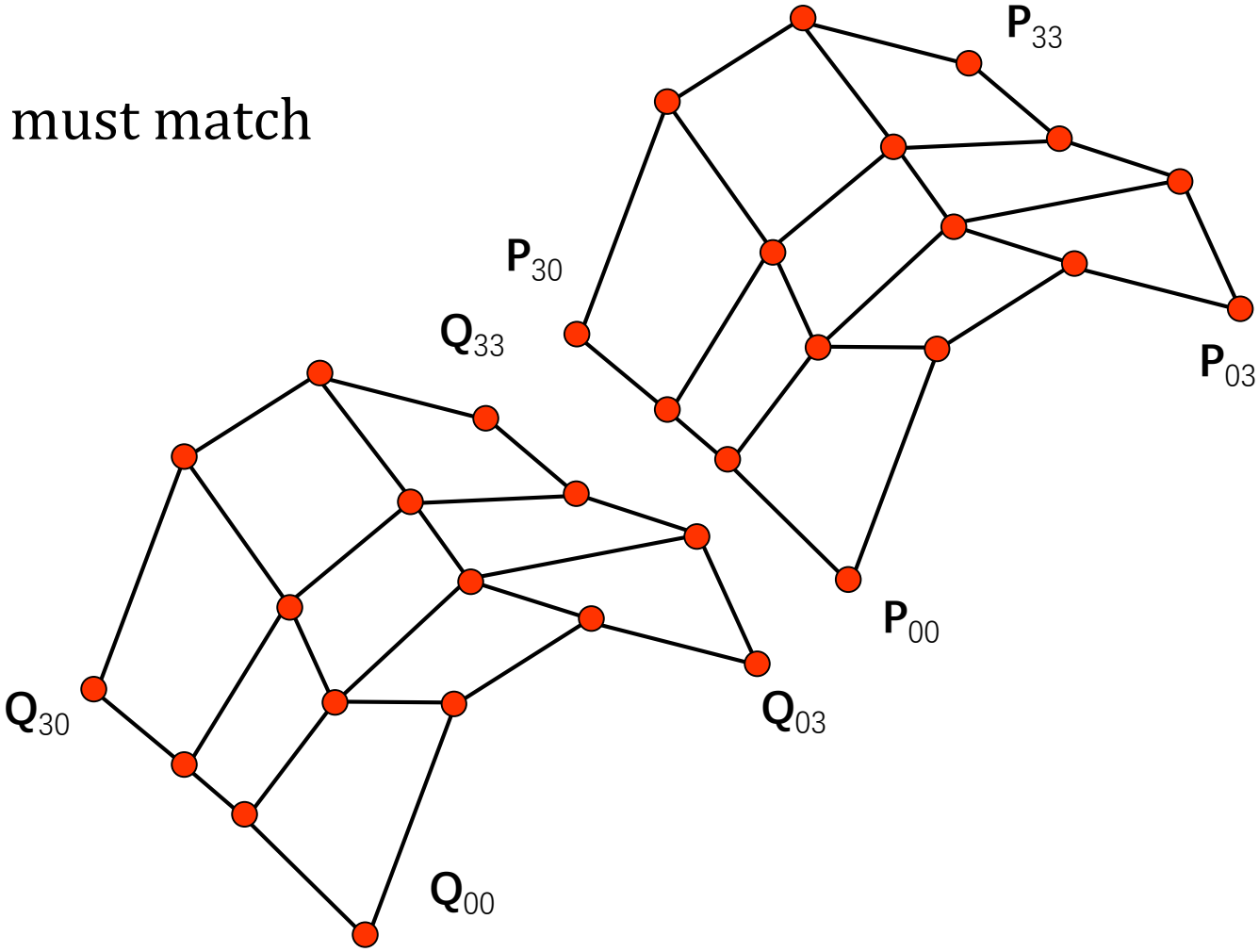
with $B_{k,n}(u) = \frac{n!}{k!\,(n-k)!}u^k(1-u)^{n-k}$

Often $m = n = 4$: ⇄ bicubic patch



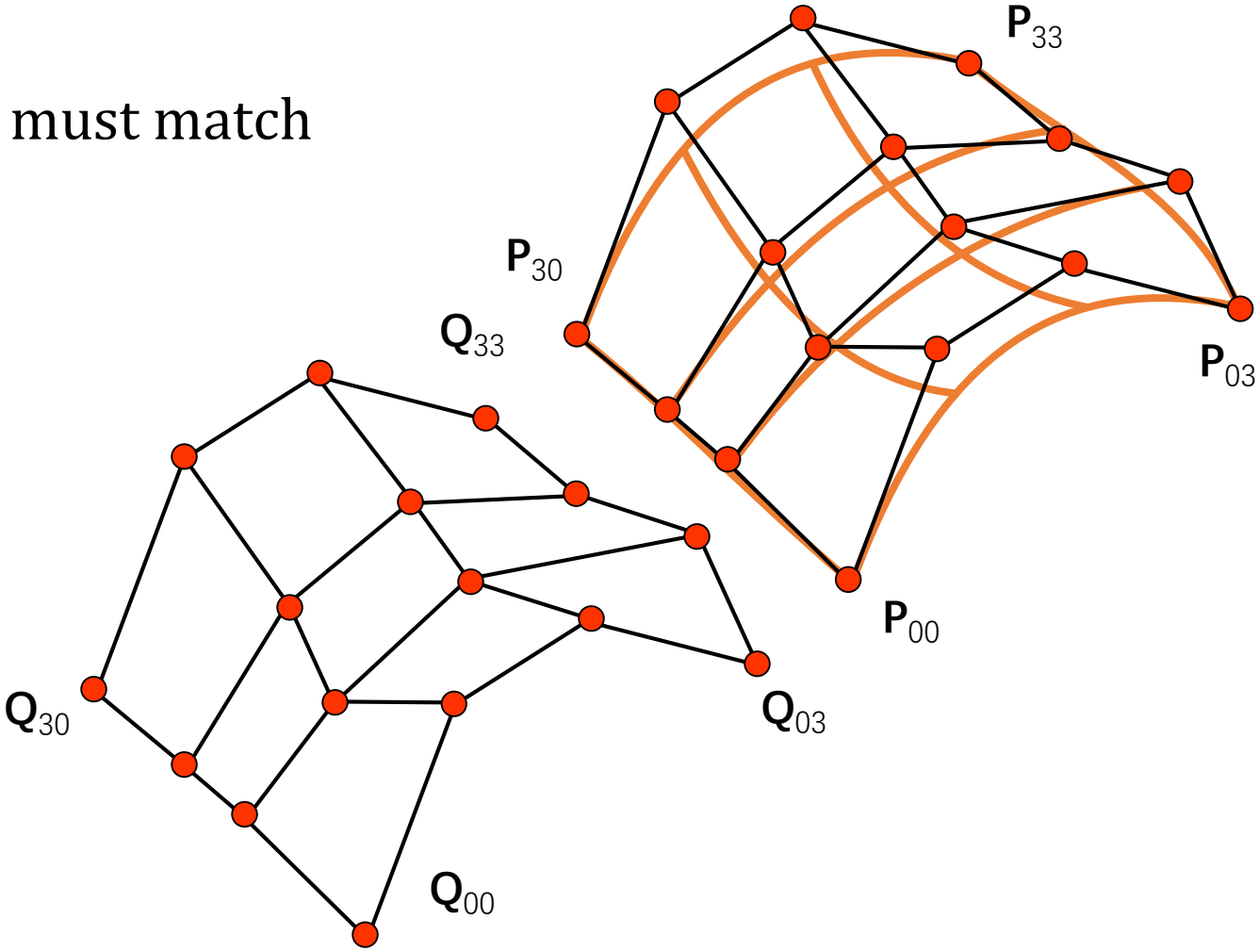
Bézier surface

Joining:
G^0: points must match



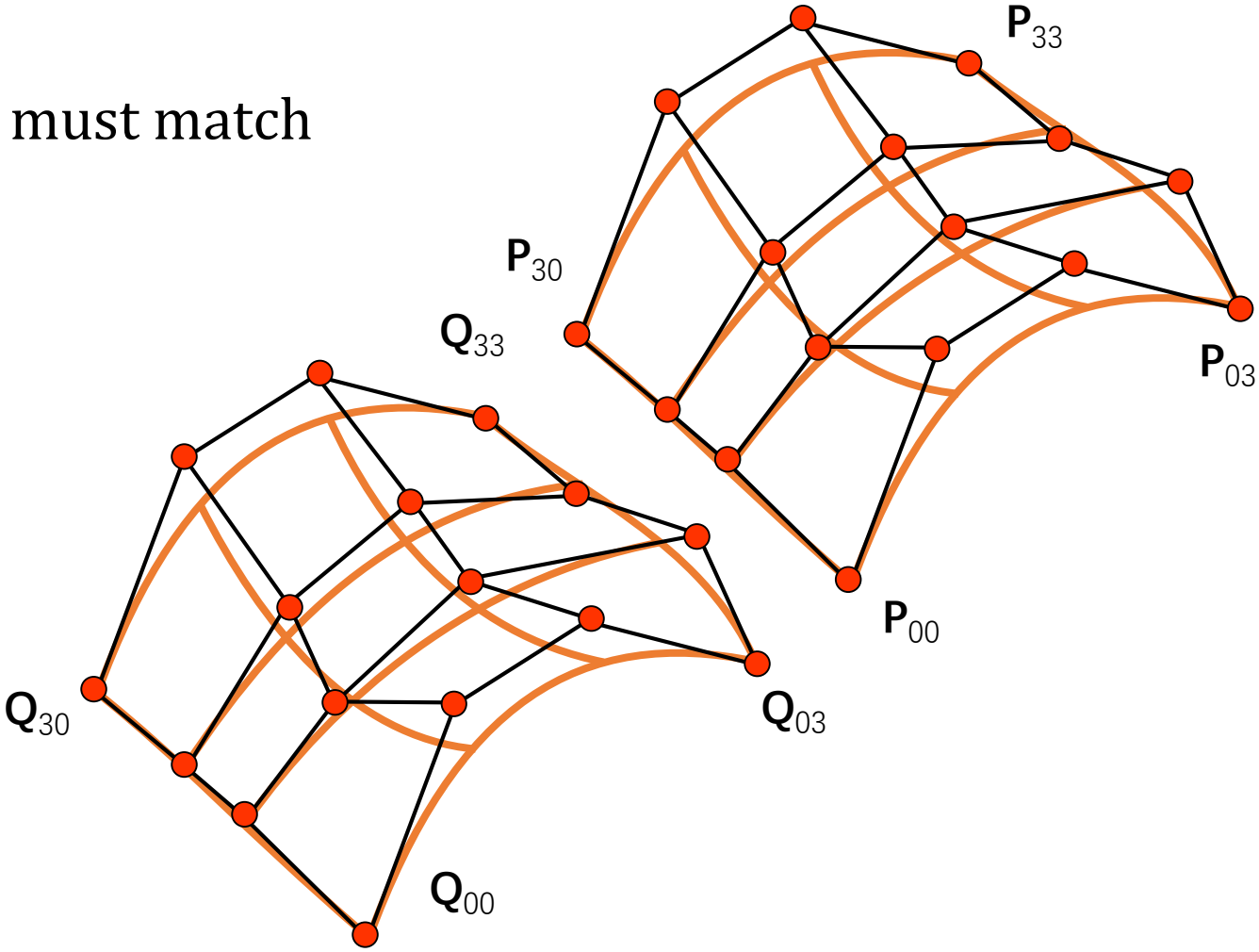
Bézier surface

Joining:
G^0: points must match



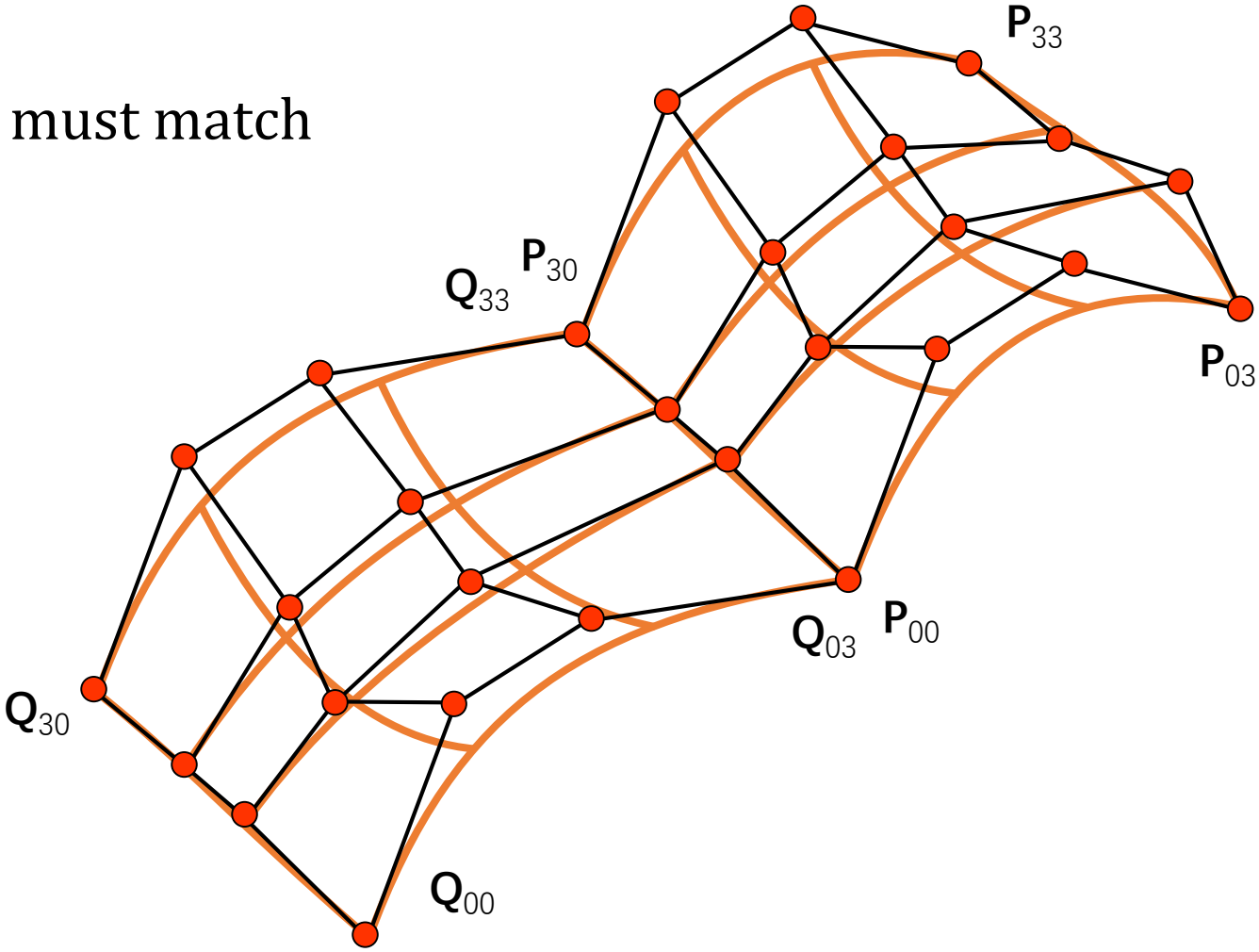
Bézier surface

Joining:
G^0: points must match



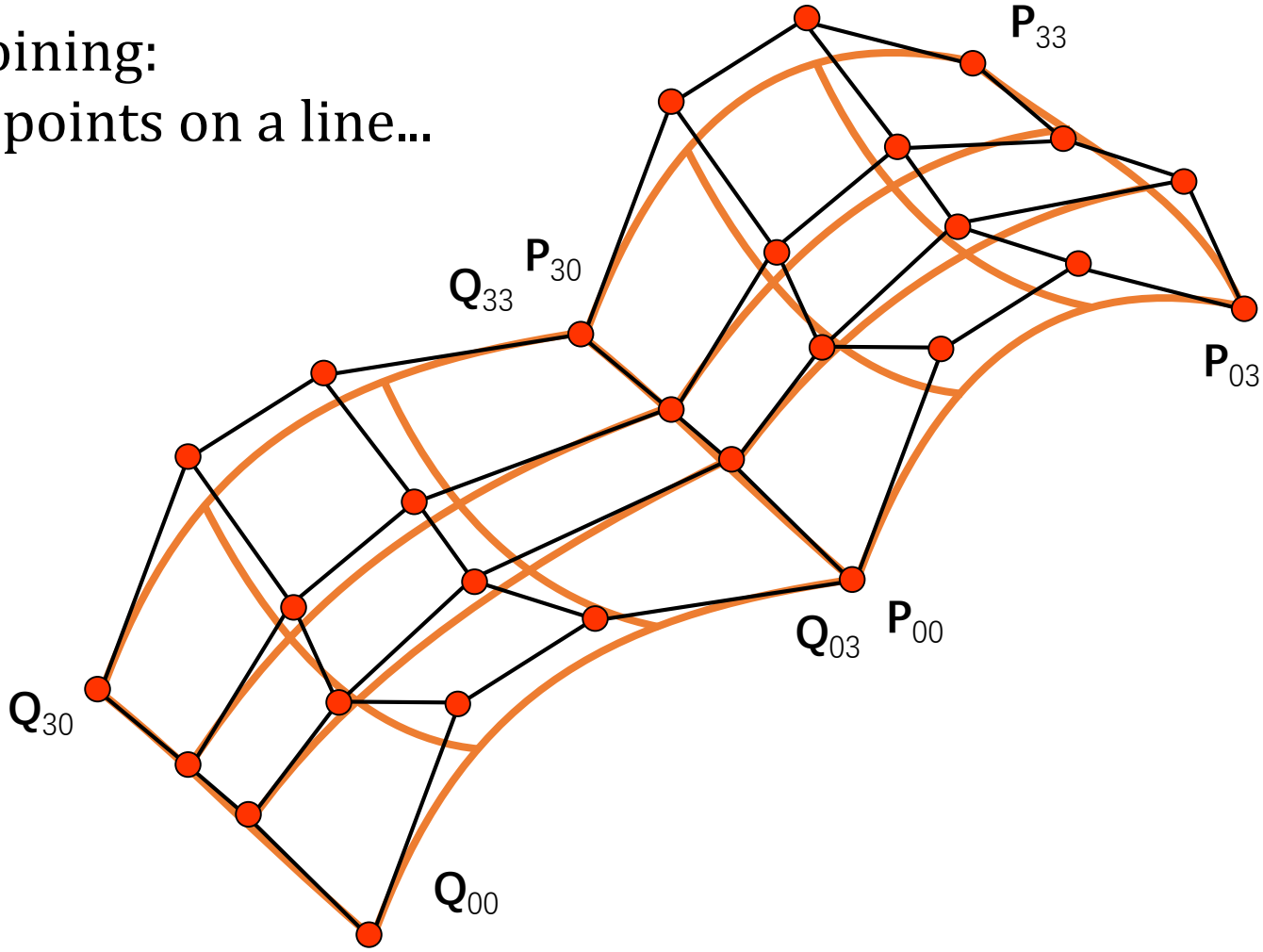
Bézier surface

Joining:
G^0: points must match



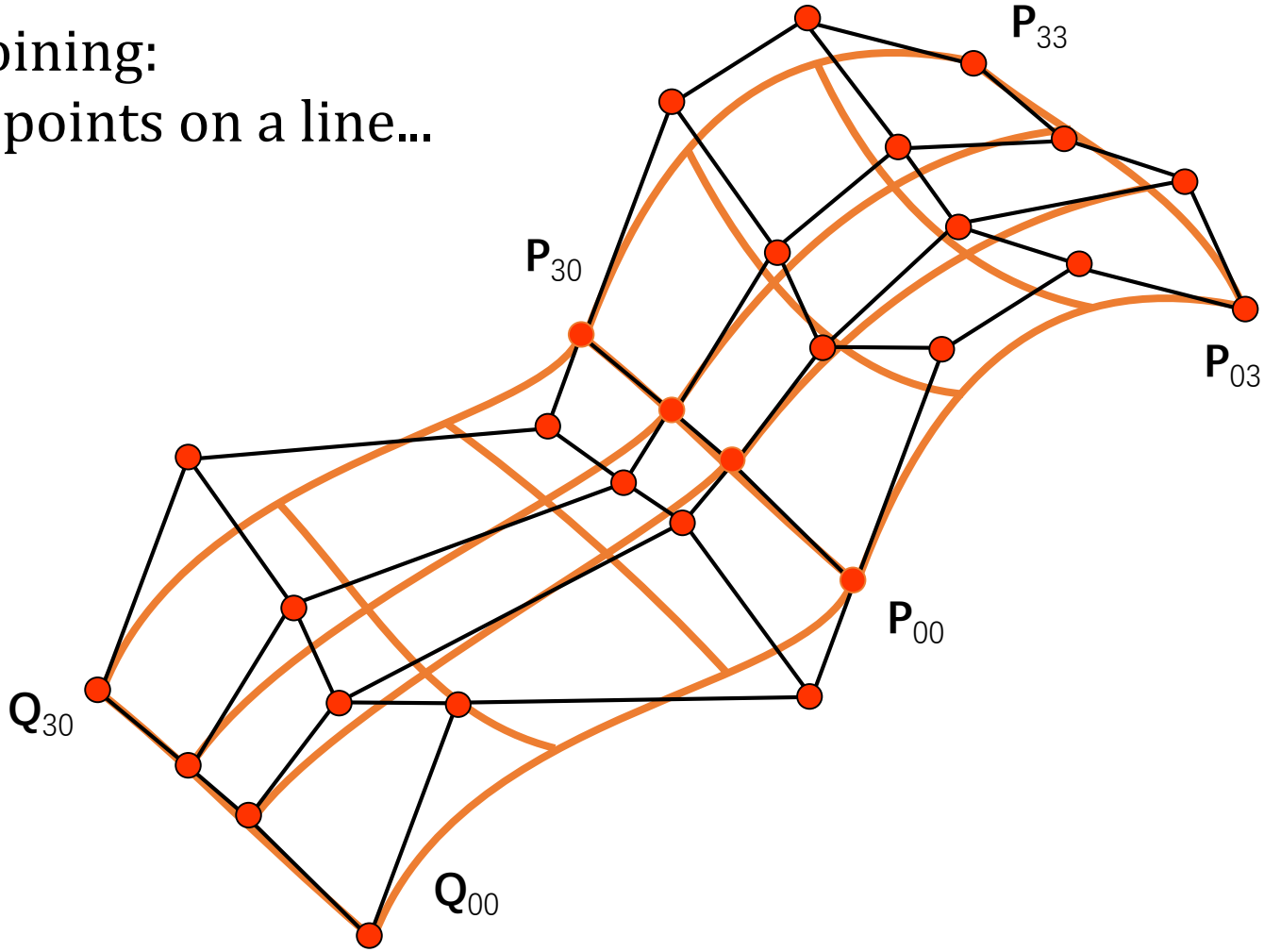
Bézier surface

Smooth joining:
G^1: three points on a line...

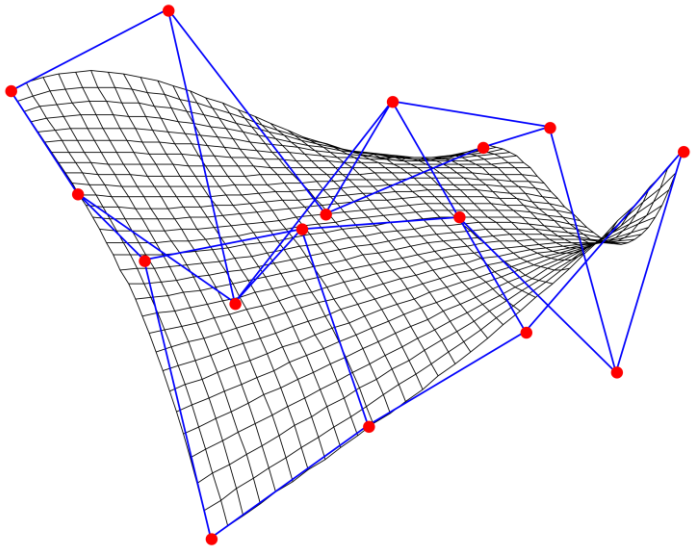


Bézier surface

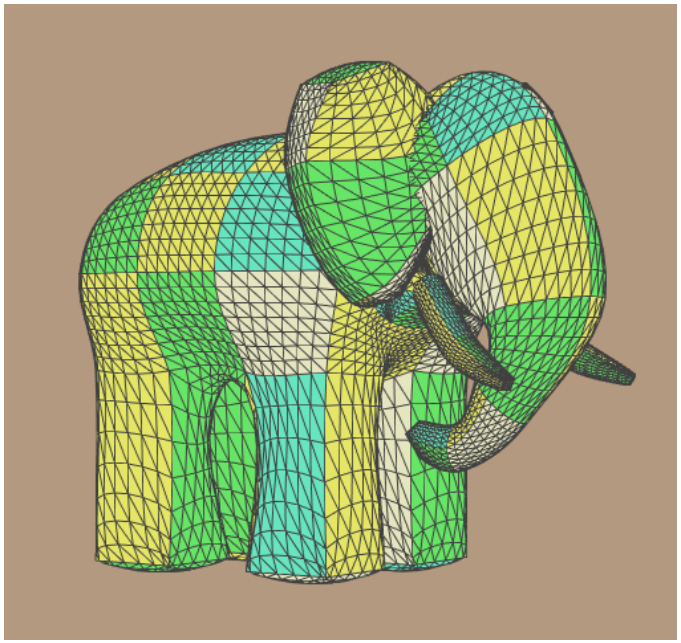
Smooth joining:
G^1: three points on a line...



Bézier Patches

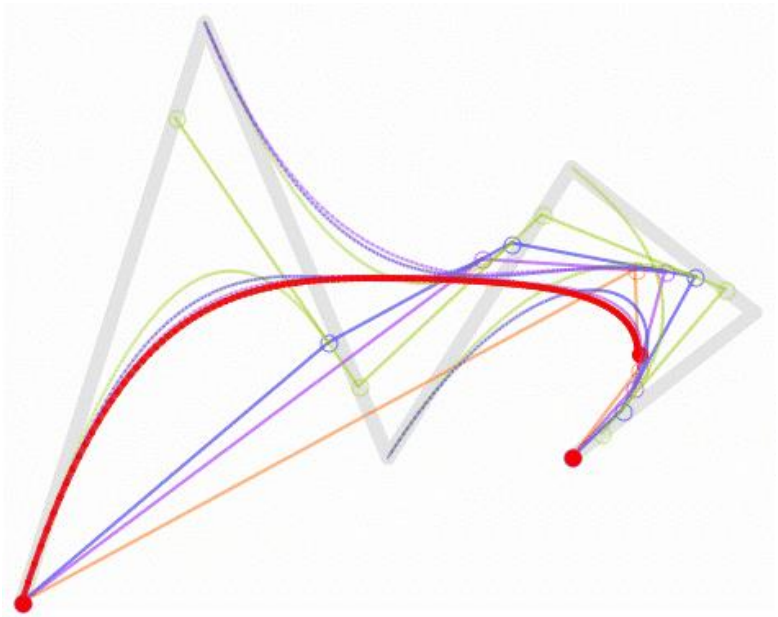


https://en.wikipedia.org/wiki/B%C3%A9zier_surface



Ed Catmull's "Gumbo" model composed from patches

Problem of Bézier Curve

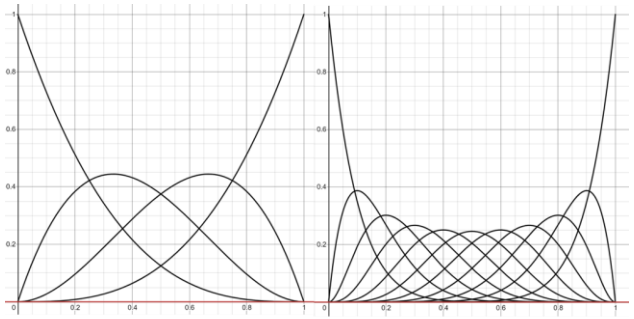


Demo: https://www.desmos.com/calculator/twyfn7nrac

$n + 1$ control points, order-n Bézier curve

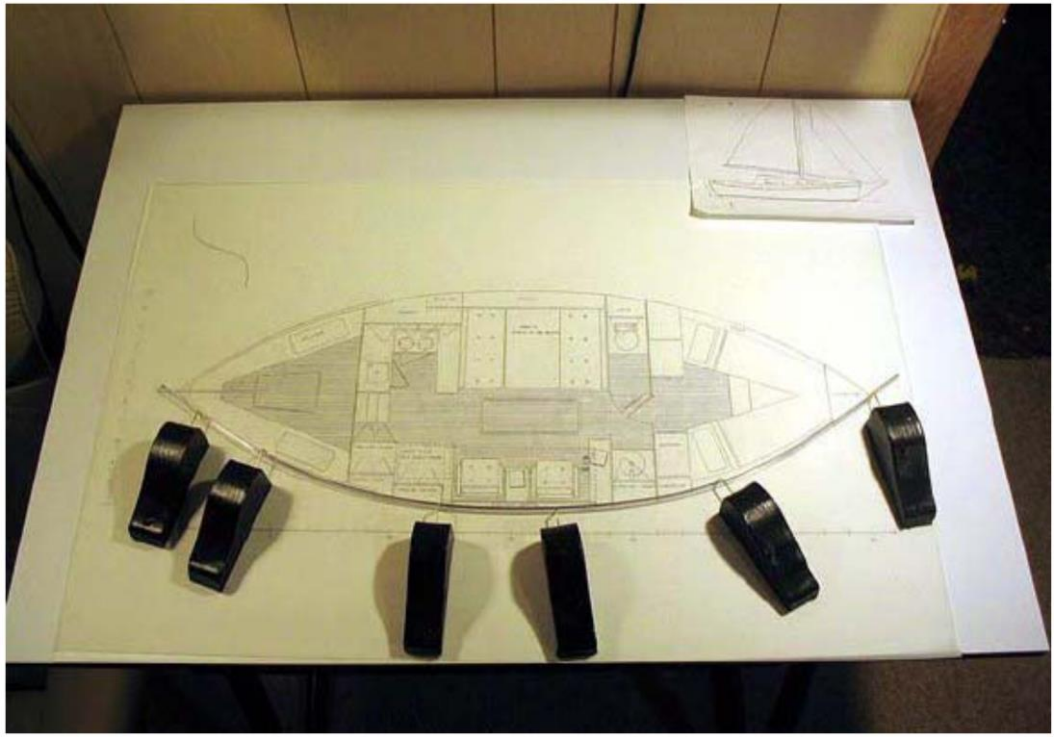
Every control point effects the entire curve

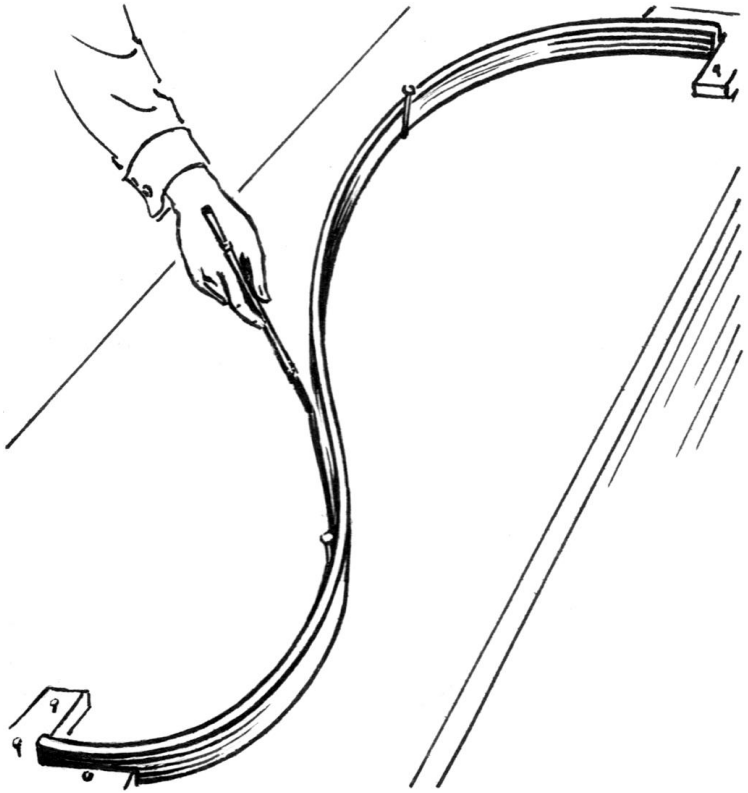
Global basis functions



$n = 3$ $n = 10$

Recall: Splines

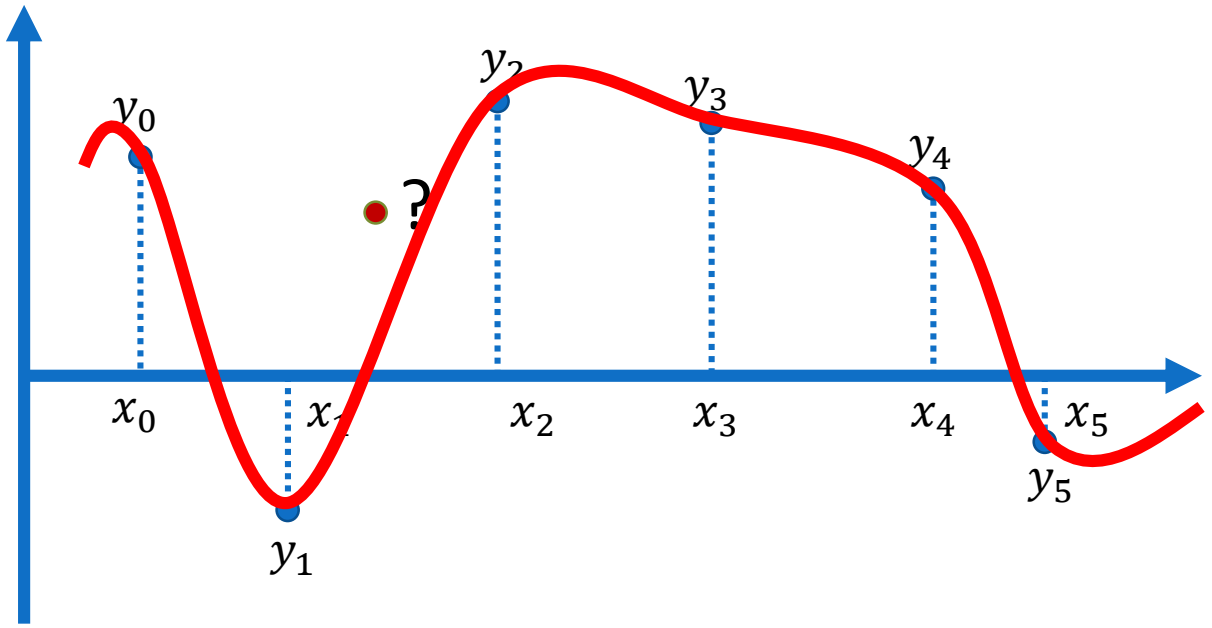




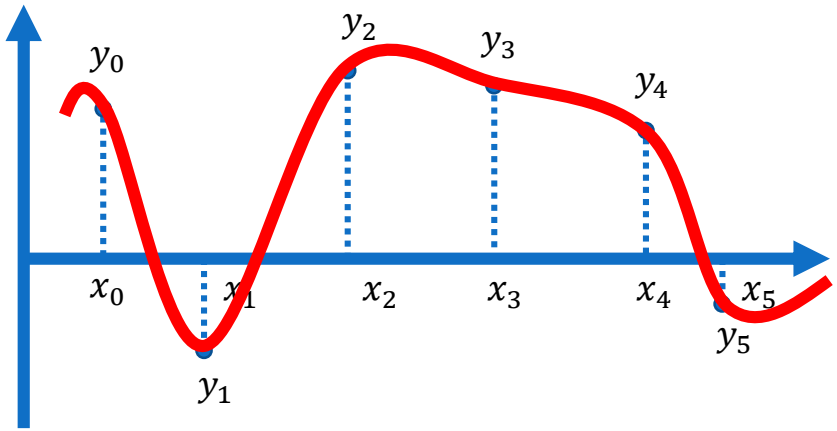
Recall: Splines

- 使用低阶多项式**分段**处理已知数据点
  - 例如：三次样条

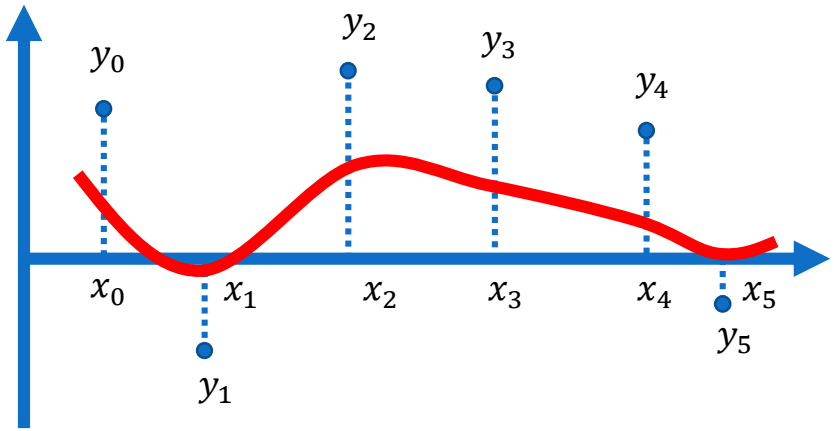
$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$$



Interpolation vs. Curve Fitting



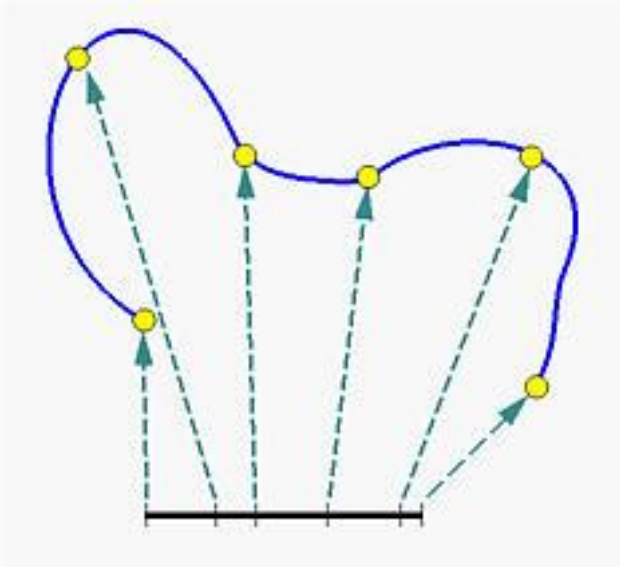
Interpolation



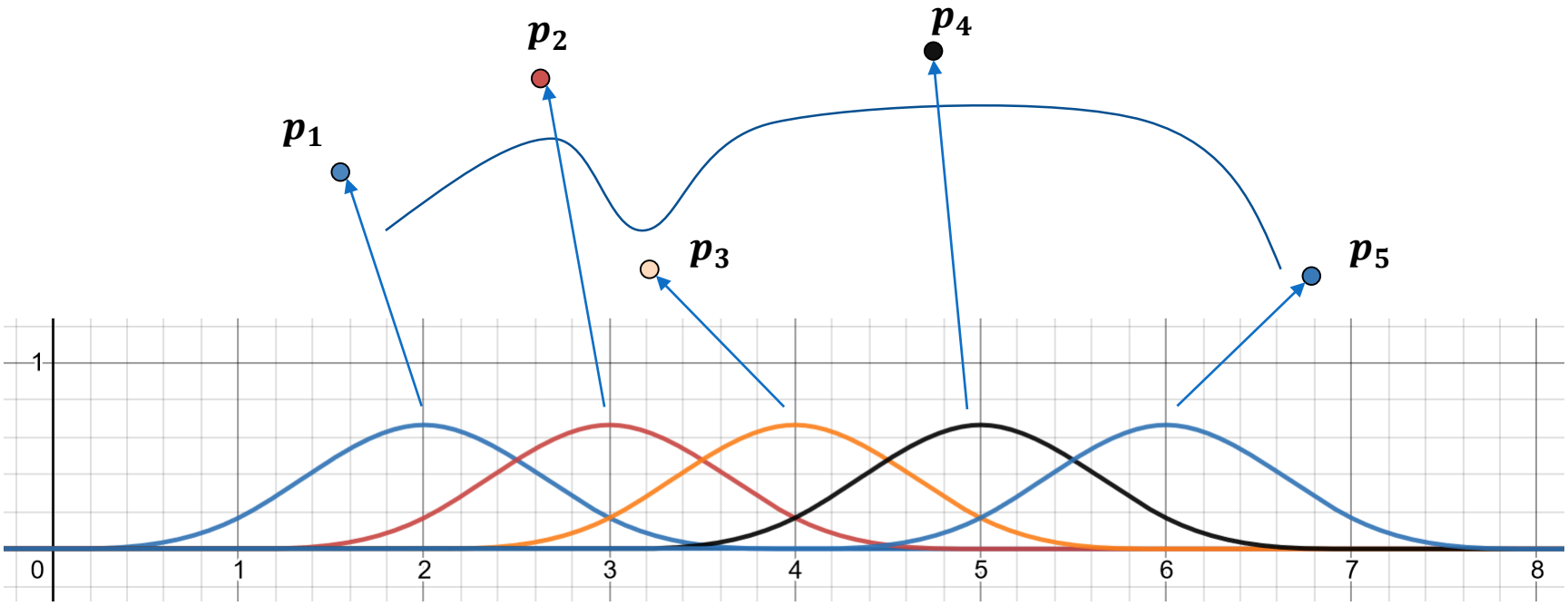
Fitting

B-Splines and Basis

- Piecewise polynomial
  - Continuous and smooth
- Local support
  - every control point only affect a part of the curve



B-Splines and Basis



$b_1(t)$ $b_2(t)$ $b_3(t)$ $b_4(t)$ $b_5(t)$

$$\boldsymbol{p}(t) = \sum_i b_i(t)\boldsymbol{p}_i$$

B-Splines and Basis

- To define B-Spline basis functions, we need one more parameter.
- The degree of these basis functions, p. The i-th B-spline basis function of degree p, written as $N_{i,p}(t)$, is defined recursively as follows (Cox-de Boor recursion):

$$N_{i,0}(0) = \begin{cases} 1 & \text{if } t_i \leq t < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

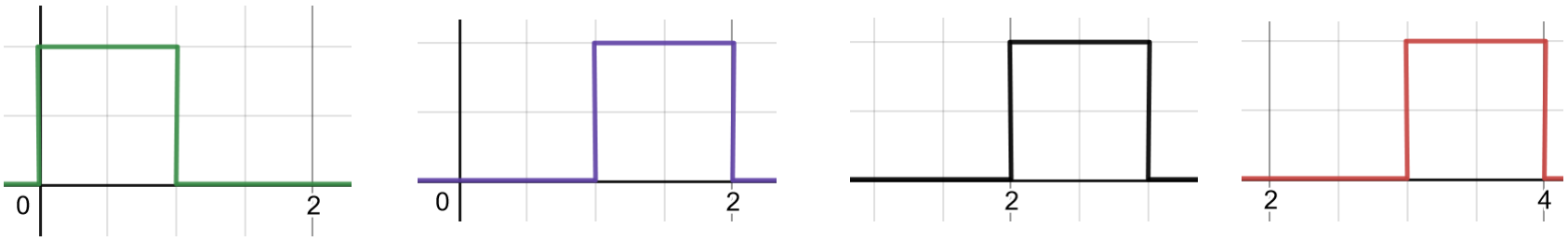
$$N_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t)$$

B-Spline basis functions from Zero to N

$$N_{i,0}(0) = \begin{cases} 1 & \text{if } t_i \le t < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t)$$

- Group of zero order base $N_{i,0}$ (here $i = 1 \dots 4$)



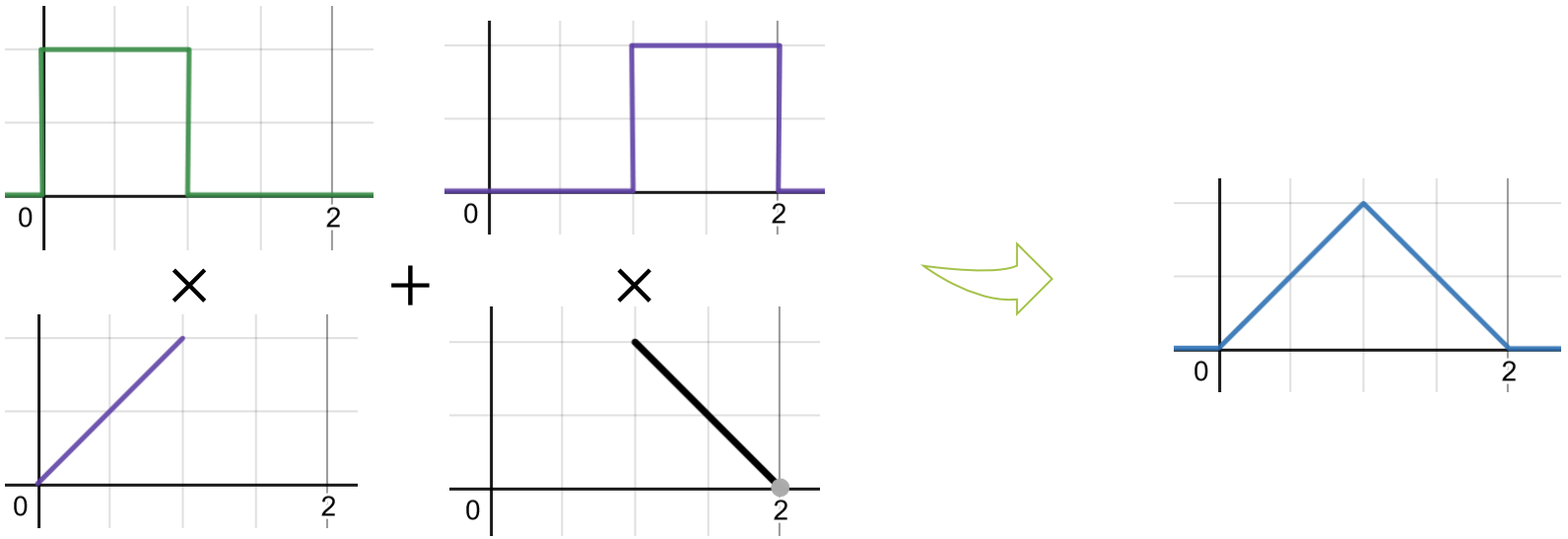
- Each control point controls a interval $[u_i, u_{i+1}]$ (e.g [0,1])

B-Spline basis functions from Zero to N

$$N_{i,0}(0) = \begin{cases} 1 & \text{if } t_i \le t < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t)$$

- The higher order is computed recursively

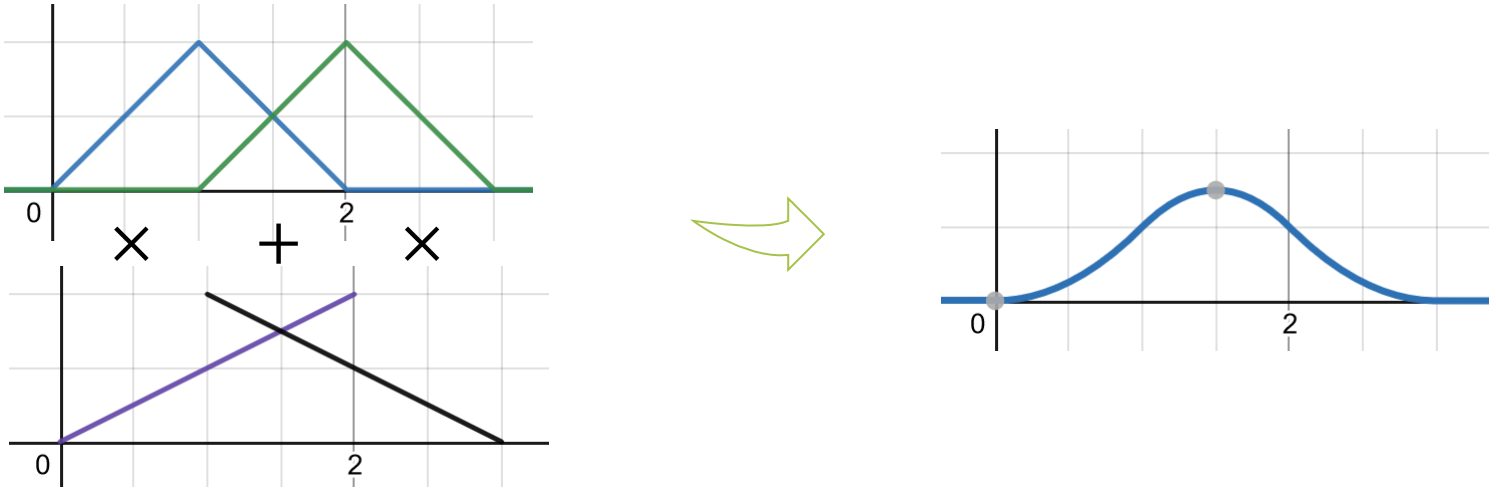


B-Spline basis functions from Zero to N

$$N_{i,0}(0) = \begin{cases} 1 & \text{if } t_i \le t < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t)$$

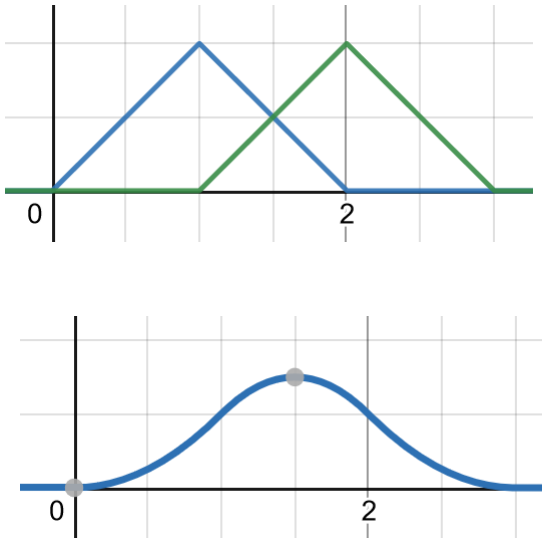
- The higher order is computed recursively



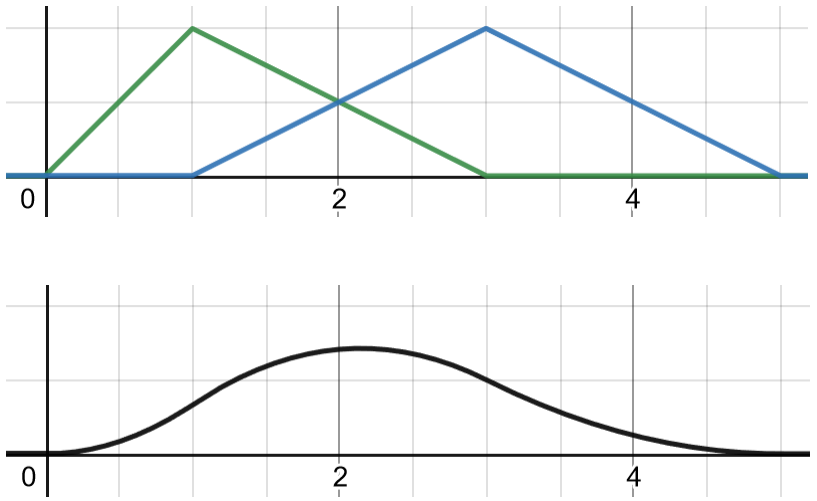
Uniform & Non-Uniform B-Splines

$$N_{i,0}(0) = \begin{cases} 1 & \text{if } t_i \le t < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t)$$



Uniform

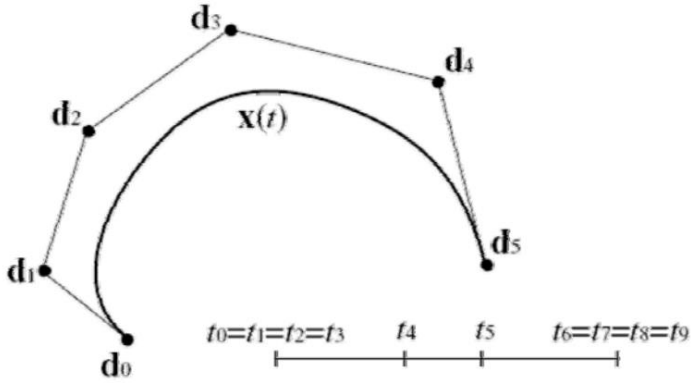


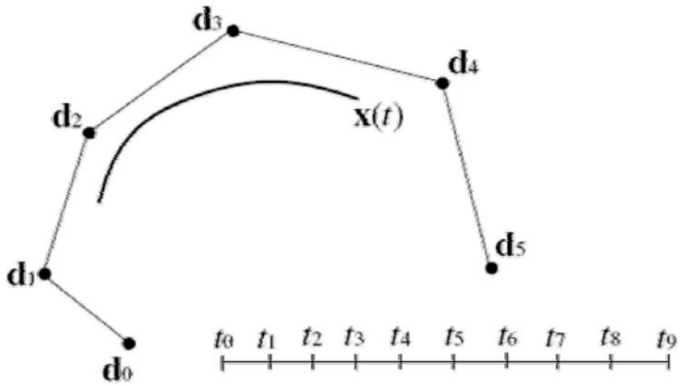
Non-Uniform

Uniform & Non-Uniform B-Splines

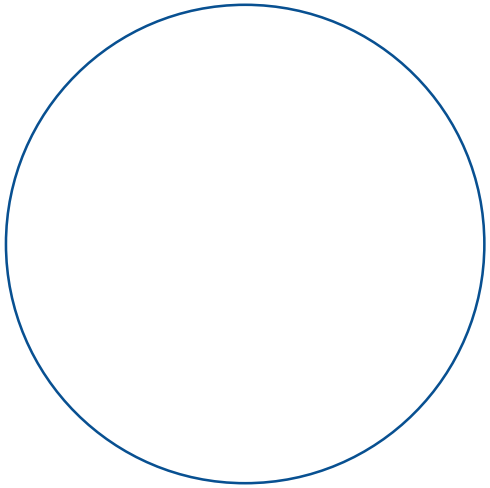
$$N_{i,0}(0) = \begin{cases} 1 & \text{if } t_i \le t < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t)$$





Problem of Bézier Curve & B-Splines



Can we create a circle using Bézier curve or B-Splines?

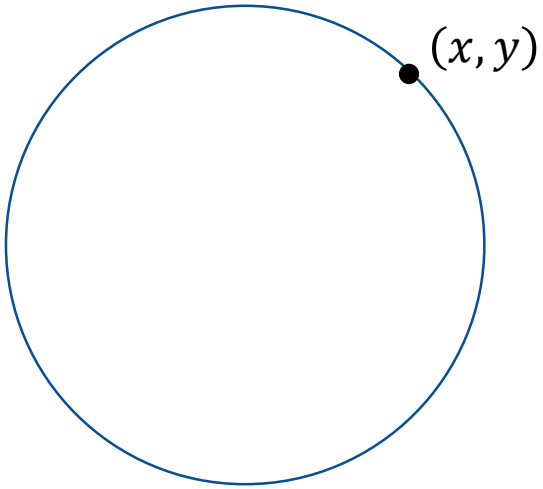
How? Why?

Hint: consider the equation of a circle

$$x^2 + y^2 = r^2$$

Also the same for any conic section

Rational Function



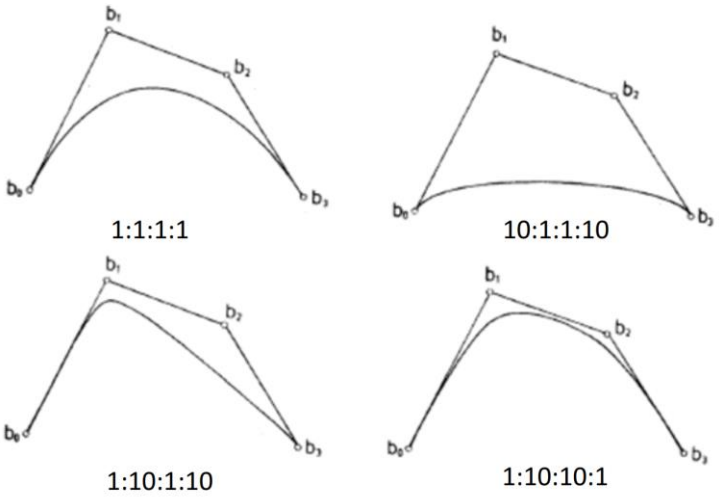
$$\begin{cases} x = \sin(\theta) = \dfrac{1 - t^2}{1 + t^2} \\ y = \cos(\theta) = \dfrac{2t}{1 + t^2} \end{cases}$$

Rational Function: $f(x) = \dfrac{a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0}{b_m x^m + b_{m-1} x^{m-1} + \cdots + b_0}$

Rational Bézier Curve

Bézier Curve

$$\boldsymbol{b}(t) = \sum_{k=0}^{n} B_{k,n}(t)\boldsymbol{u}_k$$



$$\Downarrow$$

Rational Bézier Curve

$$\boldsymbol{b}(t) = \frac{\sum_{k=0}^{n} B_{k,n}(t)\boldsymbol{u}_k}{\sum_{k=0}^{n} B_{k,n}(t)}$$

ω_k: the "weight" of a control point

$$\Downarrow$$

$$\boldsymbol{b}(t) = \frac{\sum_{k=0}^{n} B_{k,n}(t)\omega_k\boldsymbol{u}_k}{\sum_{k=0}^{n} B_{k,n}(t)\omega_k} = \sum_{k=0}^{n} \boldsymbol{u}_k \frac{B_{k,n}(t)\omega_k}{\sum_{k=0}^{n} B_{k,n}(t)\omega_k} = \sum_{k=0}^{n} q_k(t)\boldsymbol{u}_k$$

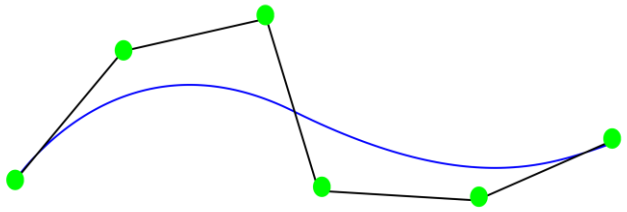
$^*\sum_{k=0}^{n} q_k(t) = 1$

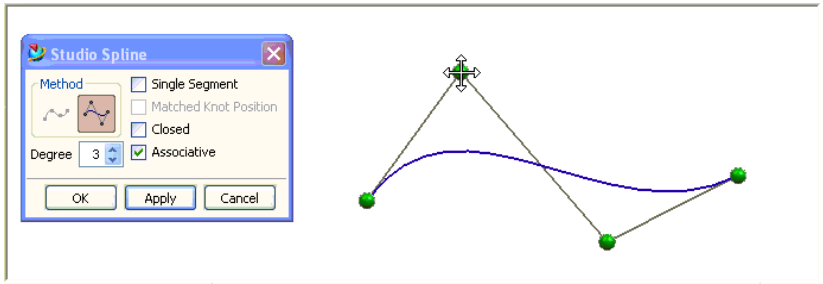
NURBS: Non-Uniform Rational B-Spline

$$N_{i,0}(0) = \begin{cases} 1 & \text{if } t_i \le t < t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t)$$

$$f(t) = \sum_{k=0}^{n} \frac{N_{k,n}(t)\omega_k \boldsymbol{u}_k}{\sum_{k=0}^{n} N_{k,n}(t)\omega_k}$$





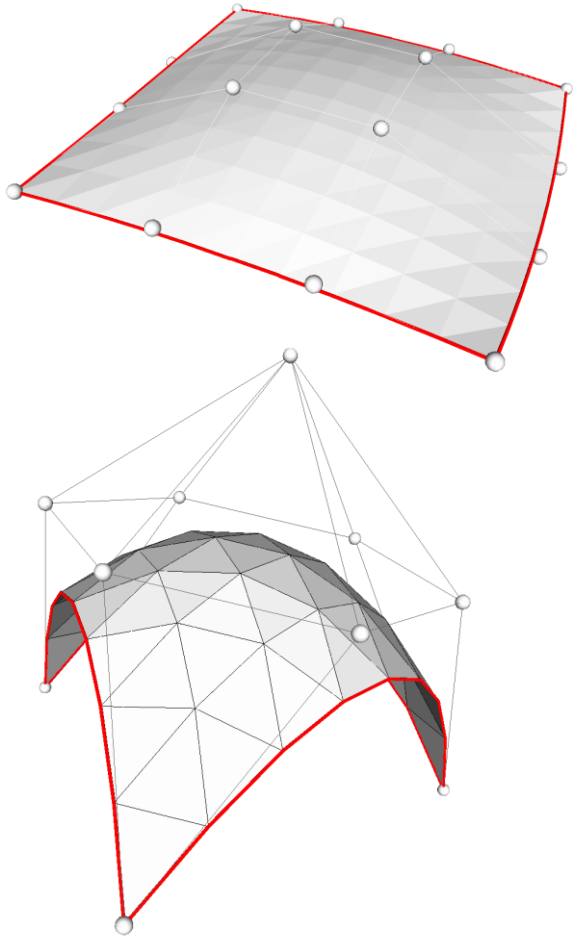
https://en.wikipedia.org/wiki/Non-uniform_rational_B-spline

NURBS Surfaces

- Tensor product surface of NURBS

$$S(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} R_{i,j}(u,v)\boldsymbol{p}_{i,j}$$

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



- Industry standard
  - AutoCAD, SolidWorks, Rhino, 3DS Max, Maya, …
- Built-in OpenGL support
  - Evaluators

Subdivision Surface

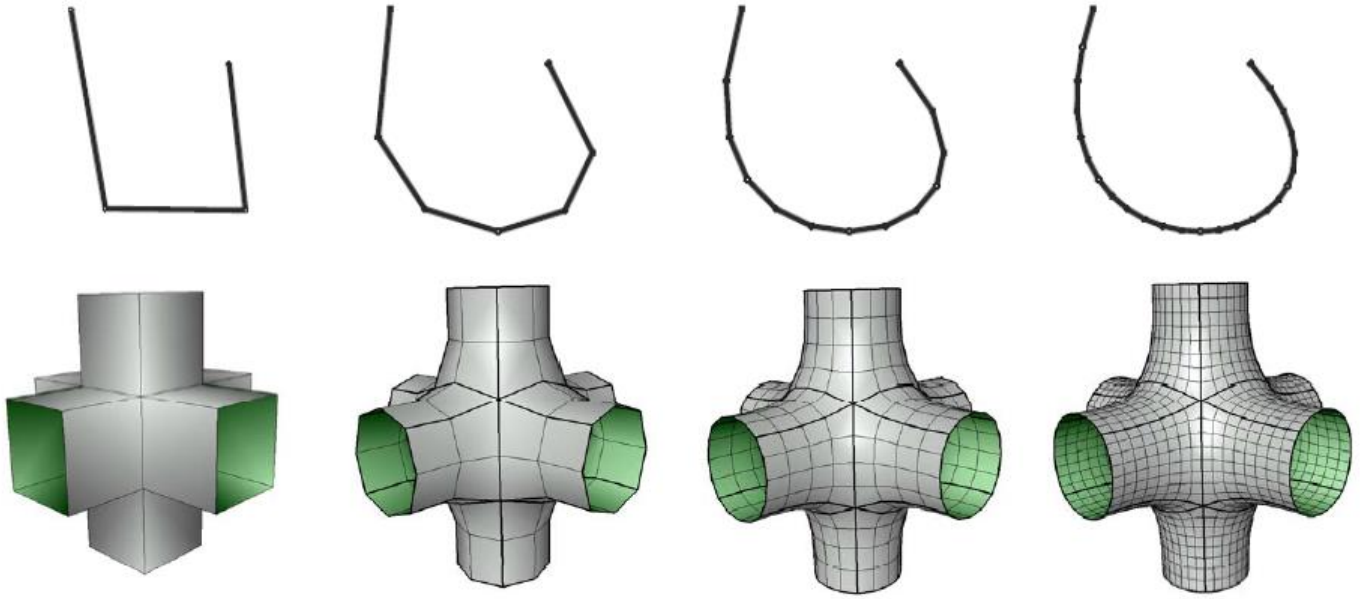
Problems of NURBS Surfaces

- Trimming is expensive and prone to numerical error
- Difficult to maintain smoothness, or even approximate smoothness, at the seams of the patchwork as the model is animated

Tony DeRose, Michael Kass, and Tien Truong. 1998. ***Subdivision surfaces in character animation***. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '98), Association for Computing Machinery, New York, NY, USA, 85–94

Subdivision surface

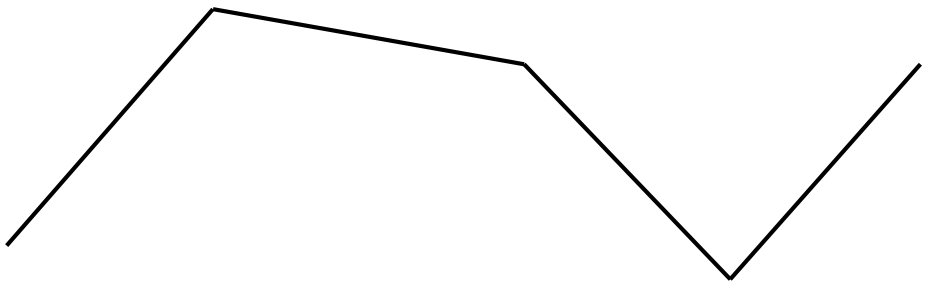
- Subdivision defines a smooth curve or surface as the limit of a sequence of successive refinements



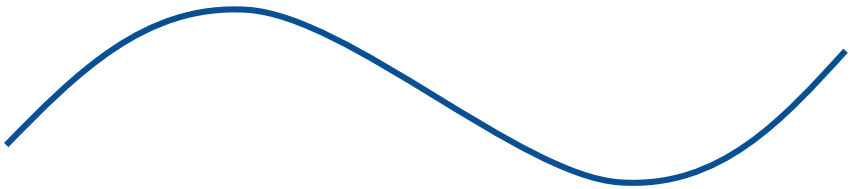
https://mrl.cs.nyu.edu/publications/subdiv-course2000/

Subdivision Curve

Given

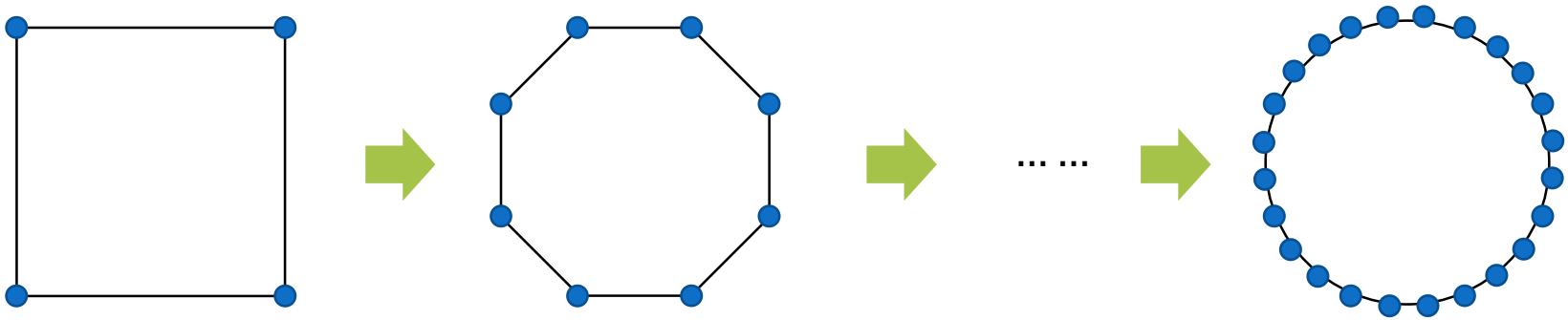


Compute



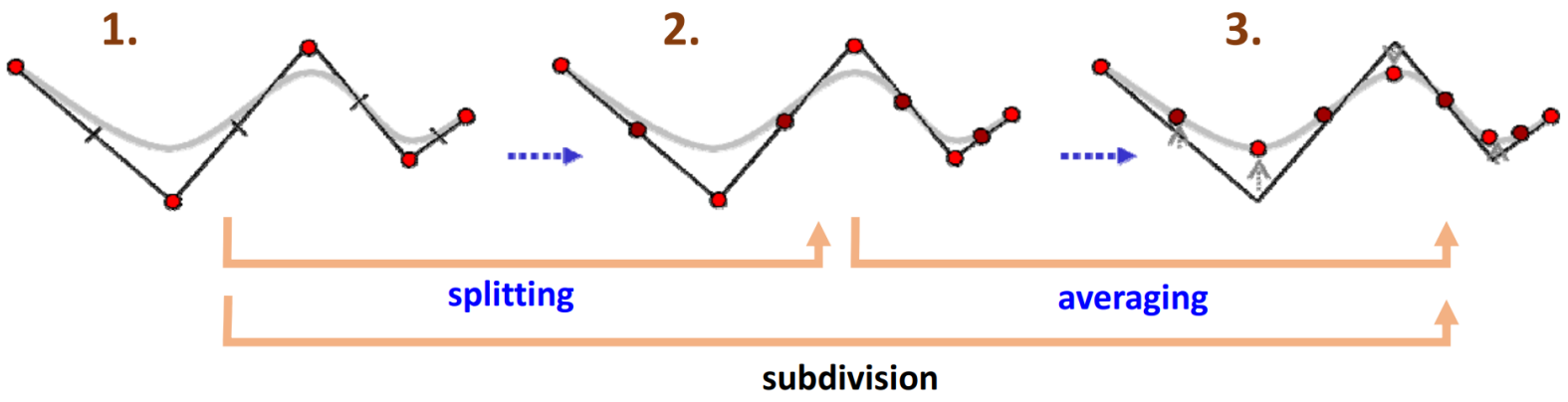
Subdivision Iteration

- Basic idea: corner cutting

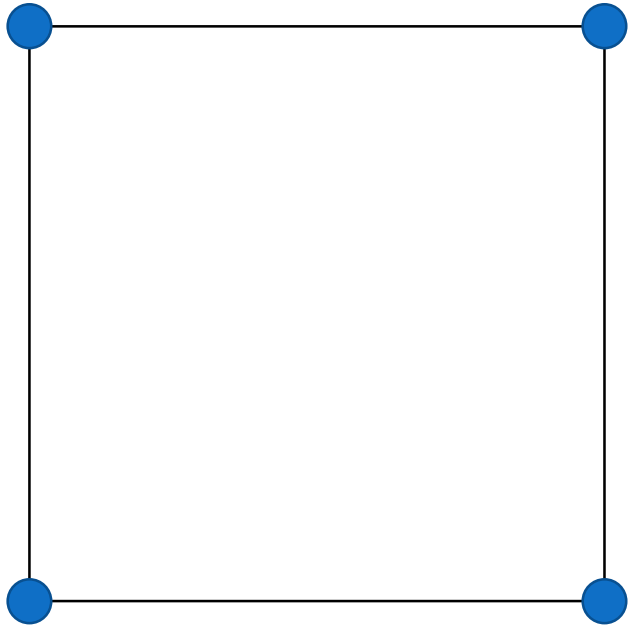


Subdivision Iteration

- Basic idea: corner cutting
- Two steps:
  - Splitting: adding new vertices
  - Averaging: moving existing vertices

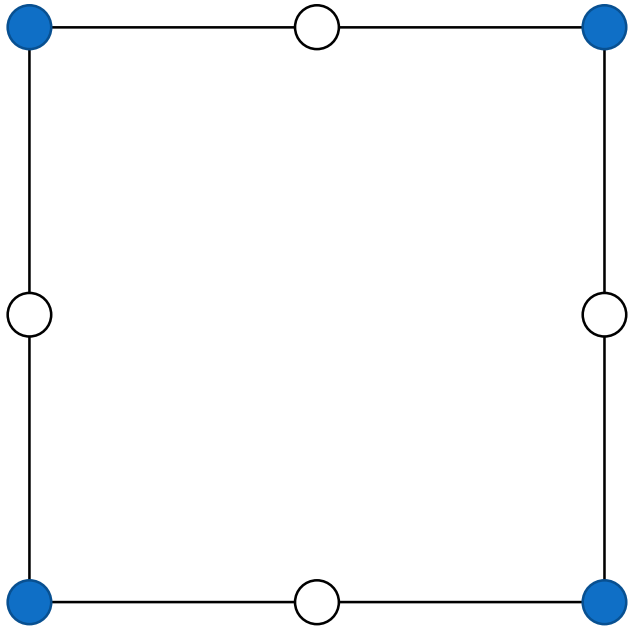


Chaikin’s Algorithm



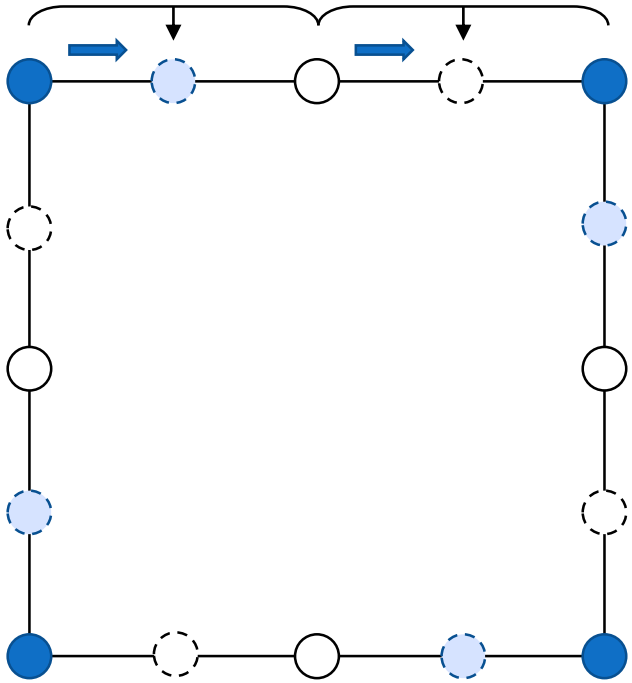
Chaikin's Algorithm

Split



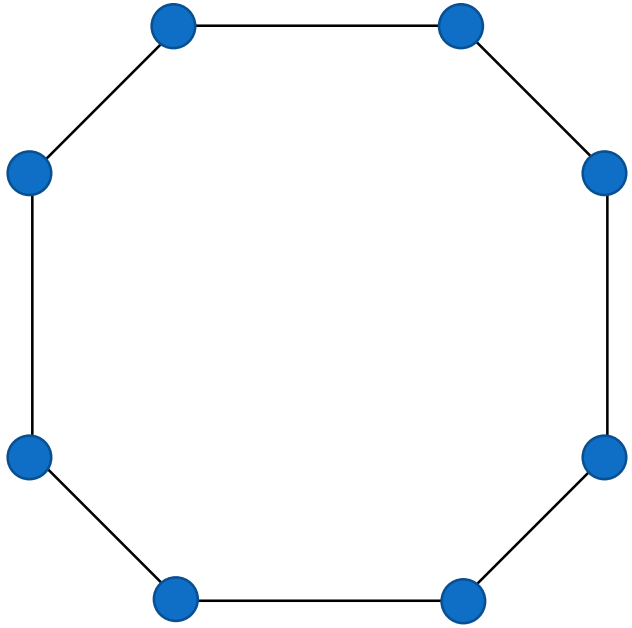
Chaikin's Algorithm

Average

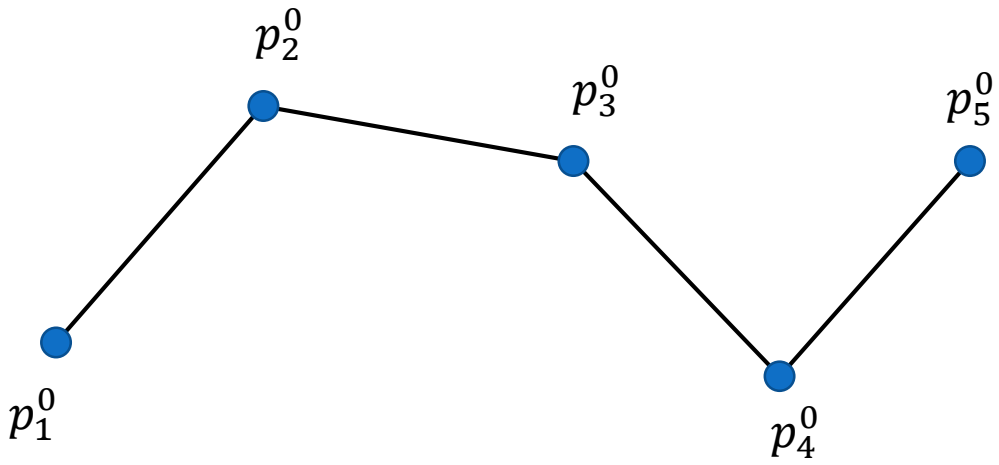


Chaikin's Algorithm

Average

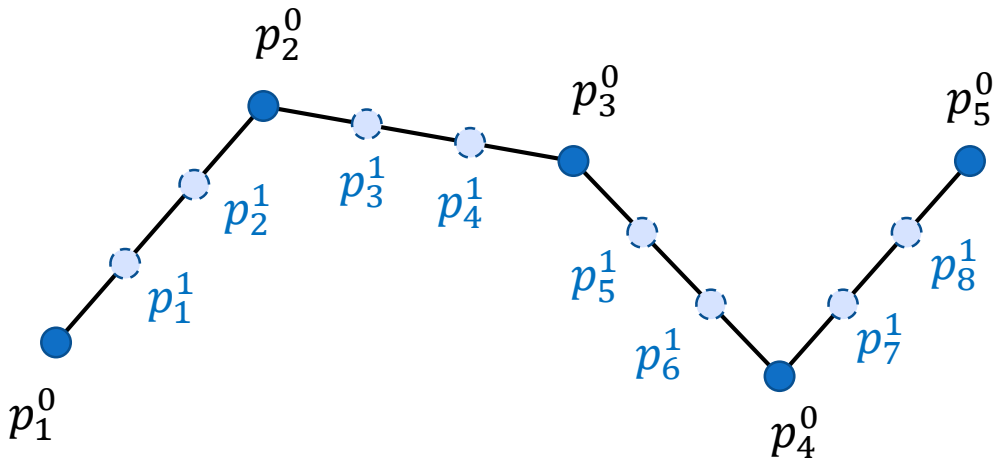


Chaikin's Algorithm



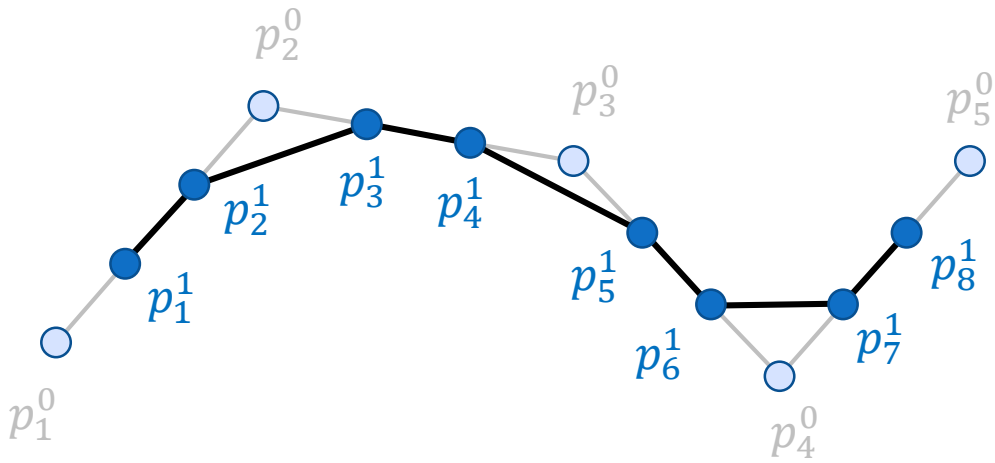
Chaikin's Algorithm

$$p_{2i-1}^{n+1} = \frac{3}{4}p_i^n + \frac{1}{4}p_{i+1}^n \qquad p_{2i}^{n+1} = \frac{1}{4}p_i^n + \frac{3}{4}p_{i+1}^n$$

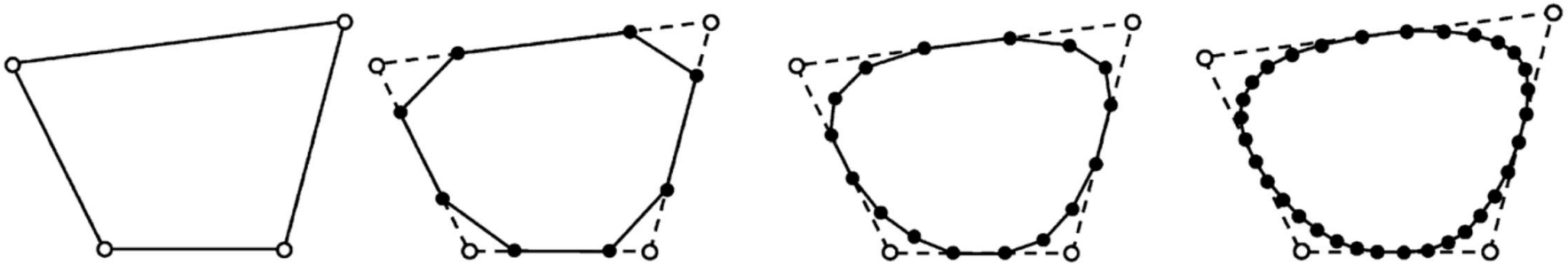


Chaikin's Algorithm

$$p_{2i-1}^{n+1} = \frac{3}{4}p_i^n + \frac{1}{4}p_{i+1}^n \qquad p_{2i}^{n+1} = \frac{1}{4}p_i^n + \frac{3}{4}p_{i+1}^n$$



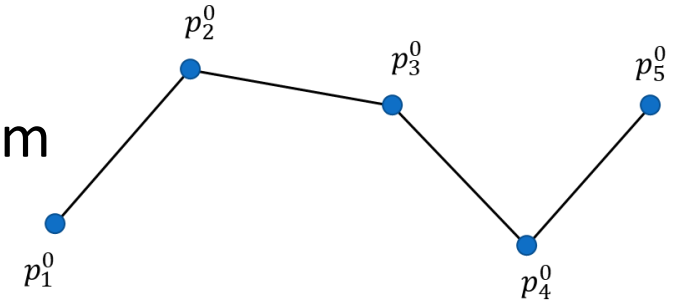
Chaikin's Algorithm



http://staff.ustc.edu.cn/~lgliu/Courses/GAMES102_2020/PPT/GAMES102-8_SubdivisionCurves.pdf

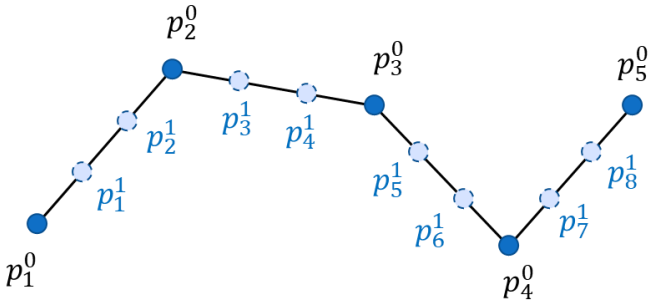
Chaikin's Algorithm

- The limit curve of Chaikin's Algorithm is quadratic uniform B-spline curve

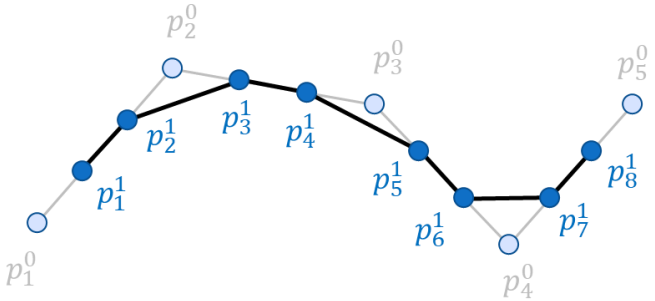


$$p_{2i-1}^{n+1} = \frac{3}{4}p_i^n + \frac{1}{4}p_{i+1}^n$$

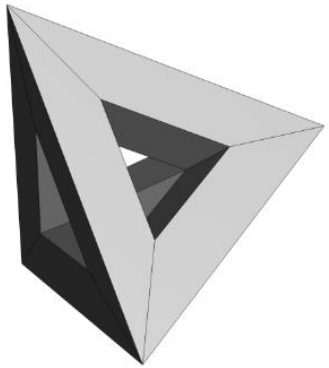
$$p_{2i}^{n+1} = \frac{1}{4}p_i^n + \frac{3}{4}p_{i+1}^n$$



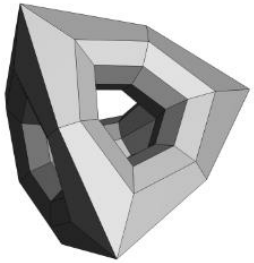
$$\begin{bmatrix} \vdots \\ p_{2i-2}^{n+1} \\ p_{2i-1}^{n+1} \\ p_{2i}^{n+1} \\ p_{2i+1}^{n+1} \\ p_{2i+2}^{n+1} \\ \vdots \end{bmatrix} = \frac{1}{4}\begin{bmatrix} \ddots & & & & & \iddots \\ & 0 & 1 & 3 & 0 & 0 \\ & 0 & 0 & 3 & 1 & 0 \\ & 0 & 0 & 1 & 3 & 0 \\ & 0 & 0 & 0 & 3 & 1 \\ & 0 & 0 & 0 & 1 & 3 \\ \iddots & & & & & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ p_{i-2}^n \\ p_{i-1}^n \\ p_i^n \\ p_{i+1}^n \\ p_{i+2}^n \\ \vdots \end{bmatrix}$$



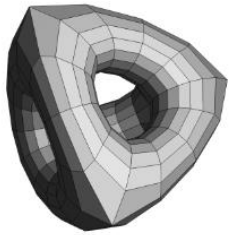
Subdivision Surface



Input mesh



After one subdivision step



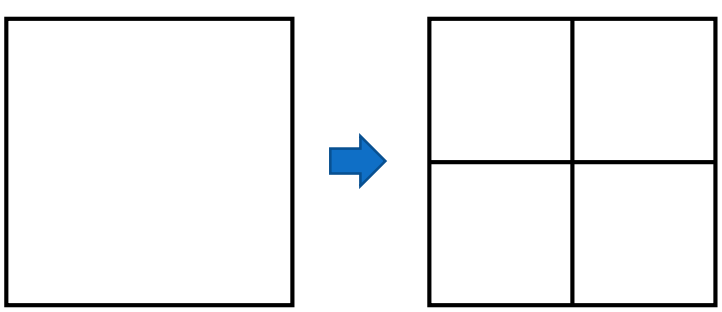
After two subdivision step

The limit surface

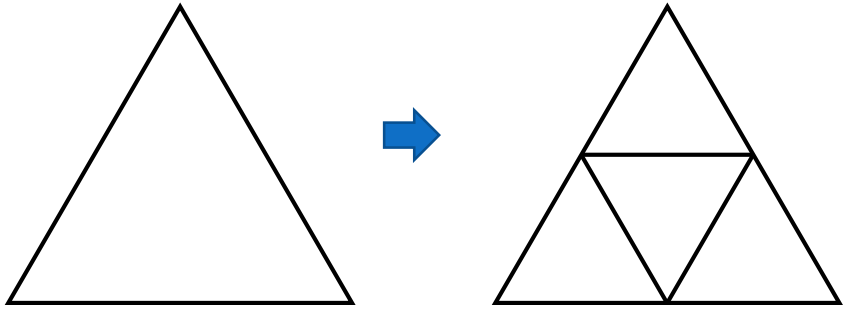
https://mrl.cs.nyu.edu/publications/subdiv-course2000/

Subdivision Surface

- New vertices are weighted average of old vertices
  - Splitting: adding new vertices on faces and edges
  - Averaging: moving existing vertices



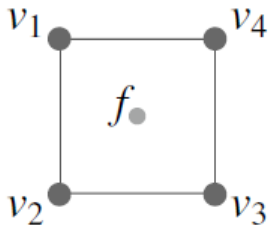
QuadMesh: Catmull-Clark



TriMesh: Loop

Catmull-Clark Subdivision

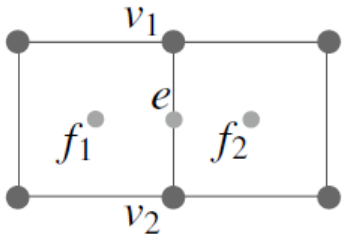
Face point

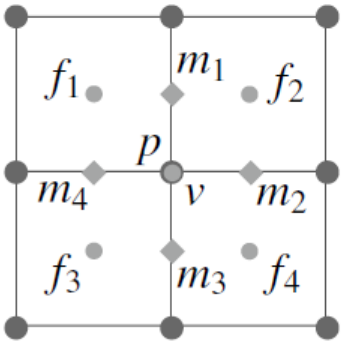


$$f = \frac{v_1 + v_2 + v_3 + v_4}{4}$$

$$e = \frac{v_1 + v_2 + f_1 + f_2}{4}$$

Edge point





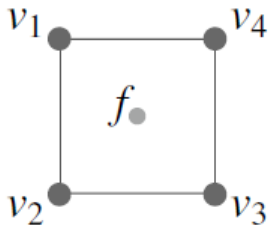
Vertex point

$$v = \frac{1}{n}Q + \frac{2}{n}R + \frac{n-3}{n}p$$

Q – the average of all new face points
R – the average of all edge-points
p – the old vertex
n – number of incident edges of p

Catmull-Clark Subdivision

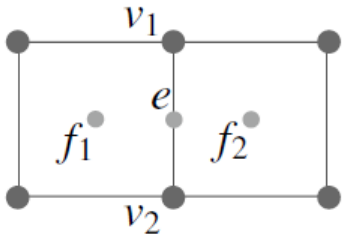
Face point

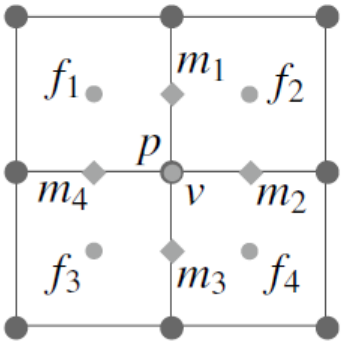


$$f = \frac{v_1 + v_2 + v_3 + v_4}{4}$$

$$e = \frac{v_1 + v_2 + f_1 + f_2}{4}$$

Edge point





Vertex point

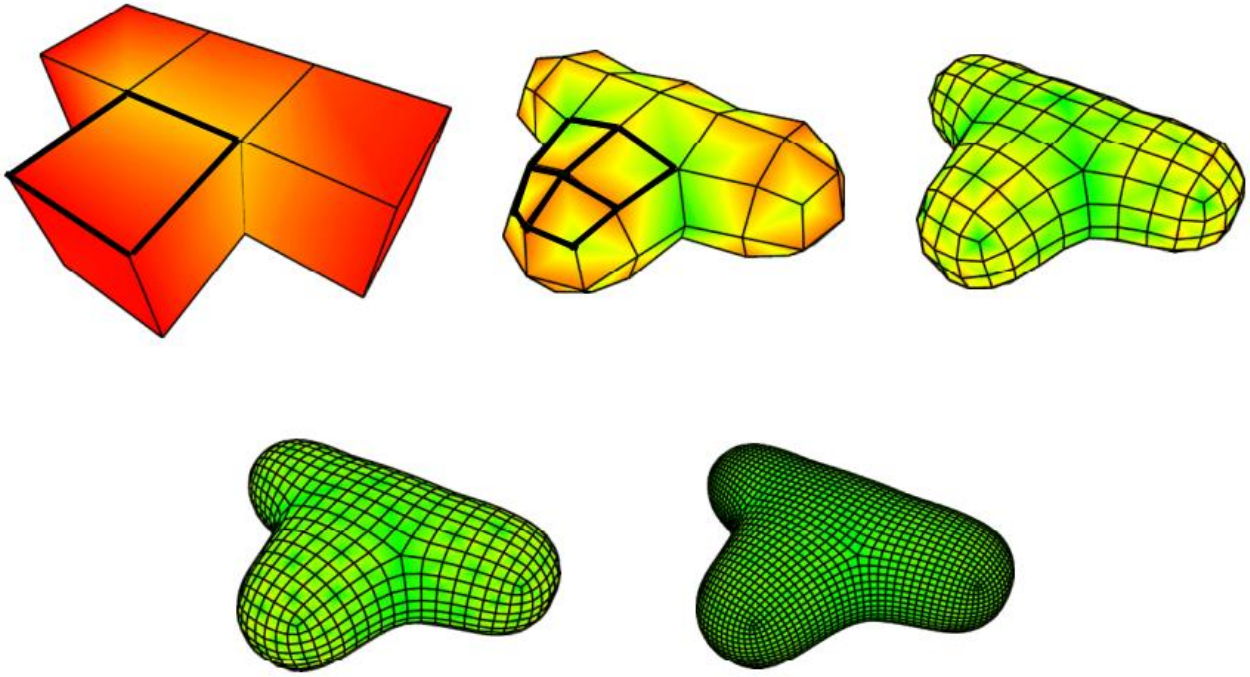
$$v = \frac{f_1 + f_2 + f_3 + f_4 + 2(m_1 + m_2 + m_3 + m_4) + 4p}{16}$$

m midpoint of edge, not "edge point"

p old "vertex point"

Catmull-Clark Subdivision

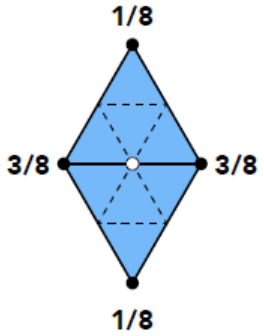
- Common subdivision rule for quad meshes
- Converge to bi-cubic B-Splines



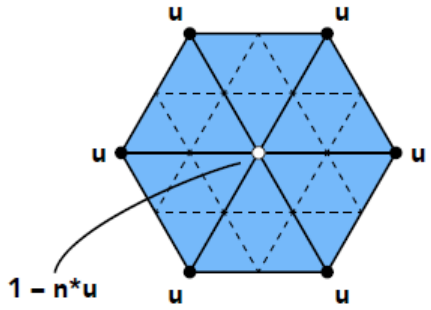
Loop subdivision algorithm

- Split each triangle into four

- Compute new vertex positions using weighted sum of prior vertex positions:



New vertices
(weighted sum of vertices on split edge, and vertices "across from" edge)



Old vertices
(weighted sum of edge adjacent vertices)

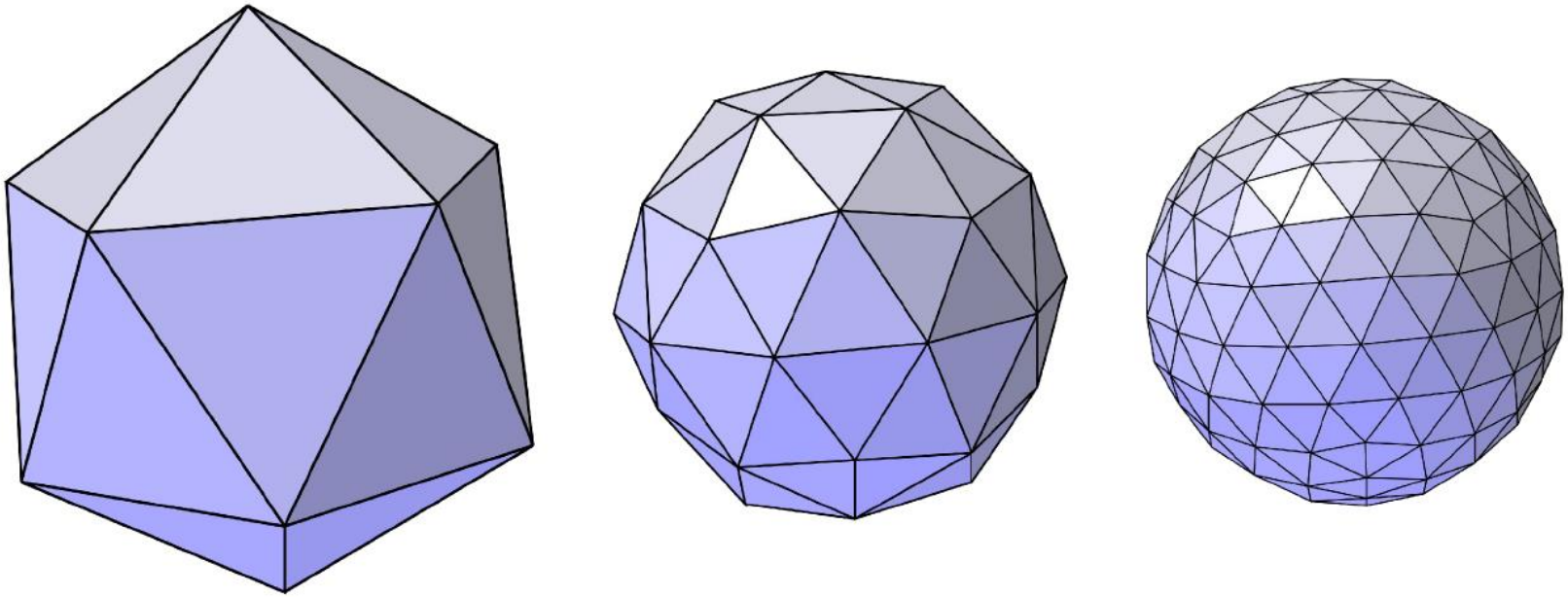
n = vertex degree

$$u = \begin{cases} \frac{3}{16}, & n = 3 \\ \frac{5}{8n} - \frac{1}{n}\left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n}\right)^2, & x > 3 \end{cases}$$

https://cmu-graphics.github.io/Scotty3D/meshedit/global/loop/

Loop subdivision

- Common subdivision rule for triangle meshes
- Converge to *Box Splines*

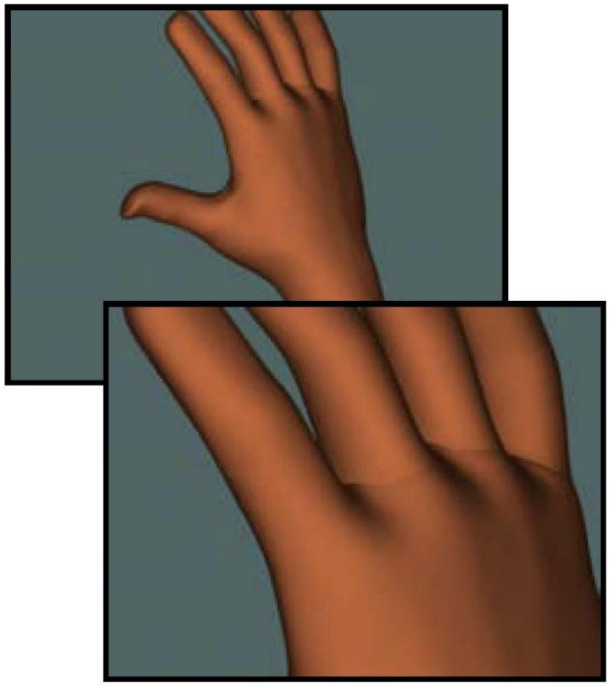


Example Geri's Game (Pixar)

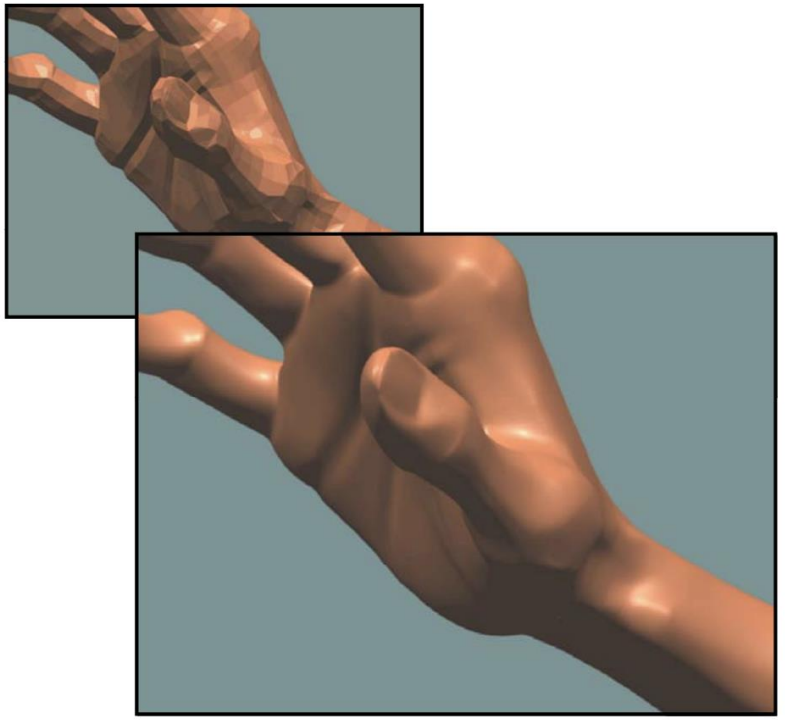
https://www.pixar.com/geris-game

Example Geri's Game (Pixar)

Woody's hand (NURBS)



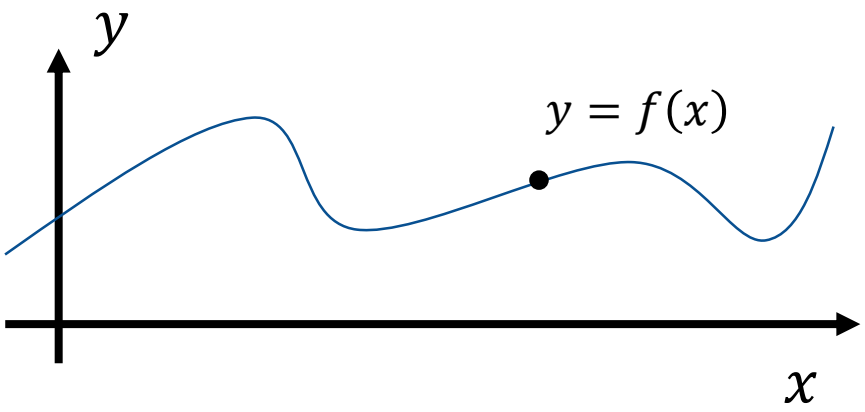
Geri's hand (subdivision)



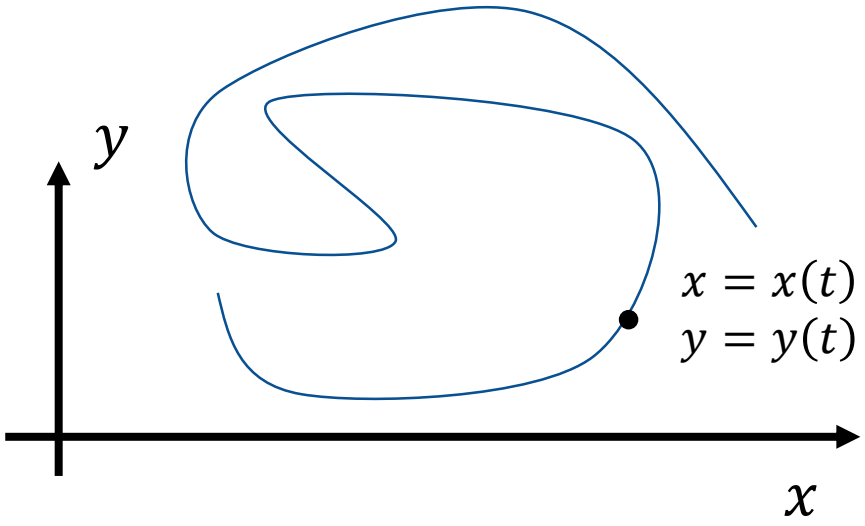
Implicit Surface

“Explicit” Surface

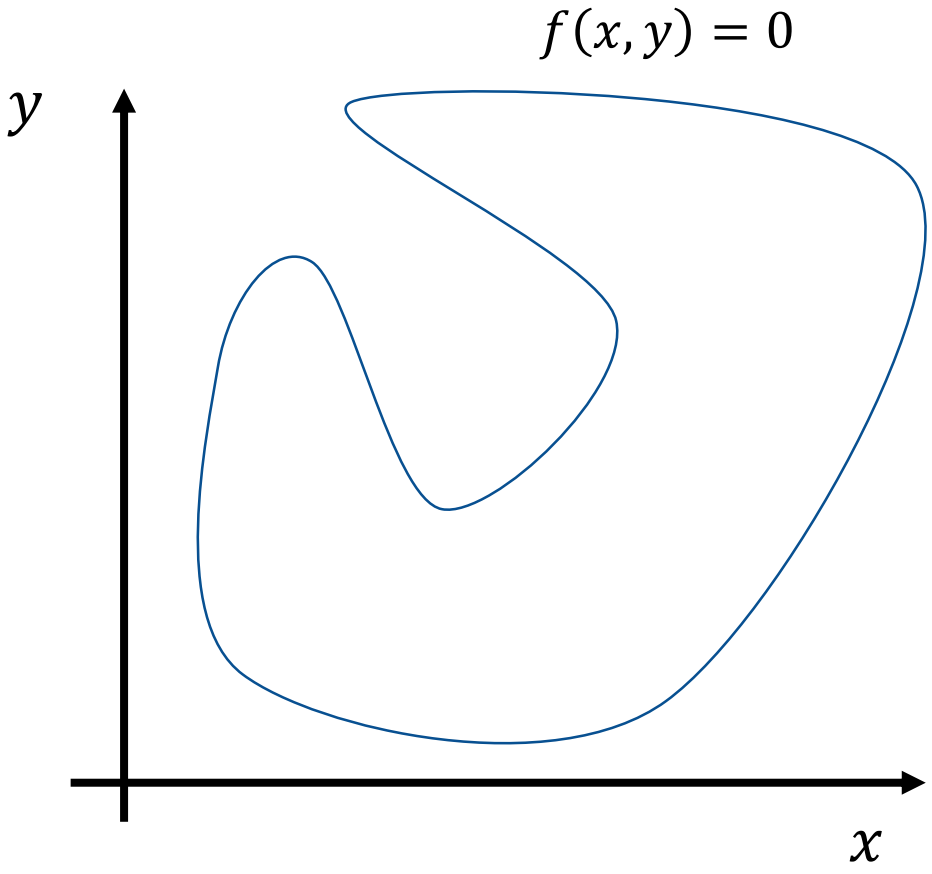
Explicit function



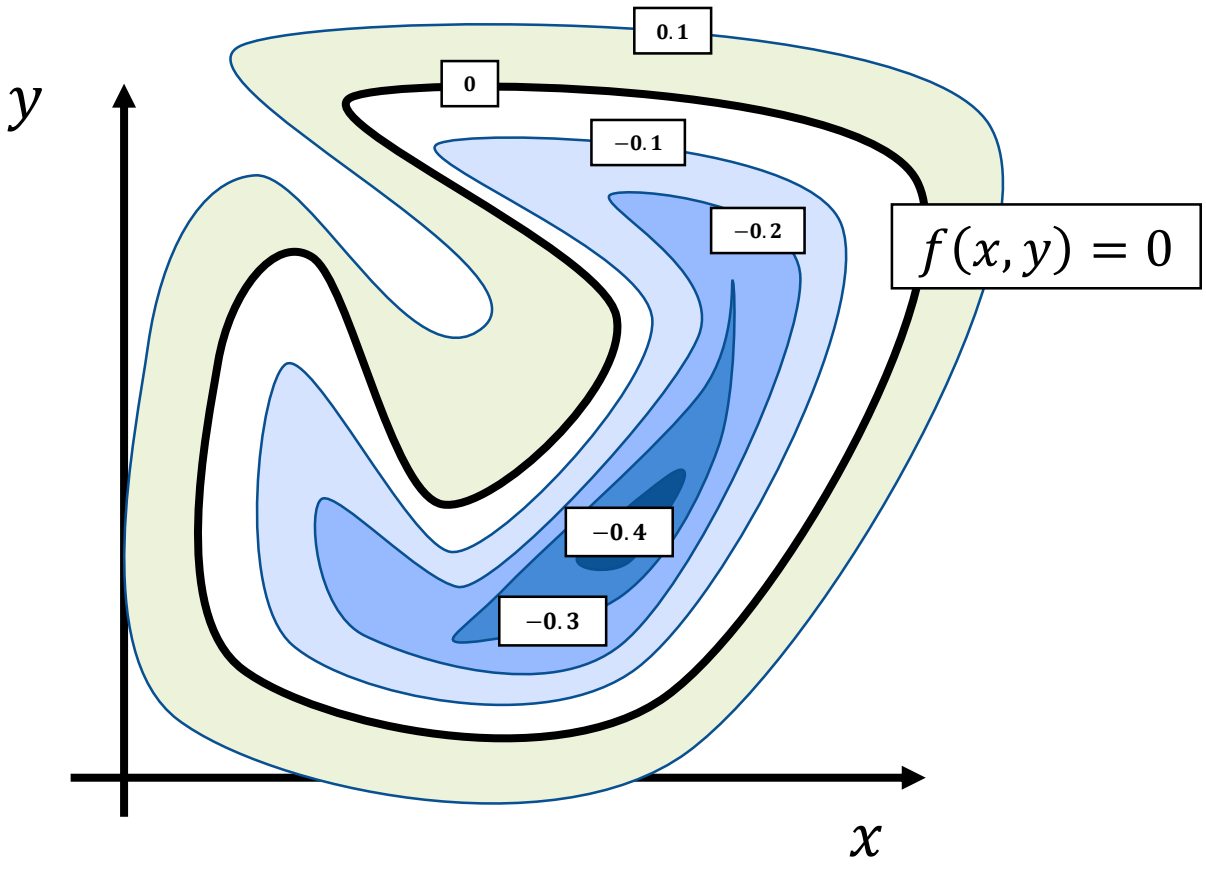
Parametric Surface



Implicit Surface

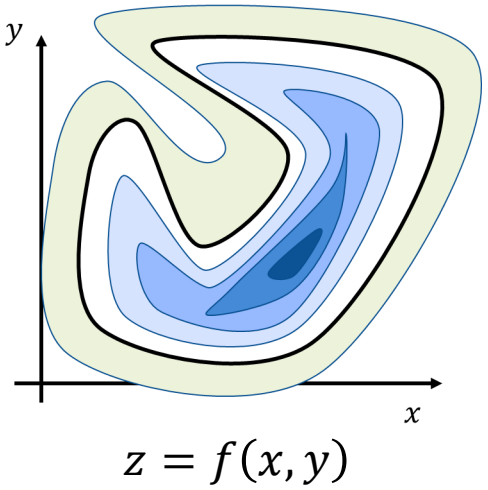


Implicit Surface



$$z = f(x, y)$$

Implicit Surface



- General case
  - Non-zero gradient at zero crossing

- Signed implicit function
  - $\text{sign}(f)$: $\begin{cases} - & \text{inside} \\ + & \text{outside} \end{cases}$, or the other way round

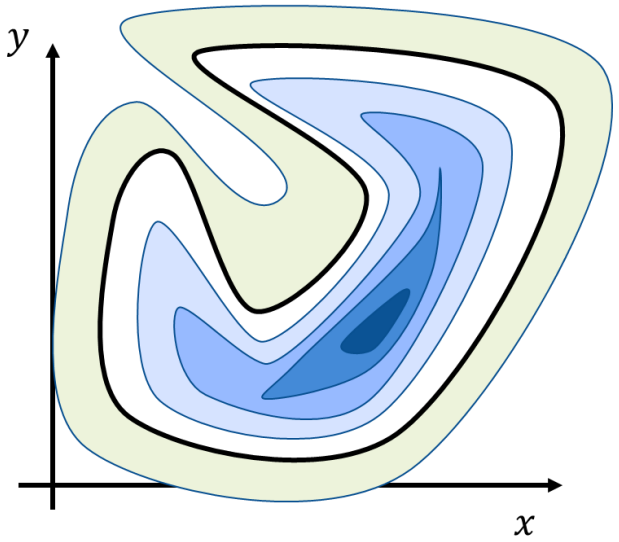
- Signed distance function (SDF)
  - $f = \pm$ distance to the surface

How to draw an implicit Surface?

- Given an implicit surface

$$f(x, y, z) = 0$$

Solve for (x, y, z) ?

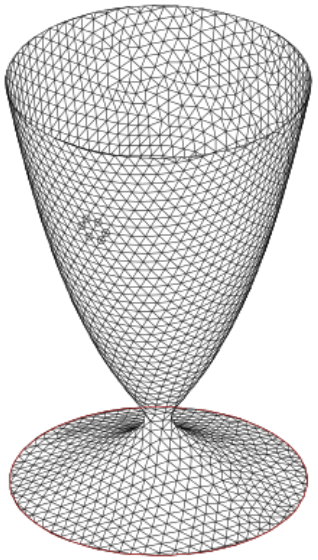


For example:

$$x + 2y - 3z + 1 = 0$$

$$x^2 + y^2 + (z - 3)^2 = 4$$

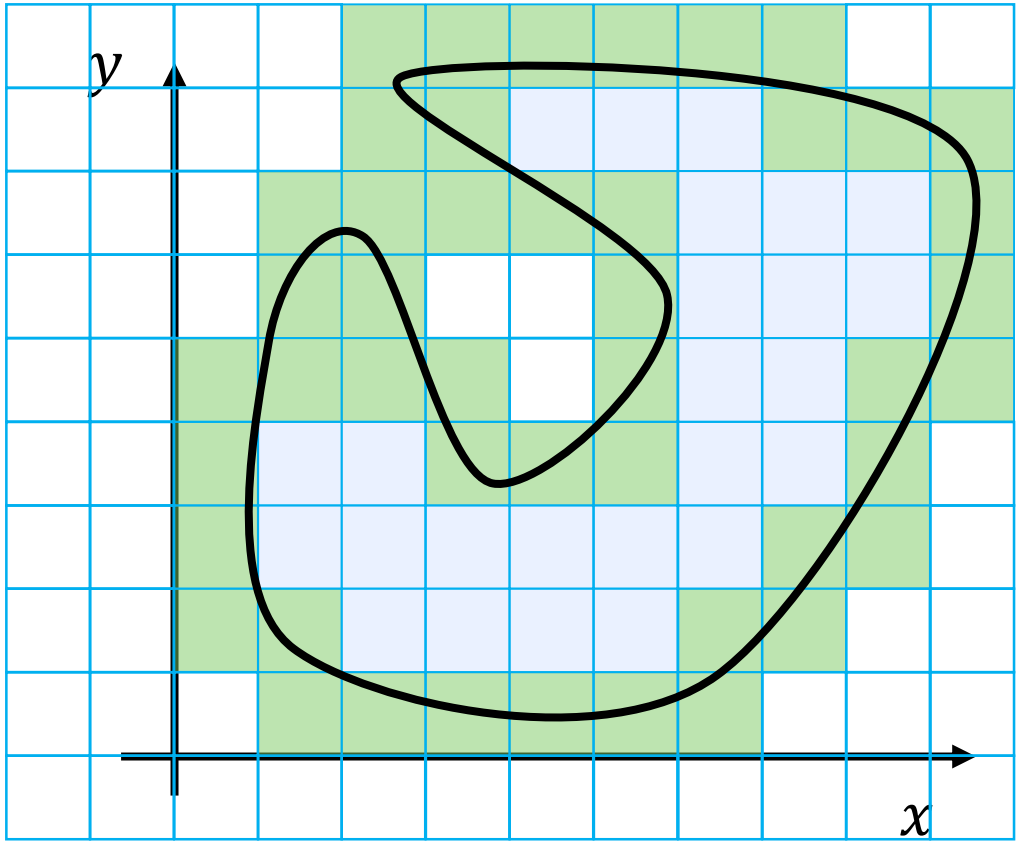
$$x^2 + y^2 - (\ln(z + 3.2))^2 - 0.02 = 0$$



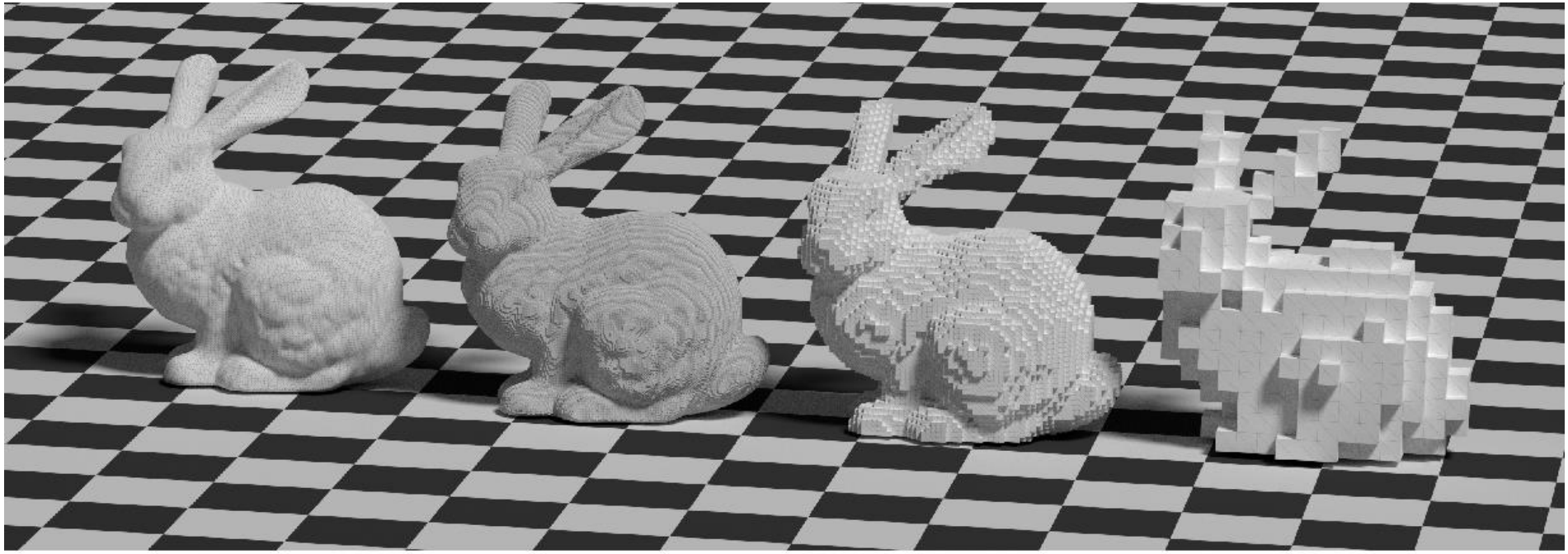
https://en.wikipedia.org/wiki/Implicit_surface

Voxel

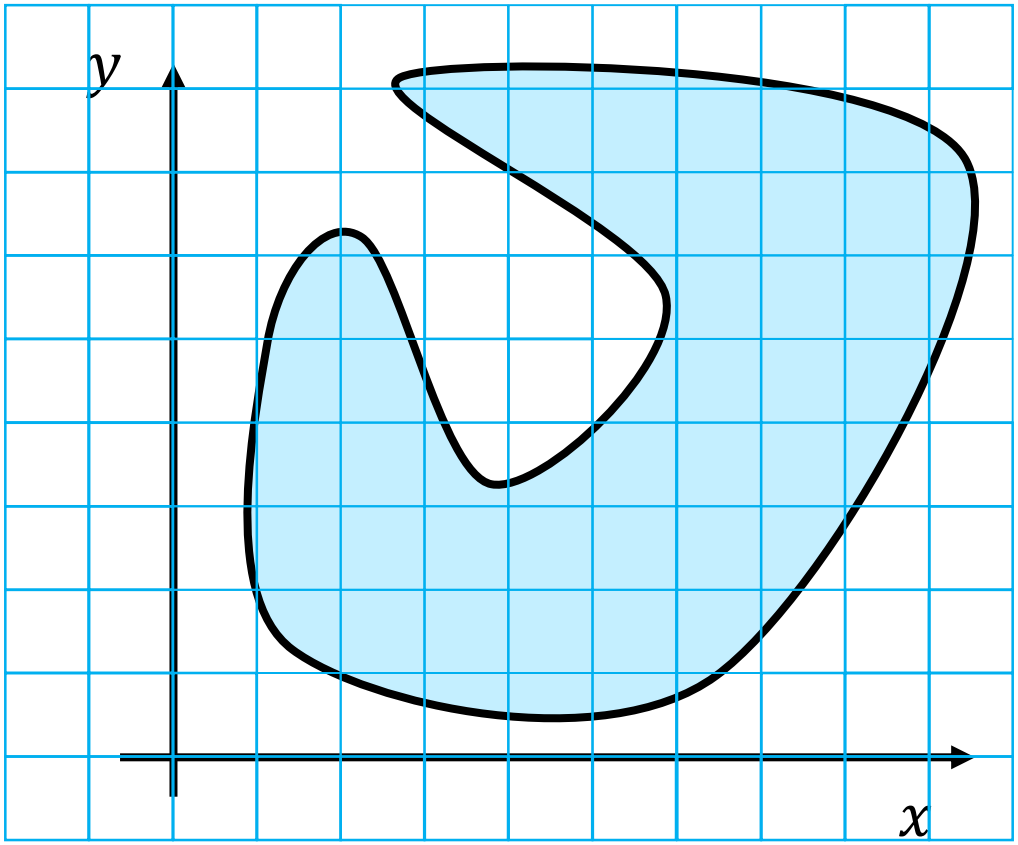
- Discretize the space into regular units
- Fill occupied units



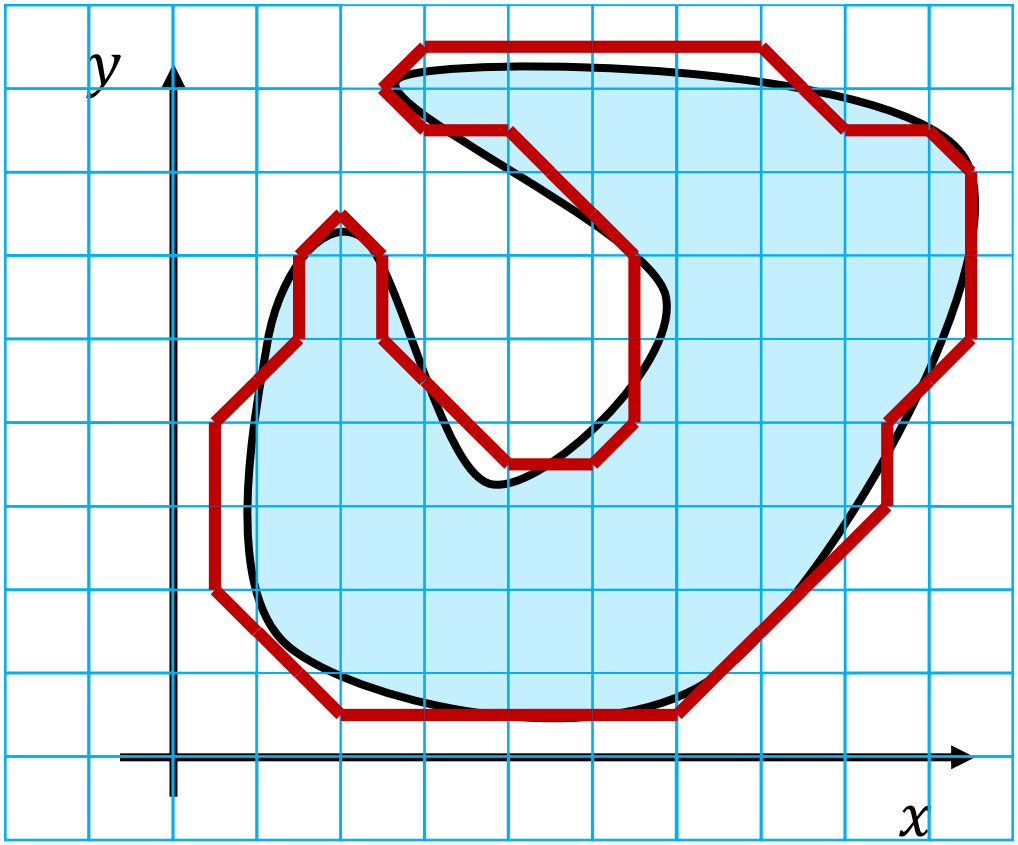
Voxel



Marching Cubes

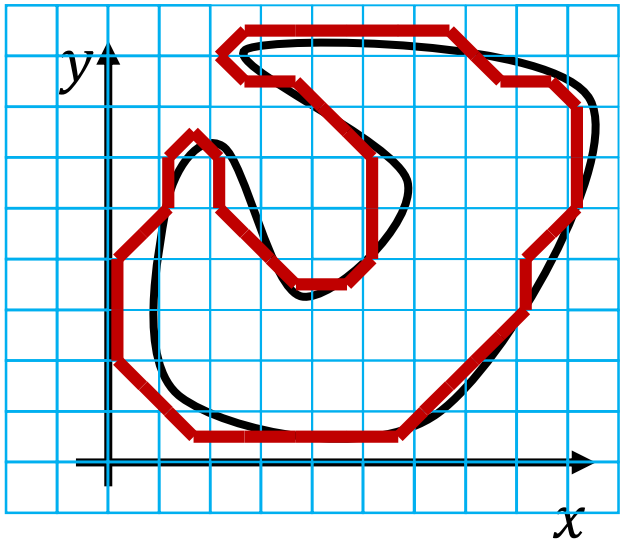


Marching Cubes



Marching Cubes

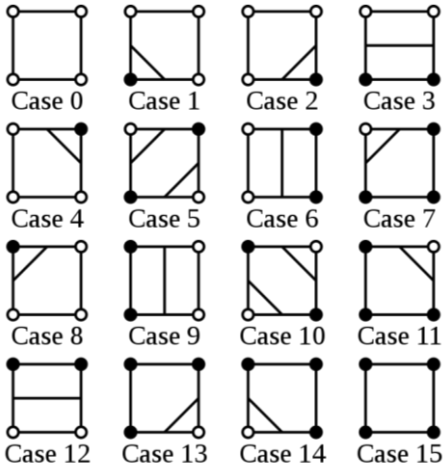
- One of the most commonly used method
- Idea:
  - Evaluate the function at grid points
  - Create edges/faces based on the patterns of the grid values.



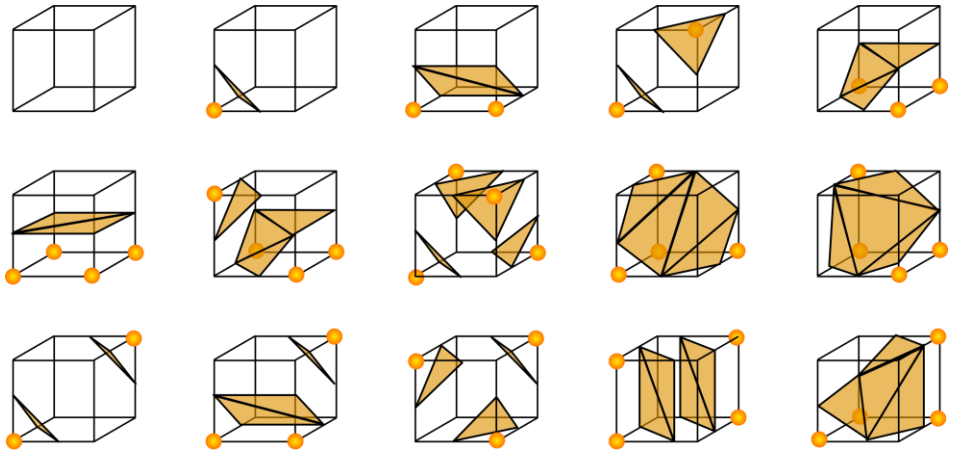
https://en.wikipedia.org/wiki/Marching_squares

William E. Lorensen and Harvey E. Cline. 1987. ***Marching cubes: A high resolution 3D surface construction algorithm***. In Proceedings of the 14th annual conference on Computer graphics and interactive techniques (SIGGRAPH '87).

Marching Cubes [Siggraph1987]



2D Marching Squares



3D Marching Cubes

https://en.wikipedia.org/wiki/Marching_squares

William E. Lorensen and Harvey E. Cline. 1987. ***Marching cubes: A high resolution 3D surface construction algorithm***. In Proceedings of the 14th annual conference on Computer graphics and interactive techniques (SIGGRAPH '87).

主要内容

- 几何的定义
- 几何表示
  - 点云 (point cloud)
  - 多边形网格 (polygon mesh)
  - 参数曲面
    - 贝塞尔曲线 Bézier Curve
    - B样条曲面与NURBS
  - 细分网格 (Subdivision Surface)
  - 隐式曲面
    - 绘制：体素/marching cube
- 参考：GAMES-102 (https://games-cn.org/games102/)

* Thank Prof. Baoquan Chen for many of the slides

Questions?