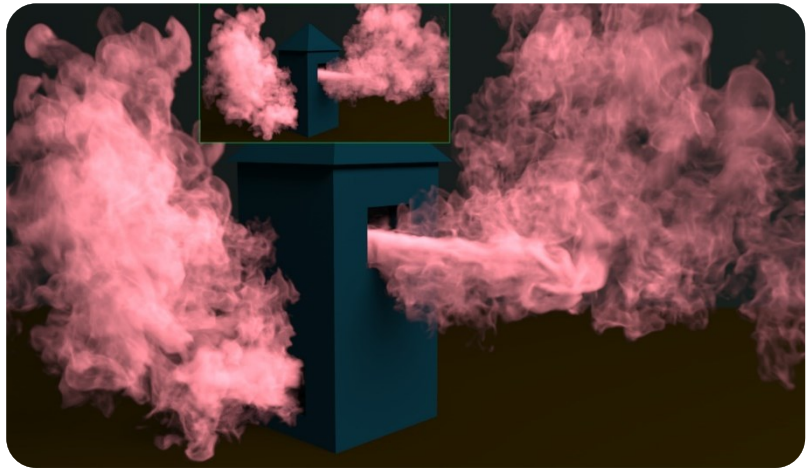
Curves

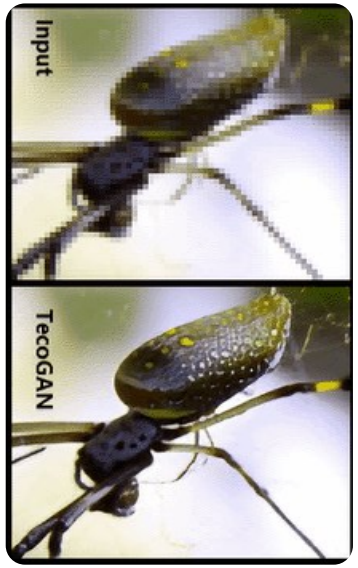
自我介绍

楚梦渝
智能学院 助理教授
mchu@pku.edu.cn
研究领域：计算机图形学，物理仿真，深度学习
个人主页：https://rachelcmy.github.io/

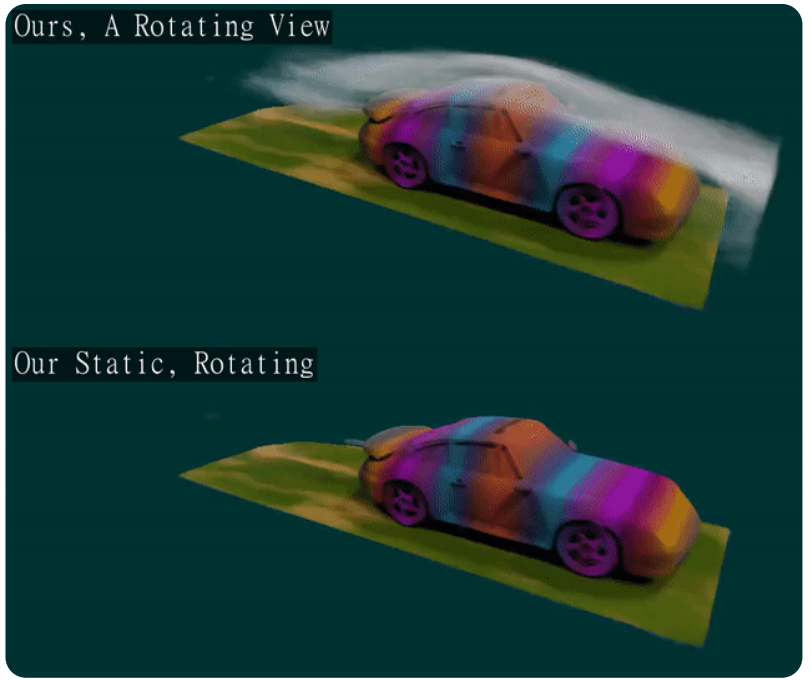
研究方向： 图形学物理仿真、物理增强深度学习
研究内容： 融合物理仿真和深度学习，为动态物理现象研究物理准确的深度学习方法，实现基于物理的动态内容的高保真生成、易用且准确的编辑重建等工作。

高分辨率的流体、视频等动态内容的快速生成

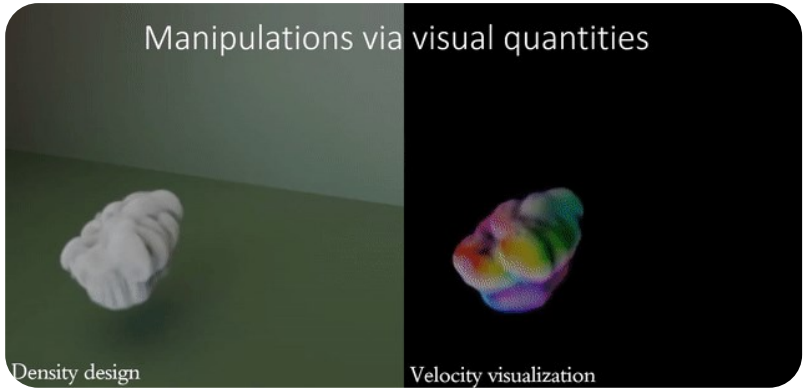


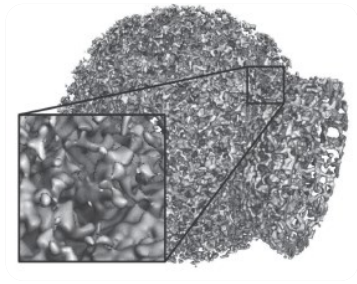


基于稀疏视频的流体和场景重建



易用的用户编辑，直观的用户控制





课程安排

- 课程安排
  - 时间：周一 1-2节课、周二 5-6节课
  - 地点：理教 303
- 课程需求
  - 线性代数
  - 高等数学
  - C++程序设计基础
- 考核方式
  - 上课考勤： 5%
  - 课堂测试、区间测试（3次）： 35%
  - 课程项目（Labs）： 40%
  - 项目报告：20%
- 网页：http://vcl.pku.edu.cn/course/vci

课程内容规划（可能有些调整）：

主题	标题	日期，星期(周次)	课时
基础	引言	24-09-09, Mon (1/16)	2h
	颜色	24-09-10, Tue (1/16)	2h
	显示	24-09-14, Sat (1/16)	2h
	画图	24-09-23, Mon (3/16)	2h
	反走样	24-09-24, Tue (3/16)	2h
	曲线	24-09-30, Mon (4/16)	2h
图像处理	图像处理	24-10-08, Tue (5/16)	2h
几何	几何表示	24-10-14, Mon (6/16)	2h
	几何变换	24-10-15, Tue (6/16)	2h
	几何处理	24-10-21, Mon (7/16)	2h
	隐式几何	24-10-22, Tue (7/16)	2h
	几何重建	24-10-28, Mon (8/16)	2h
渲染	光照和着色	24-10-29, Tue (8/16)	2h
	渲染管线	24-11-04, Mon (9/16)	2h
	纹理映射	24-11-05, Tue (9/16)	2h
	全局光照	24-11-11, Mon (10/16)	2h
	高级渲染	24-11-12, Tue (10/16)	2h

主题	标题		课时
仿真	物理模拟	24-11-18, Mon (11/16)	2h
	动画原理	24-11-19, Tue (11/16)	2h
可视化	可视化基础	24-11-25, Mon (12/16)	2h
	科学数据可视化	24-11-26, Tue (12/16)	2h
	信息数据可视化	24-12-02, Mon (13/16)	2h
	交互式可视分析	24-12-03, Tue (13/16)	2h
交互	经典交互技术	24-12-09, Mon (14/16)	2h
	三维空间交互	24-12-10, Tue (14/16)	2h
	智能交互	24-12-16, Mon (15/16)	2h
	虚拟/增强现实	24-12-17, Tue (15/16)	2h
	自然交互	24-12-23, Mon (16/16)	2h
课程测试	**课程测试**	24-12-24, Tue (16/16)	2h

Lab 1

- 提交 task.cpp，报告 pdf

可视计算与交互概论 Tutorial for Lab 1 (2D Drawing)

Lab 1 Overview

这次 lab 中，大家将会实现前七章中介绍的几种重要的算法或思想：

1. 图像处理 中的 Dithering 算法（第七章讲义 7.2节）
2. 图像处理 中的 Image Filter 算法（第七章讲义 7.3节）
3. 图像处理 中的 Poisson Editing 算法（第七章讲义 7.4节）
4. 画图 中的 Bresenham 直线算法（第四章讲义 4.2节）
5. 画图 中的 Scan Converting 三角形算法（第四章讲义 4.3节）
6. 反走样 中的 Supersampling 算法（第五章讲义 5.3节）
7. 曲线 中的 de Casteljau 算法（第六章讲义 6.2节）

Lab 1 的任务比较多，但每个任务都不是很难。大家的任务是填补 src/VCX/Labs/1-Drawing2D/tasks.cpp 中的空缺部分，每一个函数对应一个任务。请务必**独立**完成自己的代码。

Lab 1

- https://gitee.com/pku-vcl/vcx2024/blob/lab1/src/VCX/Labs/1-Drawing2D/tasks.cpp

Curves

Curve

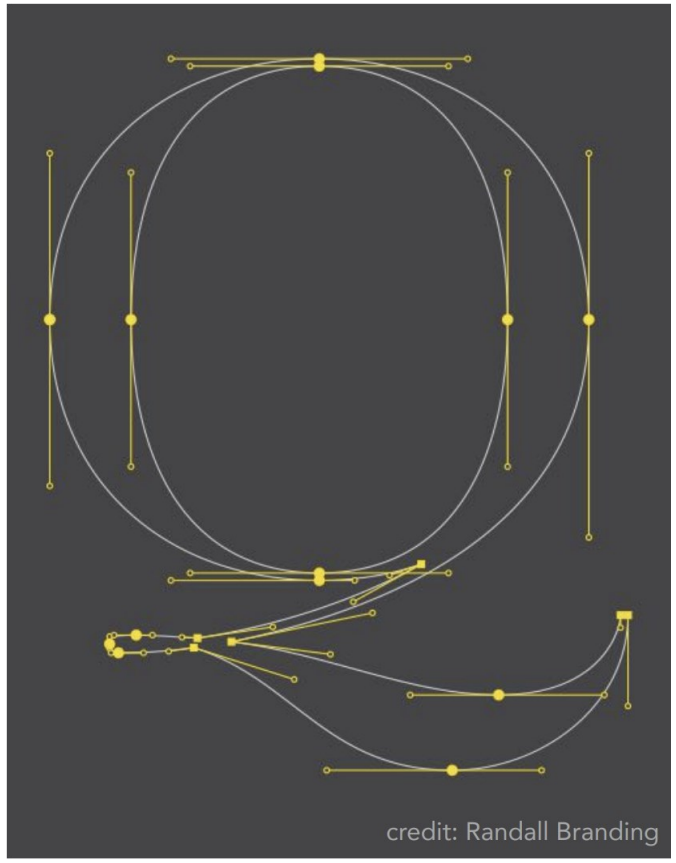
The Starry Night	
	
Artist	Vincent van Gogh
Year	1889
Medium	Oil on canvas
Dimensions	73.7 cm × 92.1 cm
Location	Museum of Modern Art, New York

《千里江山图》	
	
Artist	王希孟
Year	北宋 公元1113年4月
Medium	中国青绿山水画 绢本 矿物质颜料绘制
Dimensions	纵51.5厘米，横1188厘米
Location	故宫博物院 https://arts.cctv.com/gallery/web90/index.shtml

Curve

The S	
	
Artist	Vincen
Year	1889
Medium	Oil on
Dimensions	73.7 cr
Location	Museu

2D Curve Example



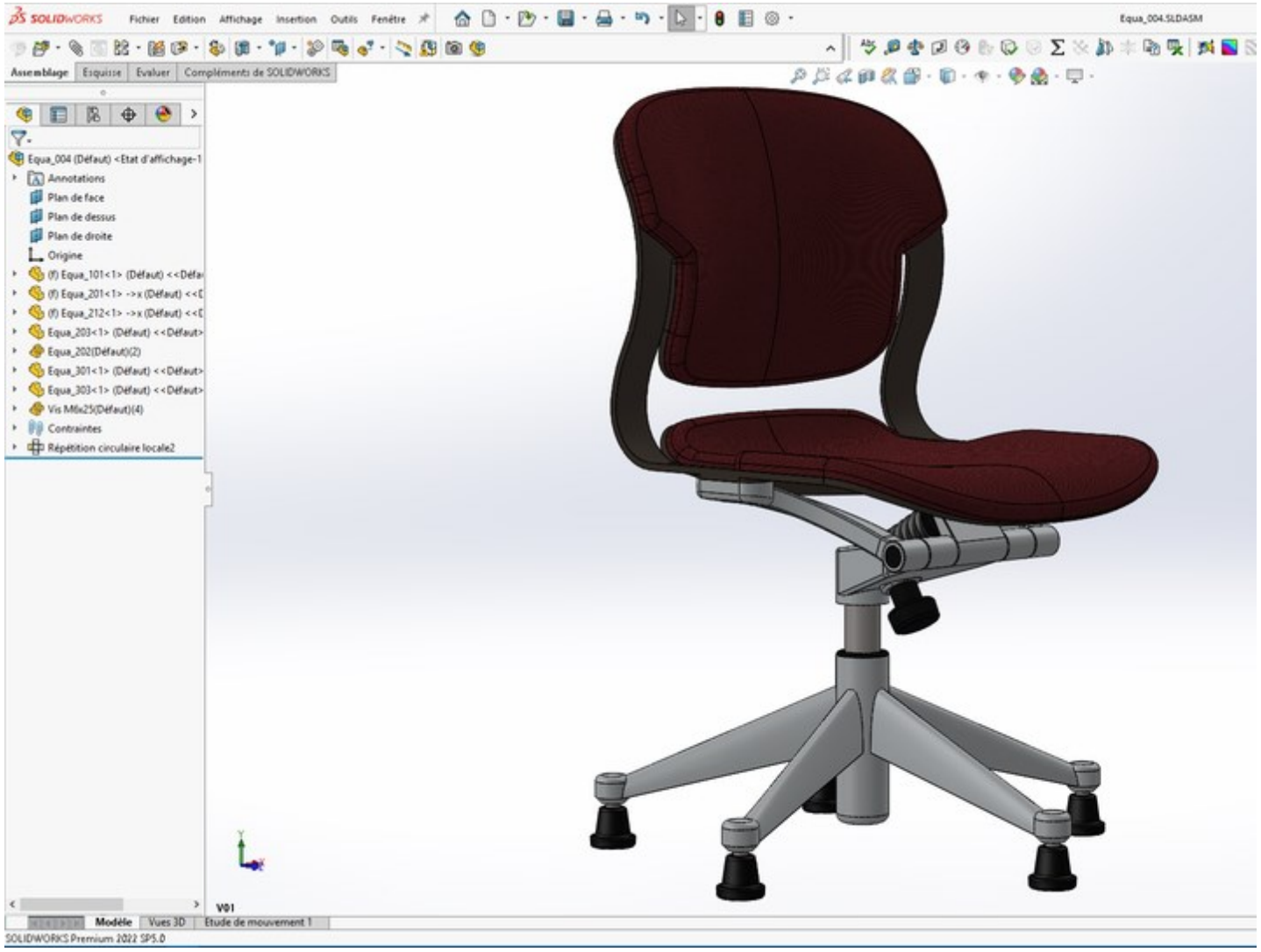
Font

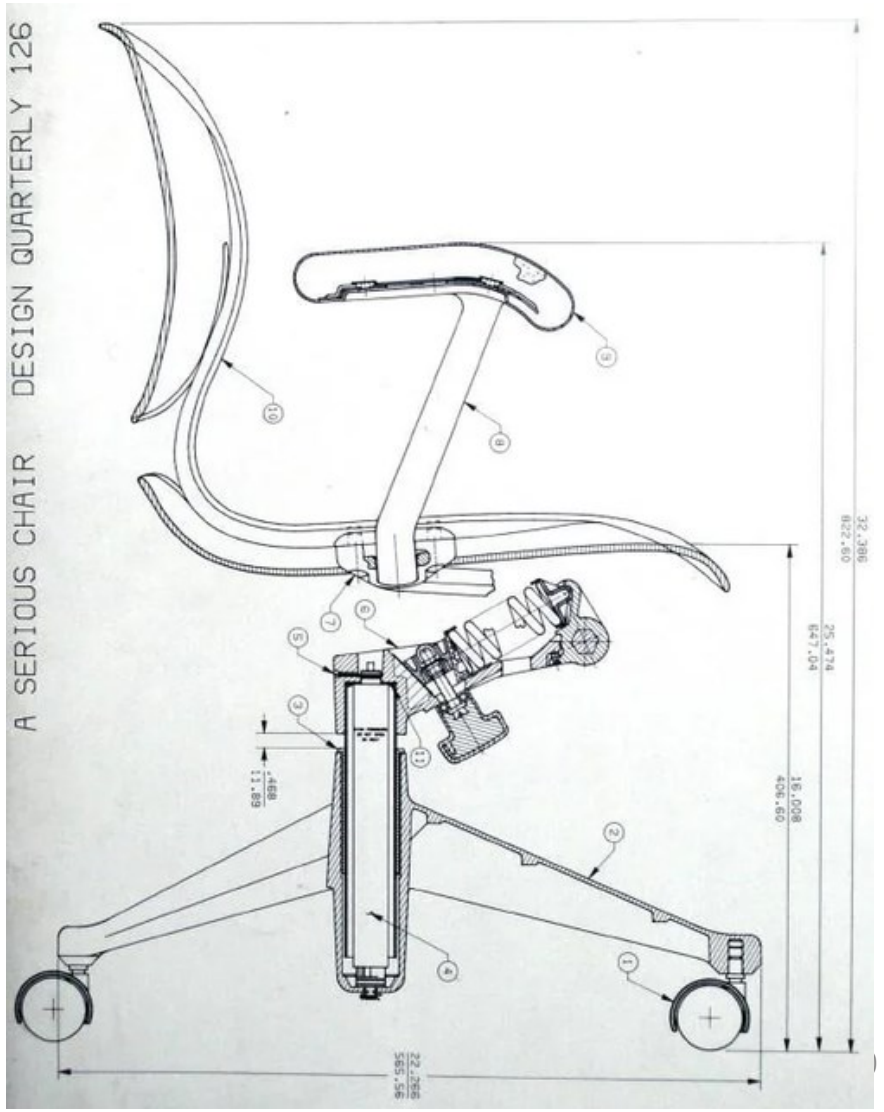
SVG (Scalable Vector Graphics)

SVG for logos

3D Curved Surfaces and Volumes

https://grabcad.com/ **Software used**: Solidworks, Autocad,… **File**: .step, .sldprt, .igs, .x_b (Parasolid)

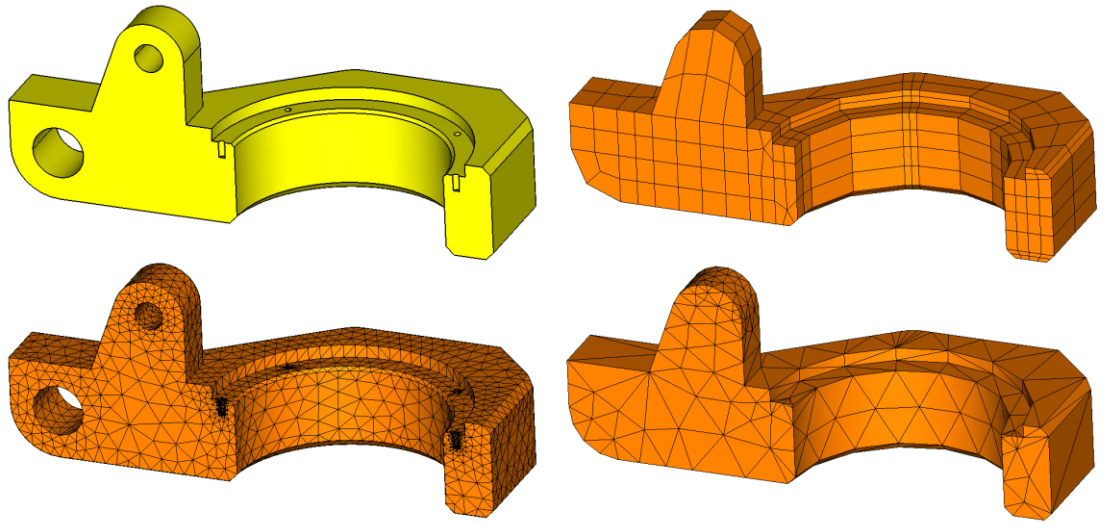


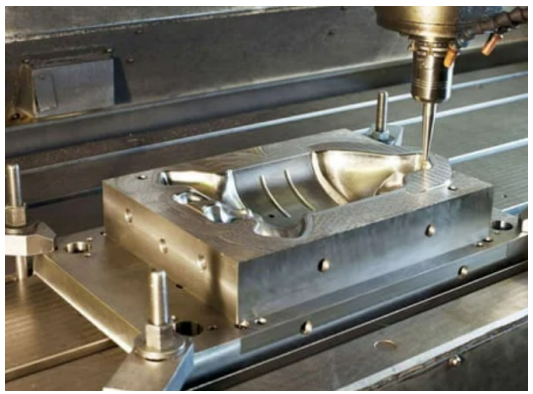


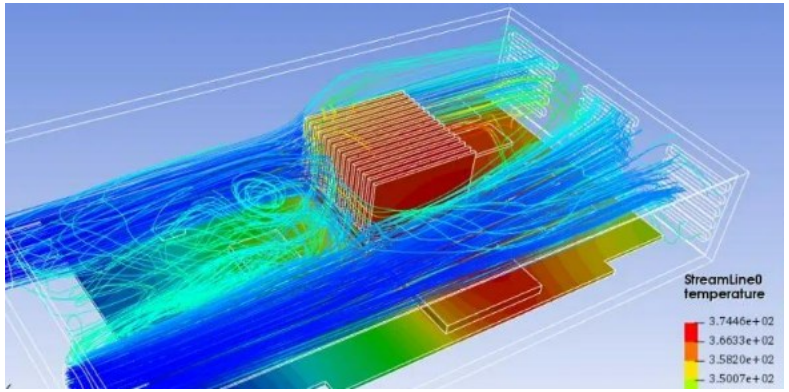
CAD/CAM/CAE

- Computer Aided Design/ Manufacturing/ Engineering

  - ✓ **CAD**: Curve-based, continuous.
  - ✓ **CAE**: Usually involves linear approximations for analysis.
  - ✓ **CAM**: Utilizes both continuous CAD data and linear approximations for toolpaths.



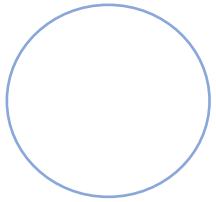




Representations of 2D curves

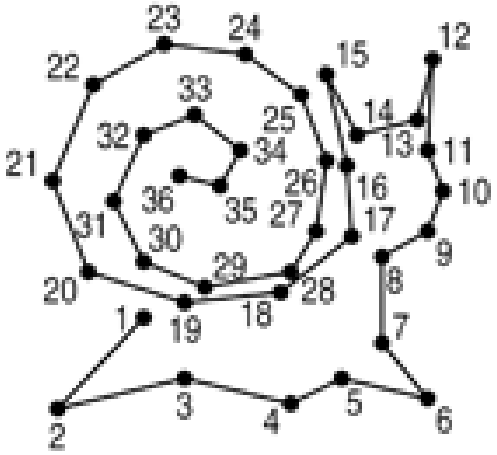
Represents of 2d curves

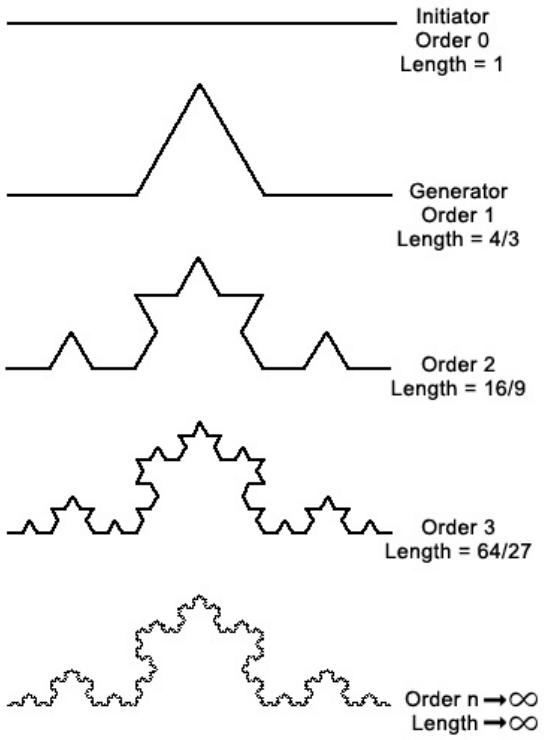
- There are many curve representations

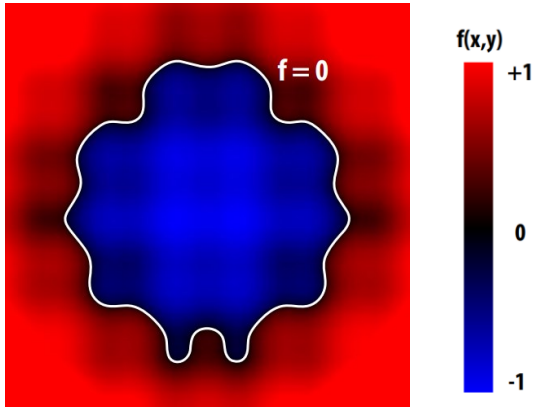


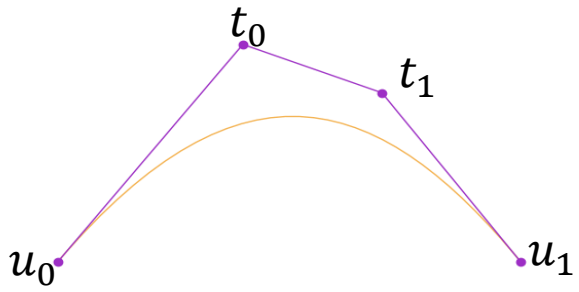
$$x^2 + y^2 = r^2$$

$$x = r\cos\theta$$
$$y = r\sin\theta$$







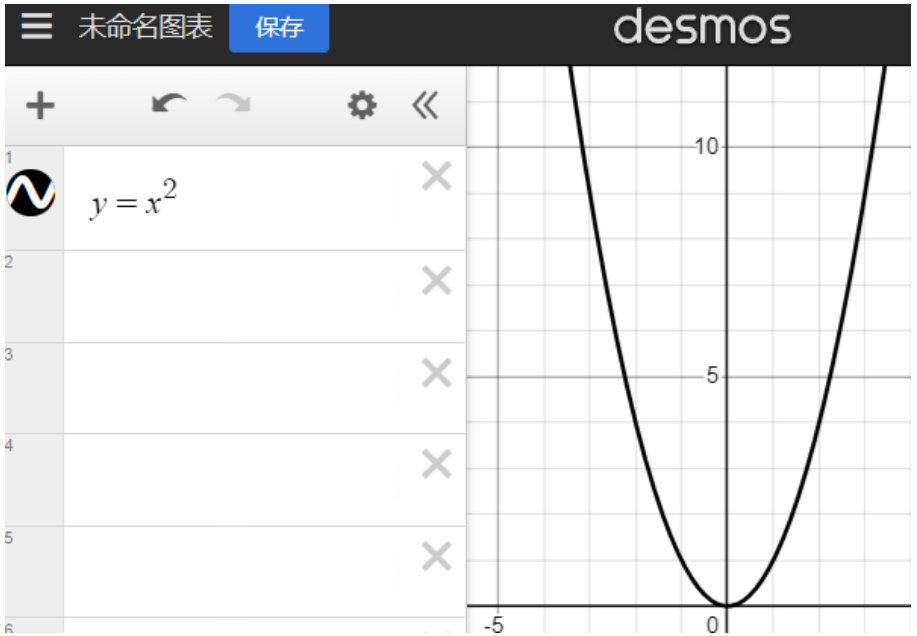


Represents of 2d curves

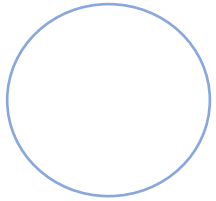
Explicit

You can directly get points on the curve

- A sequence of points
  - Formula: $y = mx + b$



- Generally, $y = f(x)$



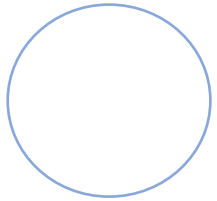
How about a circile?
Do we have a $y = f(x)$?

Represents of 2d curves

Implicit

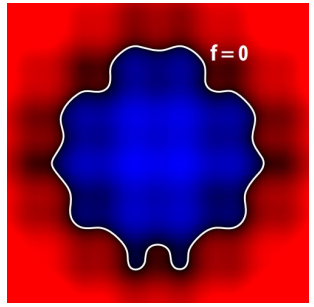
Points aren’t known directly, but satisfy some relationship

- Analytical equation

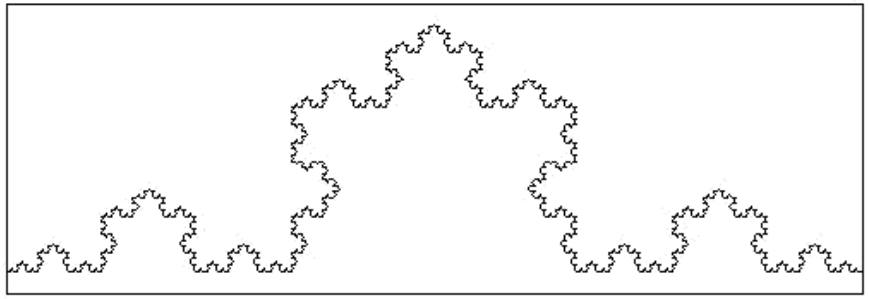


$$x^2 + y^2 = r^2$$

- Signed distance functions



- Fractal



- ……

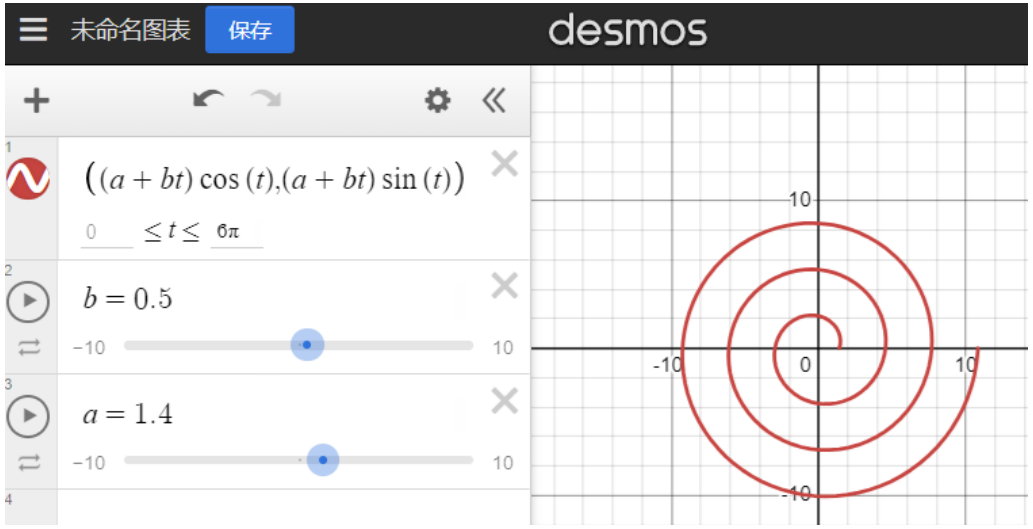
Represents of 2d curves

Parametric

- Parametric function
  - E. g. , a spiral can be expressed as:

$$x(t) = (a + bt)cos(t)$$

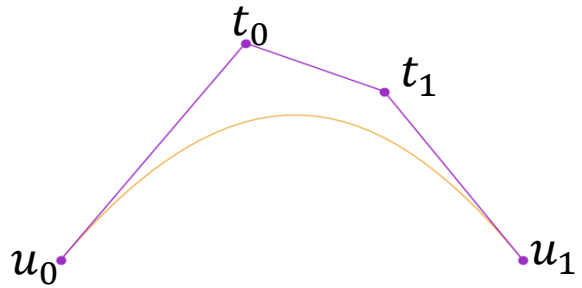
$$y(t) = (a + bt)\ sin(t)$$



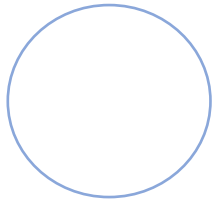
  - Bezier curve (贝塞尔曲线)

- Generally:

$$(x,y)=(\ f(t),\ g(t)\ )$$

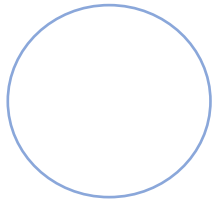


Analytical equation (implicit)



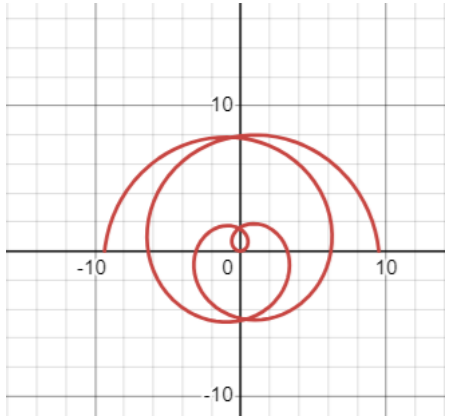
$$x^2 + y^2 = r^2$$

Parametric function (explicit)



$$x(t) = \mathrm{r}\cos t$$

$$y(t) = \mathrm{r}\sin t$$



desmos

https://help.desmos.com/hc/en-us/articles/4406906208397-Parametric-Equations

desmos HELP CENTER DESMOS HOMEPAGE

Search...

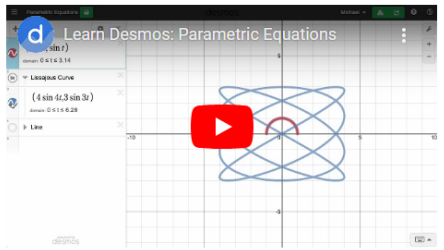
Desmos Help Center › Advanced Features › General

Parametric Equations

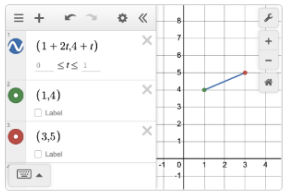
Updated 1 month ago

Graphing parametric equations on the Desmos Graphing Calculator, Geometry Tool, or the 3D Calculator Instead of numerical coordinates, use expressions in terms of the special parameter t, like ($\cos t$, $\sin t$). In the 3D Calculator, you can also use expressions in terms of the parameters u and v like ($\cos u$, $\sin v$, u).

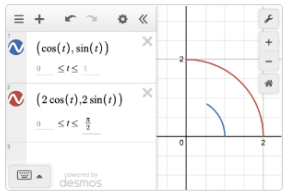
Get started exploring parametrics with the video on the right, then dive deeper with the resources below.



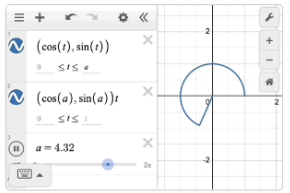
Getting Started



To graph a parametric curve, simply create an ordered pair where one or both coordinates are defined in terms of the parameter t.

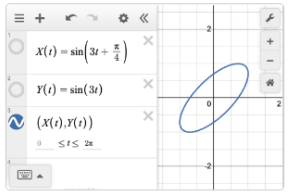


By default, parametric curves are plotted for values of t in the interval $[0, 1]$. However, you can manually adjust the domain using the inputs provided beneath the expression.



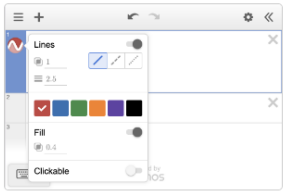
Just like other expressions, you can use dynamic variables in parametric equations, including in the bounds for the interval of t.

Parametric Curves

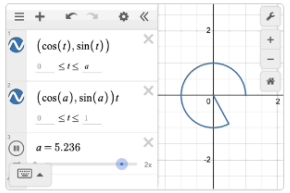


Parametric curves can be represented in a single coordinate expression or can reference other defined component functions using the parameter t.

Note: In Desmos, since lowercase x and y are reserved variables, you must use uppercase X and Y if you want to define functions for x and y.



It is also possible to fill a parametric curve by long-pressing the icon to the left of your expression which opens the options menu. From there, you can toggle on the *Fill* option to visualize the area enclosed by the parametric curve.



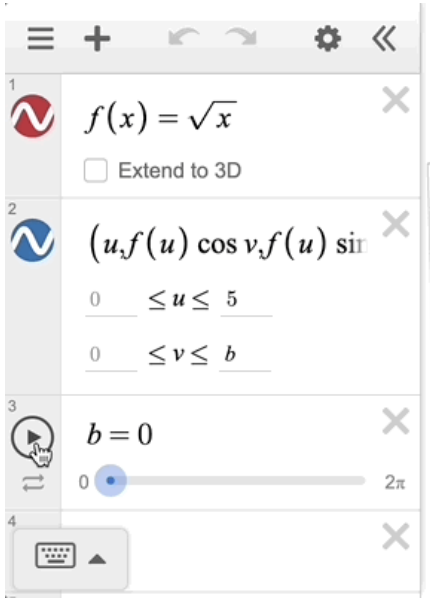
Note that when computing parametric fill in Desmos, a curve's interior is determined according to the non-zero winding rule. To explore this concept further, you can open any example in the article by clicking on the image.

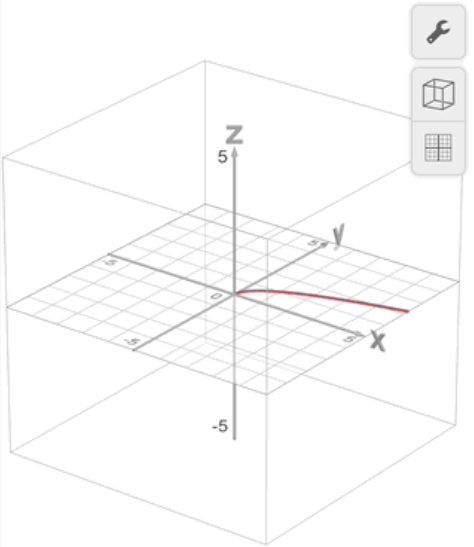
Parametrics in 3D

Graph a parametric surface in 3D with the parameters u and v.

Consider the parametric surface $(u, f(u)cosv, f(u)sinv)$ where the parameter u represents the x-values,and v represents the angle of rotation. This formula rotates any function $f(x)$ around the x-axis.

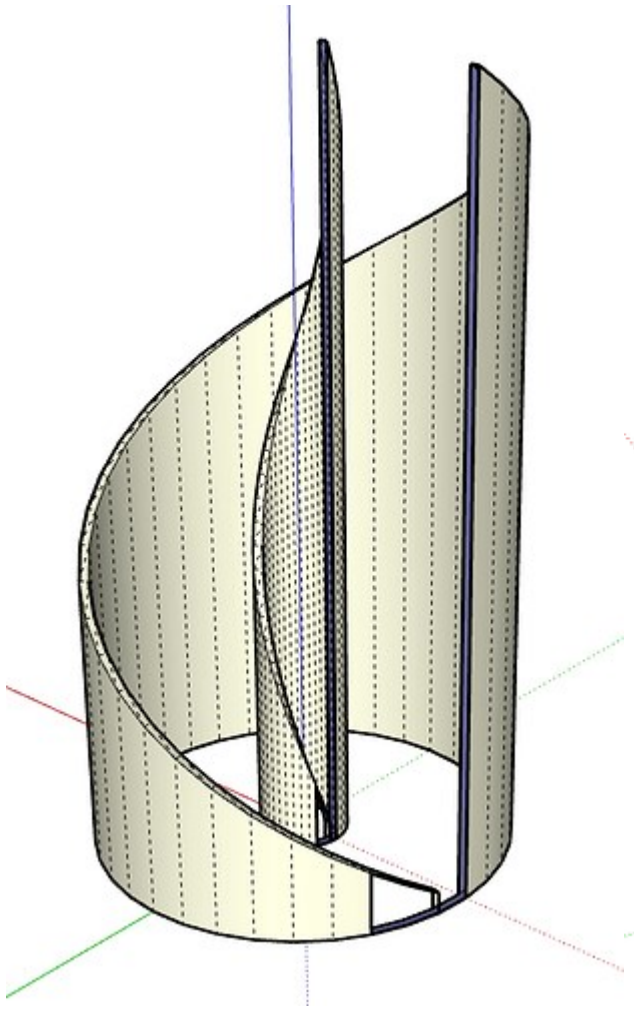
For instance, try rotating $f(x) = \sqrt{(x)}$ with this parametric surface. Adjust the inputs beneath the expression to $0 \le u \le 5$ and $0 \le v \le 2\pi$. To connect this rotation to a slider, create a slider b from 0 to 2π and adjust the input for v to $0 \le v \le b$.

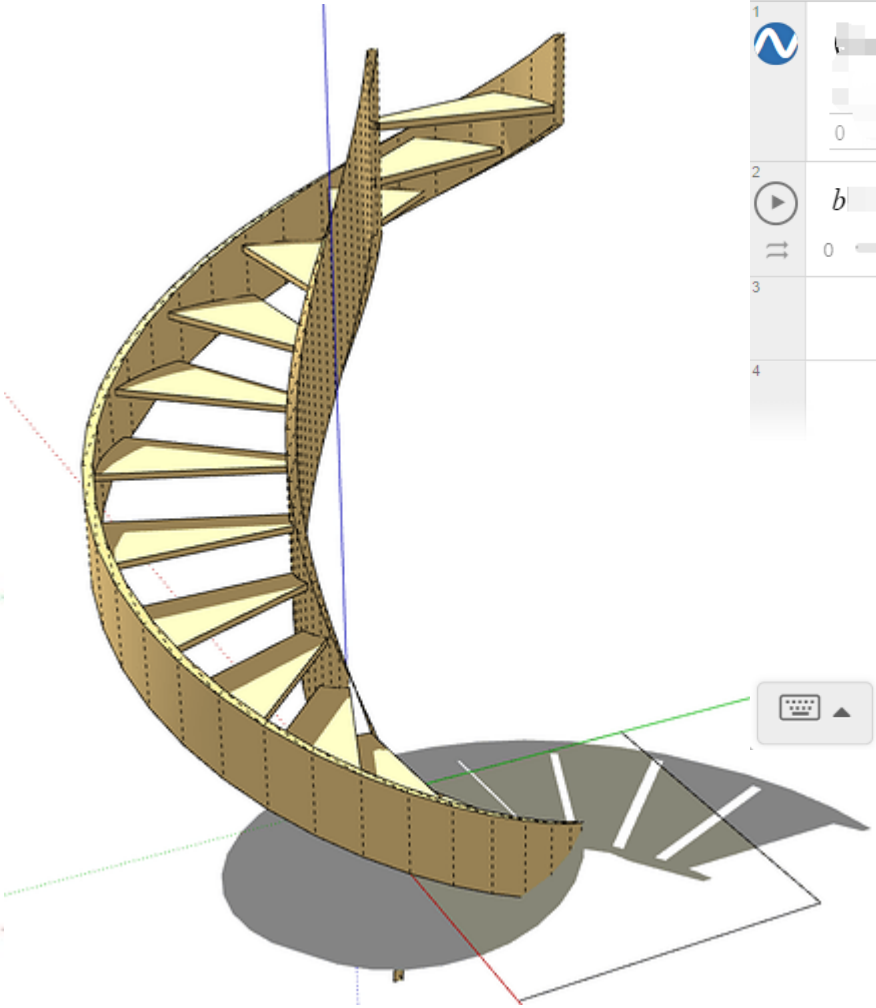


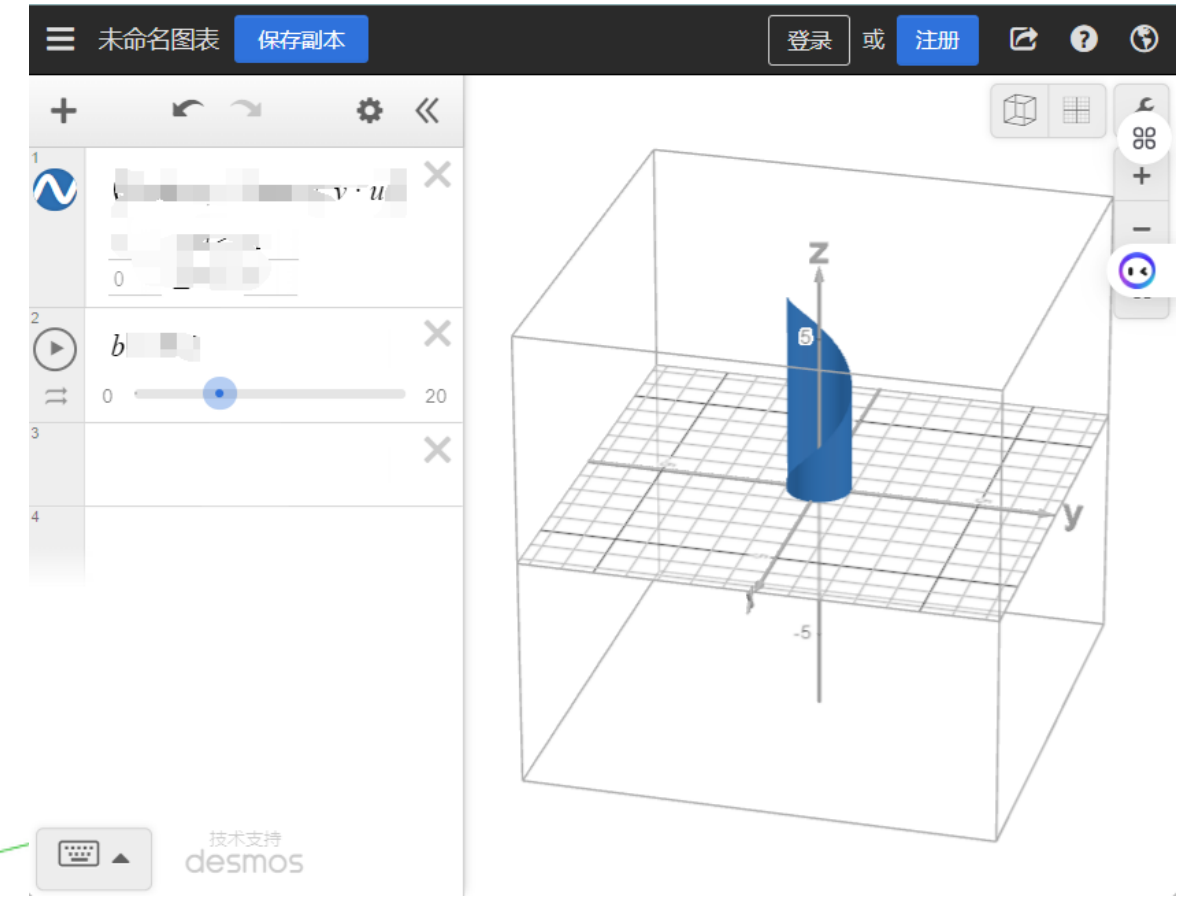


https://www.desmos.com/3d/av7u0ljluy?lang=zh-CN

desmos







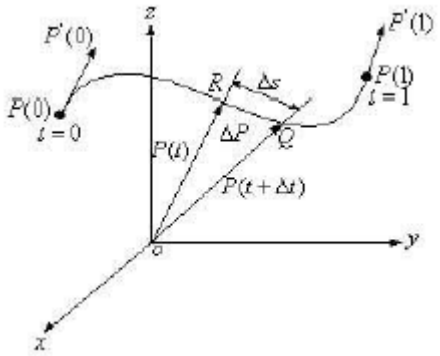
Advantages of parametric representation

- Extension to higher degree

$$\begin{aligned} x &= x(t) \\ y &= y(t) \end{aligned} \quad \Longleftrightarrow \quad \begin{aligned} x &= x(t) \\ y &= y(t) \\ z &= z(t) \end{aligned}$$

- Visualizing curve trajectory

$$P(t) = [x(t), y(t), z(t)]^T$$



- Computing derivatives

$$\frac{dP(t)}{dt} = \begin{bmatrix} \frac{dx(t)}{dt} \\ \frac{dy(t)}{dt} \\ \frac{dz(t)}{dt} \end{bmatrix}$$

The unit vector of the derivative vector is the tangent vector of the curve.

Parametric representation in polynomial form

- Parametric expression can have multiple forms

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

Any form that fulfills $x^2 + y^2 = r^2$

- The polynomial form is usually preferred

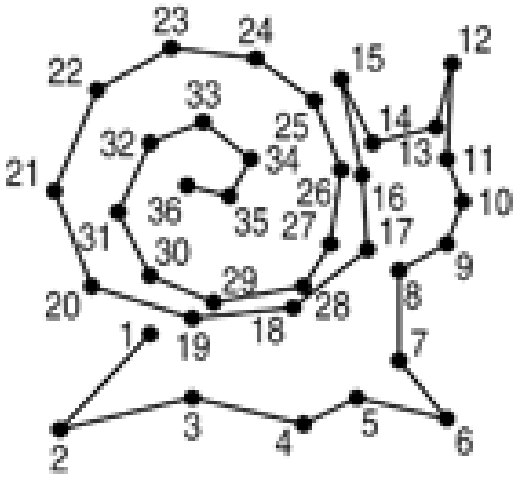
$$P(t) = at + b$$

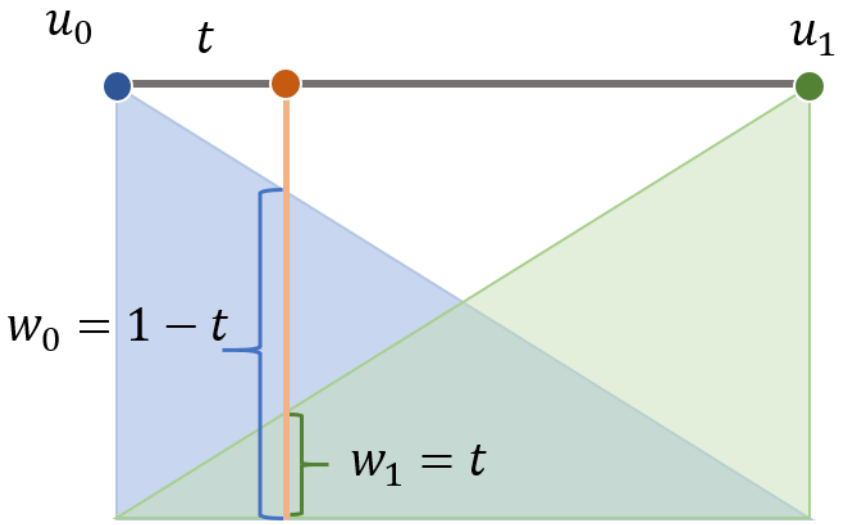
$$P(t) = at^3 + bt^2 + ct + d$$

Simple Parametric Examples

Linear interpolations
High-order polynomial approximation

Linear Interpolation



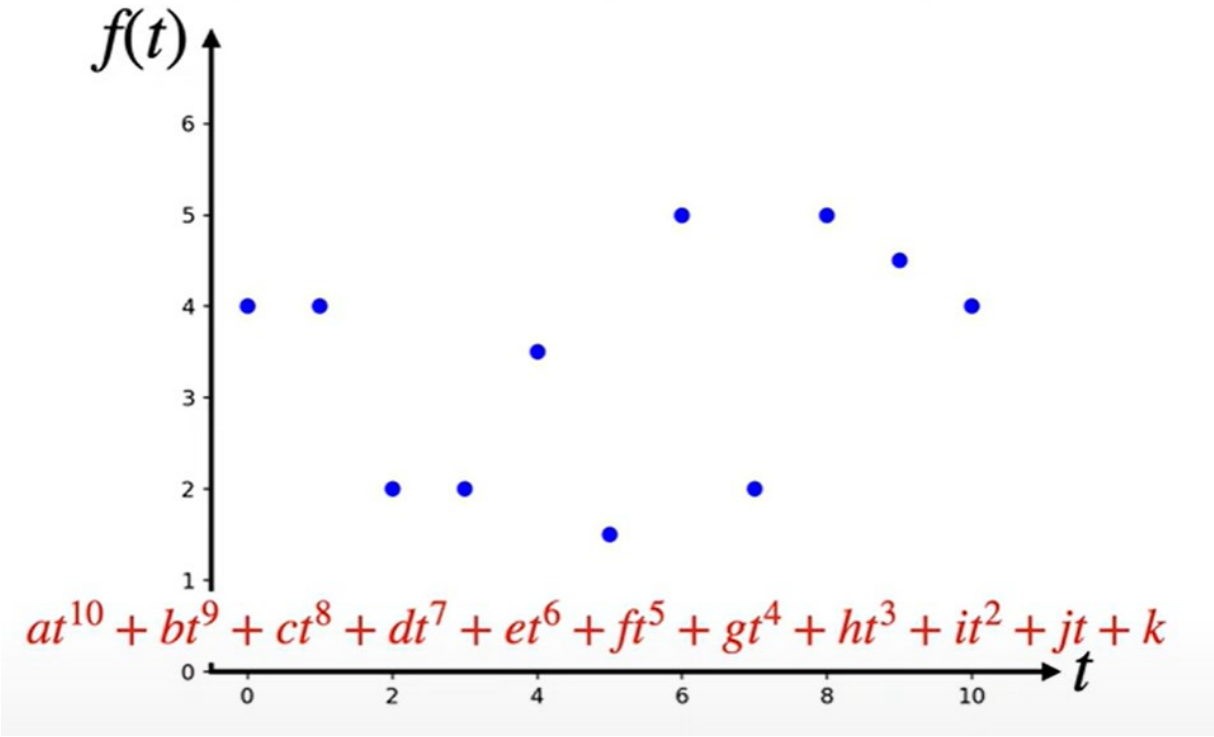


- A linear interpolation(lerp) draws a straight line between two points
- We can calculate the inbetweening by $lerp(t) = (1 - t)u_0 + tu_1$
- t is a relative distance on the line

Interpolation makes graphics!

High-order polynomial approximation

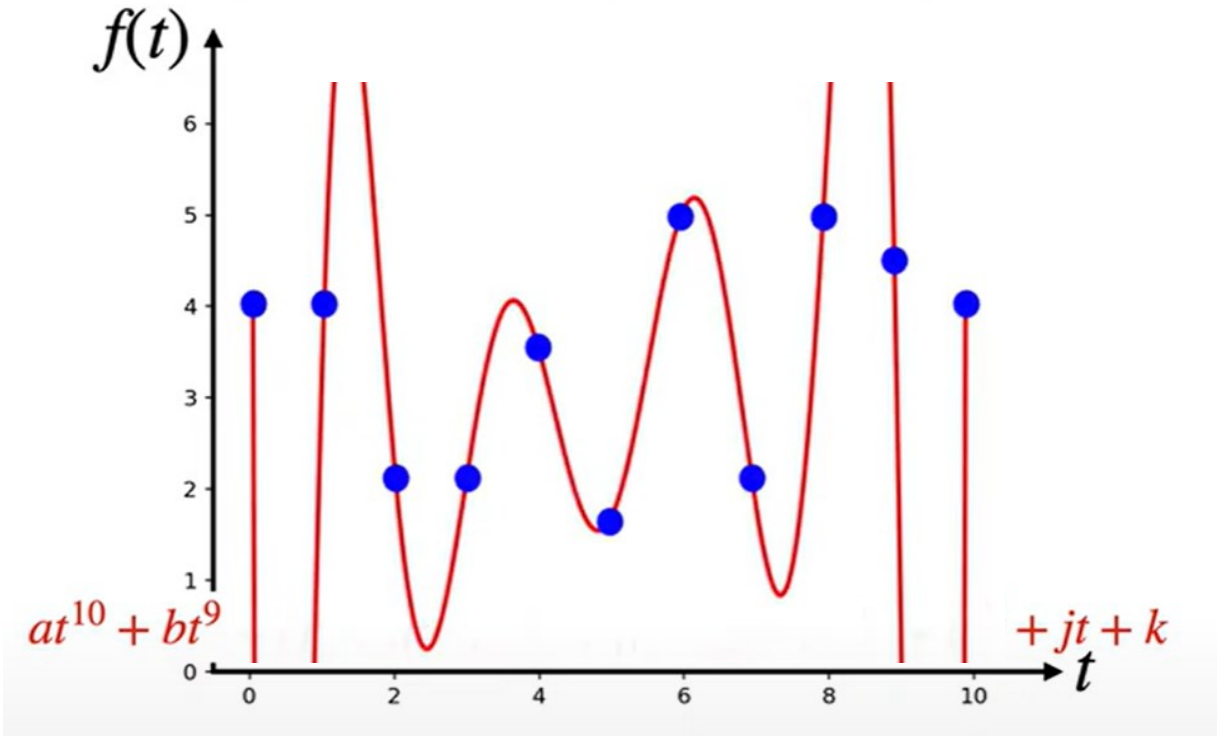
- Sample n points from the curve *you draw*
- Calculate the n parameters of a (n-1)-order polynomial curve

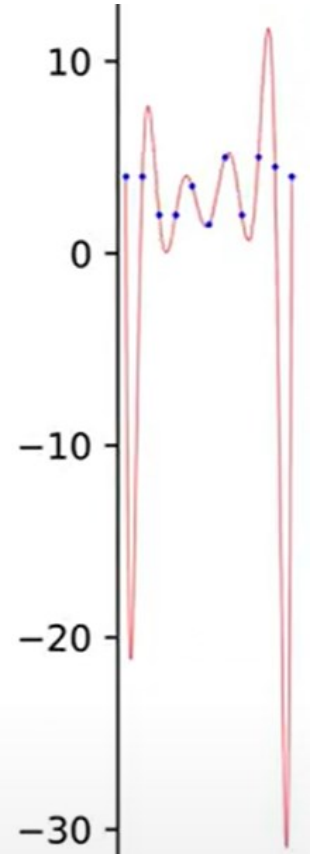


High-order polynomial approximation

- Sample n points from the curve *you draw*
- Calculate the n parameters of a (n-1)-order polynomial curve

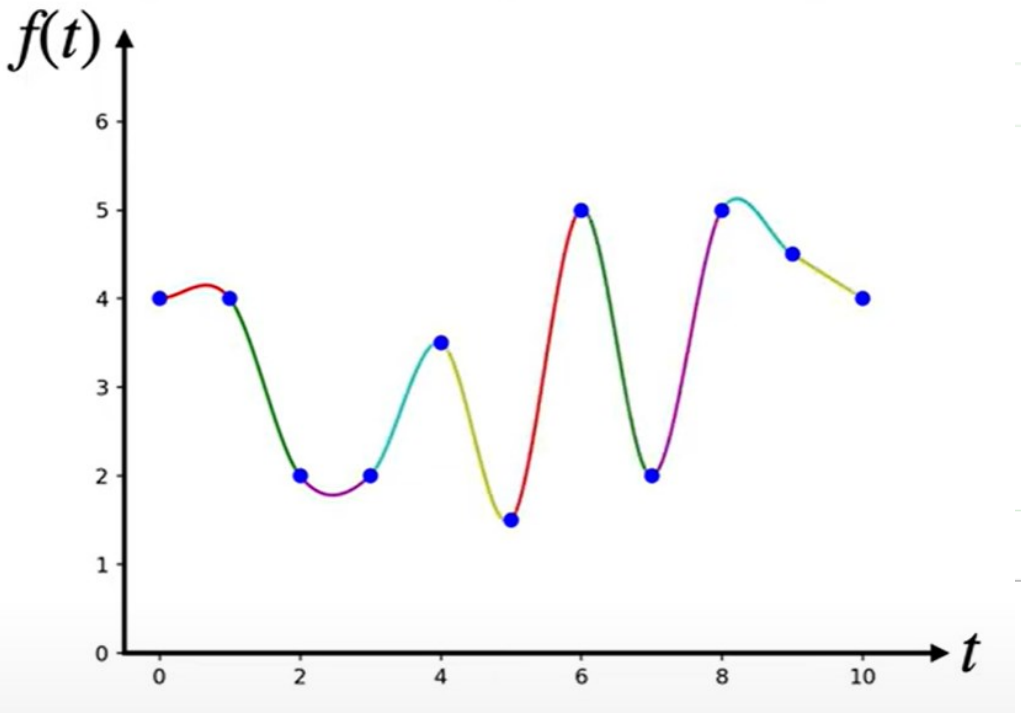
  It is difficult to represent a curve using a single polynomial function



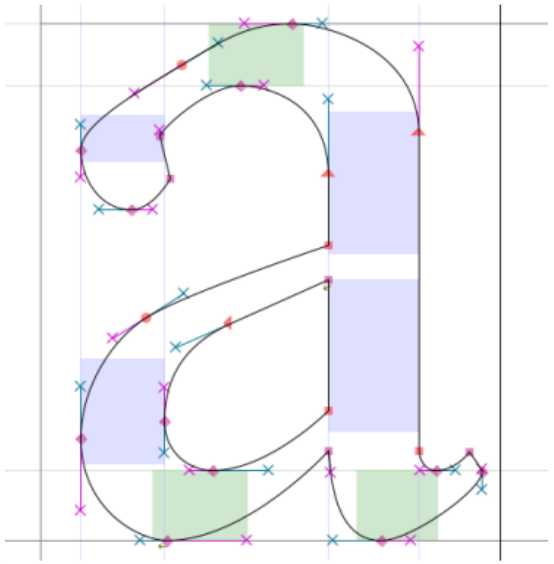


Piecewise functions

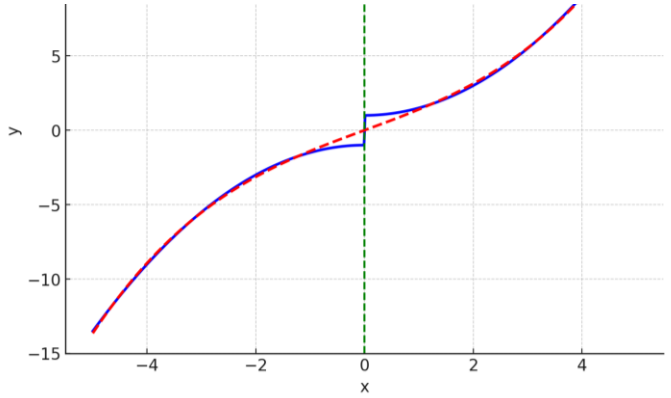
- We can use piecewise functions
- And they should be continuous

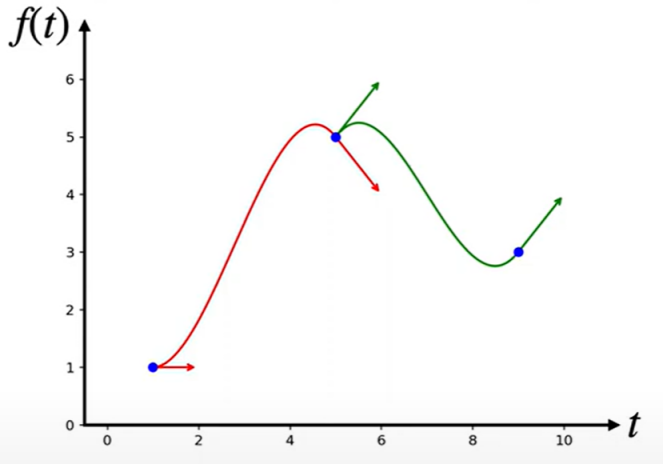


Spline Curve (Cubic)



Piecewise functions of Bezier Curve and Straight Lines

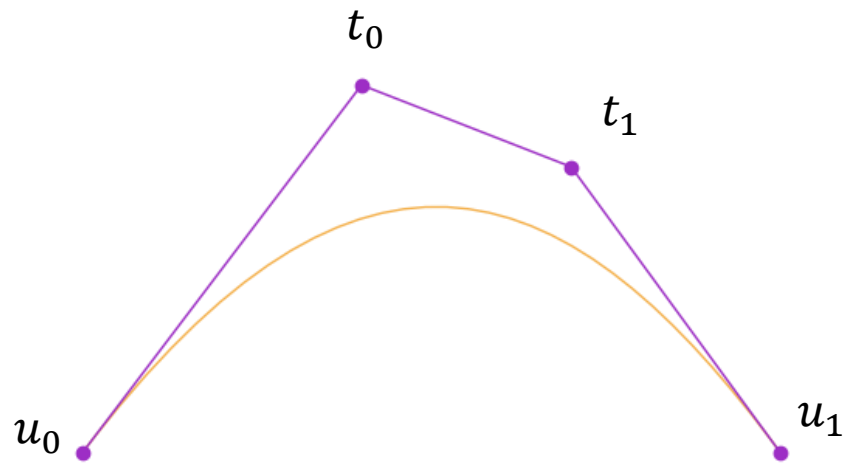




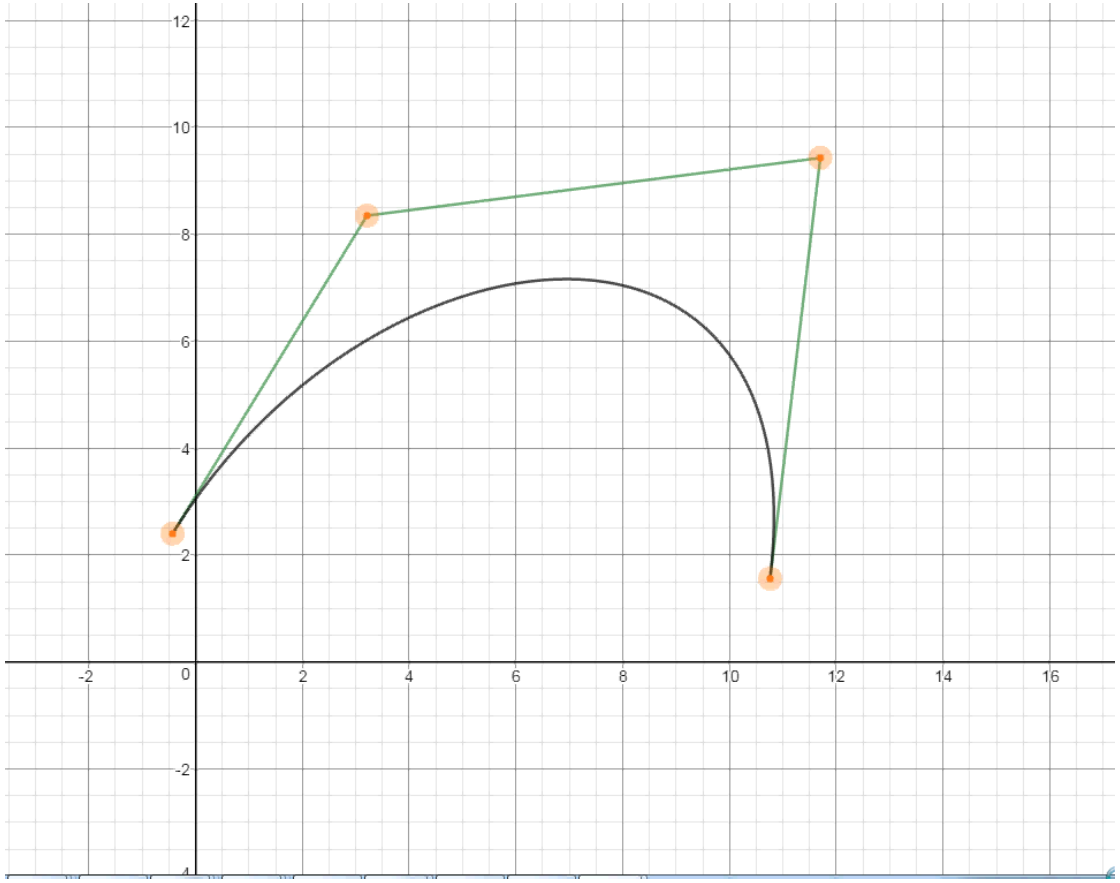
Popular Parametric Curves: Bezier Curve

Bezier Curve

In 1962, Pierre Bezier, an engineer from Renault Car company (法国雷诺汽车公司), proposed a new way of representing curve.



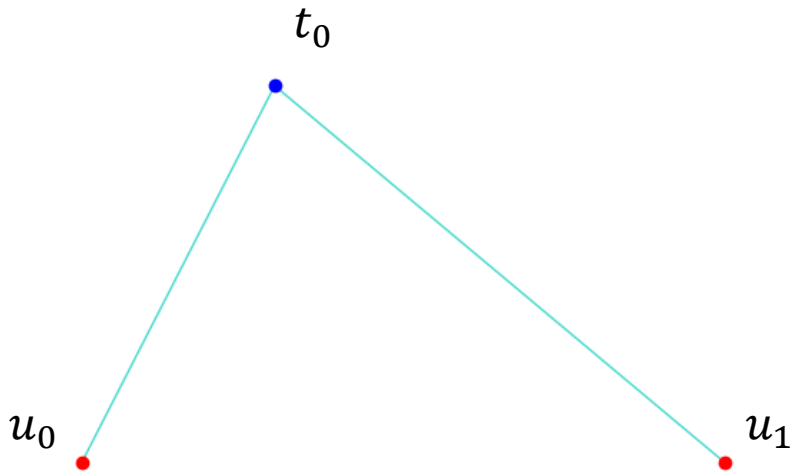
A smooth curve passing u_0 and u_1

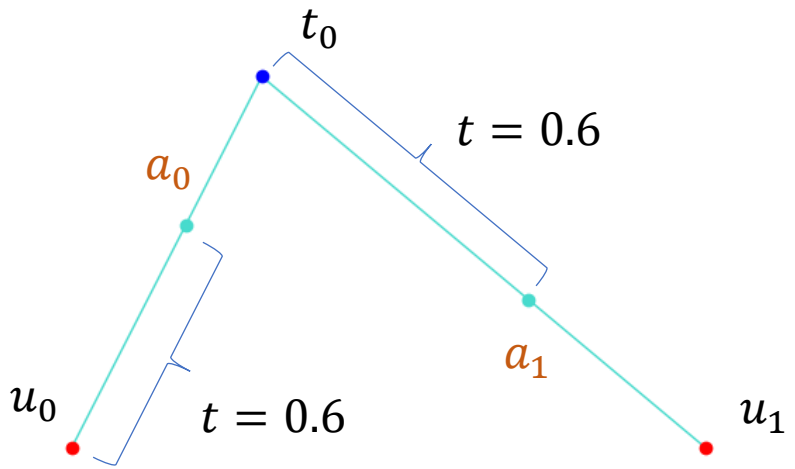


An extremely nice explanation of Bezier curve:
https://www.youtube.com/watch?v=aVwxzDHniEw

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

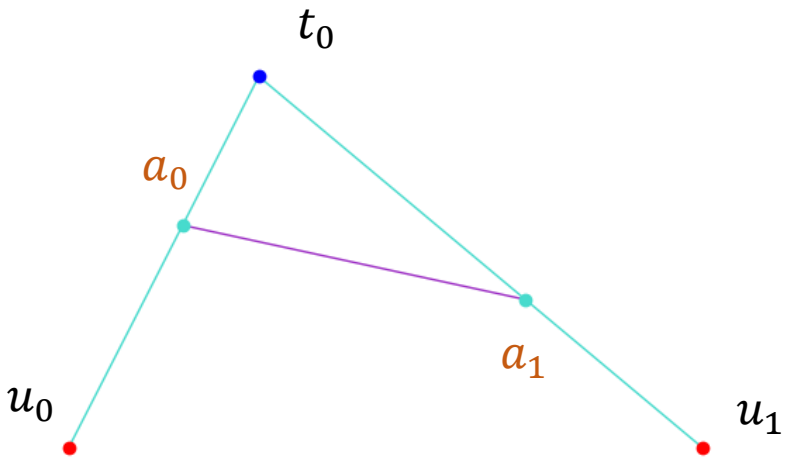
How to Get a Bezier Curve?

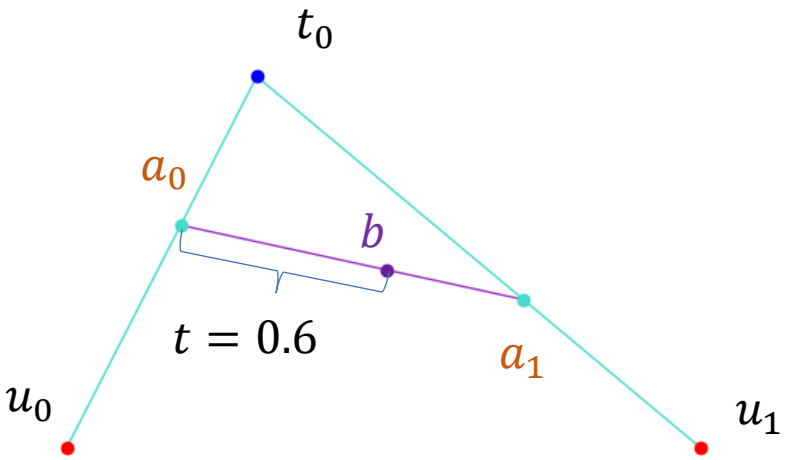




- Assume we have only one additional control point t_0
- We take two immediate points using *lerp*
- $a_0 = lerp(u_0, t_0, t)$ $a_1 = lerp(t_0, u_1, t)$

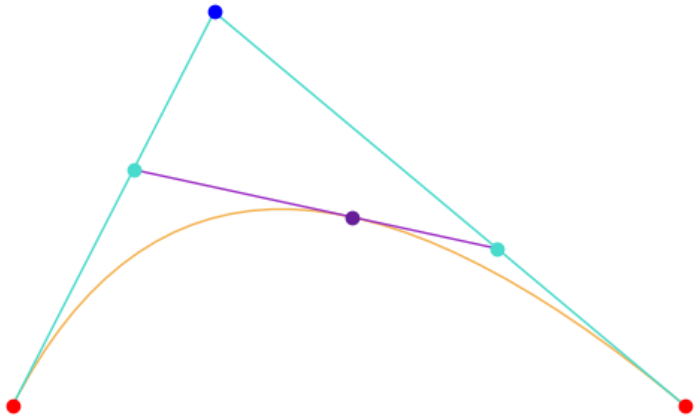
How to Get a Bezier Curve?

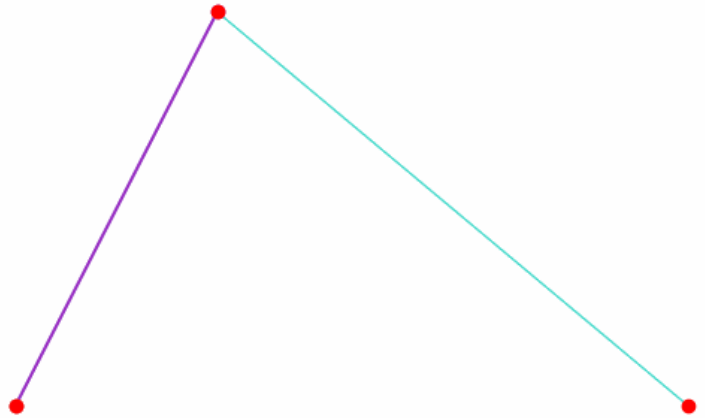




- $a_0 = lerp(u_0, t_0, t)$ $a_1 = lerp(t_0, u_1, t)$
- Connect a_0 and a_1, take a final lerp
- $b = lerp(a_0, a_1, t)$

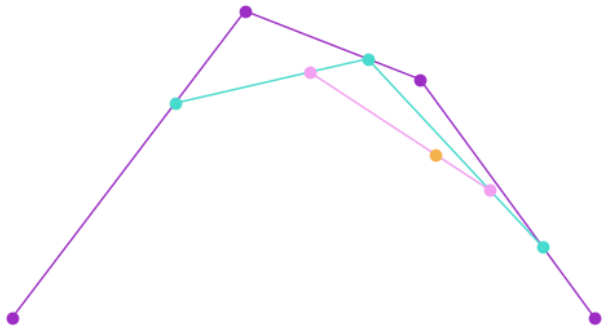
How to Get a Bezier Curve?

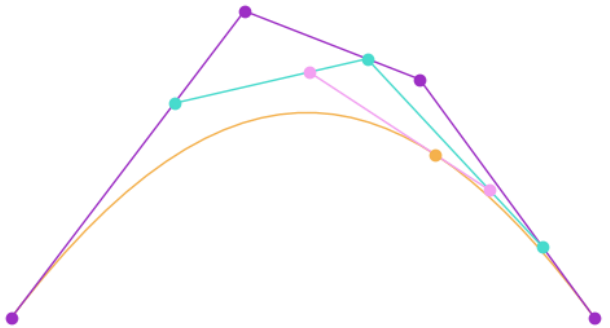




- Move t from 0 to 1, will get a smooth curve
- We call it square Bezier curve

Higher Order Bezier Curve





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

How to draw a Bezier Curve?

ChatGPT 自动

共享

用递归的算法，写一下Bezier Curve画线的算法；起始点和终止点是u0,u1,控制点是c1,...cn, 共n个

贝塞尔曲线 (Bezier Curve) 是通过一系列控制点来生成的平滑曲线。递归算法是计算贝塞尔曲线的经典方式。给定起始点 `u0`、终止点 `u1` 和一组控制点 `c1, c2, ..., cn`，我们可以用递归算法来计算贝塞尔曲线上的点。

递归贝塞尔曲线算法

假设我们有 `n+2` 个控制点：`u0, c1, c2, ..., cn, u1`，其中 `u0` 是起始点，`u1` 是终点，其余 `c1,...,cn` 是中间的控制点。我们要在参数 `t`（从 0 到 1 之间的值）上递归地计算贝塞尔曲线的点。

De Casteljau 算法（递归贝塞尔算法）

给定参数 `t`，通过递归插值的方式计算贝塞尔曲线上的点。

1. 对于每对相邻的控制点，我们根据参数 `t` 计算它们的线性插值。
2. 重复该过程直到只剩下一个点。

算法伪代码

python 复制代码

```
def bezier_recursive(points, t):
    # 如果只剩一个点，递归结束，返回该点
    ?

    # 否则对每对相邻点进行线性插值
    new_points = []
    for i in range(len(points) - 1):
        # 线性插值公式：P = (1 - t) * P1 + t * P2
        ?

    # 递归计算剩下的控制点集合
    return ?
```

如何调用

假设控制点 `u0`，`u1`，和 `c1,...,cn` 都是二维坐标 `(x, y)` 形式的点，可以传递一个包含所有点的列表：

python 复制代码

```
# 控制点的列表：u0, c1, ..., cn, u1
points = [u0, c1, c2, ..., cn, u1]

# 计算 t = 0.5 时的贝塞尔曲线上的点
point_on_curve = bezier_recursive(points, 0.5)
```

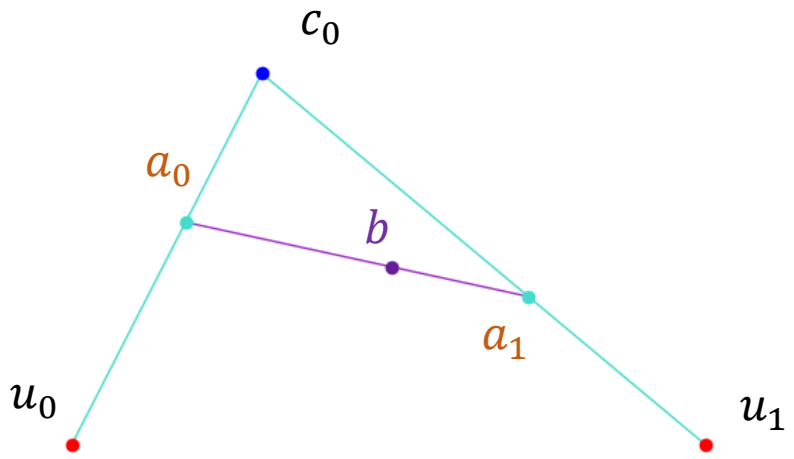
解释

- **参数** `t` 是从 0 到 1 的实数，表示曲线上某一点的位置。
- **递归终止条件** 是当控制点列表中只剩下一个点时，该点就是在参数 `t` 下贝塞尔曲线上的点。

这个递归算法通过对每对控制点的线性插值，逐步缩减控制点的数量，最终得到曲线上的一个点。这就是经典的 De Casteljau 算法，用于构造贝塞尔曲线。

Some Math in Bezier Curve

- The above method is called De Casteljau‘s algorithm (德卡斯特里奥)
- Bezier Curve also has analytical expression

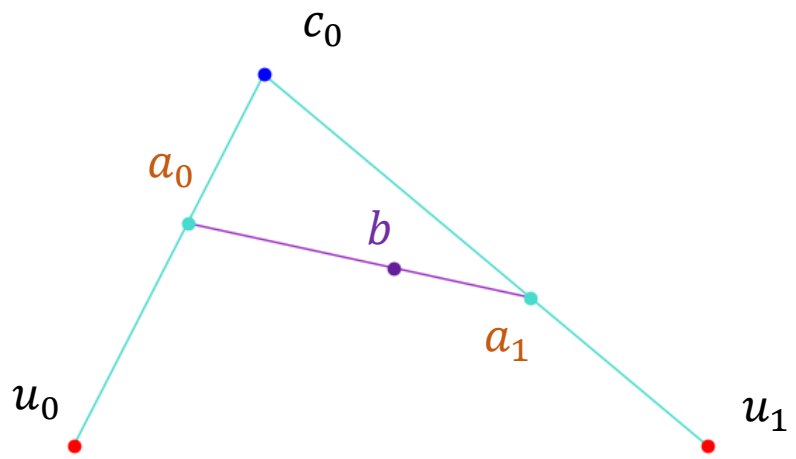


$$a_0 = (1-t)u_0 + tc_0$$

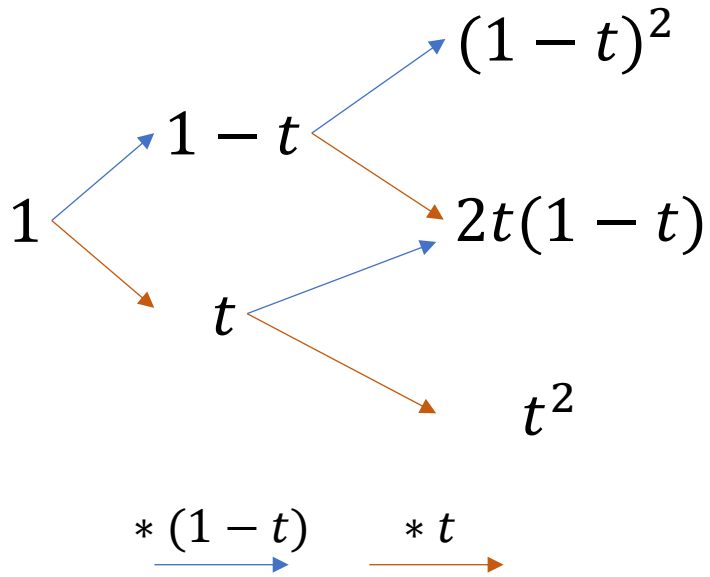
$$a_1 = (1-t)c_0 + tu_1$$

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

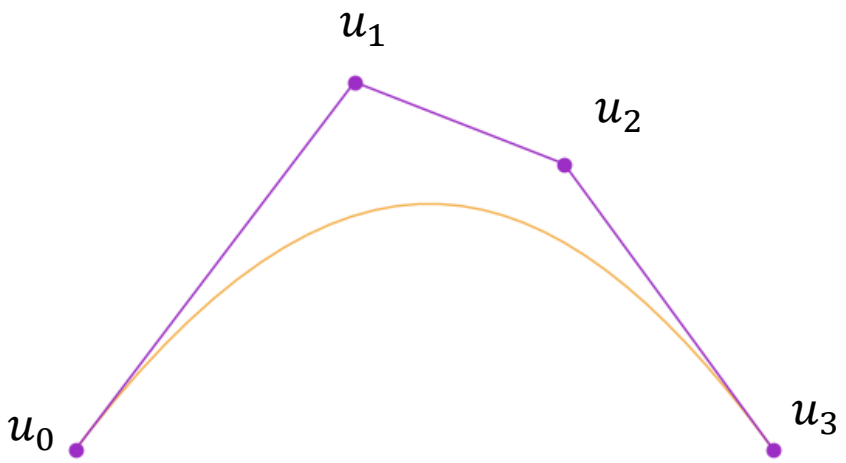
Some Math in Bezier Curve

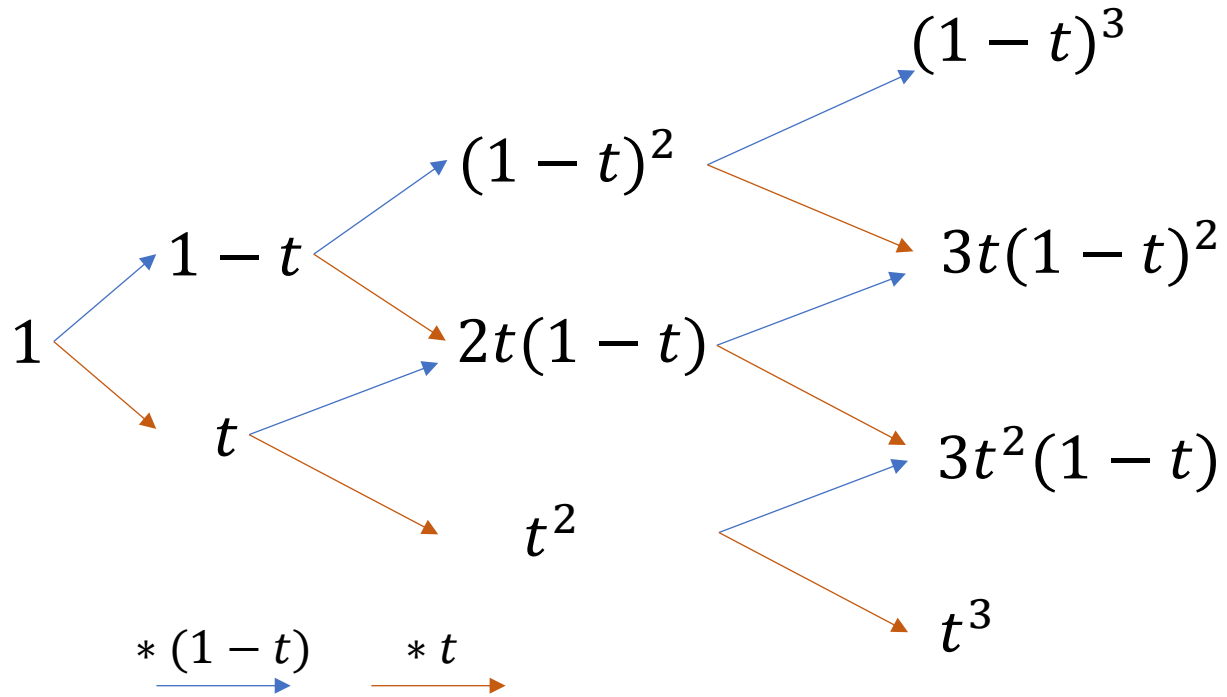


$$b = (1-t)a_0 + ta_1$$
$$= (1-t)^2 u_0 + 2t(1-t)c_0 + t^2 u_1$$



Some Math in Bezier Curve





Exactly Pascal's triangle! (杨辉三角)

Also Bernstein polynomials

1

1 1

1 2 1

1 3 3 1

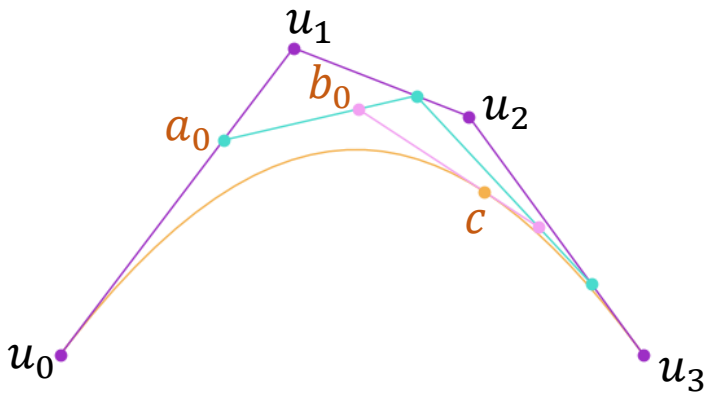
1 4 6 4 1

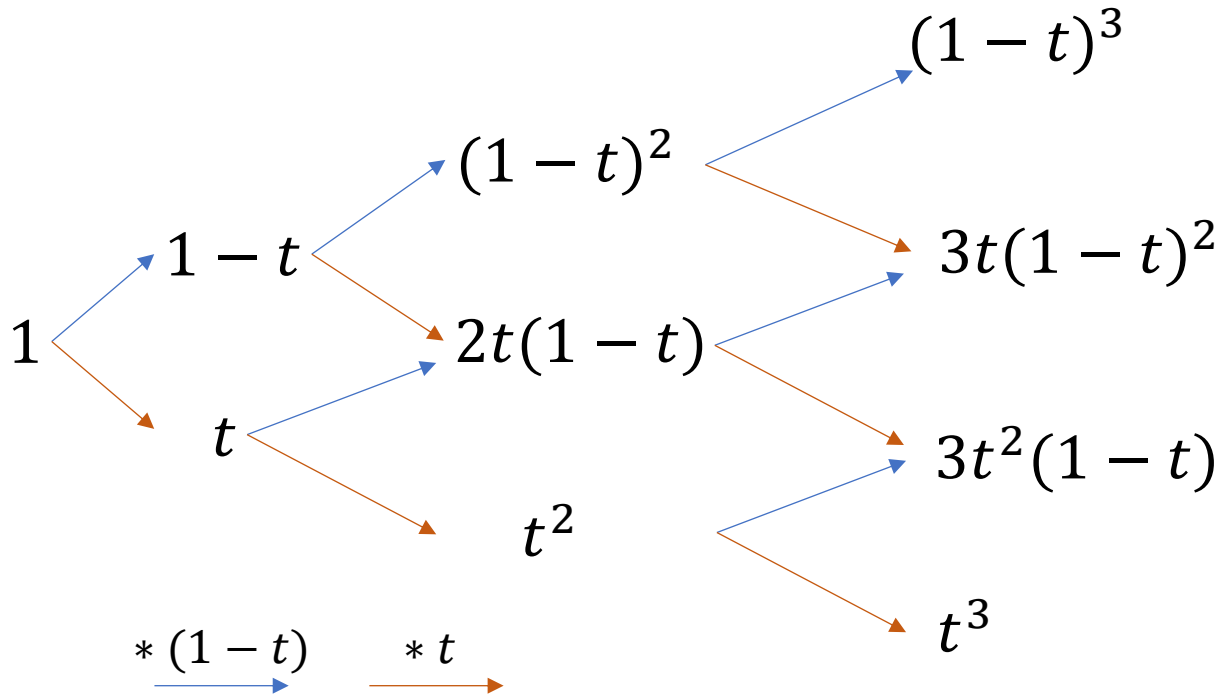
1 5 10 10 5 1

1 6 15 20 15 6 1

1 7 21 35 35 21 7 1

Some Math in Bezier Curve





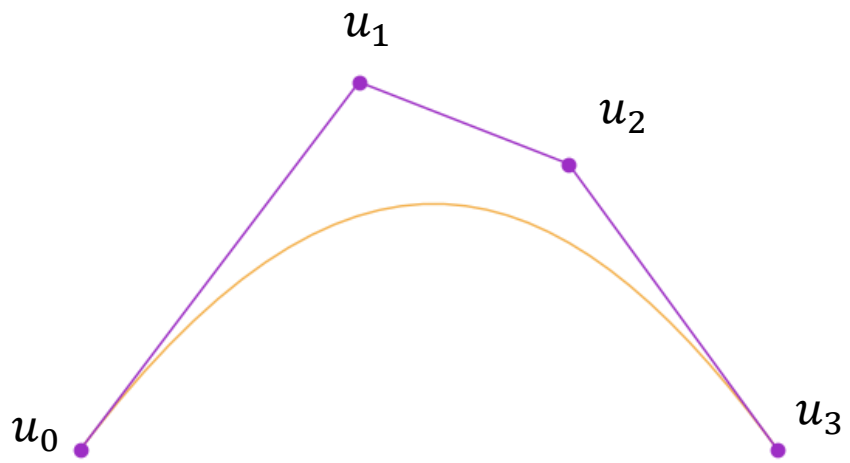
Exactly Pascal's triangle! (杨辉三角)

We can get analytical expression:

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

Some Math in Bezier Curve

- The above method is called De Casteljau‘s algorithm (德卡斯特里奥)
- Bezier Curve also has analytical expression



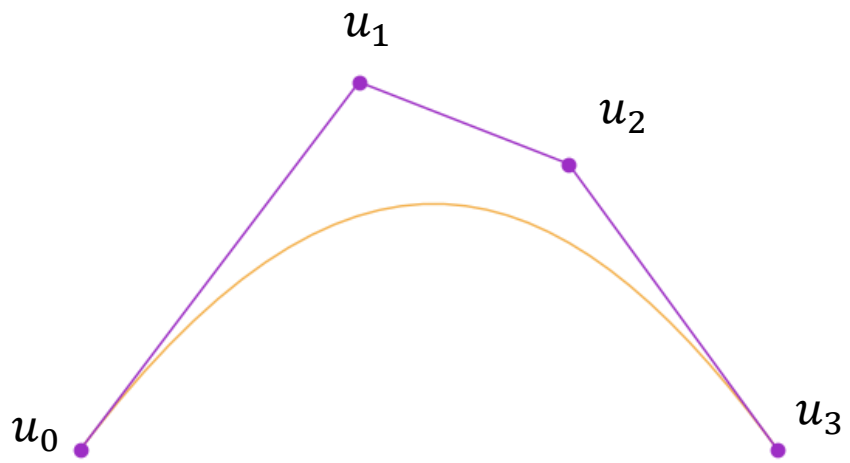
We can get analytical expression:
Pascal’s triangle! (杨辉三角)

$$B(t) = \sum_k^n \binom{n}{k} t^k (1-t)^{n-k}\, \mathbf{u}_k \ , \ k = 0,1, \dots n$$

(Bernstein polynomials)

Some Math in Bezier Curve

- The above method is called De Casteljau‘s algorithm (德卡斯特里奥)
- Bezier Curve also has analytical expression
- It’s more efficient and numerically stable comparing with analytical expression

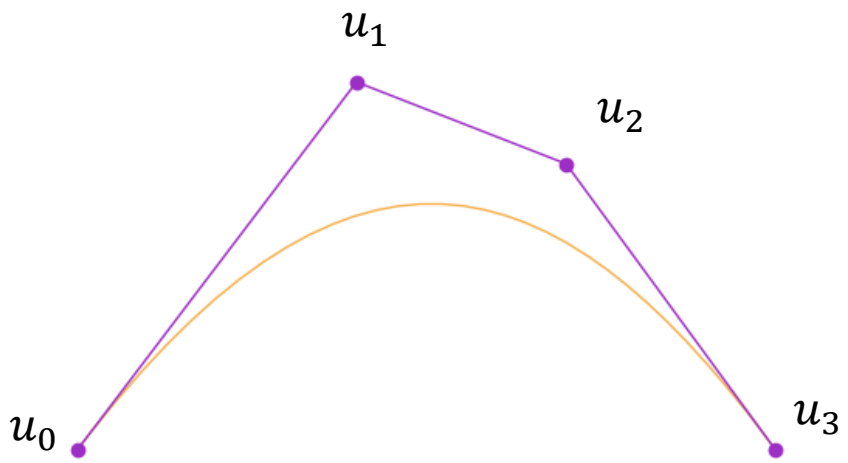


We can get analytical expression:
Pascal’s triangle! (杨辉三角)

$$B(t) = \sum_k^n \binom{n}{k} t^k (1-t)^{n-k} \, \mathbf{u}_k \;\; , \;\; k = 0,1, \ldots n$$

(Bernstein polynomials)

Some Math in Bezier Curve



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

$$B(t) = (1-t)^3 u_0 + 3t(1-t)^2 u_1 + 3t^2(1-t)u_2 + t^3 u_3$$

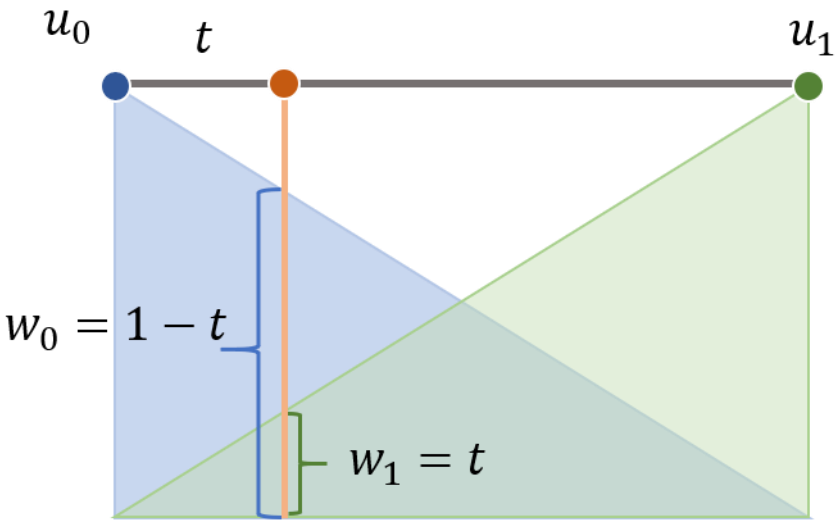
$$p(t) = \boldsymbol{\tau}^{\mathrm{T}} \mathrm{M}_B P = b(t)^T P$$

$$\mathrm{P} = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \quad \begin{bmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 3t^2(1-t) \\ t^3 \end{bmatrix}$$

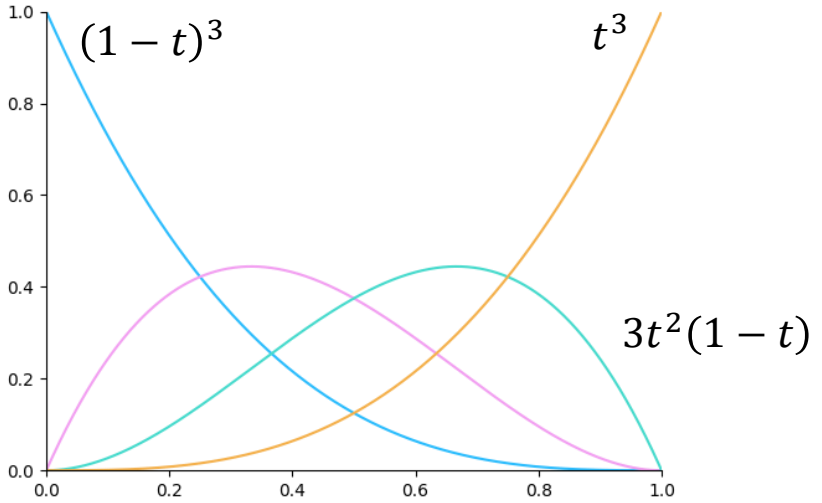
blending functions

Some Math in Bezier Curve



Weight of lerp

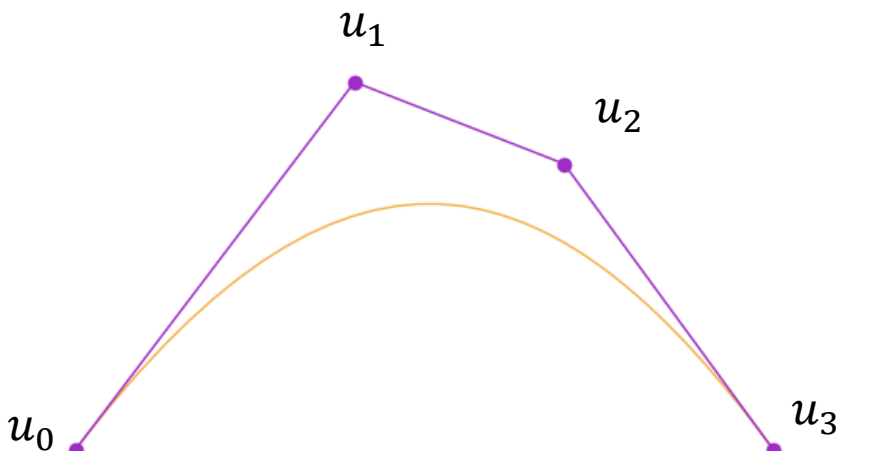
$$b(t) = \begin{bmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 2t^2(1-t) \\ t^3 \end{bmatrix}$$

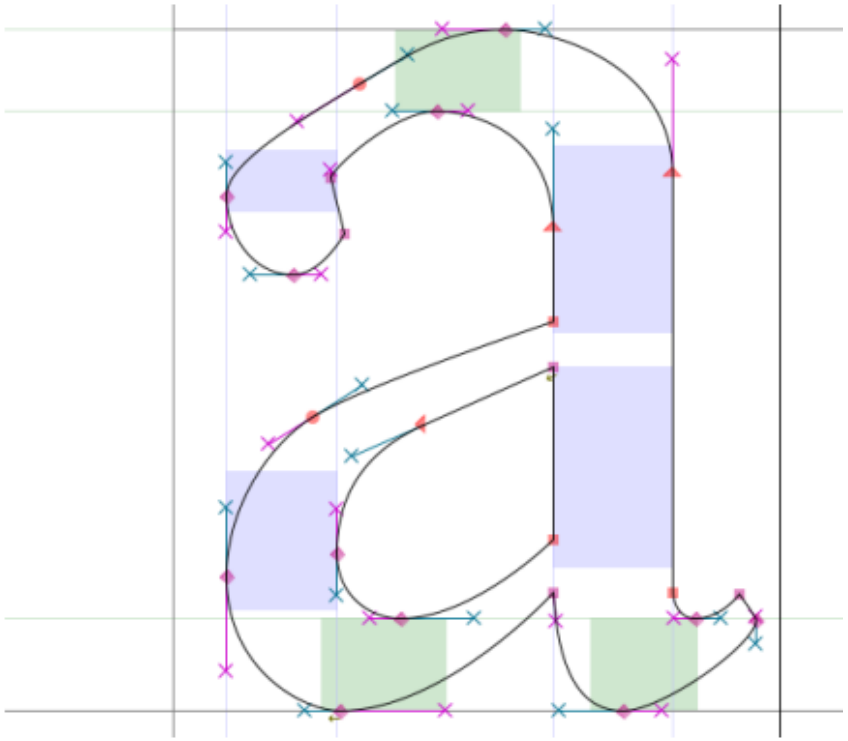


Weight of Cubic Bezier

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

Some Math in Bezier Curve





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

$$B'(0) = n(u_1 - u_0)$$

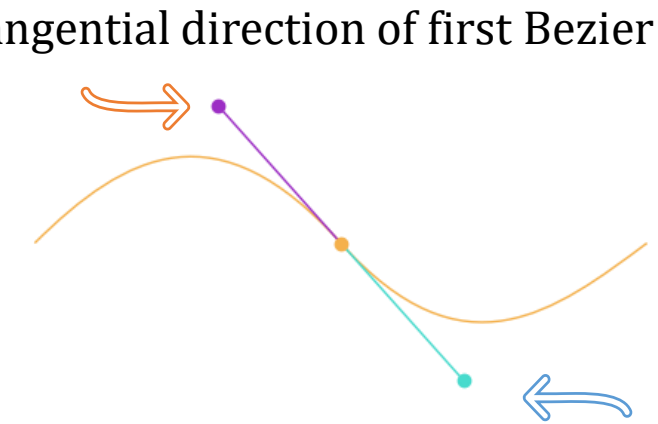
$$B'(1) = n(u_n - u_{n-1})$$

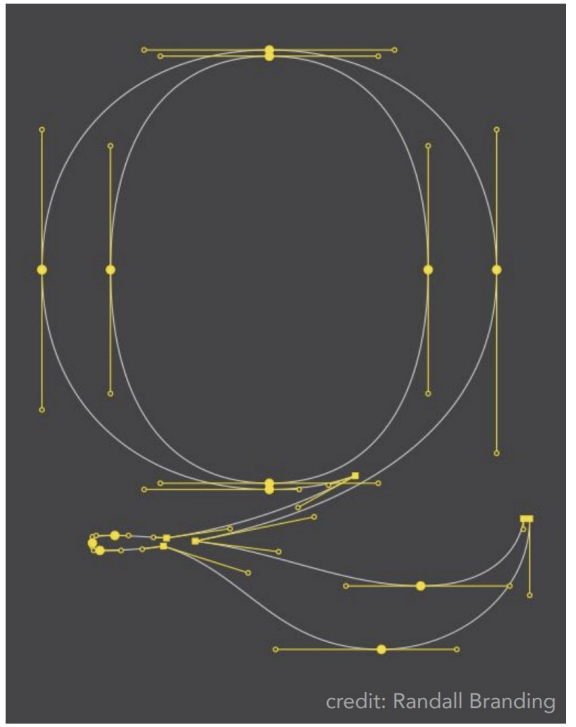
The tangential direction at u_0 is $u_0 \rightarrow u_1$!

Some Math in Bezier Curve

We can *concatenate* two Bezier curves *smoothly*

Just make a pair of central symmetry control points!



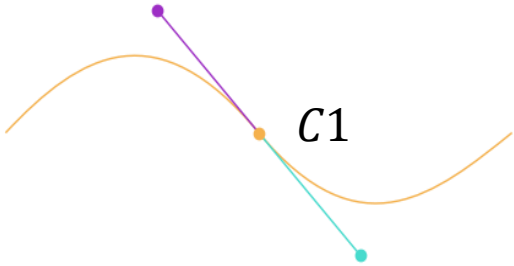


Geometry continuity

G1 continuity: aligned tangential direction (both normalized)

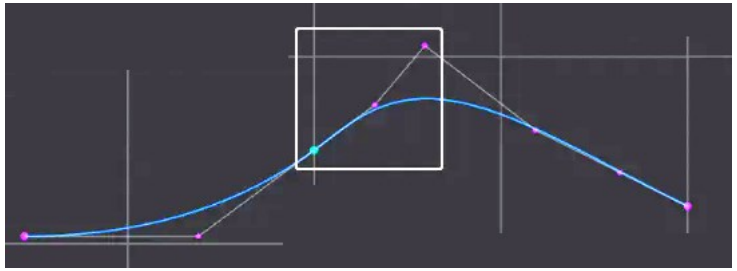
C1 continuity: Continuous wrt parameter t

Same tangential direction and norm

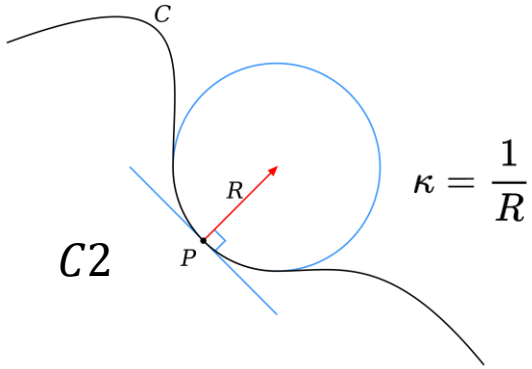


G2 continuity: G1 continuity + a common center of curvature

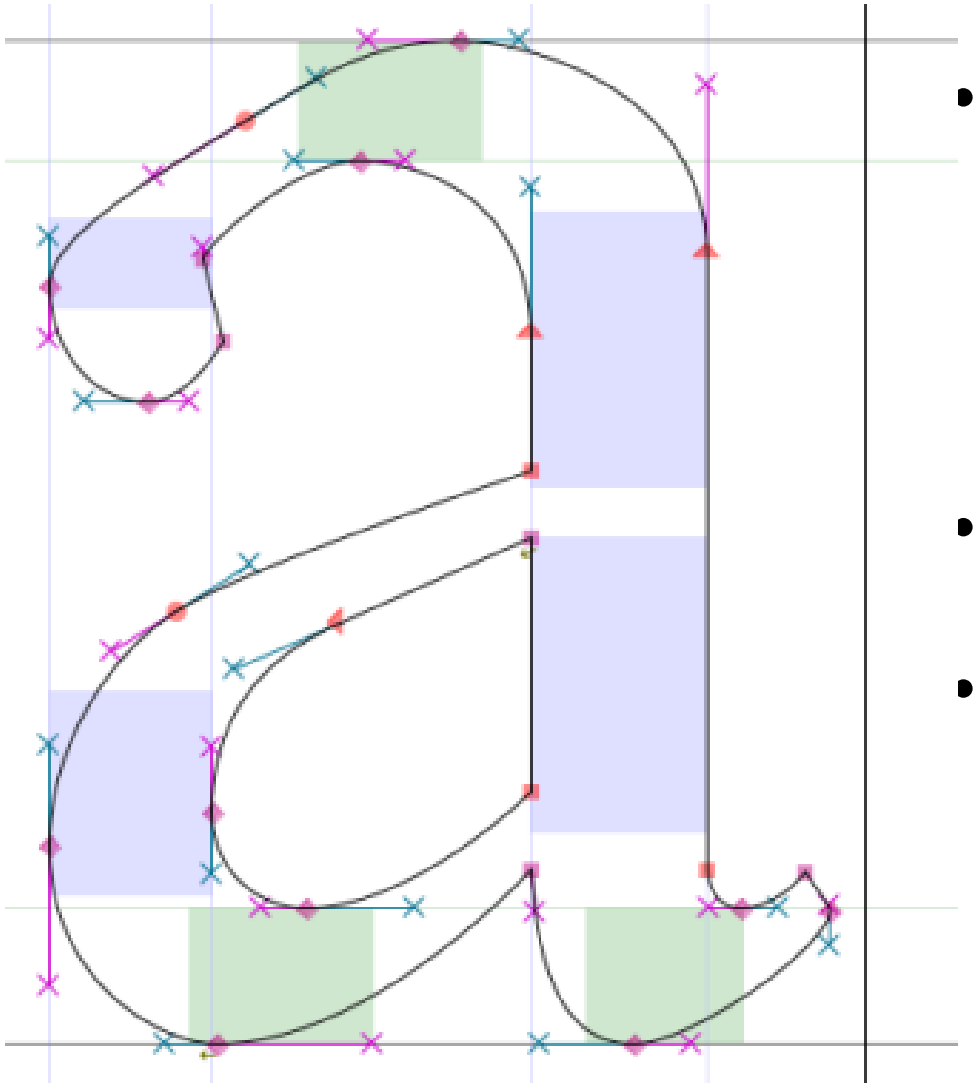
$$g^{(1)}(0) = \beta_1 f^{(1)}(1)$$
$$g^{(2)}(0) = \beta_1^2 f^{(2)}(1) + \beta_2 f^{(1)}(1)$$



C2 continuity: C1 continuity + same curvature



Piecewise cubic Bézier curves



- Piecewise -> **Local Control**:

  control a cubic Bézier curve by four control points, not affecting other parts.

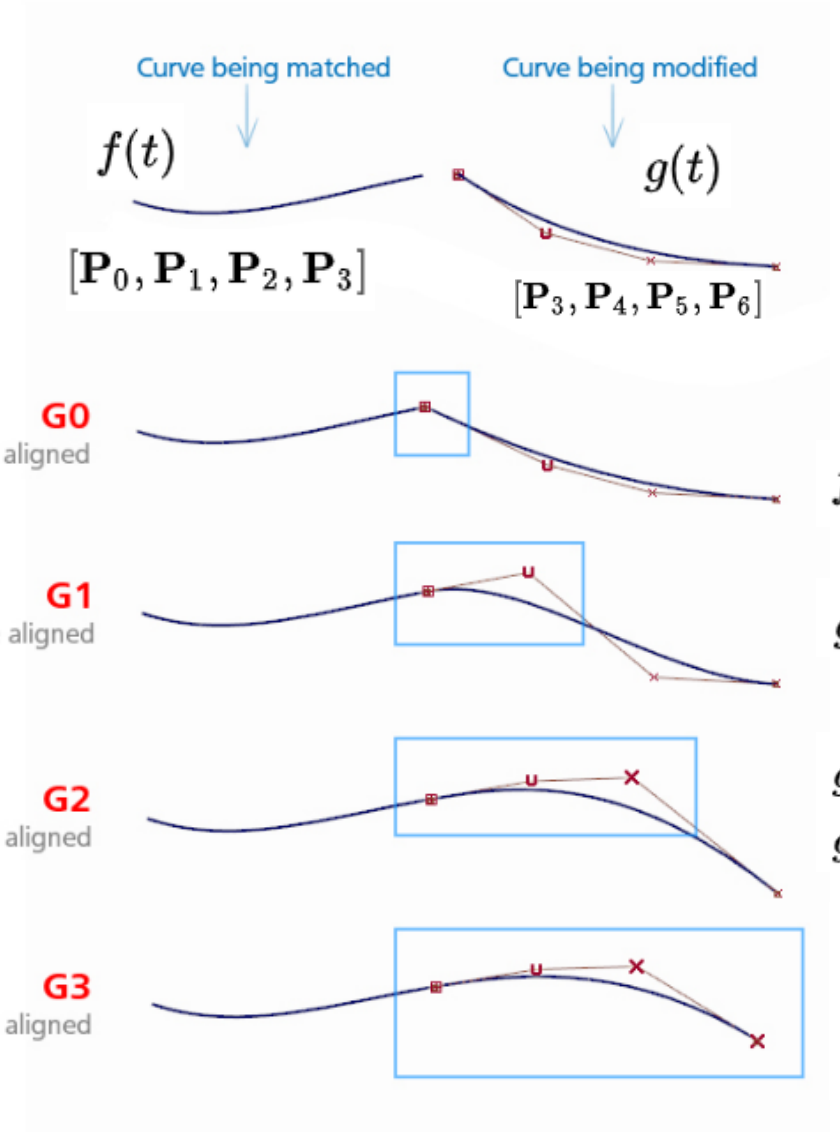
- cubic -> **Curvature** (second derivative)

- Piecewise cubic -> **Smoothness :**

  Easy to ensure G^0, G^1, C^1 and G^2 continuity

  sometimes $C^2$and G^3 continuity

Geometry continuity



$$f(1) = g(0)$$

the shared point is $\mathbf{P}_3$

$$g^{(1)}(0) = \beta_1 f^{(1)}(1)$$

$$\mathbf{P}_4 = \mathbf{P}_3 + (\mathbf{P}_3 - \mathbf{P}_2)\beta_1$$

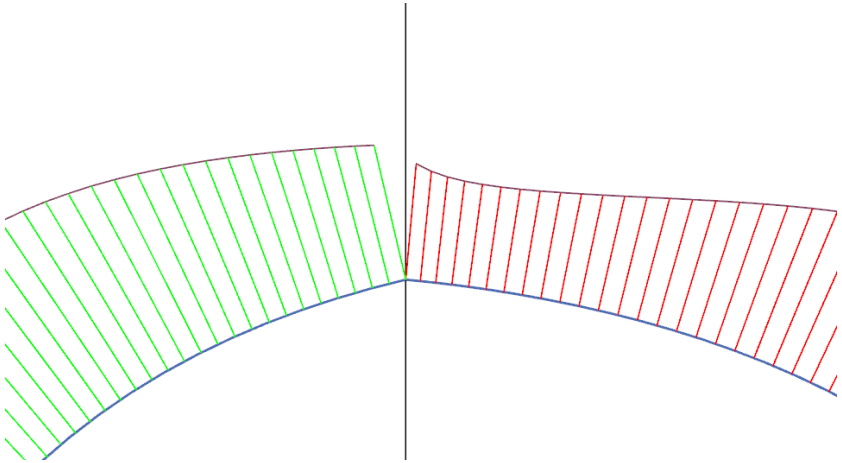
$$g^{(1)}(0) = \beta_1 f^{(1)}(1)$$
$$g^{(2)}(0) = \beta_1^2 f^{(2)}(1) + \beta_2 f^{(1)}(1)$$

$$\mathbf{P}_5 = \mathbf{P}_3 + (\mathbf{P}_3 - \mathbf{P}_2)(2\beta_1 + \beta_1^2 + \beta_2/2) + (\mathbf{P}_1 - \mathbf{P}_2)\beta_1^2$$

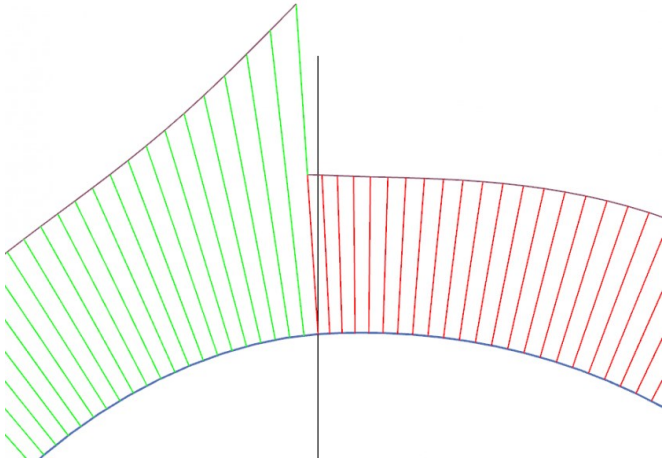
Higher degrees of geometric continuity is possible, though they get increasingly complex

$$g^{(1)}(0) = \beta_1 f^{(1)}(1)$$
$$g^{(2)}(0) = \beta_1^2 f^{(2)}(1) + \beta_2 f^{(1)}(1)$$
$$g^{(3)}(0) = \beta_1^3 f^{(3)}(1) + 3\beta_1\beta_2 f^{(2)}(1) + \beta_3 f^{(1)}(1)$$

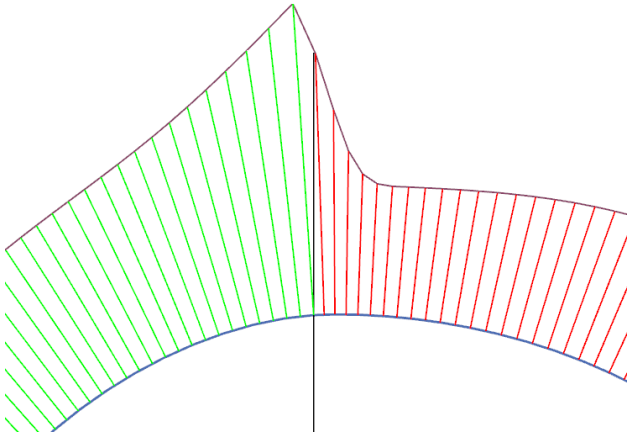
Geometry continuity



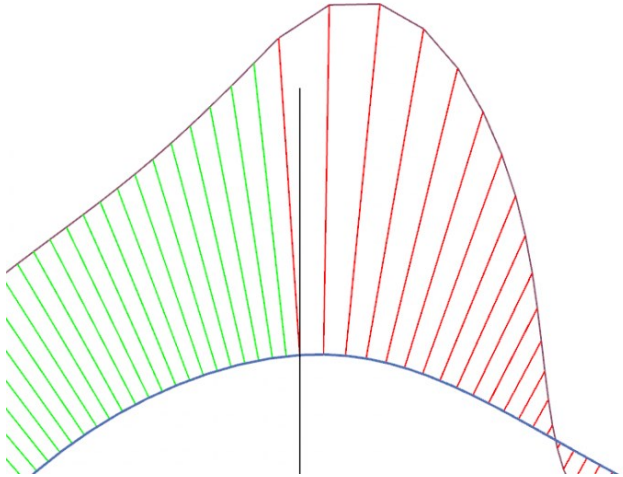
G0 Continuity – Curves Connect



G1 Continuity – Curves are Tangent (Combs Don’t Align)



G2 Continuity – Curves are Tangent & Equal Radii (Combs Connect)



G3 Continuity – Curves are Tangent, Equal Radii, Equal Rate of Change (Combs Tangent)

Bezier Patches

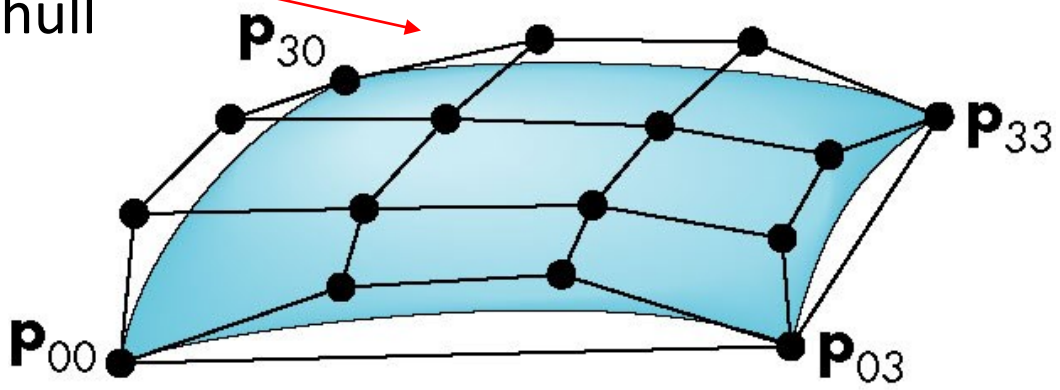
$$2D:\ p(t) = \boldsymbol{\tau}^{\mathrm{T}}\mathrm{M}_B P\ =\ b(t)^T P$$

Using same data array **P**=[p_{ij}] as with interpolating form $u=\begin{bmatrix}1\\u\\u^2\\u^3\end{bmatrix}$ $v=\begin{bmatrix}1\\v\\v^2\\v^3\end{bmatrix}$

$$p(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} b_i(u)\, b_j(v)\, p_{ij} = u^T \mathbf{M}_B \mathbf{P} \mathbf{M}_B^T v$$

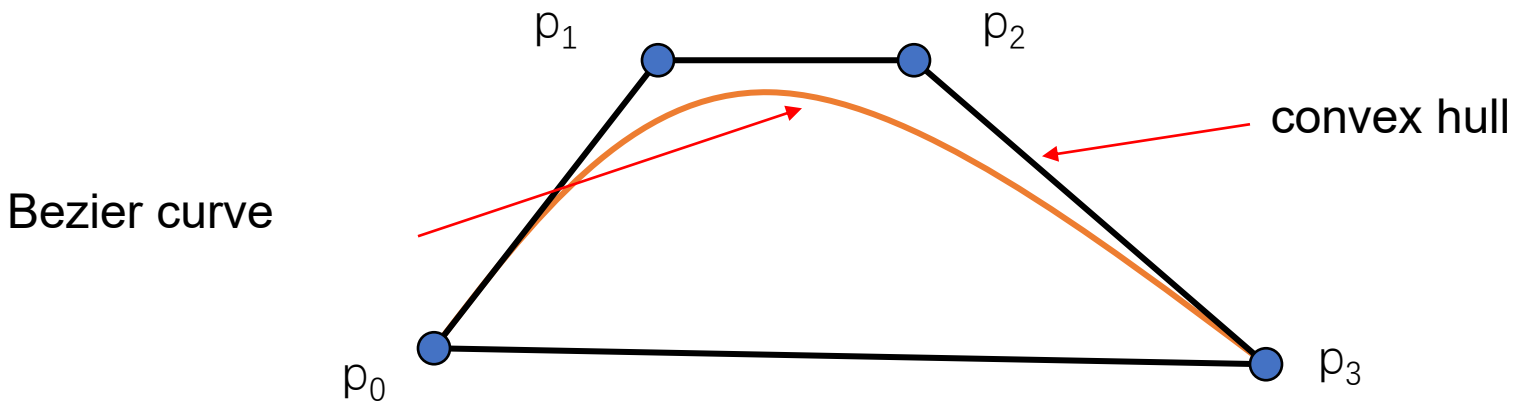
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

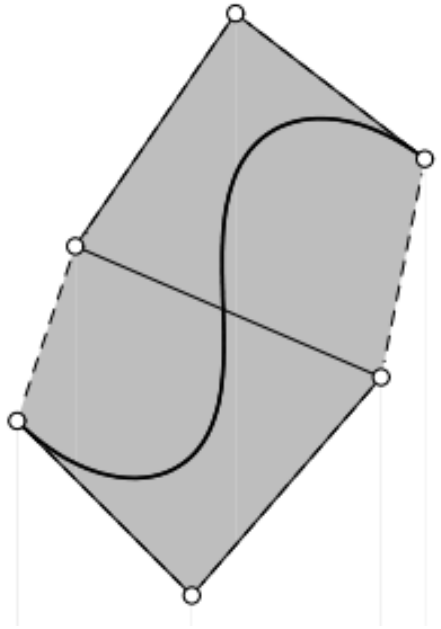
Patch lies in convex hull



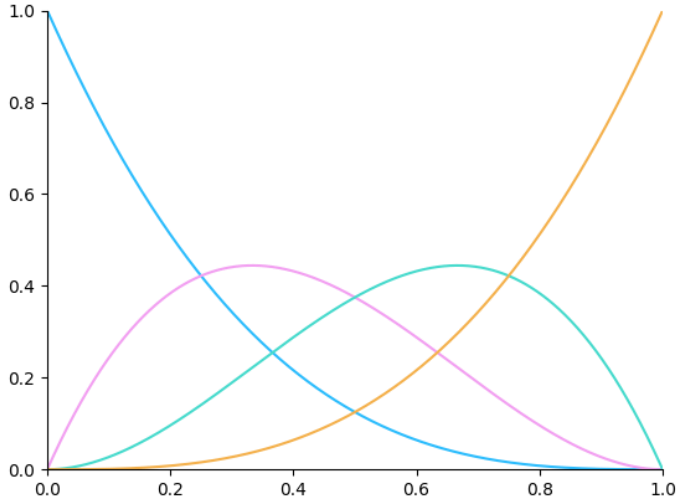
Analysis (1)

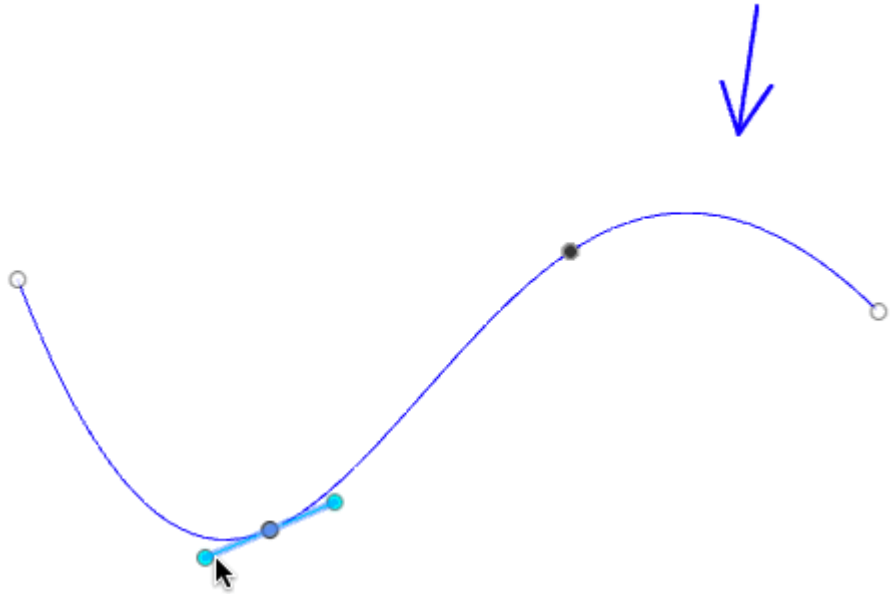
- The properties of the Bernstein polynomials ensure that all Bezier curves lie in the convex hull of their control points (although not the tightest bound!)
- Hence, even though we do not store all the data, we cannot be too far away





Analysis (2)



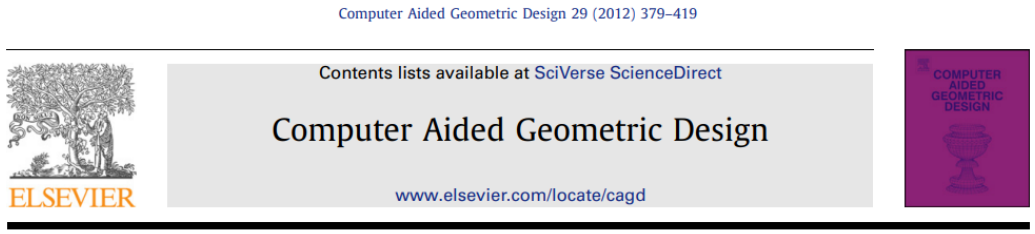


A demo: https://www.autodesk.com/products/fusion-360/blog/sketch-control-point-splines-faq/

- In a piece: Every control point effects the entire piece
- Between pieces: no influences at all
- Add control points in each piece will increase the order of polynomial (sensitivity↑)

Analysis (3)

- Increasing continuity at joint points is tricky!
  - Apply more conditions
  - Usually means order increase
  - Higher order Bezier means more work (adding a lot more control points)
- Supported by OpenGL


Computer Aided Geometric Design 29 (2012) 379–419

Contents lists available at SciVerse ScienceDirect

Computer Aided Geometric Design

www.elsevier.com/locate/cagd

ELSEVIER

The Bernstein polynomial basis: A centennial retrospective ☆

Rida T. Farouki

Department of Mechanical and Aerospace Engineering, University of California, Davis, CA 95616, United States

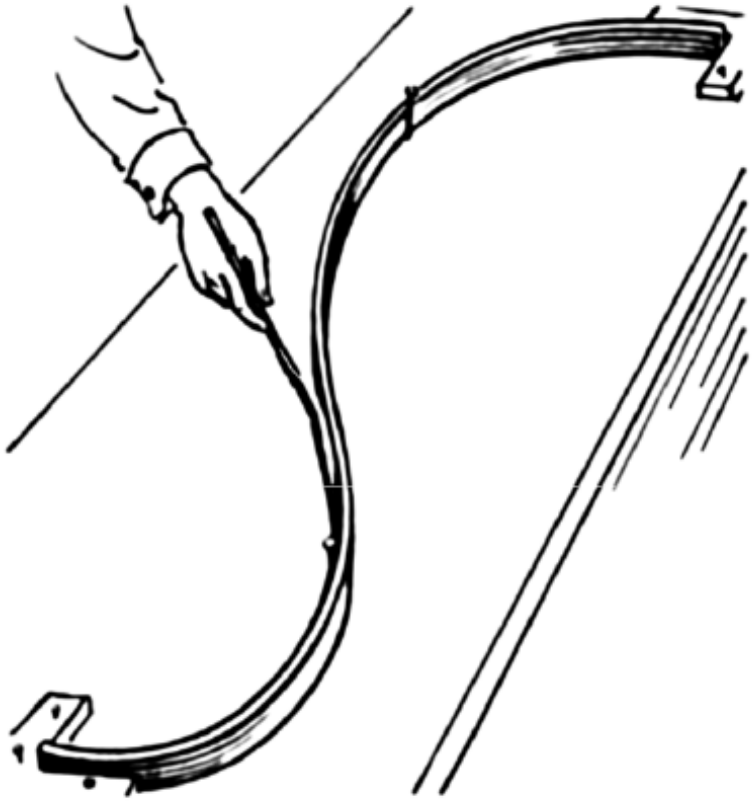
https://doi.org/10.1016/j.cagd.2012.03.001

Popular Parametric Curves:
Spline Curves

Spline Curve (Quadratic/Cubic splines, Basis-Spline)

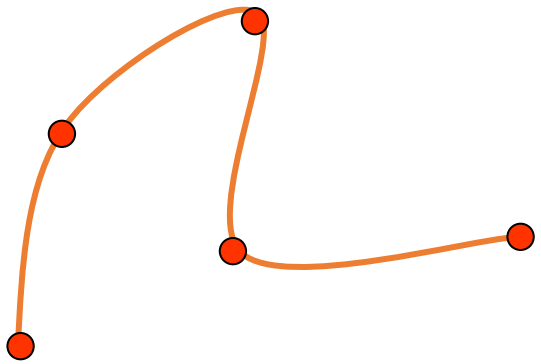
In the foreword to (Bartels et al., 1987), Robin Forrest describes "lofting", a technique used in the British aircraft industry during World War II to construct templates for airplanes by passing thin wooden strips (called "splines") through points laid out on the floor of a large design loft, a technique borrowed from ship-hull design

——Wiki

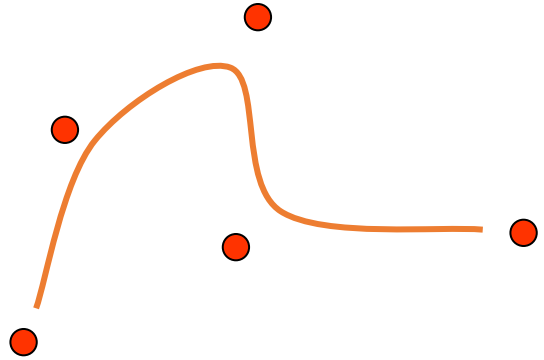


Spline Curve (Quadratic/Cubic splines, Basis-Spline)

Spline curve: smooth curve that is defined by a sequence of points



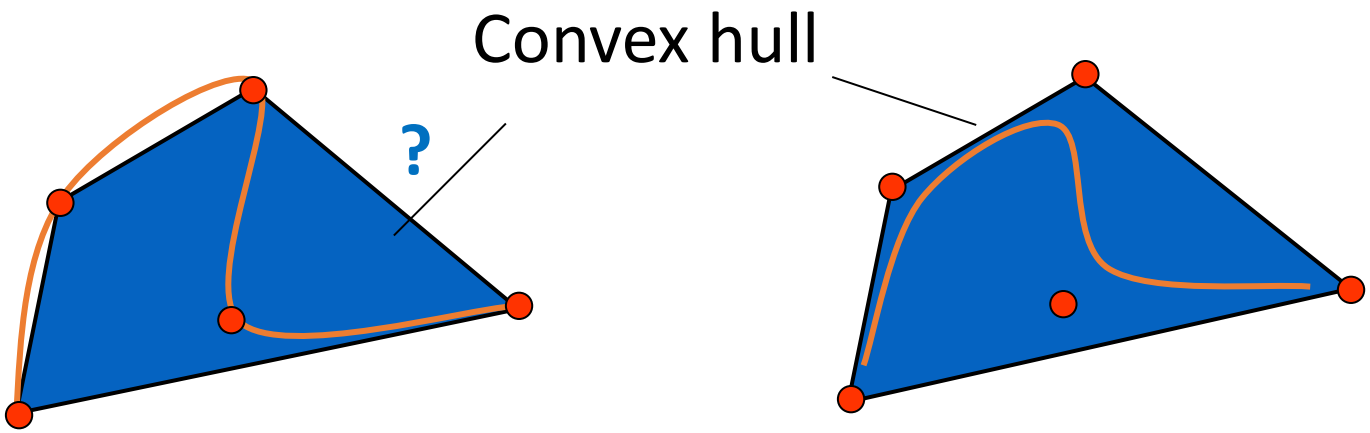
Interpolating spline



Approximating spline

Spline Curve

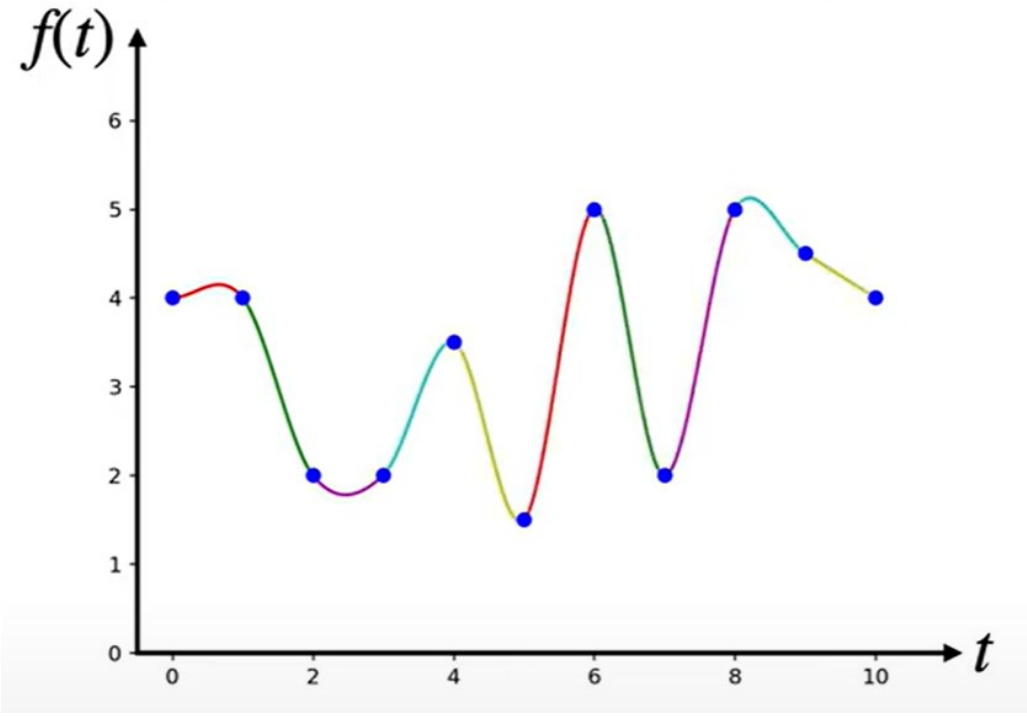
Convex hull: Smallest polygon that encloses all points



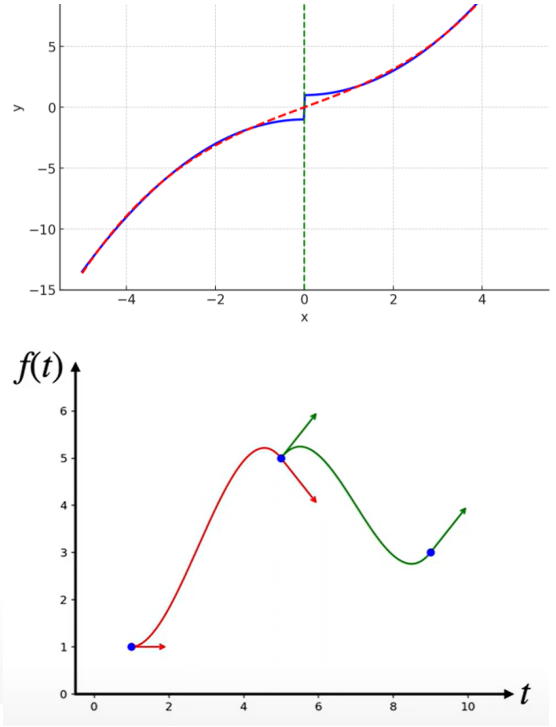
Interpolating spline

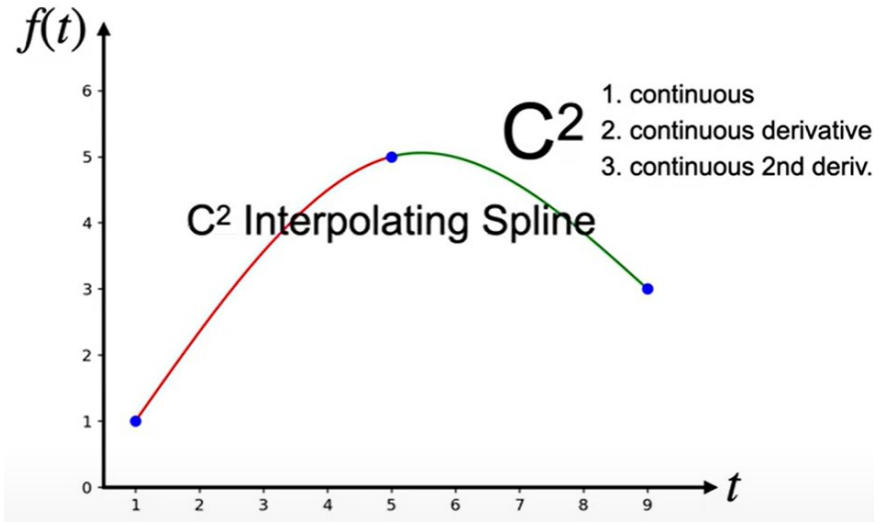
Approximating spline

Quadratic/Cubic splines



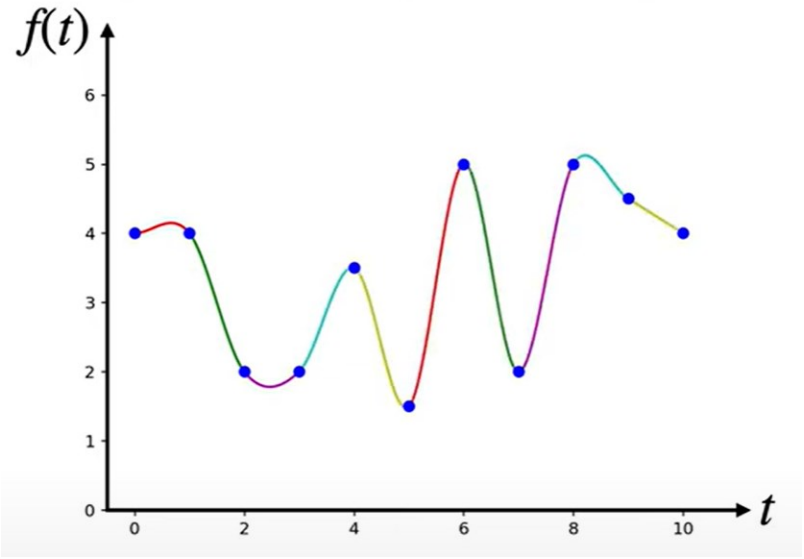
Spline Curve (Cubic)





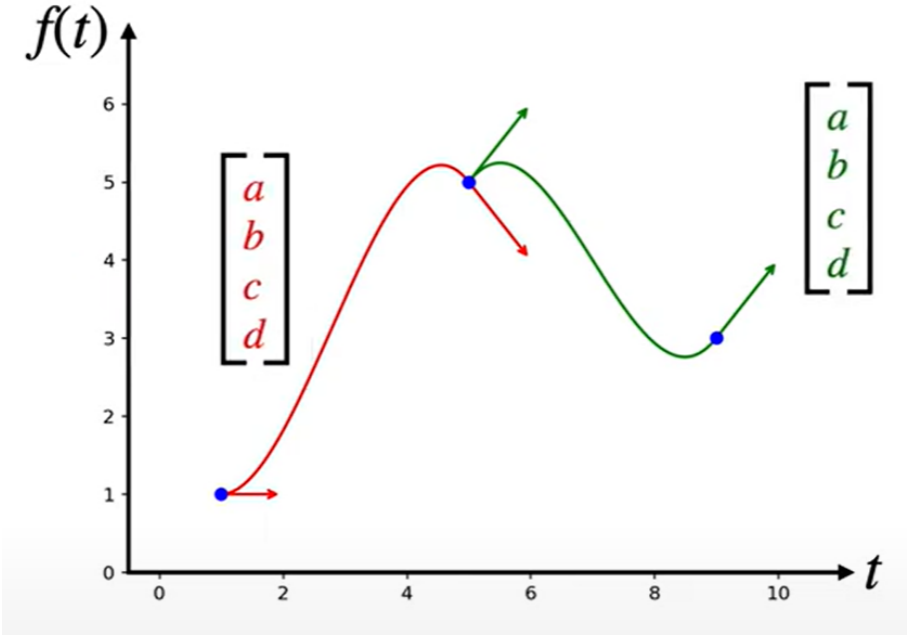
Natural Cubic Spline

Natural Cubic Spline

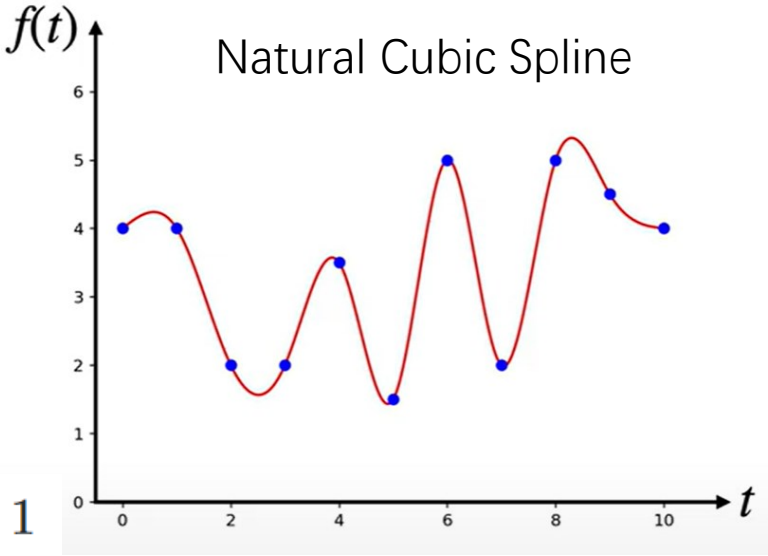


Spline Curve (Cubic)
m control points

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



$$\begin{bmatrix} s^3 & s^2 & s & 1 & 0 & 0 & 0 & 0 \\ k^3 & k^2 & k & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & k^3 & k^2 & k & 1 \\ 0 & 0 & 0 & 0 & e^3 & e^2 & e & 1 \\ 3k^2 & 2k & 1 & 0 & -3k^2 & -2k & -1 & 0 \\ 6k & 2 & 0 & 0 & -6k & -2 & 0 & 0 \\ 6s & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6e & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} c_s \\ c_k \\ c_k \\ c_e \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



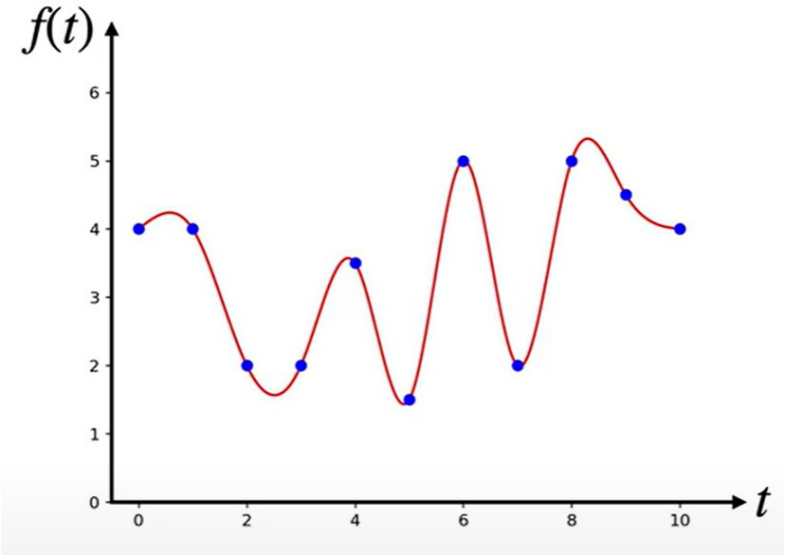
$$f_i(0) = x_i,\ f_i(1) = x_{i+1},\ i = 0, \cdots, m-1$$

$$f'_i(1) = f'_{i+1}(0), i = 1, \ldots, m-2$$
$$f''_i(1) = f''_{i+1}(0), i = 1, \ldots, m-2$$

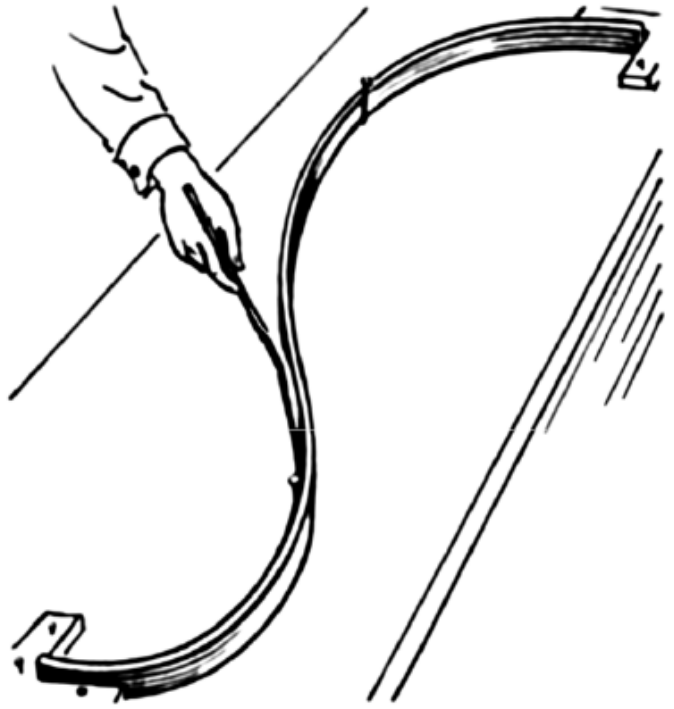
$$f'_0(0) = 0, f'_m(1) = 0; \quad f''_0(0) = 0, f''_m(1) = 0$$

Natural Cubic Spline

- Minimizing bending energy!

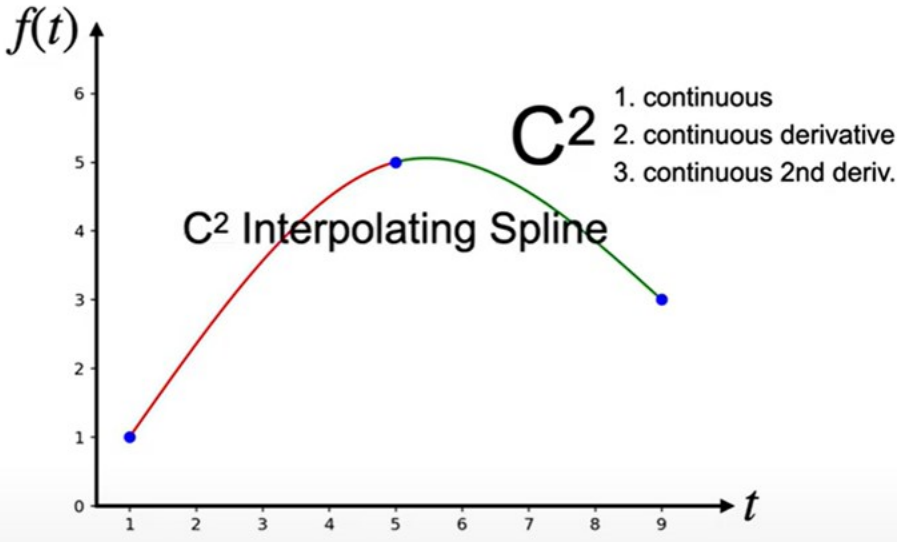


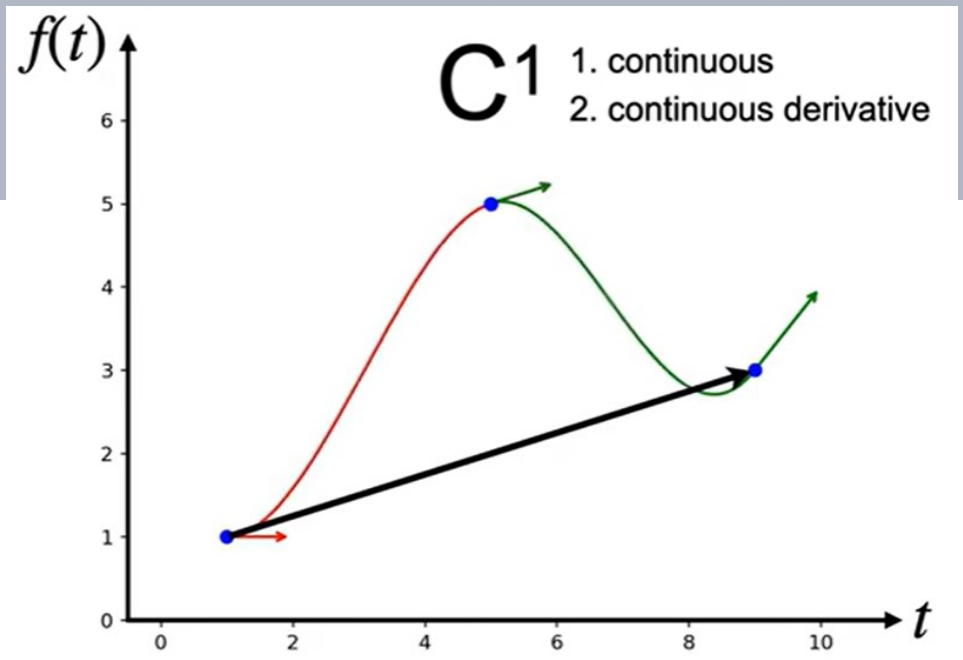
$$min \int \left(\frac{d^2 f}{dt^2} \right)^2 dt$$

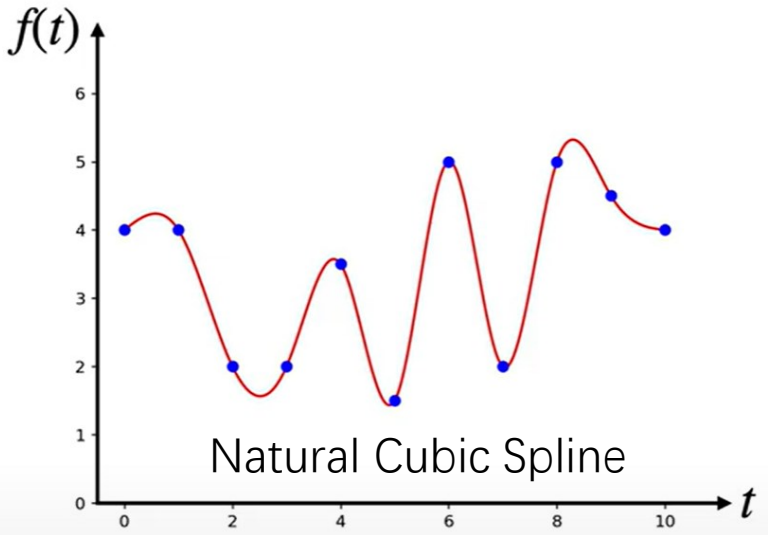


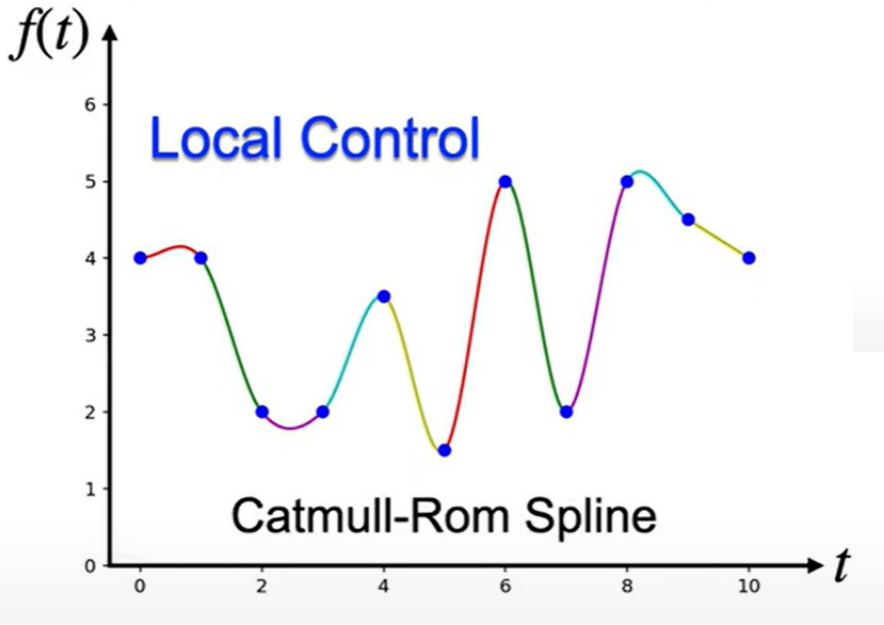
- Unfortunately global.

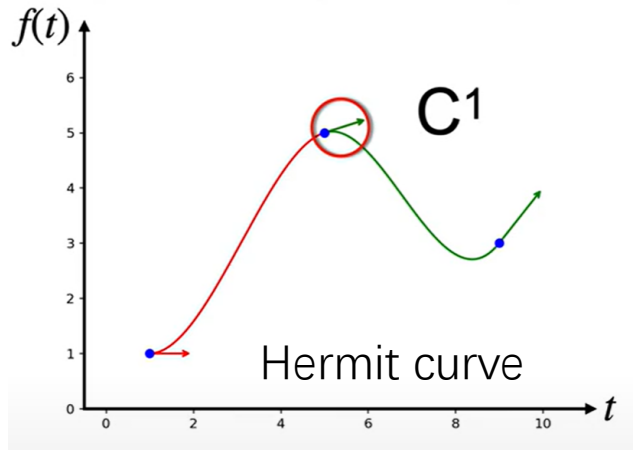
Cubic Spline











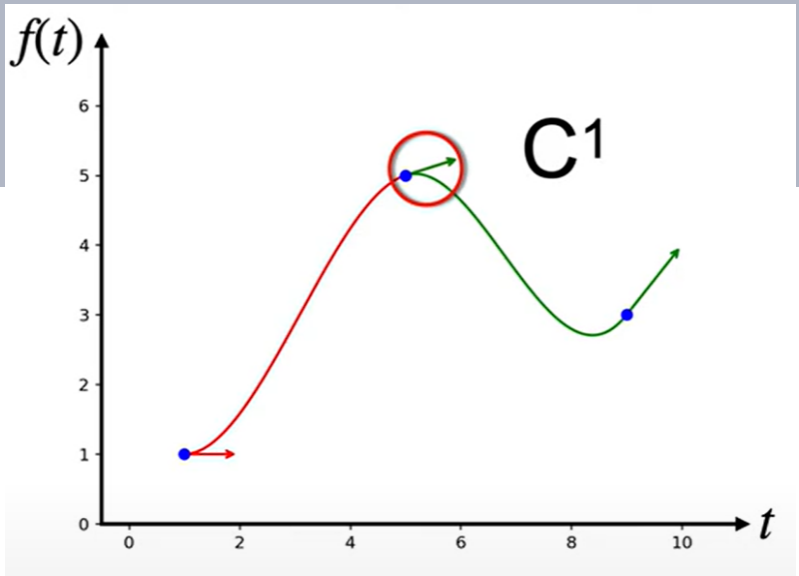
Hermit curve

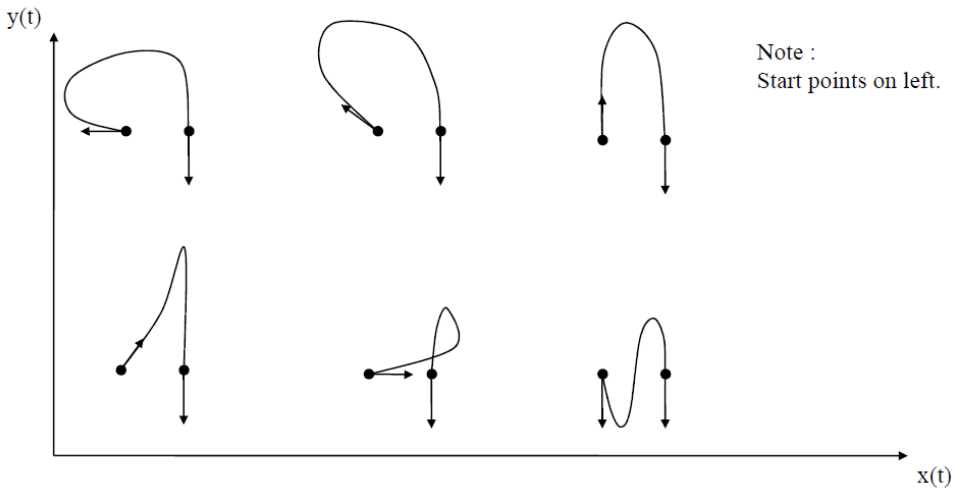
- Cubic polynomial
- Two control points and their derivatives on the two ends

$$X(t) = a_3t^3 + a_2t^2 + a_2t + a_0$$

$$X'(t) = 3a_3t^2 + 2a_2t + a_1$$

Hermite Specification



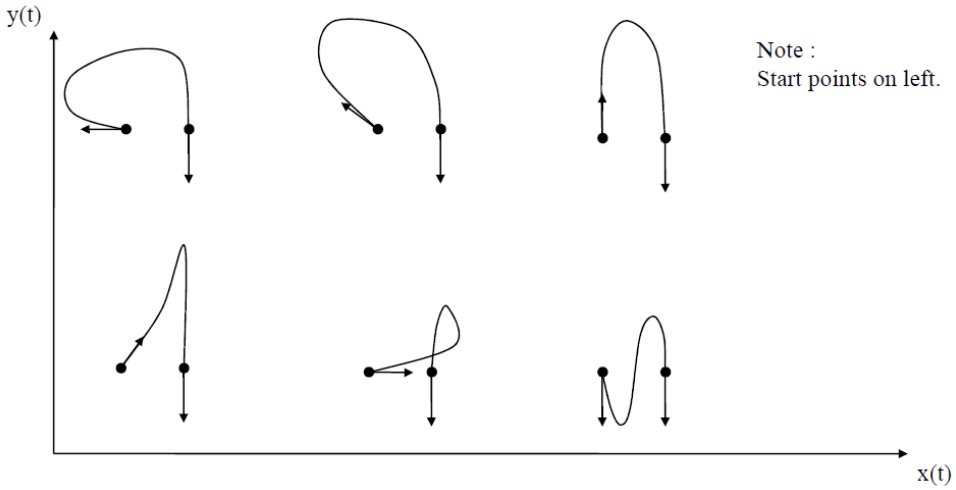


Hermit curve

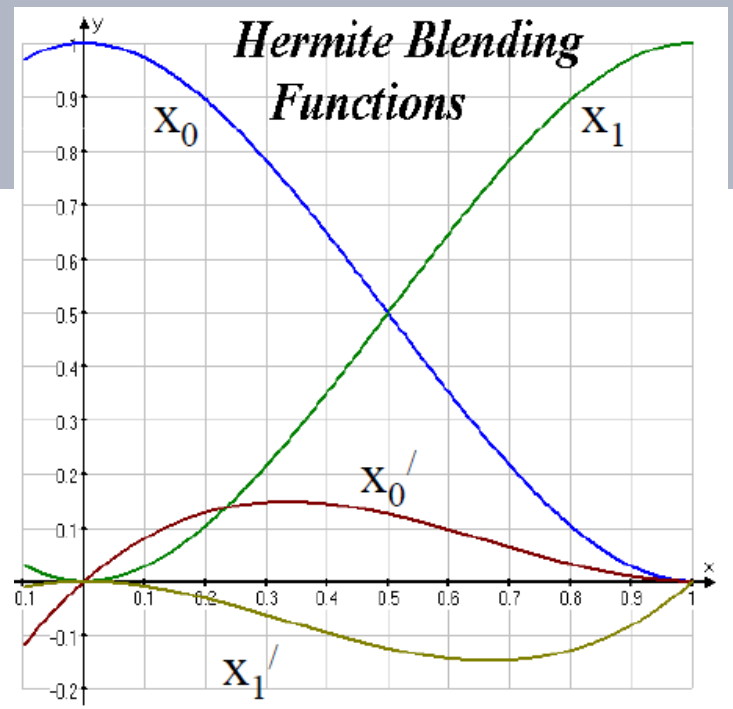
- Cubic polynomial
- Two control points and their derivatives on the two ends

$$X(t) = a_3t^3 + a_2t^2 + a_2t + a_0$$

$$X'(t) = 3a_3t^2 + 2a_2t + a_1$$



Hermite Specification



$$H(t) = h_{00}(t)\mathrm{X}_0 + h_{10}(t)X'_0 + h_{01}(t)\mathrm{X}_1 + h_{11}(t)X'_1$$

	expanded	factorized	Bernstein
$h_{00}(t)$	$2t^3 - 3t^2 + 1$	$(1+2t)(1-t)^2$	$B_0(t) + B_1(t)$
$h_{10}(t)$	$t^3 - 2t^2 + t$	$t(1-t)^2$	$\frac{1}{3}B_1(t)$
$h_{01}(t)$	$-2t^3 + 3t^2$	$t^2(3-2t)$	$B_3(t) + B_2(t)$
$h_{11}(t)$	$t^3 - t^2$	$t^2(t-1)$	$-\frac{1}{3}B_2(t)$

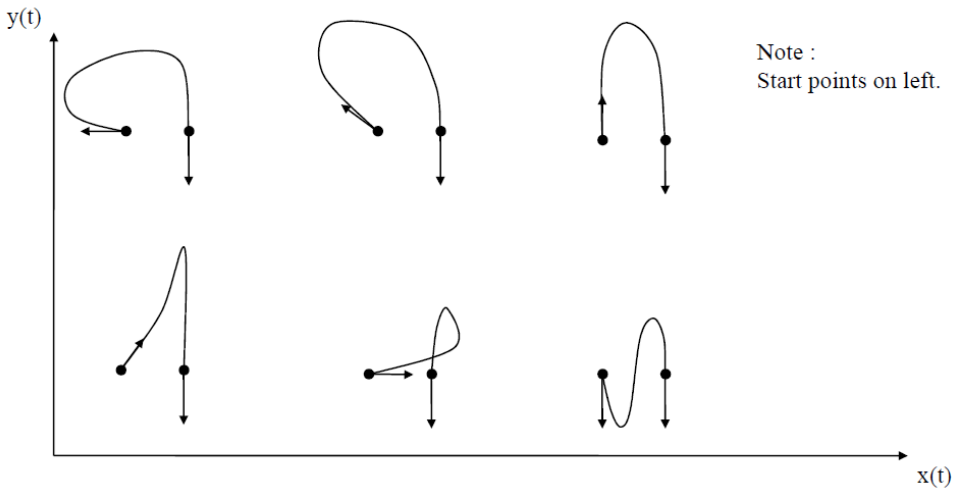
A **Hermite curve** is a **third-order Bézier curve**

Hermit curve

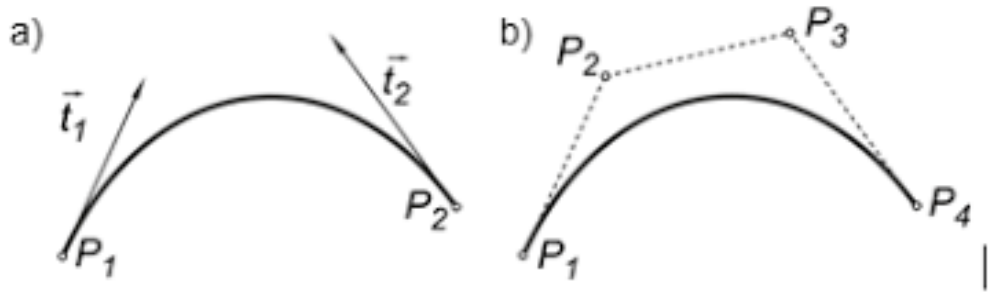
- Cubic polynomial
- Two control points and their derivatives on the two ends

$$X(t) = a_3t^3 + a_2t^2 + a_2t + a_0$$

$$X'(t) = 3a_3t^2 + 2a_2t + a_1$$



A **Hermite curve** is a **third-order Bézier curve**



$$H(t) = h_{00}(t)\mathrm{X}_0 + h_{10}(t)X'_0 + h_{01}(t)\mathrm{X}_1 + h_{11}(t)X'_1$$

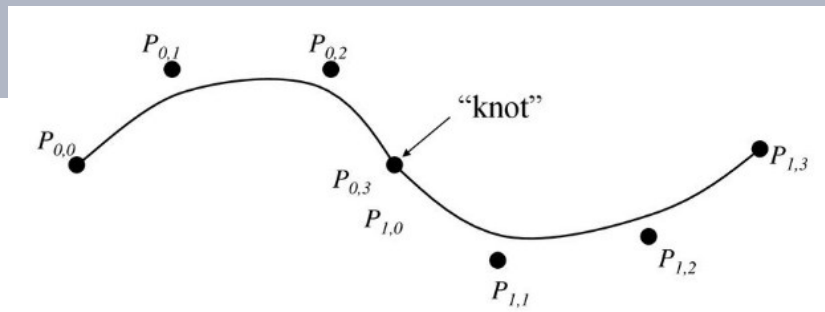
	expanded	factorized	Bernstein
$h_{00}(t)$	$2t^3 - 3t^2 + 1$	$(1+2t)(1-t)^2$	$B_0(t) + B_1(t)$
$h_{10}(t)$	$t^3 - 2t^2 + t$	$t(1-t)^2$	$\frac{1}{3}B_1(t)$
$h_{01}(t)$	$-2t^3 + 3t^2$	$t^2(3-2t)$	$B_3(t) + B_2(t)$
$h_{11}(t)$	$t^3 - t^2$	$t^2(t-1)$	$-\frac{1}{3}B_2(t)$

B-Splines and Basis

Bezier curve: $p(t) = b(t)^T P$

In a piece: Every control point effects the entire piece

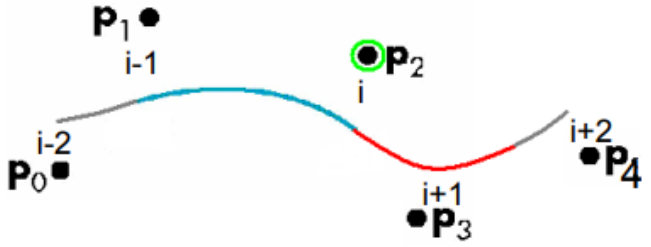
Between pieces: no influences at all

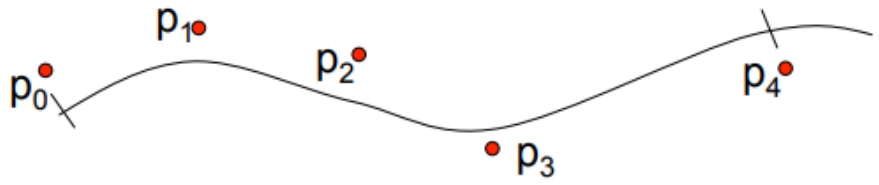


$$P = \{\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_m\}$$

B-Splines curve:

$$\mathbf{b}(t) = \sum_{i=0}^{m} N_{i,n}(t)\mathbf{P}_i$$





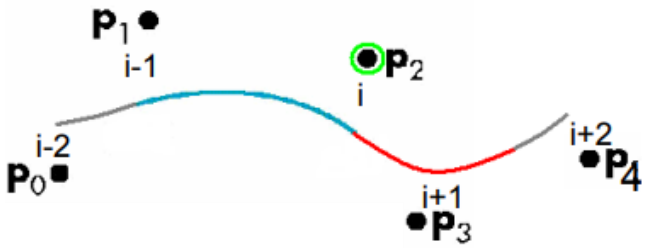
- B-Spline: a parameterized Bezier curve
- All B-spline basis functions are supposed to have their domain on $[u_0, u_m]$.
- We use $u_0 = 0$ and $u_m = 1$ so that the domain is the closed interval $[0,1]$.

B-Splines and Basis

B-Splines curve: a parameterized Bezier curve

$$\mathbf{b}(t) = \sum_{i=0}^{m} N_{i,n}(t)\mathbf{P}_i$$

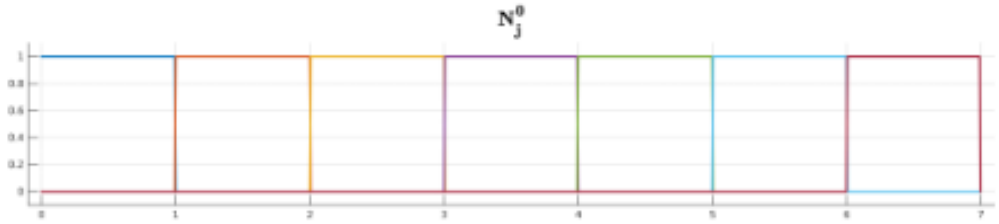
$$P = \{\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_m\}$$



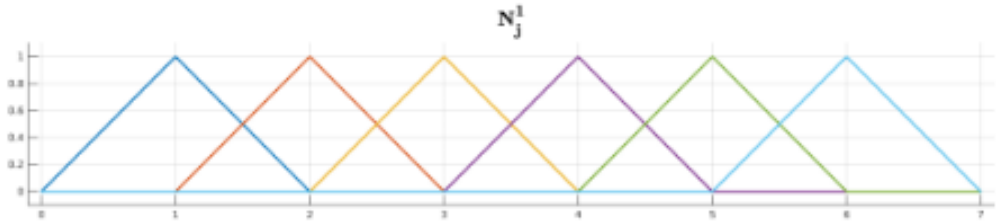
$$N_{i,0}(t) = \begin{cases} 1, & t_i \le t < t_{i+1}, \\ 0, & \text{otherwise}, \end{cases}$$

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

- $N_{i,n}(t)$ is the Basis Function of Node I, it is an n-order polynomial
- Basis Functions fulfill $\sum_i N_{i,p}(t) = 1, t \in [0,1]$



(a) 0 阶 B 样条基函数



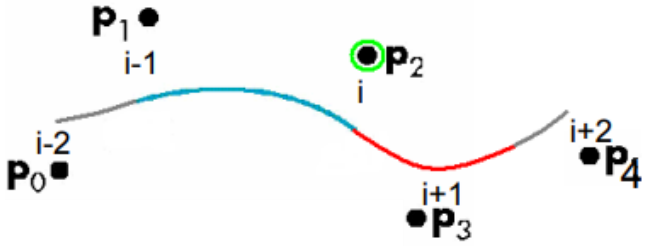
(b) 1 阶 B 样条基函数

B-Splines and Basis

B-Splines curve: a parameterized Bezier curve

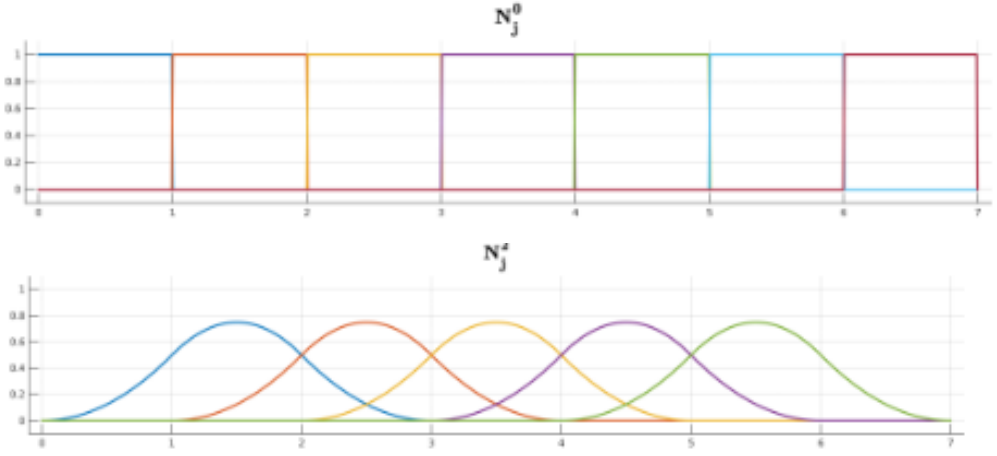
$$\mathbf{b}(t) = \sum_{i=0}^{m} N_{i,n}(t)\mathbf{P}_i$$

$$P = \{\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_m\}$$

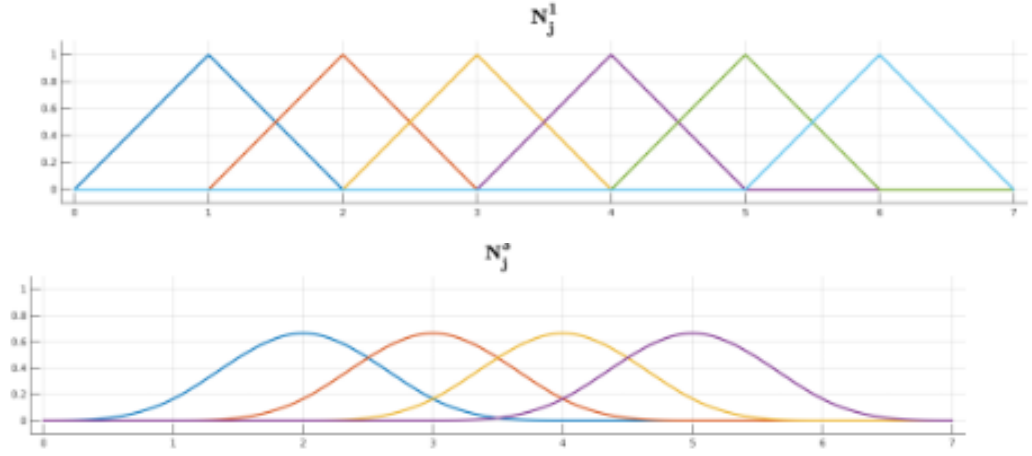


$$N_{i,0}(t) = \begin{cases} 1, & t_i \le t < t_{i+1}, \\ 0, & \text{otherwise}, \end{cases}$$

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



(c) 2 阶 B 样条基函数



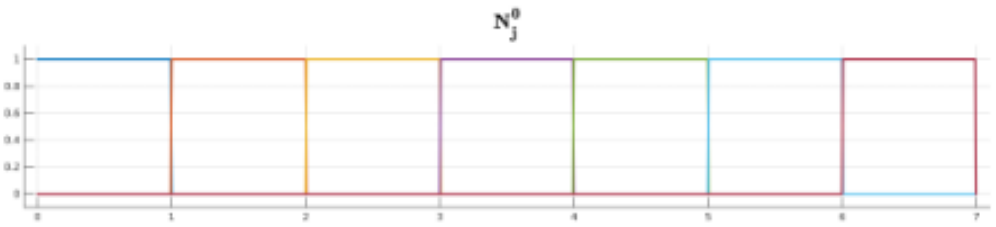
(d) 3 阶 B 样条基函数

B-Spline basis functions from Zero to N

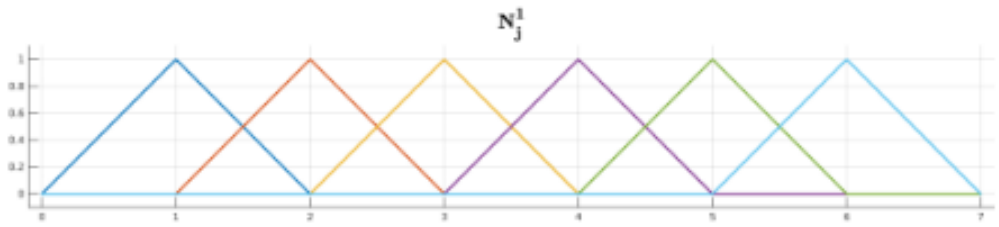
$$N_{i,0}(t) = \begin{cases} 1, & t_i \le t < t_{i+1}, \\ 0, & \text{otherwise}, \end{cases}$$

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

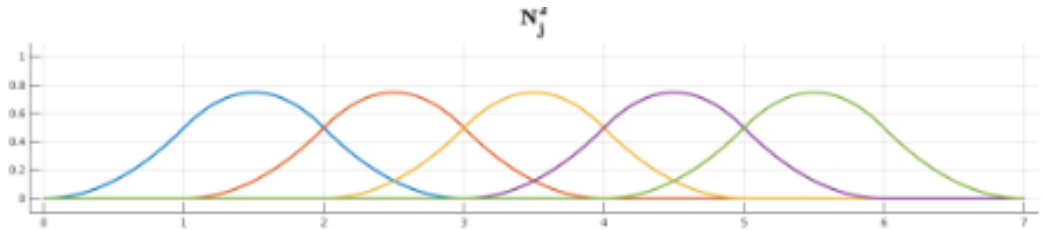
- The higher order is computed recursively (Cox de Boor recursion)



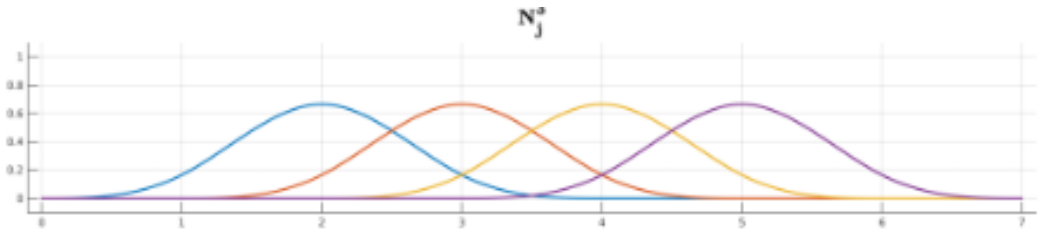
(a) 0 阶 B 样条基函数



(b) 1 阶 B 样条基函数

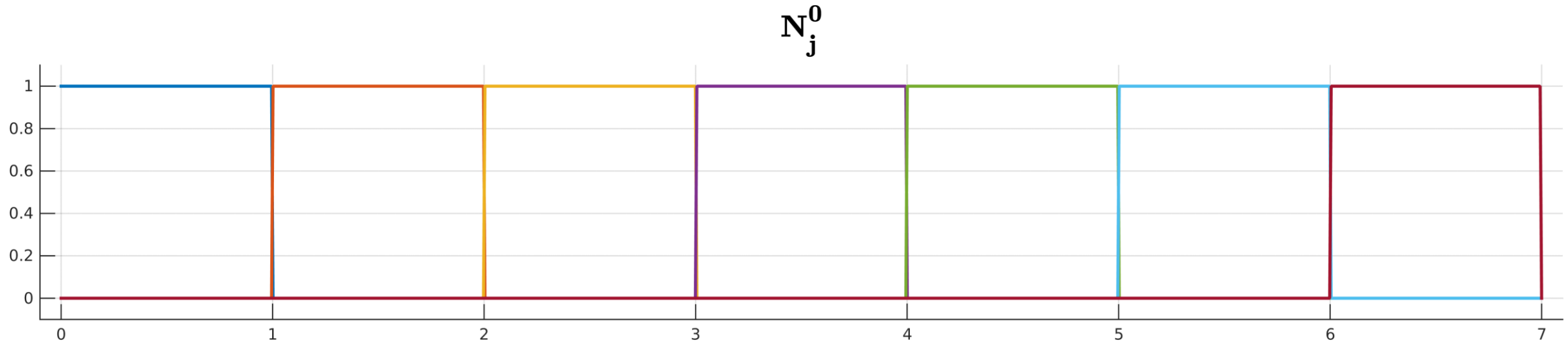


(c) 2 阶 B 样条基函数



(d) 3 阶 B 样条基函数

B-Spline basis functions from Zero to N

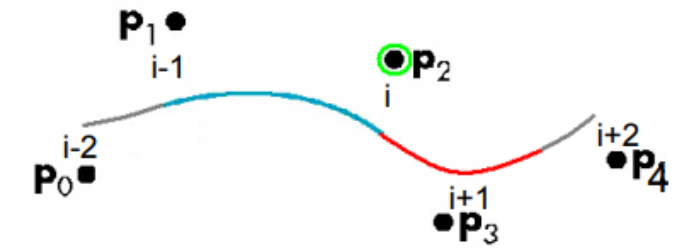


B-spline basis functions up to degree 5 for the knot sequence (0,1,2,3,4,5,6,7).

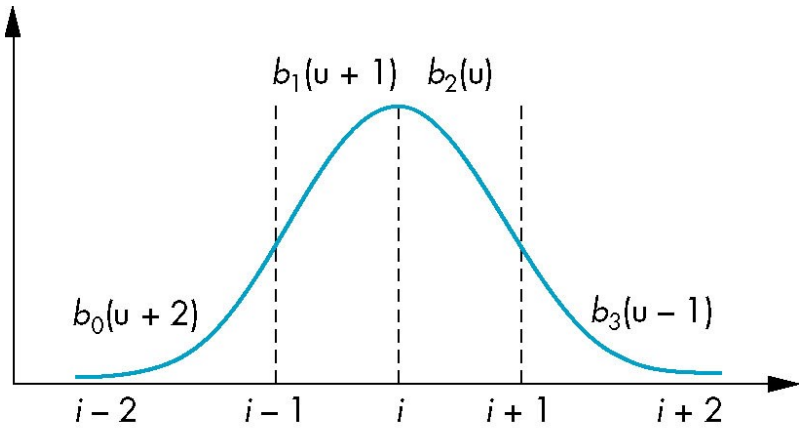
Cubic B-spline basis functions

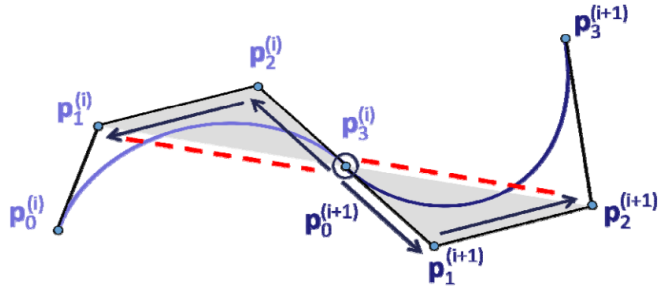
$$N_{i,0}(t) = \begin{cases} 1, & t_i \le t < t_{i+1}, \\ 0, & \text{otherwise}, \end{cases}$$

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

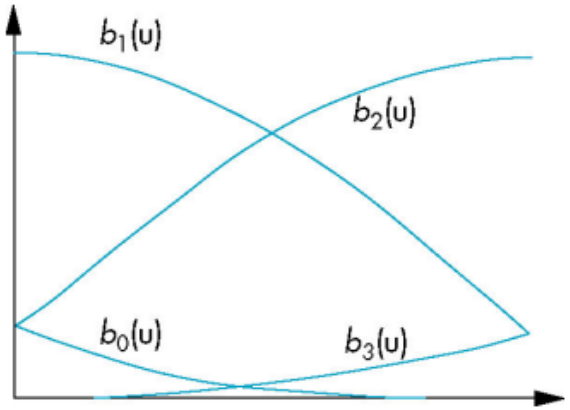


$$u = \frac{t - t_i}{t_{i+1} - t_i}$$





$$p(u) = u^T \mathbf{M}_S p = \boldsymbol{b}(u)^T p$$



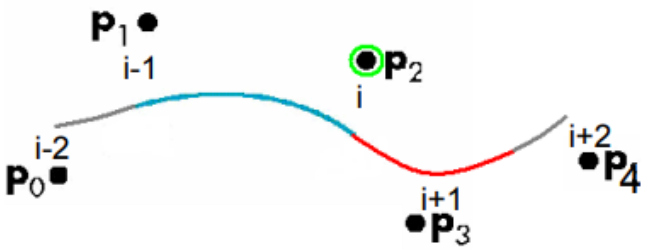
$$\mathbf{b}(u) = \frac{1}{6} \begin{bmatrix} (1-u)^3 \\ 4 - 6u^2 + 3u^3 \\ 1 + 3u + 3u^2 - 3u^3 \\ u^3 \end{bmatrix}$$

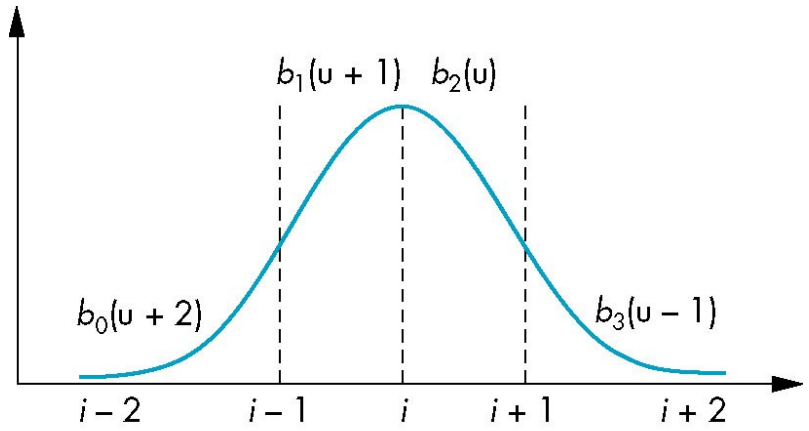
$$\mathbf{M}_S = \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

Cubic B-spline basis functions

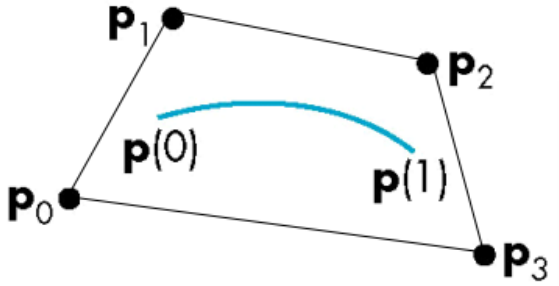
$$N_{i,0}(t) = \begin{cases} 1, & t_i \le t < t_{i+1}, \\ 0, & \text{otherwise}, \end{cases}$$

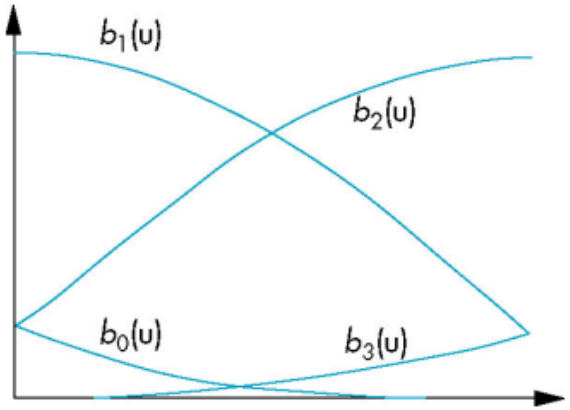
$$N_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t),$$



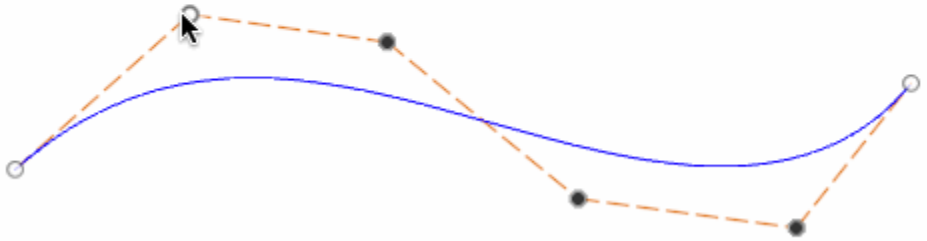


convex hull property





$$\mathbf{b}(u) = \frac{1}{6} \begin{bmatrix} (1-u)^3 \\ 4 - 6u^2 + 3u^3 \\ 1 + 3u + 3u^2 - 3u^3 \\ u^3 \end{bmatrix}$$



Cubic B-Spline

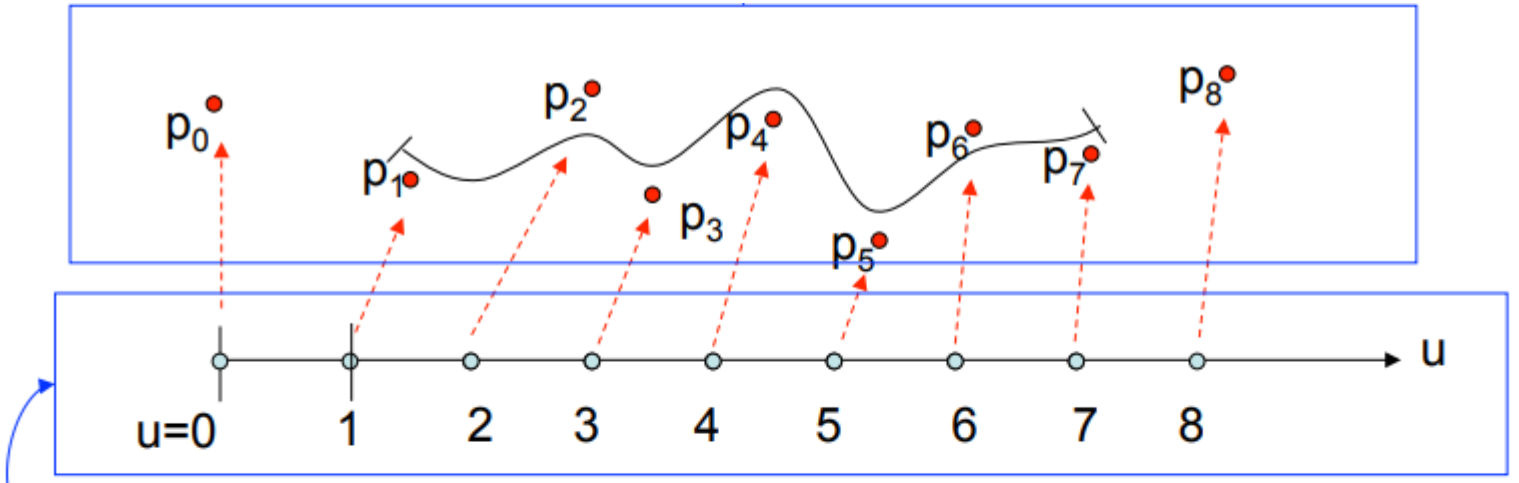
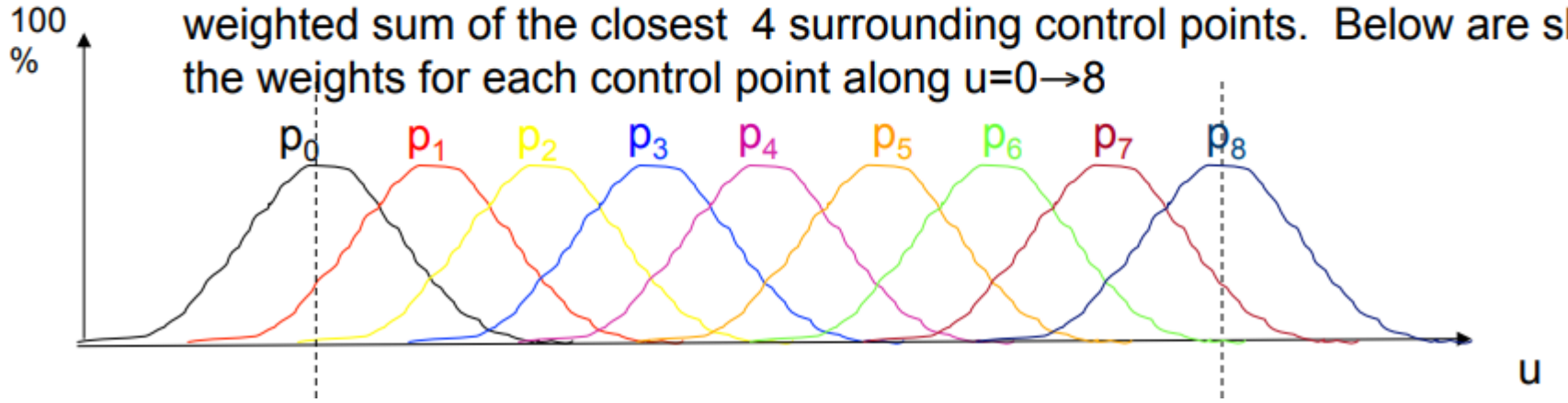


Illustration of how the control points are evenly (uniformly) distributed along the parameterisation u of the curve p(u).

In each point p(u) of the curve, for a given u, the point is defined as a weighted sum of the closest 4 surrounding control points. Below are shown the weights for each control point along u=0→8



Cubic B-Spline

- Basis splines: use the data at $p = [p_{i-2}, p_{i-1}, p_i, p_{i+1}]^T$, to define curve only between p_{i-1} and p_i
- Allows us to apply more continuity conditions to each segment
- C^{n-1} continuity, for cubics, we can have C^2 continuity (function, first and second derivatives at join points)
- Sliding window of p and M:

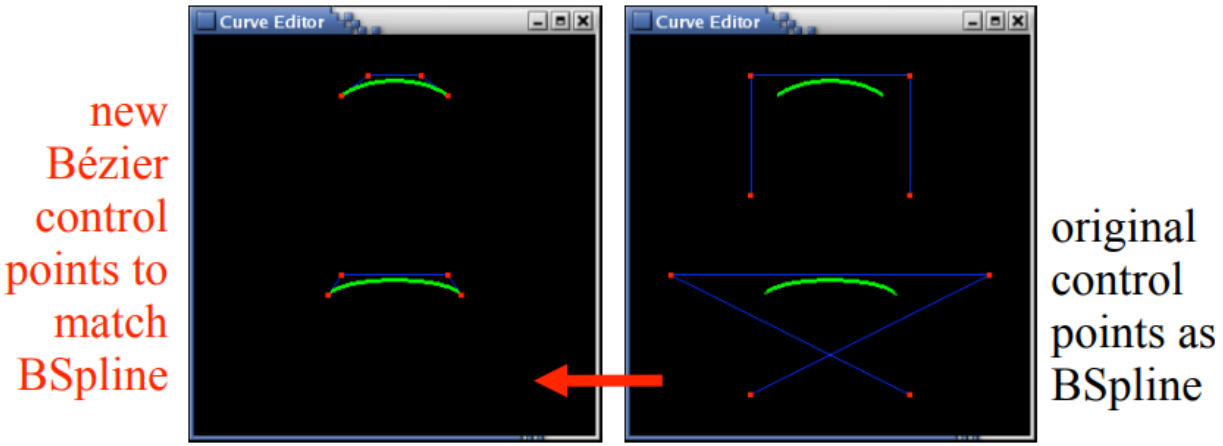
$$p(u) = u^T \mathbf{M}_S p = \boldsymbol{b}(u)^T p$$

Add one new point each time.

- In each piece, we can also write it as a cubic Bézier curve

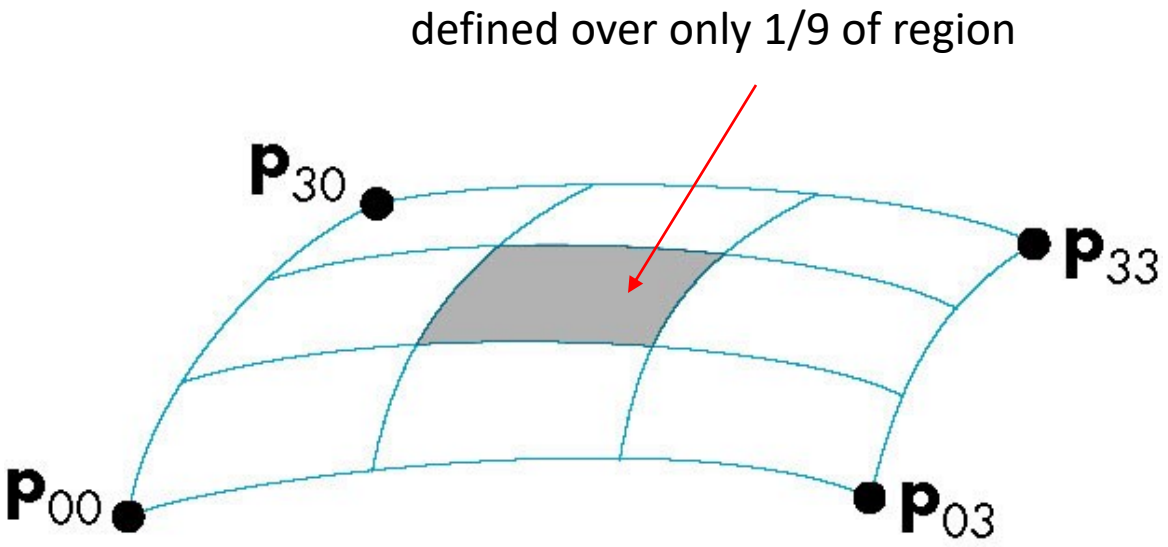
$$p(u) = u^T M_B P_B = u^T M_S P_S$$

$$P_B = (M_B)^{-1} M_S P_S$$



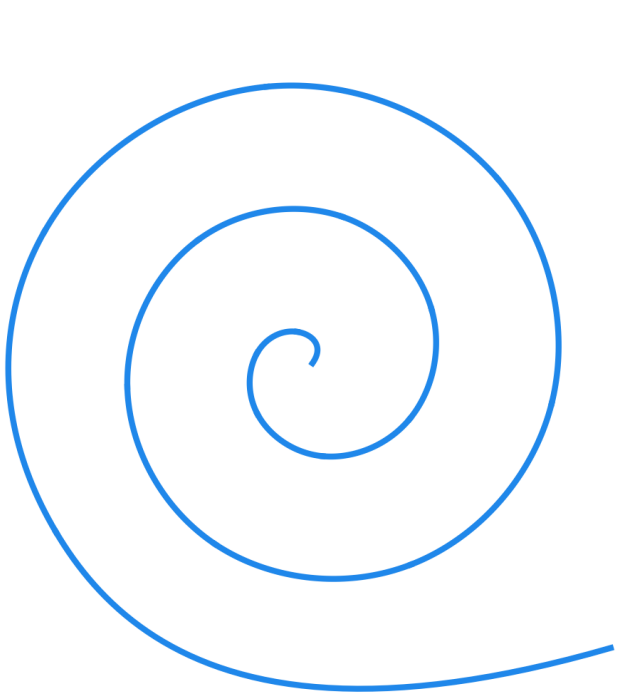
Cubic B-Spline Patches

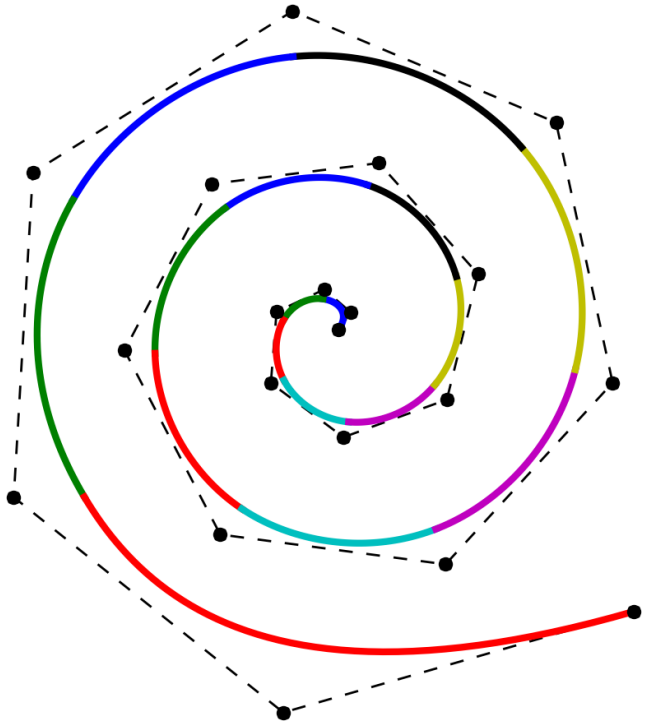
$$p(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} b_i(u)\, b_j(v)\, p_{ij} = u^T \mathbf{M}_S \mathbf{P} \mathbf{M}_S^T v$$



An Example

1. How to achieve this?
2. Again, how would you draw B-Spline curve
   (De Casteljau‘s algorithm possible)?





A cubic B-spline with 16 segments and endpoint interpolation

The following is for your own reading if interested …. Apparently there are more materials out there.

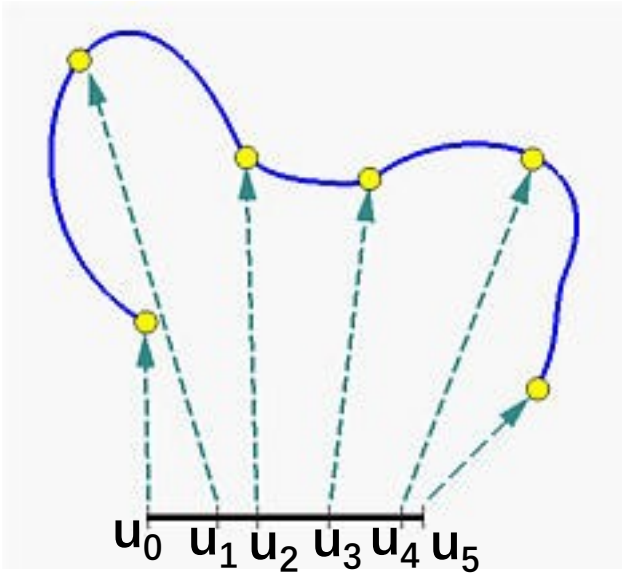
Generalizing Splines

- Let *U* be a set of *m* + 1 non-decreasing numbers, $u_0 <= u_2 <= u_3 <= ... <= u_m$. The u_i's are called ***knots***,

$$U = \{u_0, u_1, u_2, \ldots, u_m\}$$

- The set *U* is the ***knot vector.***

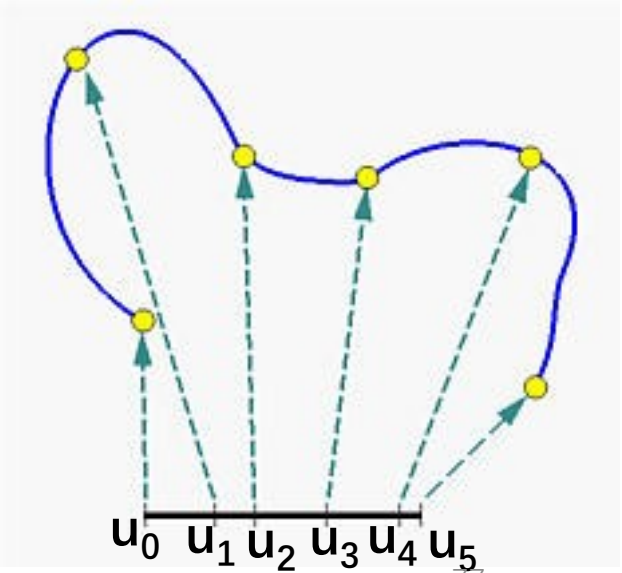
- The half-open interval $[u_i, u_{i+1})$ is the *i-th* ***knot span***.
- Some u_i's may be equal, some knot spans may not exist.



Generalizing Splines

- If a knot u_i appears k times (i.e., $u_i = u_{i+1} = \ldots = u_{i+k-1}$), where $k > 1$, u_i is a ***multiple knot*** of multiplicity k, written as $u_i(k)$.
- If u_i appears only once, it is a ***simple knot***.
- If the knots are equally spaced (i.e., $u_{i+1} - u_i$ is a constant for $0 <= i <= m - 1$), the knot vector or the knot sequence is said ***uniform***; otherwise, it is ***non-uniform***.

$$U = \{u_0, u_1, u_2, \ldots, u_m\}$$



Generalizing Splines

- We can extend to splines of any degree
- Data and conditions to not have to be given at equally spaced values (the *knots*)
  - Nonuniform and uniform splines
  - Can have repeated knots
  - Can force spline to interpolate points

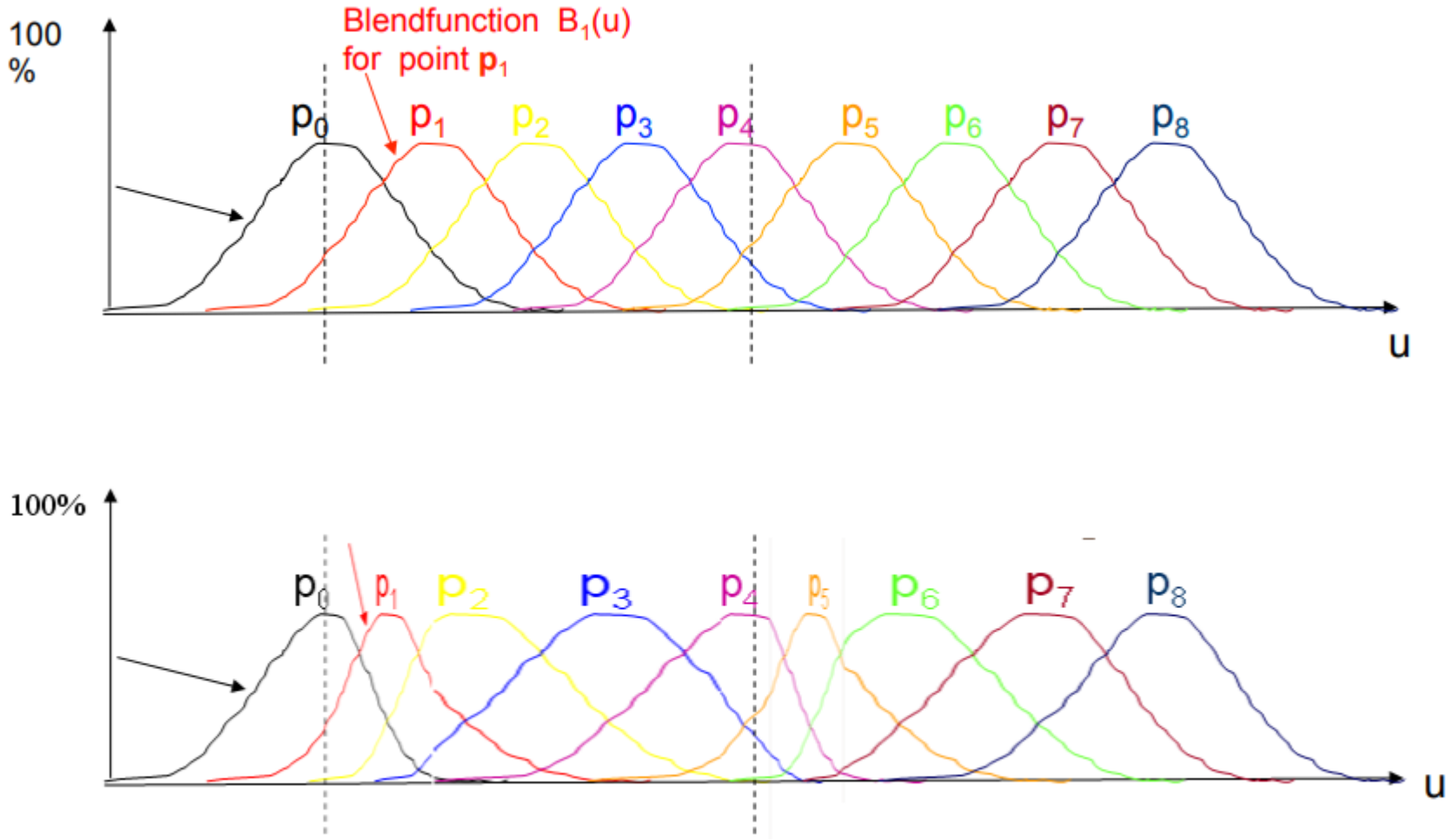
Non-Uniform B-Splines

- The knot vector {0,1,2,3,4,5} is uniform:

$$t_{i+1} - t_i = t_{i+2} - t_{i+1}$$

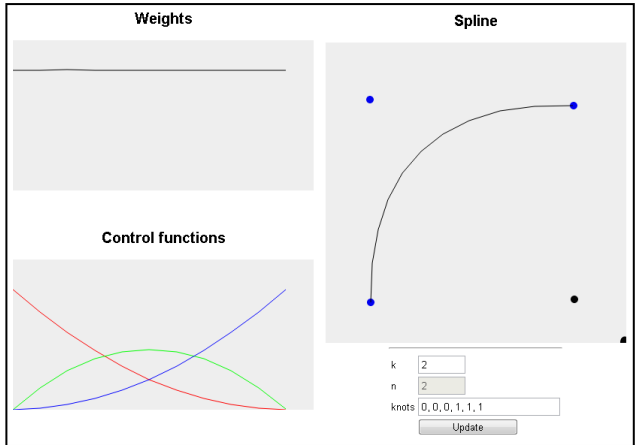
- Varying the size of an interval changes the parametric-space distribution of the weights assigned to the control functions.
- Repeating a knot value reduces the continuity of the curve in the affected span by one degree.
- Repeating a knot k+1 times will lead to a control function being influenced only by that knot value; the spline will pass through the corresponding control point with C0 continuity.

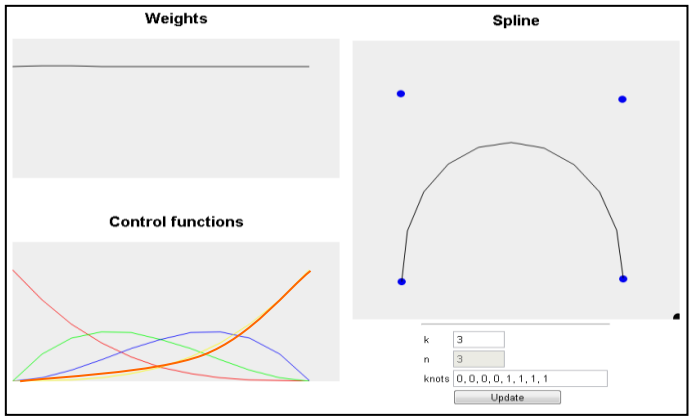
Non-Uniform B-Splines



Open vs *Closed*

- A knot vector which repeats its first and last knot values *k+1* times is called *closed*, otherwise *open*.
  - Repeating the knots *k+1* times is the only way to force the curve to pass through the first or last control point.
  - Without this, the functions $N_{0,k}$ and $N_{n,k}$ which weight P_0 and P_n would still be 'ramping up' and not yet equal to one at the first and last t_i.
- Two examples:
  - *k*=2, *n*+1=3 control points, knots={0,0,0,1,1,1}
  - *k*=3, *n*+1=4 control points, knots={0,0,0,0,1,1,1,1}





Non-uniform Rational Basis Spline(NURBS)

- NURBS is an extended B-spline
- It gives each control point an additional weight w_i
- The weight function becomes $\frac{B_{i,n}(t)*w_i}{\sum B_{j,n}(t)*w_j}$
  - $B_{i,n}$ is the weight function of B-spline
  - It's a rational function
  - If all $w_i = 1$, the denominator becomes $\sum B_{j,n}(t) = 1$, NURBS degenerate to B-spline

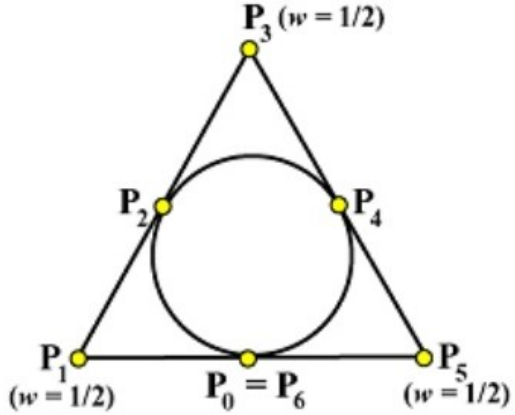
Non-Uniform Rational B-Splines

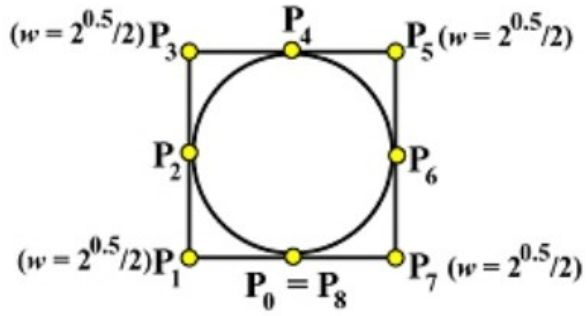
- Recall: $[x, y, z, \omega]_H \rightarrow [x / \omega, y / \omega, z / \omega]$
  - Or: $[x, y, z, 1] \rightarrow [x\omega, y\omega, z\omega, \omega]_H$
- The control point

  $P_i=(x_i, y_i, z_i)$

  becomes the homogeneous control point

  $P_{iH}=(x_i\omega_i, y_i\omega_i, z_i\omega_i)$



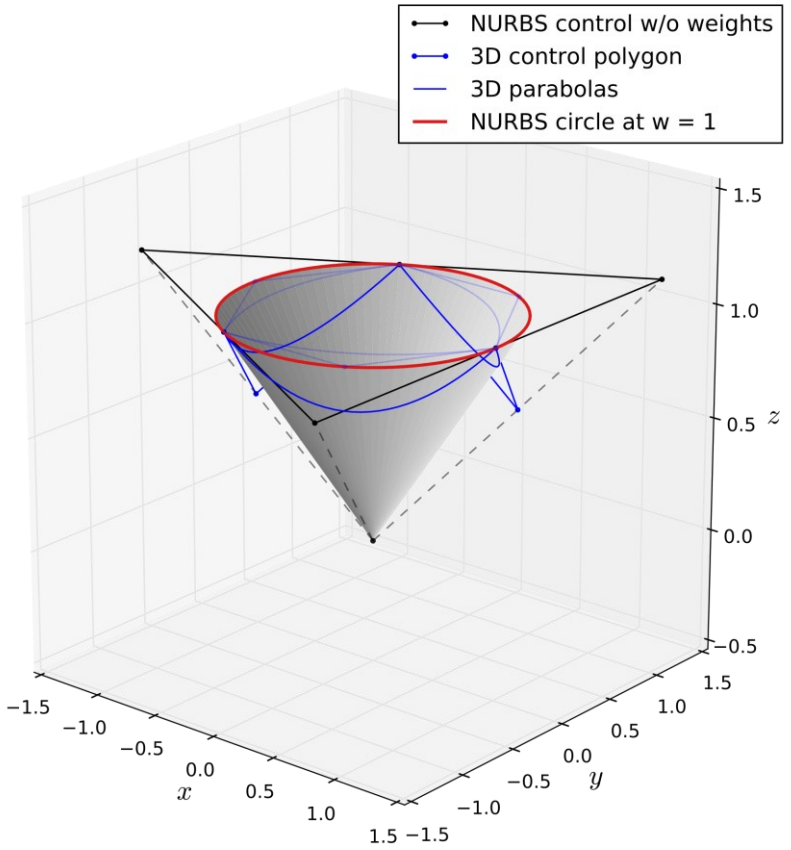


- A NURBS in homogeneous coordinates is:

$$P_H(t) = \sum_{i=0}^{n} N_{i,k}(t) P_{iH}, \quad t_{\min} \le t < t_{\max}$$

NURBS to draw circle (from wiki)

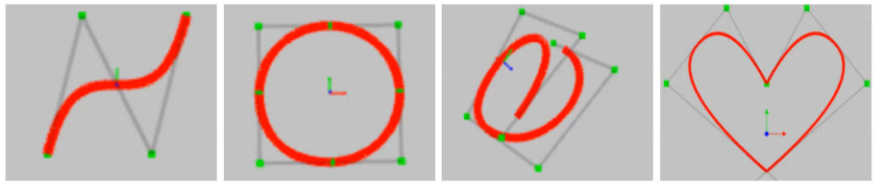
x	y	z	Weight
1	0	0	1
1	1	0	$\frac{\sqrt{2}}{2}$
0	1	0	1
-1	1	0	$\frac{\sqrt{2}}{2}$
-1	0	0	1
-1	-1	0	$\frac{\sqrt{2}}{2}$
0	-1	0	1
1	-1	0	$\frac{\sqrt{2}}{2}$
1	0	0	1



Knot Vector:

$$\{0, 0, 0, \pi/2, \pi/2, \pi, \pi, 3\pi/2, 3\pi/2, 2\pi, 2\pi, 2\pi\}$$

Non-Uniform Rational B-Splines in action



Select the type of the curve NURBS

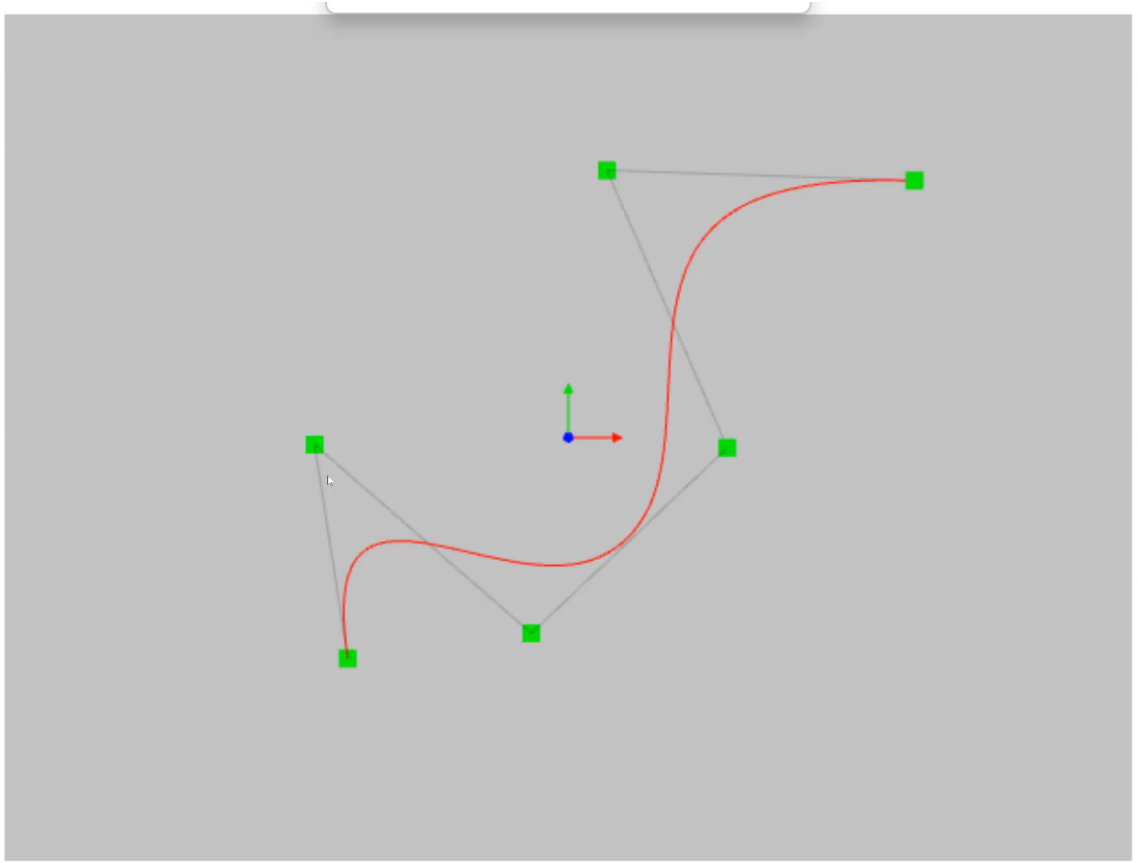
Degree : 3 ?

Control Points : x, y, z, w ?

Point1 :	-4	-4	0	1
Point2 :	-4.603	-0.128	0	1
Point3 :	-0.671	-3.548	0	1
Point4 :	2.886	-0.186	0	1
Point5 :	0.889	6.13	-4.92	1
Point6 :	7.084	5.252	-2.365	1

Add Remove

Knots : ?



Build with http://nurbscalculator.in/

Non-Uniform Rational B-Splines

- To convert from homogeneous coords to normal coordinates:

$$x_H(t) = \sum_{i=0}^{n} (x_i \omega_i)(N_{i,k}(t)) \qquad x(t) = x_H(t) / \omega(t)$$

$$y_H(t) = \sum_{i=0}^{n} (y_i \omega_i)(N_{i,k}(t)) \qquad y(t) = y_H(t) / \omega(t)$$

$$z_H(t) = \sum_{i=0}^{n} (z_i \omega_i)(N_{i,k}(t)) \qquad z(t) = z_H(t) / \omega(t)$$

$$\omega(t) = \sum_{i=0}^{n} (\omega_i)(N_{i,k}(t))$$

Non-Uniform Rational B-Splines

- A piecewise rational curve is thus defined by:

$$P(t) = \sum_{i=0}^{n} R_{i,k}(t) P_i, \quad t_{\min} \le t < t_{\max}$$

  with supporting *rational basis functions*:

$$R_{i,k}(t) = \frac{\omega_i N_{i,k}(t)}{\sum_{j=0}^{n} \omega_j N_{j,k}(t)}$$

  - This is essentially an average re-weighted by the ω's.
- Such a curve can be made to pass arbitrarily far or near to a control point by changing the corresponding weight.

Tensor products for surfaces

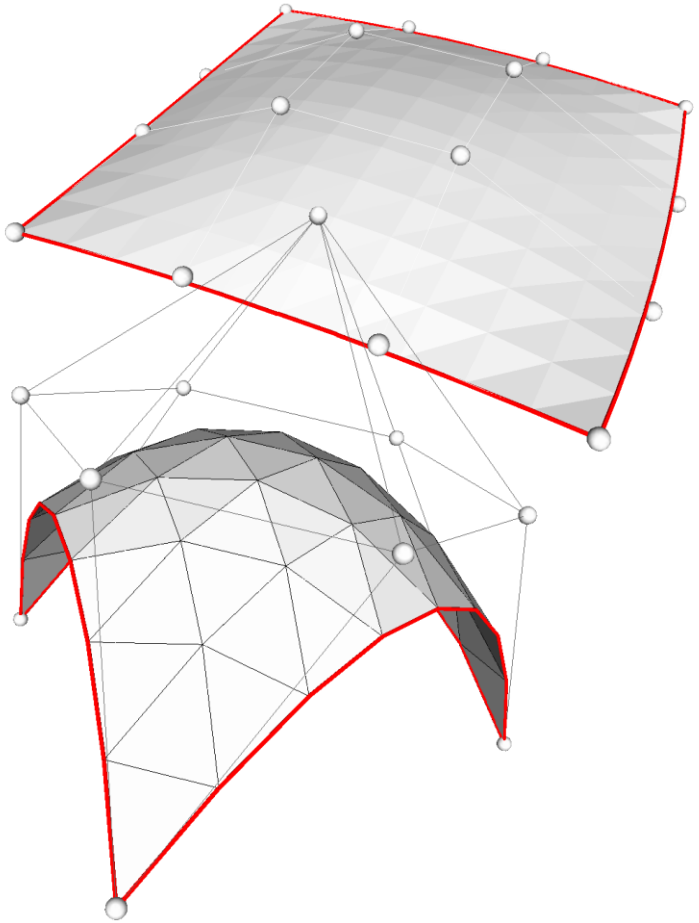
- The *tensor product* of two vectors is a matrix.

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \otimes \begin{bmatrix} d \\ e \\ f \end{bmatrix} = \begin{bmatrix} ad & ae & af \\ bd & be & bf \\ cd & ce & cf \end{bmatrix}$$

- Can also take the tensor of two polynomials.

NURBS surfaces

- The tensor product of the polynomial coefficients of two NURBS splines is a matrix of polynomial coefficients.
  - If curve A has degree *k* and *n+1* control points and curve B has degree *j* and *m+1* control points then A⊗B is an (*n*+1)x(*m*+1) matrix of polynomials of degree *max*(*j,k*).
  - So all you need is (*n+1*)(*m+1*) control points and you've got a rectangular surface patch!
- This approach generalizes to triangles and arbitrary *n*-gons.



References

- Les Piegl and Wayne Tiller, *The NURBS Book*, Springer (1997)
- Alan Watt, *3D Computer Graphics*, Addison Wesley (2000)
- G. Farin, J. Hoschek, M.-S. Kim, ***Handbook of Computer Aided Geometric Design***, North-Holland (2002)

Additional Readings

- **An extremely nice explanation of Bezier curve:**

  https://www.youtube.com/watch?v=aVwxzDHniEw

- **Graphics in 5 Minutes:**

  - Splines in 5 minutes:

    Part1 cubic curves; Part2 Catmull-Rom and Natural Cubic Spline; Part3 B-splines and 2D

    https://www.youtube.com/watch?v=YMl25iCCRew

    https://www.youtube.com/watch?v=DLsqkWV6Cag

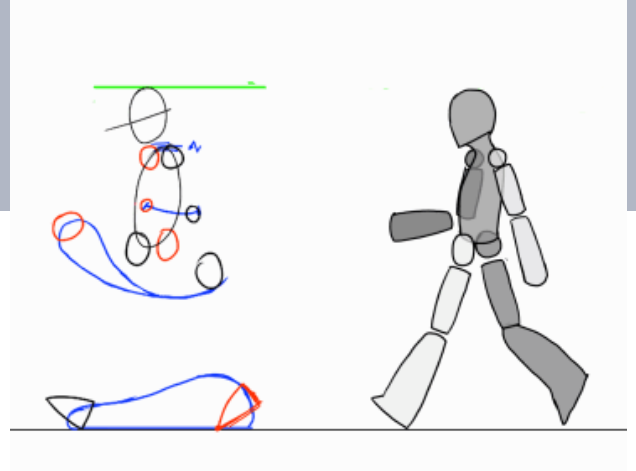
    https://www.youtube.com/watch?v=JwN43QAlF50

- **GAMES102:**

  3-参数曲线拟合； 4-三次样条函数； 5-Bezier曲线； 6-B样条曲线；
  7-NURBS曲线； 8-细分曲线； 9-隐式曲线； 10-NURBS曲面

  http://staff.ustc.edu.cn/~lgliu/Courses/GAMES102_2020/default.html#Resource

Quiz 1



Discuss the advantages and disadvantages of:

Bezier curves, B-splines, natural cubic splines.

(you can use 3rd-order curse as examples)

Specifically, compare their characteristics on continuity and control.

What types of tasks are each of the curves best suited for?

	continuity	control	Type of tasks
Bezier curves			
B-splines			
Natural cubic splines