

Computer Aided Geometric Design

Fall Semester 2024

Introduction

陈仁杰

renjiec@ustc.edu.cn

<http://staff.ustc.edu.cn/~renjiec>

Course Lecturer: 陈仁杰

- Education/Job Experience
 - 2001.9-2005.6, Zhejiang U., Applied Math, B.Sc
 - 2005.9-2010.6, Zhejiang U., Applied Math, Ph.D
 - 2011.3-2013.9, Technion, Israel, Postdoc
 - 2013.9-2015.6, UNC-Chapel Hill, Postdoc
 - 2015.7-2019.6, Max-Planck Institute for Informatics, Senior Researcher
 - 2019.7-now, USTC, Professor
- webpage: <http://staff.ustc.edu.cn/~renjiec>
- email: renjiec@ustc.edu.cn

Teaching Assist

- 吴中昊

wzh2001@mail.ustc.edu.cn

- 吴川

809329975@qq.com

Introduction: Geometric Modeling

- Motivation
- Overview: Topics
- Basic modeling techniques

Motivation

Motivation

This lecture covers two related areas:

- Classical geometric modeling (CAGD)
- Geometry processing

Common techniques (*math*, models, terminology), but different problems

Geometric Modeling

- Start with a blank screen, design a geometric model
- Challenge: mathematical description of shape information
 - Computer friendly
 - User friendly
- Typical techniques:
 - Spline curves & surfaces
 - Constructive solid geometry (CSG)
 - Subdivision surfaces

Geometric Modeling



Geometric Processing

- A (discrete) sampling of the model is readily available
 - Typically: 3D scanner (point cloud)
- Challenge: make sense of large complex, unstructured data
 - Analyze and edit the geometry
- Typical issues
 - Noise removal, filtering
 - Surface reconstruction
 - Analysis (features, symmetry, hole-filling, etc...)
 - Parameterization (mapping textures)
 - Editing, deforming

Examples

Geometric Modeling

The Modern World...



designed on a computer
(the building)

designed on a computer as well
(the cars)

fortunately, not (yet) designed
on a computer (the trees)

Impact of Geometric Modeling

We live in a world designed using CAD

- Almost any man-made structure designed with computers
 - Architecture
 - Commodities
 - Bike, car
 - Spline curves invented in automotive industry
 - Fonts
- Our abilities in geometric modeling shape the world we live in each day

Different Modeling Tasks

Different requirement for different setups



Different Modeling Needs

CAD / CAM

- Precision Guarantees
- Handle geometric constraints exactly (e.g. exact circles)
- Modeling guided by rules and constraints



Different Modeling Tasks

Photorealistic Rendering

- Has to “look” good
- Ad-hoc techniques are ok
- Using textures & shaders to “fake” details
- More complexity, but less rigorous



[Deussen et al: Realistic modeling and rendering of plant ecosystems, SIGGRAPH 1998]

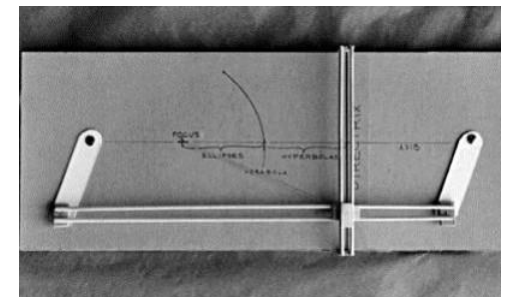
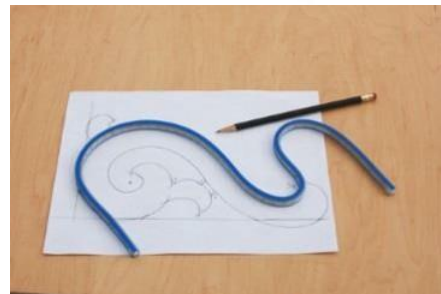
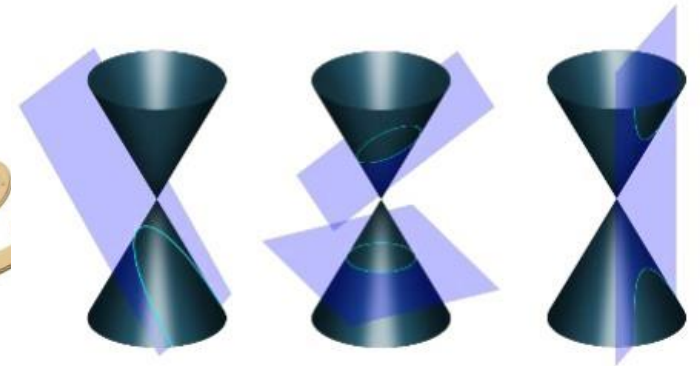
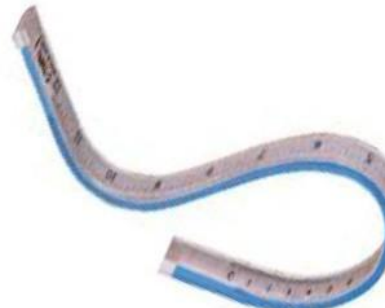
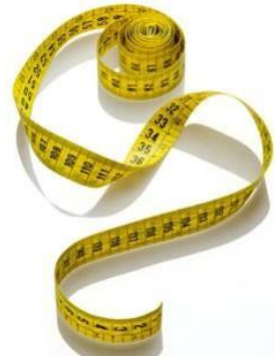
Geometric Modeling

A look back

Modeling the old way

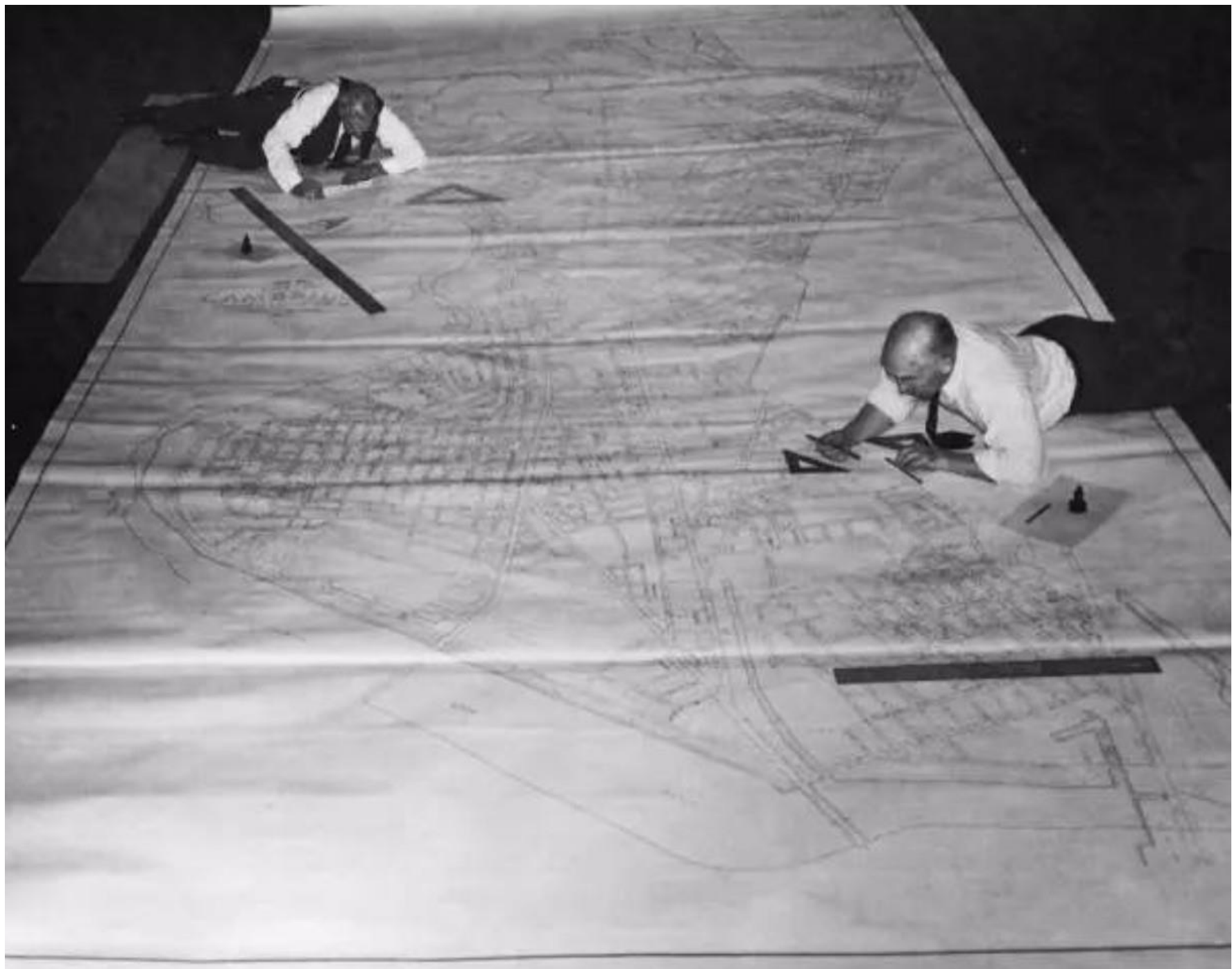
Basic tools

- Measuring and drafting tools











Industrial modeling developments

Industrial modeling: Two distinct shape classes

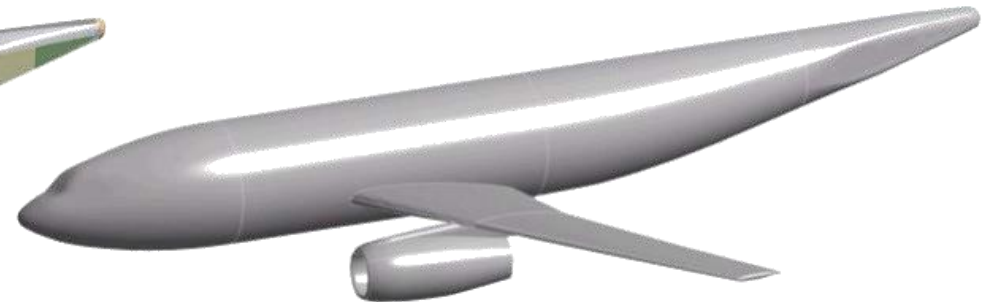
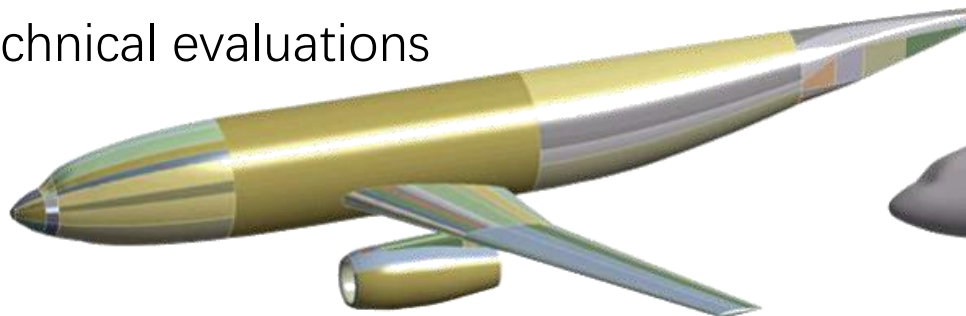
- Complex combination of elementary surfaces
 - Easy to model (blueprint)
 - Easy to produce
 - Easy technical evaluations (volume, moment of inertia)



Industrial modeling developments

Industrial modeling: Two distinct shape classes

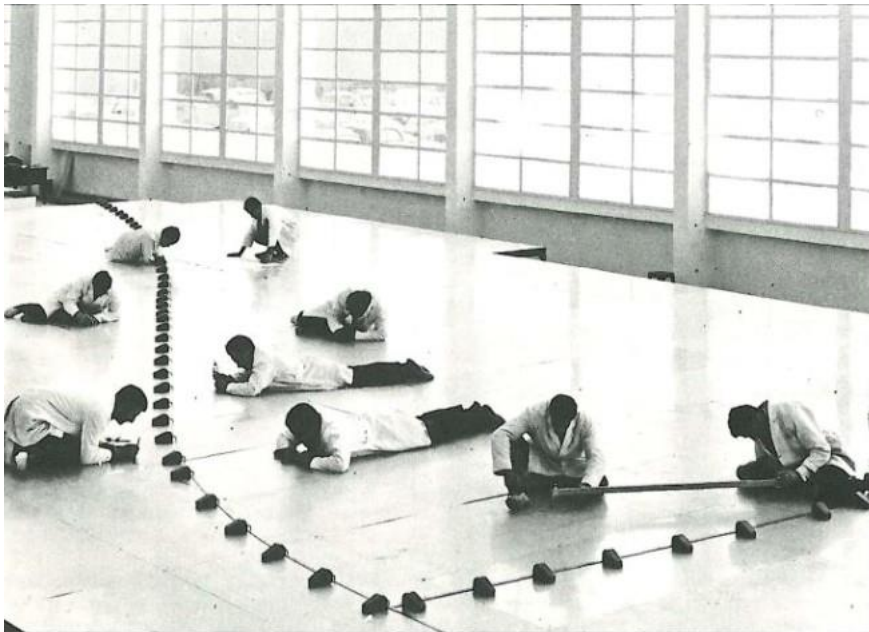
- Complex combination of elementary surfaces
 - Easy to model (blueprint)
 - Easy to produce
 - Easy technical evaluations (volume, moment of inertia)
- Free-form shapes
 - Required mainly by modern industries e.g. aeronautics, shipbuilding, auto industry
 - Not easy to describe mathematically
 - Harder technical evaluations



Early modeling of free-form curves and surfaces

Splines

- Thin flexible band made out of wood, plastic or steel
- Can be held in shape using weights
- Smooth energy minimizing curves

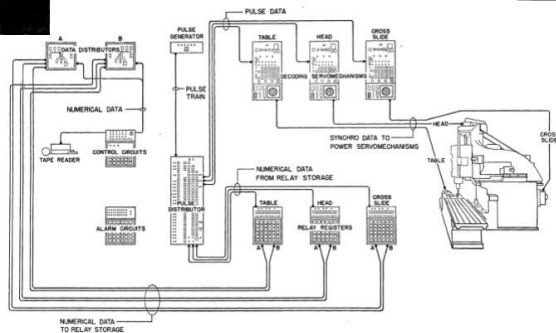
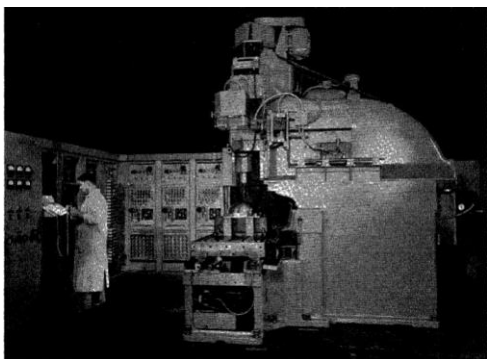


Birth of Computer Aided Design (CAD)

Two major events

Numerical Control: 数字控制机床
铣床

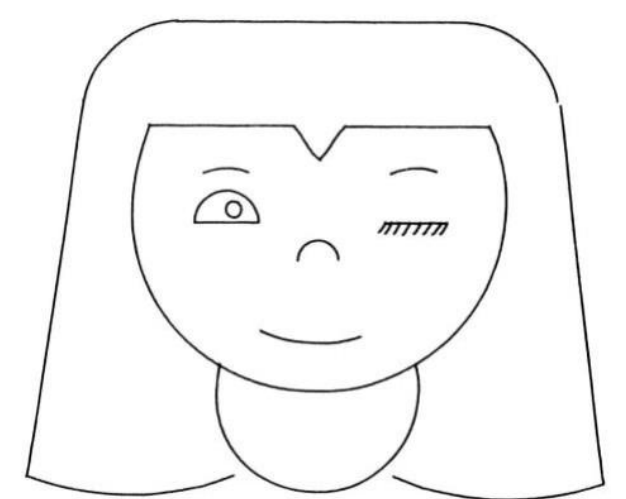
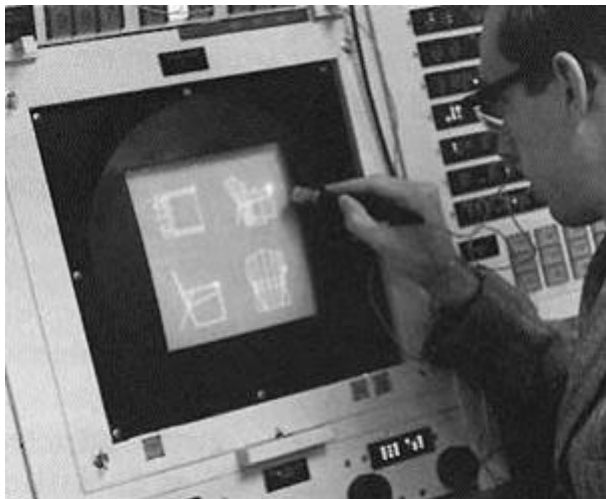
- The world's first NC 3D milling machine (MIT 1951)
 - Shapes can be described mathematically
 - Read shape information from drawing



Birth of Computer Aided Design (CAD)

Two major events

- I. Sutherland sketchpad: A man-machine graphical communication system (MIT 1963)
 - Shape became visible (Not just a formula)
 - Direct interaction with shape



Birth of Computer Aided Design (CAD)

Development of mathematical descriptions of Free form curves and surfaces

- Ferguson curves and surfaces (Boeing 1961)
 - Vector description and use of parameters
- Coon surface patches (MIT 1964)
 - Control through positions and tangents
- de Casteljau Algorithm (Citroën 1959)
- Bézier curves (Renault 1971, UNISURF system)
- B-splines, NURBS, T-splines, **AST, S-splines**...

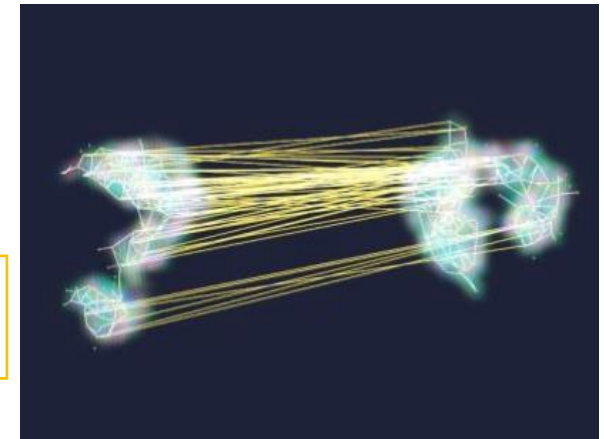
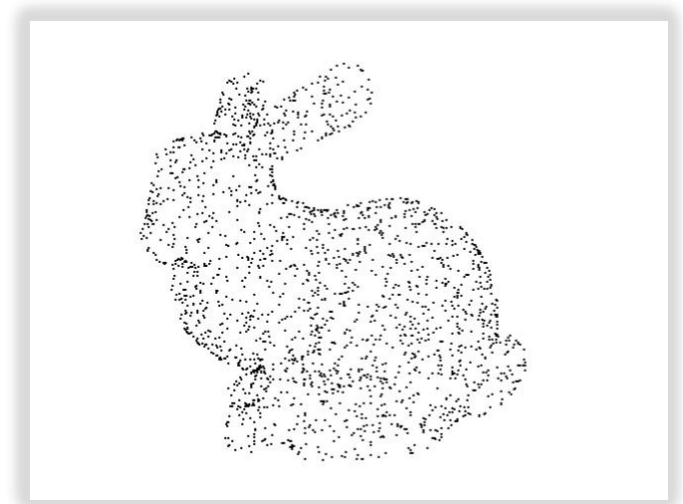
Geometry Processing

Examples

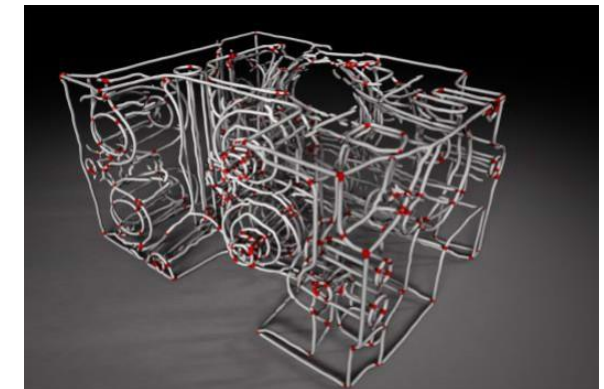
Geometry Processing

A rather new area

- Motivation: 3D scanning
 - 3D scanners
 - Clouds of millions of measurement points
- Sources of spatial data:
 - Science: CT, MRI,...
 - 3D movie making
 - Game / movie industry:
 - Servers with GBs of “polygon soup”
 - Crawl the internet
- Need to process the geometry further



Computed Tomography: 计算机断层扫描
Magnetic Resonance Imaging: 磁共振成像

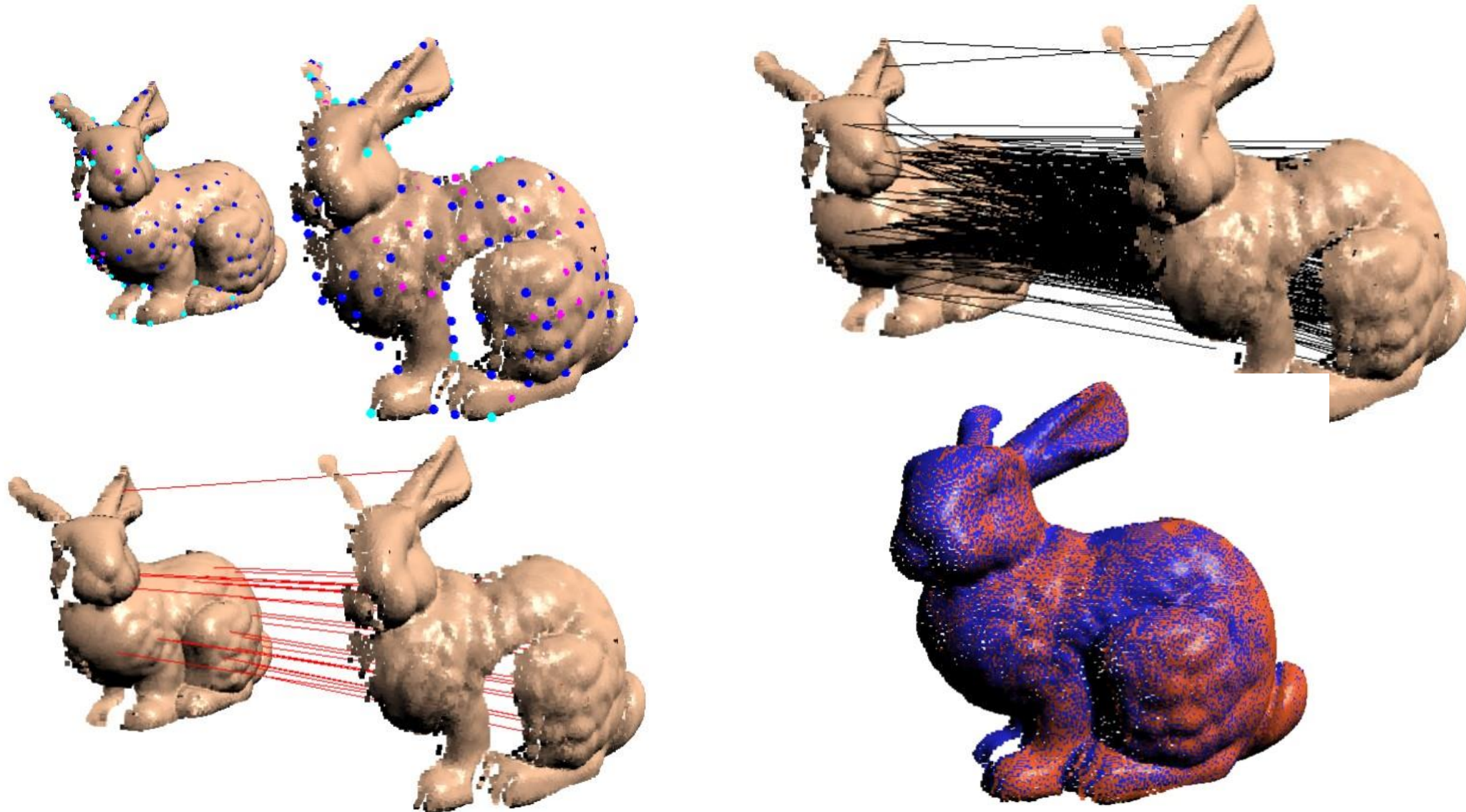


Photoshopping Geometry

Geometry Processing:

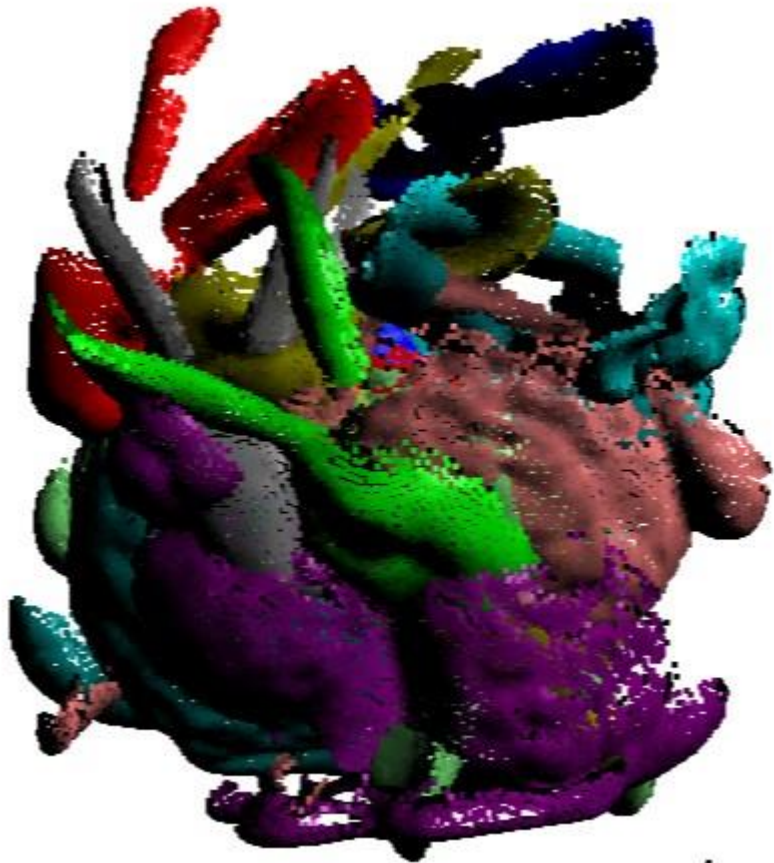
- Cleanup:
 - Remove inconsistencies
 - Make watertight (well defined inside/outside, for 3D printers)
 - Simplify – keep only the main “structure”
 - Remove noise, small holes, etc...
- Touch-up / Edit:
 - Texturing, painting, carving
 - Deformation
 - Stitch together pieces
- Lots of other stuff – similar to image processing

Scan Registration



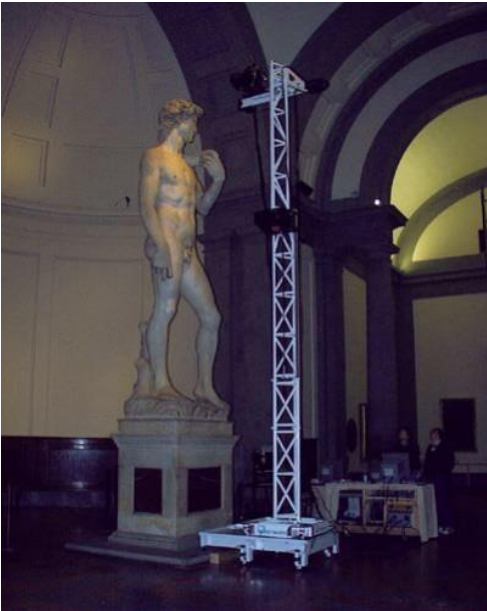
Feature Tracking

Fully Automatic:

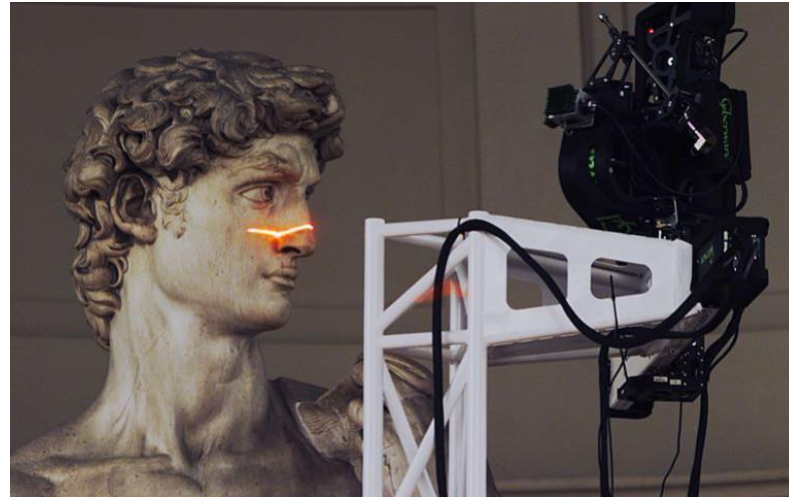


Example

Example: The Stanford “Digital Michelangelo Project”



scanning

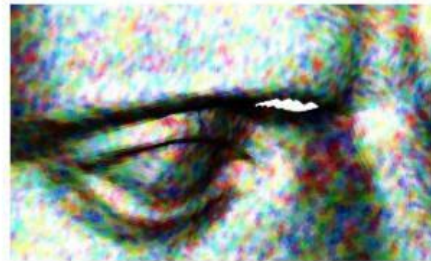


scanning



**rendered
reproduction**

Denoising



Size ~300mm

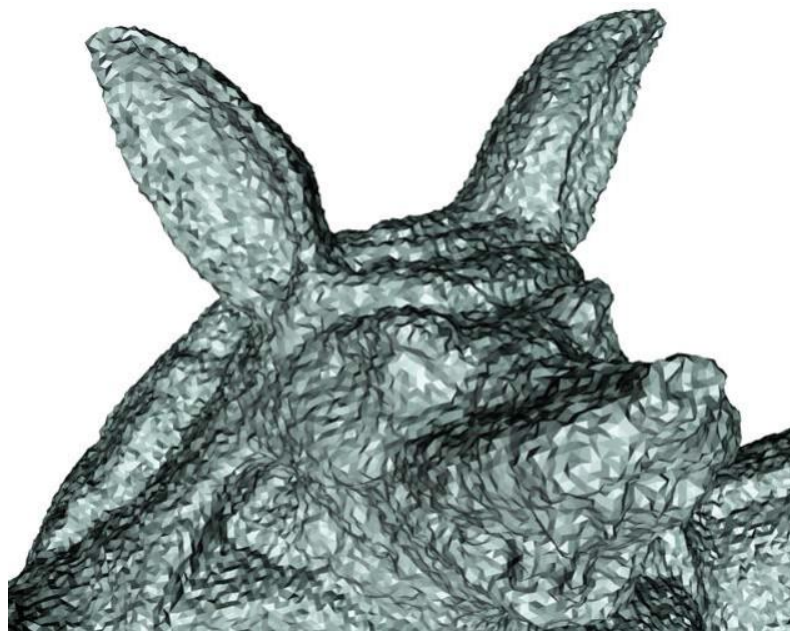
Max. deviation : 0.1mm

Denoising

Remove noise, but preserve features



Original



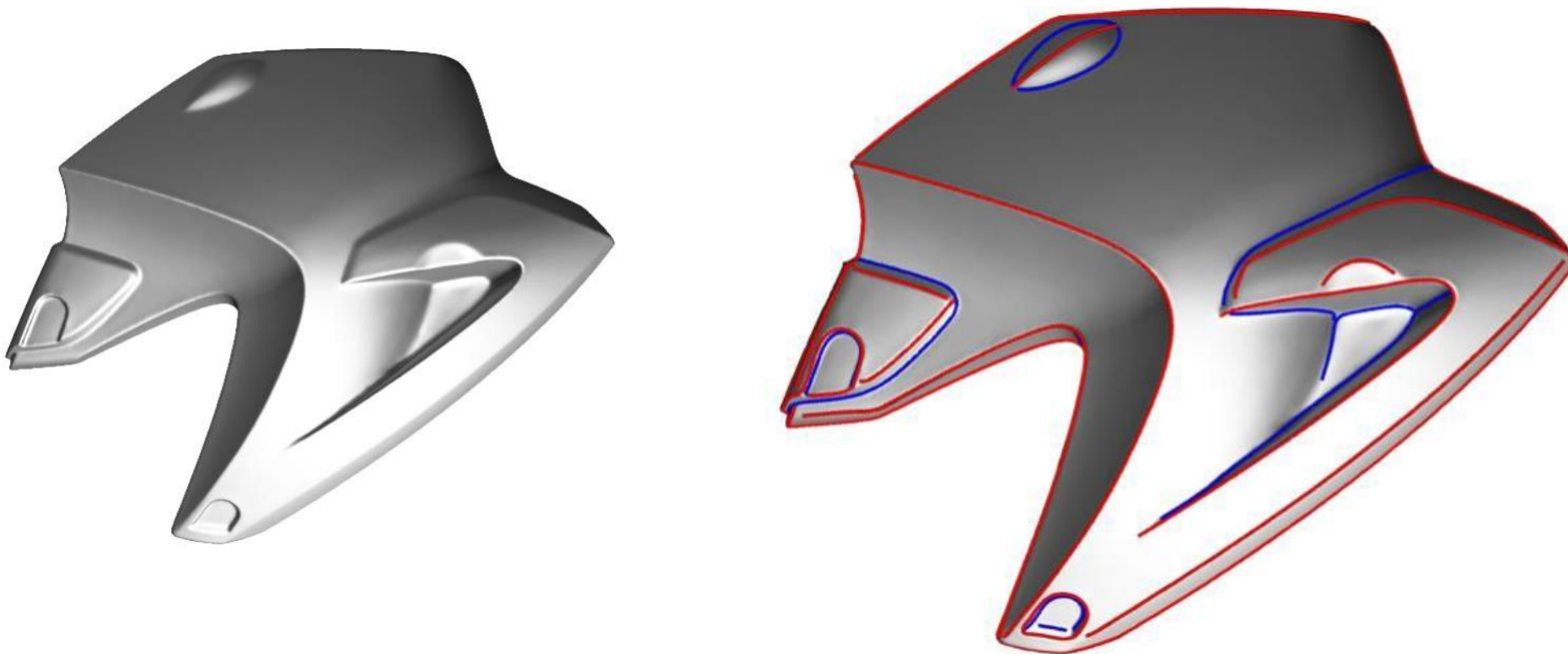
Noise added



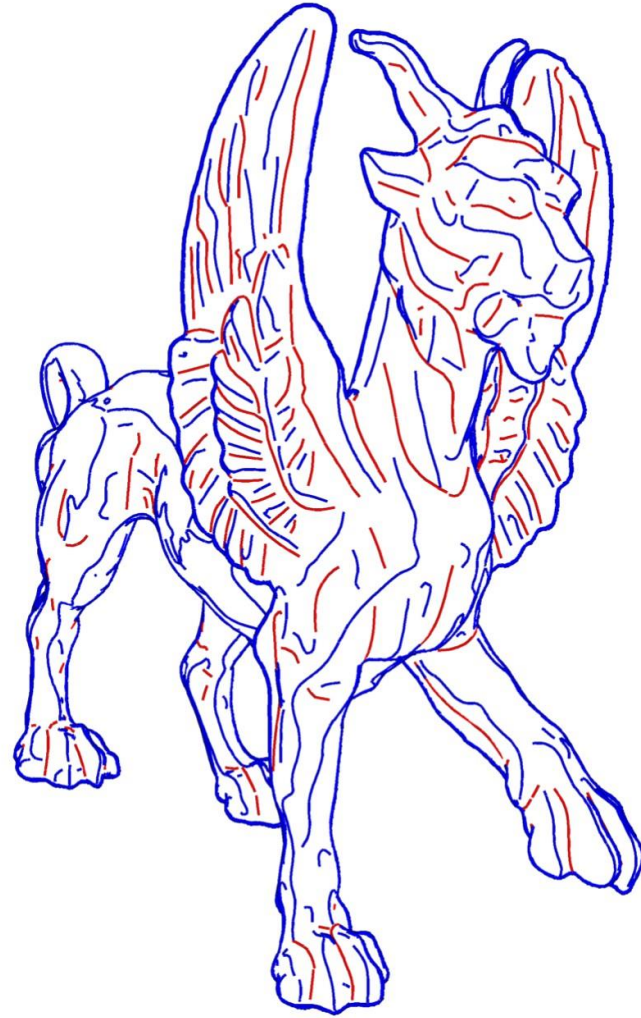
Denoised

Feature Detection

Find curves on a surface that carry visually most prominent characteristics



Feature Detection



Surface Parameterization: Motivation

Texture mapping

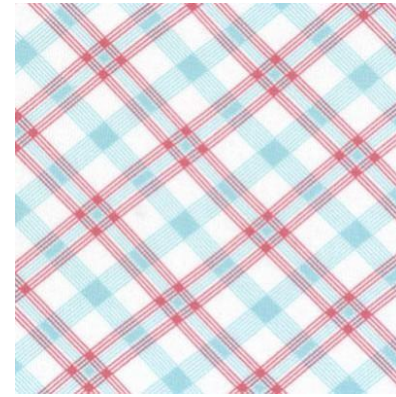
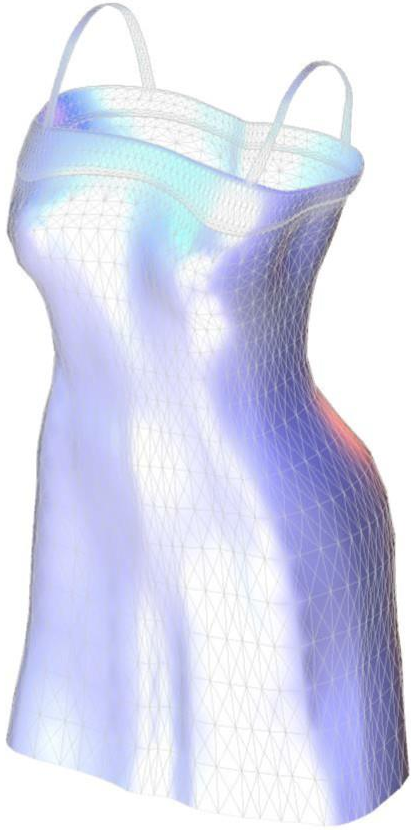


Graphics ingredients



Real world

Texture mapping example



Surface Parameterization



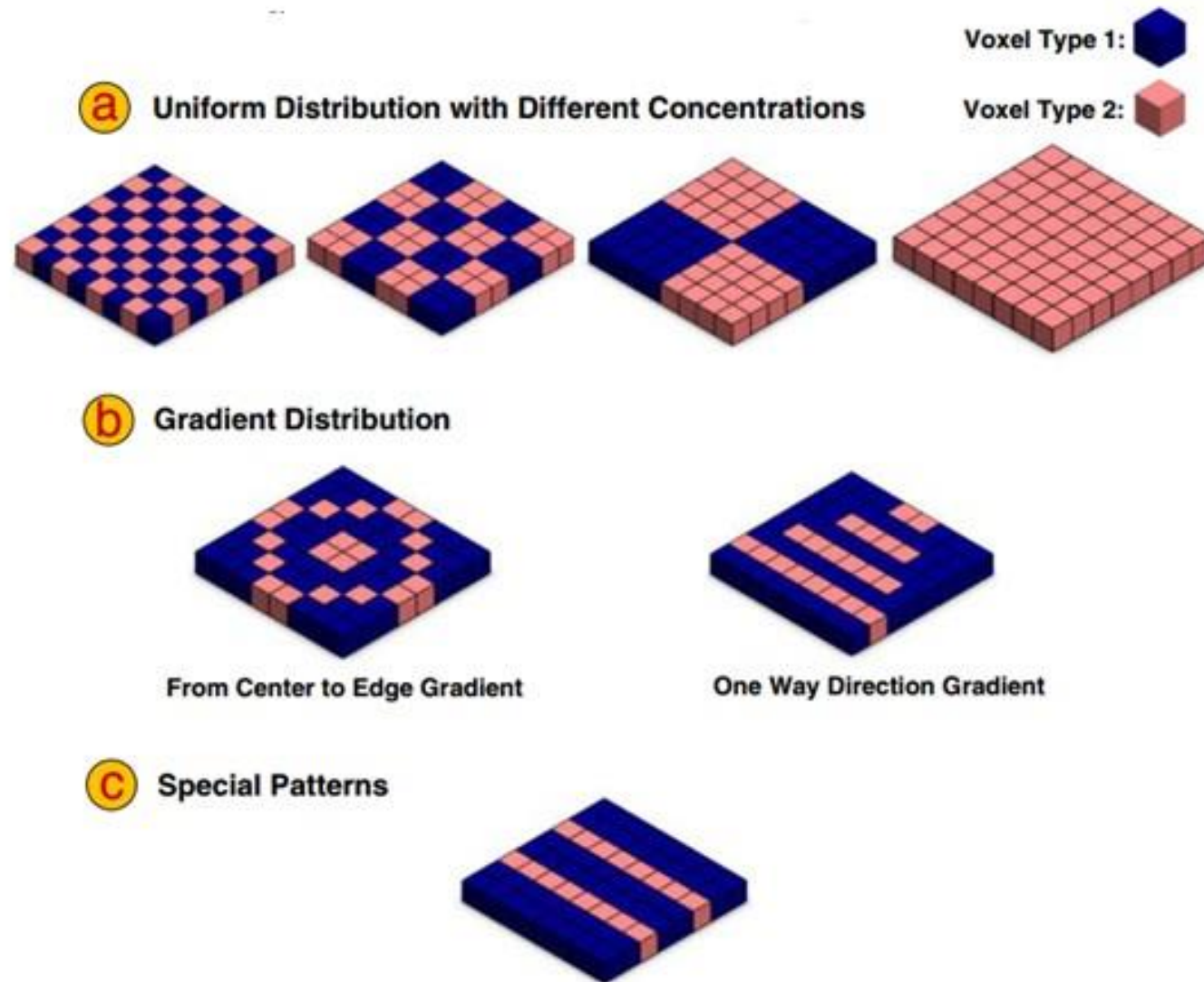
Surface Parameterization



Latest Development

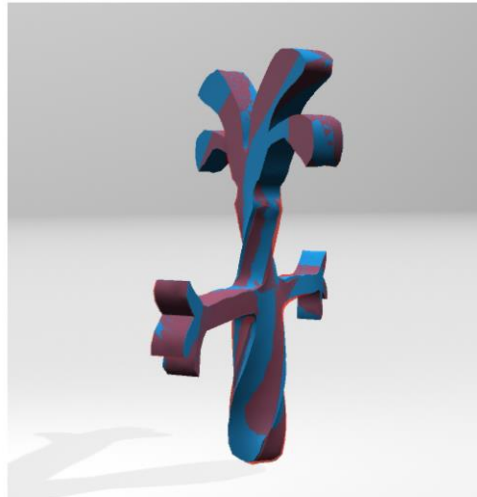
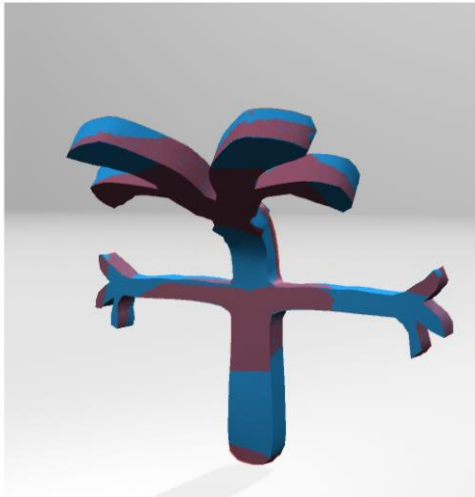
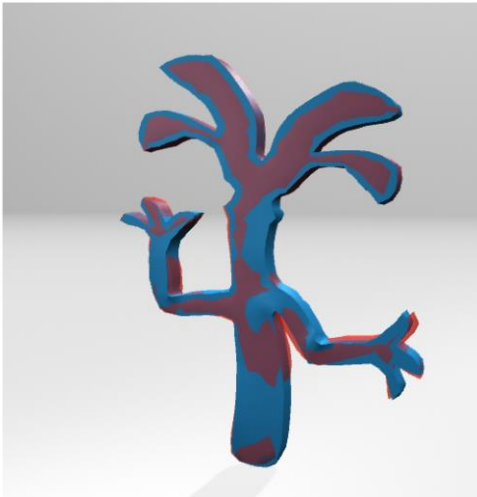
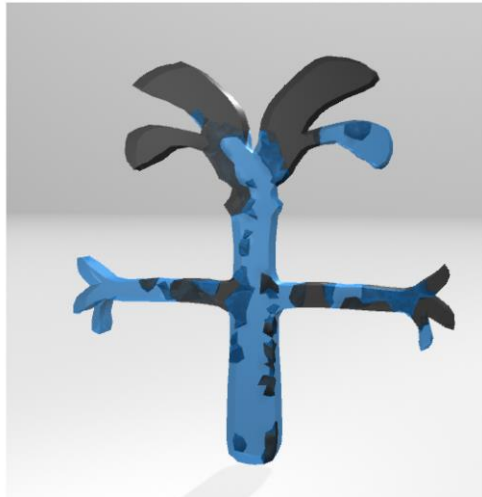
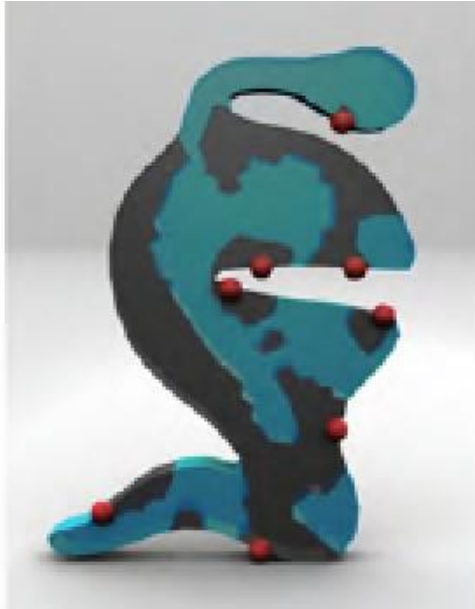
Geometric Design

Trend 1: Material design



Material distribution optimization

[Skouras et al. 2013]



Trend 2: Structure Design



← Design →



Small scale:

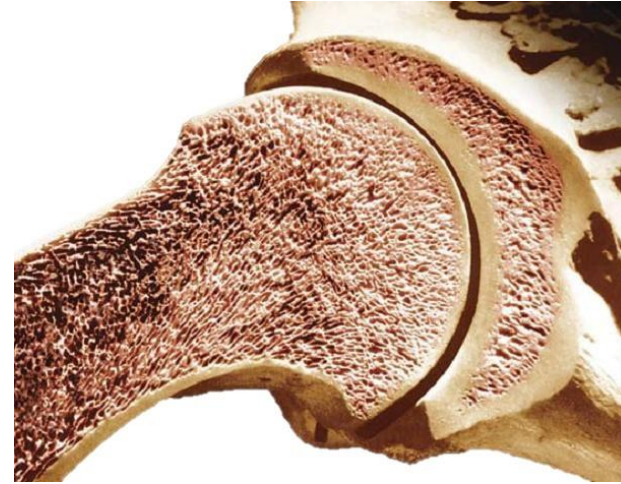
Micro-structure,
Frame, Foam, etc.

Large scale:

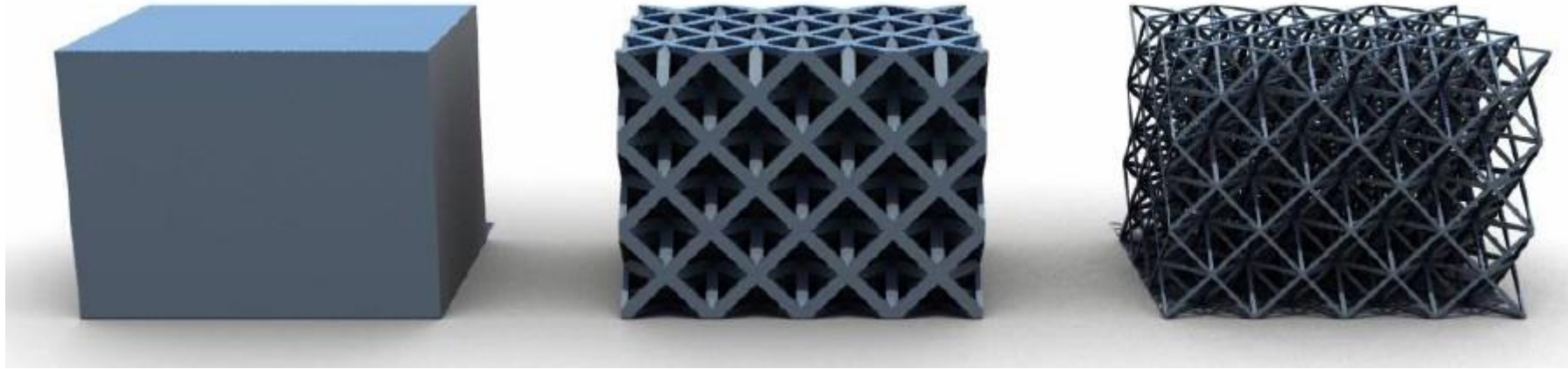
Assembly,
Mechanical, etc.

Microstructure Design

Microstructure in nature



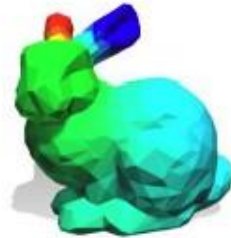
Material Structure



Problem

Input

- Shape with assigned Material Parameters

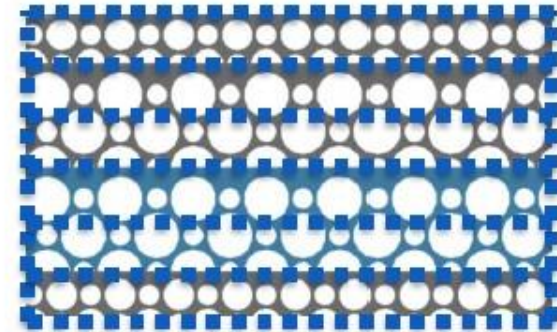


- Deformation Specification

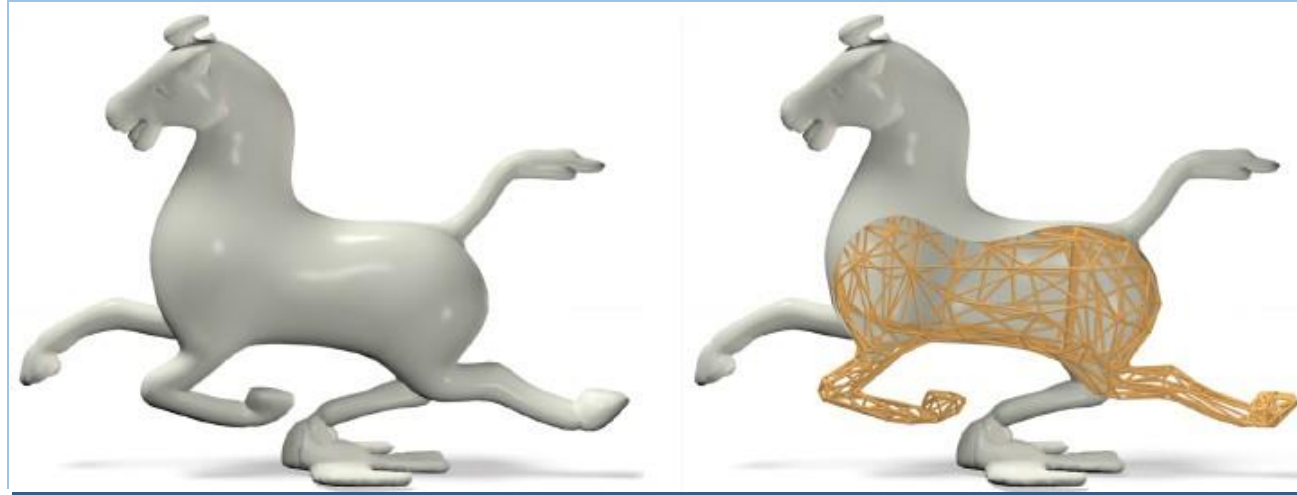


Output

- Spatially-varying material structure



Meso-structure



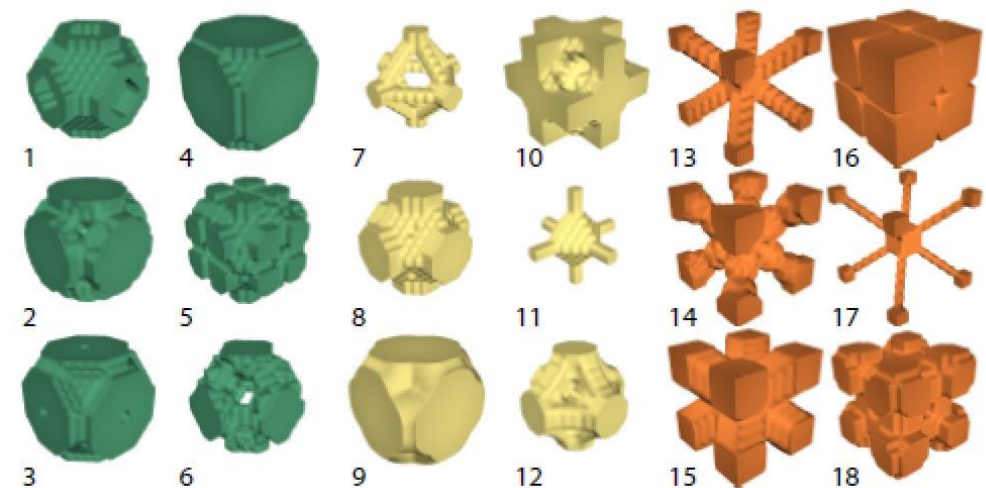
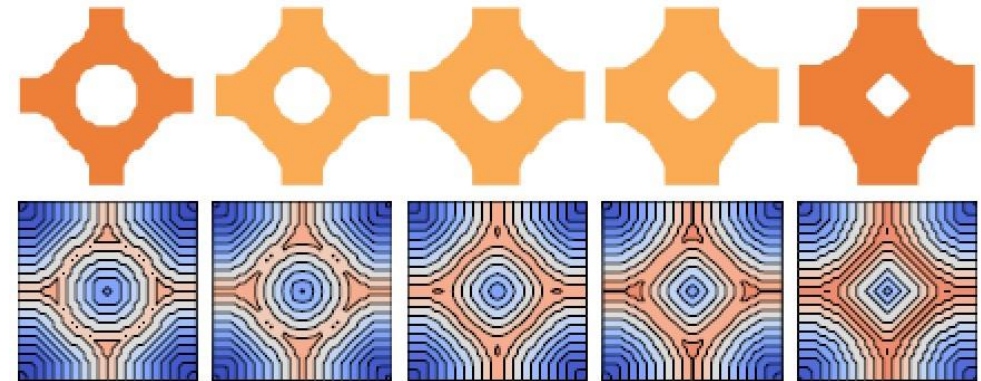
[Wang et al. 2013]



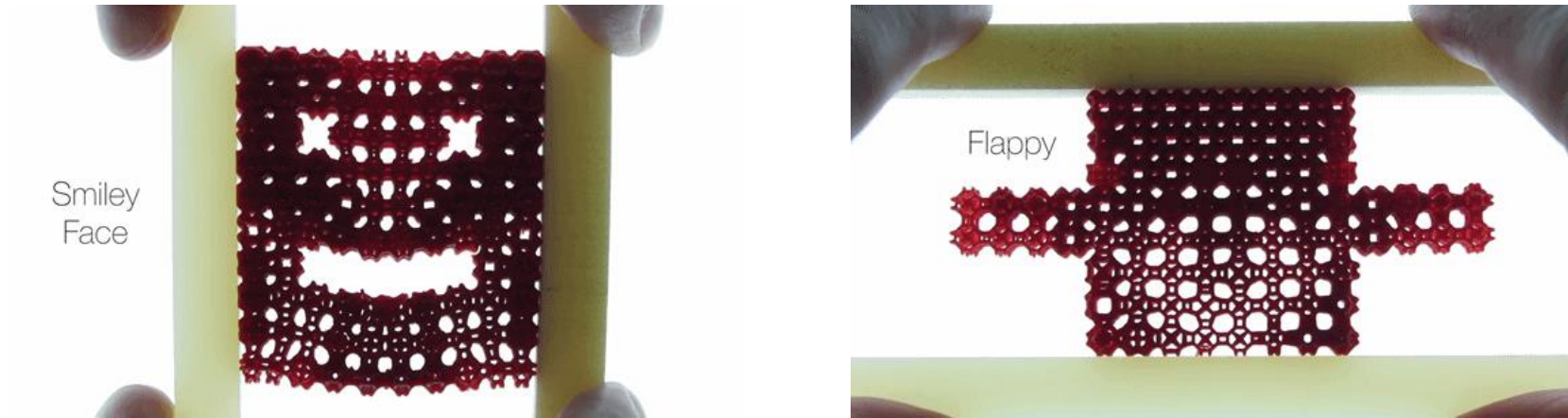
[Lv et al. 2014]

Elastic microstructure design

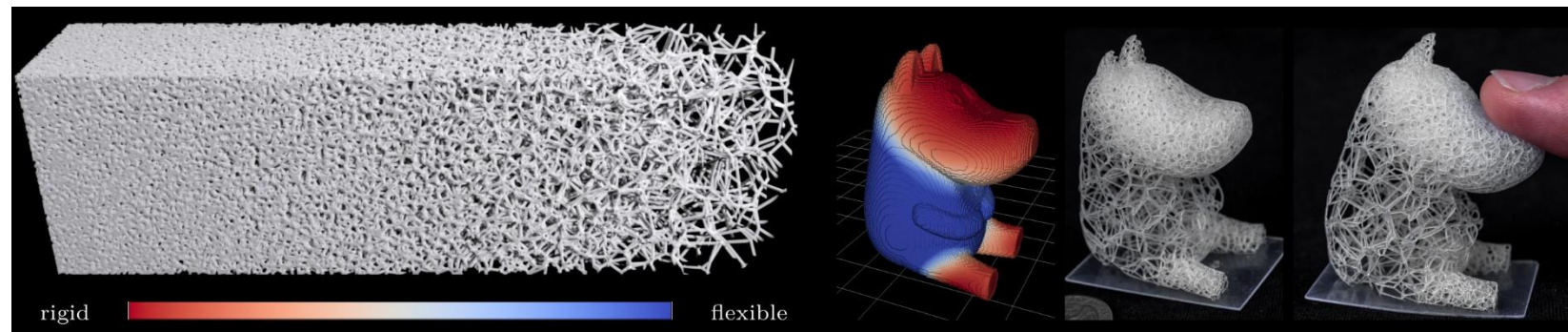
[Schumacher et al. 2015]



Elastic microstructure design



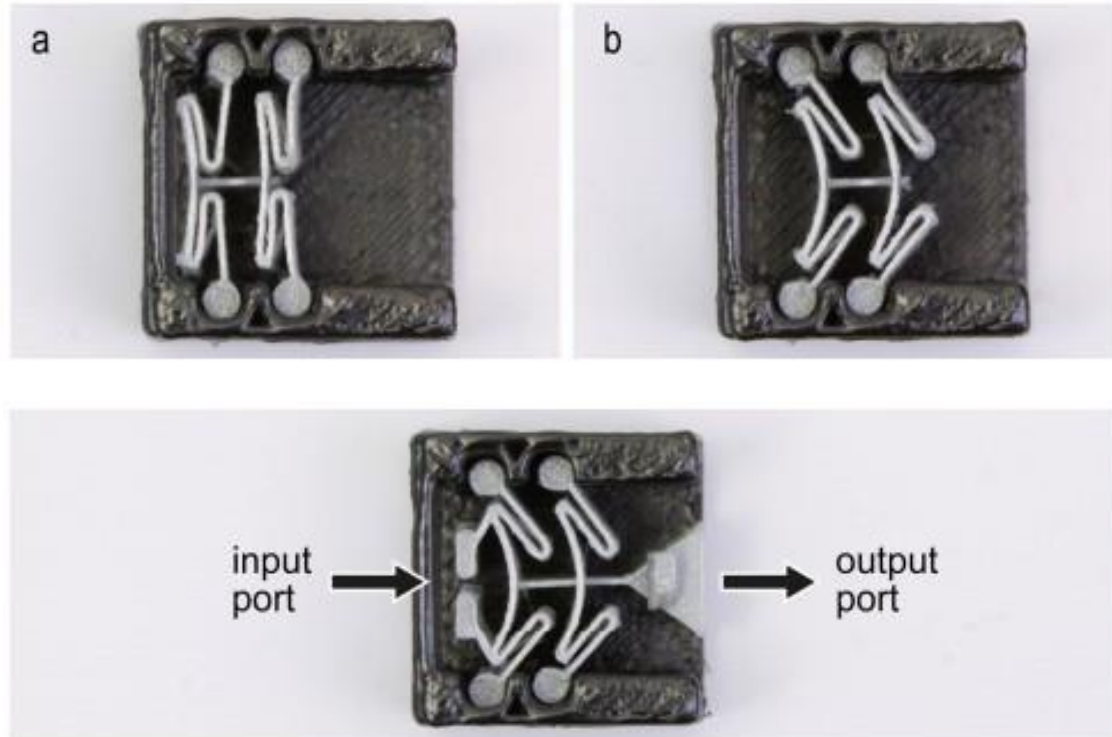
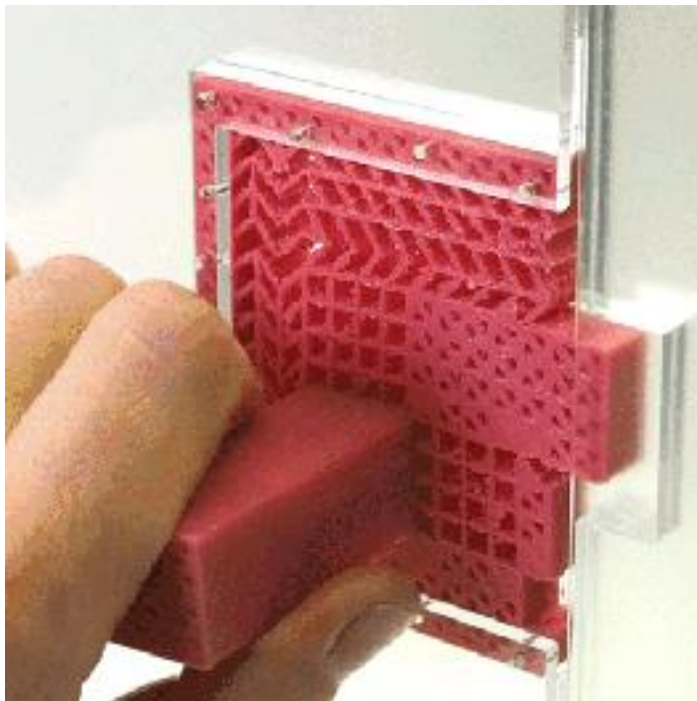
[Panetta et al. 2015]



[Martínez et al. 2016]

Functional microstructure design

[Ion et al. 2017]



Course Content

Homepage

http://staff.ustc.edu.cn/~renjie/CAGD_2024S1/default.htm



Renjie Chen (陈仁杰)

[Graphics & Geometric Computing Laboratory \(GCL\)](#)
[School of Mathematical Sciences](#)
[University of Science and Technology of China \(USTC\)](#)

Email: renjie at ustc.edu.cn

Teaching

[Computer Aided Geometric Design \(Autumn-Winter 2024-2025\)](#)

[Mathematical Modelling \(Spring-Summer 2023-2024\)](#)

[Computer Graphics \(Spring-Summer 2022-2023\)](#)

[GAMES 301: Surface Parameterization](#)

[Summer School for Advances in Computer Graphics 2022 \(计算机图形学前沿进展\)](#)

Course Objectives

- Basic methods for geometry representations
- Geometric modeling and processing
- New developments in computer graphics and geometric design
- It is better to teach people how to fish than to give them fish

Prerequisites: Mathematics

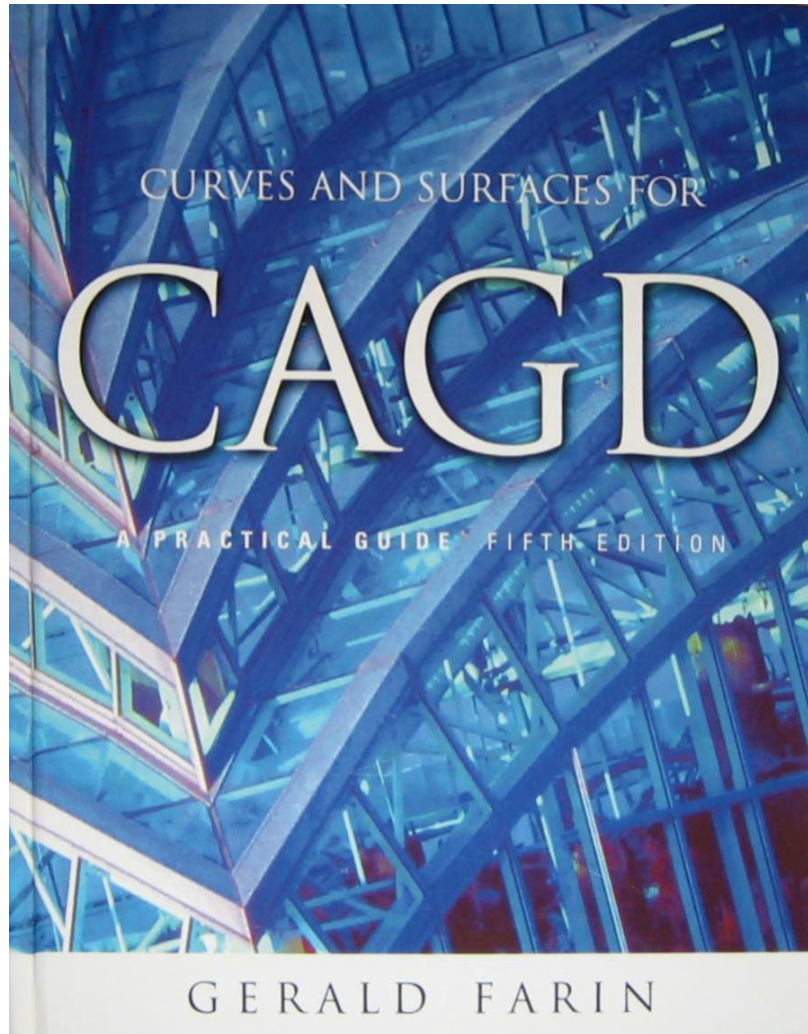
- Linear Algebra
- Calculus
- Geometry: space geometry、 differential geometry
- Differential equations
- Optimization
- Numerical methods and computations
- ...

Prerequisites: Programming

- Programming can realize and see the ideas in your mind
- Algorithm: rigorous logical thinking
- Matlab
- Python
- C++
- Various professional application software
 - Photoshop, 3D Max, Maya, AutoCAD, Adobe Products...

Literature

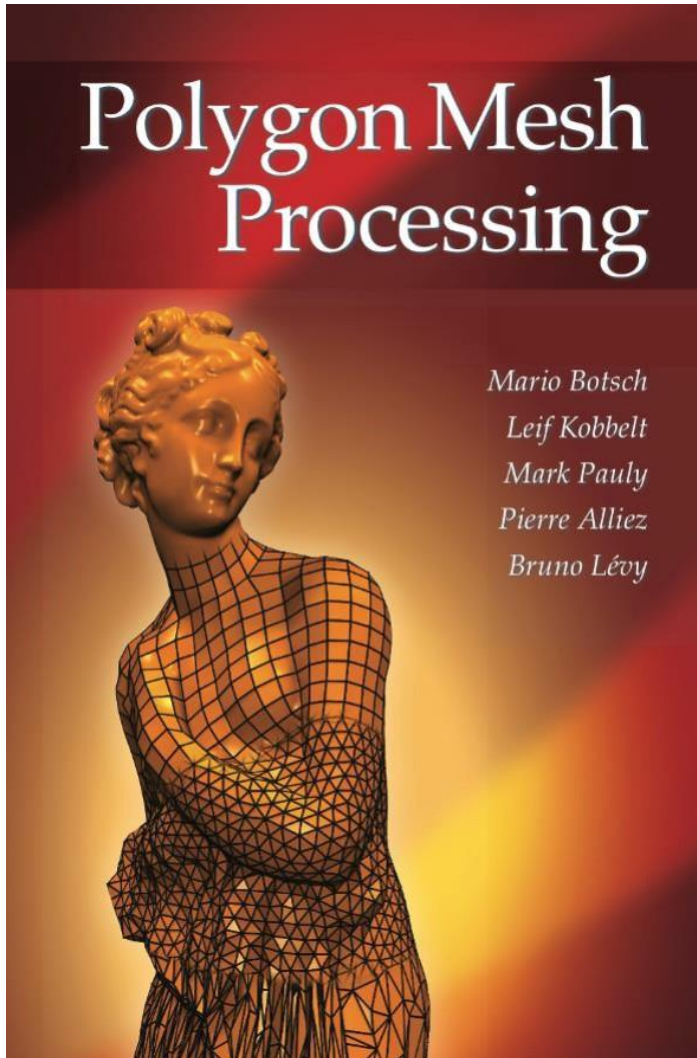
Textbook: Splines



Gerald Farin

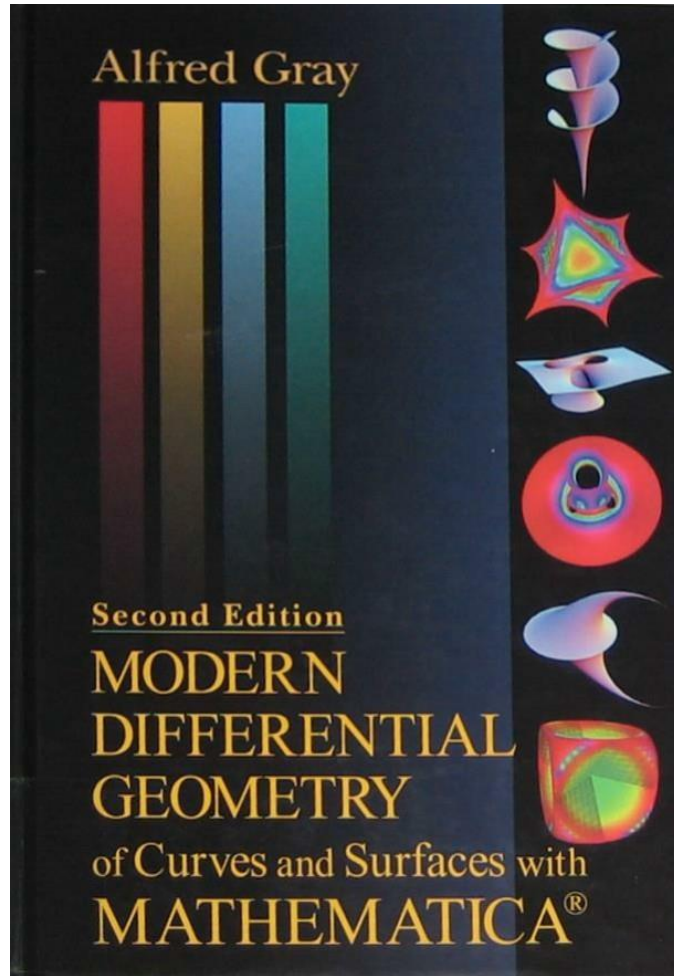
Curves and Surfaces for Computer
Aided Geometric Design
(Fifth Edition)

Textbook: Mesh Processing



**Mario Botsch, Leif Kobbelt, Mark Pauly,
Pierre Alliez, Bruno Levy**
Polygon Mesh Processing

Differential Geometry



Alfred Gray

Modern Differential Geometry of Curves and
Surfaces with Mathematica®

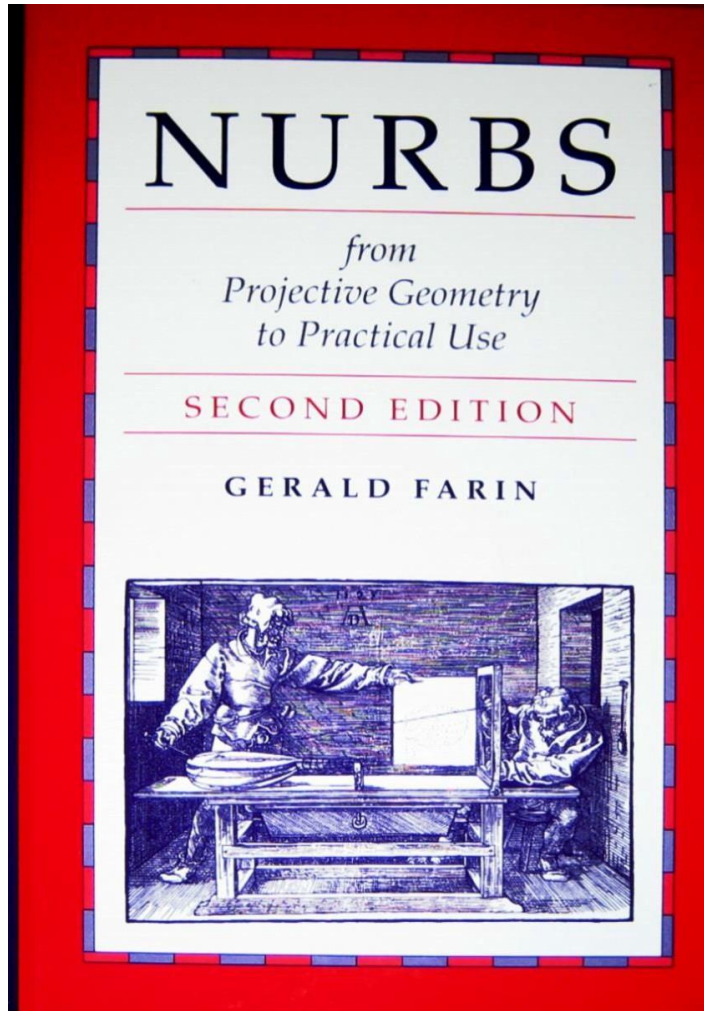
(Second Edition)

More on Rational Splines

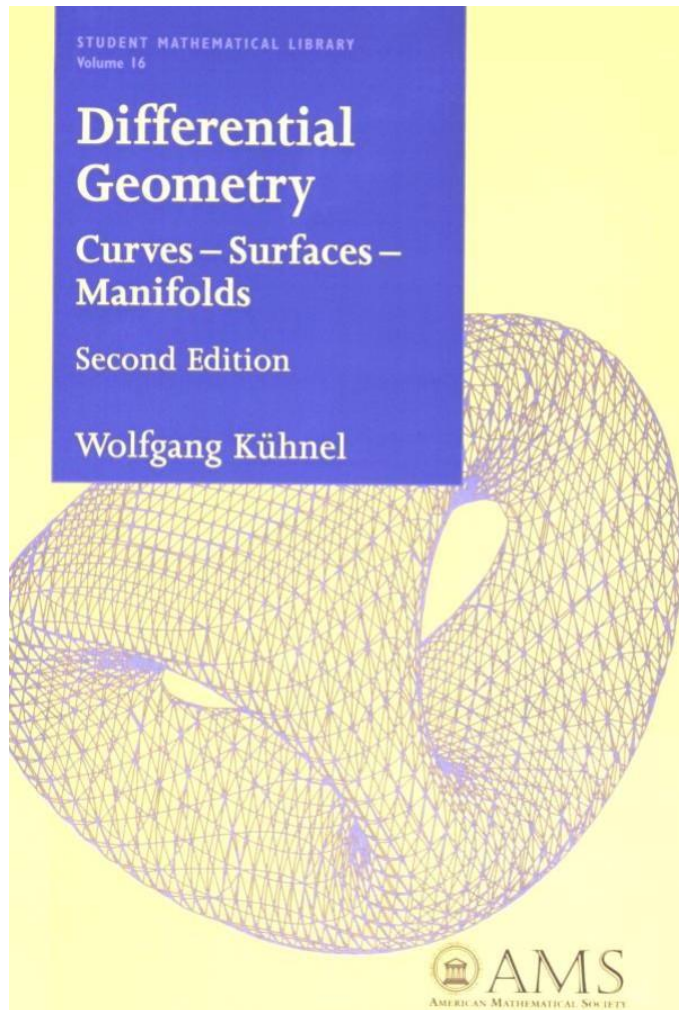
Gerald Farin

NURBS – from Projective Geometry to Practical Use
(Second Edition)

- More details on rational curves & surfaces and projective geometry



More on Differential Geometry



Wolfgang Kühnel

Differential Geometry:

Curves – Surfaces – Manifolds

More on CAGD

- Josef Hoschek and Dieter Lasser. Fundamentals of Computer Aided Geometric Design. A K Peters/CRC Press. 1996.
- Thomas W. Sederberg. Computer Aided Geometric Design. Lecture notes. 2012.
- 朱心雄. 自由曲线曲面造型技术. 科学出版社. 2000.
- 王国瑾, 汪国昭, 郑建民. 计算机辅助几何设计. 高等教育出版社. 2001.
- 施法中. 计算机辅助几何设计与非均匀有理B样条. 高等教育出版社. 2001.

Some materials are available for download on the course homepage

Homework

- Assigned on Monday evening
- Due on Sunday
- Explained on the following Monday

Course Message

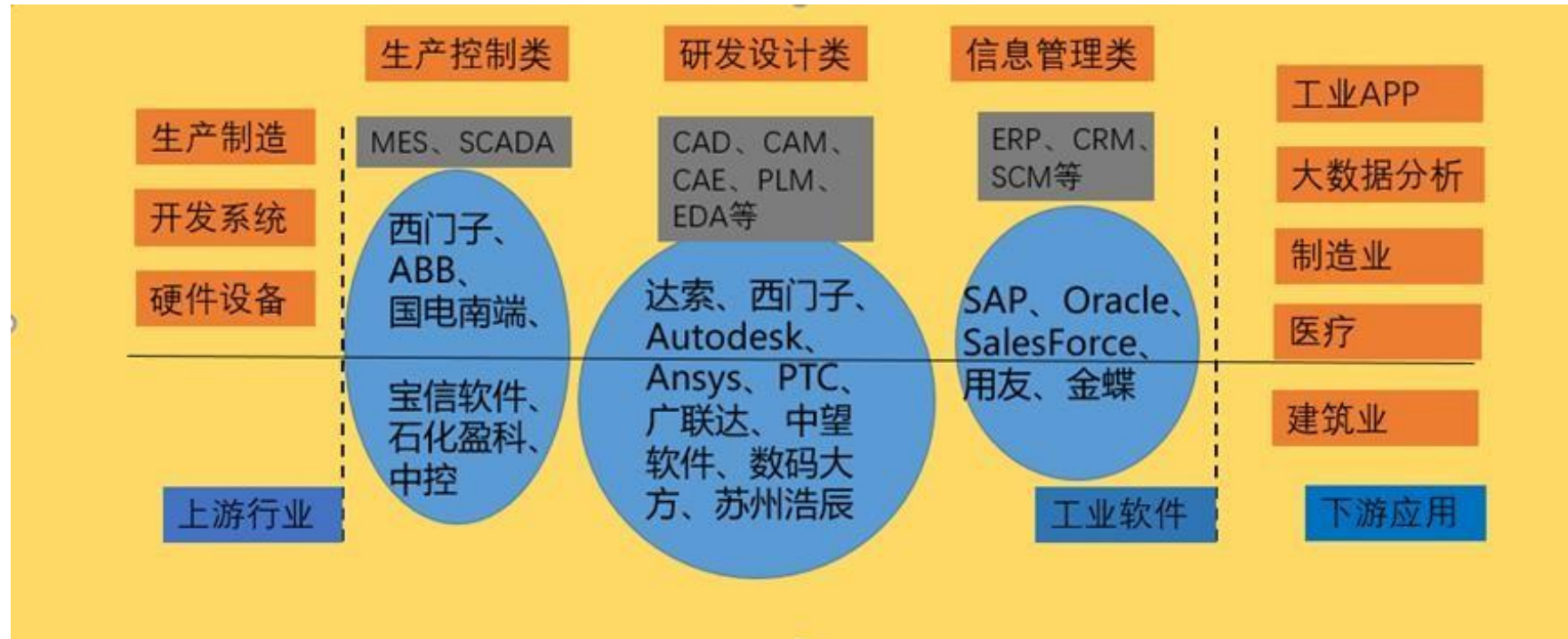
截止2020年

序号	实体
1	华为云计算技术
2	华为云北京
3	华为云大连
4	华为云广州
5	华为云贵阳
6	华为云香港
7	华为云上海
8	华为云深圳
9	华为OpenLab苏州
10	乌兰察布华为云计算技术

三

序号	实体
1	昌吉溢达纺织有限公司
2	合肥宝龙达信息技术有限公司
3	合肥美菱有限公司
4	新疆和田浩林发饰品有限公司
5	和田泰达服饰有限公司
6	今创集团
7	南京新一棉纺织印染有限公司
8	南昌欧菲光科技有限公司
9	碳元科技
10	新疆丝路华大基因科技有限公司
11	北京六合华大基因科技有限公司

工业软件的核心地位



According to statistics, the global market size of industrial software products has been growing at a rate of about 5% per year, exceeding US\$400 billion in 2019, and will reach US\$433.2 billion by 2020. At present, the domestic industrial software market is relatively small, accounting for only about 6% of the world's total, far lower than China's 16% share of GDP. Therefore, in the context of future intelligent manufacturing, domestic substitution, and software cloud computing, there is huge potential for development.

Development of CAx software in China

- Mid-1960s, the application of CAD/CAM technology in aviation and shipbuilding engineering began to be studied.
- After mid-1970s, CAx technology developed rapidly.
- 1975, Xi'an Jiaotong University, 751 light pen graphic display.
- 1984, Professor Tang Rongxi of Beihang University developed China's first polyhedron solid modeling prototype system PANDA.
- Mid-1980s, statistics showed that various colleges and research institutions had developed more than 2000 CAD systems.
 - Our country's first shipbuilding integrated production system;
 - Nanhang B-SURF (3D-CAD) system can build full-machine digital models of two types of drones, and present various perspective views and cross-section views of the whole machine and its components on the IBM4341 graphic terminal.
- 1992, the Panda system of very large-scale integrated circuit computer-aided design (IC-CAD) passed the national appraisal.
- At present, this type of EDA software is the weak point in the chip field.
 - 2022, Cadence stopped supplying EDA to ZTE.

Development of CAx software in China

- Foreign industrial software giants have technological and market advantages
 - Foreign industrial software giants have almost all the core technologies and standards in the industry, including Dassault, Siemens, Autodesk, etc. in the field of R&D design software, and Siemens, etc. in the field of production control software
- The catch-up road for local R&D design software companies is still quite long
 - The combined market share of the top three domestic companies in the R&D design software market is only 9%. Zhongwang, as the domestic company with the highest market share in China's CAD market, has an operating income of only 360 million and a profit scale of less than 100 million, which is 60-100 times lower than Dassault and Autodesk

ZWSoft opened at 420 yuan on March 12, 2021, with a market value of **25.335 billion yuan!**

Challenges of CAGD&CG

- CAGD&CG has formed a complete industry chain in the United States: scientific research, games, movies, entertainment, education, art, industry...
 - In China, it is gradually forming
- China is in urgent need of 3D talents!!!

A vast world with a lot to offer!

Interesting, fun and promising!!

Thank you!

Questions?

Computer Aided Geometric Design

Fall Semester 2024

Mathematical background: Linear algebra

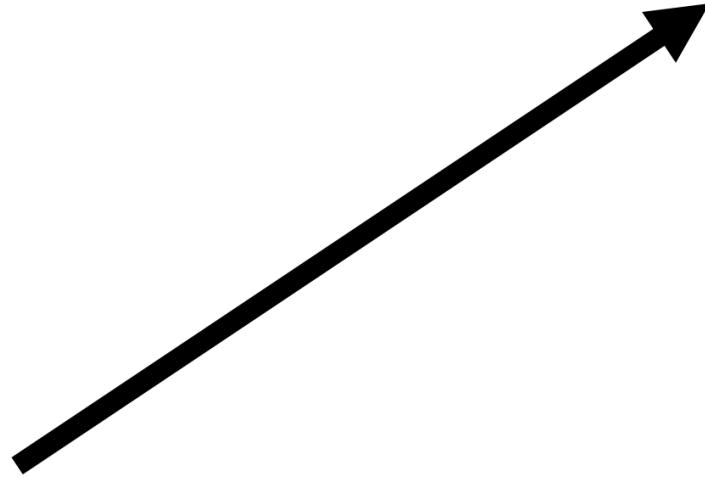
陈仁杰

renjiec@ustc.edu.cn

<http://staff.ustc.edu.cn/~renjiec>

Vector Spaces

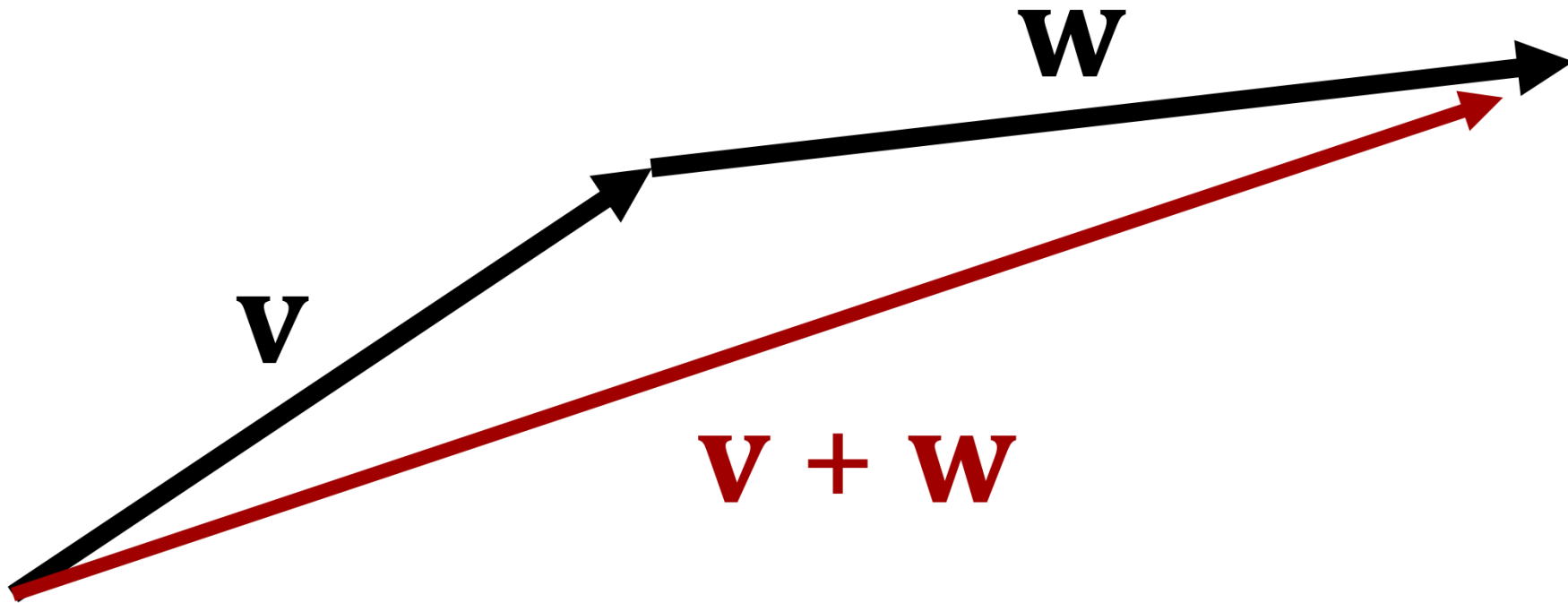
Vectors



Vectors are arrows in space

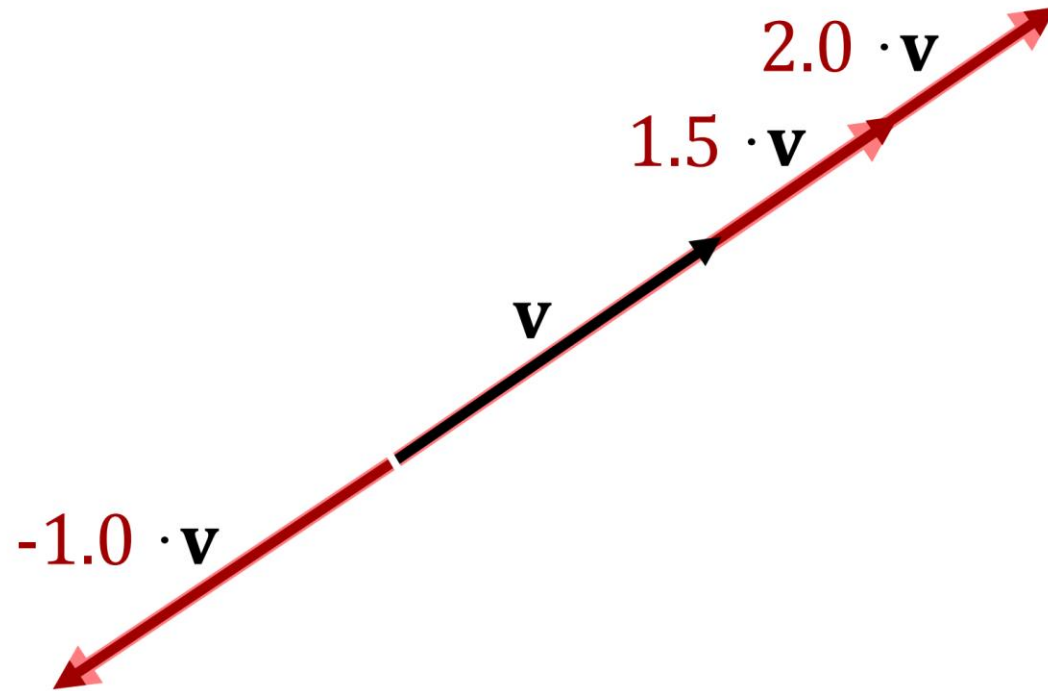
Classically: 2 or 3 dim. Euclidean space

Vector Operations



“Adding” Vectors:
concatenation

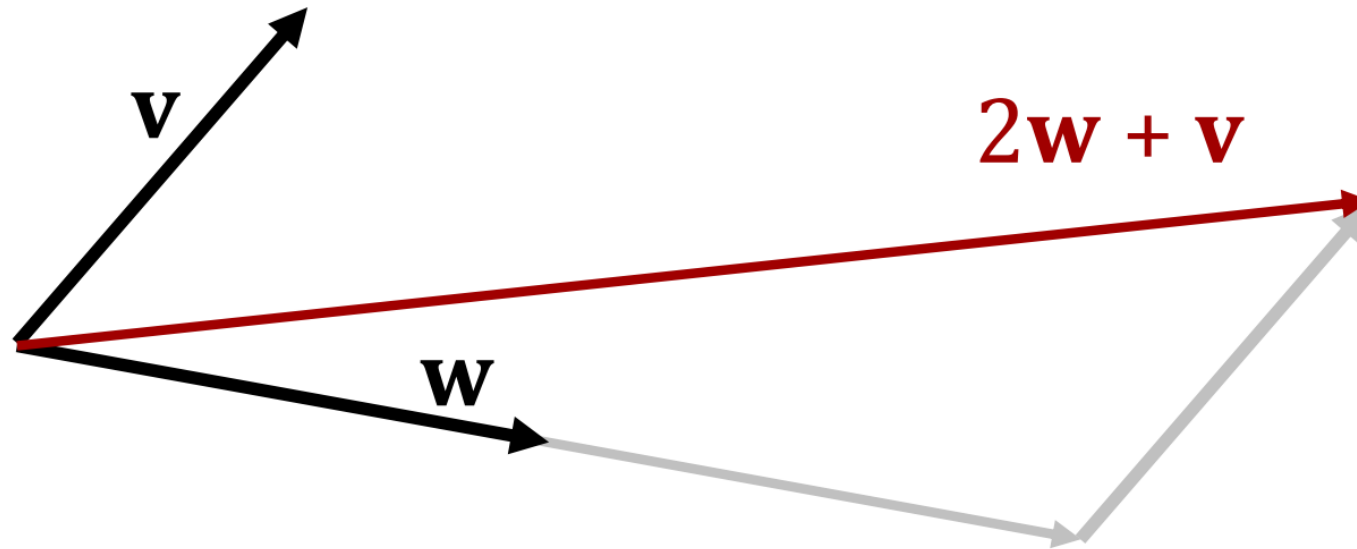
Vector Operations



Scalar Multiplication:

Scaling vectors (incl. mirroring)

You can combine it...



Linear Combinations:

This is basically all you can do.

$$\mathbf{r} = \sum_{i=1}^n \lambda_i \mathbf{v}_i$$

Vector Spaces

- Definition: A *vector space* over a field F (e.g. \mathbb{R}) is a set V together with two operations
 - Addition of vectors $u = v + w$
 - Multiplication with scalars $w = \lambda v$such that

1. $\forall u, v, w \in V: (u + v) + w = u + (v + w)$

2. $\forall u, v \in V: u + v = v + u$

3. $\exists 0_V \in V: \forall v \in V: v + 0_V = v$

4. $\forall v \in V: \exists w \in V: v + w = 0_V$

$(V, +)$ is an Abelian group

5. $\forall v \in V, \lambda, \mu \in F: \lambda(\mu v) = (\lambda\mu)v$

6. for $1_F \in F: \forall v \in V: 1_F v = v$

7. $\forall \lambda \in F: \forall v, w \in V: \lambda(v + w) = \lambda v + \lambda w$

8. $\forall \lambda, \mu \in F, v \in V: (\lambda + \mu)v = \lambda v + \mu v$

The multiplication is compatible with the addition

Vector spaces

- **Subspaces**

- A non-empty subset $W \subset V$ is a *subspace* if W is a vector space (w.r.t the induced addition and scalar multiplication).
- Only need to check if the addition and scalar multiplication are closed.

$$\boldsymbol{v}, \boldsymbol{w} \in W \quad \Rightarrow \boldsymbol{v} + \boldsymbol{w} \in W$$

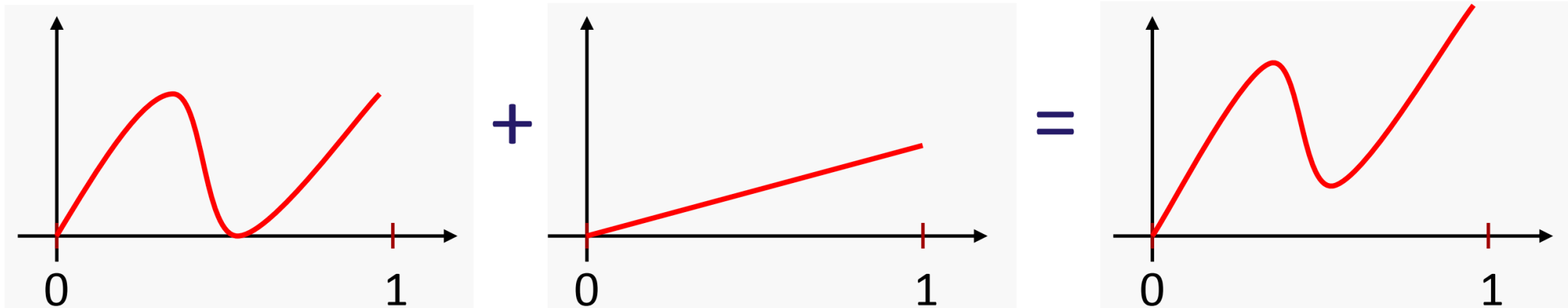
$$\boldsymbol{v} \in W, \lambda \in F \quad \Rightarrow \lambda \boldsymbol{v} \in W$$

- What are the subspaces of \mathbb{R}^3 ?

Examples Spaces

- **Function spaces:**

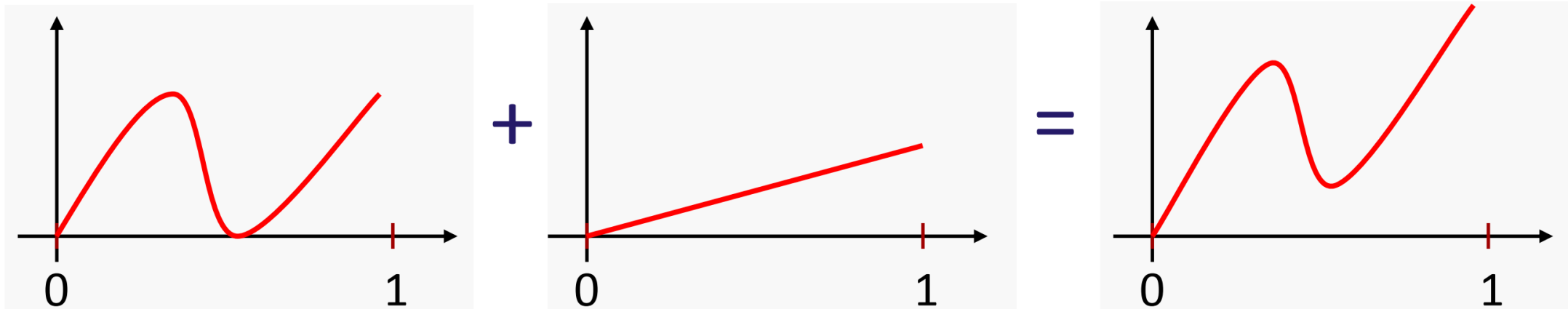
- Space of all functions $f: \mathbb{R} \rightarrow \mathbb{R}$
- Addition: $(f + g)(x) = f(x) + g(x)$
- Scalar multiplication: $(\lambda f)(x) = \lambda f(x)$
- Check the definition



Examples Spaces

- **Function spaces:**

- Domains and codomain need to be \mathbb{R}
- For example: space of all functions $f: [0,1]^5 \rightarrow \mathbb{R}^8$
- Codomain must be a vector space (Why?)



Examples of Subspaces

- **Continuous / differentiable functions**

- The continuous / differentiable functions form a subspace of the space of all functions $f: D \subset \mathbb{R}^m \rightarrow \mathbb{R}^n$
- Why?

- **Polynomials**

- The polynomials form a subspace of the space of functions $f: \mathbb{R} \rightarrow \mathbb{R}$
- The polynomials of degree $\leq n$ again form a subspace
- Adding polynomials

$$\sum_{i=1}^n a_i x^i + \sum_{i=1}^n b_i x^i = \sum_{i=1}^n (a_i + b_i) x^i$$

Constructing Spaces

Linear Span

- The *linear span* of a subset $S \subset V$ is the “smallest subspace” of V that contains S
- What does that mean?
 - For any subspace W such that $S \subset W \subset V$, we have $\text{span}(S) \subset W$
- Construction: Any $v \in \text{span}(S)$ is a finite linear combination of elements of S

$$v = \sum_{i=1}^n \lambda_i s^i$$

Spanning set

- A subset $S \subset V$ is a *spanning set* of V if $\text{span}(S) = V$

Vector spaces

- **Linear independence**

- A subset $S \subset V$ is *linearly independent* if no vector of S is a finite linear combination of the other vectors of S

- **Basis**

- A *basis* of a vector space is a linearly independent spanning set.

Dimension

- **Lemma**

- If V has a finite basis of n elements, then all bases of V have n elements

- **Dimension**

- If V has a finite basis, then the dimension of V is the number of elements of the basis
 - If V has no finite basis, then the dimension of V is infinite

Examples

- **Polynomials of degree $\leq n$**

- A basis? What is the dimension?

Solution:

- An example of a basis is $\{1, x, x^2, \dots, x^n\}$
- Dimension is $n + 1$

- **Space of all polynomials**

- A basis? What is the dimension?

Solution:

- An example of a basis is $\{1, x, x^2, \dots\}$
- Dimension is infinite

Finite dimensional vector spaces

- **Vector spaces**

- Any finite-dim., real vector space is isomorphic to \mathbb{R}^n
 - Array of numbers
 - Behave like arrows in a flat (Euclidean) geometry
- Proof:
 - Construct basis
 - Represent as span of basis vectors

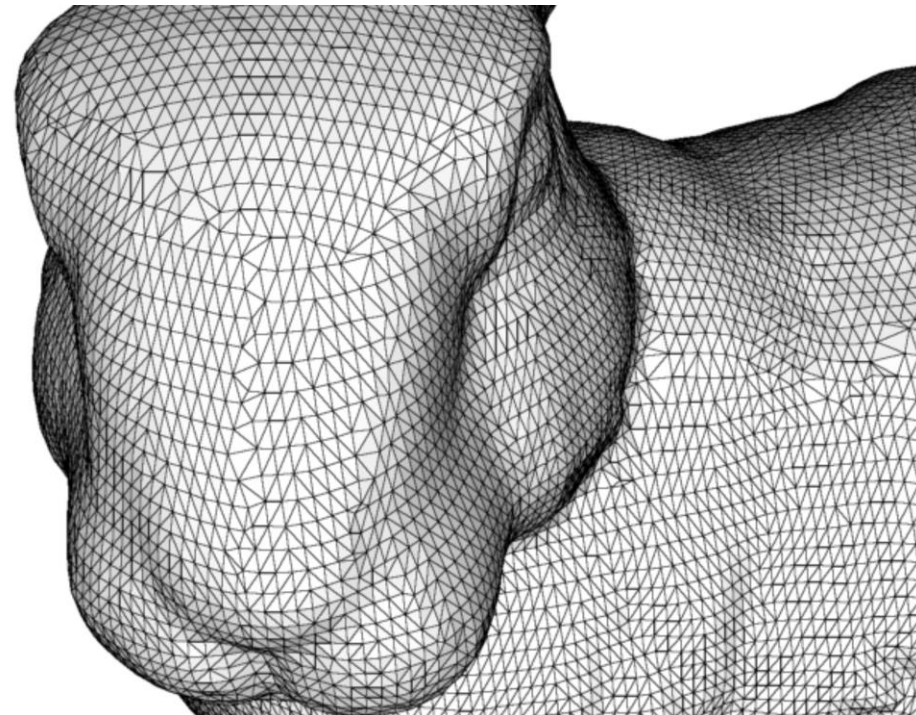
Isomorphism is not unique, since we can choose different bases

Another Example of a Vector Space

Representation of a triangle mesh in \mathbb{R}^3

- Vertices : a finite set $\{v_1, \dots, v_n\}$ of points in \mathbb{R}^3
- Faces: a list of triplets, e.g. $\{\{2, 34, 7\}, \dots, \{14, 7, 5\}\}$

Number of Vertices		34835		
Index	X	Y	Z	
<input type="checkbox"/> 0	-0.0378297	0.12794	0.00447467	↗
<input type="checkbox"/> 1	-0.0447794	0.128887	0.00190497	↗
<input type="checkbox"/> 2	-0.0680095	0.151244	0.0371953	↗
<input type="checkbox"/> 3	-0.00228741	0.13015	0.0232201	↗
<input type="checkbox"/> 4	-0.0226054	0.126675	0.00715587	↗
Center		0.0	0.0	0.0
Number of Elements		69473		
Vertices per Element		3		
Index	0	1	2	
<input type="checkbox"/> 1640	10645	10769	10768	↗
<input type="checkbox"/> 1640	10644	10645	10768	↗
<input type="checkbox"/> 1640	780	10996	10992	↗
<input type="checkbox"/> 1640	9978	9765	8572	↗
<input type="checkbox"/> 1640	7146	10960	10616	↗



Another Example of a Vector Space

- **Shape space**

- Vary the vertices, but keep the face list fixed
- Is isomorphic to \mathbb{R}^{3n}

Linear Maps

Linear Maps

Definition

- A map $L: V \rightarrow W$ between vector spaces V, W is linear if
 - $\forall v_1, v_2 \in V: \quad L(v_1 + v_2) = L(v_1) + L(v_2)$
 - $\forall v \in V, \lambda \in F: \quad L(\lambda v) = \lambda L(v)$

This means that L is compatible with the linear structure of V and W

Linear Maps

Definition

- A map $L: V \rightarrow W$ between vector spaces V, W is linear if
 - $\forall v_1, v_2 \in V: \quad L(v_1 + v_2) = L(v_1) + L(v_2)$
 - $\forall v \in V, \lambda \in F: \quad L(\lambda v) = \lambda L(v)$

Some properties

- $L(0_V) = 0_W$
- Proof: $L(0_V) = L(0 \cdot 0_V) = 0L(0_V) = 0_W$

Linear Maps

Definition

- A map $L: V \rightarrow W$ between vector spaces V, W is linear if
 - $\forall v_1, v_2 \in V: \quad L(v_1 + v_2) = L(v_1) + L(v_2)$
 - $\forall v \in V, \lambda \in F: \quad L(\lambda v) = \lambda L(v)$

Some properties

- The image $L(V)$ is a subspace of W
- Proof: Show addition and scalar multiplication is closed

$$L(v_1) + L(v_2) = L(v_1 + v_2) \in W$$

$$\lambda L(v) = L(\lambda v) \in W$$

Linear Maps

Definition

- A map $L: V \rightarrow W$ between vector spaces V, W is linear if
 - $\forall v_1, v_2 \in V: \quad L(v_1 + v_2) = L(v_1) + L(v_2)$
 - $\forall v \in V, \lambda \in F: \quad L(\lambda v) = \lambda L(v)$

Some properties

- The set of linear maps from V to W forms a **subspace** of the space of all functions
- Proof: If L, \tilde{L} are linear, then $L + \tilde{L}$ is linear
If L is linear, then λL is linear

Linear Map Representation

Construction

- A linear map $L: V \rightarrow W$ is uniquely determined if we specify the image of each basis vector of a basis of V
- Proof: We have $v = \sum_j \alpha_j v_j$, hence

$$L(v) = L\left(\sum_j \alpha_j v_j\right) = \sum_j \alpha_j L(v_j)$$

Matrix Representation

- Let V and W be vector spaces with respective bases $v = (v_1, v_2, \dots, v_n)$ and $w = (w_1, w_2, \dots, w_m)$
- Suppose $L: V \rightarrow W$ is a linear mapping, such that

$$L(v_1) = a_{11}w_1 + a_{21}w_2 + \cdots + a_{m1}w_m$$

.....

$$L(v_n) = a_{1n}w_1 + a_{2n}w_2 + \cdots + a_{mn}w_m$$

- The matrix representation of L w.r.t. the basis v and w is

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

The j^{th} -column of A is formed by the coefficients of $L(v_j)$

Example

- $L: \mathbb{R}^2 \rightarrow \mathbb{R}^3$, s. t. $(x, y) \rightarrow (x + 3y, 2x + 5y, 7x + 9y)$
- Find the matrix representation of L w.r.t the standard bases of \mathbb{R}^2 and \mathbb{R}^3
- Answer: $L(1,0) = (1,2,7)$, $L(0,1) = (3,5,9)$, hence the matrix of L , w.r.t the standard bases is the 3×2 matrix

$$\begin{pmatrix} 1 & 3 \\ 2 & 5 \\ 7 & 9 \end{pmatrix}$$

Matrix Representation

Explicitly

- The coefficients α_j and β_i are related by $\beta_i = \sum_j a_{ij} \alpha_j$

$$\begin{aligned} L(v) &= L\left(\sum_j \alpha_j v_j\right) = \sum_j \alpha_j L(v_j) = \sum_j \alpha_j \sum_i a_{ij} w_i \\ &= \sum_i \left(\sum_j a_{ij} \alpha_j\right) w_i = \sum_i \beta_i w_i = w \end{aligned}$$

This can be written as a matrix-vector product

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_m \end{pmatrix}$$

Example Matrices

Shearing

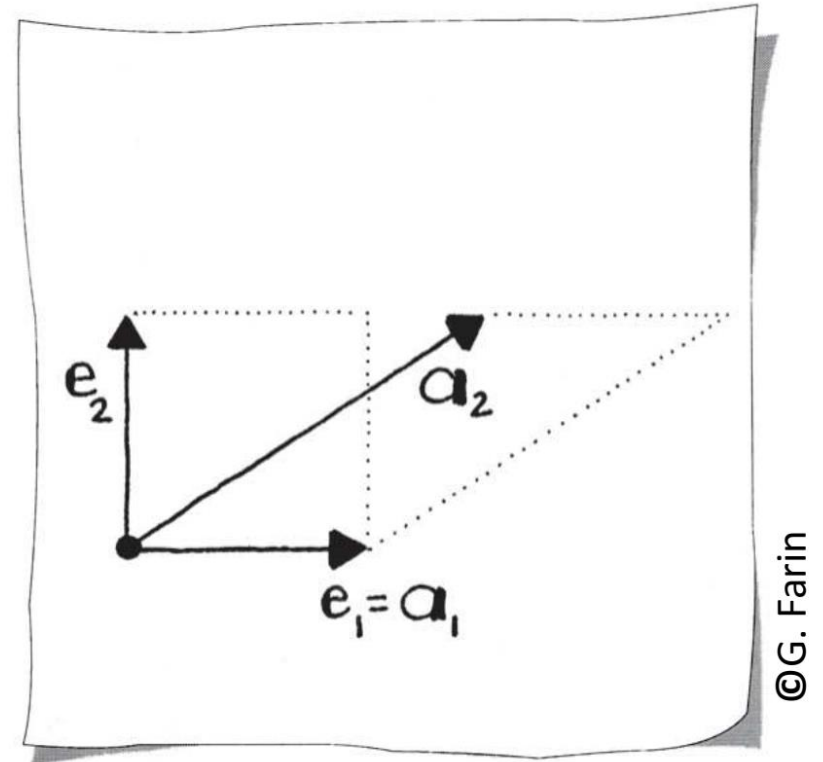
- Consider the standard basis of \mathbb{R}^2
 - Matrix?
 - First row

$$A \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

- Second row

$$A \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.3 \\ 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 0 \\ 1.3 & 1 \end{pmatrix}$$



Example Matrices

Shearing

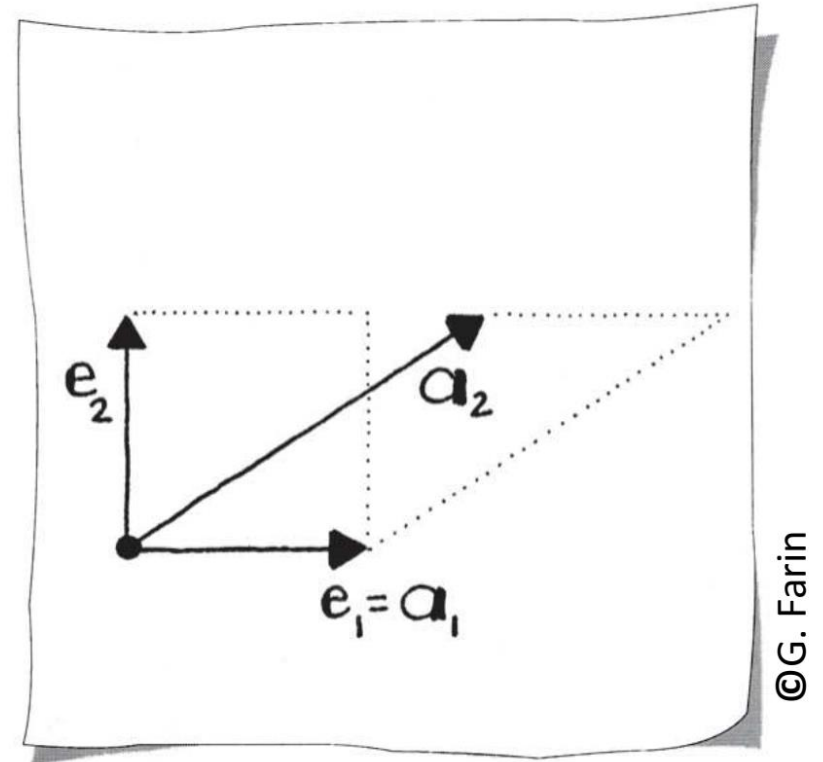
- Consider the standard basis of \mathbb{R}^2
 - Matrix?
 - First row

$$A \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

- Second row

$$A \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.3 \\ 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 1.3 \\ 0 & 1 \end{pmatrix}$$



Reminder: Properties of Matrices

Symmetric

- $A^T = A$

Orthogonal

$$A^T = A^{-1}$$

Product is not commutative!

- Find an example with $AB \neq BA$

Product of symmetric matrices may not be symmetric

- Find an example

Product of orthogonal matrices *is* orthogonal

$$(AB)^T = B^T A^T = B^{-1} A^{-1} = (AB)^{-1}$$

Example of Matrices

Rotation of the plane

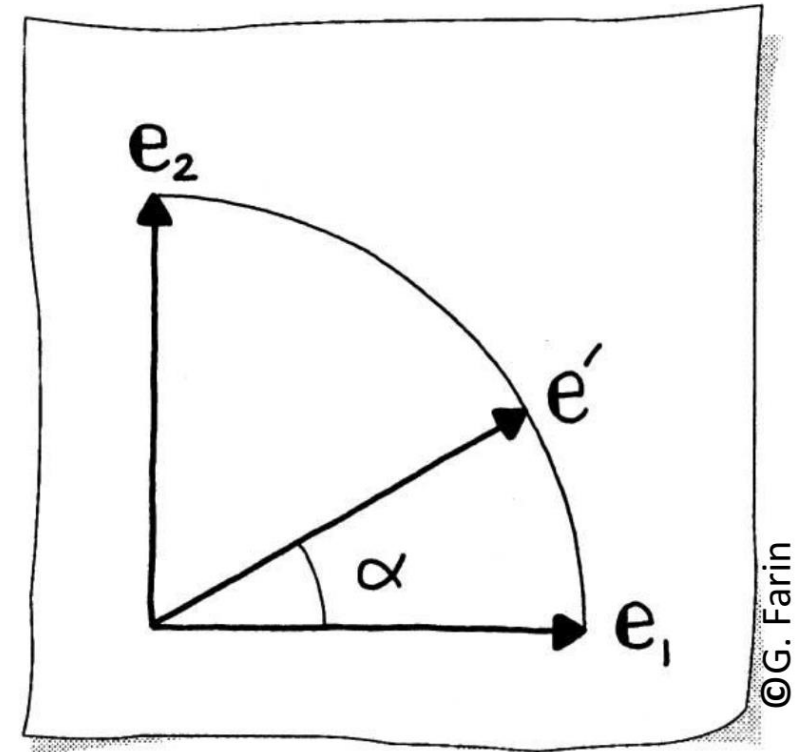
- Linear?
- Consider standard basis of \mathbb{R}^2
Matrix?

$$\begin{pmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{pmatrix}$$

- Transposition reverse orientation of the rotation

$$\begin{pmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \end{pmatrix}$$

Hence matrix is orthogonal $A^T = A^{-1}$



Examples of Linear Maps

Linear operators on a function space

Derivatives

- Differentiation maps functions to functions

$$\frac{\partial}{\partial x} : C^i(\mathbb{R}) \mapsto C^{i-1}(\mathbb{R})$$

$$f \mapsto \frac{\partial}{\partial x} f$$

Why is it linear?

- Basic rules of differentiation

$$\frac{\partial}{\partial x} (f + g) = \frac{\partial}{\partial x} f + \frac{\partial}{\partial x} g \quad \text{and} \quad \frac{\partial}{\partial x} (\lambda f) = \lambda \frac{\partial}{\partial x} f$$

Matrix Representation

Derivative on a space of polynomials

- Consider polynomials of degree ≤ 3 and the monomial basis
- What is the matrix representation of the derivative?
- Solution: Evaluate $\frac{\partial}{\partial x}$ on the basis
- $\frac{\partial}{\partial x} 1 = 0, \frac{\partial}{\partial x} x = 1, \frac{\partial}{\partial x} x^2 = 2x, \frac{\partial}{\partial x} x^3 = 3x^2$

Results are the columns of the matrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Examples of Linear Maps

Integrals on $C^0([a, b])$

- Integration maps a continuous function to a number

$$I: C^0([a, b]) \mapsto \mathbb{R}$$

$$I(f) = \int_a^b f dx$$

- The map is linear:

$$\int_a^b (f + g) dx = \int_a^b f dx + \int_a^b g dx$$

$$\int_a^b \lambda f dx = \lambda \int_a^b f dx$$

Matrix Representation

Integrals on a space of polynomials

- Consider polynomials of degree ≤ 3 over the interval $[0,1]$ and the monomial basis.
- What is the matrix representation of the integral?
- Solution: Evaluate $\int_0^1 dx$ on the basis

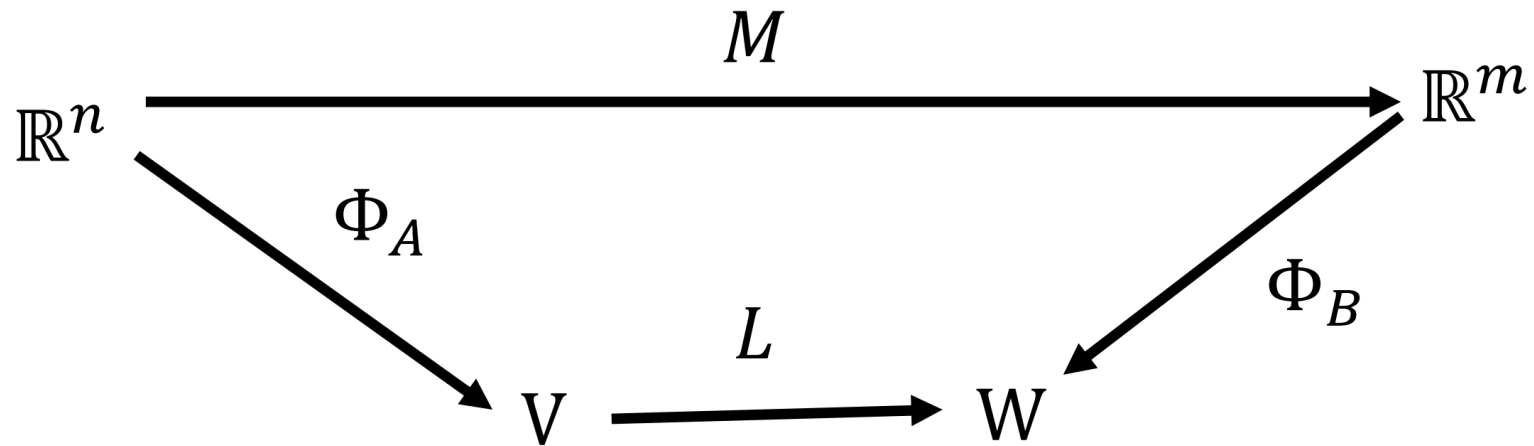
$$\int_0^1 1dx = 1, \quad \int_0^1 xdx = \frac{1}{2}, \quad \int_0^1 x^2dx = \frac{1}{3}, \quad \int_0^1 x^3dx = \frac{1}{4}$$

Results are the columns of the matrix

$$\begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \end{pmatrix}$$

Basis Transformations

Matrix representation of L



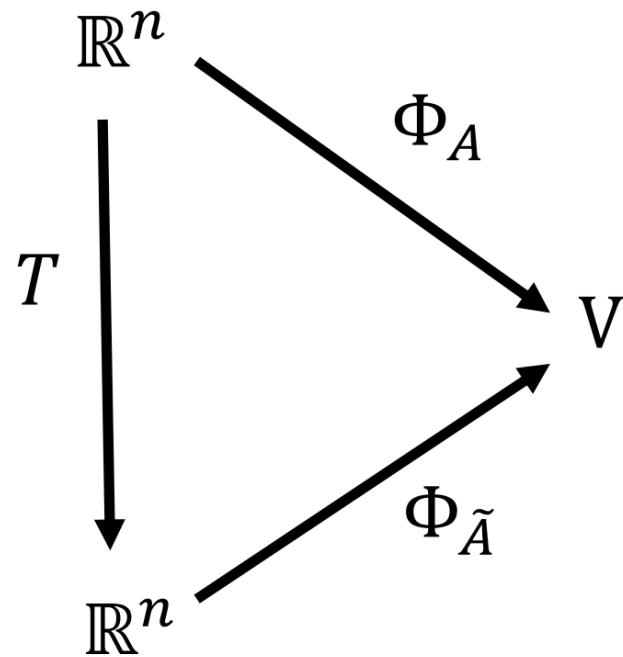
- $A = \{v_1, v_2, \dots, v_n\}$
- $\Phi_A(e_i) = v_i$
- M maps e_i to $\Phi_B^{-1} \circ L \circ \Phi_A(e_i)$

$$B = \{w_1, w_2, \dots, w_n\}$$

$$\Phi_B(e_i) = w_i$$

Basis Transformations

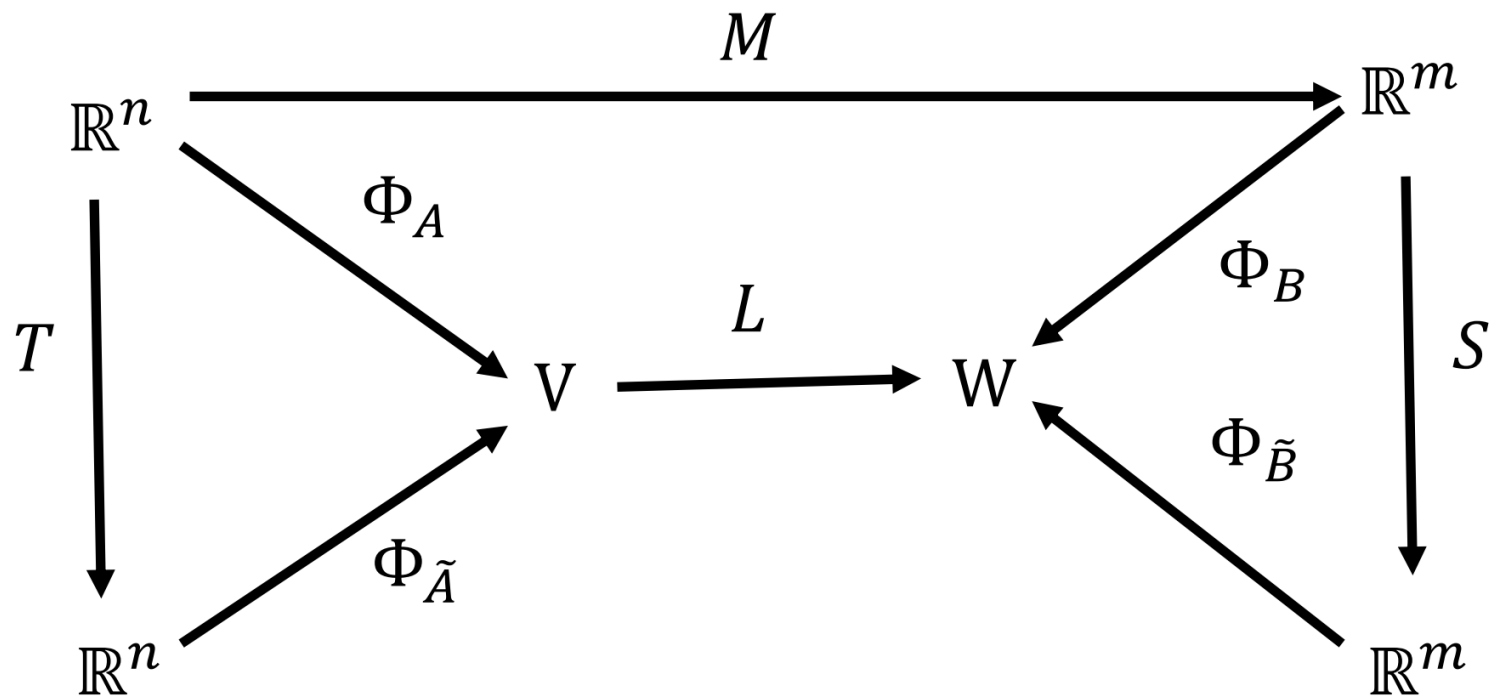
- Basis transformation



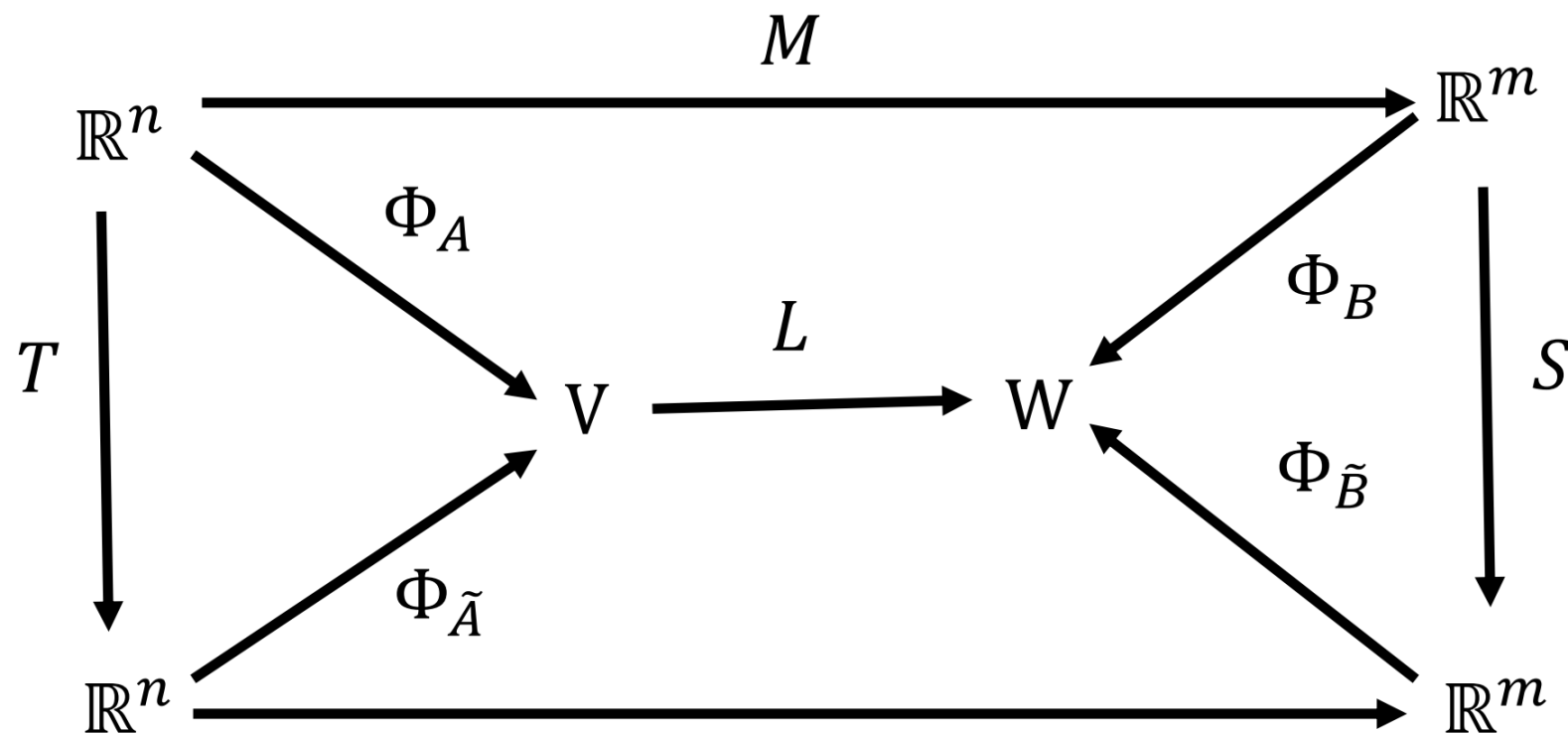
- $A = \{v_1, v_2, \dots, v_n\}$
- $\Phi_A(e_i) = v_i$
- T maps e_i to $\Phi_{\tilde{A}}^{-1} \circ \Phi_A(e_i)$

$$\tilde{A} = \{\tilde{v}_1, \tilde{v}_2, \dots, \tilde{v}_n\}$$
$$\Phi_{\tilde{A}}(e_i) = \tilde{v}_i$$

Basis Transformations



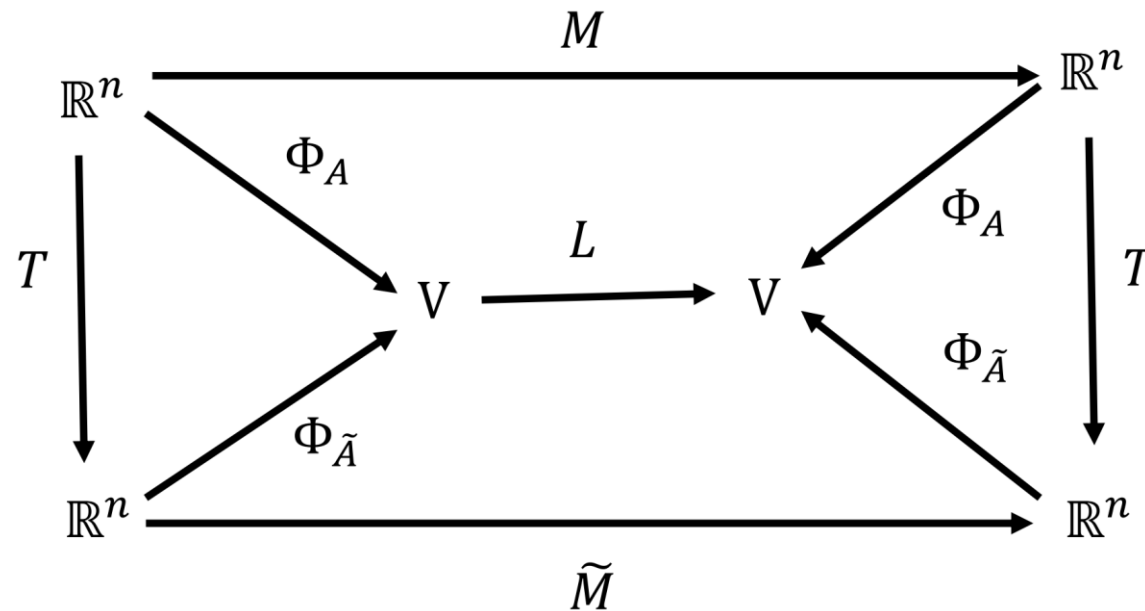
Basis Transformations



$$\tilde{M} = SMT^{-1}$$

Basis Transformations

In the special case that V equals W :



$$\tilde{M} = TMT^{-1}$$

Computer Aided Geometric Design

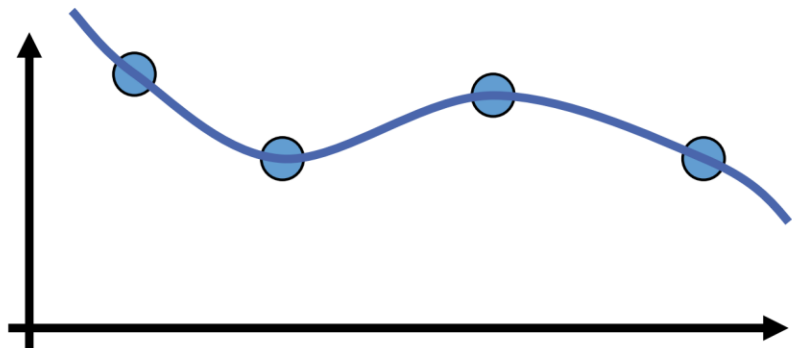
Fall Semester 2024

Interpolation & Approximation

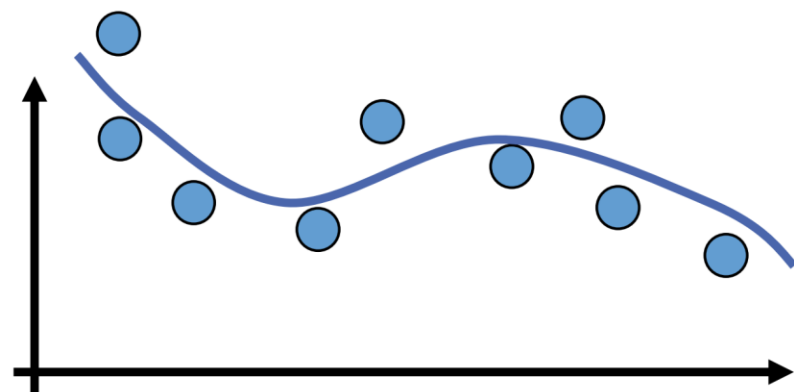
陈仁杰

renjiec@ustc.edu.cn

<http://staff.ustc.edu.cn/~renjiec>



Interpolation



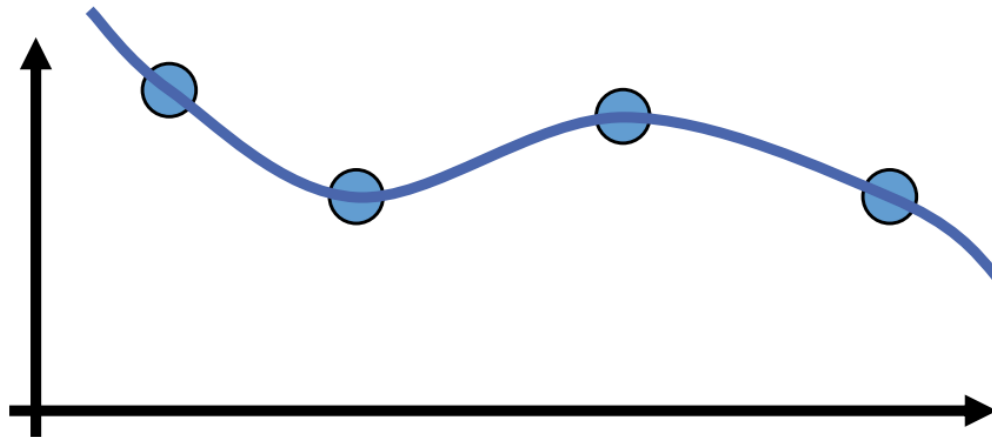
Approximation

Interpolation

General interpolation and polynomial interpolation

Interpolation Problem

- Our first attempt at modeling smooth objects:
 - Given a set of points along a curve or surface
 - Choose basis functions that span a suitable function space
 - Smooth basis functions
 - Any linear combination will be smooth, too
 - Find a linear combination such that the curve/surface interpolates the given points



General Formulation

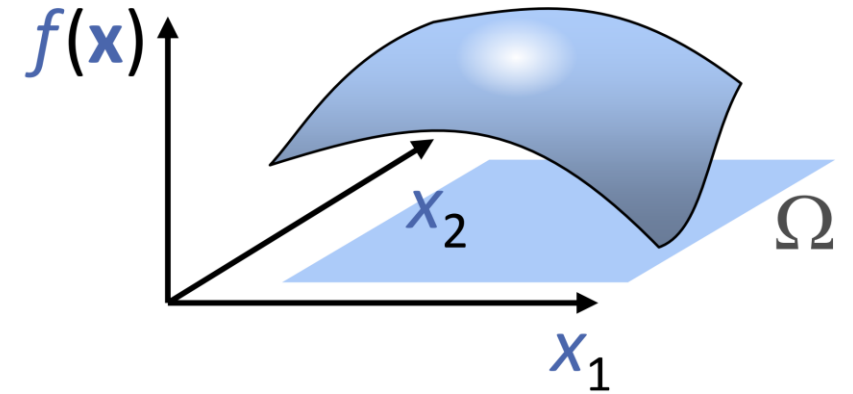
- Settings

- Domain $\Omega \subseteq \mathbb{R}^d$, mapping to \mathbb{R}
- Looking for a function $f: \Omega \rightarrow \mathbb{R}$
- Basis set: $B = \{b_1, \dots, b_n\}, b_i: \Omega \rightarrow \mathbb{R}$
- Represent f as linear combination of basis functions

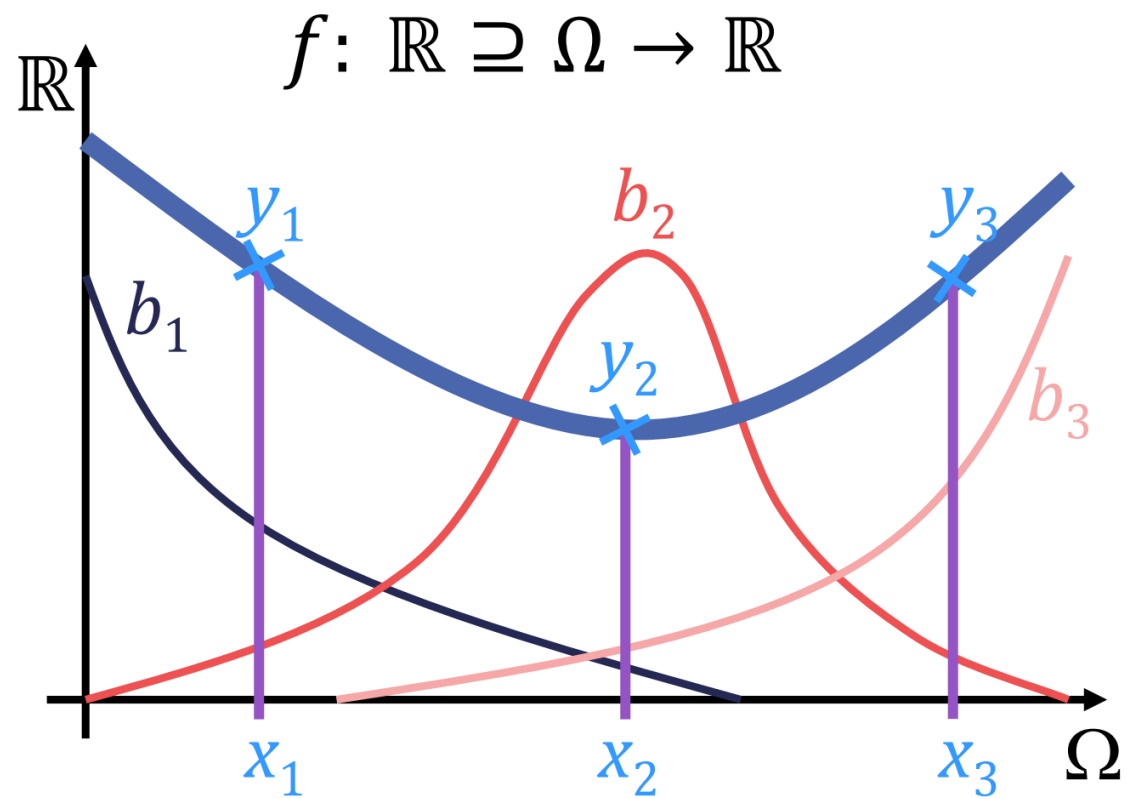
$$f_{\lambda}(x) = \sum_{k=0}^n \lambda_i b_i(x)$$

i.e. f is just determined by $\lambda = \begin{pmatrix} \lambda_1 \\ \dots \\ \lambda_n \end{pmatrix}$

- Function values: $\{(x_1, y_1), \dots, (x_n, y_n)\}, (x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$
- We want to find λ such that: $f_{\lambda}(x_i) = y_i$ for all i



Illustration



1D Example

Solving the Interpolation Problem

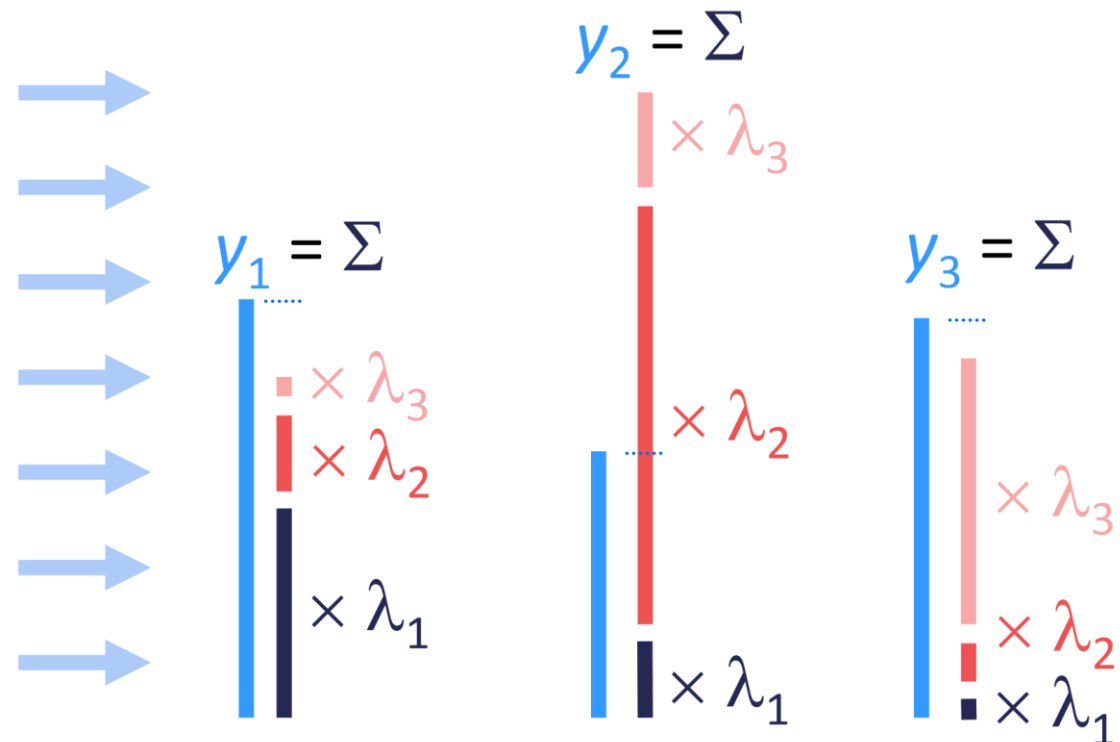
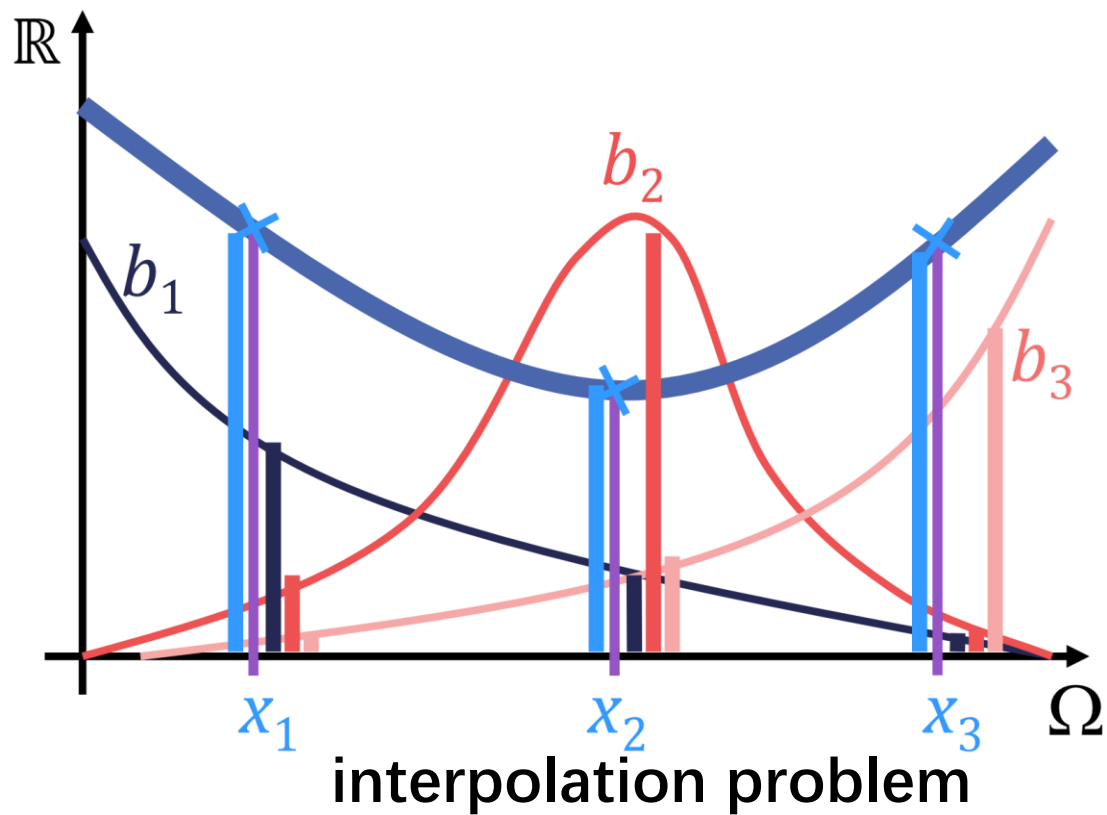
- Solution: linear system of equations
 - Evaluate basis functions at points x_i :

$$\forall i \in \{1, \dots, n\}: \sum_{i=1}^n \lambda_i b_i(x_i) = y_i$$

- Matrix form:

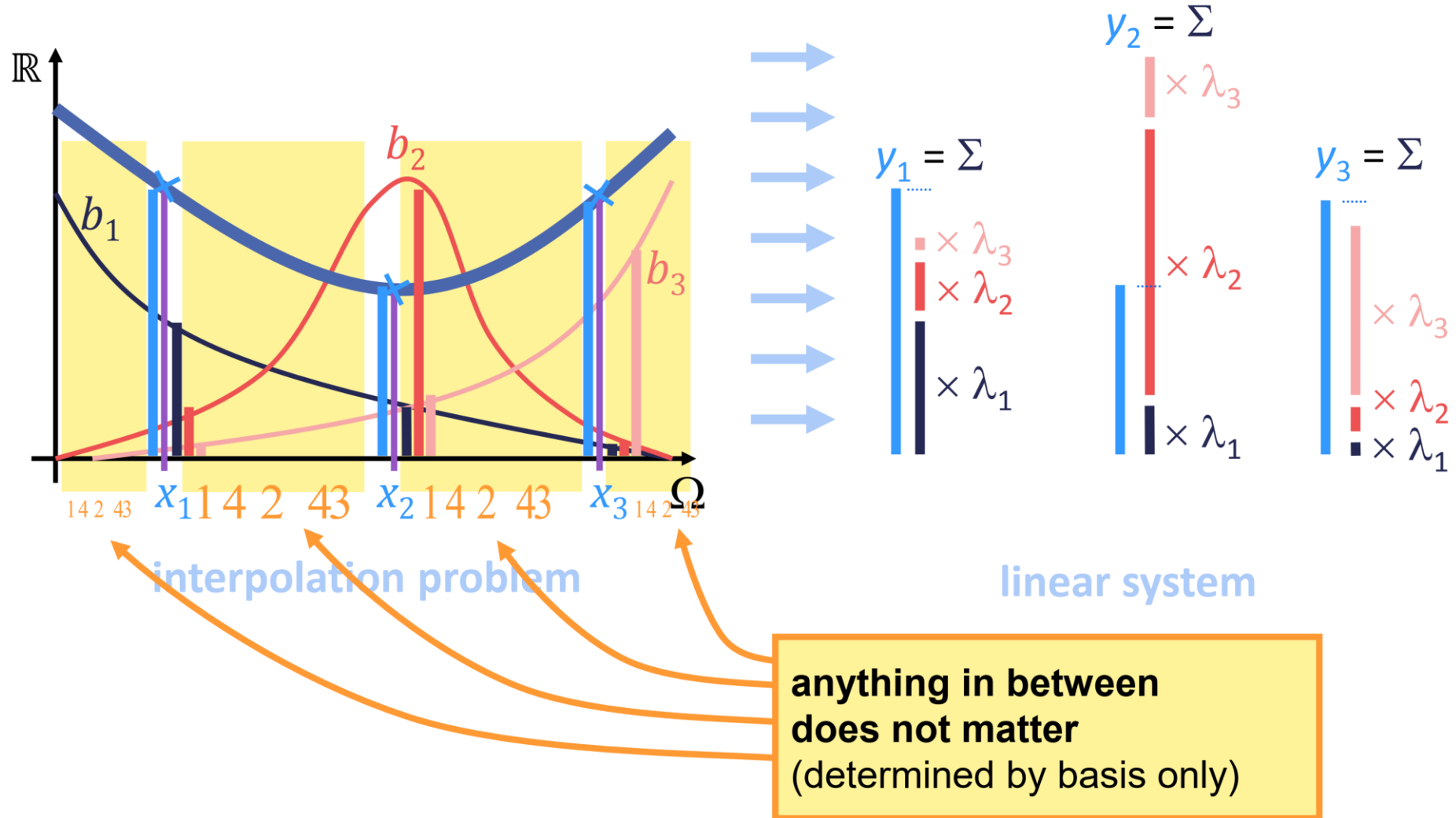
$$\begin{pmatrix} b_1(x_1) & \cdots & b_n(x_1) \\ \vdots & \ddots & \vdots \\ b_1(x_n) & \cdots & b_n(x_n) \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Illustration



linear system

Illustration



Example

Polynomial Interpolation

- Monomial basis $B = \{1, x, x^2, x^3, \dots, x^{n-1}\}$
- Linear system to solve

$$\begin{pmatrix} 1 & x_1 & \dots & x_1^{n-1} \\ 1 & x_2 & \dots & x_2^{n-1} \\ \dots & \dots & \dots & \dots \\ 1 & x_n & \dots & x_n^{n-1} \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \dots \\ \lambda_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix}$$

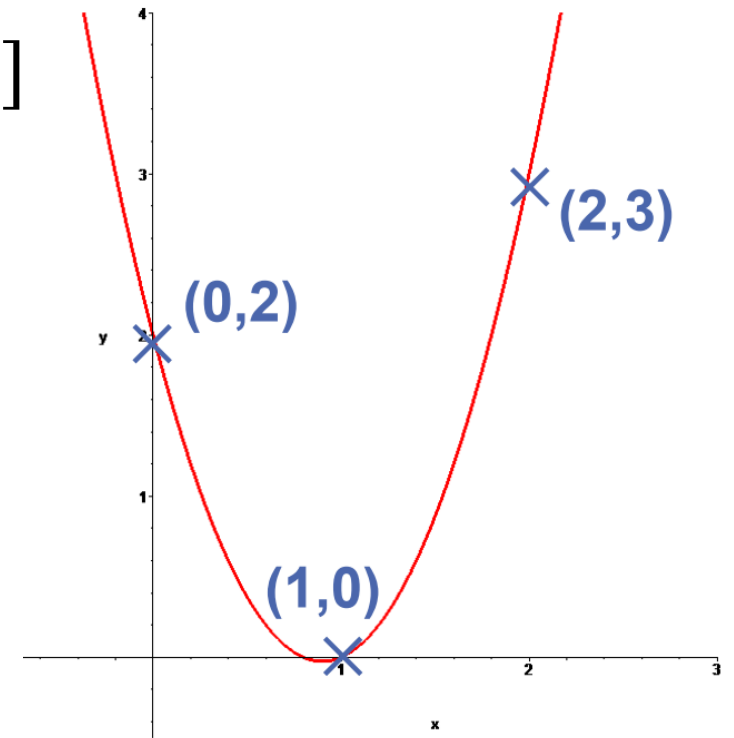
Vandermonde Matrix

Example with Numbers

- Quadratic monomial basis $B = \{1, x, x^2\}$
- Function values: $\{(0, 2), (1, 0), (2, 3)\} \quad [(x, y)]$
- Linear system to solve:

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 3 \end{pmatrix}$$

- Result: $\lambda_1 = 2, \lambda_2 = -\frac{9}{2}, \lambda_3 = \frac{5}{2}$




Problems with interpolation


- The arising system matrix is generally dense
- Depending on the choice of the basis, the matrix can be ill-conditioned (difficult to invert/solve)

ill-conditioning example

- Consider the system
 - Clearly (1,1) is a solution
- Now perturb the right hand side of the second equation by 0.001 (order 10^{-3})
 - The solution is then (0.000,3.000) (order 1)
- Now consider perturbing the coefficient
 - The solution (2.000, -1.000)

$$\begin{aligned}x_1 + 0.5x_2 &= 1.5 \\0.667x_1 + 0.333x_2 &= 1\end{aligned}$$

$$\begin{aligned}x_1 + 0.5x_2 &= 1.5 \\0.667x_1 + 0.333x_2 &= 0.999\end{aligned}$$


$$\begin{aligned}x_1 + 0.5x_2 &= 1.5 \\0.667x_1 + 0.334x_2 &= 1\end{aligned}$$


ill-conditioning

- Small change in the input data induces relatively large change in the output (solution)
- Thinking of equations as lines (hyperplanes), when the system is ill-conditioned the lines become almost parallel
 - Obtaining a solution (intersection) becomes difficult and imprecise

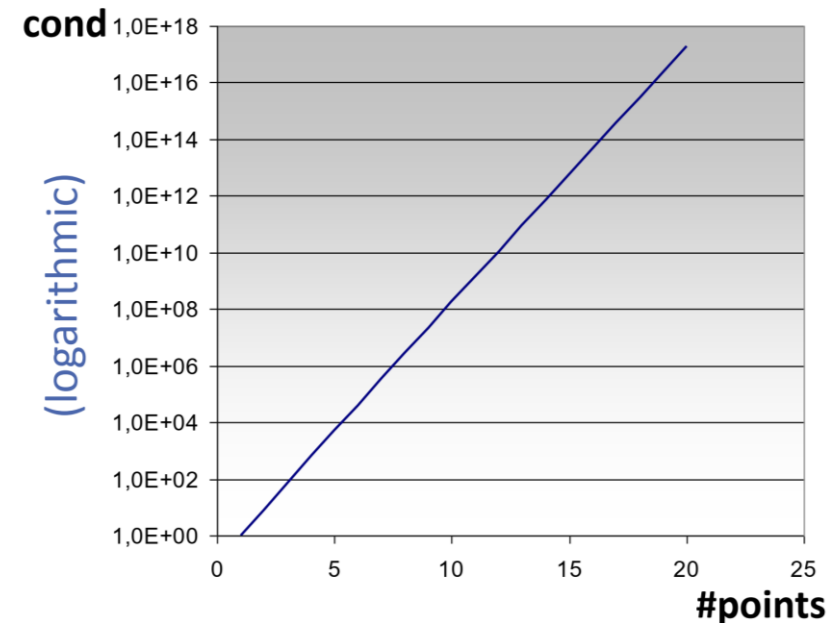
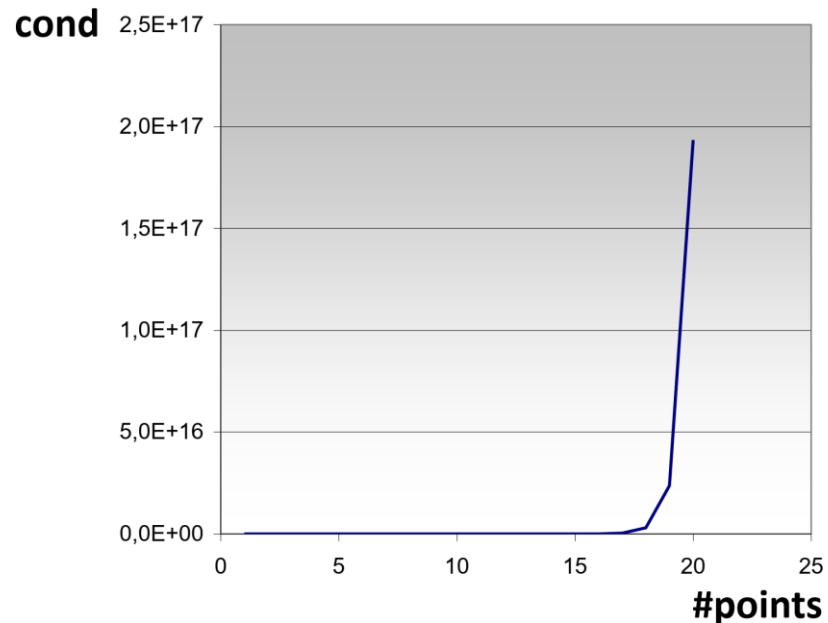
Condition number

$$\kappa_2(A) = \frac{\max_{x \neq 0} \frac{\|Ax\|}{\|x\|}}{\min_{x \neq 0} \frac{\|Ax\|}{\|x\|}}$$

- Can be regarded as the ratio of highest eigenvalues / lowest eigenvalue
- When the condition number is high it reflects there is too much interdependence between the elements of the basis

Condition Number...

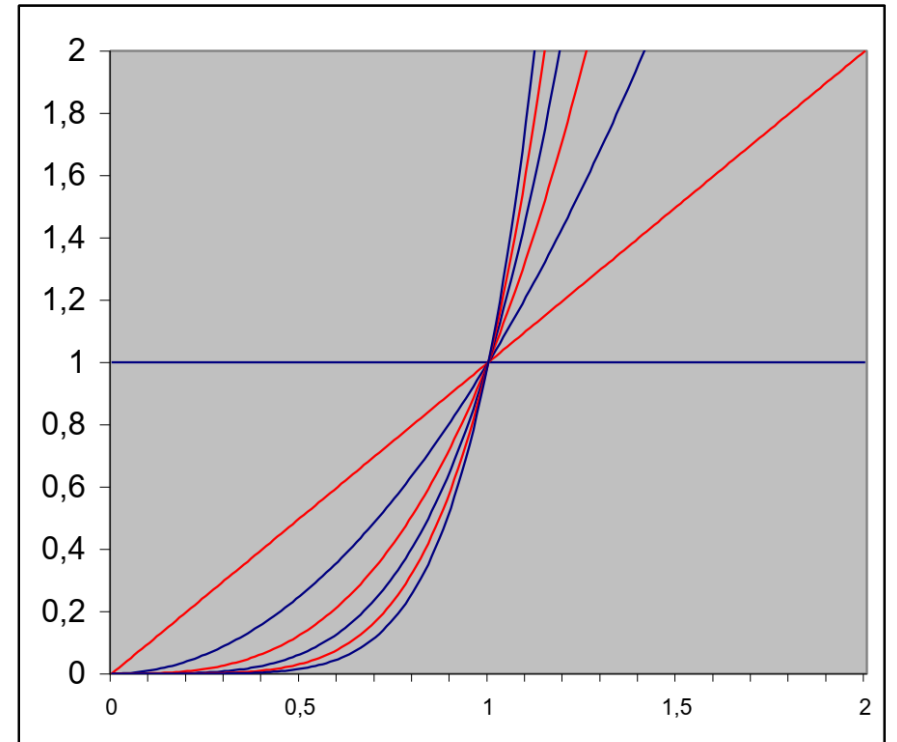
- The interpolation problem is ill conditioned:
- For equidistant x_i , the condition number of the Vandermode matrix grows exponentially with n
 - (maximum degree+1 = number of points to interpolate)



Why is that??

Monomial Basis:

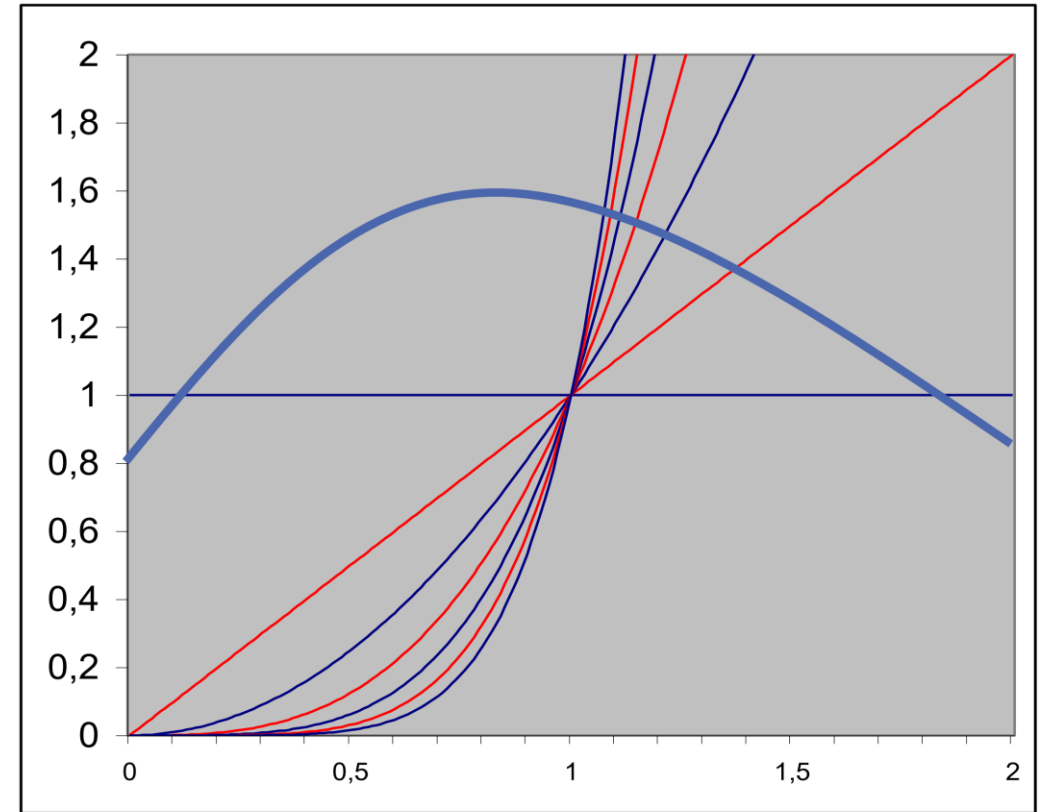
- Functions become increasingly indistinguishable with degree
- Only differ in growing rate
 - x^i grows faster than x^{i-1}



Monomial basis

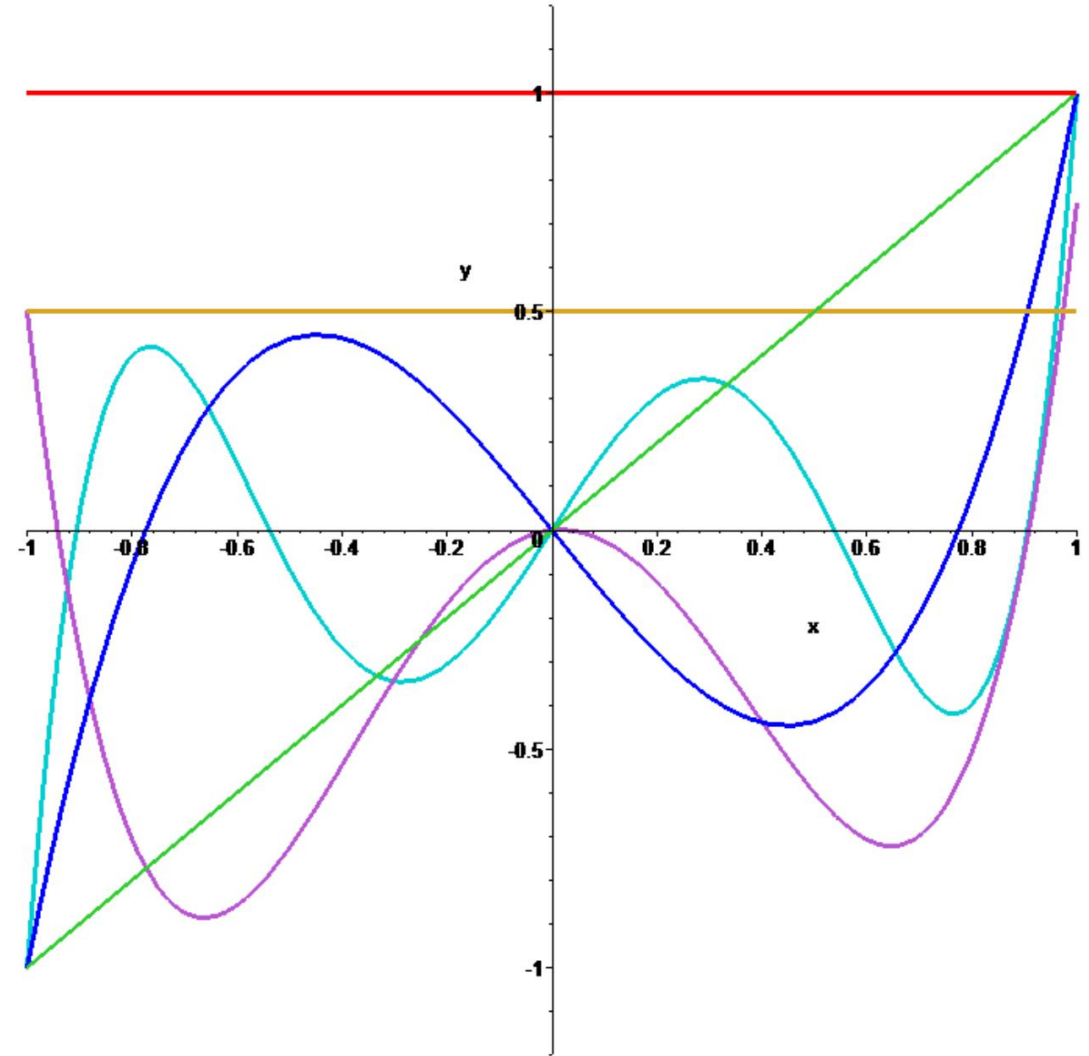
Cancellation

- Monomials:
- From left to right in x- direction...
 - First 1 dominates
 - Then x grows faster
 - Then x^2 grows faster
 - Then x^3 grows faster
 - ...
- Tendency:
 - Well behaved functions often require alternating sequence of coefficients (left turn, right turn, left turn,...)
 - *Cancellation* problems



The Cure...

- This problem can be fixed:
 - Use orthogonal polynomial basis
 - How to get one? \rightarrow e.g. Gram-Schmidt orthogonalization



Alternative approach

- Can we avoid solving a system in the first place?

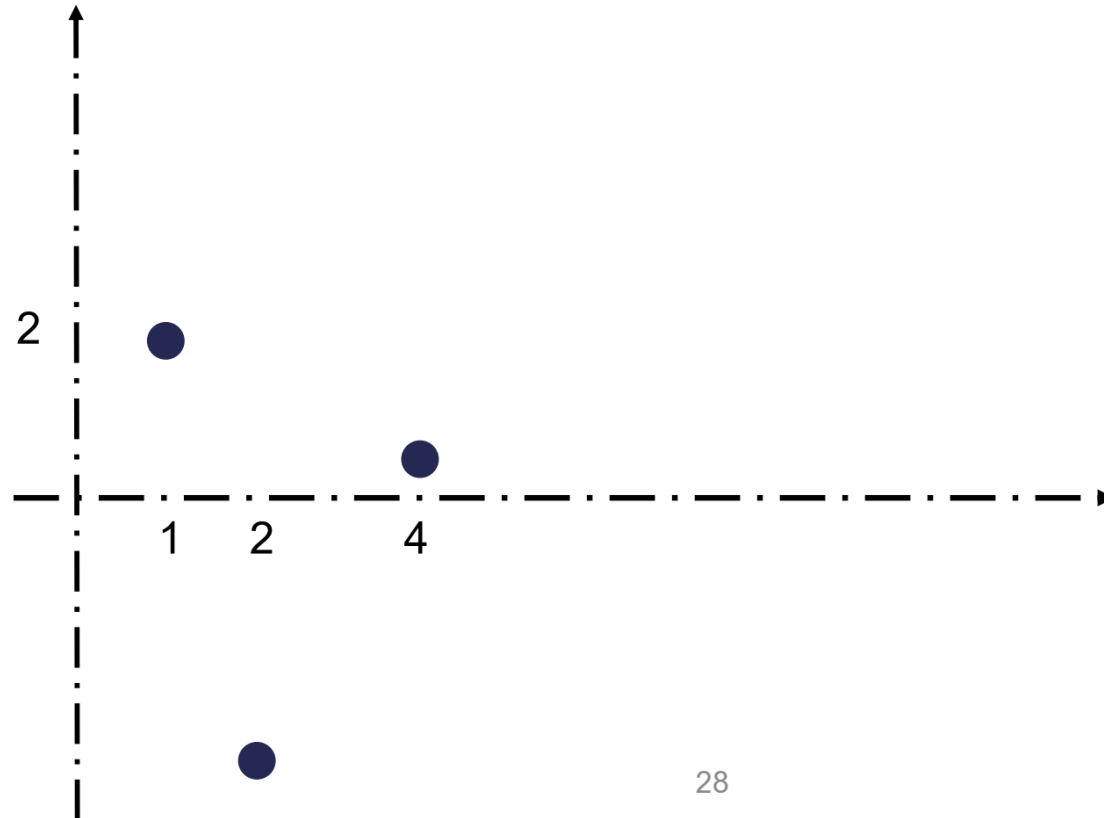
Alternative approach

- Can we avoid solving a system in the first place?

Think of a **different basis!**

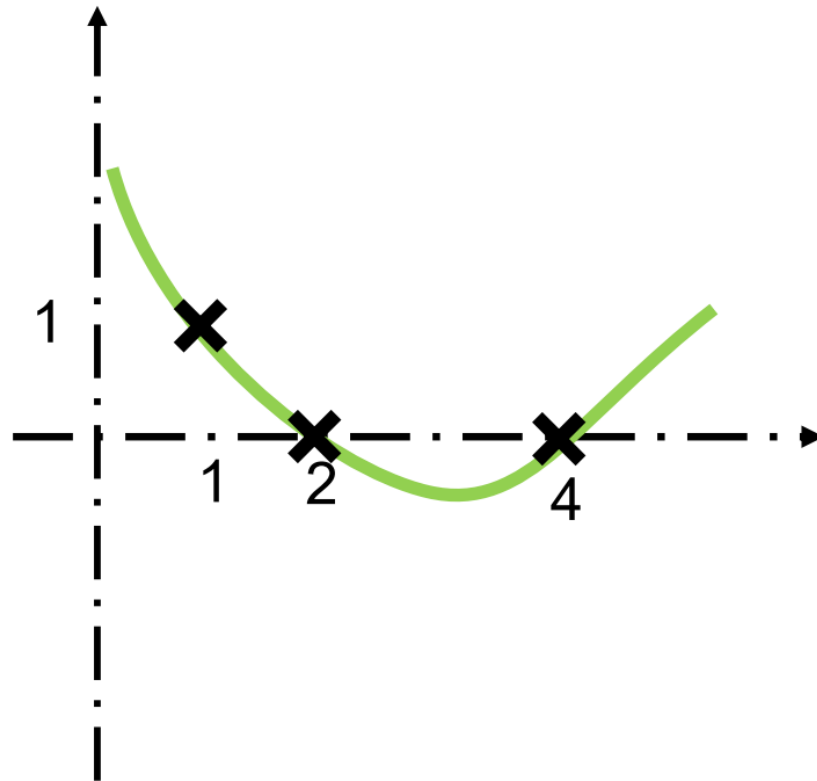
Alternative approach: Example

- Pass a quadratic polynomial through $(1, 2)$, $(2, -3)$, $(4, 0.5)$



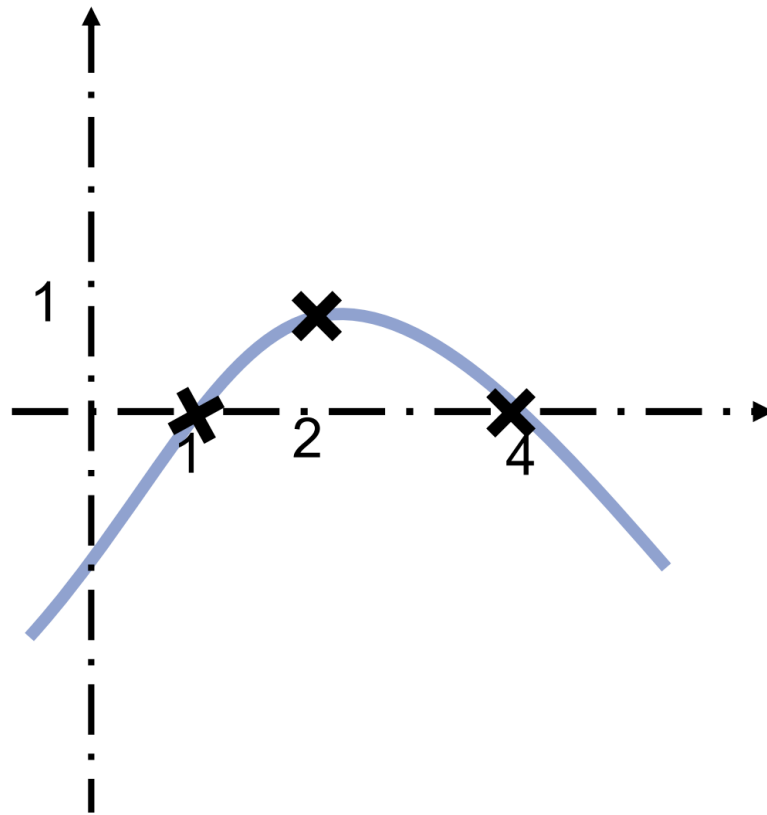
Alternative approach: Example

- Assume we can construct a quadratic polynomial $P_0(x)$ such that it is equal to 1 at x_0 , and equals zero at the other two points x_1, x_2 :



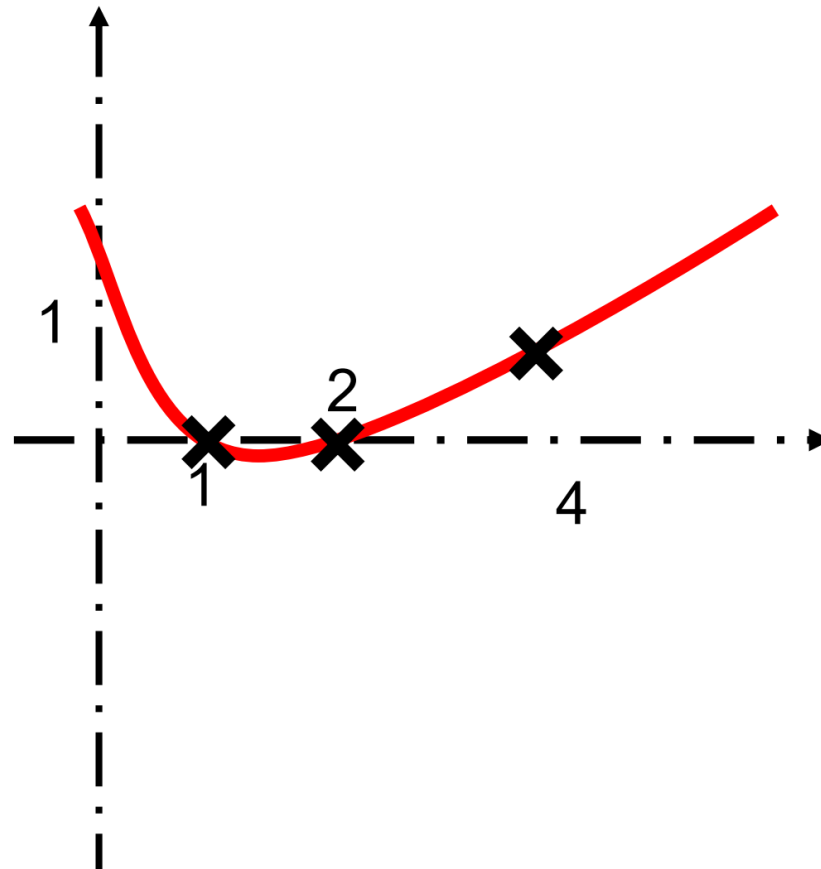
Alternative approach: Example

- $P_1(x)$ is constructed similarly and set equal to 1 at location x_1 , and to zero at x_0, x_2 :



Alternative approach: Example

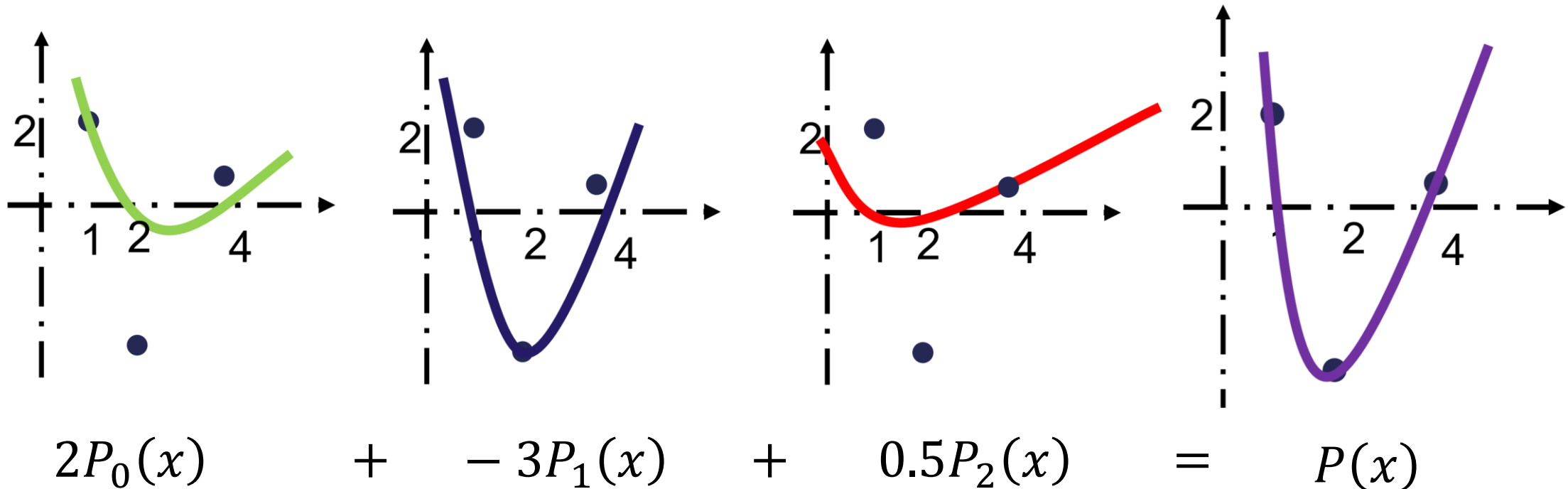
- $P_2(x)$ is set equal to 1 at location x_2 , and to zero at x_0, x_1



Alternative approach: Example

- Now, the idea is to scale each $P_i(x)$ such that $P_i(x_i) = y_i$ and add them all together:

$$P(x) = y_0P_0(x) + y_1P_1(x) + y_2P_2(x)$$



Alternative approach: general case

- Construction of general solution to the interpolation problem:
 - For a set of $n + 1$ points $\{(x_0, y_0), \dots, (x_n, y_n)\}$, we seek a basis of polynomials l_i of degree n such that

$$l_i(x_j) = \begin{cases} 1, & \text{若 } i = j \\ 0, & \text{若 } i \neq j \end{cases}$$

- The solution to the interpolation problem is then given as

$$P(x) = y_0 l_0(x) + y_1 l_1(x) + \dots + y_n l_n(x) = \sum_{i=0}^n y_i l_i(x)$$

Alternative approach: general case

- How can we find the polynomials $l_i(x)$?
- They are polynomials of degree n and have the following n roots

$$x_0, x_1, x_2, \dots, x_{i-1}, x_{i+1}, \dots, x_n$$

- They can be expressed as

$$\begin{aligned} l_i(x) &= C_i(x - x_0)(x - x_1) \dots (x - x_{i-1})(x - x_{i+1}) \dots (x - x_n) \\ &= C_i \prod_{j \neq i} (x - x_j) \end{aligned}$$

- Since $l_i(x_i) = 1$

$$1 = C_i \prod_{j \neq i} (x_i - x_j) \Rightarrow C_i = \frac{1}{\prod_{j \neq i} (x_i - x_j)}$$

Alternative approach: general case

- Finally we have

$$l_i(x) = \frac{\prod_{j \neq i} (x - x_j)}{\prod_{j \neq i} (x_i - x_j)}$$

- The polynomials $l_i(x)$ are called Lagrange polynomials

Question

- Is the solution to the interpolation problem obtained using the **Lagrange polynomials** different from the solution obtained using the Vandermonde matrix (**monomial basis**)?

Computer Aided Geometric Design

Fall Semester 2024

Bézier Curves

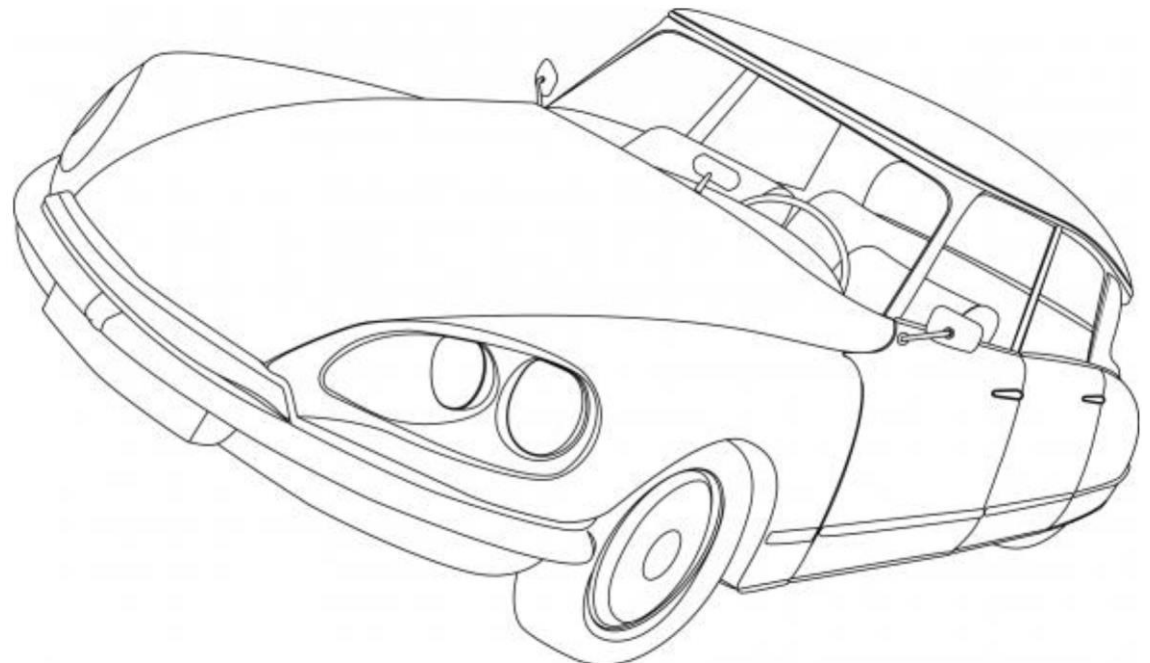
陈仁杰

renjiec@ustc.edu.cn

<http://staff.ustc.edu.cn/~renjiec>

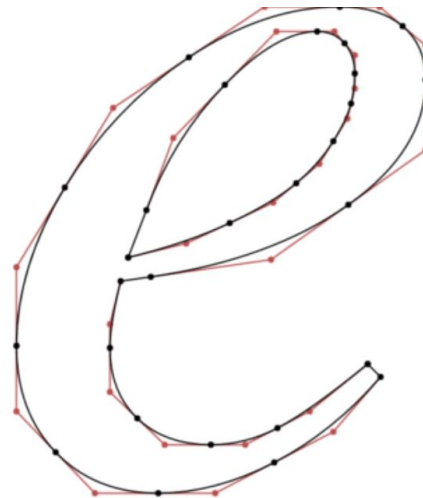
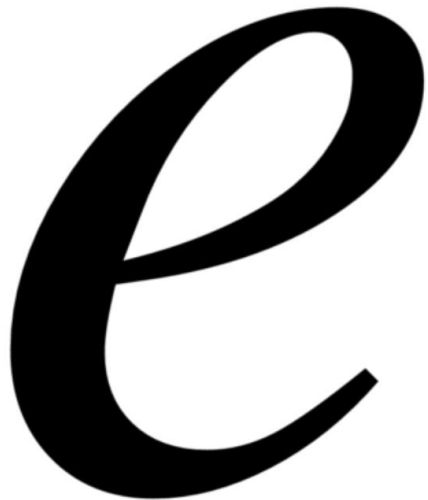
Bézier curves

- Bézier curves/splines developed by
 - Paul de Casteljau at Citroen (1959)
 - Pierre Bézier at Renault (1963)for free-form parts in automotive design



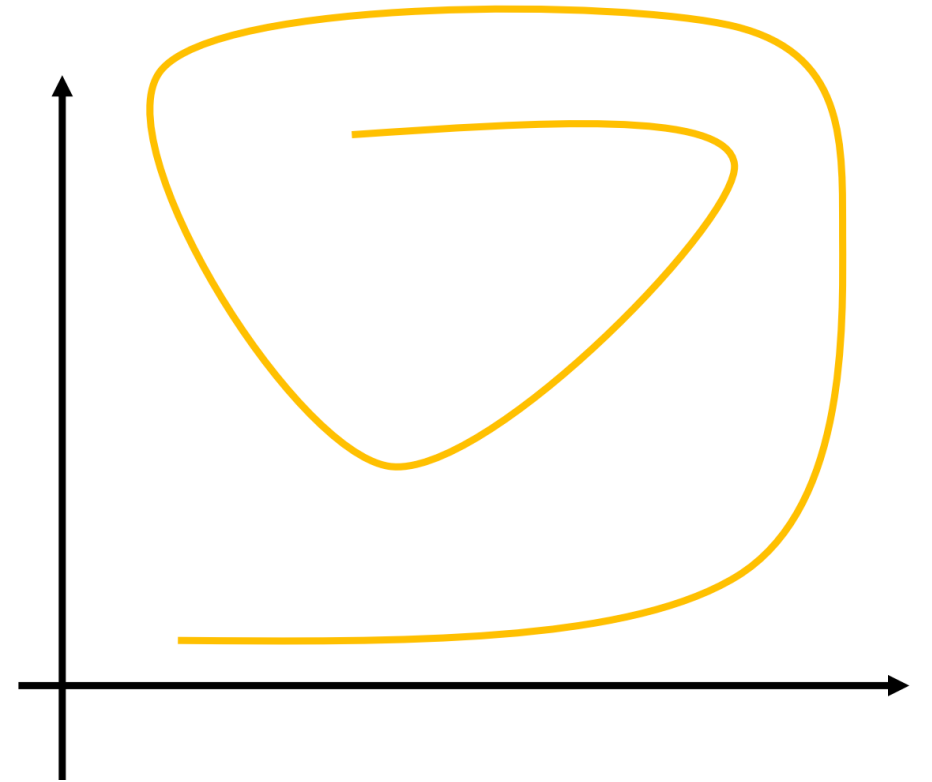
Bézier curves

- Today: Standard tool for 2D curve editing
- Cubic 2D Bézier curves are everywhere:
 - Inkscape, Corel Draw, Adobe Illustrator, Powerpoint, ...
 - PDF, Truetype (quadratic curves), Windows GDI, ...
- Widely used in 3D curve & surface modeling as well



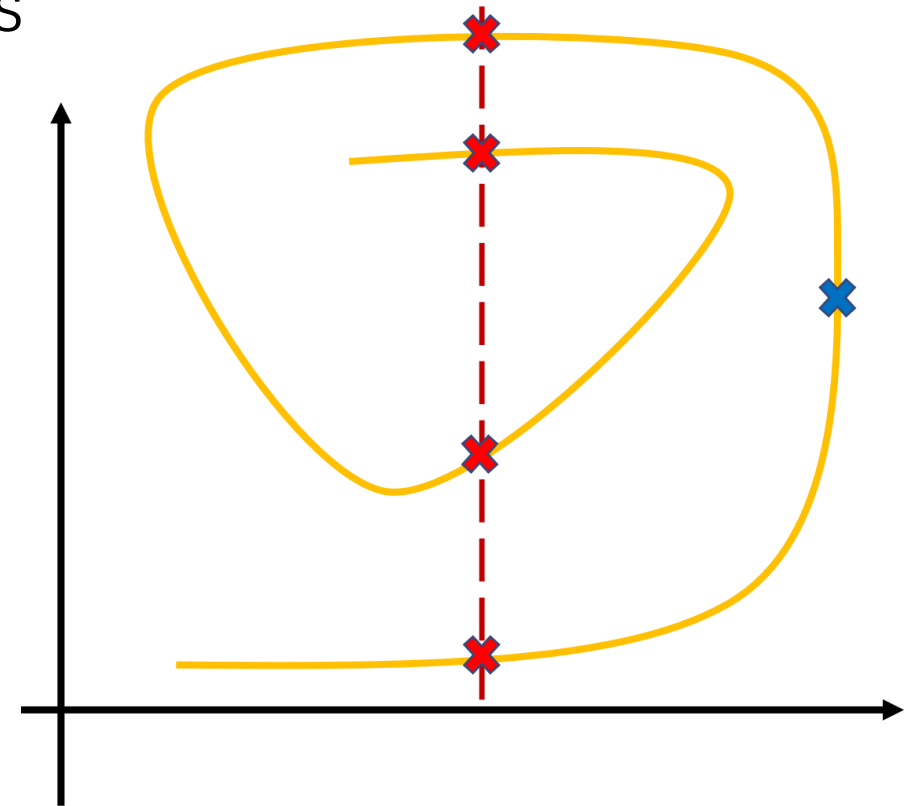
Curve representation

- The implicit curve form $f(x, y) = 0$ suffers from several limitations:



Curve representation

- The implicit curve form $f(x, y) = 0$ suffers from several limitations:
 - Multiple values for the same x -coordinates
 - Undefined derivative $\frac{dy}{dx}$ (see blue cross)
 - Not invariant w.r.t axes transformations

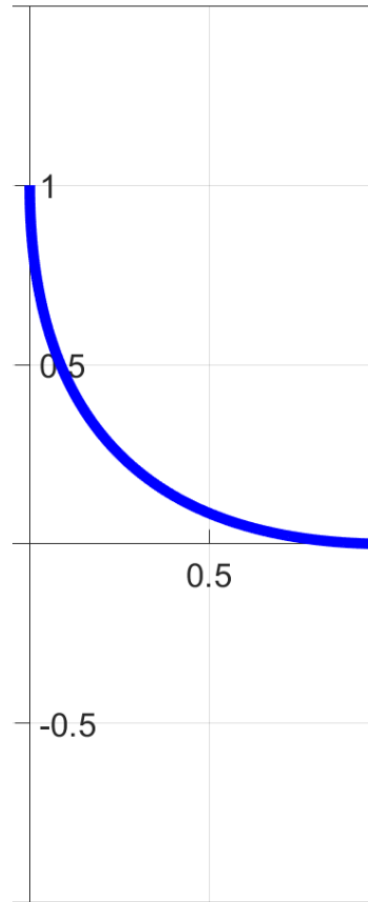


Parametric representation

- Remedy: parametric representation $c(t) = (x(t), y(t))$
 - Easy evaluations
 - The parameter t can be interpreted as time
 - The curve can be interpreted as the path traced by a moving particle

Modeling with the power basis, ...

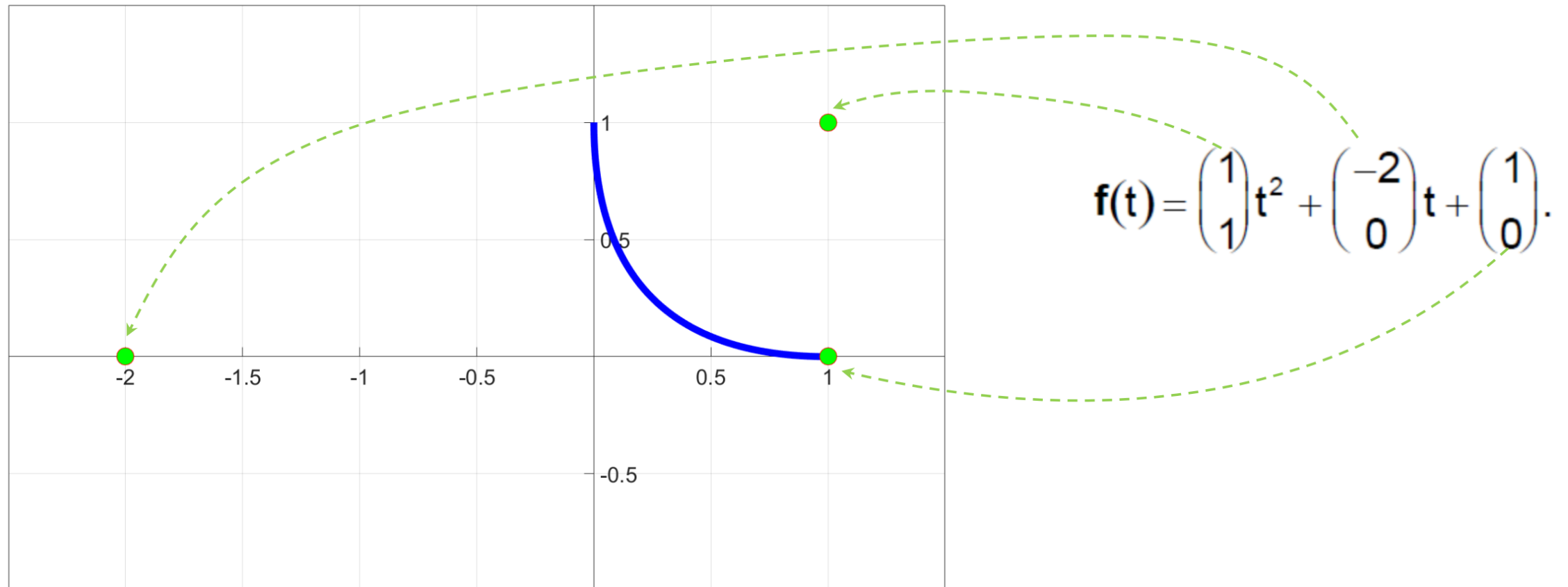
- Example of a parabola: $f(t) = at^2 + bt + c$



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

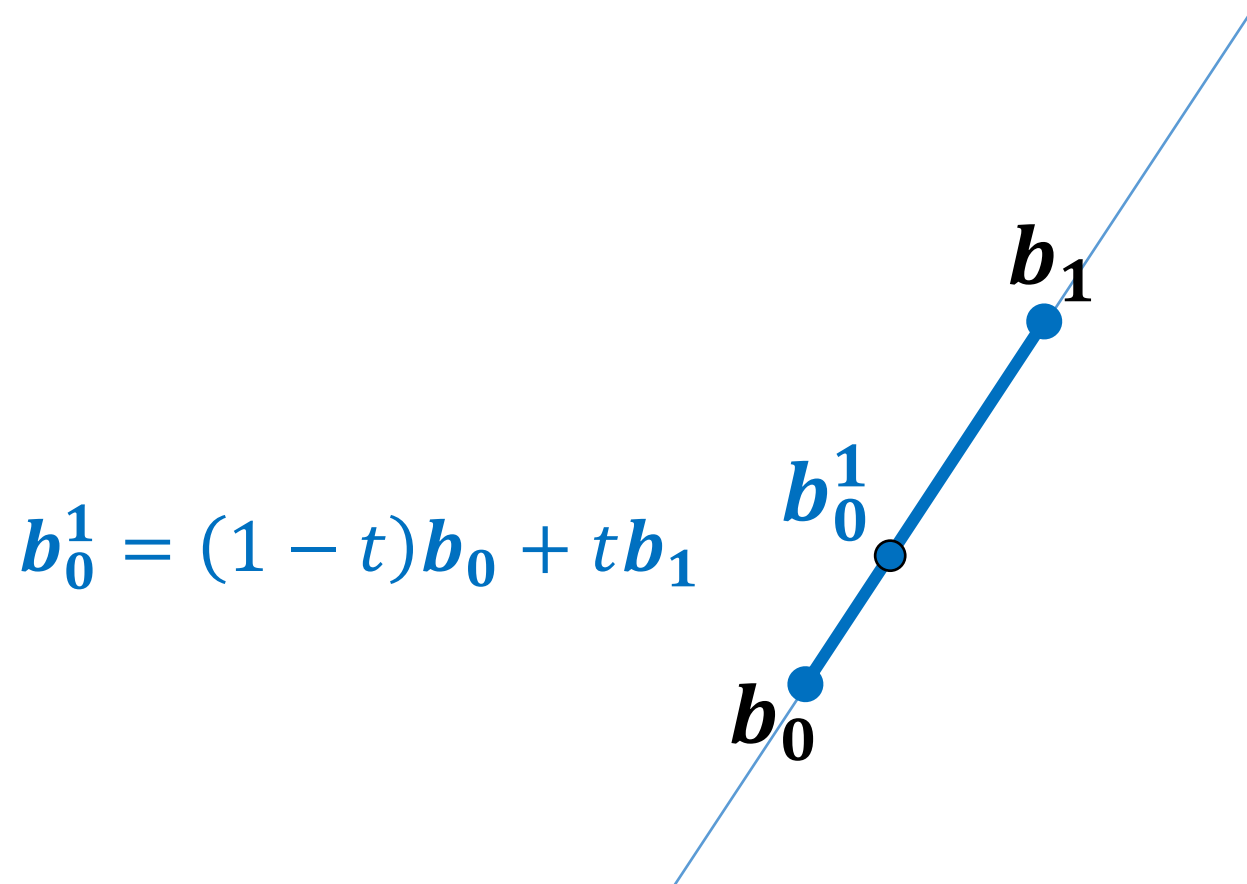
Modeling with the power basis, ... no thanks!

- Examples of a parabola: $f(t) = at^2 + bt + c$: the coefficients of the power basis lack intuitive geometric meaning



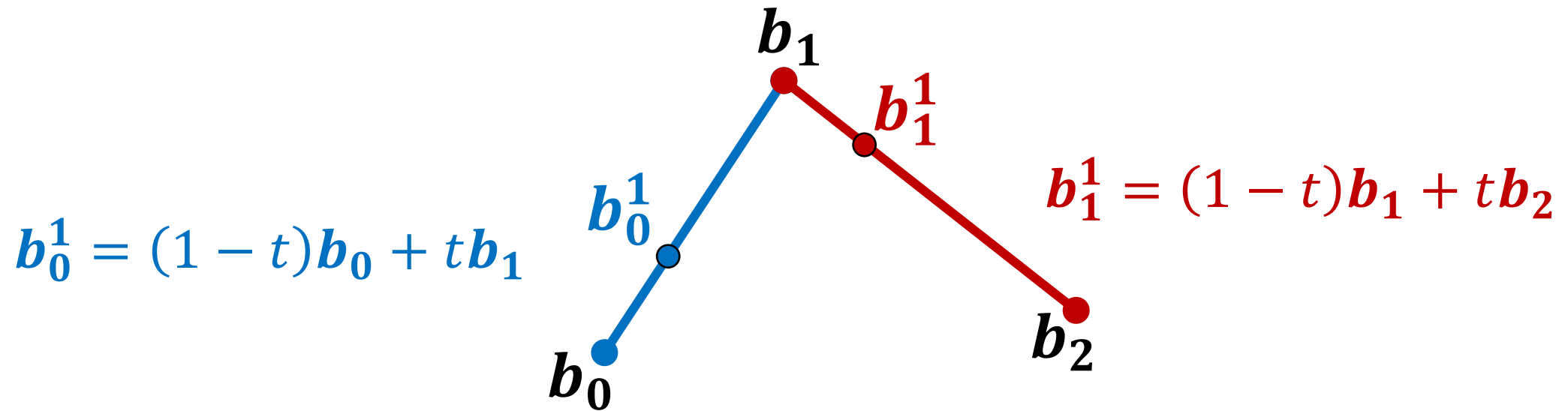
Back to the drawing board

- A point on a parametric line



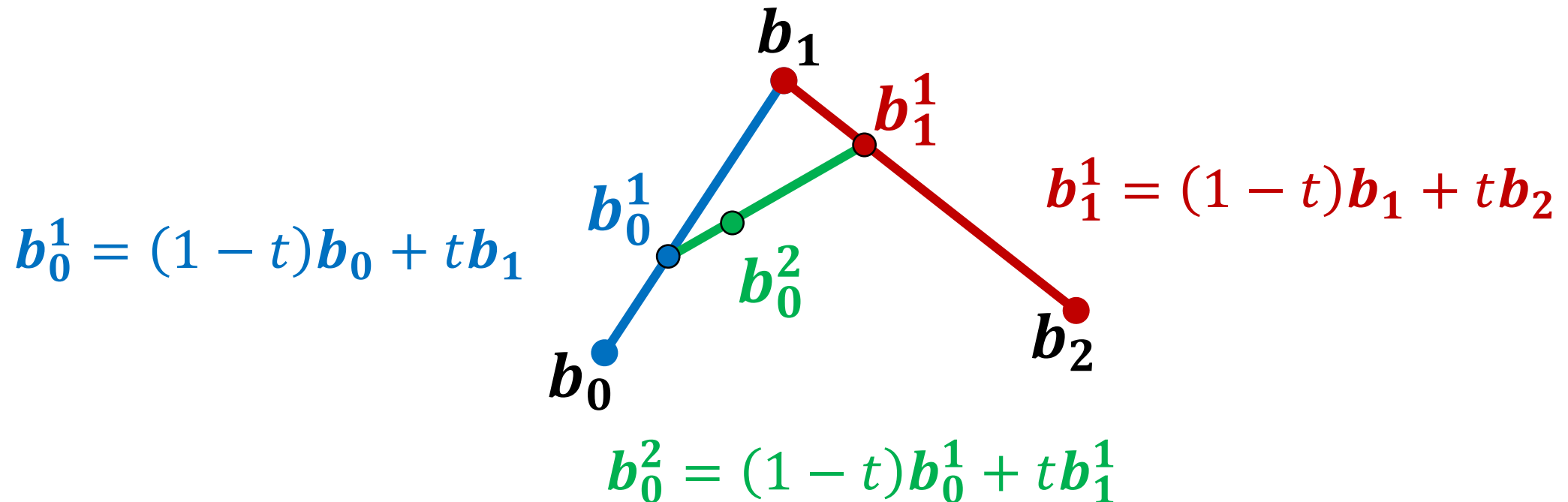
Back to the drawing board

- Another point on a second parametric line



Back to the drawing board

- A third point on the line defined by the first two points



Back to the drawing board

- And then simplify...

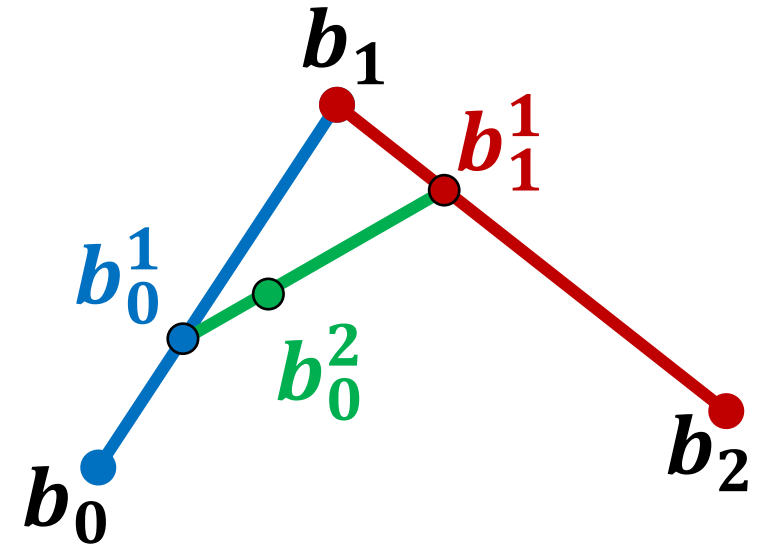
$$b_0^1 = (1 - t)b_0 + tb_1$$

$$b_0^2 = (1 - t)b_0^1 + tb_1^1$$

$$b_1^1 = (1 - t)b_1 + tb_2$$

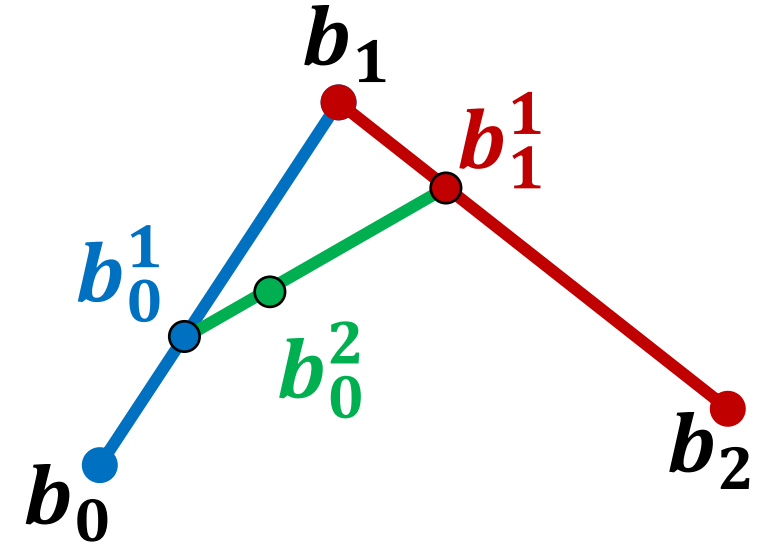
$$b_0^2 = (1 - t)[(1 - t)b_0 + tb_1] + t[(1 - t)b_1 + tb_2]$$

$$b_0^2 = (1 - t)^2 b_0 + 2t(1 - t)b_1 + t^2 b_2$$



Back to the drawing board

- We obtained another description of parabolic curves
- The coefficients b_0, b_1, b_2 have a geometric meaning



$$b_0^2 = (1 - t)^2 b_0 + 2t(1 - t)b_1 + t^2 b_2$$

Example re-visited

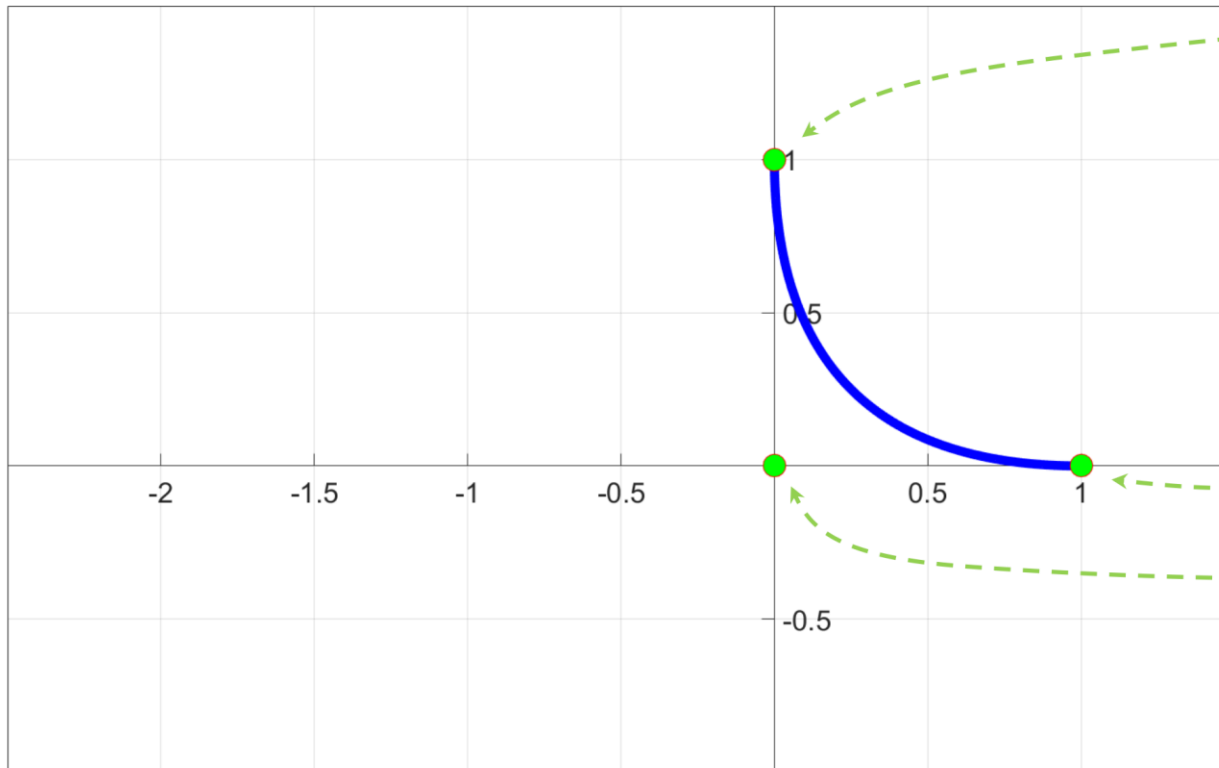
- Let's rewrite our initial parabolic curve example in the new basis

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

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

Example re-visited

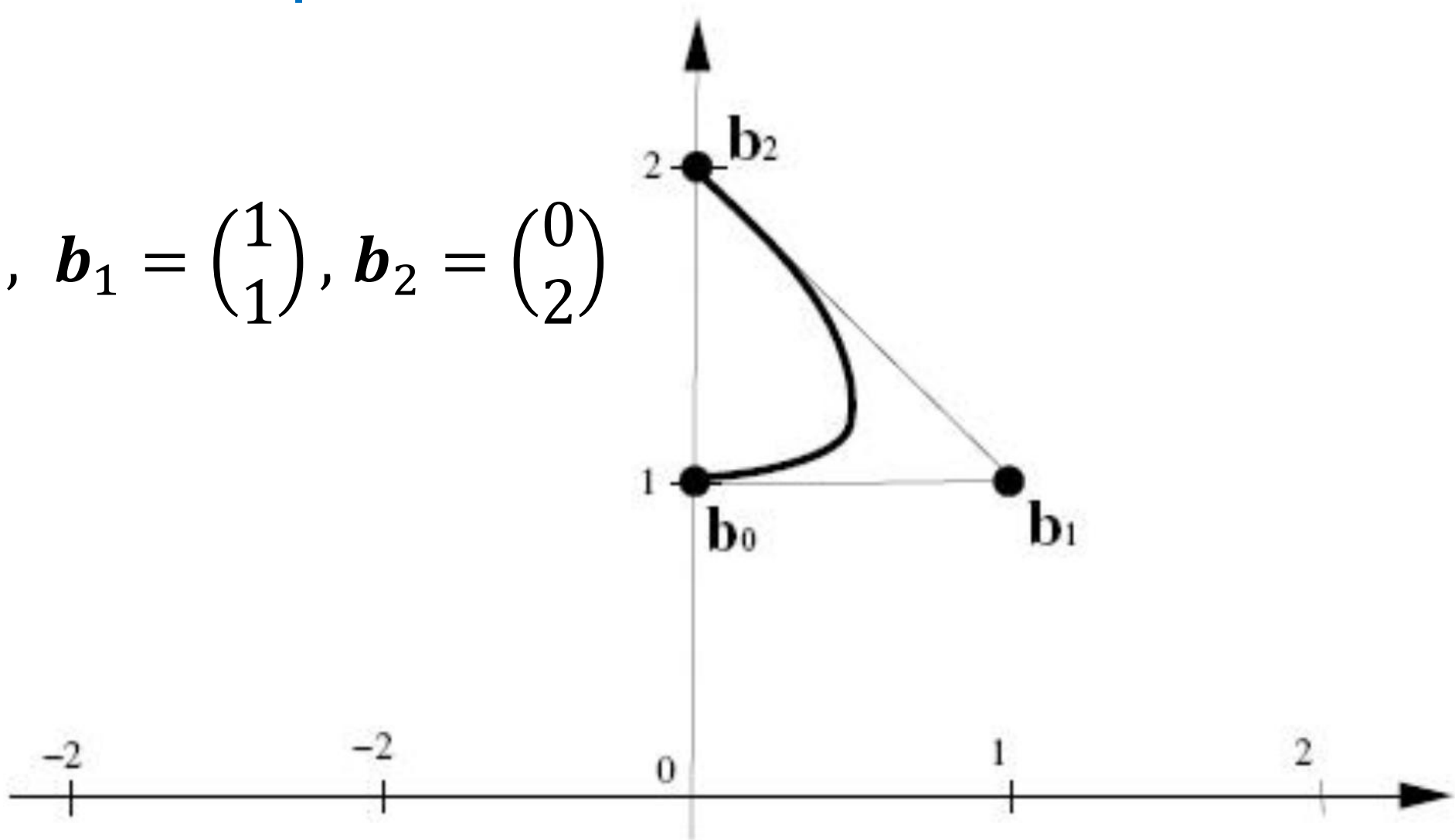
- The coefficients have a geometric meaning
- More intuitive for curve manipulation



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

Another example

$$\mathbf{b}_0 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \mathbf{b}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \mathbf{b}_2 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$



Going further

- Cubic approximation

- Given 4 points: $\mathbf{p}_0^0(t) = \mathbf{p}_0, \mathbf{p}_1^0(t) = \mathbf{p}_1, \mathbf{p}_2^0(t) = \mathbf{p}_2, \mathbf{p}_3^0(t) = \mathbf{p}_3$

- First iteration $\mathbf{p}_0^1 = (1 - t)\mathbf{p}_0 + t\mathbf{p}_1$

$$\mathbf{p}_1^1 = (1 - t)\mathbf{p}_1 + t\mathbf{p}_2$$

$$\mathbf{p}_2^1 = (1 - t)\mathbf{p}_2 + t\mathbf{p}_3$$

- 2nd iteration $\mathbf{p}_0^2 = (1 - t)^2\mathbf{p}_0 + 2t(1 - t)\mathbf{p}_1 + t^2\mathbf{p}_2$

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

- Curve

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

Throughout these examples, we just re-invented a primitive version of the de Casteljau algorithm

Now let's examine it more closely ...

CAGD杂志将出版专辑，纪念Paul de Casteljau的开创性贡献

原创 ggc 图形学与几何计算 2022-09-18 15:58 发表于北京

收录于合集

#图形资讯

62个 >

2022年3月24日，CAGD的先驱之一，长期在法国雪铁龙公司工作的Paul de Faget de Casteljau先生不幸逝世。为了纪念他的开创性贡献，CAGD杂志准备出版一期专辑怀念他，欢迎投稿！

de Casteljau先生的历史性贡献

de Casteljau先生于1930年11月19日出生于法国的Besançon，是一位法国的物理学家和数学家，任职于雪铁龙公司，研究汽车外形设计的算法和系统。他和法国另一个汽车公司雷诺公司的工程师Pierre Bézier，分别独立地发展了一套后来被称为Bézier曲线曲面的理论。

de Casteljau先生因为他名字命名的de Casteljau算法闻名，对于一条 $n+1$ 个控制顶点的Bézier曲线，

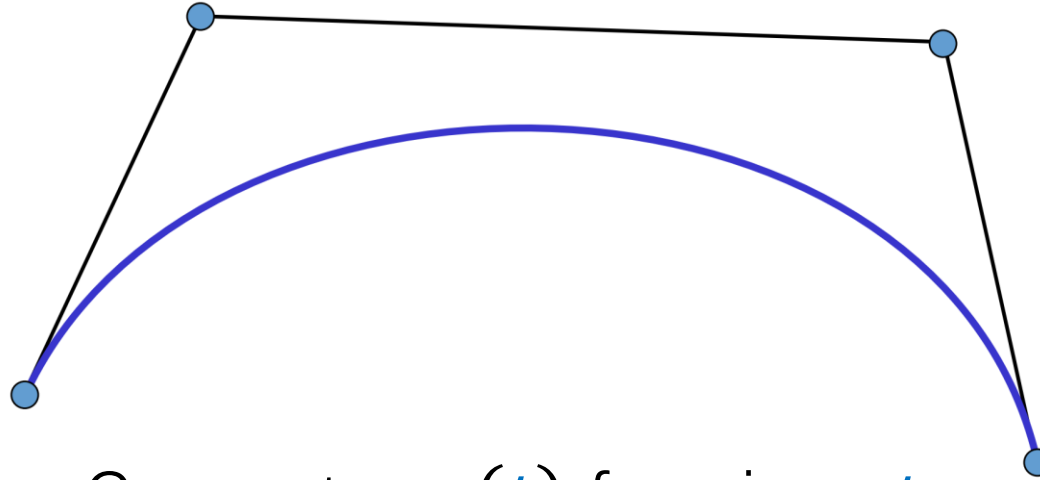
$$P(t) = \sum_{i=0}^n P_i B_{i,n}(t), \quad t \in [0,1]$$

曲线上参数 t 对应的型值点可由如下递归算法计算：|

$$P_i^k = \begin{cases} P_i & k = 0 \\ (1-t)P_i^{k-1} + tP_{i+1}^{k-1} & k = 1, 2, \dots, n, \\ & i = 0, 1, \dots, n-k \end{cases}$$

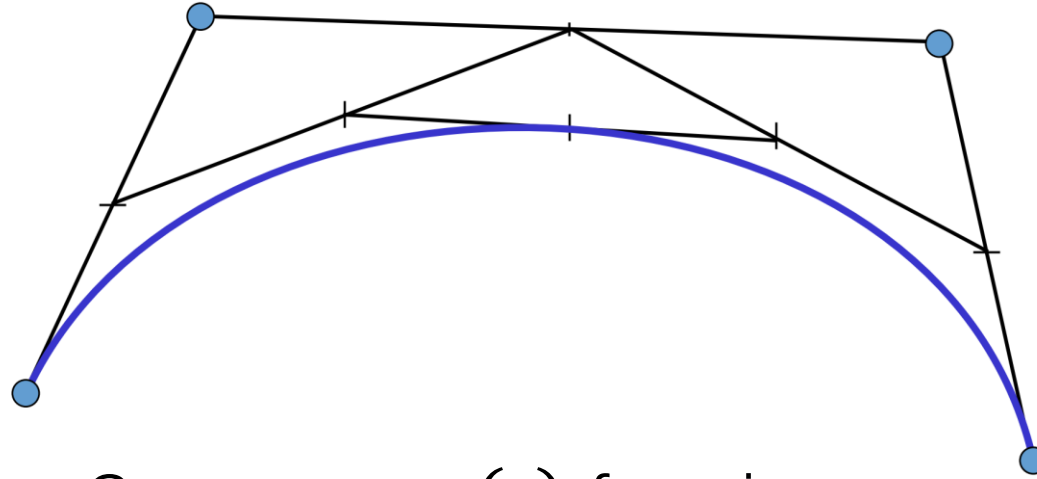
de Casteljau先生于2012年荣获Bézier奖。这是几何造型领域的最高奖，于2007由Solid Modeling Association (SMA) 设立，并以另一位CAGD先驱Pierre Bézier的名字命名。由Vadim Shairo (主席), Pere Brunet, Christoph Hoffmann, Shi-Min Hu, Kunwoo Lee, Diensh Manocha和Malcolm Sabin组成的Bézier奖委员会在其获奖公告中提到了de Casteljau先生的学术贡献。

De Casteljau algorithm



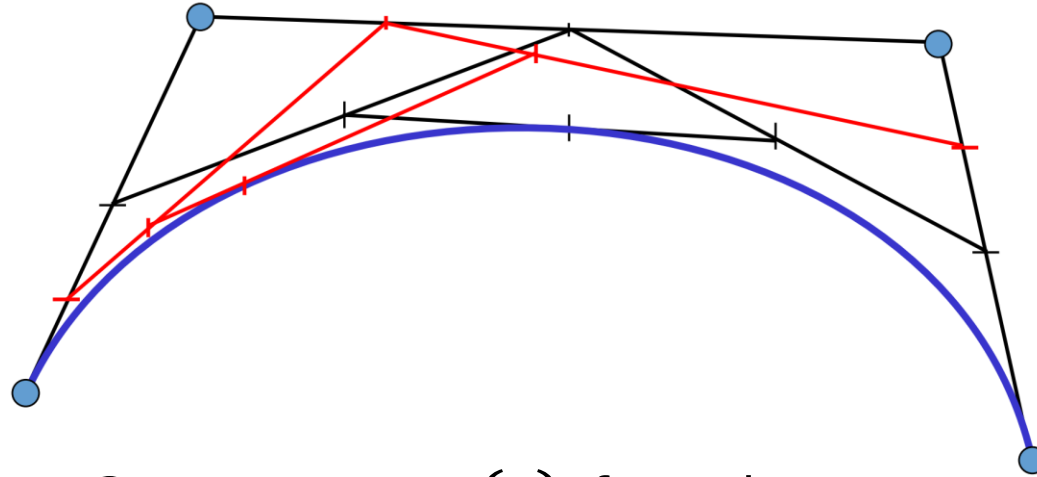
- De Casteljau Algorithm: Computes $x(t)$ for given t
 - Bisect control polygon in ratio $t: (1 - t)$
 - Connect the new dots with lines (adjacent segments)
 - Interpolate again with the same ratio
 - Iterate, until only one points is left

De Casteljau algorithm



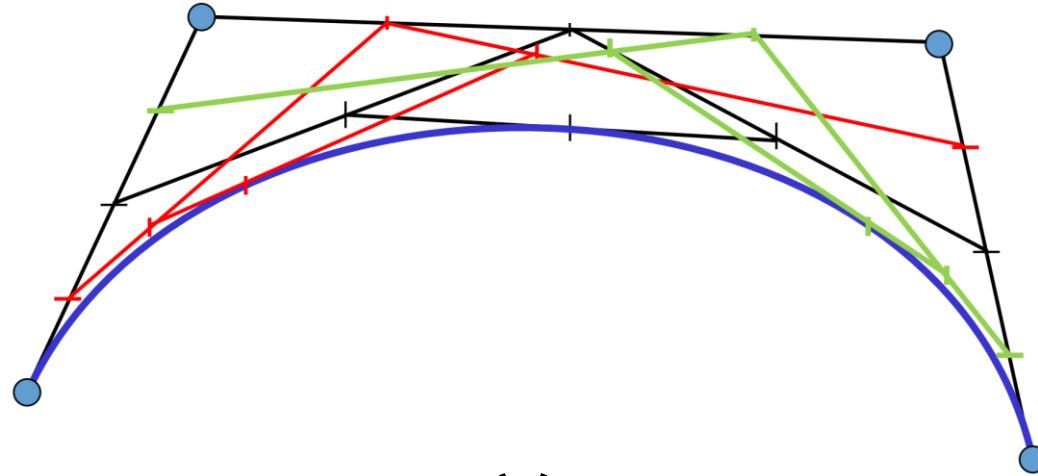
- De Casteljau Algorithm: Computes $x(t)$ for given t
 - Bisect control polygon in ratio $t: (1 - t)$
 - Connect the new dots with lines (adjacent segments)
 - Interpolate again with the same ratio
 - Iterate, until only one points is left

De Casteljau algorithm



- De Casteljau Algorithm: Computes $x(t)$ for given t
 - Bisect control polygon in ratio $t: (1 - t)$
 - Connect the new dots with lines (adjacent segments)
 - Interpolate again with the same ratio
 - Iterate, until only one point is left

De Casteljau algorithm



- De Casteljau Algorithm: Computes $x(t)$ for given t
 - Bisect control polygon in ratio $t: (1 - t)$
 - Connect the new dots with lines (adjacent segments)
 - Interpolate again with the same ratio
 - Iterate, until only one point is left

De Casteljau algorithm

- Algorithm description

- Input: points $\mathbf{b}_0, \mathbf{b}_1, \dots, \mathbf{b}_n \in \mathbb{R}^3$
- Output: curve $\mathbf{x}(t), t \in [0,1]$

- Geometric construction of the points $\mathbf{x}(t)$ for given t :

$$\mathbf{b}_i^0(t) = \mathbf{b}_i, \quad i = 0, \dots, n$$

$$\mathbf{b}_i^r(t) = (1 - t)\mathbf{b}_i^{r-1}(t) + t \mathbf{b}_{i+1}^{r-1}(t)$$

$$r = 1, \dots, n \quad i = 0, \dots, n - r$$

- Then $\mathbf{b}_0^n(t)$ is the searched curve point $\mathbf{x}(t)$ at the parameter value t

De Casteljau algorithm

- Repeated convex combination of control points

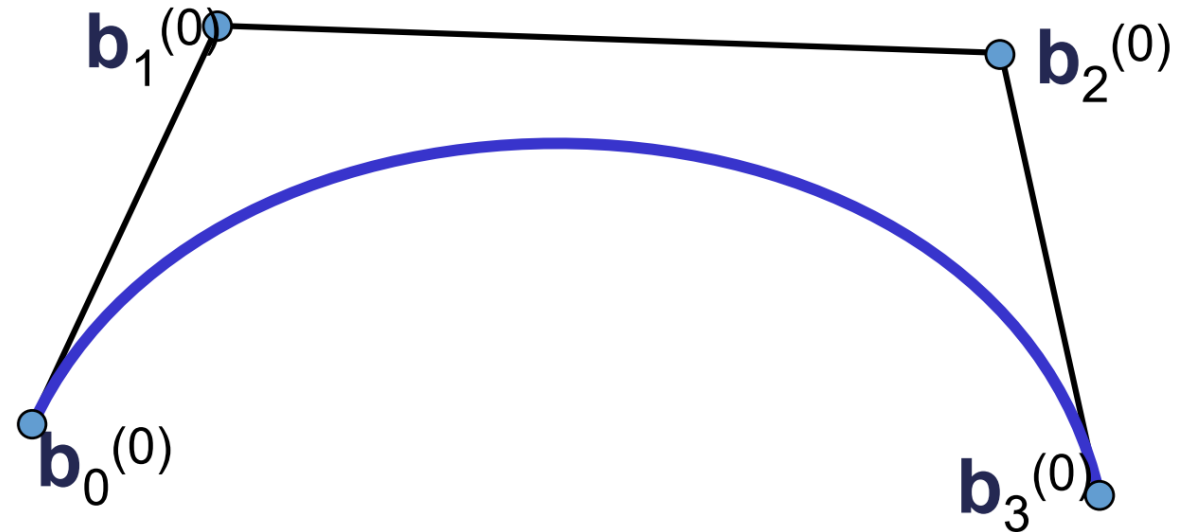
$$\mathbf{b}_i^{(r)} = (1 - t)\mathbf{b}_i^{(r-1)} + t\mathbf{b}_{i+1}^{(r-1)}$$

$\mathbf{b}_0^{(0)}$

$\mathbf{b}_1^{(0)}$

$\mathbf{b}_2^{(0)}$

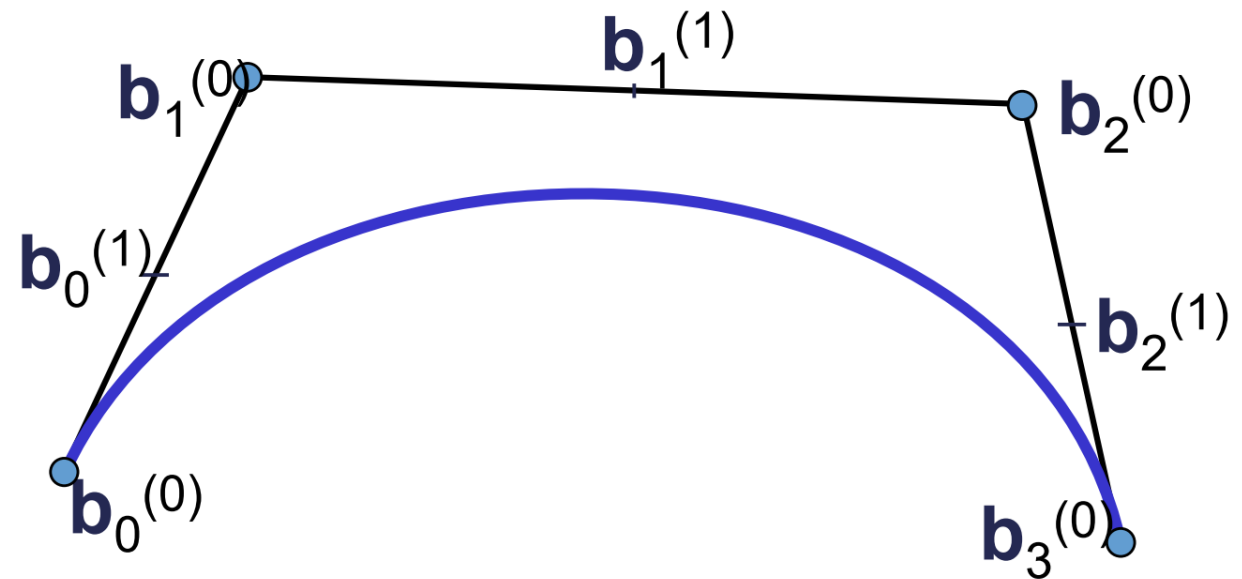
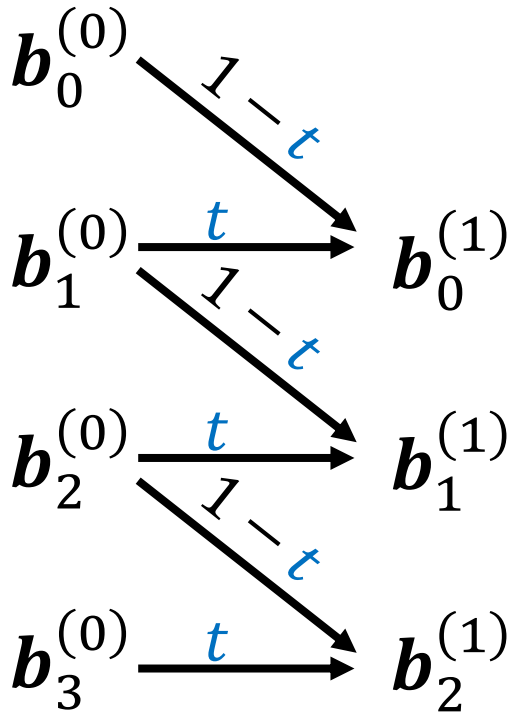
$\mathbf{b}_3^{(0)}$



De Casteljau algorithm

- Repeated convex combination of control points

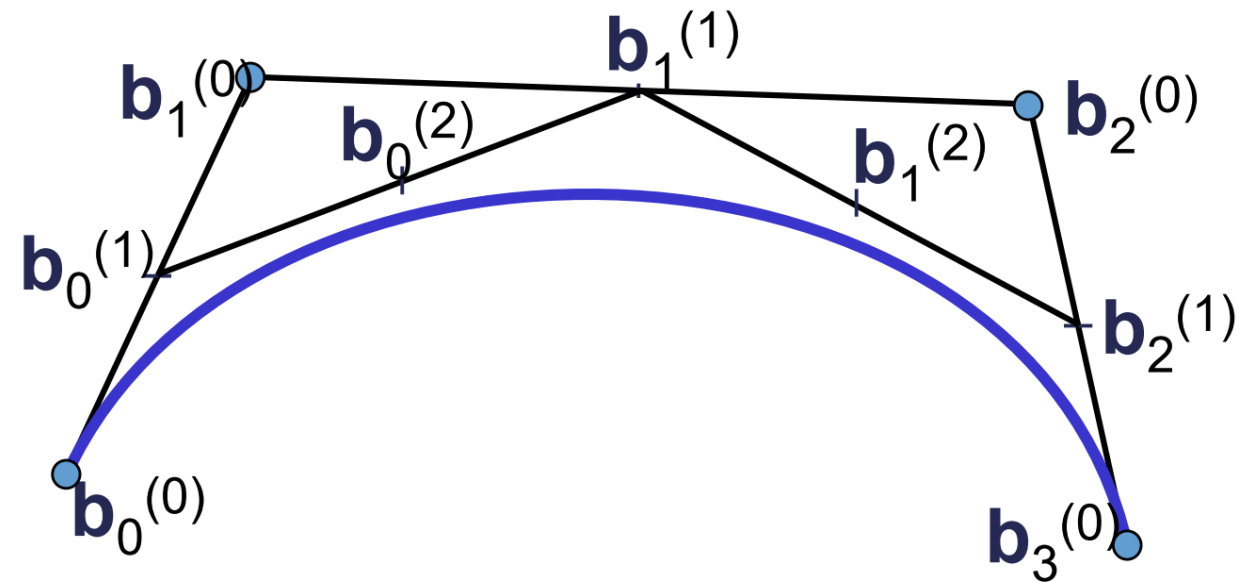
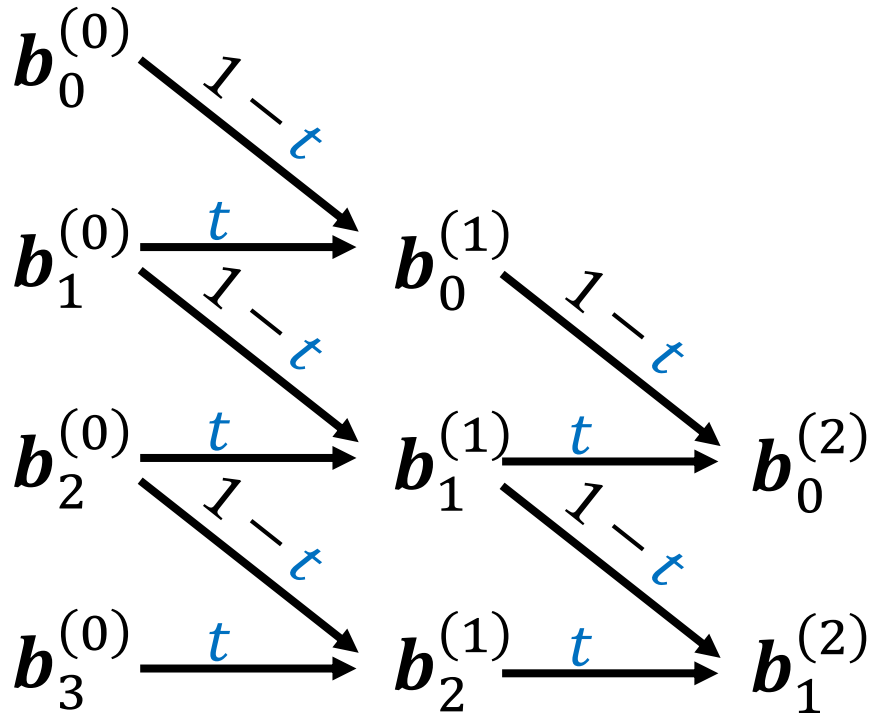
$$\mathbf{b}_i^{(r)} = (1 - t)\mathbf{b}_i^{(r-1)} + t\mathbf{b}_{i+1}^{(r-1)}$$



De Casteljau algorithm

- Repeated convex combination of control points

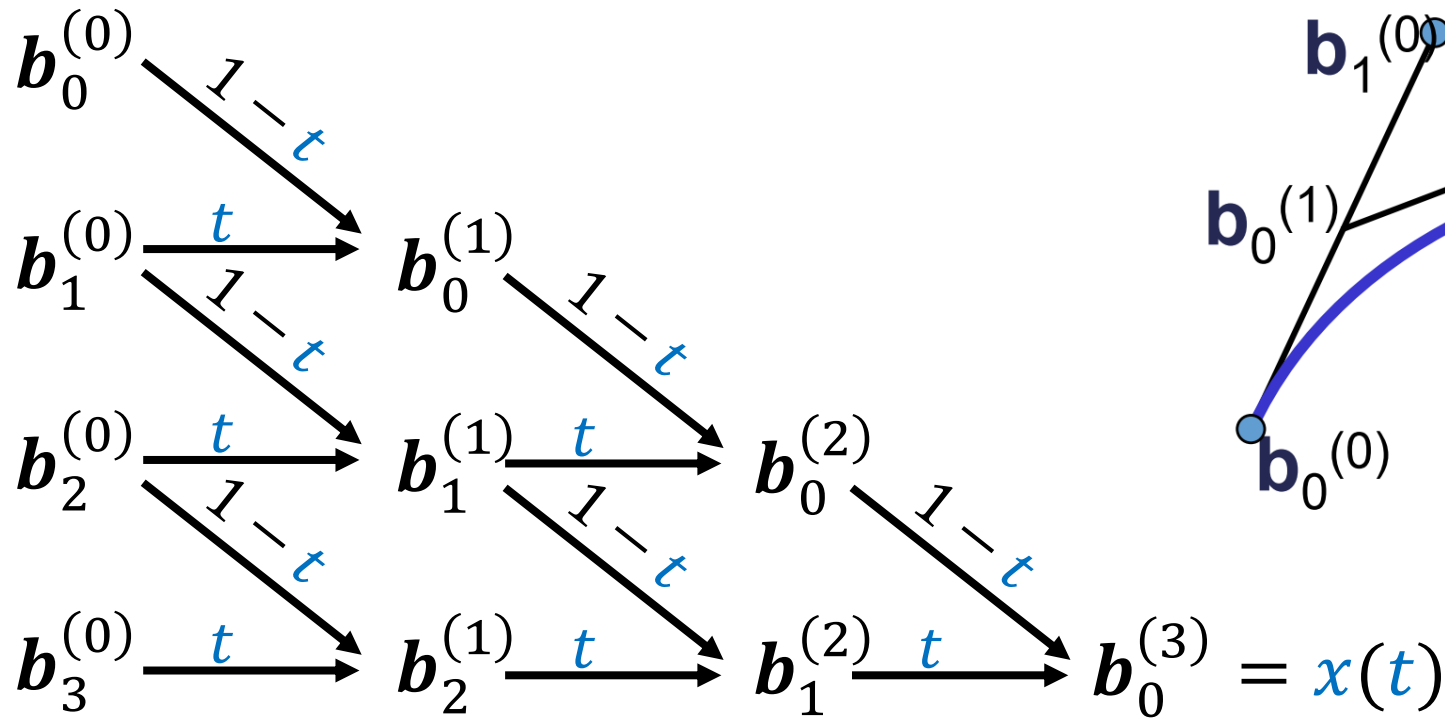
$$\mathbf{b}_i^{(r)} = (1 - t)\mathbf{b}_i^{(r-1)} + t\mathbf{b}_{i+1}^{(r-1)}$$



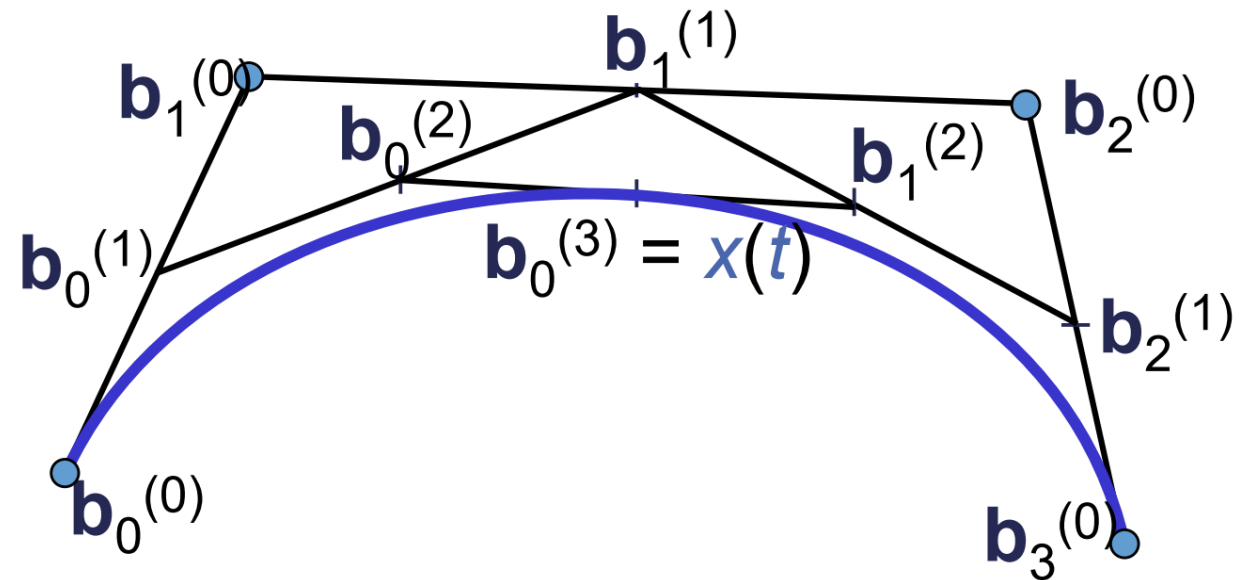
De Casteljau algorithm

- Repeated convex combination of control points

$$\mathbf{b}_i^{(r)} = (1 - t)\mathbf{b}_i^{(r-1)} + t\mathbf{b}_{i+1}^{(r-1)}$$



De Casteljau scheme



De Casteljau algorithm

- The intermediate coefficients $\mathbf{b}_i^r(t)$ can be written in a triangular matrix: the de Casteljau scheme:

$$\mathbf{b}_0 = \mathbf{b}_0^0$$

$$\mathbf{b}_1 = \mathbf{b}_1^0 \quad \mathbf{b}_0^1$$

$$\mathbf{b}_2 = \mathbf{b}_2^0 \quad \mathbf{b}_1^1 \quad \mathbf{b}_0^2$$

$$\mathbf{b}_3 = \mathbf{b}_3^0 \quad \mathbf{b}_2^1 \quad \mathbf{b}_1^2 \quad \mathbf{b}_0^3$$

.....

$$\mathbf{b}_{n-1} = \mathbf{b}_{n-1}^0 \quad \mathbf{b}_{n-2}^1 \quad \dots \quad \mathbf{b}_0^{n-1}$$

$$\mathbf{b}_n = \mathbf{b}_n^0 \quad \mathbf{b}_{n-1}^1 \quad \dots \quad \mathbf{b}_1^{n-1} \quad \mathbf{b}_0^n = x(t)$$

De Casteljau algorithm

Algorithm:

for $r=1..n$

for $i=0..n-r$

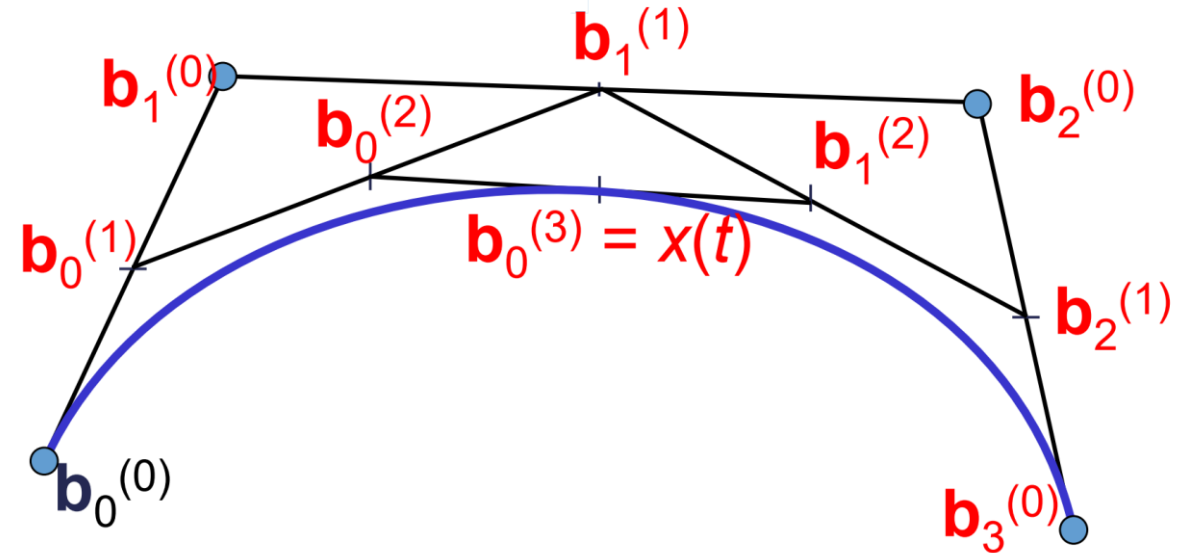
$$\mathbf{b}_i^{(r)} = (1 - t) \mathbf{b}_i^{(r-1)} + t \mathbf{b}_{i+1}^{(r-1)}$$

end

end

return $\mathbf{b}_0^{(n)}$

The whole algorithm consists only of repeated linear interpolations.



De Casteljau algorithm: Properties

- The polygon consisting of the points $\mathbf{b}_0, \dots, \mathbf{b}_n$ is called **Bézier polygon** (control polygon)
- The points \mathbf{b}_i are called **Bézier points** (control points)
- The curve defined by the Bézier points $\mathbf{b}_0, \dots, \mathbf{b}_n$ and the de Casteljau algorithm is called **Bézier curve**
- The de Casteljau algorithm is numerically stable, since only convex combinations are applied.
- Complexity of the de Casteljau algorithm
 - $O(n^2)$ time
 - $O(n)$ memory
 - with n being the number of Bézier points

De Casteljau algorithm: Properties

- **Properties of Bézier curves:**

- Given: Bézier points $\mathbf{b}_0, \dots, \mathbf{b}_n$

Bézier curve $\mathbf{x}(t)$

- Bézier curve is polynomial curve of degree n
 - End points interpolation: $\mathbf{x}(0) = \mathbf{b}_0$, $\mathbf{x}(1) = \mathbf{b}_n$. The remaining Bézier points are only approximated in general

- **Convex hull property:**

Bézier curve is completely inside the convex hull of its Bézier polygon

De Casteljau algorithm: Properties

- **Variation diminishing**
 - No line intersects the Bézier curve more often than its Bézier polygon
- **Influence of Bézier points:** global but pseudo-local
 - Global: moving a Bézier points changes the whole curve progression
 - Pseudo-local: \mathbf{b}_i has its maximal influence on $x(t)$ at $t = \frac{i}{n}$
- **Affine invariance:**
 - Bézier curve and Bézier polygon are invariant under affine transformations
- **Invariance under affine parameter transformations**

De Casteljau algorithm: Properties

- **Symmetry**

- The following two Bézier curves coincide, they are only traversed in opposite directions:

$$\mathbf{x}(t) = [\mathbf{b}_0, \dots, \mathbf{b}_n] \quad \mathbf{x}'(t) = [\mathbf{b}_n, \dots, \mathbf{b}_0]$$

- **Linear Precision:**

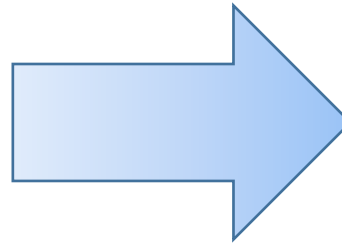
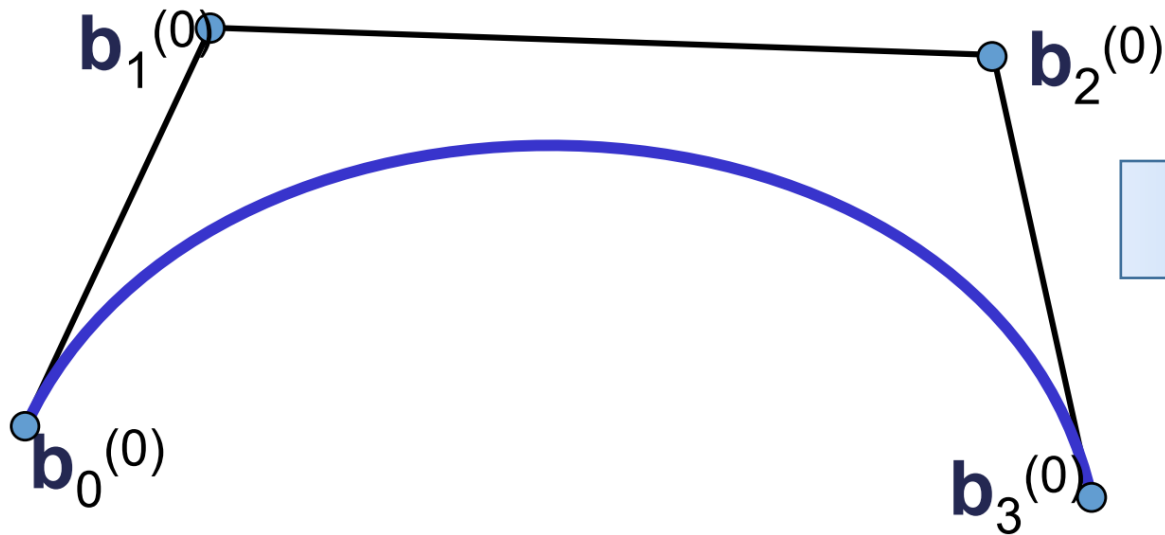
- Bézier curve is line segment, if $\mathbf{b}_0, \dots, \mathbf{b}_n$ are colinear
- Invariance under barycentric combinations

Bézier Curves

Towards a polynomial description

Bézier Curves

Towards a polynomial description



$$x(t) = \sum_{i=0}^n B_i^n(t) \cdot b_i$$

Polynomial description of Bézier curves

- The same problem as before:
 - Given: $(n + 1)$ control points $\mathbf{b}_0, \dots, \mathbf{b}_n$
 - Wanted: Bézier curve $\mathbf{x}(t)$ with $t \in [0,1]$
- Now with an algebraic approach using basis functions

Desirable Properties

- Useful requirements for a basis:
 - Well behaved curve
 - Smooth basis functions

Desirable Properties

- Useful requirements for a basis:
 - Well behaved curve
 - Smooth basis functions
 - Local control (or at least semi-local)
 - Basis functions with compact support

Desirable Properties

- Useful requirements for a basis:
 - Well behaved curve
 - Smooth basis functions
 - Local control (or at least semi-local)
 - Basis functions with compact support
 - Affine invariance:
 - Applying an affine map $\mathbf{x} \rightarrow A\mathbf{x} + \mathbf{b}$ on
 - Control points
 - Curve
- Should have the same effect**
- In particular: rotation, translation
 - Otherwise: interactive curve editing very difficult

Desirable Properties

- Useful requirements for a basis:
 - **Convex hull property:**
 - The curve lays within the convex hull of its control points
 - Avoids at least too weird oscillations
- Advantages
 - Computational advantages (recursive intersection tests)
 - More predictable behavior

Summary

- Useful properties
 - Smoothness
 - Local control / support
 - **Affine invariance**
 - **Convex hull property**

Notations

Curve basis function control points

$$f(t) = \sum_{i=1}^n b_i(t) \mathbf{p}_i$$

Affine Invariance

- Affine map: $\mathbf{x} \rightarrow A\mathbf{x} + \mathbf{b}$
- **Part I:** Linear invariance – we get this automatically
 - Linear approach: $\mathbf{f}(t) = \sum_{i=1}^n b_i(t)\mathbf{p}_i = \sum_{i=1}^n b_i(t) \begin{pmatrix} p_i^{(x)} \\ p_i^{(y)} \\ p_i^{(z)} \end{pmatrix}$
 - Therefore: $A(\mathbf{f}(t)) = A(\sum_{i=1}^n b_i(t)\mathbf{p}_i) = \sum_{i=1}^n b_i(t)(A\mathbf{p}_i)$

Affine Invariance

- Affine Invariance:
 - Affine map: $\mathbf{x} \rightarrow A\mathbf{x} + \mathbf{b}$
 - **Part II:** Translational invariance

$$\sum_{i=1}^n b_i(t)(\mathbf{p}_i + \mathbf{b}) = \sum_{i=1}^n b_i(t)\mathbf{p}_i + \sum_{i=1}^n b_i(t)\mathbf{b} = \mathbf{f}(t) + \left(\sum_{i=1}^n b_i(t) \right) \mathbf{b}$$

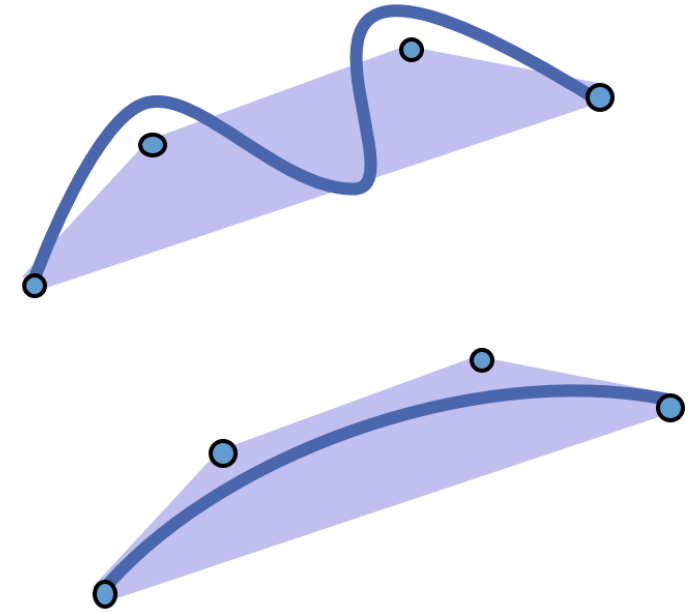
- For translational invariance, the sum of the basis functions must be one *everywhere* (for all parameter values t that are used).
- This is called “**partition of unity** property”
- The b_i ’s form an “**affine combination**” of the control points \mathbf{p}_i
- This is very important for modeling

Convex Hull Property

- Convex combinations:
 - A convex combination of a set of points $\{\mathbf{p}_1, \dots, \mathbf{p}_n\}$ is any point of the form:
$$\sum_{i=1}^n \lambda_i \mathbf{p}_i \text{ with } \sum_{i=1}^n \lambda_i = 1 \text{ and } \forall i = 1 \dots n: 0 \leq \lambda_i \leq 1$$
 - (Remark: $\lambda_i \leq 1$ is redundant)
 - The set of all admissible convex combinations forms the convex hull of the point set
 - Easy to see (exercise): The convex hull is the smallest set that contains all points $\{\mathbf{p}_1, \dots, \mathbf{p}_n\}$ and every complete straight line between two elements of the set

Convex Hull Property

- Accordingly:
 - If we have this property
$$\forall t \in \Omega: \sum_{i=1}^n b_i(t) = 1 \text{ and } \forall t \in \Omega, \forall i: b_i(t) \geq 0$$
the constructed curves / surfaces will be:
 - Affine invariant (translations, linear maps)
 - Be restricted to the convex hull of the control points
 - Corollary: Curves will have *linear precision*
 - All control points lie on a straight line
 \Rightarrow Curve is a straight line segment
 - Surfaces with planar control points will be flat, too



Convex Hull Property

- Very useful property in practice
 - Avoids at least the worst oscillations
 - no escape from convex hull, unlike polynomial interpolation
 - Linear precision property is intuitive (people expect this)
 - Can be used for fast range checks
 - Test for intersection with convex hull first, then the object
 - Recursive intersection algorithms in conjunctions with subdivision rules (more on this later)



Polynomial description of Bézier curves

- The same problem as before:
 - Given: $(n + 1)$ control points $\mathbf{b}_0, \dots, \mathbf{b}_n$
 - Wanted: Bézier curve $\mathbf{x}(t)$ with $t \in [0,1]$
- Now with an algebraic approach using basis functions
- Need to define $n + 1$ basis functions
 - Such that this describes a Bézier curve:

$$\mathbf{x}(t) = \sum_{i=0}^n B_i^n(t) \cdot \mathbf{b}_i$$

Bernstein Basis

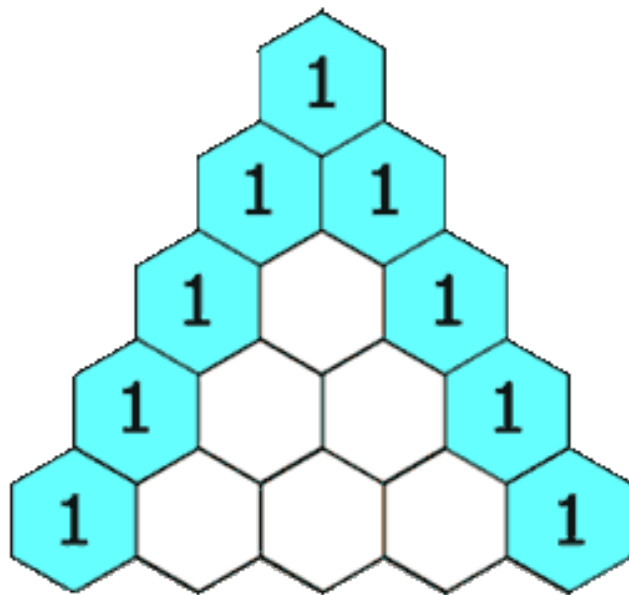
- Let's examine the Bernstein basis: $B = \{B_0^{(n)}, B_1^{(n)}, \dots, B_n^{(n)}\}$
 - Bernstein basis of degree n :

$$B_i^{(n)}(t) = \binom{n}{i} t^i (1-t)^{n-i} = B_{i\text{-th basis function}}^{(\text{degree})}$$

where the binomial coefficients are given by:

$$\binom{n}{i} = \begin{cases} \frac{n!}{(n-i)! i!} & \text{for } 0 \leq i \leq n \\ 0 & \text{otherwise} \end{cases}$$

Binomial Coefficients and Theorem



$$\binom{n}{i} + \binom{n}{i+1} = \binom{n+1}{i+1}$$

					1					
				1		1				
		1		2		1				
	1		3		3		1			
	1	4		6		4		1		
1	5	10		10		5		1		

$$(x + y)^n = \sum_{i=0}^n \binom{n}{i} x^i y^{n-i}$$

Examples: The first few

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

- The first three Bernstein bases:

$$B_0^{(0)} := 1$$

$$B_0^{(1)} := 1 - t \quad B_1^{(1)} := t$$

$$B_0^{(2)} := (1 - t)^2 \quad B_1^{(2)} := 2t(1 - t) \quad B_2^{(2)} := t^2$$

$$B_0^{(3)} := (1 - t)^3 \quad B_1^{(3)} := 3t(1 - t)^2 \quad B_2^{(3)} := 3t^2(1 - t) \quad B_3^{(3)} := t^3$$

Examples: The first few

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

$$B_0^{(0)} := 1$$

$$B_0^{(1)} := 1 - t$$

$$B_1^{(1)} := t$$

$$B_0^{(2)} := (1-t)^2$$

$$B_1^{(2)} := 2t(1-t)$$

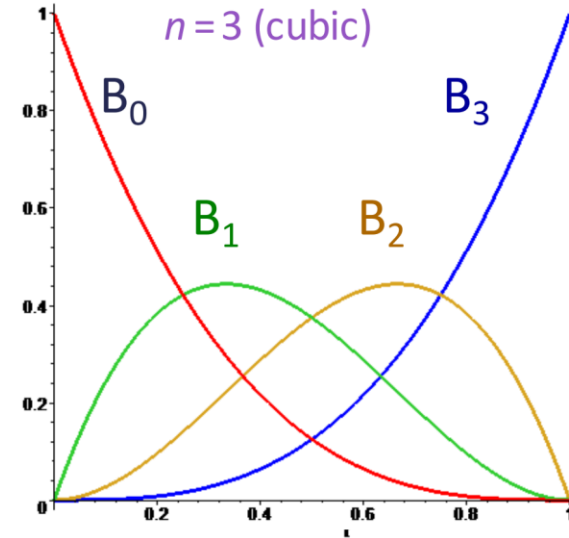
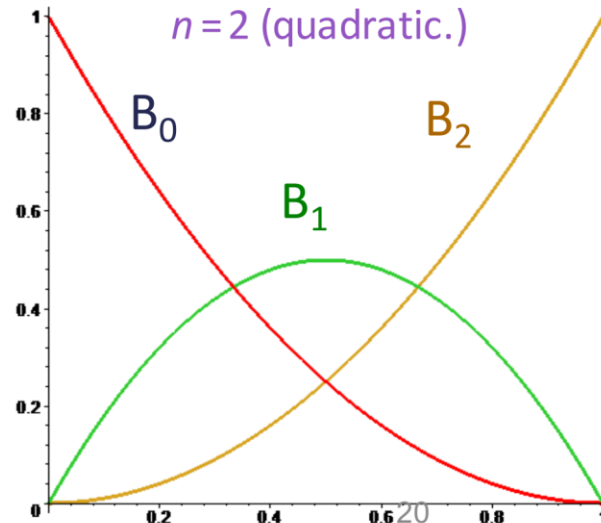
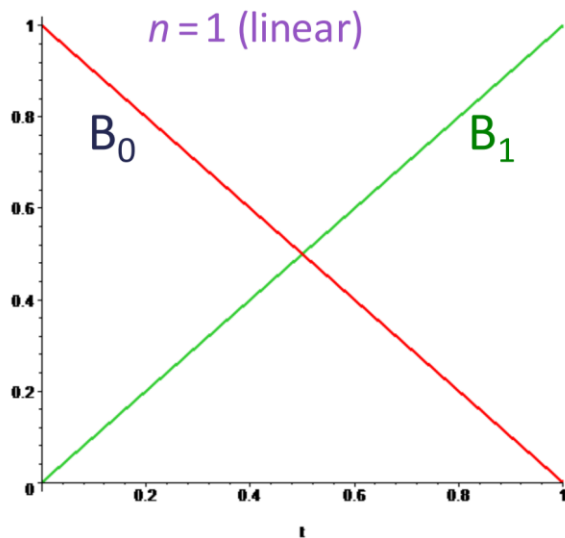
$$B_2^{(2)} := t^2$$

$$B_0^{(3)} := (1-t)^3$$

$$B_1^{(3)} := 3t(1-t)^2$$

$$B_2^{(3)} := 3t^2(1-t)$$

$$B_3^{(3)} := t^3$$

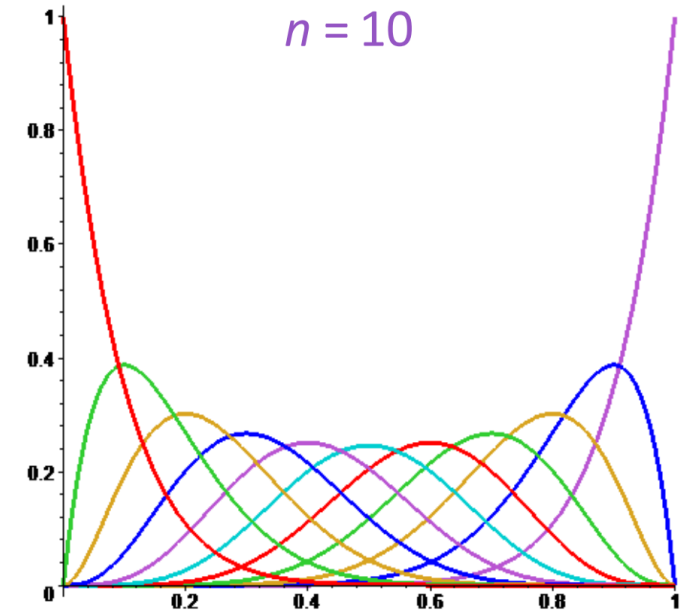
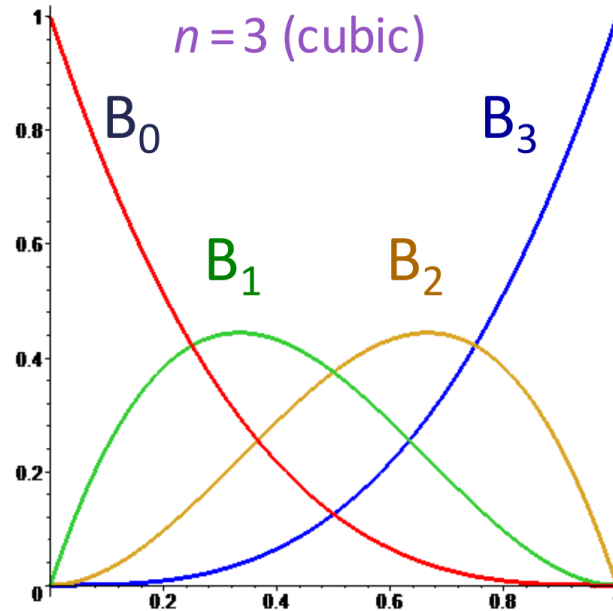
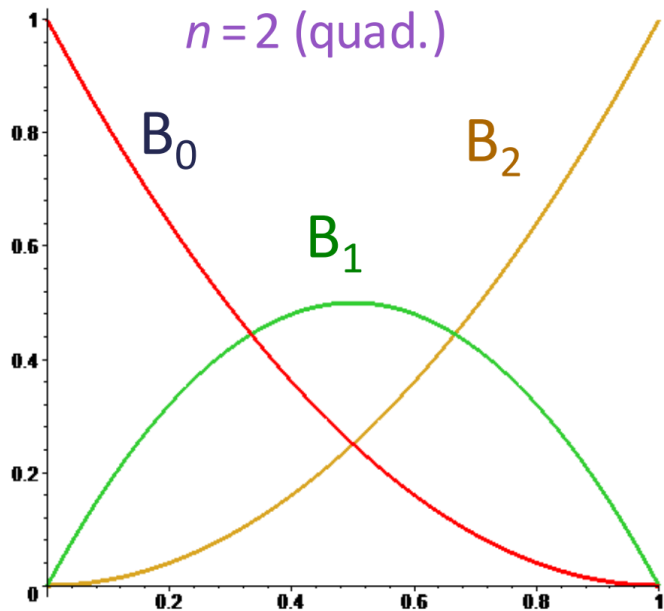


Bernstein Basis

- Bézier curves use the Bernstein basis: $B = \{B_0^{(n)}, B_1^{(n)}, \dots, B_n^{(n)}\}$

- Bernstein basis of degree n :

$$B_i^{(n)}(t) = \binom{n}{i} t^i (1-t)^{n-i} = B_{i\text{-th basis function}}^{(\text{degree})}$$



Bernstein Basis

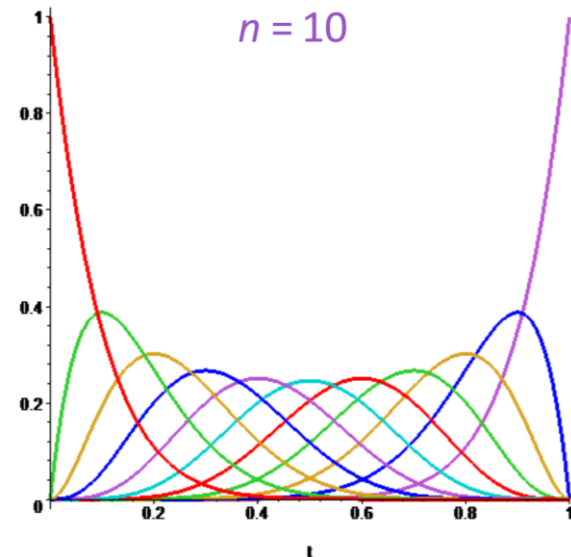
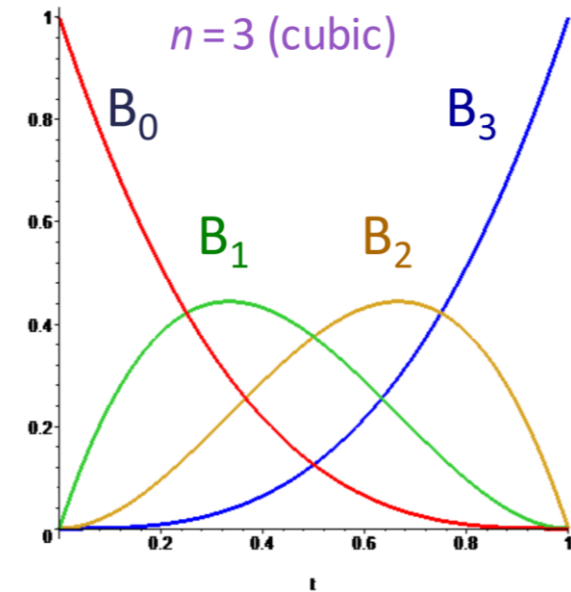
- What about the desired properties?
 - Smoothness
 - Local control / support
 - Affine invariance
 - Convex hull property

Bernstein Basis: Properties

- $B = \{B_0^{(n)}, B_1^{(n)}, \dots, B_n^{(n)}\}$, $B_i^{(n)}(t) = \binom{n}{i} t^i (1-t)^{n-i}$
- Basis for polynomials of degree n
- Each basis function $B_i^{(n)}$ has its maximum at $t = \frac{i}{n}$

Smoothness

Local control (semi-local)



Bernstein Basis: Properties

- $B = \{B_0^{(n)}, B_1^{(n)}, \dots, B_n^{(n)}\}$, $B_i^{(n)}(t) = \binom{n}{i} t^i (1-t)^{n-i}$

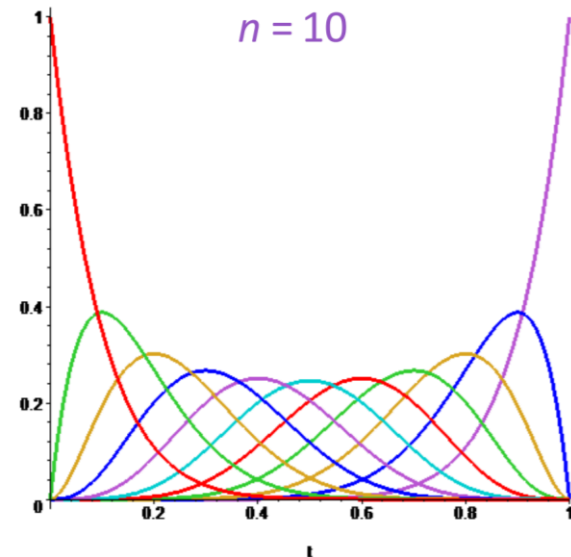
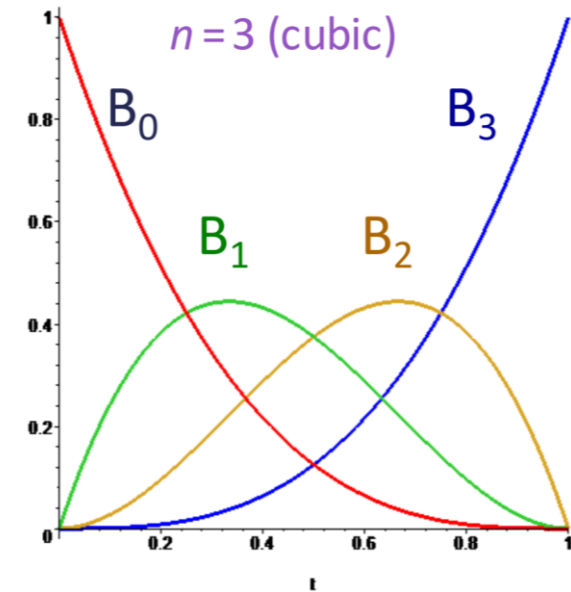
Affine invariance

Convex hull property

- Partition of unity (binomial theorem)

$$1 = (1 - t + t)$$

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



What about the desired properties?

- Smoothness Yes
- Local control / support To some extent
- Affine invariance Yes
- Convex hull property Yes

$$\binom{n-1}{i} + \binom{n-1}{i-1} = \binom{n}{i}$$

Bernstein Basis: Properties

- $B = \{B_0^{(n)}, B_1^{(n)}, \dots, B_n^{(n)}\}$, $B_i^{(n)}(t) = \binom{n}{i} t^i (1-t)^{n-i}$

- Recursive computation

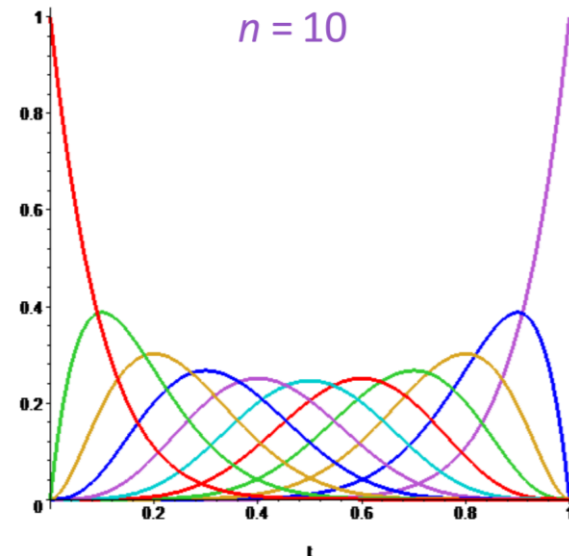
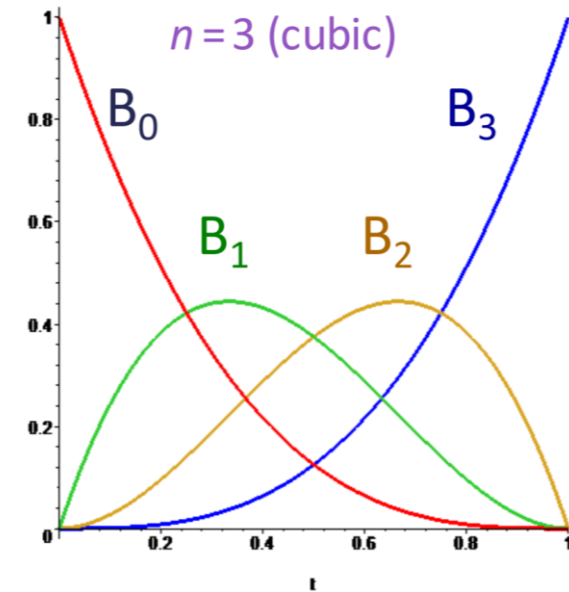
$$B_i^n(t) := (1-t)B_i^{(n-1)}(t) + tB_{i-1}^{(n-1)}(1-t)$$

with $B_0^0(t) = 1$, $B_i^n(t) = 0$ for $i \notin \{0 \dots n\}$

- Symmetry

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

- Non-negativity: $B_i^{(n)}(t) \geq 0$ for $t \in [0..1]$



Bernstein Basis: Properties

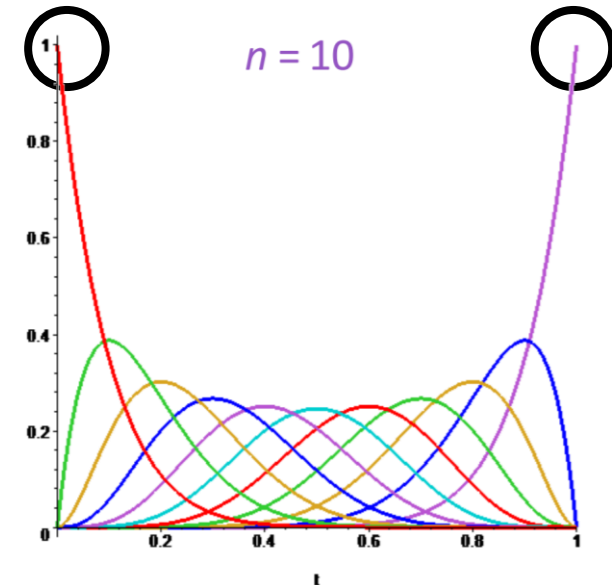
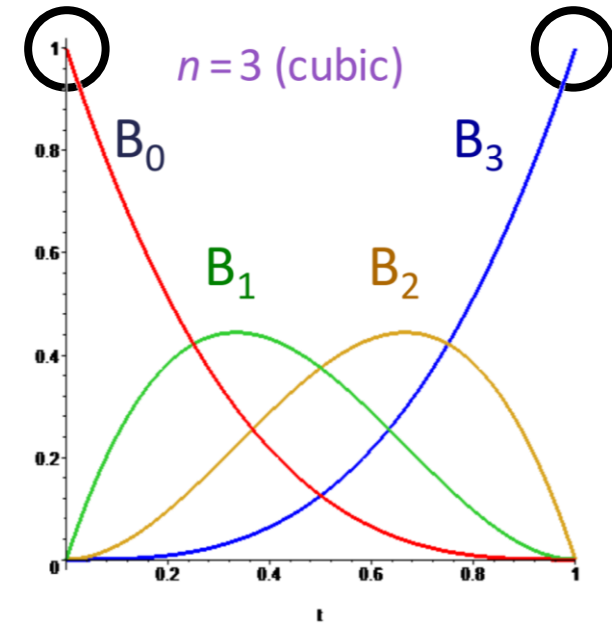
- $B = \{B_0^{(n)}, B_1^{(n)}, \dots, B_n^{(n)}\}$, $B_i^{(n)}(t) = \binom{n}{i} t^i (1-t)^{n-i}$

- Non-negativity II

$$B_i^n(t) > 0 \text{ for } 0 < t < 1$$

$$B_0^n(0) = 1, \quad B_1^n(0) = \dots = B_n^n(0) = 0$$

$$B_0^n(1) = \dots = B_{n-1}^n(1) = 0, \quad B_n^n(1) = 1$$



Computer Aided Geometric Design

Fall Semester 2024

Bézier Curves (continue)

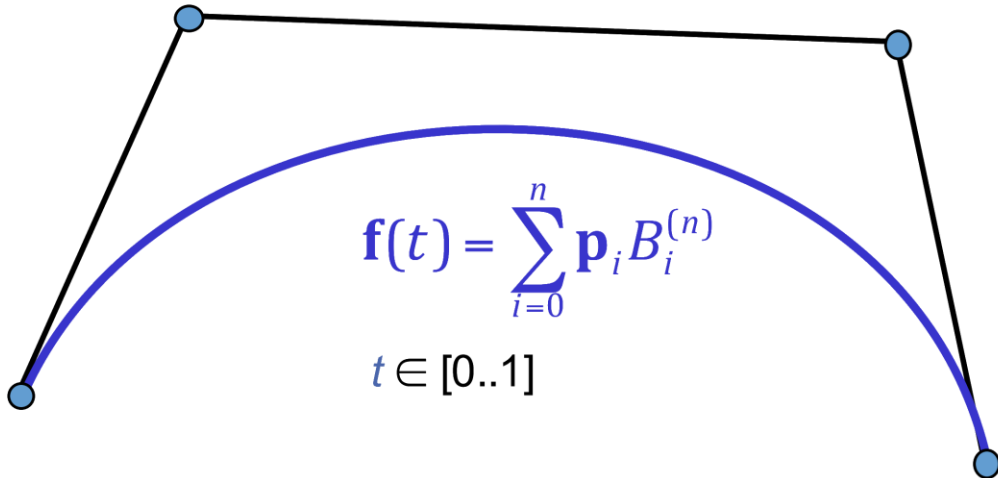
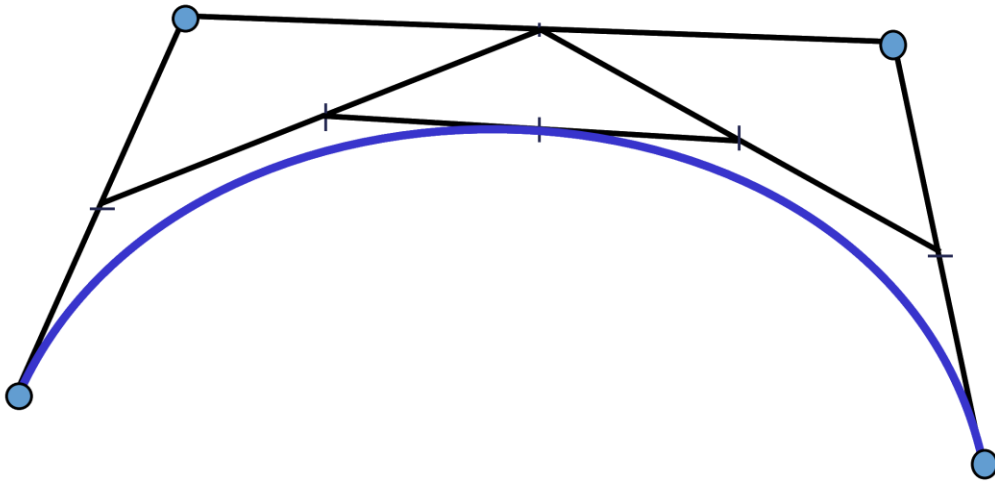
陈仁杰

renjiec@ustc.edu.cn

<http://staff.ustc.edu.cn/~renjiec>

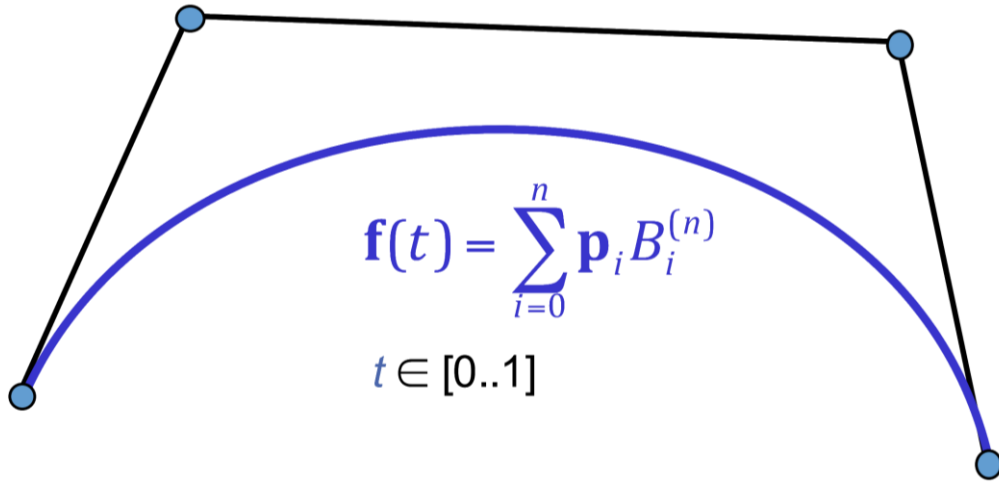
Recap

de Casteljau algorithm



Bernstein form

Recap



Bernstein form

Curve basis function control points

$\mathbf{f}(t) = \sum_{i=1}^n B_i(t) \mathbf{p}_i$

Dashed arrows point from the labels to the corresponding parts of the equation: 'Curve' to $\mathbf{f}(t)$, 'basis function' to $B_i(t)$, and 'control points' to \mathbf{p}_i .

Useful properties for basis functions

- Smoothness
- Local control / support
- Affine invariance
- Convex hull property

Degree elevation

- Given: $b_0, \dots, b_n \rightarrow x(t)$
- Wanted: $\bar{b}_0, \dots, \bar{b}_n, \bar{b}_{n+1} \rightarrow \bar{x}(t)$ with $x = \bar{x}$
- Solution:

Degree elevation

- Given: $\mathbf{b}_0, \dots, \mathbf{b}_n \rightarrow \mathbf{x}(t)$
- Wanted: $\bar{\mathbf{b}}_0, \dots, \bar{\mathbf{b}}_n, \bar{\mathbf{b}}_{n+1} \rightarrow \bar{\mathbf{x}}(t)$ with $\mathbf{x} = \bar{\mathbf{x}}$
- Solution:

$$\bar{\mathbf{b}}_0 = \mathbf{b}_0$$

$$\bar{\mathbf{b}}_{n+1} = \mathbf{b}_n$$

$$\bar{\mathbf{b}}_j = \frac{j}{n+1} \mathbf{b}_{j-1} + \left(1 - \frac{j}{n+1}\right) \mathbf{b}_j \quad \text{for } j = 1, \dots, n$$

Proof

- Let's consider

$$\begin{aligned}(1-t)B_i^n(t) &= (1-t) \binom{n}{i} (1-t)^{n-i} t^i = \binom{n}{i} (1-t)^{n+1-i} t^i \\&= \frac{n+1-i}{n+1} \binom{n+1}{i} (1-t)^{n+1-i} t^i \\&= \frac{n+1-i}{n+1} B_i^{n+1}(t)\end{aligned}$$

Similarly

$$tB_i^n(t) = \frac{i+1}{n+1} B_{i+1}^{n+1}(t)$$

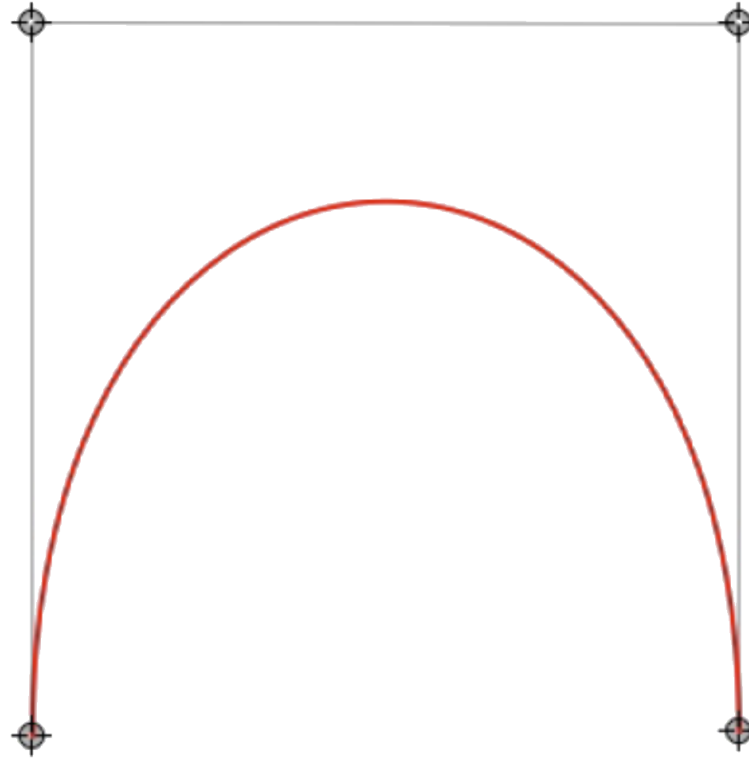
Proof

Using results from
last slide

$$\begin{aligned}
 f(t) &= [(1-t) + t]f(t) = [(1-t) + t] \sum_{i=0}^n B_i^n(t) \mathbf{P}_i = \sum_{i=0}^n [(1-t)B_i^n(t) + tB_i^n(t)] \mathbf{P}_i \\
 &= \sum_{i=0}^n \left[\frac{n+1-i}{n+1} B_i^{n+1}(t) + \frac{i+1}{n+1} B_{i+1}^{n+1}(t) \right] \mathbf{P}_i = \sum_{i=0}^n \frac{n+1-i}{n+1} B_i^{n+1}(t) \mathbf{P}_i + \sum_{i=0}^n \frac{i+1}{n+1} B_{i+1}^{n+1}(t) \mathbf{P}_i \\
 &= \sum_{i=0}^n \frac{n+1-i}{n+1} B_i^{n+1}(t) \mathbf{P}_i + \sum_{i=1}^{n+1} \frac{i}{n+1} B_i^{n+1}(t) \mathbf{P}_{i-1} \\
 &= \sum_{i=0}^{n+1} \frac{n+1-i}{n+1} B_i^{n+1}(t) \mathbf{P}_i + \sum_{i=0}^{n+1} \frac{i}{n+1} B_i^{n+1}(t) \mathbf{P}_{i-1} \\
 &= \sum_{i=0}^{n+1} B_i^{n+1}(t) \left[\frac{n+1-i}{n+1} \mathbf{P}_i + \frac{i}{n+1} \mathbf{P}_{i-1} \right]
 \end{aligned}$$

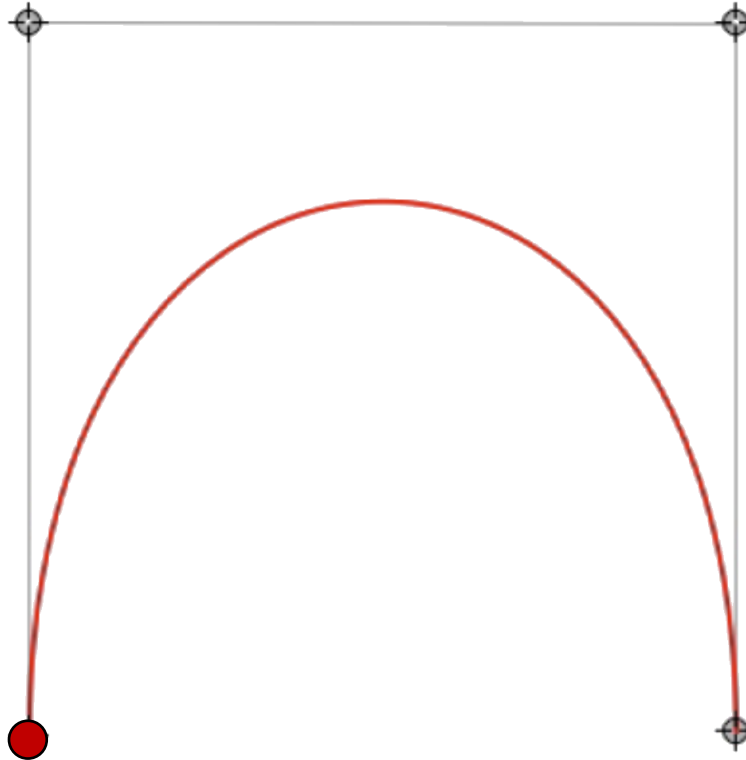
Adding null terms, $i = n+1, i = 0$

Degree elevation: Example



- $\bar{b}_0 = b_0$
 - $\bar{b}_{n+1} = b_n$
- $$\bar{b}_j = \frac{j}{n+1} b_{j-1} + \left(1 - \frac{j}{n+1}\right) b_j$$
- $$j = 1, \dots, n$$

Degree elevation: Example



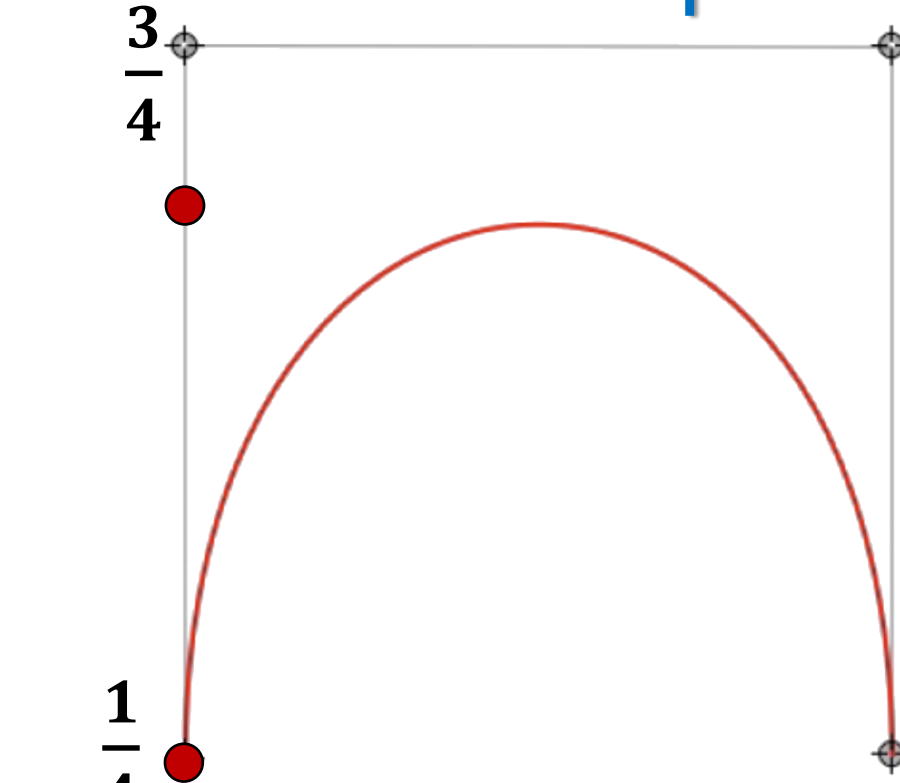
- $\bar{b}_0 = b_0$

$$\bar{b}_j = \frac{j}{n+1} b_{j-1} + \left(1 - \frac{j}{n+1}\right) b_j$$

- $\bar{b}_{n+1} = b_n$

$$j = 1, \dots, n$$

Degree elevation: Example



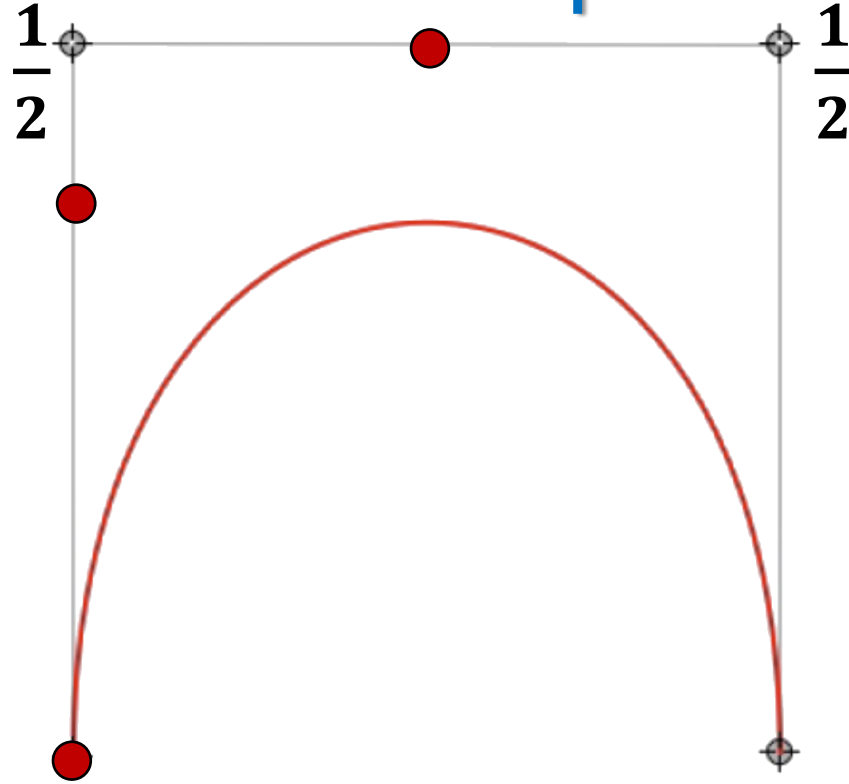
- $\bar{b}_0 = b_0$

- $\bar{b}_{n+1} = b_n$

$$\bar{b}_j = \frac{j}{n+1} b_{j-1} + \left(1 - \frac{j}{n+1}\right) b_j$$

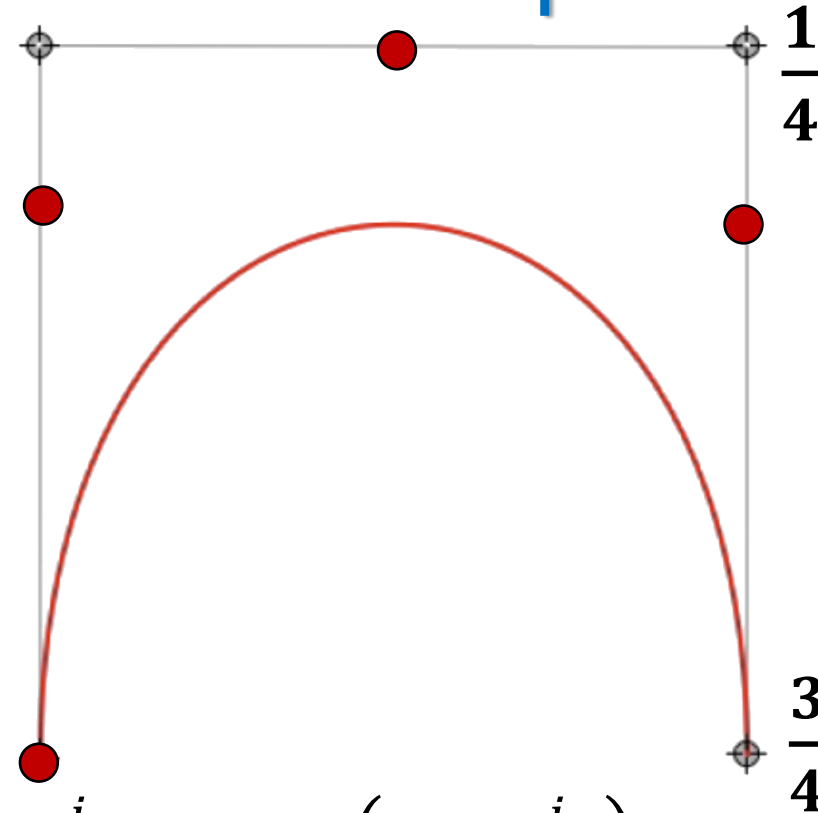
$$j = 1, \dots, n$$

Degree elevation: Example



- $\bar{b}_0 = b_0$
 - $\bar{b}_{n+1} = b_n$
- $$\bar{b}_j = \frac{j}{n+1} b_{j-1} + \left(1 - \frac{j}{n+1}\right) b_j$$
- $$j = 1, \dots, n$$

Degree elevation: Example



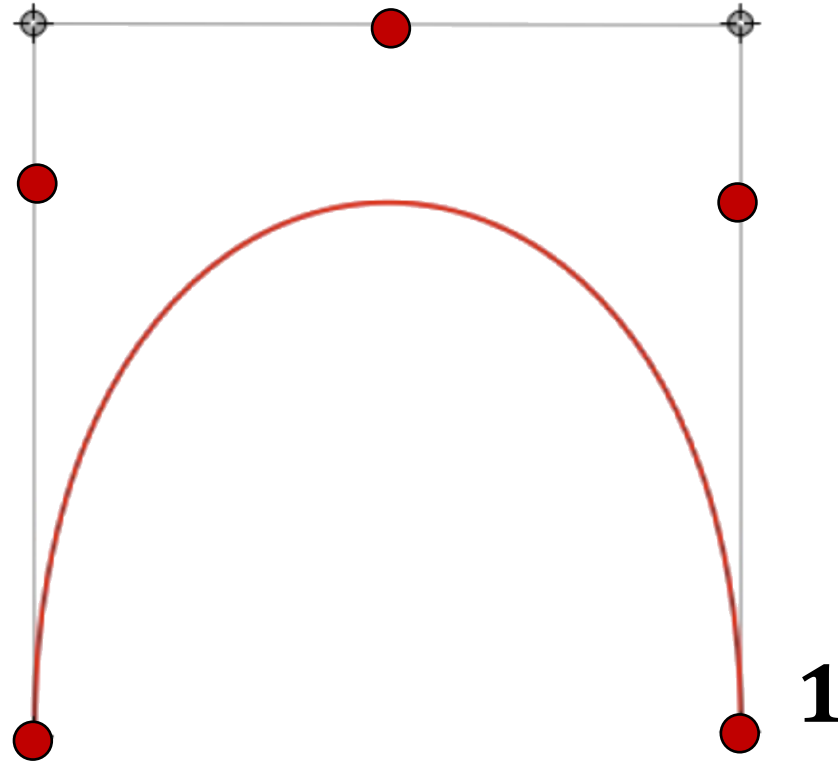
- $\bar{b}_0 = b_0$

- $\bar{b}_{n+1} = b_n$

$$\bar{b}_j = \frac{j}{n+1} b_{j-1} + \left(1 - \frac{j}{n+1}\right) b_j$$

$$j = 1, \dots, n$$

Degree elevation: Example

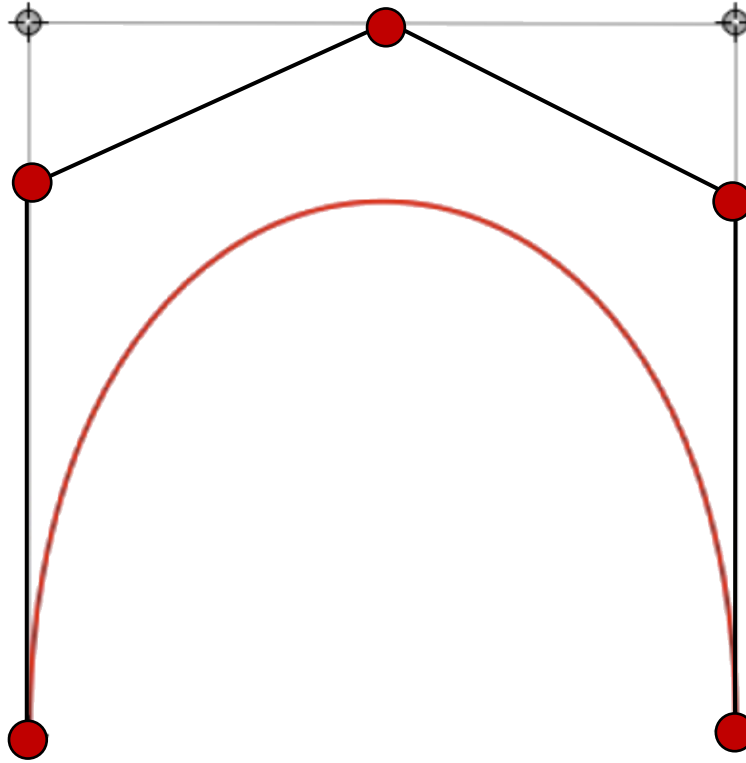


- $\bar{b}_0 = b_0$

- $\bar{b}_{n+1} = b_n$

$$\bar{b}_j = \frac{j}{n+1} b_{j-1} + \left(1 - \frac{j}{n+1}\right) b_j$$
$$j = 1, \dots, n$$

Degree elevation: Example

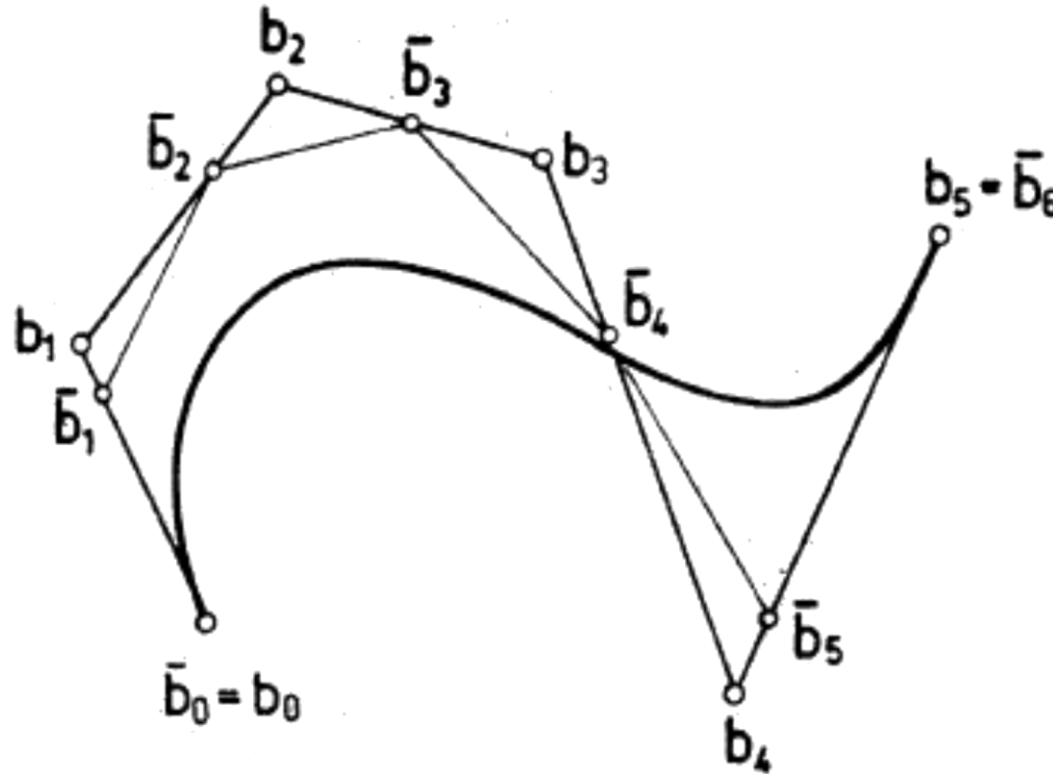


- $\bar{b}_0 = b_0$

- $\bar{b}_{n+1} = b_n$

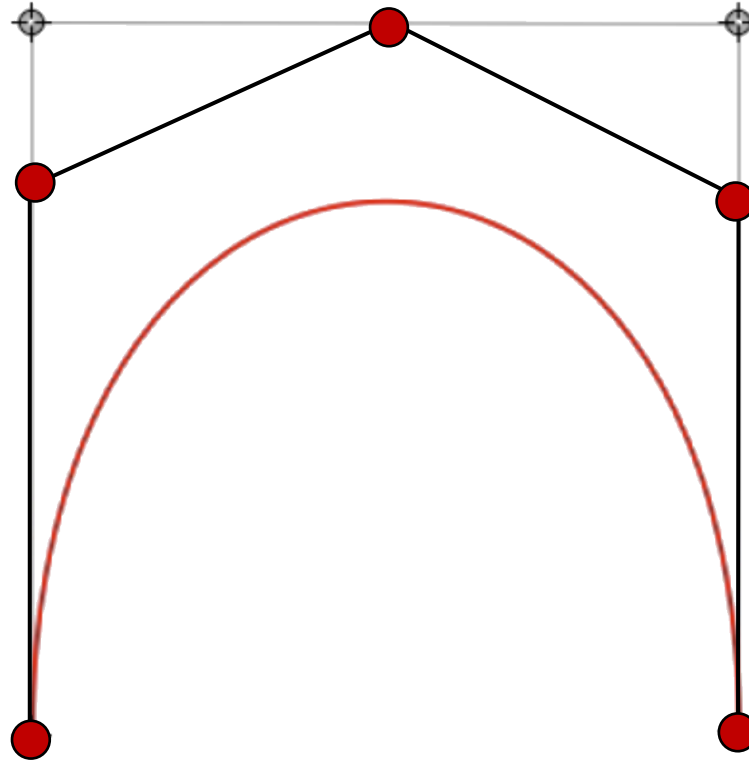
$$\bar{b}_j = \frac{j}{n+1} b_{j-1} + \left(1 - \frac{j}{n+1}\right) b_j$$
$$j = 1, \dots, n$$

Degree elevation



For repeated degree elevation, the Bézier polygon converges to the Bézier curve. (slow convergence)

Degree elevation



- $\bar{b}_0 = b_0$

- $\bar{b}_{n+1} = b_n$

$$\bar{b}_j = \frac{j}{n+1} b_{j-1} + \left(1 - \frac{j}{n+1}\right) b_j$$
$$j = 1, \dots, n$$

Bézier Curves

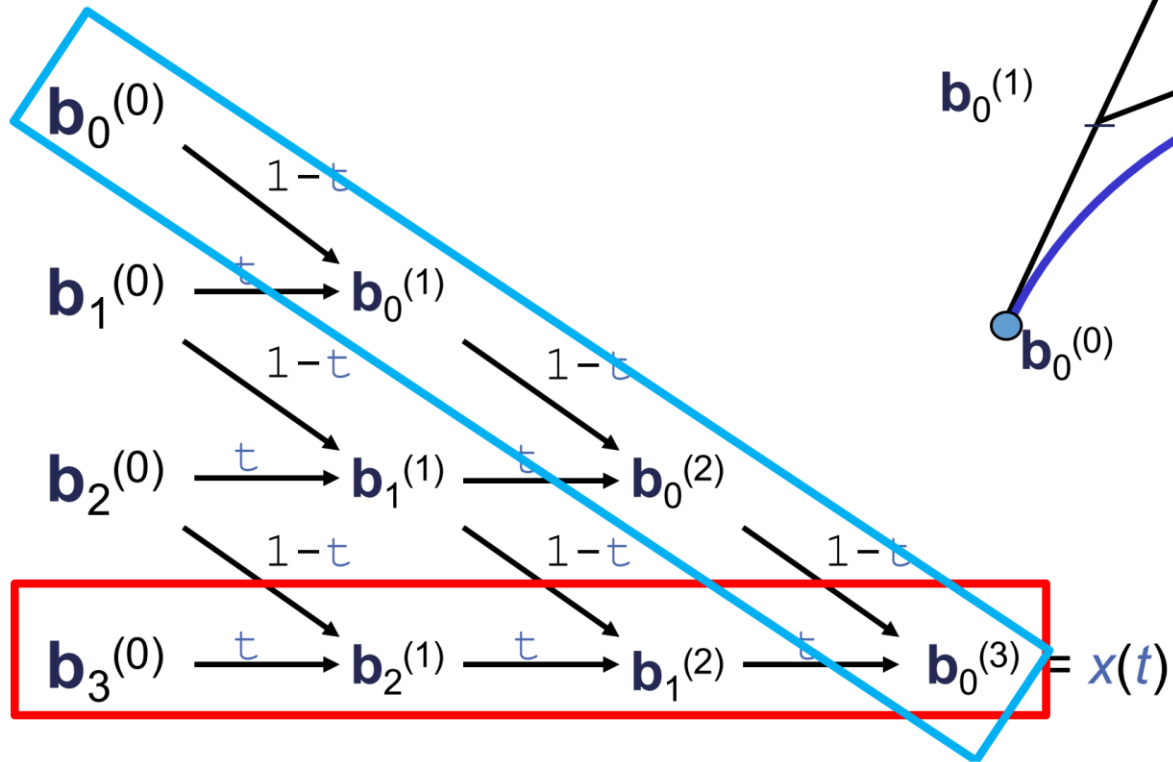
Subdivision

Subdivision

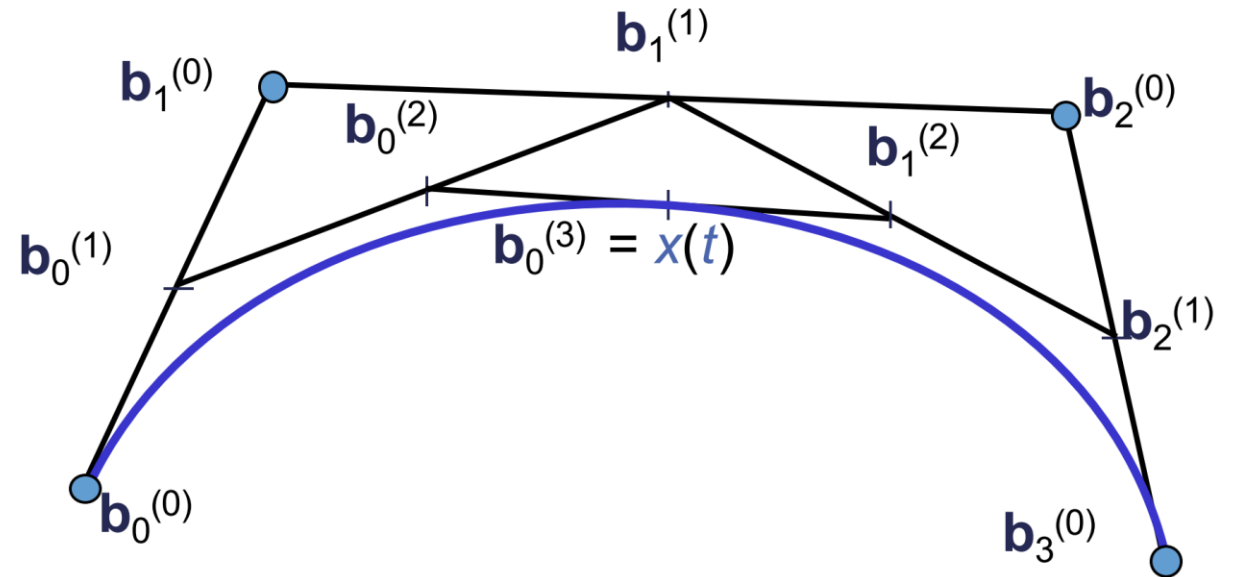
- **Given:** $b_0, \dots, b_n \rightarrow x(t), t \in [0,1]$
- **Wanted:** $b_0^{(1)}, \dots, b_n^{(1)} \rightarrow x^{(1)}(t),$
 $b_0^{(2)}, \dots, b_n^{(2)} \rightarrow x^{(2)}(t),$

with $x = x^{(1)} \cup x^{(2)}$

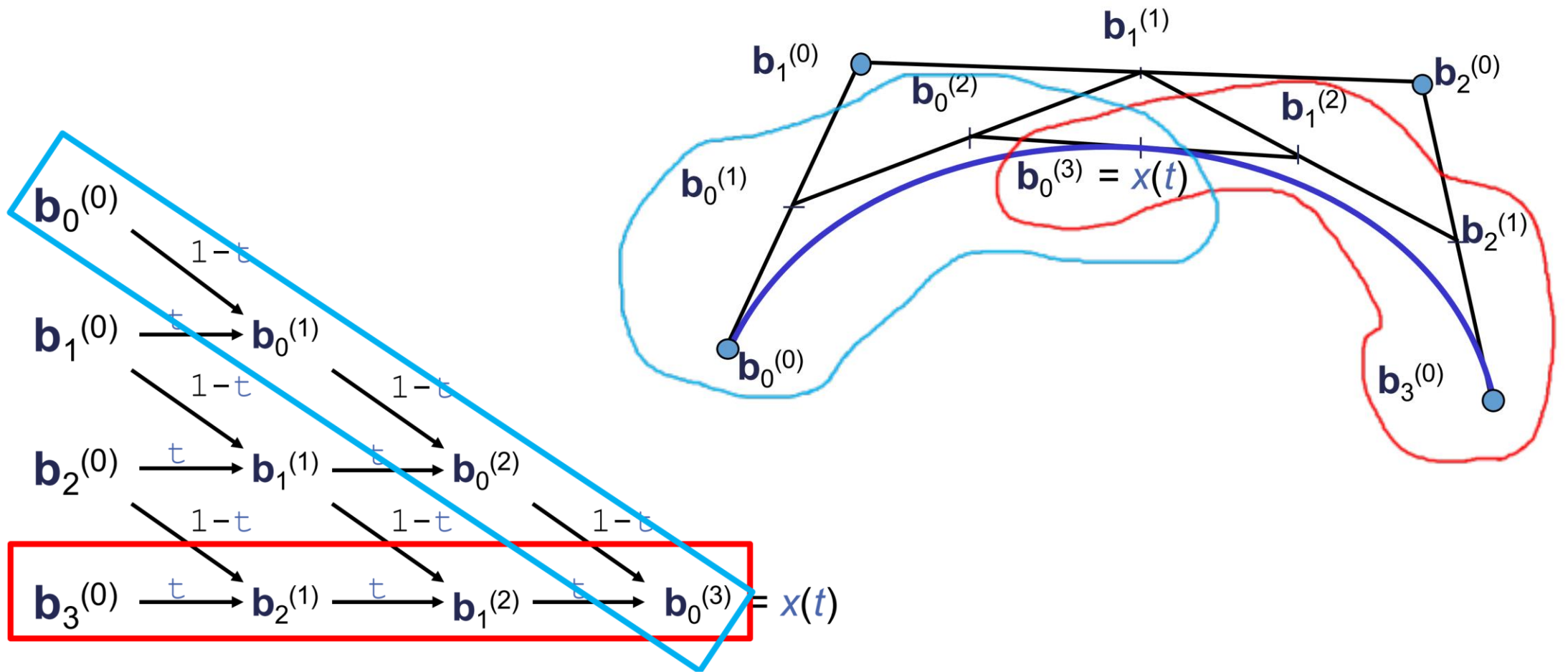
Subdivision: Example



de Casteljau scheme



Subdivision: Example

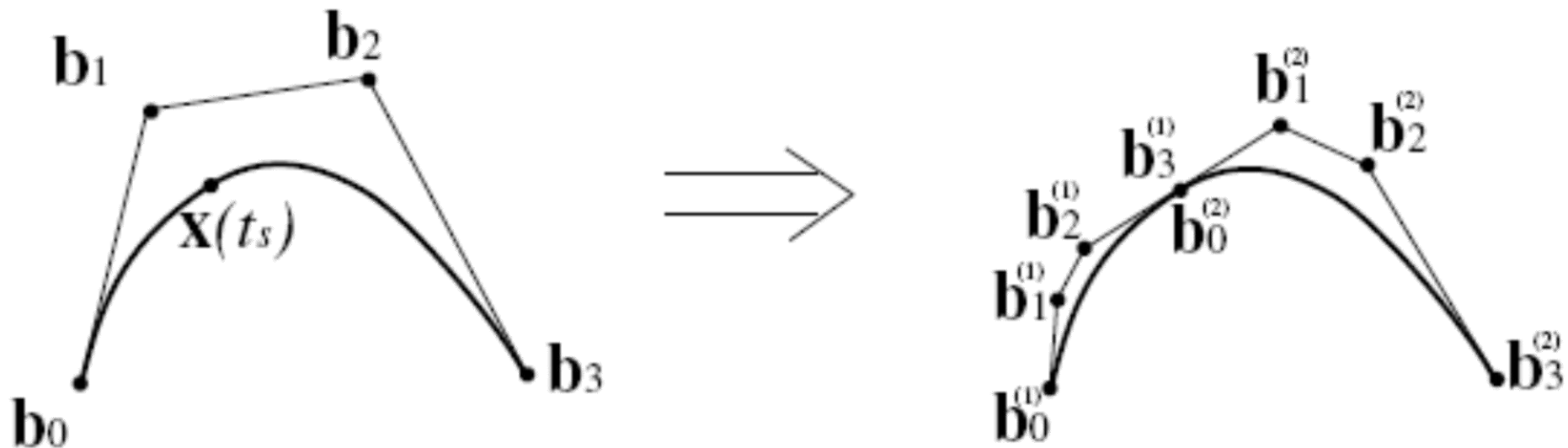


de Casteljau scheme

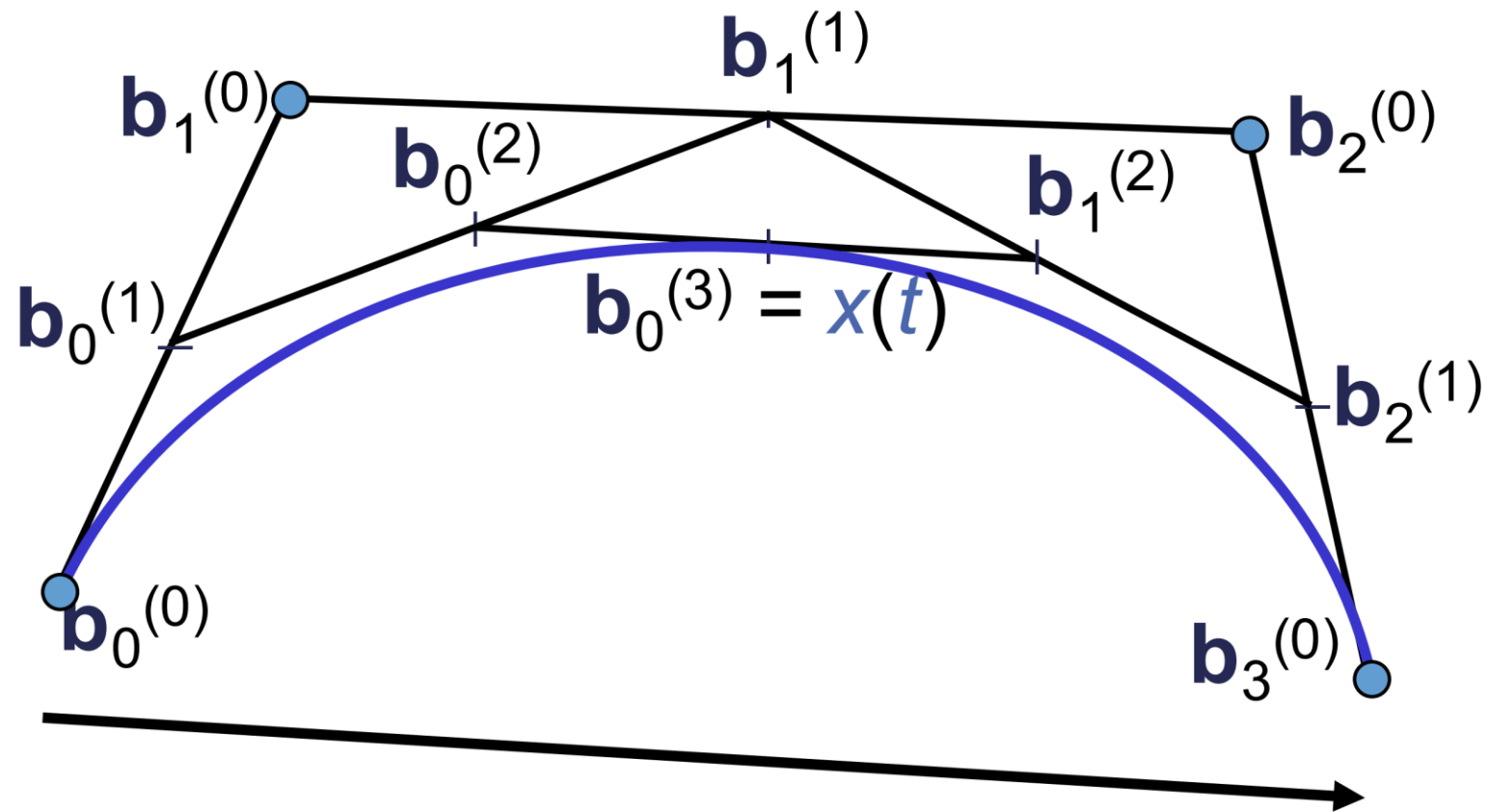
Subdivision

Solution: $b_i^{(1)} = b_0^i$, $b_i^{(2)} = b_0^{n-i}$ for $i = 0, \dots, n$

That means that the new points are intermediate points of the de Casteljau algorithm!

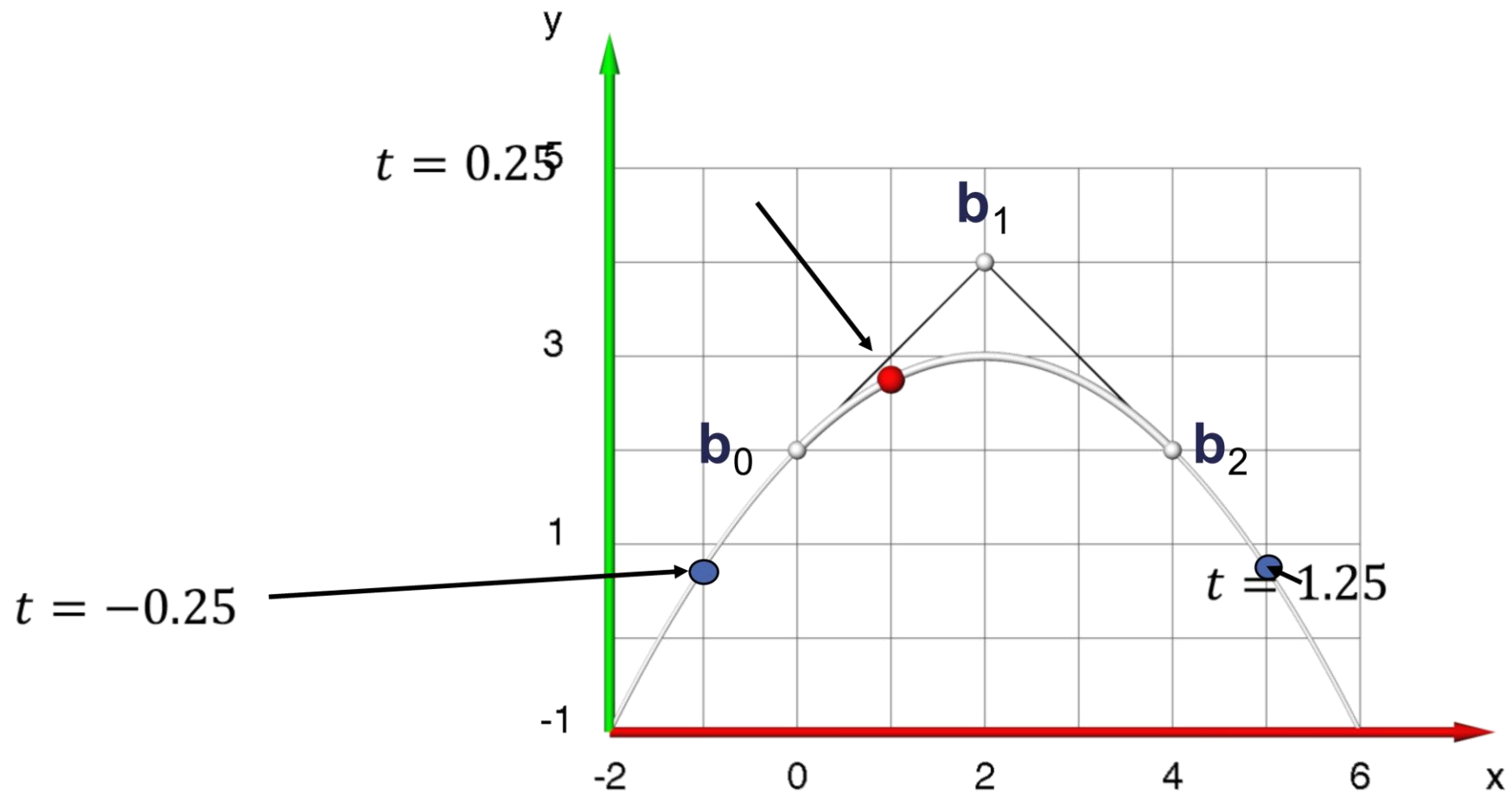


Curve range



parameterization: $t \in [0, 1]$

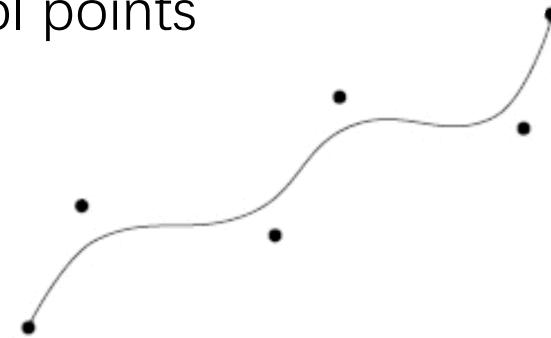
Curve range



Summary & Outlook

- **Bézier curves and curve design**

- The rough form is specified by the position of the control points
- Results: smooth curve approximating the control points
- Computation / Representation:
 - de Casteljau algorithm
 - Bernstein form



- Problems:
 - High polynomial degree
 - Moving a control point can change the whole curve
 - Interpolation of points
 - → **Bézier splines**

Matrix representations
(common in software implementations)

Homogeneous coordinates

$$[P] = \begin{pmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ \dots & \dots & \dots & \dots \\ x_n & y_n & z_n & 1 \end{pmatrix}$$

Transformations

- Basic representation $[P^*] = [P][T]$
 - $[P^*]$ is the new coordinates matrix
 - $[P]$ is the original coordinates matrix, or points matrix
 - $[T]$ is the transformation matrix

Transformations

- Translation (2D example)

$$[T_t] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x & y & 0 & 1 \end{pmatrix}$$

$$[P^*] = [P][T_t]$$

Transformations

- Basic representation $[P^*] = [P][T]$

$[P^*]$ is the new coordinates matrix

$[P]$ is the original coordinates matrix, or points matrix

$[T]$ is the transformation matrix

$$[P] = \begin{pmatrix} x_1 & y_1 & 0 \\ x_2 & y_2 & 0 \\ x_3 & y_3 & 0 \\ \dots & \dots & \dots \\ x_n & y_n & 0 \end{pmatrix}$$

Transformations

- Uniform scaling

$$[T] = \begin{pmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Non-uniform scaling

$$[T] = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformations

- Rotation 2D

$$\left. \begin{aligned} x &= r \cos \alpha \\ y &= r \sin \alpha \end{aligned} \right\} \text{Original coordinates of point } P$$

$$\left. \begin{aligned} x^* &= r \cos(\alpha + \theta) \\ y^* &= r \sin(\alpha + \theta) \end{aligned} \right\} \text{The new coordinates}$$

$$\begin{bmatrix} x^* & y^* & 0 & 1 \end{bmatrix} = \begin{bmatrix} x & y & 0 & 1 \end{bmatrix} \begin{pmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformations

- Rotation about an arbitrary axis (2D)
 - Translate the fixed axis so it coincides with z-axis
→ apply to object
 - Rotate object about the axis
 - Translate object back

$$[P^*] = [P][T_t][T_r][T_{-t}]$$

Transformations

- Rotation about an arbitrary axis (2D)

Step 1: Translate the fixed axis so it coincides with z-axis

Step 2: Rotate object about the axis

Step 3: Translate the fixed axis back to the original position

$$[P^*] = [P][T_t][T_r][T_{-t}]$$

Transformations

- Scaling with an arbitrary point (x, y)

$$[P^*] = [P][T_t][T_s][T_{-t}]$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x & -y & 0 & 1 \end{pmatrix} \begin{pmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x & y & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ x - sx & y - sy & 0 & 1 \end{pmatrix}$$

Transformations

- Rotation about an arbitrary point (x, y)

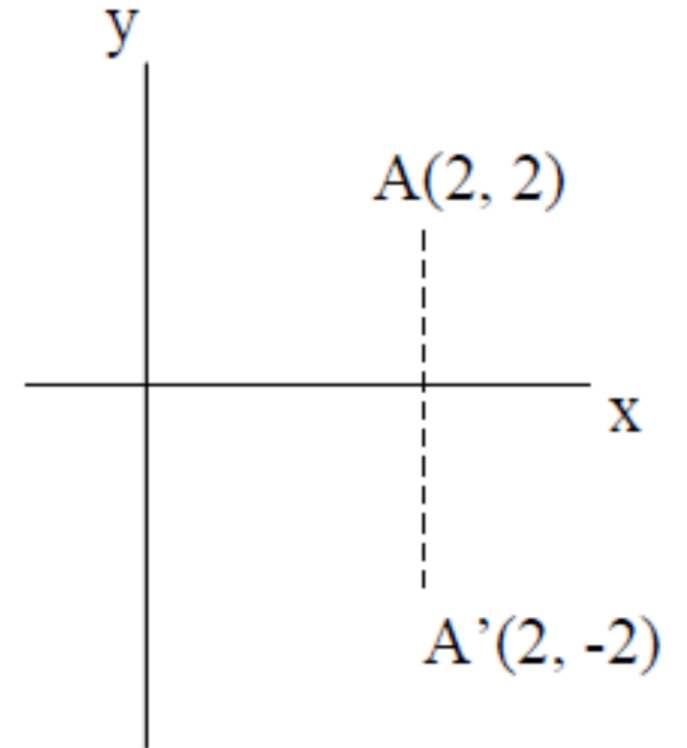
$$[T_{\text{cond}}] = [T_t][T_s][T_{-t}]$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x & -y & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x & y & 0 & 1 \end{pmatrix}$$

Transformations

- Mirroring about x-axis (negative scaling along y-axis)

$$[P^*] = [2 \quad 2 \quad 0 \quad 1] \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$
$$= [2 \quad -2 \quad 0 \quad 1]$$



Transformations

- Mirroring about arbitrary axis
 - Translate line to pass through origin
 - Rotate axis to coincide with x -axis
 - Mirror about x -axis
 - Rotate back
 - Translate back to original position

$$[P^*] = [P][T_t][T_r][T_m][T_{-r}][T_{-t}]$$

Transformations

- Rotation about coordinates axes (3D)

$$[T_{rz}] = \begin{pmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$[T_{rx}] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$[T_{ry}] = \begin{pmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformations

- Rotation θ about an arbitrary axis (3D)
 1. Translate the given line so that it will pass through the origin
 2. Rotate about the x -axis so that the line lies in the xz -plane (angle α)
 3. Rotate about the y -axis so that the line coincides with the z -axis (angle ϕ)
 4. Rotate the geometric object about the z -axis (angle θ – given rotation angle)
 5. Reverse of step 3
 6. Reverse of step 2
 7. Reverse of step 1

$$[P^*] = [P][T_t][T_r]_\alpha[T_r]_\phi[T_r]_\theta[T_r]_{-\phi}[T_r]_{-\alpha}[T_{-t}]$$

Alternatively you can use Quaternions!

Bézier Curves

- Cubic Bézier curves $f(t) = P_0B_0^{(3)} + P_1B_1^{(3)} + P_2B_2^{(3)} + P_3B_3^{(3)}$

$$B_0^{(3)}(t) = \frac{3!}{0!3!}t^0(1-t)^3 = (1-t)^3$$

$$B_1^{(3)} = \frac{3!}{1!2!}t^1(1-t)^2 = 3t(1-t)^2$$

$$B_2^{(3)} = \frac{3!}{2!1!}t^2(1-t)^1 = 3t^2(1-t)$$

$$B_3^{(3)} = \frac{3!}{3!0!}t^3(1-t)^0 = t^3$$

Bézier Curves

- Cubic Bézier curves
- In Matrix form:
 - -The curve
 - -The tangent

$$f(t) = P_0 B_0^{(3)} + P_1 B_1^{(3)} + P_2 B_2^{(3)} + P_3 B_3^{(3)}$$

$$f(t) = [t^3 \quad t^2 \quad t \quad 1] \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

$$f'(t) = [3t^2 \quad 2t \quad 1 \quad 0] \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

B-spline Curves (to be covered later)

- Uniform cubic B-Spline curve

$$f_i(t) = \frac{1}{6} [t^3 \quad t^2 \quad t \quad 1] \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix} \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

B-spline Curves (to be covered later)

- Other splines:

- Catmull-Rom

$$f_i(t) = [t^3 \quad t^2 \quad t \quad 1] \frac{1}{2} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

- Cardinal splines

$$\begin{pmatrix} -a & 2-a & a-2 & a \\ 2a & a-3 & 3-2a & -a \\ -a & 0 & a & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

- Tensioned splines

$$\frac{1}{6} \begin{pmatrix} -a & 12-9a & 9a-12 & a \\ 2a & a-3 & 18-15a & -a \\ -3a & 0 & 3a & 0 \\ 0 & 6-2a & a & 0 \end{pmatrix}$$

Computer Aided Geometric Design

Fall Semester 2024

Differential Geometry of Curves

陈仁杰

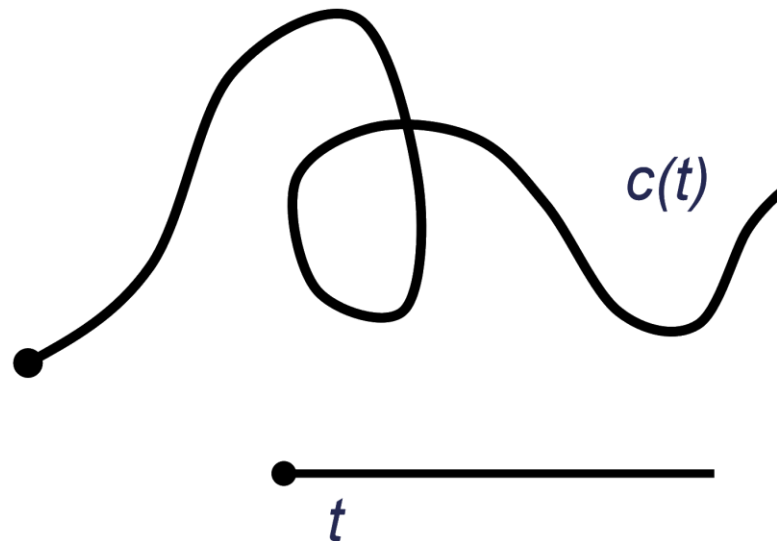
renjiec@ustc.edu.cn

<http://staff.ustc.edu.cn/~renjiec>

Parametric Curves

- **Parametric Curves:**

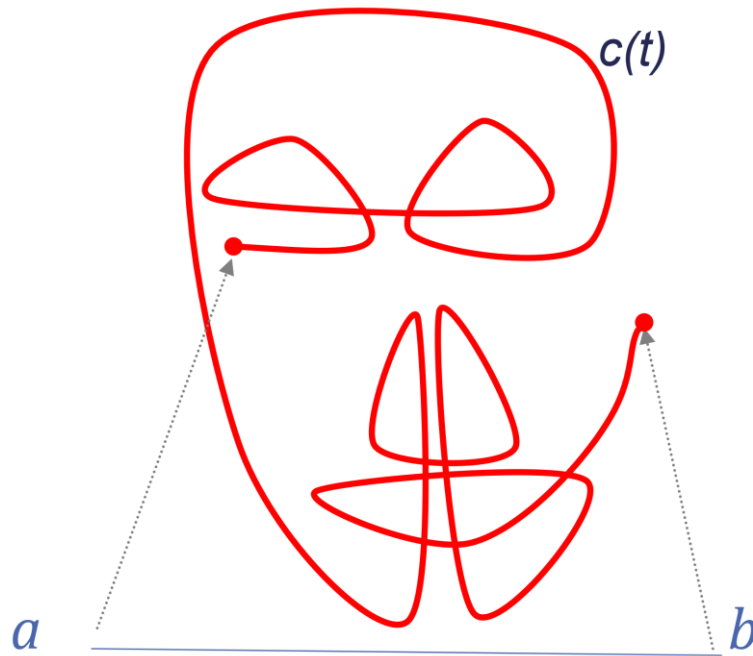
- Think of a curve c as the path of a moving particle
- Not always enough to know **where** a particle went – we also want to know **when** it got there $\rightarrow c(t)$
- Parameter t is often thought of as time



Parametric Curves

- **Parametric Curves:**

- A *parameterization* of class \mathcal{C}^k ($k \geq 1$) of a curve in \mathbb{R}^n is a smooth map $c: I = [a, b] \subset \mathbb{R} \mapsto \mathbb{R}^n$, where c is of class \mathcal{C}^k



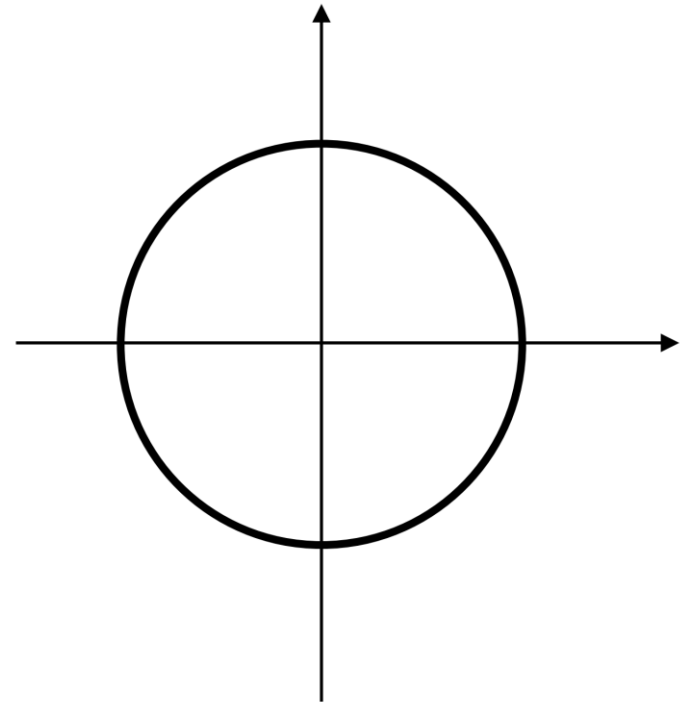
Parametric Curves

- **Parametric Curves:**

- The image set $c(I)$ is called the *trace* of the curve
 - Different parameterizations can have the same trace.
- A point in the trace, which corresponds to more than one parameter value t , is called *self-intersection* of the curve

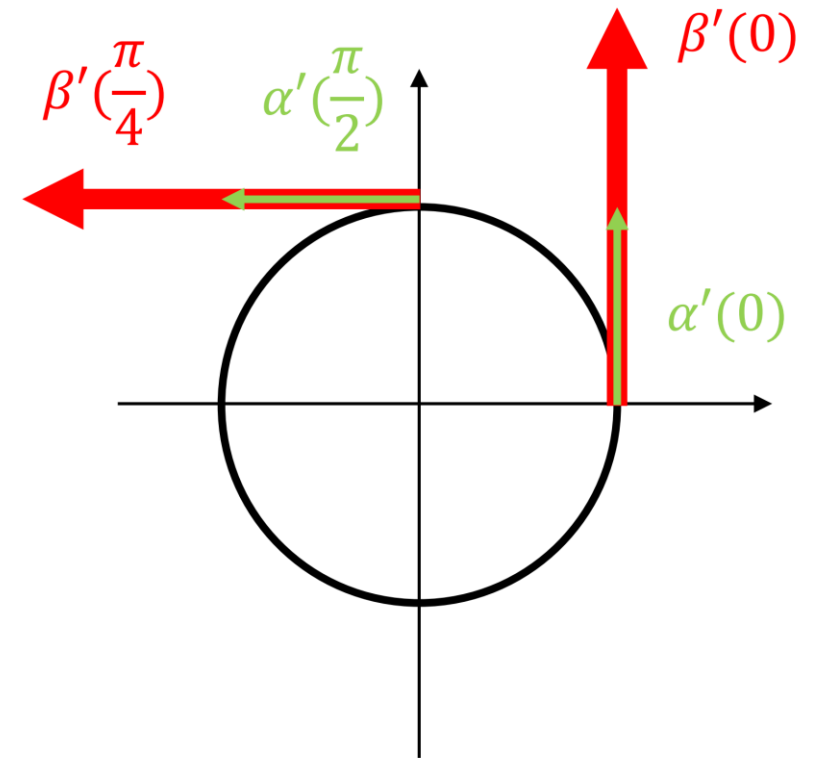
Parametric Curves: Examples

- The *positive* x-axis
 - $c(t) = (t, 0), t \in (0, \infty)$
 - $c(t) = (e^t, 0), t \in \mathbb{R}$
- **Circle**
 - $c(t) = (\cos t, \sin t), \quad t \in [0, 2\pi]$
 - $c(t) = (\cos 2t, \sin 2t), \quad t \in [0, \pi]$
 - $c(t) = (\cos t, \sin t), \quad t \in \mathbb{R}$



The velocity vector

- The derivative $c'(t)$ is called the **velocity vector** to the curve c at time t
 - $c'(t)$ gives the direction of the movement
 - $|c'(t)|$ gives the speed
- Example
 - $\alpha(t) = (\cos t, \sin t), \quad t \in [0, 2\pi]$
 - $\beta(t) = (\cos 2t, \sin 2t), \quad t \in [0, \pi]$



Regular parametric curves

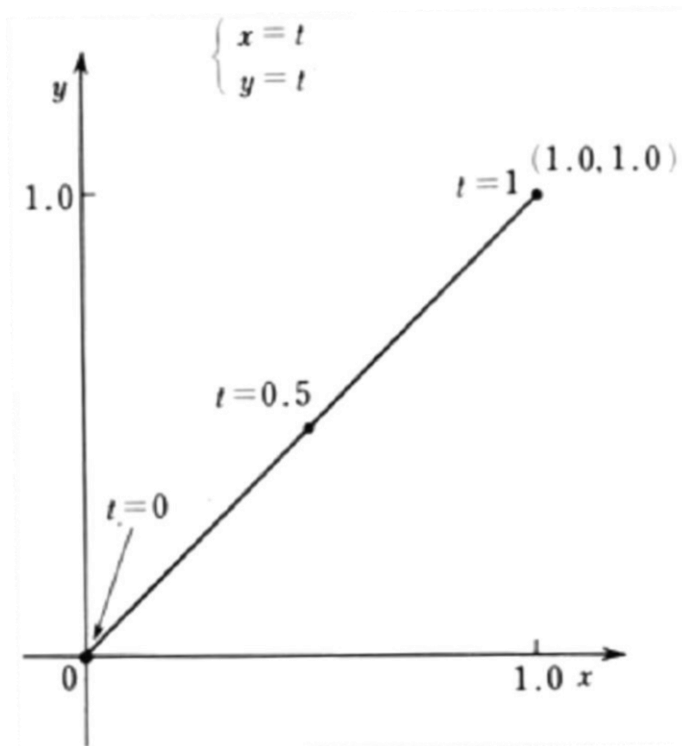
- **Regular parametrization**

- A parameterization is called *regular* if $c'(t) \neq 0$ for all t
- A point at which a curve is regular is called an *ordinary* point
- A point at which a curve is non-regular is called an *singular* point

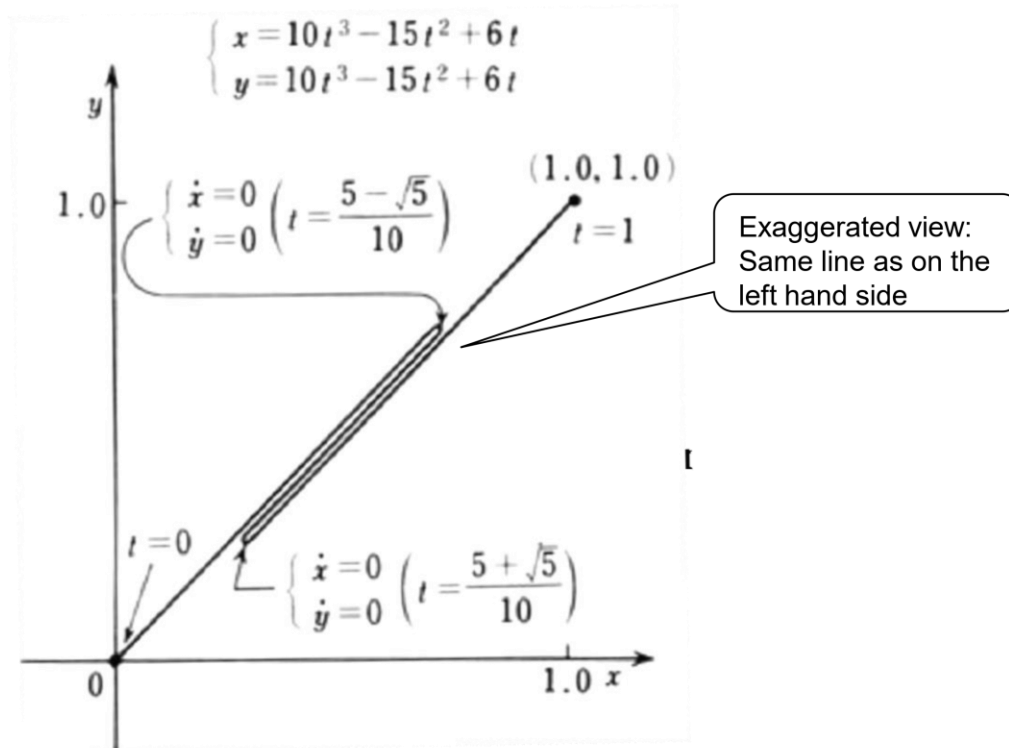
Examples: regularity

- Examples: issues with non-regular parameterization

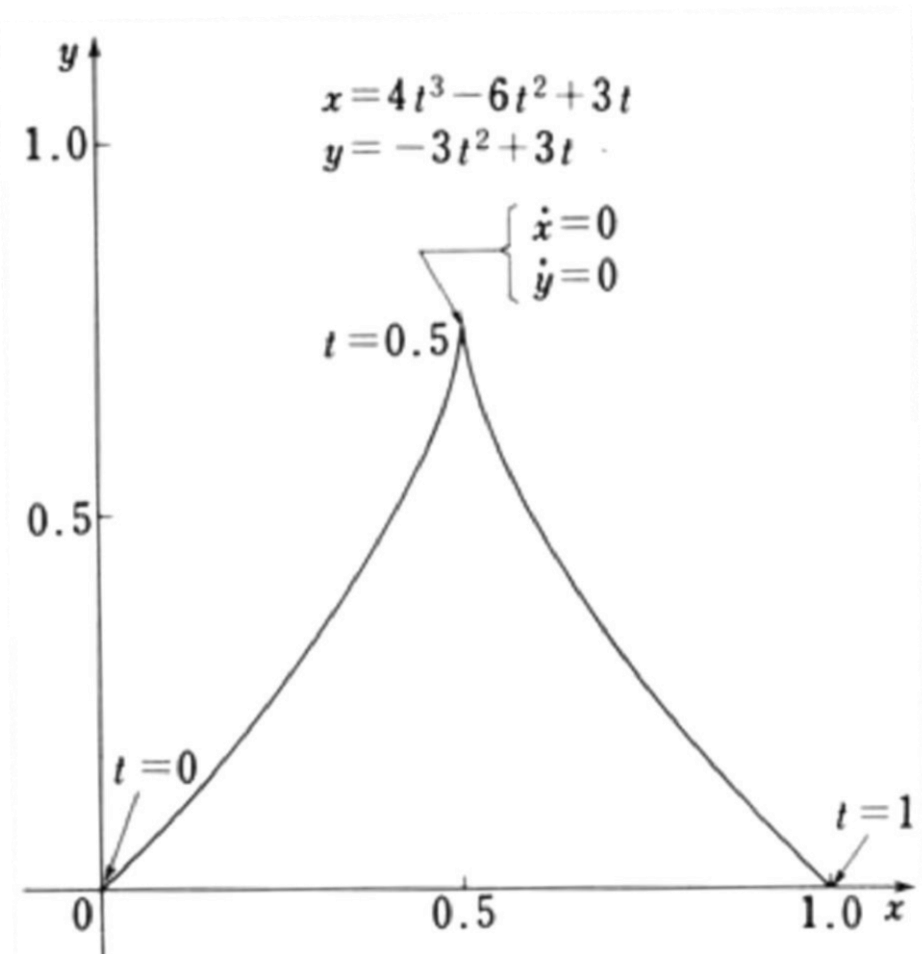
Regular parametrization



Non-regular parametrization

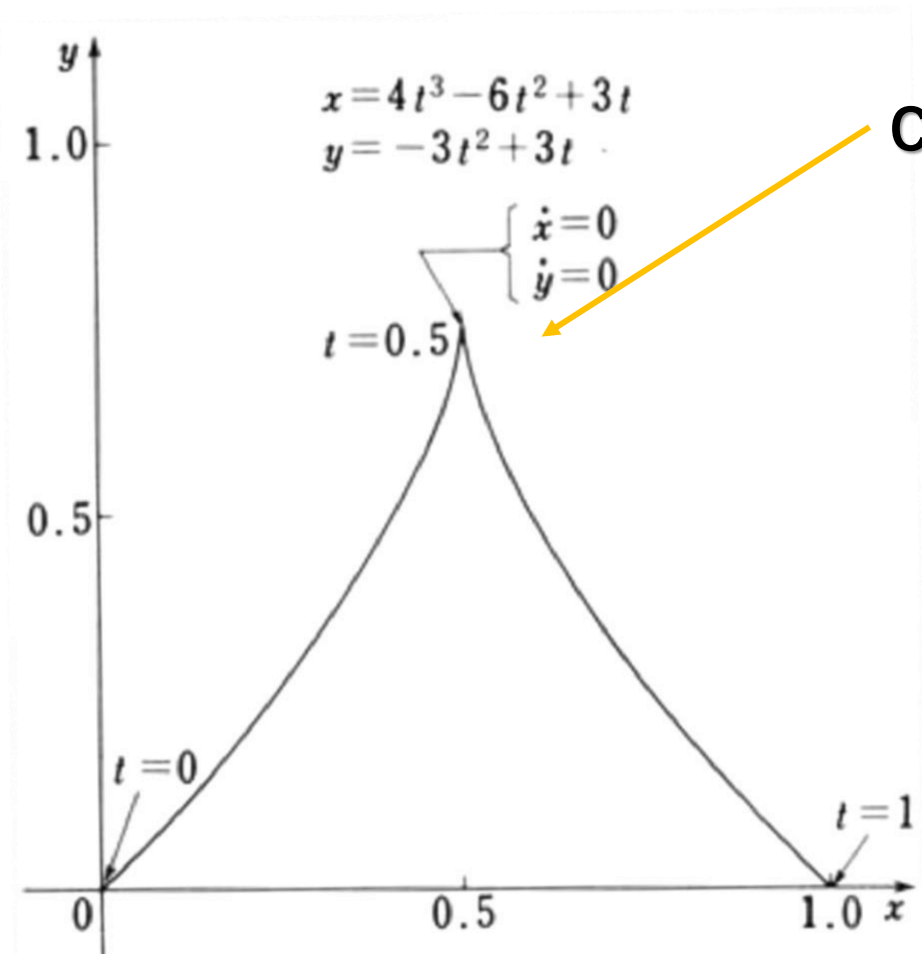


Examples: cusps



Singularities can be desired design features

Examples: cusps



Cusp



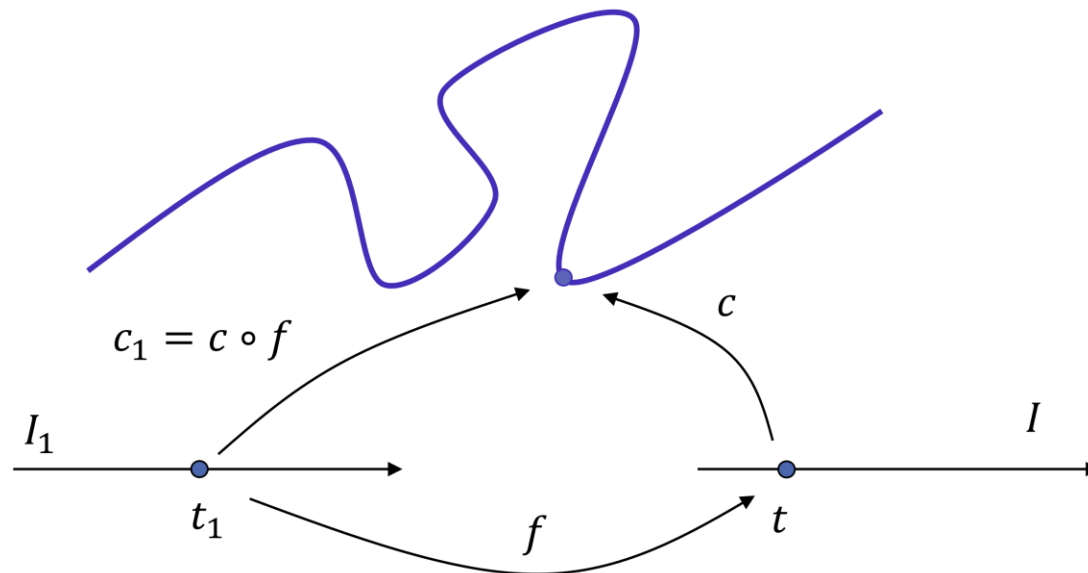
Singularities can be desired design features

Change of parameterization

- Given a smooth regular parametrization, an **allowable** change of parameter is any real smooth (differentiable) function

$$f: I_1 \rightarrow I \text{ such that } f' \neq 0 \text{ on } I_1$$

- It is **orientation preserving** when $f' > 0$



Change of parameterization

- **Parameter Transformations:**

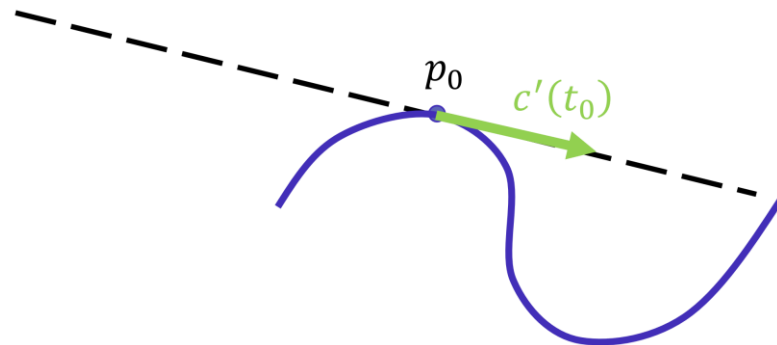
- We can regard a *regular curve* as a collection of regular parameterizations, any two of which are reparameterizations of each other (equivalence class)
- We are interested in properties that are *invariant* under parameter transformations

Geometric observations

- **Tangent vector:**

- The tangent line to a regular curve $c(t)$ at $p_0 = c(t_0)$ can be defined as points p which satisfy $p - p_0 \parallel c'_0$, where $c'_0 = c'(t_0)$

- The normalized vector $t = \frac{c'}{|c'|}$ is called the tangent vector

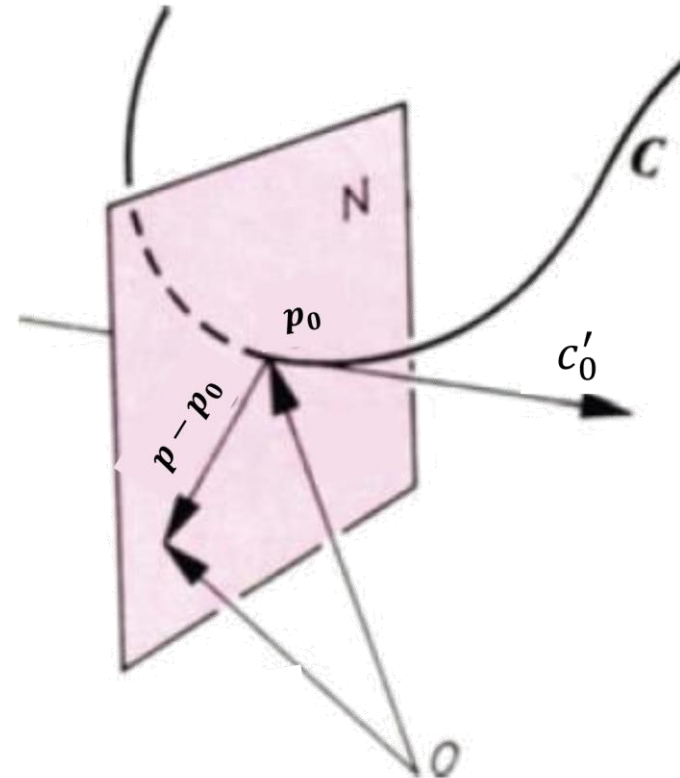


Geometric observations

- **The normal plane:**

- The normal plane can be obtained as points p whose coordinates satisfy $p - p_0 \perp c'_0$

$$\Leftrightarrow (p - p_0) \cdot c'_0 = 0$$

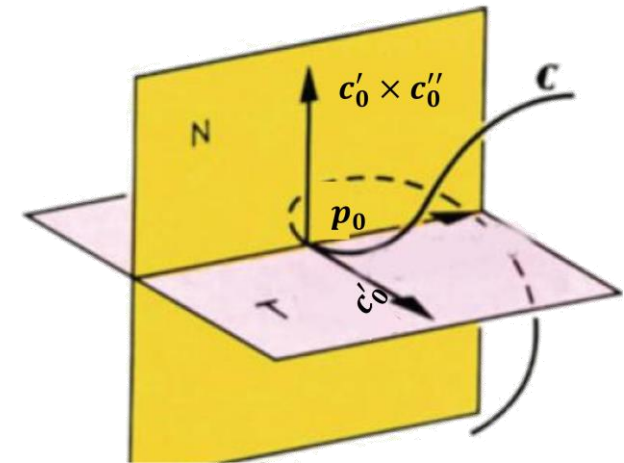


Geometric observations

- **Osculating plane: 密切平面**

- Assume the curve $c(t)$ is not a straight line. Any three arbitrary non-collinear points p_1, p_2, p_3 determine a plane
- If p_1, p_2, p_3 tend to the same points p_0 of c , then their plane converges to a plane called the osculating plane T of c at p_0
- The osculating plane is well defined if the first two derivatives c'_0 and c''_0 at p_0 are linearly independent and is give as:

$$(c'_0 \times c''_0) \cdot (p - p_0) = 0$$



Geometric observations

Observe the distance between $P(t_0 + \Delta t)$ and a given plane passing through $P(t_0)$ with normal vector a

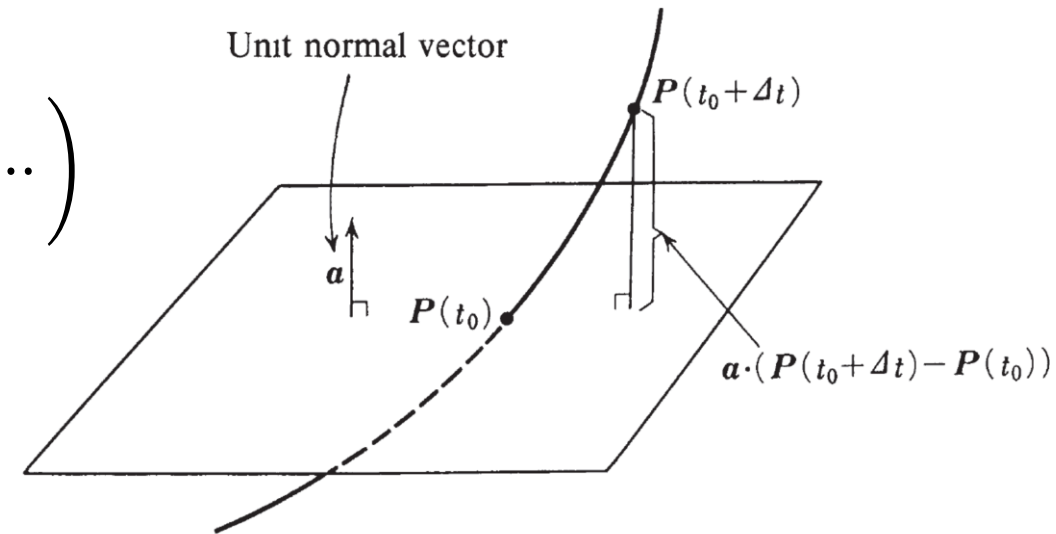
$$a \cdot (P(t_0 + \Delta t) - P(t_0)) = a \cdot \left(\dot{P}(t_0)\Delta t + \frac{\ddot{P}(t_0)}{2!}\Delta t^2 + \dots \right)$$

The distance is minimal when

$$a \cdot \dot{P}(t_0) = 0, a \cdot \ddot{P}(t_0) = 0$$

That is when the plane is osculating

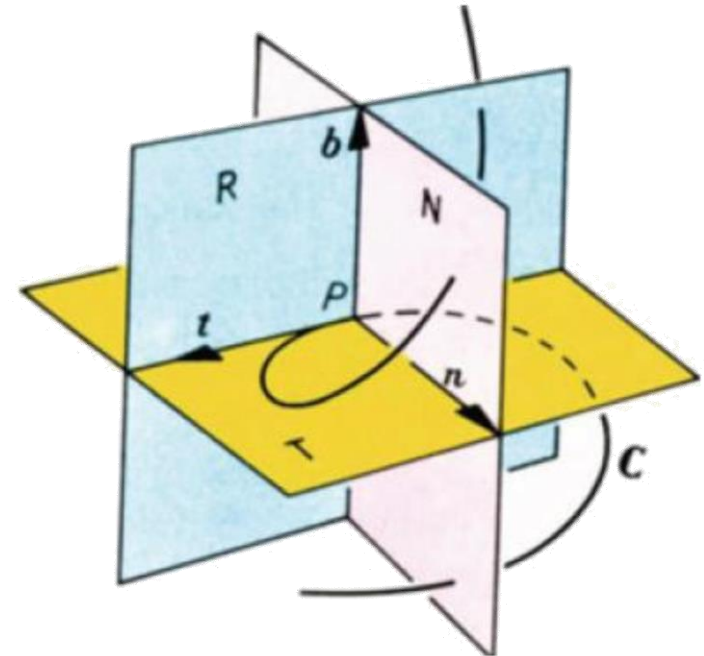
→ The osculating plane is the plane that best fits the curve at $P(t_0)$



Geometric observations

- The rectifying plane: 从切平面
 - The plane normal to both, the osculating plane and the normal plane, is called the rectifying plane R and can be obtained as points p whose coordinates satisfy

$$(c'_0 \times (c'_0 \times c''_0)) \cdot (p - p_0) = 0$$



Geometric observations

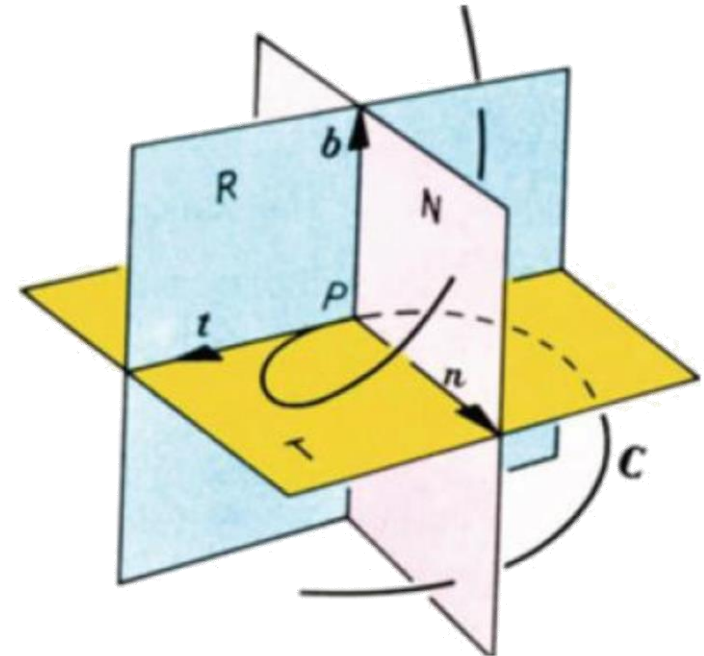
Normals: any vector in the normal plane is normal to the curve, in particular:

- The normal \mathbf{n} lying in the osculating plane is called the **principal normal** at p_0 .

It has a direction $(c'_0 \times c''_0) \times c'_0$

- The normal \mathbf{b} lying in the rectifying plane is called the **binormal**. 副法向

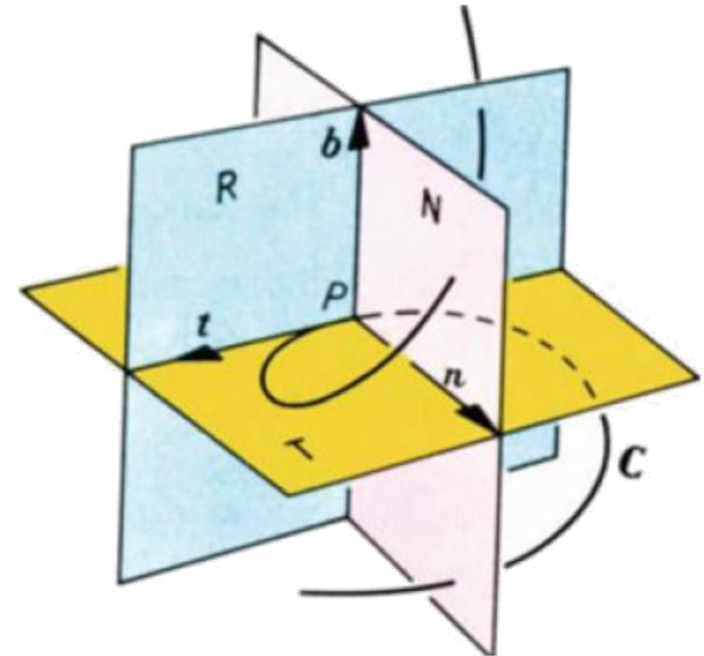
It has a direction $c'_0 \times c''_0$



The Frenet frame

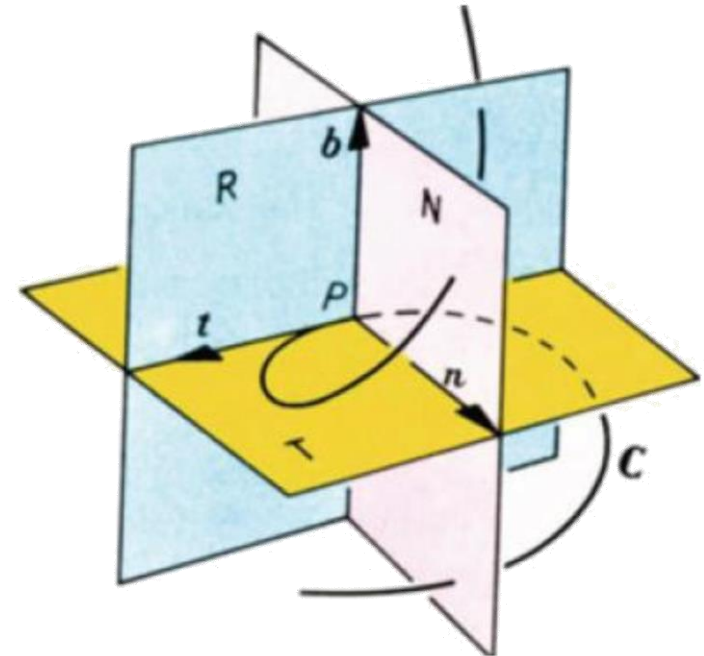
We can define a local coordinates system on the curve by three vectors

- The tangent $t = \frac{c'}{\|c'_0\|}$
- The binormal $b = \frac{c'_0 \times c''_0}{\|c'_0 \times c''_0\|}$
- The principal normal $n = b \times t$



The Frenet frame and associated planes

- The tangent $\mathbf{t} = \frac{\mathbf{c}'}{\|\mathbf{c}'\|}$
 - the normal plane $(\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{t} = 0$
- The binormal $\mathbf{b} = \frac{\mathbf{c}' \times \mathbf{c}''}{\|\mathbf{c}' \times \mathbf{c}''\|}$
 - the osculating plane $(\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{b} = 0$
- The principal normal $\mathbf{n} = \mathbf{b} \times \mathbf{t}$
 - the rectifying plane $(\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{n} = 0$



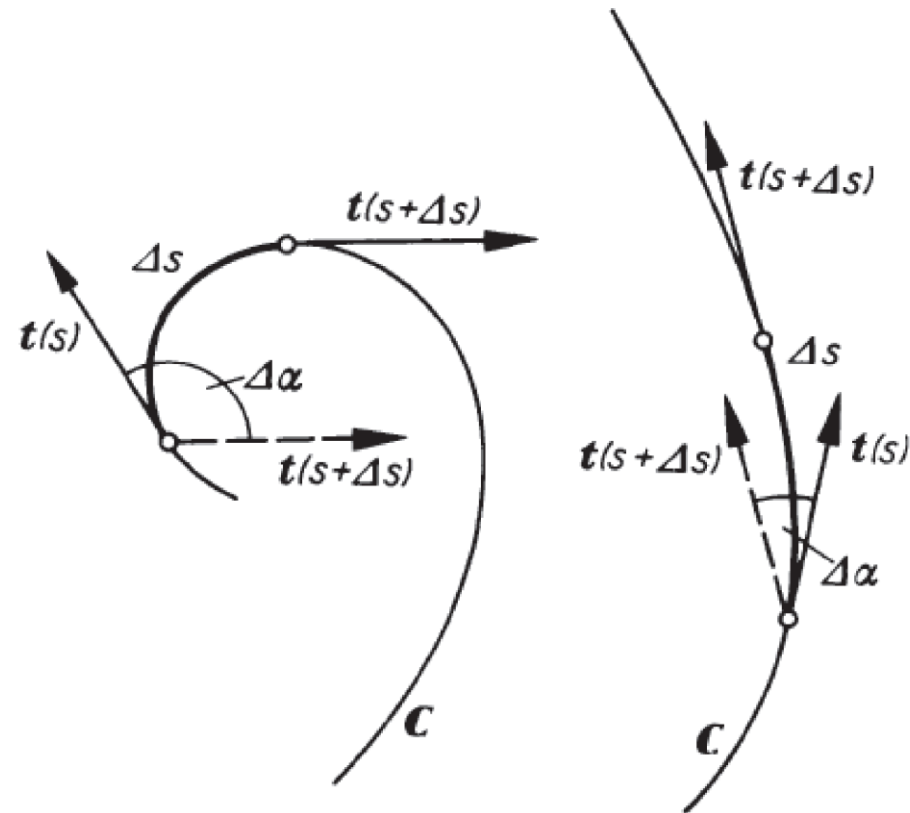
Curvature

- Common conceptions of curvature
 - Measures bending of a curve
 - A straight line does not bend \rightarrow 0 curvature
 - A circle has constant bending \rightarrow constant curvature

Curvature

Euler's heuristic approach for planar curves

- Variation of the tangent angle: how much does the curve differ from a straight line



Curvature for regular parameterization

The curvature is denoted by κ and defined as

$$\kappa(t) = \frac{\|c'(t) \times c''(t)\|}{\|c'(t)\|^3}$$

Examples:

- Consider the circle $c(t) = (r \cos t, r \sin t, 0)$

The **curvature** is given by

$$\kappa(t) = \frac{\|(-r \sin t, r \cos t, 0) \times (-r \cos t, -r \sin t, 0)\|}{r^3} = \frac{\|(0, 0, r^2)\|}{r^3} = \frac{1}{r}$$

- Consider the helix $c(t) = (r \cos t, r \sin t, at)$, the **curvature** is

$$\kappa(t) = \frac{r}{r^2 + a^2}$$

$$\kappa(t) = \frac{\|c'(t) \times c''(t)\|}{\|c'(t)\|^3}$$

Special case: planar curves

- For a regular planar curve $c(t) = (x(t), y(t))$

$$\kappa(t) = \frac{|x'y'' - x''y'|}{(x'^2 + y'^2)^{\frac{3}{2}}}$$

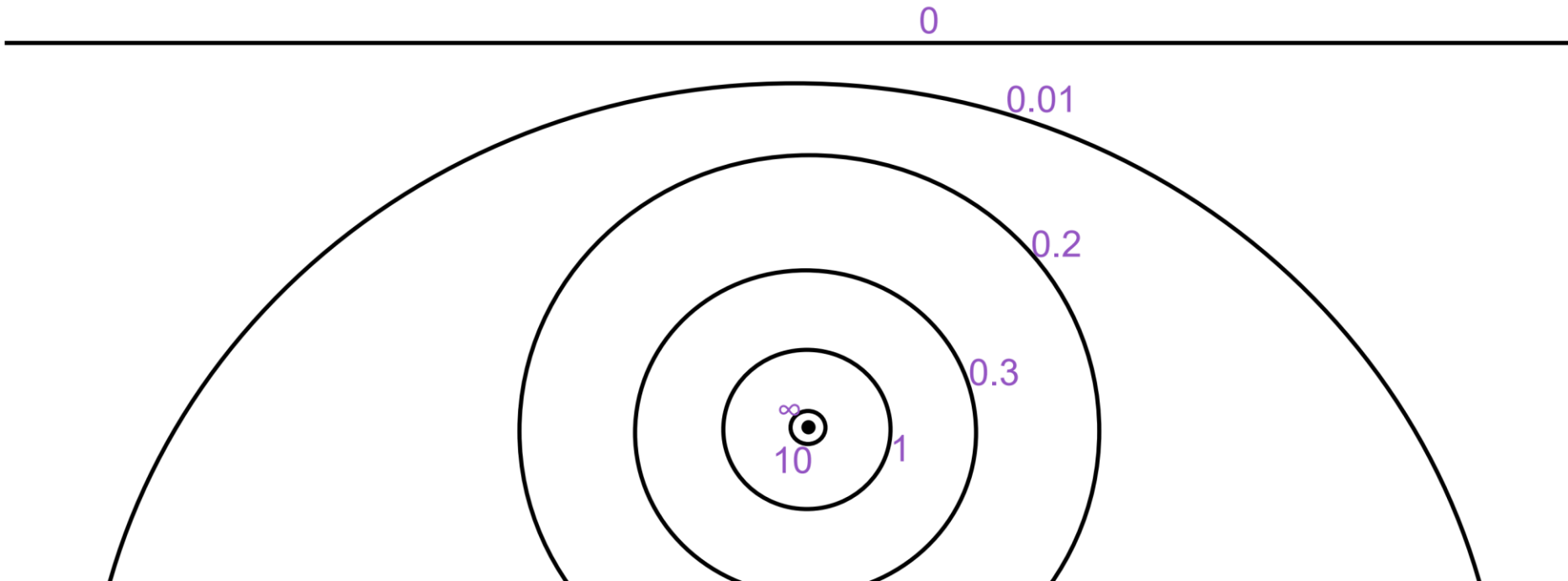
- Sometimes we talk about **signed curvature**, and then curvature can be allowed to be signed (negative, zero, or positive)

$$\kappa(t) = \frac{x'y'' - x''y'}{(x'^2 + y'^2)^{\frac{3}{2}}}$$

Examples

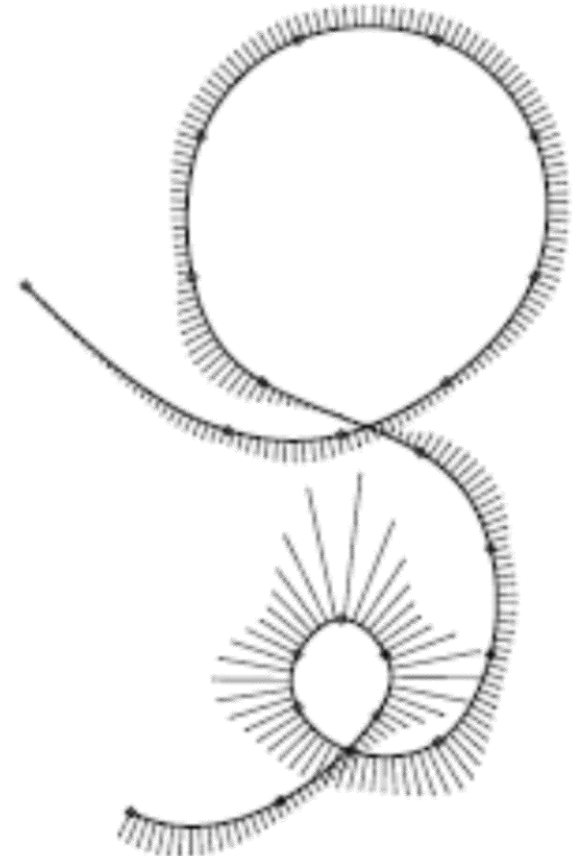
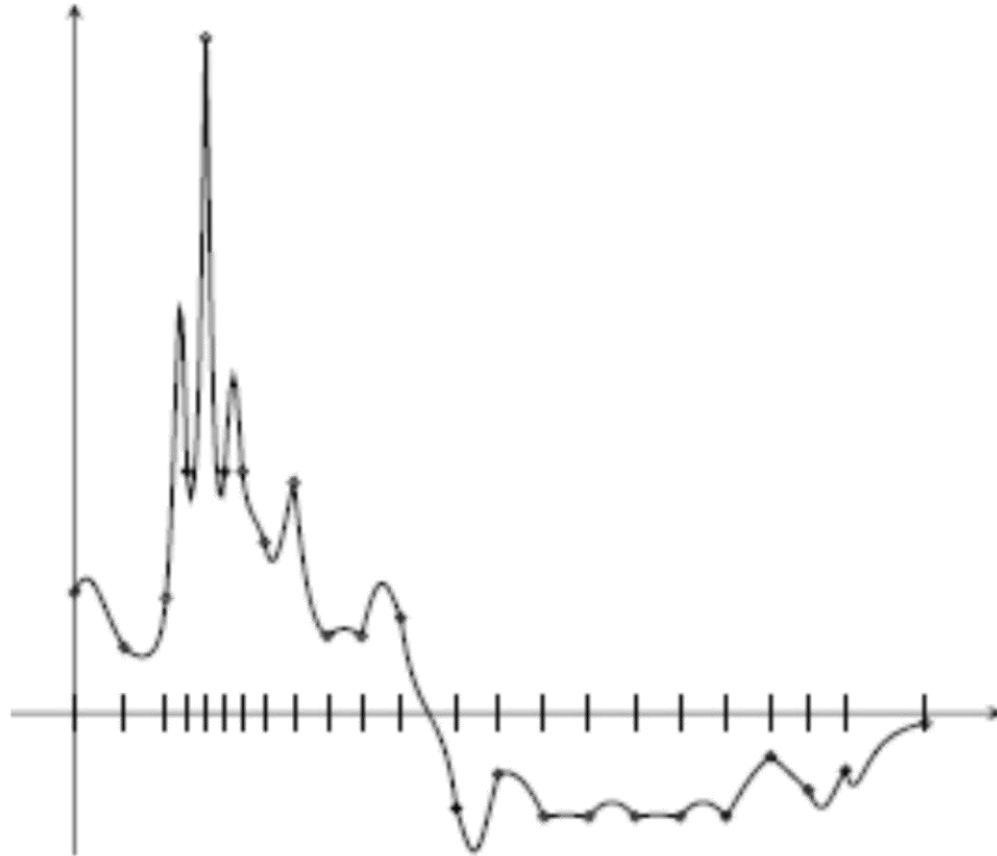
Curvature of circles

- Curvature of a circle is constant, $\kappa \equiv \frac{1}{r}$ (r = radius)
- Accordingly: define *radius of curvature* as $\frac{1}{\kappa}$



Curvature in practice

Most of commercial package allow inspecting the quality of the curvature

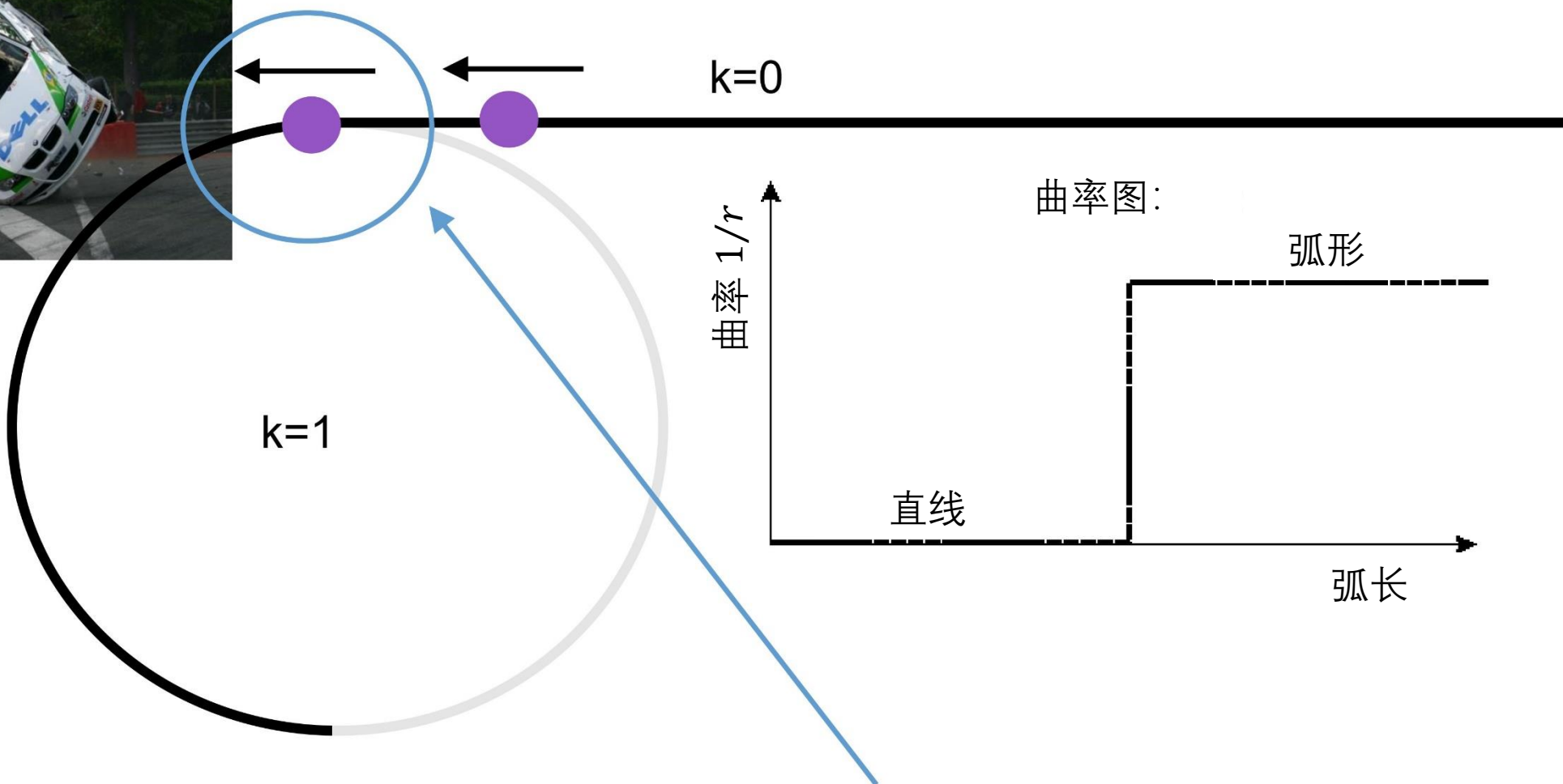


Curvature in practice

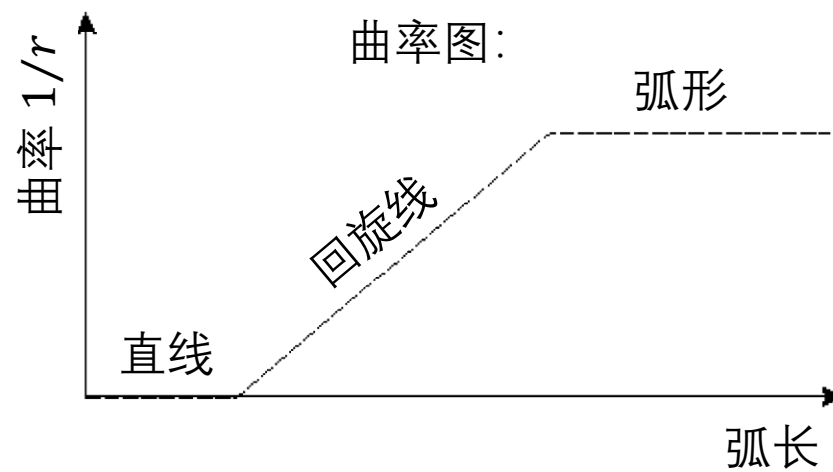
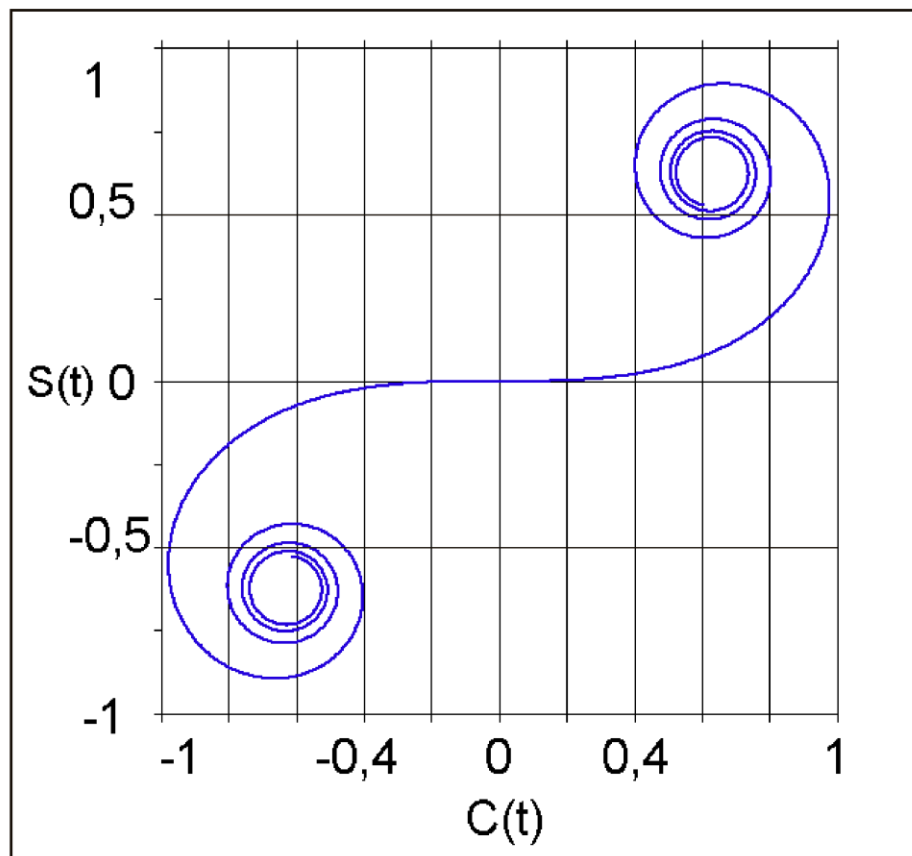
Most commercial package allow checking the quality of the curvature even meticulously!



Curvature and Road Construction



Clothoide, Euler Spiral 羊角螺线



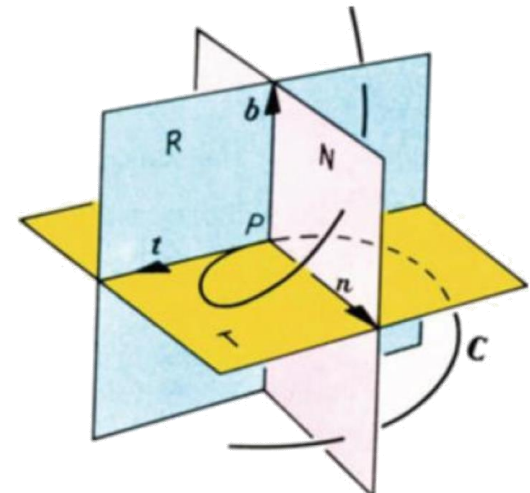
$$c(t) = \begin{pmatrix} \int_0^t \cos \frac{\pi}{2} u^2 du \\ \int_0^t \sin \frac{\pi}{2} u^2 du \end{pmatrix}$$

Torsion for regular parameterization

Definition

- The torsion τ measures the variation of the binormal vector
- (deviation of the curve from its projection on the osculating plane, can be regarded as **how far is the curve is from being a planar curve**) and is given by

$$\tau(t) = \frac{(c' \times c'') \cdot c'''}{\|c' \times c''\|^2}$$



Torsion

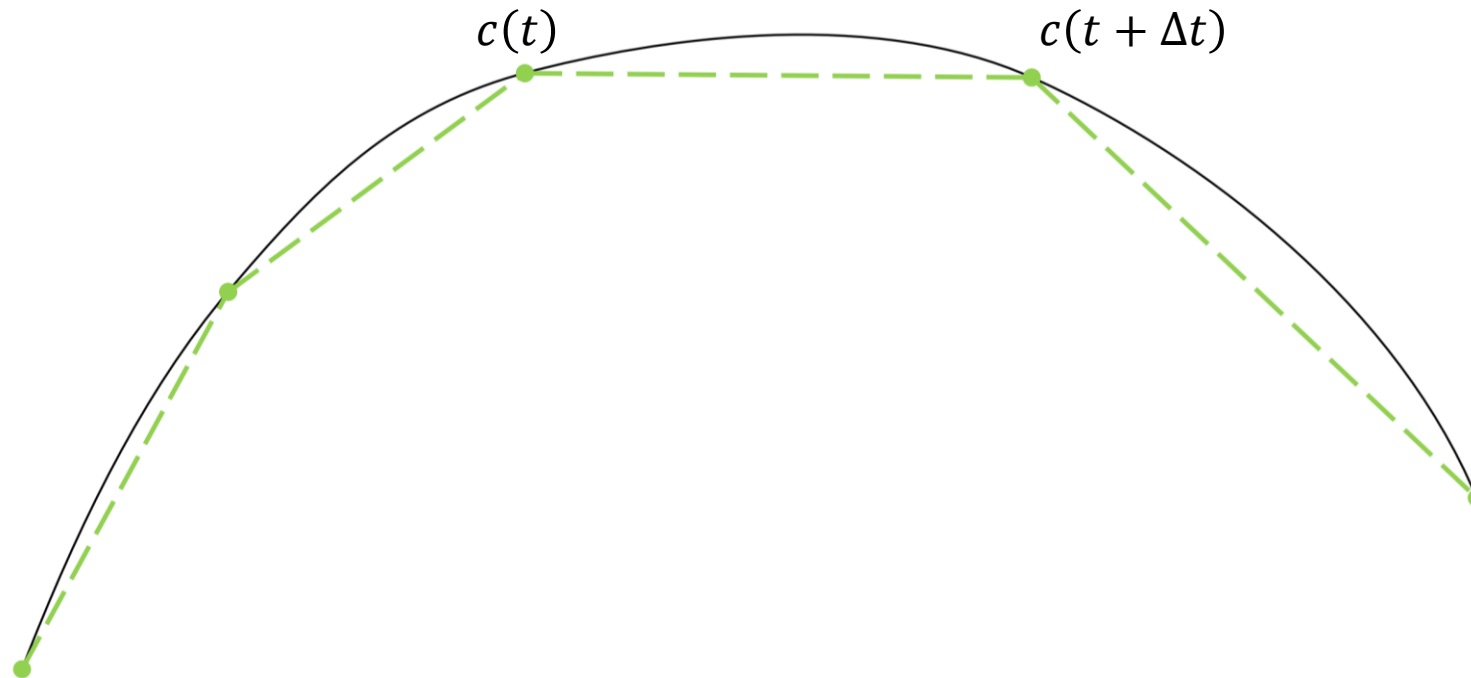
Examples:

- Torsion for a planar curve
- Torsion for a quadratic curve

Measuring lengths on curves

The arc length of a curve

- Can be regarded as the limit of the sum of infinitesimal segments along the curve



Measuring lengths on curves

The arc length of a curve

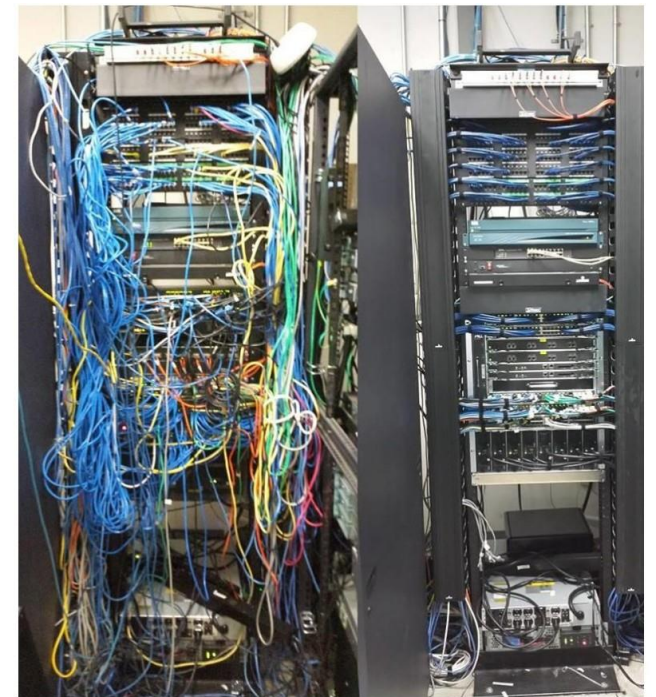
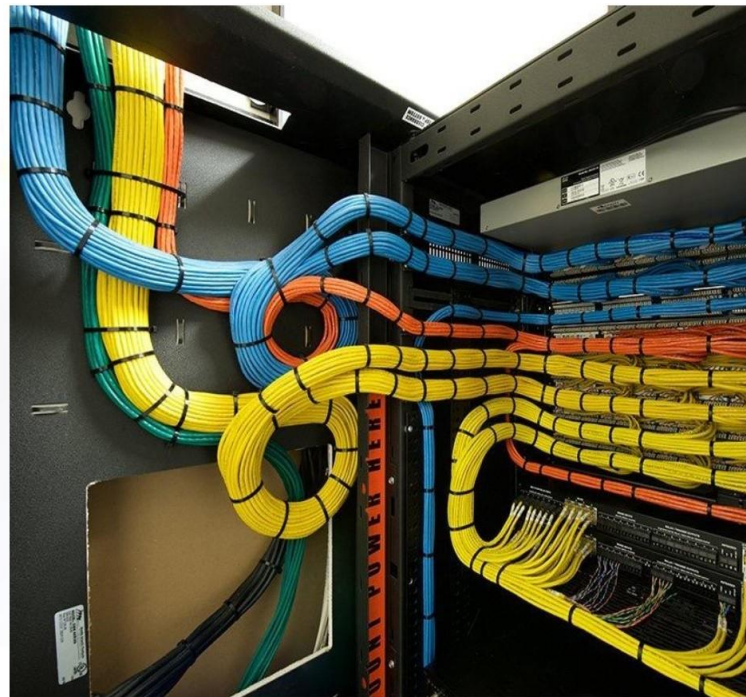
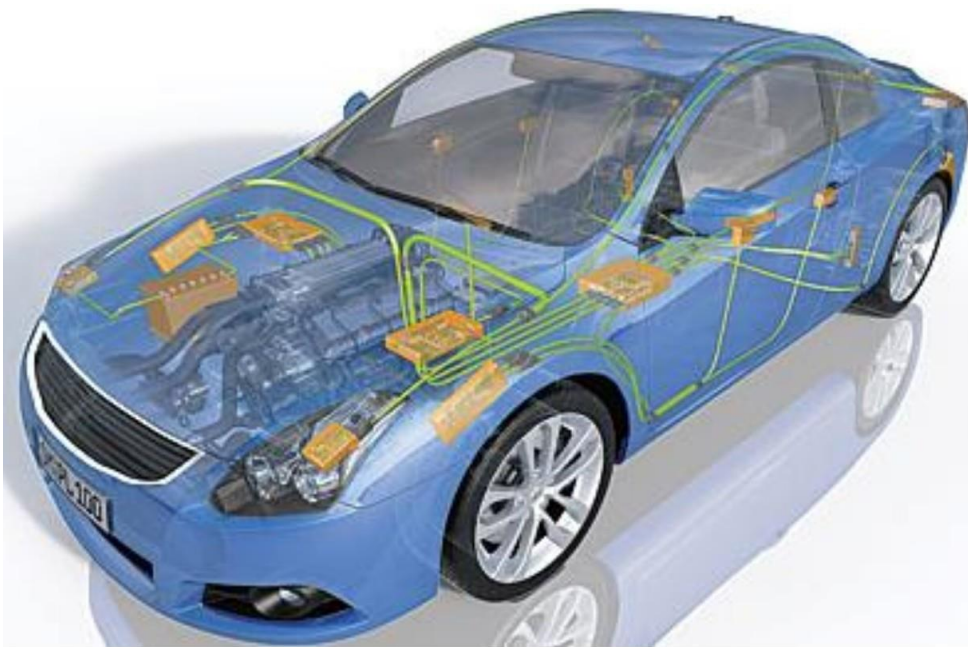
- The arc length of a regular curve C is defined as :

$$\text{length}_C = \int_a^b \|c'\| dt$$

- Independent of the parameterization (to prove this, use integration by substitution)

Measuring lengths on curves

Curve arc length matters in practice (e.g., cable routing problems)



Arc-length parametrized curves

Arc length parametrization

- Consider the portion of $c(t)$ spanned from 0 to t , the length s of this arc is a function of t :

$$s(t) = \int_0^t \|c'(u)\| dt$$

- Since $\frac{ds}{dt} = \|c'(u)\| > 0$ (why?) $\rightarrow s$ can be introduced as a new parameterization

Arc length parametrization

- Consider the portion of $c(t)$ spanned from 0 to t , the length s of this arc is a function of t :

$$s(t) = \int_0^t \|c'(u)\| dt$$

- Since $\frac{ds}{dt} = \|c'(u)\| > 0$ (why?) $\rightarrow s$ can be introduced as a new parameterization
- We have $c'(s) = \frac{dc}{ds} = \frac{dc/dt}{ds/dt} \Rightarrow \|c'(s)\| = 1$
- $c(s)$ is called an *arc-length* (or *unit-speed*) *parametrized curve*, the parameter s is called the *arc length* of c or the *natural parameter*

Reparameterization by arc length

- Arc-length (or unit-speed) parameterization:
 - Any regular curve admits an arc-length parameterization
 - This does not mean that the arc-length parameterization can be computed

Examples

$$s(t) = \int_0^t \|c'(u)\| dt$$

- Find an arc-length parameterization for the Helix: $\begin{pmatrix} \cos t \\ \sin t \\ t \end{pmatrix}$

Examples

$$s(t) = \int_0^t \|c'(u)\| dt$$

- Find an arc-length parameterization for the Helix: $\begin{pmatrix} \cos t \\ \sin t \\ t \end{pmatrix}$

$$s(t) = \int_0^t \sqrt{(-\sin u)^2 + (\cos u)^2 + 1^2} du = t\sqrt{2} \Rightarrow t = \frac{s}{\sqrt{2}}$$

The arc-length parameterized Helix: $\begin{pmatrix} \cos \frac{s}{\sqrt{2}} \\ \sin \frac{s}{\sqrt{2}} \\ \frac{s}{\sqrt{2}} \end{pmatrix}$

Examples

$$s(t) = \int_0^t \|c'(u)\| dt$$

- How about the ellipse $\alpha(t) = \begin{pmatrix} 2 \cos t \\ \sin t \\ 0 \end{pmatrix}$?

Examples

$$s(t) = \int_0^t \|c'(u)\| dt$$

- How about the ellipse $\alpha(t) = \begin{pmatrix} 2 \cos t \\ \sin t \\ 0 \end{pmatrix}$?

$$s(t) = \int_0^t \sqrt{4(-\sin u)^2 + (\cos u)^2} du = \int_0^t \sqrt{4 - 3 \cos^2 u} du$$

Does not admit any closed form antiderivative

Examples

$$s(t) = \int_0^t \|c'(u)\| dt$$

- How about $\alpha(t) = \begin{pmatrix} t \\ t^2 \\ 2 \\ 0 \end{pmatrix}$?

Examples

$$s(t) = \int_0^t \|c'(u)\| dt$$

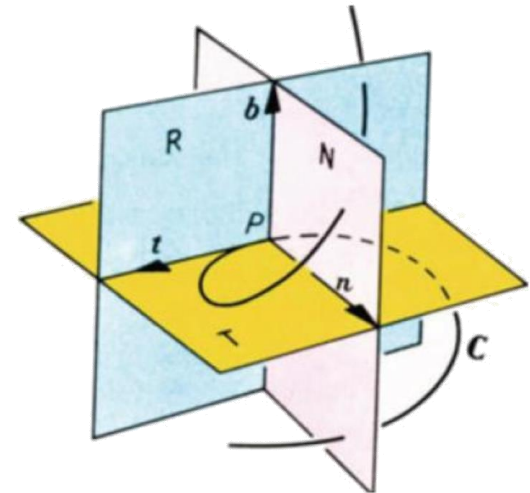
- How about $\alpha(t) = \begin{pmatrix} t \\ \frac{t^2}{2} \\ 0 \end{pmatrix}$?

$$s(t) = \int_0^t \sqrt{1 + u^2} du = t\sqrt{1 + t^2} + \ln(t + \sqrt{1 + t^2})$$

- No straightforward way to write t as a function of s !

Geometric consequences of Arc length parameterization

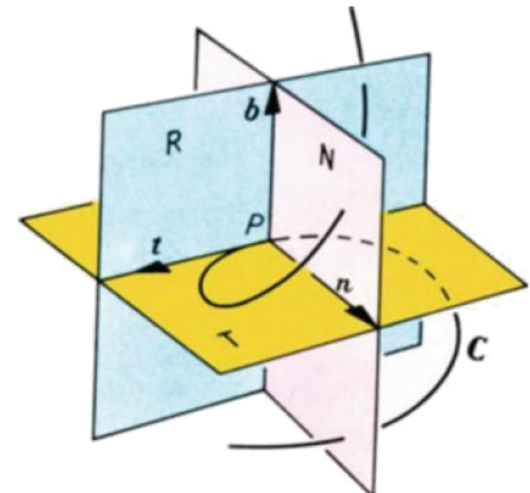
- Since $\|c'(u)\| = 1$



Geometric consequences of Arc length parameterization

- Since $\|c'(u)\| = 1$, by noting that $c' \cdot c' = 1$ and taking the derivative, we have $c' \cdot c'' = 0$
- c'' is perpendicular to c' (both live on the osculating plane)
- Therefore c'' is a direction vector of the principal normal (provided that $c'' \neq 0$)

$$\Rightarrow n = \frac{c''}{\|c''\|}$$



$$\kappa(t) = \frac{\|c'(t) \times c''(t)\|}{\|c'(t)\|^3}$$

Curvature again

- The curvature of an **arc-length parametrized** curve (unit speed curve) $c(t)$ simplifies to

$$\kappa = \|c''(u)\|$$

Further mathematical formulations: Frenet Curves

Frenet Curves

- Frenet curves
 - A *Frenet curve* is an arc-length parametrized curve c in \mathbb{R}^n such that $c'(s), c''(s), \dots, c^{n-1}(s)$ are linearly independent

Frenet Curves

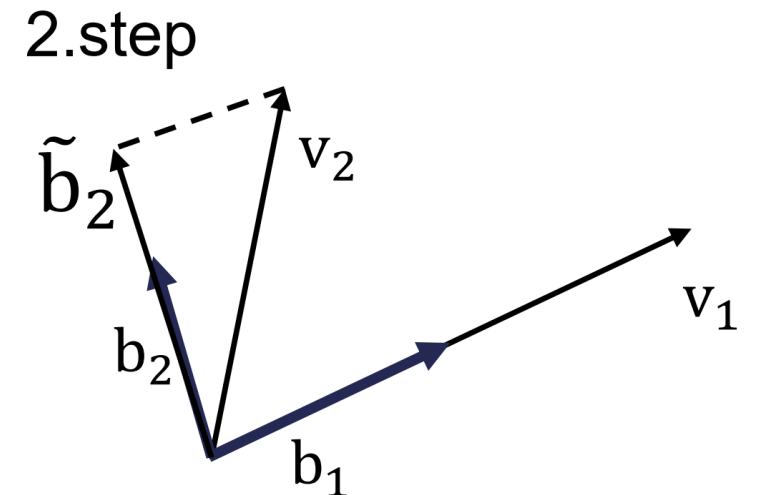
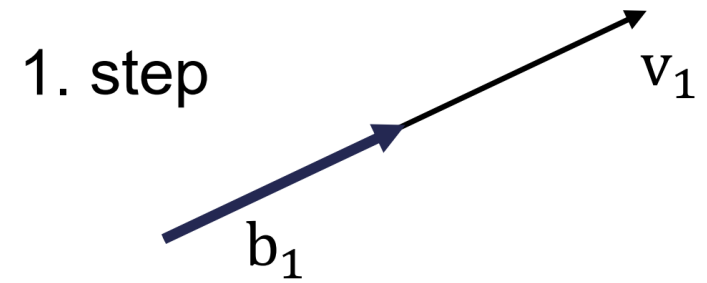
- Frenet curves
 - A *Frenet curve* is an arc-length parametrized curve c in \mathbb{R}^n such that $c'(s), c''(s), \dots, c^{n-1}(s)$ are linearly independent
- Frenet frame
 - Every Frenet curve has a unique Frenet frame $e_1(s), e_2(s), \dots, e_n(s)$ that satisfies
 - $e_1(s), e_2(s), \dots, e_n(s)$ is orthonormal and positively oriented

Frenet Curves

- Frenet curves
 - A *Frenet curve* is an **arc-length** parametrized curve c in \mathbb{R}^n such that $c'(s), c''(s), \dots, c^{n-1}(s)$ are linearly independent
- Frenet frame
 - Every Frenet curve has a unique Frenet frame $e_1(s), e_2(s), \dots, e_n(s)$ that satisfies
 - $e_1(s), e_2(s), \dots, e_n(s)$ is orthonormal and positively oriented
 - Apply the Gram-Schmidt process to $\{c', c'', \dots, c^n\}$

Gram-Schmidt Process: Construction of Orthonormal Bases

- Input: Linear independent set $\{v_1, v_2, \dots, v_n\}$
- Output: Orthogonal set $\{b_1, b_2, \dots, b_n\}$
 - Set $b_1 = \frac{v_1}{\|v_1\|}$
 - For $k = 2, \dots, n$
 - $\widetilde{b}_k = v_k - \sum_{i=1}^{k-1} \langle v_k, b_i \rangle b_i$
 - $b_k = \frac{\widetilde{b}_k}{\|\widetilde{b}_k\|}$



$$\kappa(t) = \frac{\|c'(t) \times c''(t)\|}{\|c'(t)\|^3}$$

Planar Curves

The Frenet Frame of an arc-length parametrized planar curve

Tangent vector

$$e_1(s) = c'(s)$$

Normal vector

$$e_2(s) = R^{90^\circ} e_1(s)$$

Frame equation

$$\begin{pmatrix} e_1(s) \\ e_2(s) \end{pmatrix}' = \begin{pmatrix} 0 & \kappa(s) \\ -\kappa(s) & 0 \end{pmatrix} \begin{pmatrix} e_1(s) \\ e_2(s) \end{pmatrix}$$

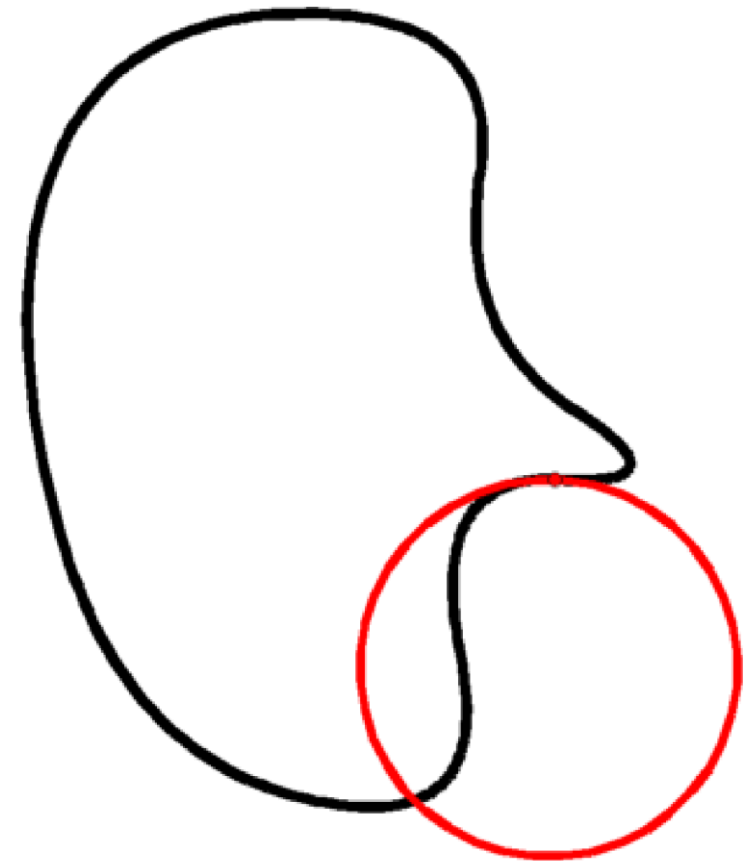
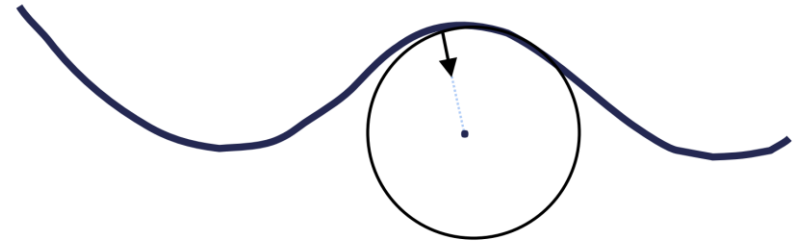
Signed Curvature

$\kappa(s) = \langle e_1'(s), e_2(s) \rangle$ is called the *signed curvature* of the curve

Osculating circle

Osculating circle

- Radius: $1/\kappa$
- Center: $c(s) + \frac{1}{\kappa}e_2(s)$



Properties

- Rigid motions
 - Rigid motion: $x \rightarrow Ax + b$ with orthogonal A (in other words: affine maps that preserve distances)
 - Orientation preserving (no mirroring) if $\det A = +1$
 - Mirroring leads to $\det A = -1$
- Invariance under rigid motions for planar curves
 - Curvature is invariant under rigid motion
 - Absolute value is invariant
 - Signed value is invariant for orientation preserving rigid motion
- Rigidity of planar curves
 - Two Frenet curves with identical signed curvature function differ only by an orientation preserving rigid motion

Fundamental Theorem

Fundamental theorem for planar curves

- Let $\kappa: (a, b) \mapsto \mathbb{R}$ be a smooth function. For some $s_0 \in (a, b)$, suppose we are given a point p_0 and two orthonormal vectors t_0 and n_0 . Then there exists a unique Frenet curve $c: (a, b) \mapsto \mathbb{R}^2$ such that
 - $c(s_0) = p_0$
 - $e_1(s_0) = t_0$
 - $e_2(s_0) = n_0$
 - The curvature of c equals the given function κ
- In other words: for every smooth function there is a unique (up to rigid motion) curve that has this function as its curvature

Arc-length Derivative

- Arc-length parameterization
 - Finding an arc-length parameterization for a parameterized curve is usually difficult
 - Still one can compute the Frenet frame and its derivatives. For this we define the so called arc-length derivative
- Arc-length derivative
 - For a parameterized curve $c: [a, b] \mapsto \mathbb{R}^n$, we define the *arc-length derivative* of any differentiable function $f: [a, b] \mapsto \mathbb{R}$ as

$$f'(t) = \frac{1}{\|c'(t)\|} f'(t)$$

Compute the signed curvature

- Computing the Frenet frame
 - For $c: [a, b] \mapsto \mathbb{R}^2$, the Frenet frame at $c(t)$ can be computed as (using **arc length derivative**)

$$e_1(t) = \mathbf{c}'(t) = \frac{c'(t)}{\|c'(t)\|}$$
$$e_2(t) = R^{90^\circ} e_1(t)$$

- Computing the signed curvature
 - The signed curvature is given by

$$\kappa(t) = \langle e_1'(t), e_2(t) \rangle = \frac{\langle c''(t), R^{90^\circ} c'(t) \rangle}{\|c'(t)\|^3}$$

Space Curves

- Frenet frame of **arc-length parametrized** space curves

- Frenet frame of a Frenet curve in \mathbb{R}^3

- Tangent vector

$$e_1(s) = c'(s)$$

- Normal vector

$$e_2(s) = \frac{1}{\|c''(t)\|} c''(t)$$

- Binormal vector

$$e_3(s) = e_1(s) \times e_2(s)$$

Frenet Frame of Space Curves

- Frenet–Serret equations

$$\begin{pmatrix} e_1(s) \\ e_2(s) \\ e_3(s) \end{pmatrix}' = \begin{pmatrix} 0 & \kappa(s) & 0 \\ -\kappa(s) & 0 & \tau(s) \\ 0 & -\tau(s) & 0 \end{pmatrix} \begin{pmatrix} e_1(s) \\ e_2(s) \\ e_3(s) \end{pmatrix}$$

- The signed curvature still is $\kappa(s) = \langle e_1'(s), e_2(s) \rangle$

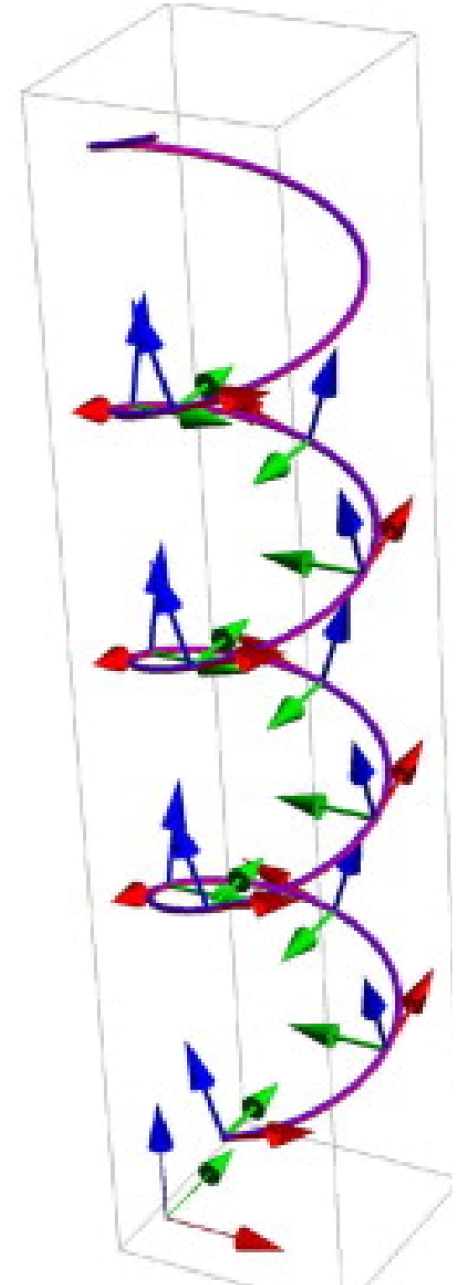
$$\tau(t) = \frac{(c' \times c'') \cdot c'''}{\|c' \times c''\|^2}$$

Frenet Frame of Space Curves

- Frenet–Serret equations

$$\begin{pmatrix} e_1(s) \\ e_2(s) \\ e_3(s) \end{pmatrix}' = \begin{pmatrix} 0 & \kappa(s) & 0 \\ -\kappa(s) & 0 & \tau(s) \\ 0 & -\tau(s) & 0 \end{pmatrix} \begin{pmatrix} e_1(s) \\ e_2(s) \\ e_3(s) \end{pmatrix}$$

- The torsion $\tau(s) = \langle e_2'(s), e_3(s) \rangle$ measures how the curve bends out of the plane spanned by e_1 and e_2



Frenet Frame of Space Curves

- Frenet equations for curves in \mathbb{R}^n

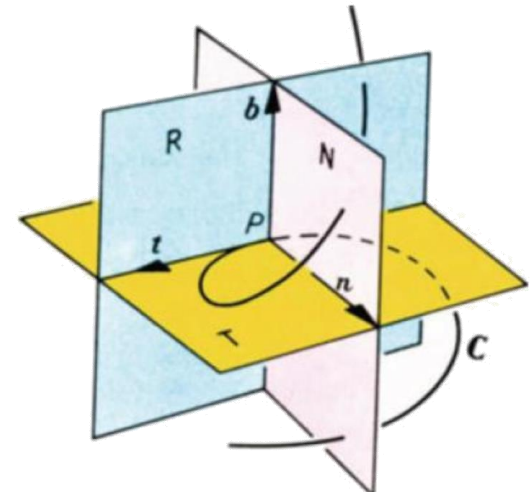
$$\begin{pmatrix} e_1(s) \\ e_2(s) \\ \dots \\ e_n(s) \end{pmatrix}' = \begin{pmatrix} 0 & \kappa_1(s) & 0 & \dots & 0 \\ -\kappa_1(s) & 0 & \kappa_2(s) & \dots & 0 \\ 0 & -\kappa_2(s) & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \kappa_{n-1}(s) \\ 0 & \dots & -\kappa_{n-1}(s) & 0 & 0 \end{pmatrix} \begin{pmatrix} e_1(s) \\ e_2(s) \\ \dots \\ e_n(s) \end{pmatrix}$$

- The function $\kappa_i(s)$ are called the i^{th} Frenet curvatures

Summary of relations

- For regular curves:

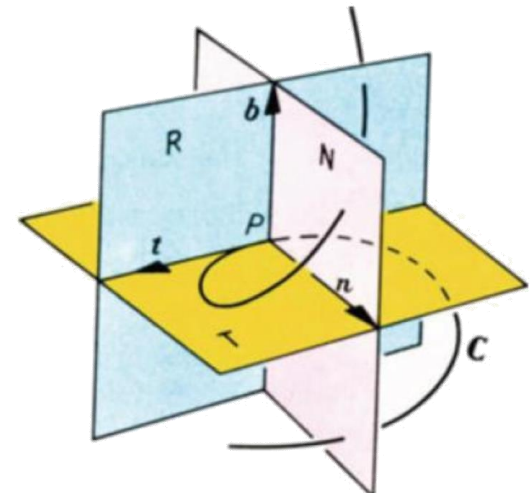
- The tangent $\mathbf{t} = \frac{\mathbf{c}'}{\|\mathbf{c}'\|}$, the normal plane $(\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{t} = 0$
- The binormal $\mathbf{b} = \frac{\mathbf{c}' \times \mathbf{c}''}{\|\mathbf{c}' \times \mathbf{c}''\|}$, the osculating plane $(\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{b} = 0$
- The principal normal $\mathbf{n} = \mathbf{b} \times \mathbf{t}$, the rectifying plane $(\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{n} = 0$
- The curvature $\kappa(t) = \frac{\|\mathbf{c}' \times \mathbf{c}''\|}{\|\mathbf{c}'\|^3}$
- The torsion $\tau(t) = \frac{(\mathbf{c}' \times \mathbf{c}'') \cdot \mathbf{c}'''}{\|\mathbf{c}' \times \mathbf{c}''\|^2}$



Summary of relations

For an arc-length parameterized (unit speed) curves $c(s)$:

- The tangent $t = c'$
- The binormal $b = t \times n$
- The principal normal $n = \frac{t'}{\|t'\|} = \frac{c''}{\|c''\|}$
- The curvature $\kappa(t) = \|t'\| = \|c''\|$
- The signed curvature $\kappa(s) = t' = c''$
- The torsion $\tau(t) = -b' \cdot n$



Special case: planar curves

- For a regular planar curve $c(t) = (x(t), y(t))$, it is defined as

$$\kappa(t) = \frac{|x'y'' - x''y'|}{(x'^2 + y'^2)^{\frac{3}{2}}}$$

- Sometimes we talk about **signed curvature**, and then curvature can be allowed to be signed (negative, zero, or positive)

$$\kappa(t) = \frac{x'y'' - x''y'}{(x'^2 + y'^2)^{\frac{3}{2}}}$$

Computer Aided Geometric Design

Fall Semester 2024

Bézier Splines

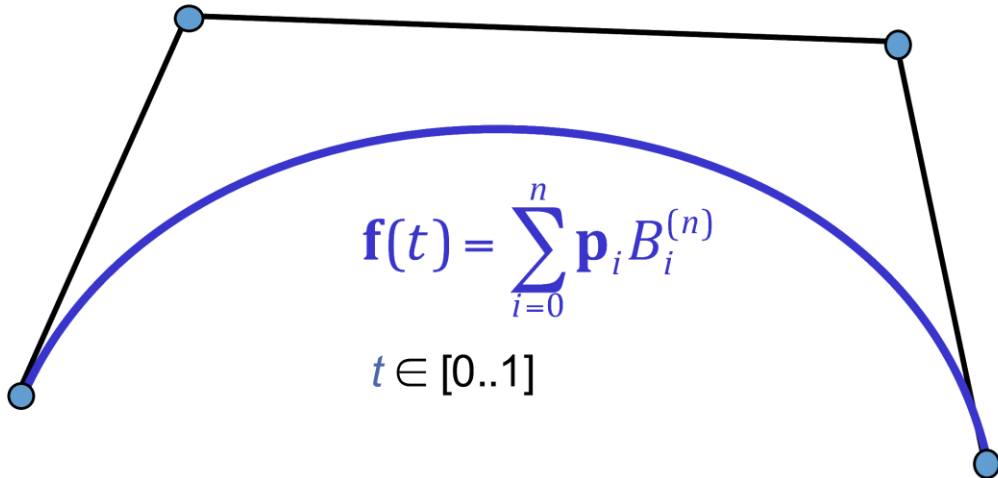
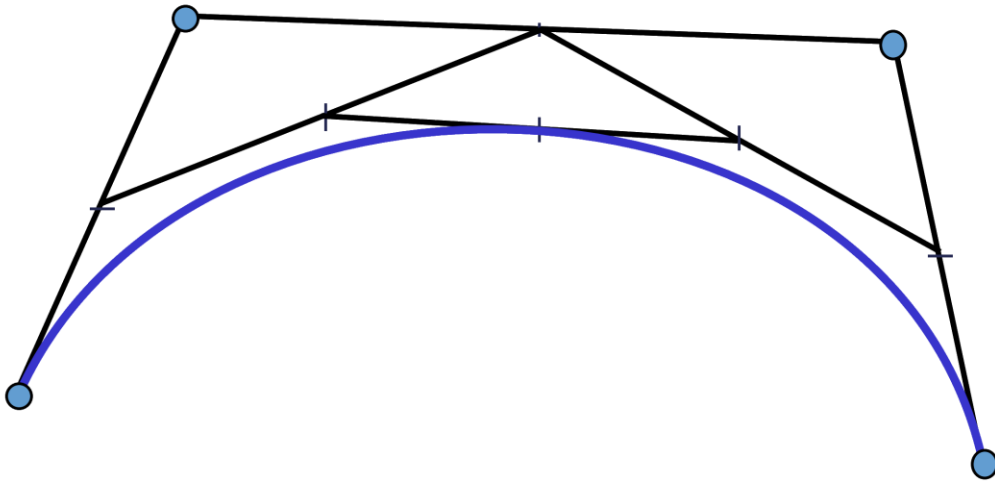
陈仁杰

renjiec@ustc.edu.cn

<http://staff.ustc.edu.cn/~renjiec>

Recap

de Casteljau algorithm

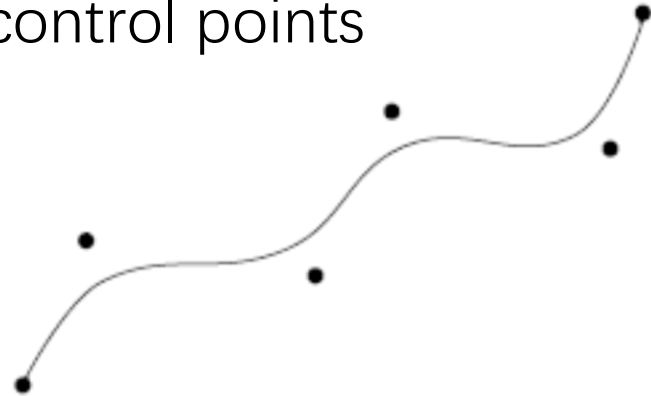


Bernstein form

Recap

- **bézier curves and curve design:**

- The rough form is specified by the position of the control points
- Results: smooth curve approximating the control points
- Computation / Representation
 - de Casteljau algorithm
 - Bernstein form



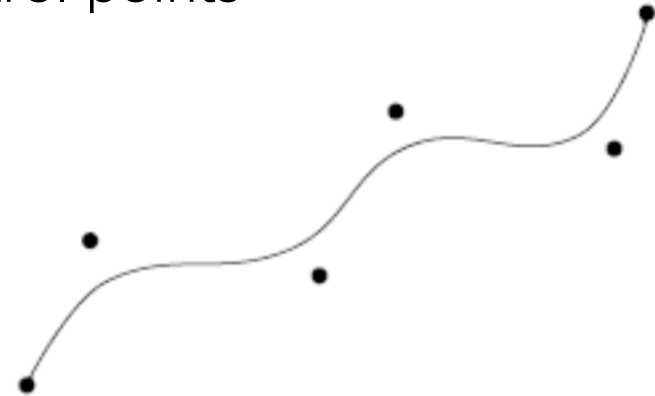
Recap

- **Bézier curves and curve design:**

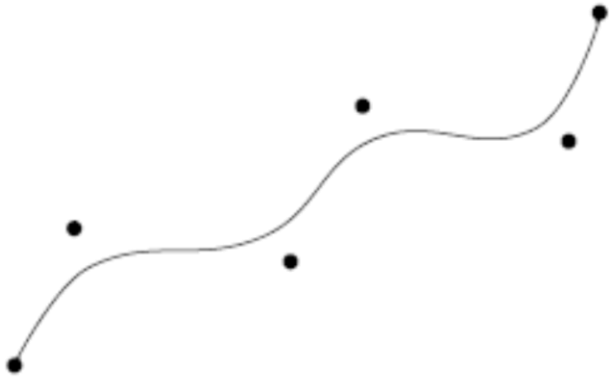
- The rough form is specified by the position of the control points
- Results: smooth curve approximating the control points
- Computation / Representation
 - de Casteljau algorithm
 - Bernstein form

- Problems:

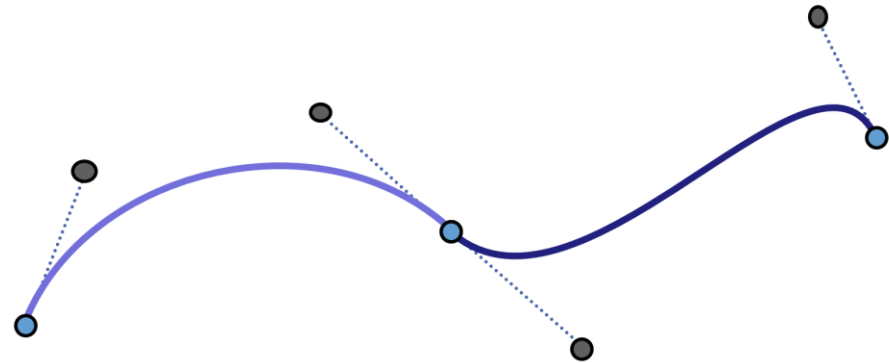
- High polynomial degree
- Moving a control point can change the whole curve
- Interpolation of points
- → **Bézier splines**



Recap



Approximation



Interpolation

Towards Bézier Splines

- Interpolation problems:

- given:

$$\mathbf{k}_0, \dots, \mathbf{k}_n \in \mathbb{R}^3 \quad \text{control points}$$

$$t_0, \dots, t_n \in \mathbb{R} \quad \text{knot sequence}$$

$$t_i < t_{i+1}, \text{ for } i = 0, \dots, n - 1$$

- wanted

- Interpolating curve $\mathbf{x}(i)$, i.e. $\mathbf{x}(t_i) = \mathbf{k}_i$ for $i = 0, \dots, n$

- Approach: “Joining” of n Bézier curves with certain intersection conditions

Towards Bézier Splines

- **The following issues arise when stitching together Bézier curves:**
 - Continuity
 - Parameterization
 - Degree

Bézier Splines

Parametric and Geometric Continuity

Parametric Continuity

Joining curves – continuity

- Given: 2 curves

$\mathbf{x}_1(t)$ over $[t_0, t_1]$

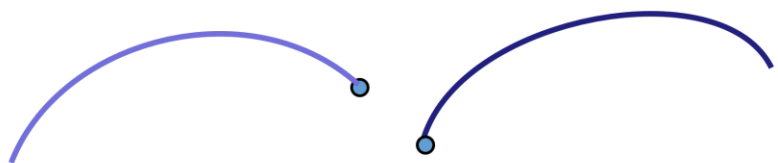
$\mathbf{x}_2(t)$ over $[t_1, t_2]$

- \mathbf{x}_1 and \mathbf{x}_2 are C^r continuous at t_1 , if all their 0^{th} to r^{th} derivative vectors coincides at t_1

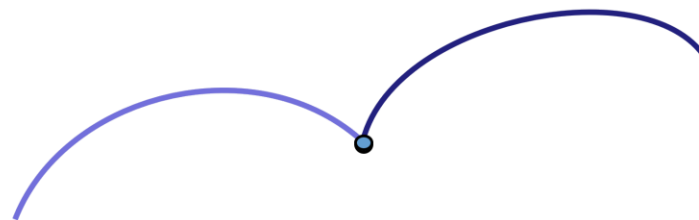
Parametric Continuity

- C^0 : position varies continuously
- C^1 : First derivative is continuous across junction
 - In other words: the velocity vector remains the same
- C^2 : Second derivative is continuous across junction
 - The acceleration vector remains the same

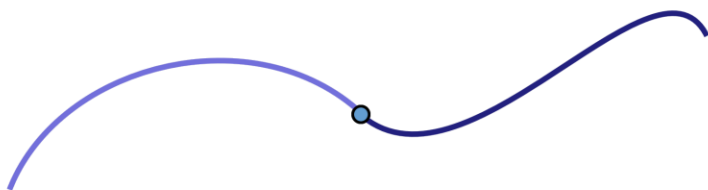
Parametric Continuity



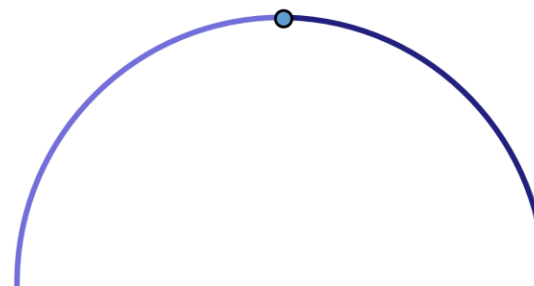
C^{-1} continuity



C^0 continuity



C^1 continuity



C^2 continuity

Continuity

Parametric Continuity C^r :

- C^0 , C^1 , C^2 ... continuity
- Does a particle moving on this curve have a smooth trajectory (position, velocity, acceleration, ...)?
- **Depends** on parameterization
- Useful for animation (object movement, camera paths)

Geometric Continuity G^r :

- Is the curve itself smooth?
- **Independent** of parameterization
- More relevant for modeling (curve design)

Geometric continuity:

Geometric continuity of curves

- Given: 2 curves
 $\mathbf{x}_1(t)$ over $[t_0, t_1]$
 $\mathbf{x}_2(t)$ over $[t_1, t_2]$
- \mathbf{x}_1 and \mathbf{x}_2 are G^r continuous in t_1 , if they can be reparameterized in such a way that they are C^r continuous in t_1

Geometric continuity:

- $G^0 = C^0$: position varies continuously (connected)
- G^1 : tangent direction varies continuously (same tangent)
 - In other words: the **normalized** tangent varies continuously
 - Equivalently: The curve can be reparameterized so that it becomes C^1
 - Also equivalent: A unit speed parameterization would be C^1
- G^2 : curvature varies continuously (same tangent and curvature)
 - Equivalently: The curve can be reparameterized so that it becomes C^2
 - Also equivalent: A unit speed parameterization would be C^2

$$\kappa = \|c''\|$$

Bézier Splines

Parameterization

Bézier spline curves

Local and global parameters:

- Given:
 - b_0, \dots, b_n
 - $y(u)$: Bézier curve in interval $[0,1]$
 - $x(t)$: Bézier curve in interval $[t_i, t_{i+1}]$
- Setting $u(t) = \frac{t-t_i}{t_{i+1}-t_i}$
- Results in $x(t) = y(u(t))$

The *local* parameter u runs from 0 to 1,
while the *global* parameter t runs from t_i to t_{i+1}

Bézier spline curves

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

$$x(t) = y(u(t))$$

Derivatives:

$$x'(t) = y'(u(t)) \cdot u'(t) = \frac{y'(u(t))}{t_{i+1} - t_i}$$

$$x''(t) = y''(u(t)) \cdot (u'(t))^2 + y'(u(t)) \cdot u''(t) = \frac{y''(u(t))}{(t_{i+1} - t_i)^2}$$

...

$$x^{[n]}(t) = \frac{y^{[n]}(u(t))}{(t_{i+1} - t_i)^n}$$

Bézier Curve

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

- Function value at $\{0,1\}$:

$$f(0) = \mathbf{p}_0$$

$$f(1) = \mathbf{p}_3$$

- First derivative vector at $\{0,1\}$

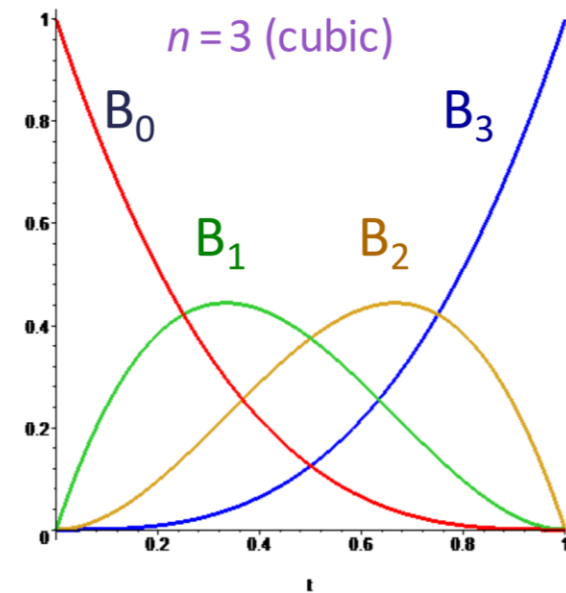
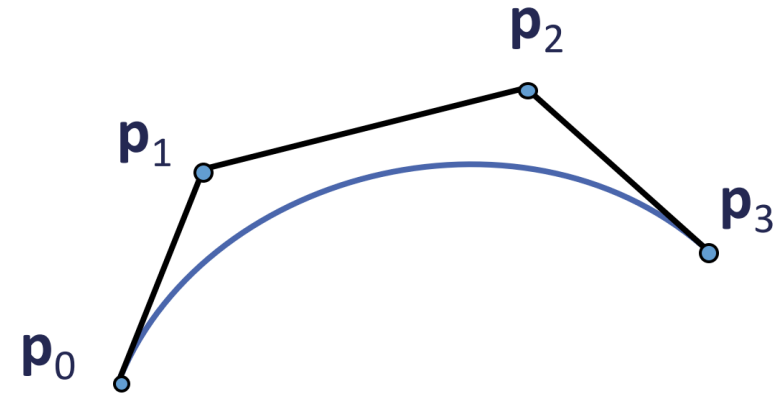
$$f'(0) = n[\mathbf{p}_1 - \mathbf{p}_0]$$

$$f'(1) = n[\mathbf{p}_n - \mathbf{p}_{n-1}]$$

- Second derivative vector at $\{0,1\}$

$$f''(0) = n(n-1)[\mathbf{p}_2 - 2\mathbf{p}_1 + \mathbf{p}_0]$$

$$f''(1) = n(n-1)[\mathbf{p}_n - 2\mathbf{p}_{n-1} + \mathbf{p}_{n-2}]$$



Bézier spline curves

Special cases:

$$\mathbf{x}'(t_i) = \frac{n \cdot (\mathbf{p}_1 - \mathbf{p}_0)}{t_{i+1} - t_i}$$

$$\mathbf{x}'(t_{i+1}) = \frac{n \cdot (\mathbf{p}_n - \mathbf{p}_{n-1})}{t_{i+1} - t_i}$$

$$\mathbf{x}''(t_i) = \frac{n \cdot (n-1) \cdot (\mathbf{p}_2 - 2\mathbf{p}_1 + \mathbf{p}_0)}{(t_{i+1} - t_i)^2}$$

$$\mathbf{x}''(t_{i+1}) = \frac{n \cdot (n-1) \cdot (\mathbf{p}_n - 2\mathbf{p}_{n-1} + \mathbf{p}_{n-2})}{(t_{i+1} - t_i)^2}$$

Bézier Splines

General Case

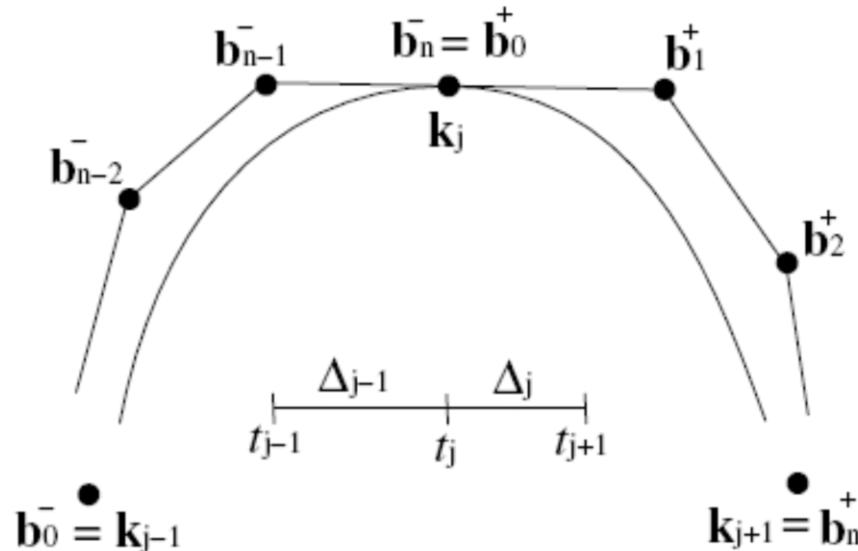
Bézier spline curves

Joining Bézier curves:

- Given: 2 Bézier curves of **degree n** through

$$k_{j-1} = b_0^-, b_1^-, \dots, b_n^- = k_j$$

$$k_j = b_0^+, b_1^+, \dots, b_n^+ = k_{j+1}$$

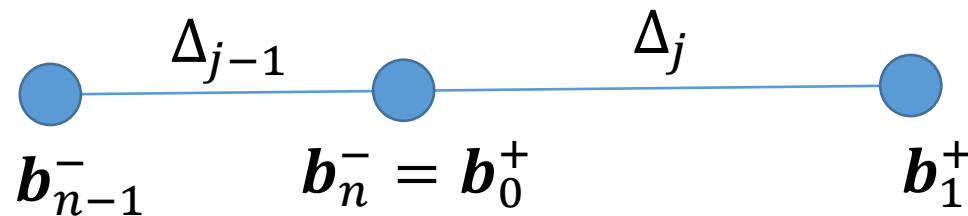


$$\mathbf{x}'(t_i) = \frac{n \cdot (\mathbf{b}_1 - \mathbf{b}_0)}{t_{i+1} - t_i}$$

Bézier spline curves

- Required: C^1 -continuity at \mathbf{k}_j :
- $\mathbf{b}_{n-1}^-, \mathbf{k}_j, \mathbf{b}_1^+$ collinear and

$$\frac{\mathbf{b}_n^- - \mathbf{b}_{n-1}^-}{t_j - t_{j-1}} = \frac{\mathbf{b}_1^+ - \mathbf{b}_0^+}{t_{j+1} - t_j}$$



Bézier spline curves

- Required: G^1 -continuity at \mathbf{k}_j :
 - \mathbf{b}_{n-1}^- , \mathbf{k}_j , \mathbf{b}_1^+ collinear
- Less restrictive than C^1 -continuity

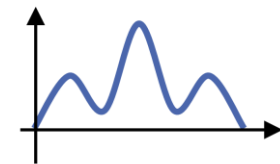
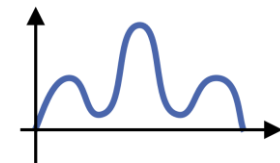
Bézier Splines

Choosing the degree

Choosing the Degree

Candidates:

- $d = 0$ (piecewise constant) : not smooth
- $d = 1$ (piecewise linear) : not smooth enough
- $d = 2$ (piecewise quadratic) : constant 2nd derivative, still too inflexible
- $d = 3$ (piecewise cubic): degree of choice for computer graphics applications



Cubic Splines

Cubic piecewise polynomials:

- We can attain C^2 continuity without fixing the second derivative throughout the curve

Cubic Splines

Cubic piecewise polynomials:

- We can attain C^2 continuity without fixing the second derivative throughout the curve
- C^2 continuity is perceptually important
 - Motion: continuous position, velocity & acceleration
Discontinuous acceleration noticeable (object/camera motion)
 - We can see second order shading discontinuities
(esp.: reflective objects)

Cubic Splines

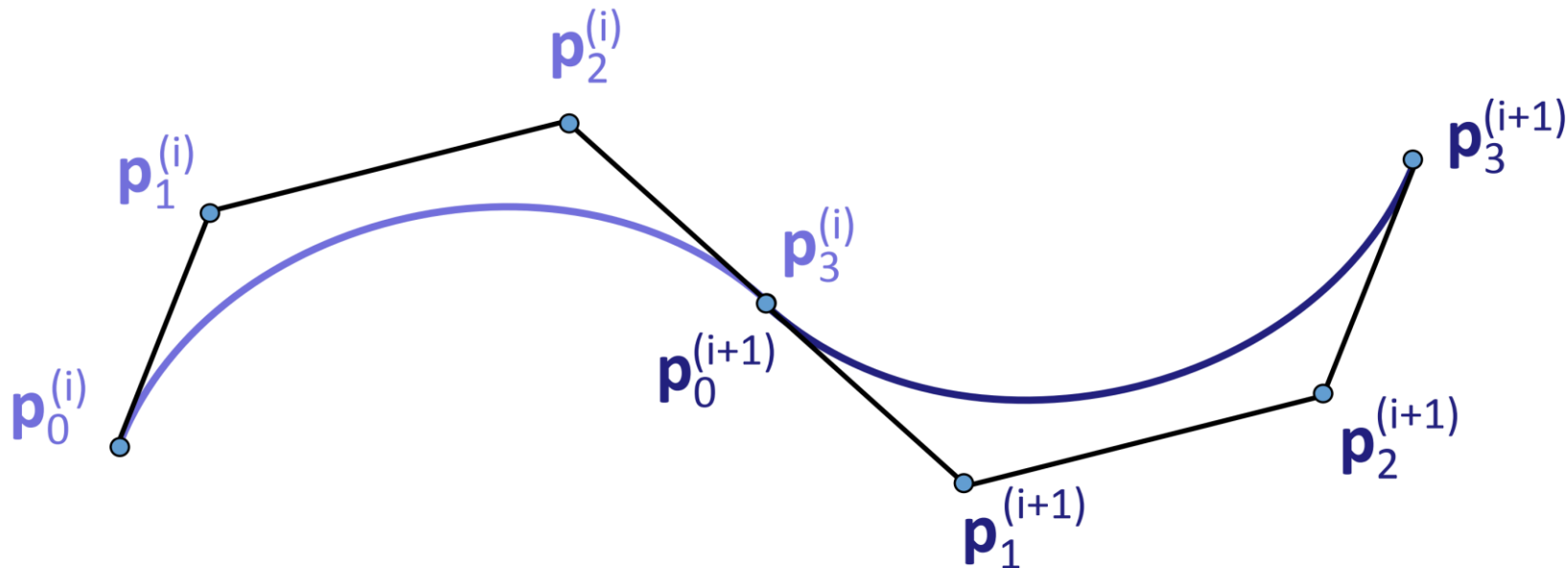
Cubic piecewise polynomials

- We can attain C^2 continuity without fixing the second derivative throughout the curve
- C^2 continuity is perceptually important
 - We can see second order shading discontinuities (esp.: reflective objects)
 - Motion: continuous *position, velocity & acceleration*
Discontinuous acceleration noticeable (object/camera motion)
- One more argument for cubics:
 - Among all C^2 curves that interpolate a set of points (and obey to the same end condition), a piecewise cubic curve has the least integral acceleration (“smoothest curve you can get”).

Bézier Splines

Local control: Bézier splines

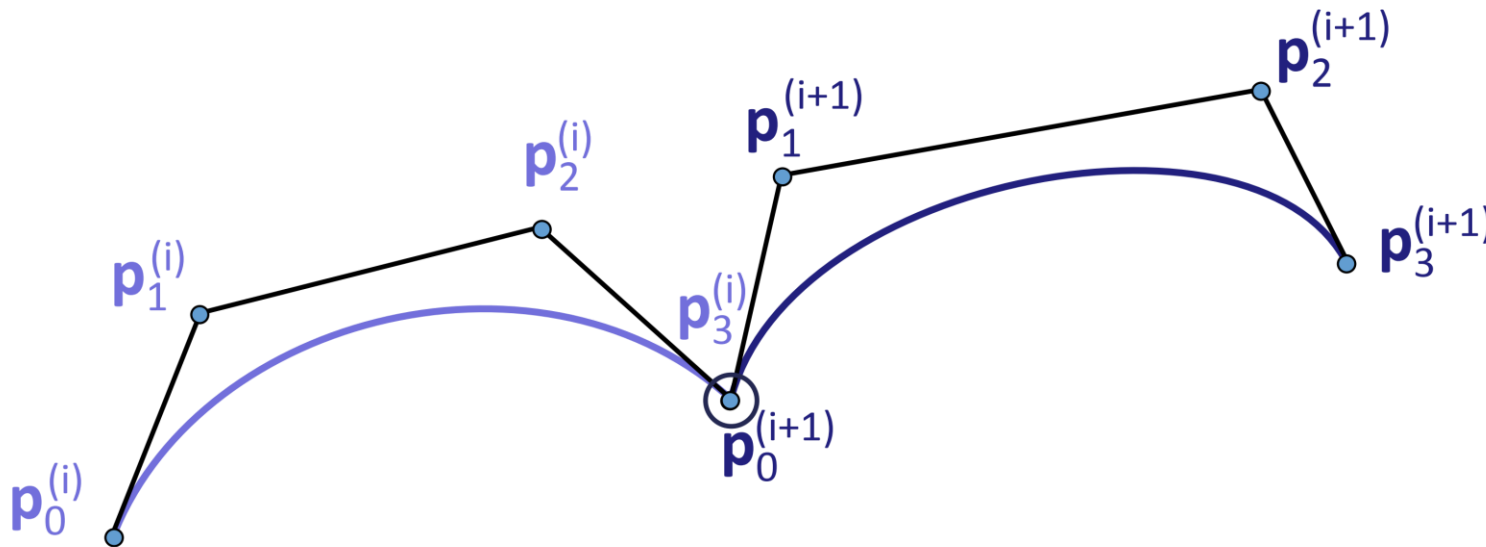
- Concatenate several curve segments
- Question: Which constraints to place upon the control points in order to get C^{-1} , C^0 , C^1 , C^2 continuity?



Bézier Spline Continuity

Rules for Bézier spline continuity:

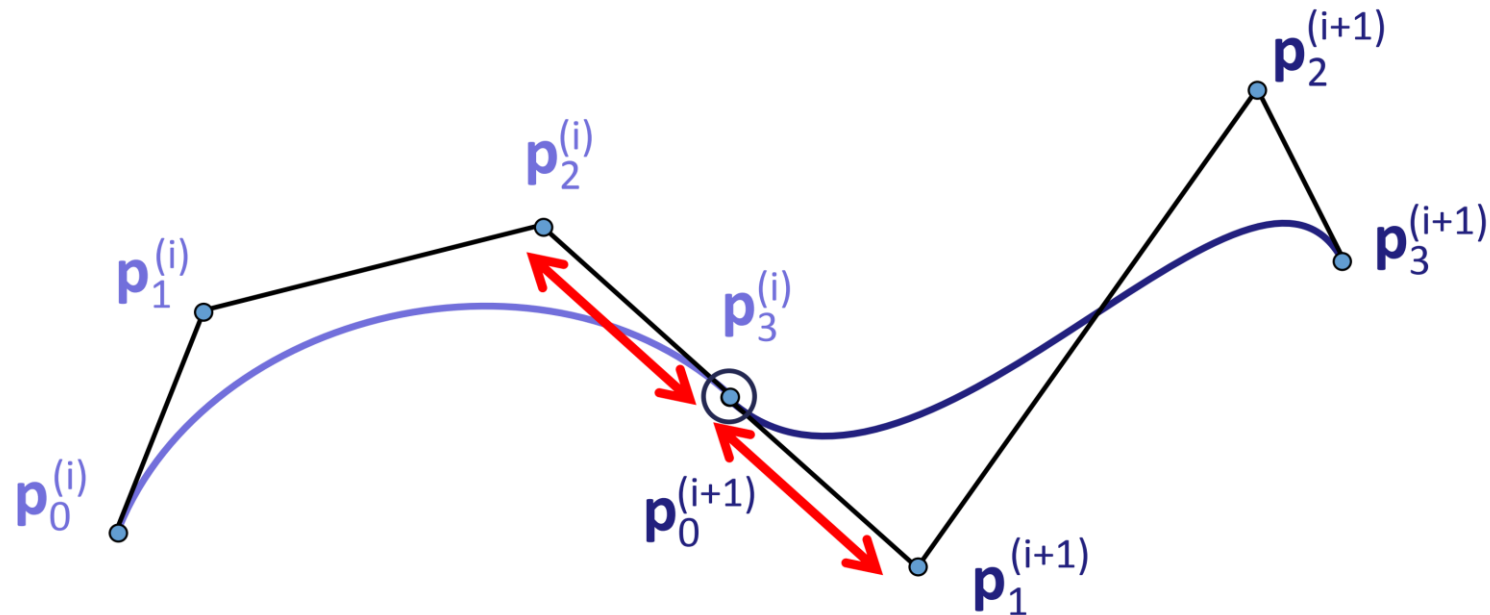
- C^0 continuity:
 - Each spline segment interpolates the first and last control point
 - Therefore: Points of neighboring segments have to coincide for C^0 continuity



Bézier Spline Continuity

Rules for Bézier spline continuity:

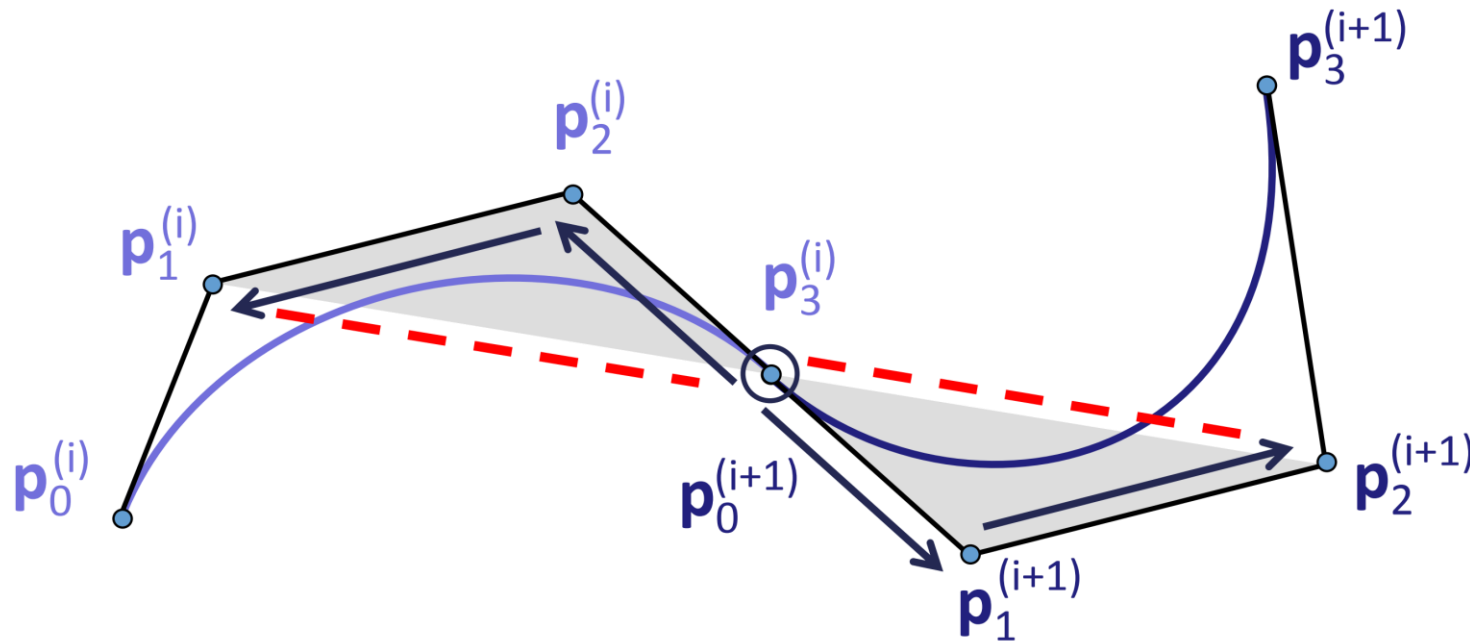
- Additional requirement for C^1 continuity:
 - Tangent vectors are proportional to differences $\mathbf{p}_1 - \mathbf{p}_0$, $\mathbf{p}_n - \mathbf{p}_{n-1}$
 - Therefore: These vectors must be **identical** for C^1 continuity



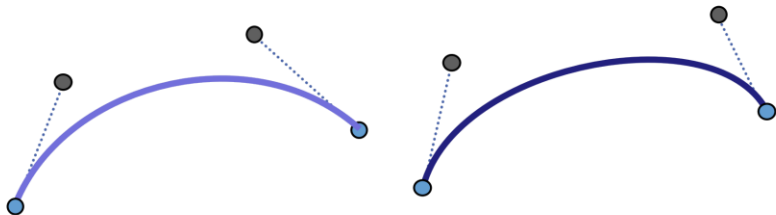
Bézier Spline Continuity

Rules for Bézier spline continuity

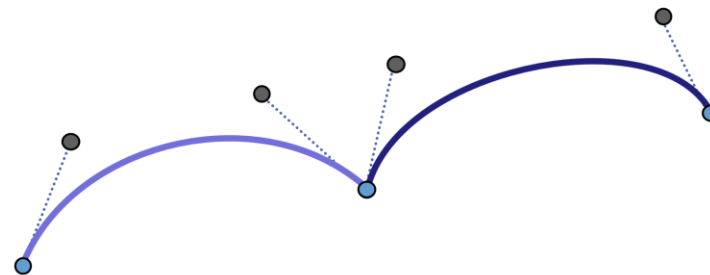
- Additional requirement for C^2 continuity:
 - d^2/dt^2 vectors are prop. to $\mathbf{p}_2 - 2\mathbf{p}_1 + \mathbf{p}_0$, $\mathbf{p}_n - 2\mathbf{p}_{n-1} + \mathbf{p}_{n-2}$
 - Tangents must be the same (C^2 implies C^1)



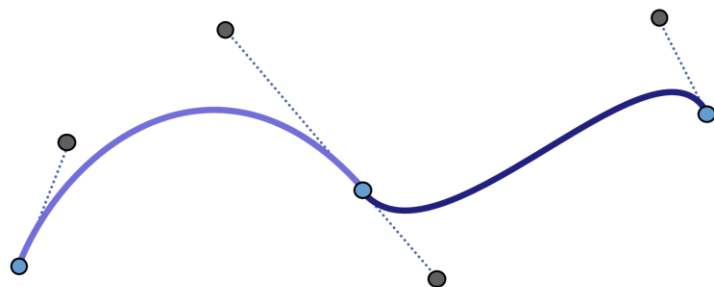
Continuity



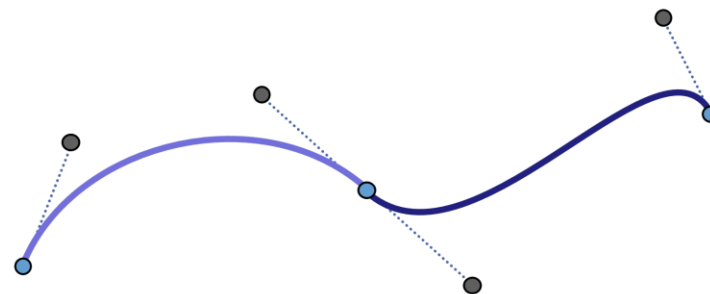
C^{-1} continuity



C^0 continuity



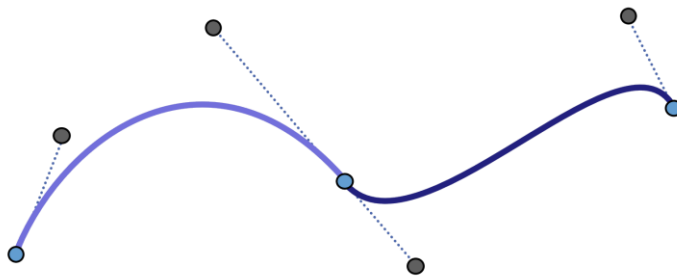
G^1 continuity



C^1 continuity

Continuity for Bézier Splines

This means



G^1 continuity

This Bézier curve is G^1 : It can be reparameterized to become C^1 .
(Just increase the speed for the second segment by ratio of tangent vector lengths)

In Practice

- Everyone is using cubic Bézier curves
- Higher degree are rarely used (some CAD/CAM applications)
- Typically: “points & handles” interface
- Four modes:
 - Discontinuous (two curves)
 - C^0 Continuous (points meet)
 - G^1 continuous: Tangent direction continuous
 - Handles point into the same direction, but different length
 - C^1 continuous
 - Handle points have symmetric vectors
- C^2 is more restrictive: control via k_i

Bézier spline curves

- Required: C^2 -continuity at \mathbf{k}_j

- C^1 implies
$$\frac{\mathbf{b}_n^- - \mathbf{b}_{n-1}^-}{t_j - t_{j-1}} = \frac{\mathbf{b}_1^+ - \mathbf{b}_0^+}{t_{j+1} - t_j}$$

- C^2 implies
$$\frac{\mathbf{b}_n^- - 2\mathbf{b}_{n-1}^- + \mathbf{b}_{n-2}^-}{(t_j - t_{j-1})^2} = \frac{\mathbf{b}_2^+ - 2\mathbf{b}_1^+ + \mathbf{b}_0^+}{(t_{j+1} - t_j)^2}$$

Bézier spline curves

$$\frac{t_{j+1} - t_j}{t_j - t_{j-1}} = \frac{\Delta_j}{\Delta_{j-1}}$$

- Required: C^2 -continuity at \mathbf{k}_j :

- Introduce $\mathbf{d}^- = \mathbf{b}_{n-1}^- + \frac{\Delta_j}{\Delta_{j-1}} (\mathbf{b}_{n-1}^- - \mathbf{b}_{n-2}^-)$

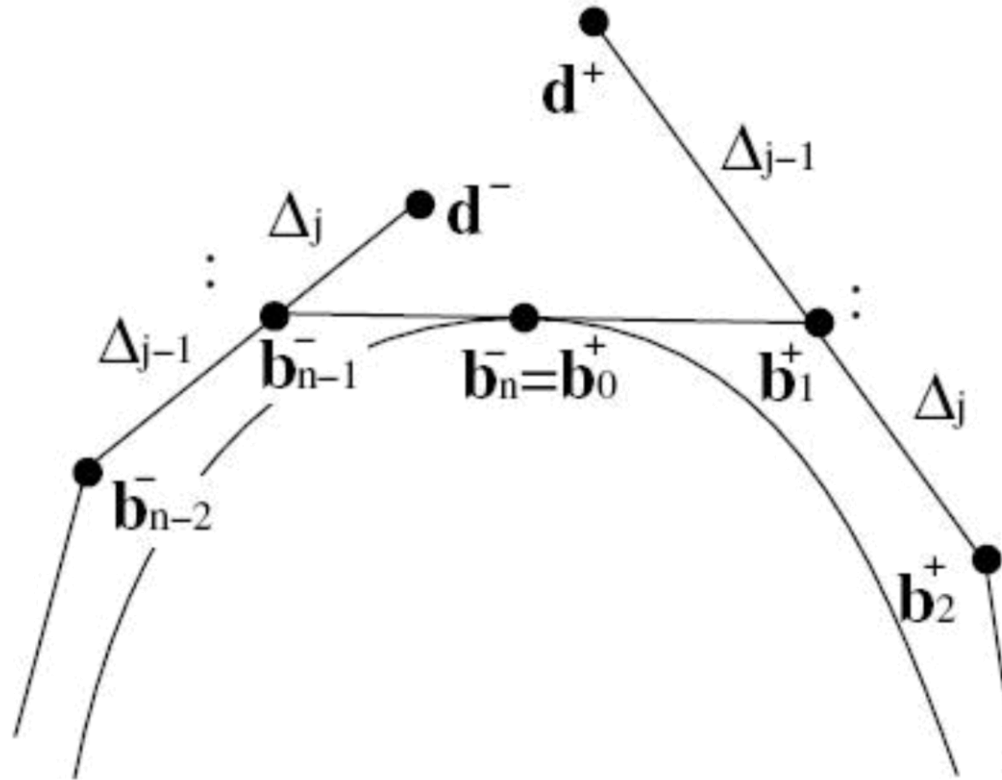
and $\mathbf{d}^+ = \mathbf{b}_1^+ - \frac{\Delta_{j-1}}{\Delta_j} (\mathbf{b}_2^+ - \mathbf{b}_1^+)$

$$\frac{\mathbf{b}_n^- - 2\mathbf{b}_{n-1}^- + \mathbf{b}_{n-2}^-}{(t_j - t_{j-1})^2} = \frac{\mathbf{b}_2^+ - 2\mathbf{b}_1^+ + \mathbf{b}_0^+}{(t_{j+1} - t_j)^2}$$

- By manipulating equation from the previous slides
- C^2 -continuity $\Leftrightarrow C^1$ -continuity and $\mathbf{d}^- = \mathbf{d}^+$

Bézier spline curves

C^2 -continuity $\Leftrightarrow C^1$ -continuity and $\mathbf{d}^- = \mathbf{d}^+$



$$\mathbf{d}^- = \mathbf{b}_{n-1}^- + \frac{\Delta_j}{\Delta_{j-1}} (\mathbf{b}_{n-1}^- - \mathbf{b}_{n-2}^-)$$

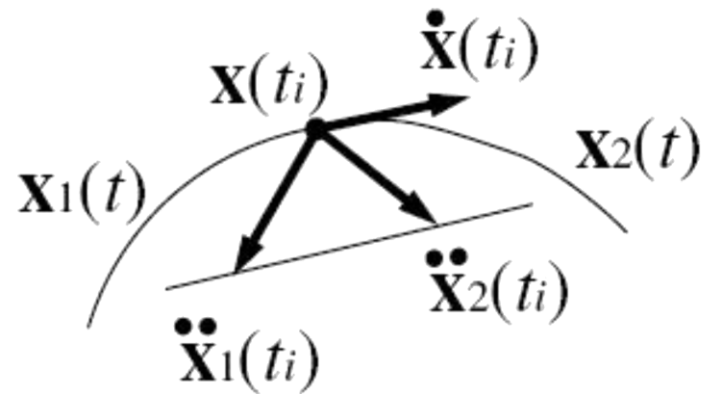
$$\mathbf{d}^+ = \mathbf{b}_1^+ - \frac{\Delta_{j-1}}{\Delta_j} (\mathbf{b}_2^+ - \mathbf{b}_1^+)$$

Bézier spline curves

- G^2 -continuity in general (for all types of curves):
- Given:
 - $\mathbf{x}_1(t), \mathbf{x}_2(t)$ with
 - $\mathbf{x}_1(t_i) = \mathbf{x}_2(t_i) = \mathbf{x}(t_i)$
 - $\mathbf{x}'_1(t_i) = \mathbf{x}'_2(t_i) = \mathbf{x}'(t_i)$
- Then the requirement for G^2 -continuity at $t = t_i$:

$$\mathbf{x}''_2(t_i) - \mathbf{x}''_1(t_i) \parallel \mathbf{x}'(t_i)$$

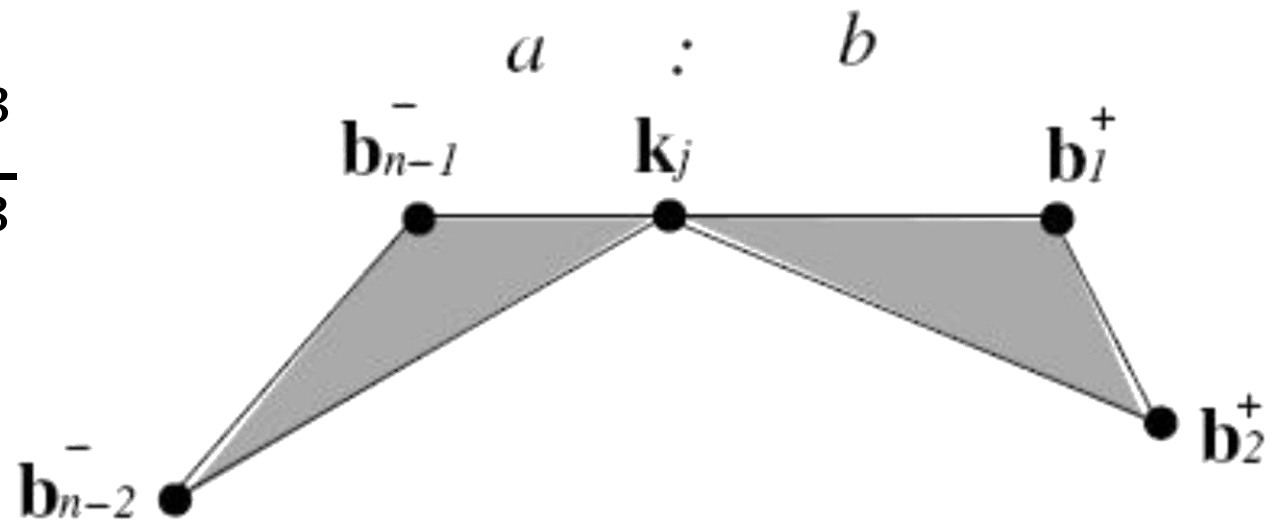
Parallel



Bézier spline curves

- Required: G^2 -continuity at k_j :
- G^1 -continuity
- Co-planarity for : $\mathbf{b}_{n-2}^-, \mathbf{b}_{n-1}^-, \mathbf{k}_j, \mathbf{b}_1^+, \mathbf{b}_2^+$

- And:
$$\frac{\text{area}(\mathbf{b}_{n-2}^-, \mathbf{b}_{n-1}^-, \mathbf{k}_j)}{\text{area}(\mathbf{k}_j, \mathbf{b}_1^+, \mathbf{b}_2^+)} = \frac{a^3}{b^3}$$



Bézier Splines

C^2 Cubic Bézier Splines

Cubic Bézier Splines

Cubic Bézier spline curves

- Given:

$\mathbf{k}_0, \dots, \mathbf{k}_n \in \mathbb{R}^3$ control points

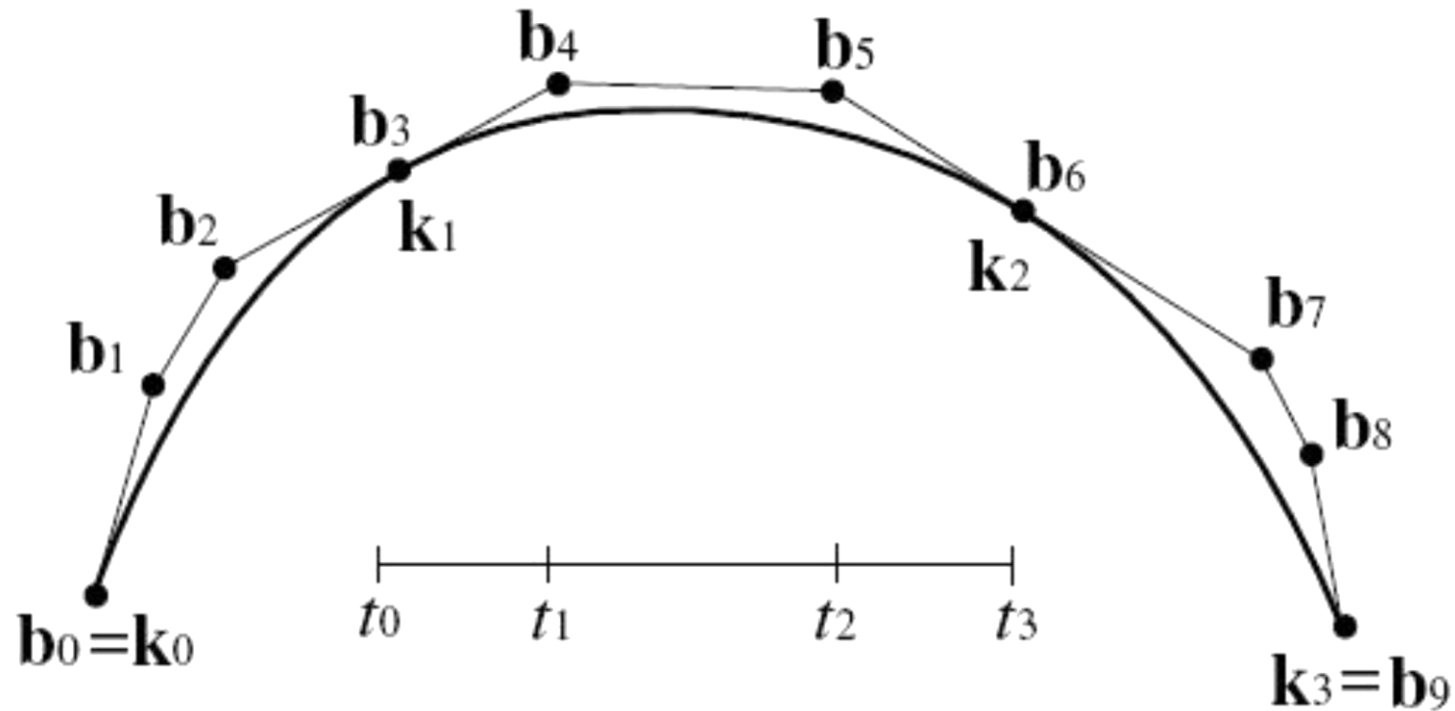
$t_0, \dots, t_n \in \mathbb{R}$ knot sequence

$t_i < t_{i+1}$, for $i = 0, \dots, n_1$

- Wanted: Bézier points $\mathbf{b}_0, \dots, \mathbf{b}_{3n}$ for an interpolating \mathcal{C}^2 -continuous piecewise cubic Bézier spline curve

Cubic Bézier Splines

Examples: $n = 3$:

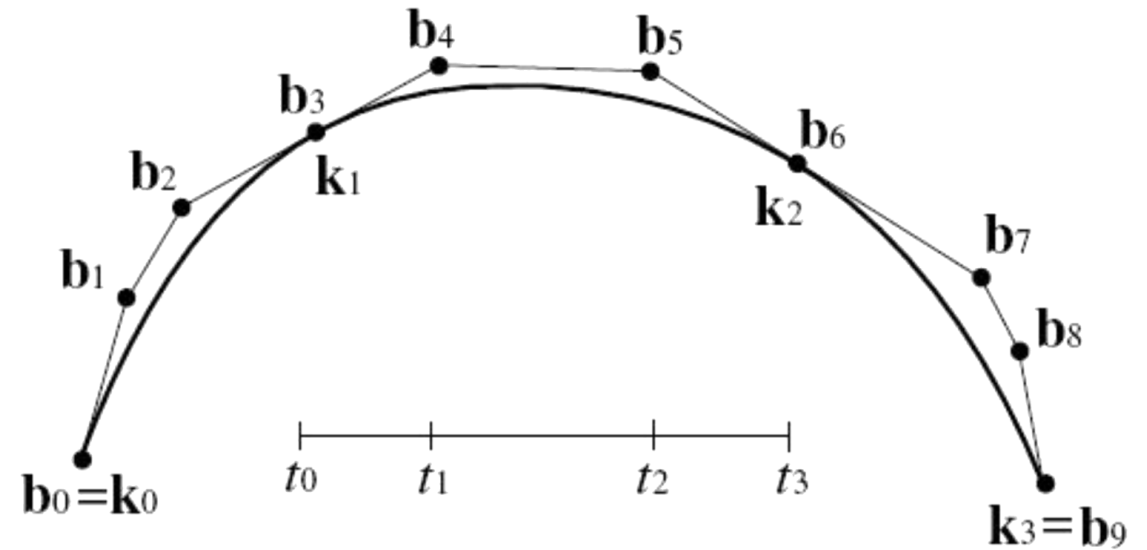


Cubic Bézier Splines

- $3n + 1$ unknown points
 - $b_{3i} = k_i$ for $i = 0, \dots, n$
 $n + 1$ equations
 - C^1 in points k_i for $i = 1, \dots, n - 1$
 $n - 1$ equations
 - C^2 in points k_i for $i = 1, \dots, n - 1$
 $n - 1$ equations
-

$3n - 1$ equations

⇒ **2 additional conditions necessary: end conditions**



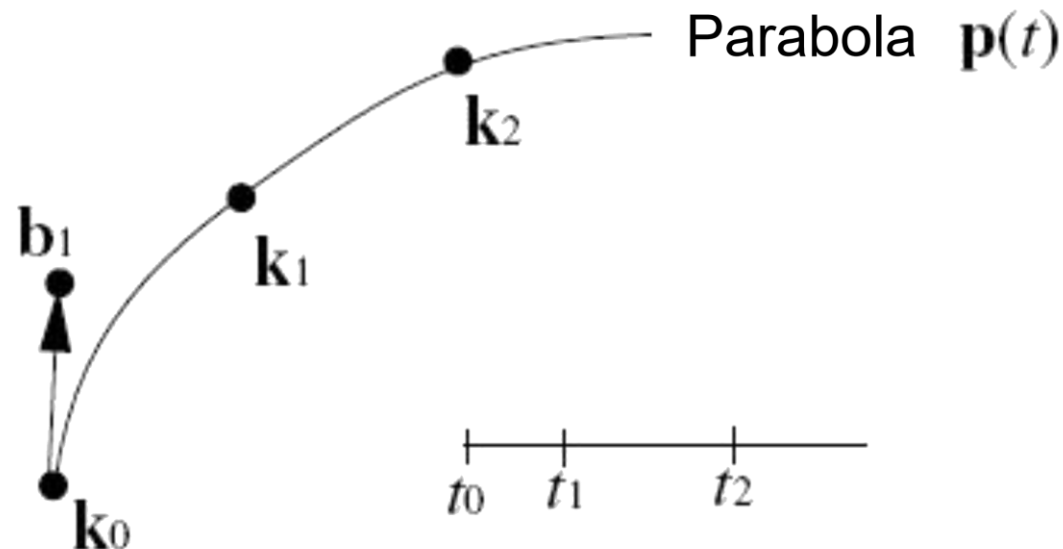
Bézier Splines

C^2 Cubic Bézier Splines: End conditions

Bézier spline curves: End conditions

Bessel's end condition

- The tangential vector in \mathbf{k}_0 is equivalent to the tangential vector of the parabola interpolating $\{\mathbf{k}_0, \mathbf{k}_1, \mathbf{k}_2\}$ at \mathbf{k}_0 :



$$\dot{\mathbf{x}}(t_i) = \frac{n \cdot (\mathbf{b}_1 - \mathbf{b}_0)}{t_{i+1} - t_i}$$

Bézier spline curves: End conditions

Parabola Interpolating $\{\mathbf{k}_0, \mathbf{k}_1, \mathbf{k}_2\}$

$$\mathbf{p}(t) = \frac{(t_2 - t)(t_1 - t)}{(t_2 - t_0)(t_1 - t_0)} \mathbf{k}_0 + \frac{(t_2 - t)(t - t_0)}{(t_2 - t_1)(t_1 - t_0)} \mathbf{k}_1 + \frac{(t_0 - t)(t_1 - t)}{(t_2 - t_1)(t_2 - t_0)} \mathbf{k}_2$$

Its derivative

$$\mathbf{p}'(t_0) = -\frac{(t_2 - t_0) + (t_1 - t_0)}{(t_2 - t_0)(t_1 - t_0)} \mathbf{k}_0 + \frac{(t_2 - t_0)}{(t_2 - t_1)(t_1 - t_0)} \mathbf{k}_1 - \frac{(t_1 - t_0)}{(t_2 - t_1)(t_2 - t_0)} \mathbf{k}_2$$

Location of \mathbf{b}_1

$$\mathbf{b}_1 = \mathbf{b}_0 + \frac{t_1 - t_0}{3} \mathbf{p}'(t_0)$$

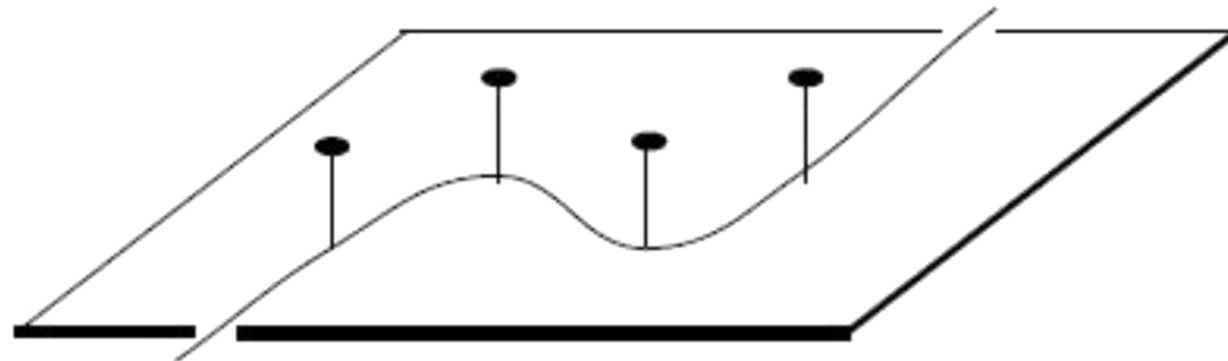
$$\ddot{\mathbf{x}}(t_i) = \frac{n \cdot (n-1) \cdot (\mathbf{b}_2 - 2\mathbf{b}_1 + \mathbf{b}_0)}{(t_{i+1} - t_i)^2}$$

Bézier spline curves: End conditions

- Natural end condition:

$$\mathbf{x}''(t_0) = 0 \Leftrightarrow \mathbf{b}_1 = \frac{\mathbf{b}_2 + \mathbf{b}_0}{2}$$

$$\mathbf{x}''(t_n) = 0 \Leftrightarrow \mathbf{b}_{3n-1} = \frac{\mathbf{b}_{3n-2} + \mathbf{b}_{3n}}{2}$$



End conditions: Examples

- *Besse/* end condition



Curve: circle of radius 1



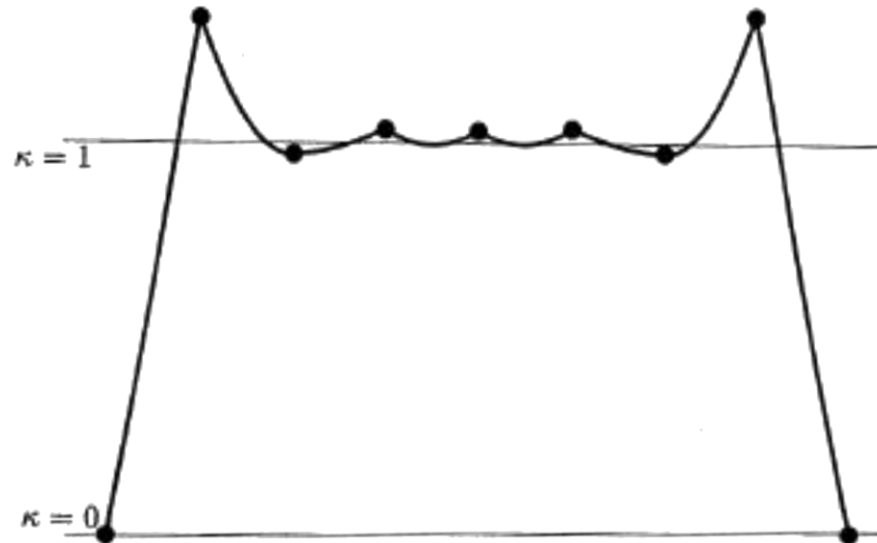
Curvature plot

End conditions: Examples

- *Natural* end condition



Curve: circle of radius 1



Curvature plot

Bézier Splines

C^2 Cubic Bézier Splines: parameterization

Bézier spline curves: Parameterization

Approach so far:

- Given: control points $\mathbf{k}_0, \dots, \mathbf{k}_n$ and knot sequence $t_0 < \dots < t_n$
- Wanted: interpolating curve
- Problem: Normally, the knot sequence is not given, but it influences the curve

Bézier spline curves: Parameterization

- **Equidistant (uniform) parameterization**

- $t_{i+1} - t_i = \text{const}$
- e.g. $t_i = i$
- Geometry of the data points is not considered

- **Chordal parameterization**

- $t_{i+1} - t_i = \|\mathbf{k}_{i+1} - \mathbf{k}_i\|$
- Parameter intervals proportional to the distances of neighbored control points

Bézier spline curves: Parameterization

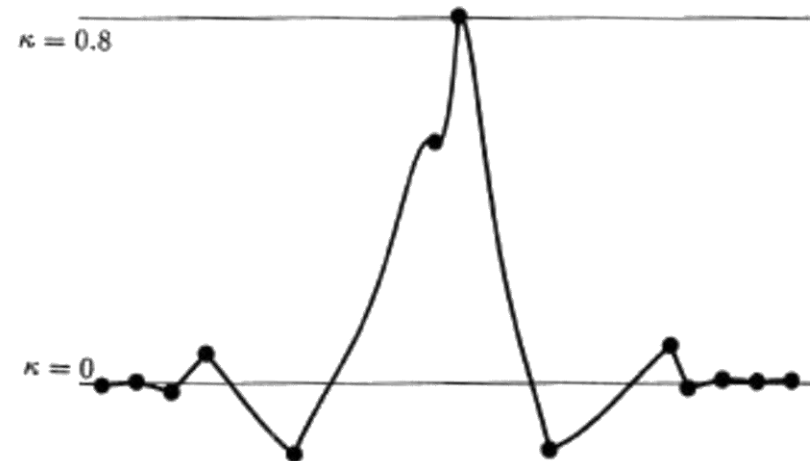
- Centripetal parameterization
 - $t_{i+1} - t_i = \sqrt{\|\mathbf{k}_{i+1} - \mathbf{k}_i\|}$
- Foley parameterization
 - Involvement of angles in the control polygon
 - $$t_{i+1} - t_i = \|\mathbf{k}_{i+1} - \mathbf{k}_i\| \cdot \left(1 + \frac{3}{2} \frac{\hat{\alpha}_i \|\mathbf{k}_i - \mathbf{k}_{i-1}\|}{\|\mathbf{k}_i - \mathbf{k}_{i-1}\| + \|\mathbf{k}_{i+1} - \mathbf{k}_i\|} + \frac{3}{2} \frac{\hat{\alpha}_{i+1} \|\mathbf{k}_{i+1} - \mathbf{k}_i\|}{\|\mathbf{k}_{i+1} - \mathbf{k}_i\| + \|\mathbf{k}_{i+2} - \mathbf{k}_{i+1}\|} \right)$$
 - with $\hat{\alpha}_i = \min\left(\pi - \alpha_i, \frac{\pi}{2}\right)$
 - and $\alpha_i = \text{angle}(\mathbf{k}_{i-1}, \mathbf{k}_i, \mathbf{k}_{i+1})$
- Affine invariant parameterization
 - Parameterization on the basis of an affine invariant distance measure (e.g. G. Nielson)

Bézier spline curves: Parameterization

- Examples: Chordal parameterization



Curve



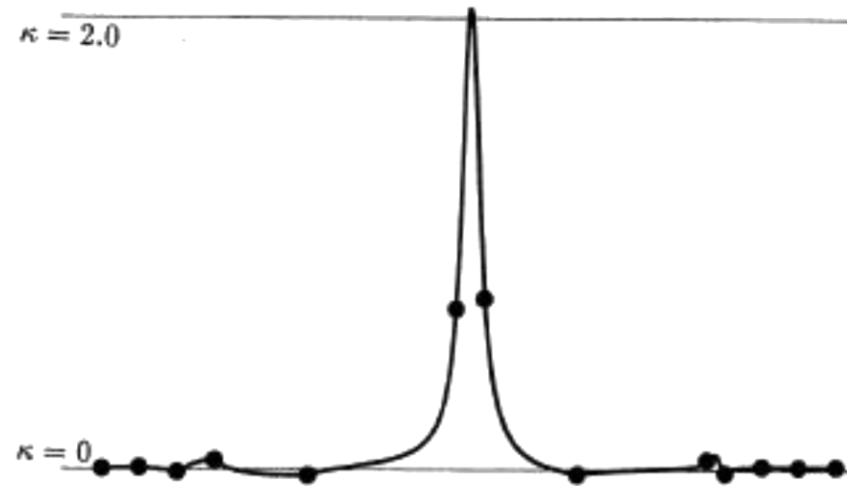
Curvature plot

Bézier spline curves: Parameterization

- Examples: Centripetal parameterization



Curve



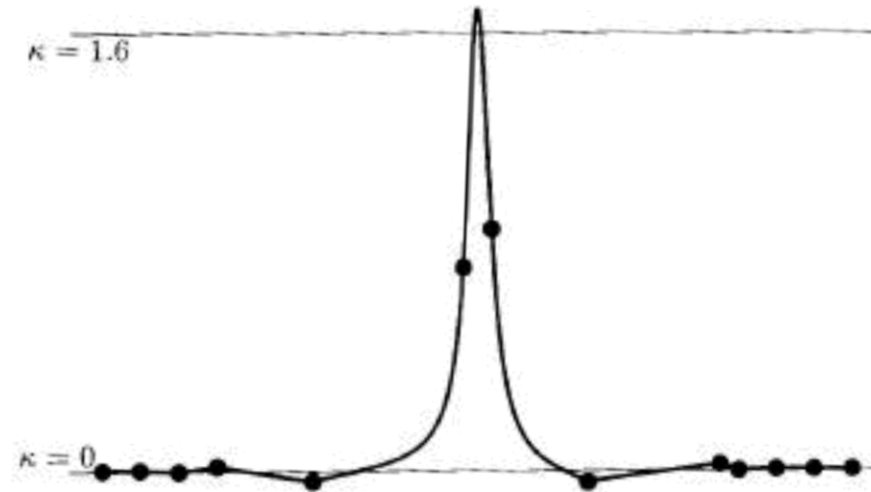
Curvature plot

Bézier spline curves: Parameterization

- Examples: Foley parameterization



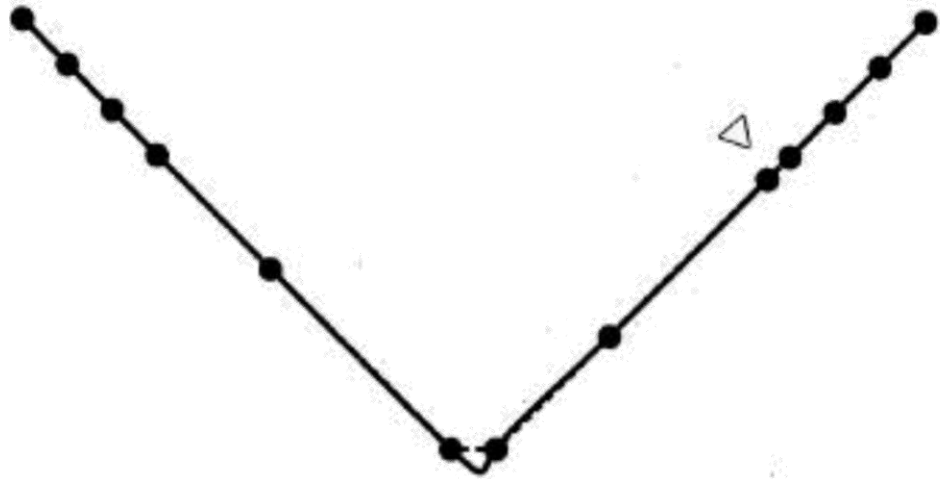
Curve



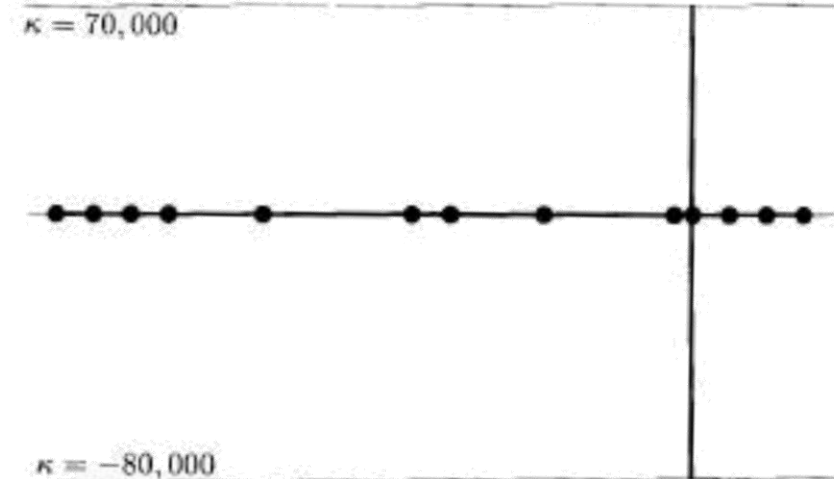
Curvature plot

Bézier spline curves: Parameterization

- Examples: Uniform parameterization



Curve



Curvature plot

Bézier Splines

C^2 Cubic Bézier Splines: closed curves

Closed cubic Bézier spline curves

Closed cubic Bézier spline curves

- Given:

$\mathbf{k}_0, \dots, \mathbf{k}_{n-1}, \mathbf{k}_n = \mathbf{k}_0$: control points

$t_0 < \dots < t_n$: knot sequence

- As an “end condition” for the piecewise cubic curve we place:

$$\mathbf{x}'(t_0) = \mathbf{x}'(t_n)$$

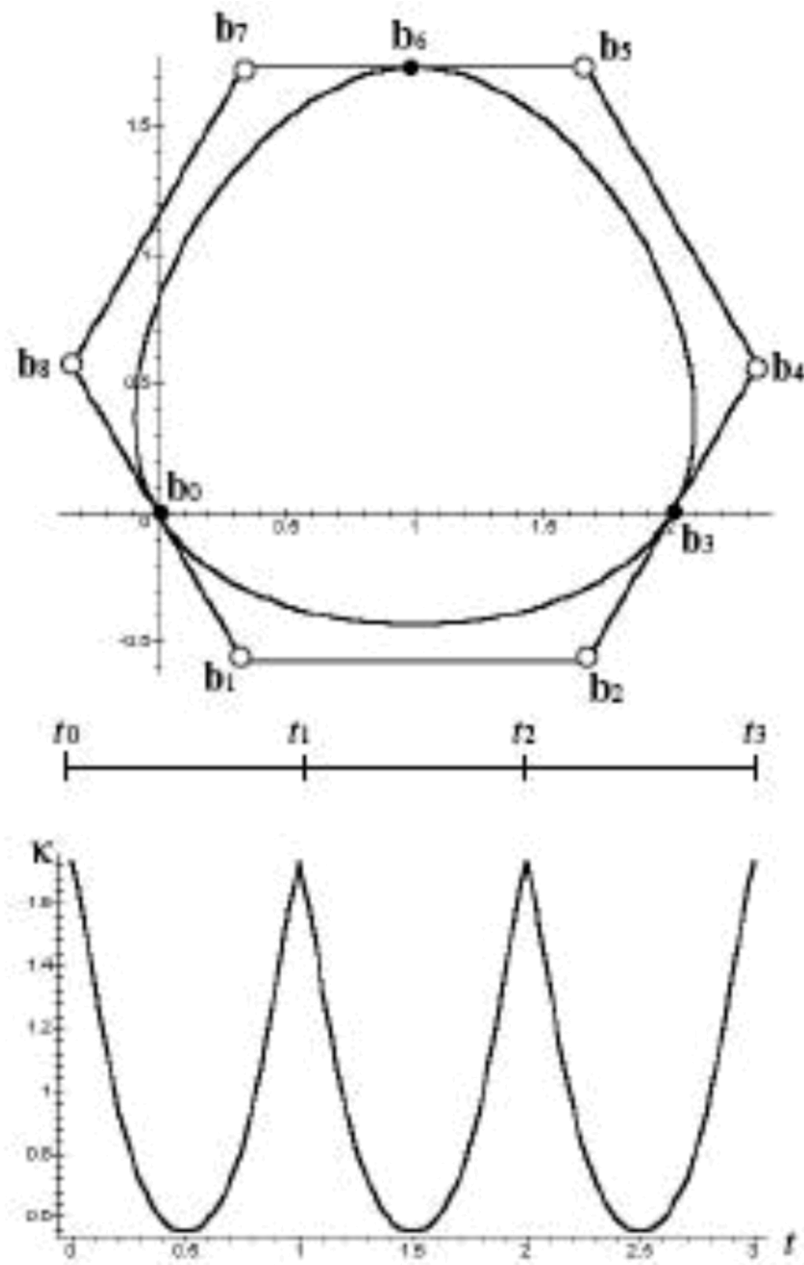
$$\mathbf{x}''(t_0) = \mathbf{x}''(t_n)$$

Closed cubic Bézier spline curves

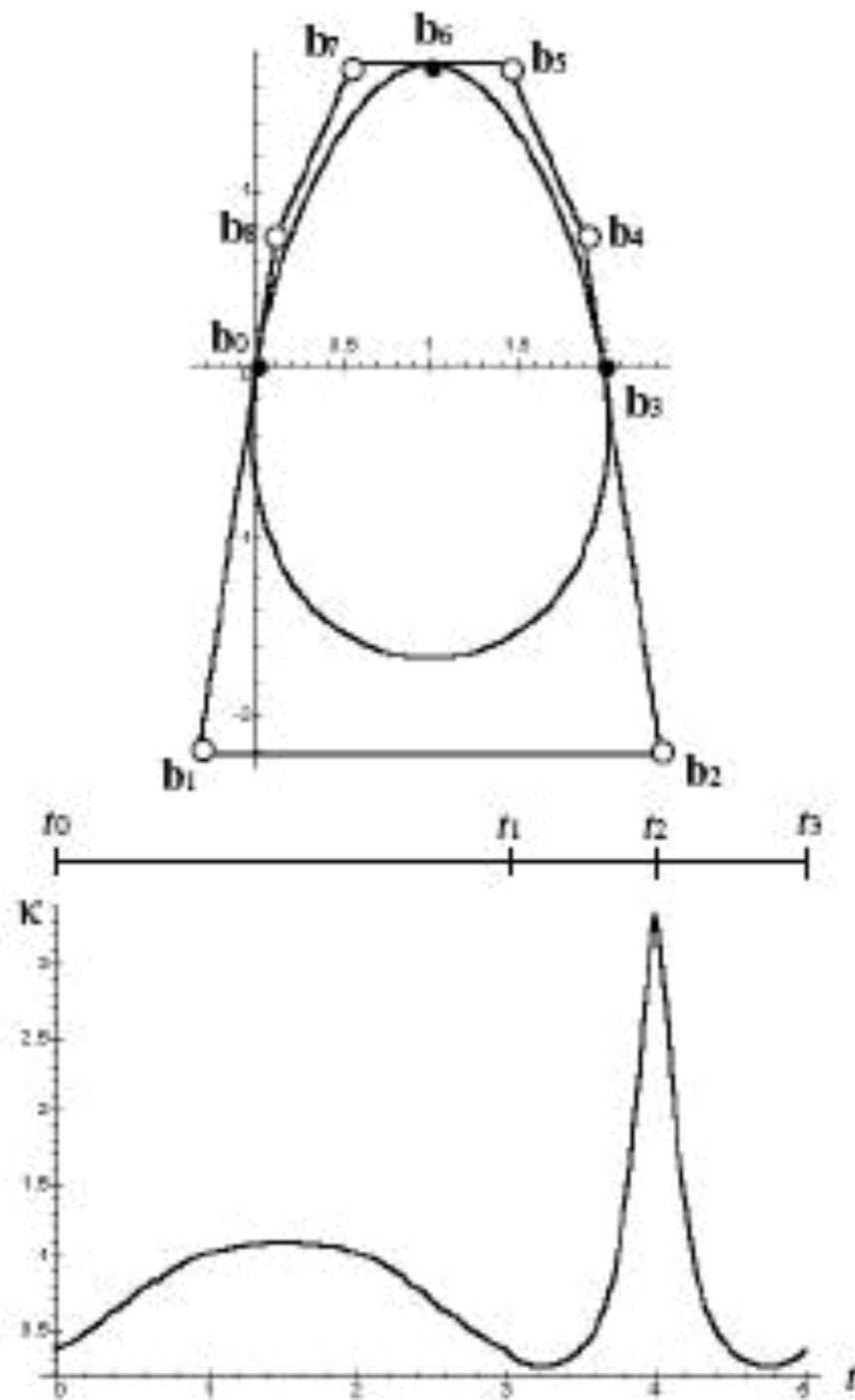
Closed cubic Bézier spline curves

- $\rightarrow \mathcal{C}^2$ -continuous and closed curve
- Advantage of closed curves: selection of the end condition is not necessary!
- Examples (on the next 3 slides): $n = 3$

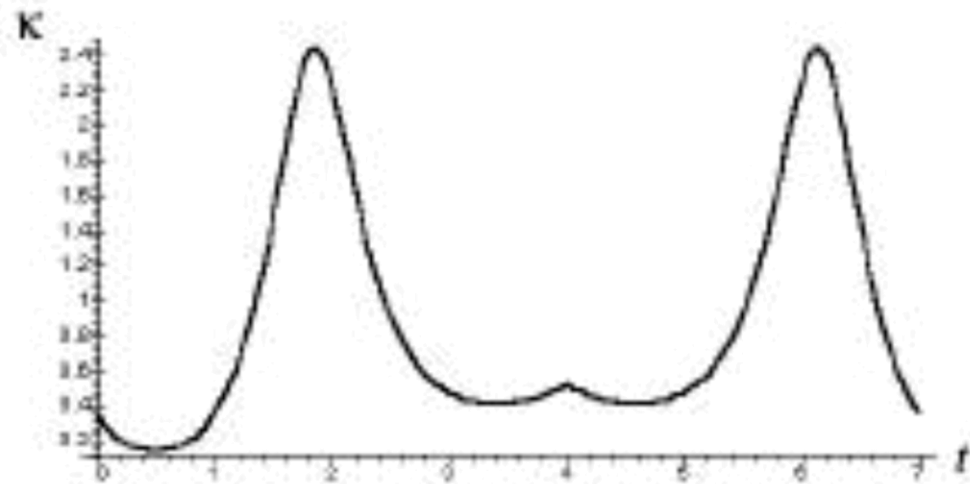
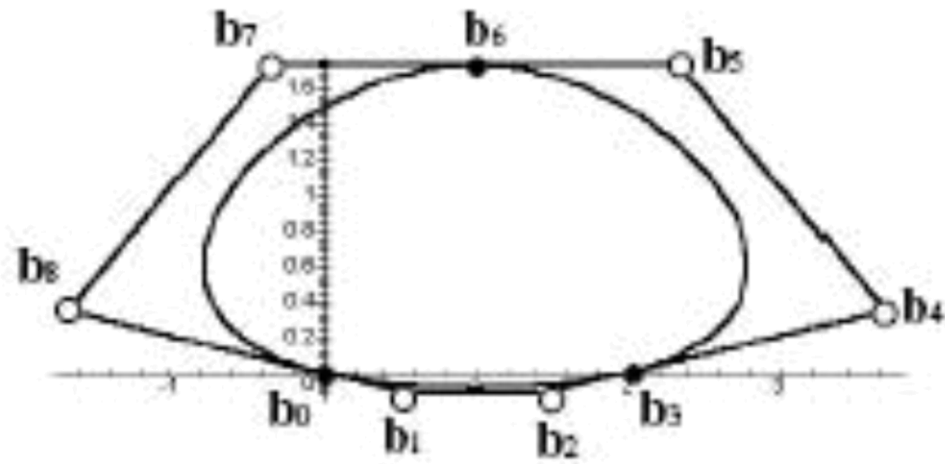
Examples



Examples



Examples



Computer Aided Geometric Design

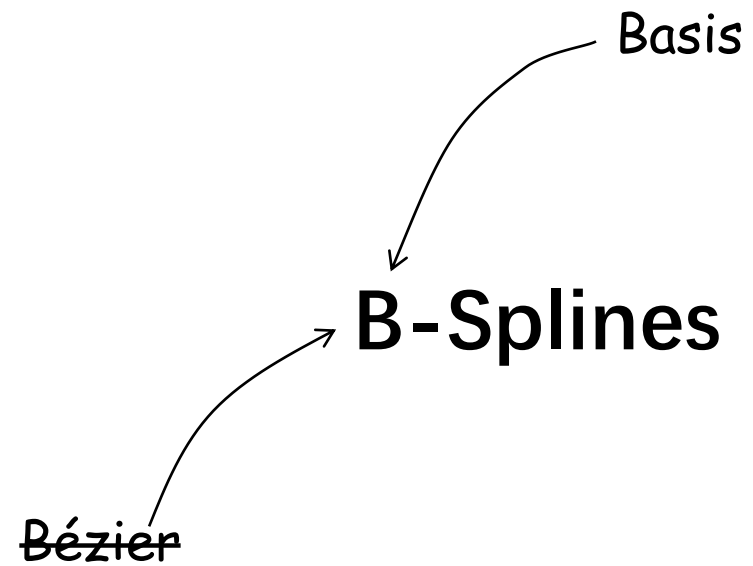
Fall Semester 2024

B-Splines

陈仁杰

renjiec@ustc.edu.cn

<http://staff.ustc.edu.cn/~renjiec>



Mathematical view: spline functions

Graphics view: spline curves (created using spline functions)

Motivation

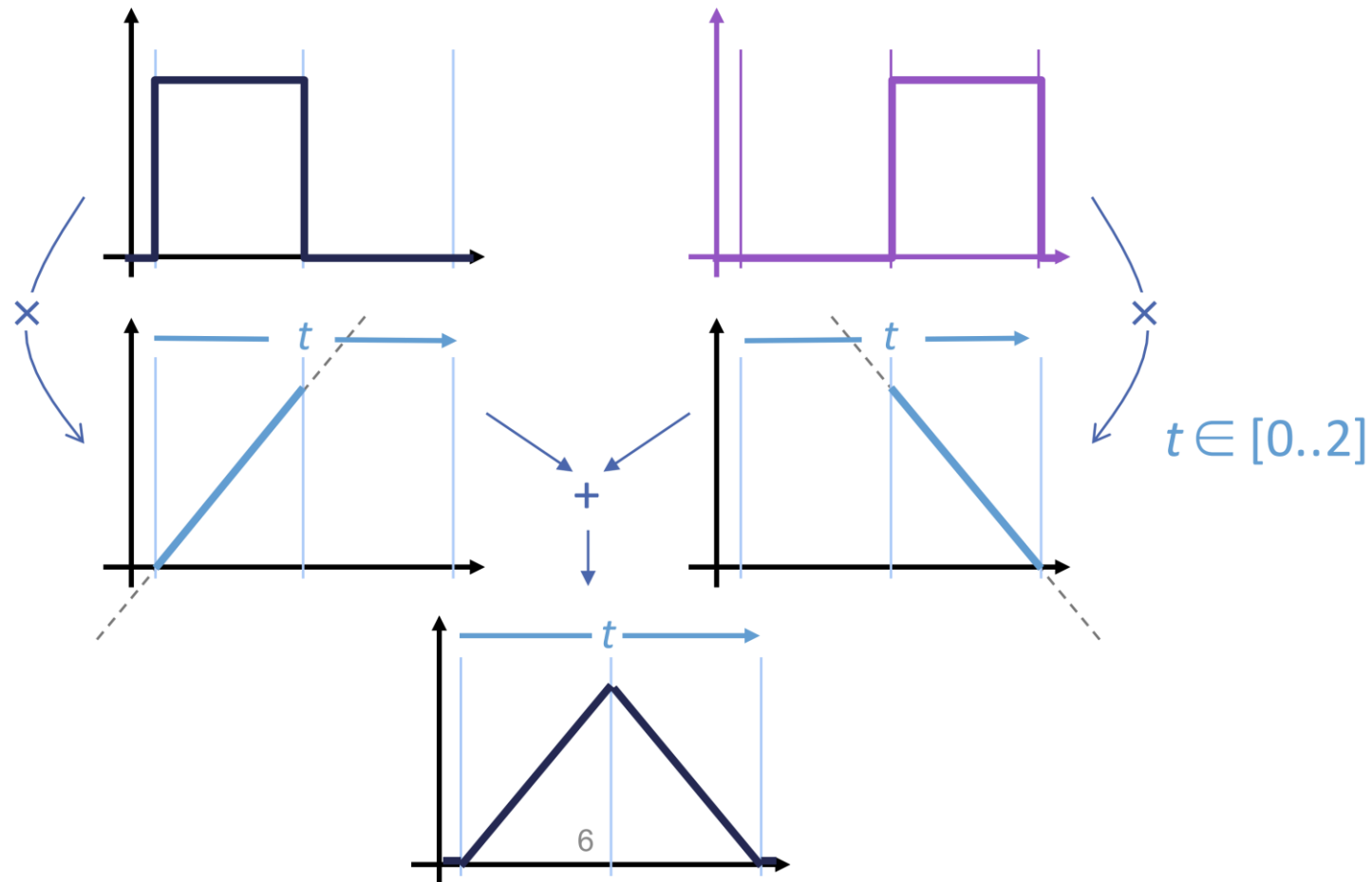
- Back to the algebraic approach for Bézier curves
→ Bernstein polynomials
- Problem: global influence of the Bézier points
- Introduction of new basis function
→ B-spline functions

Some history

- **Early use of splines on computers for data interpolation**
 - Ferguson at Boeing, 1963
 - Gordon and de Boor at General Motors
 - B-splines, de Boor 1972
- **Free form curve design**
 - Gordon and Riesenfeld, 1974 → B-splines as a generalization of Bézier curves

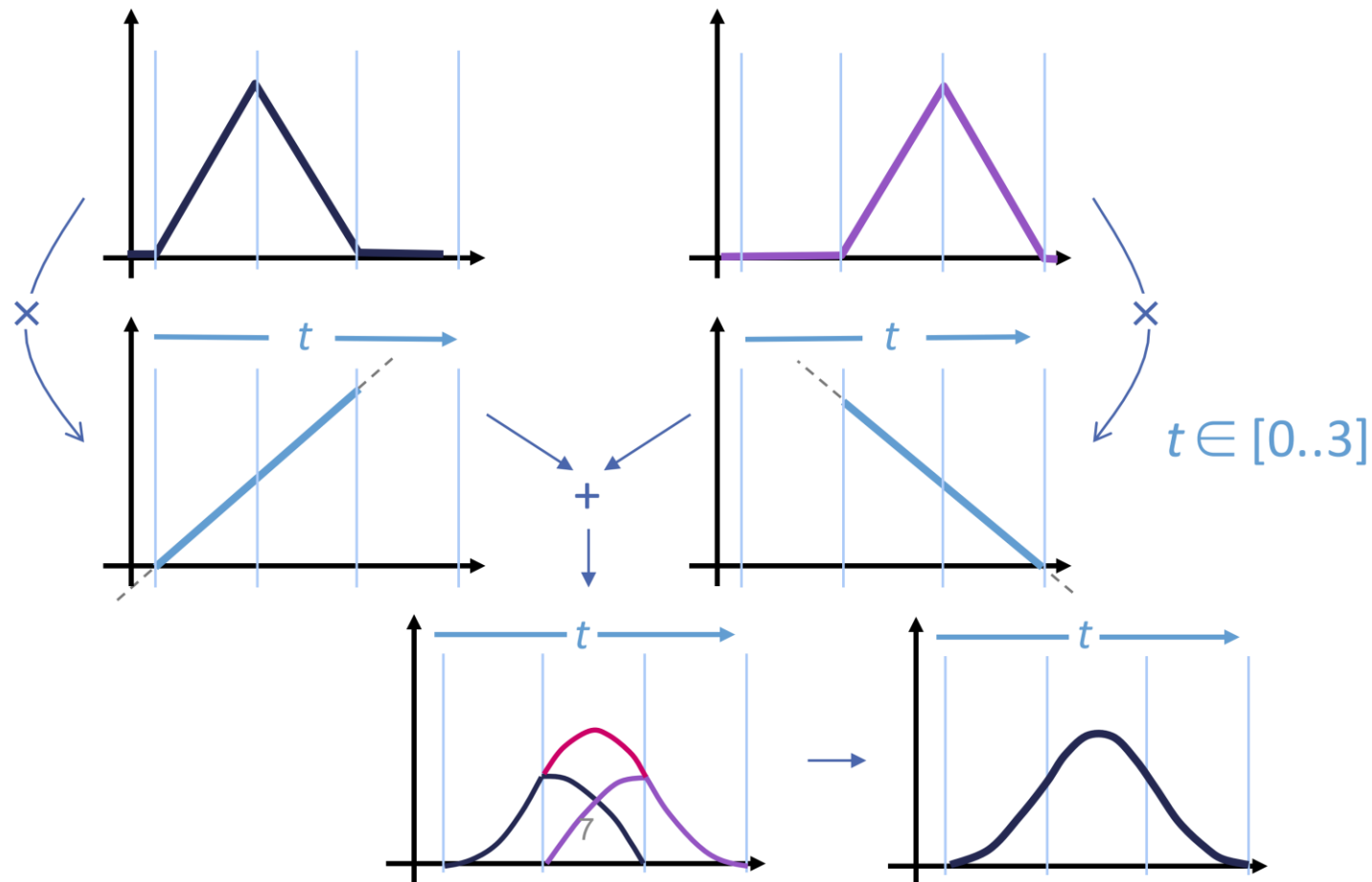
Repeated linear interpolation

Another way to increase smoothness:



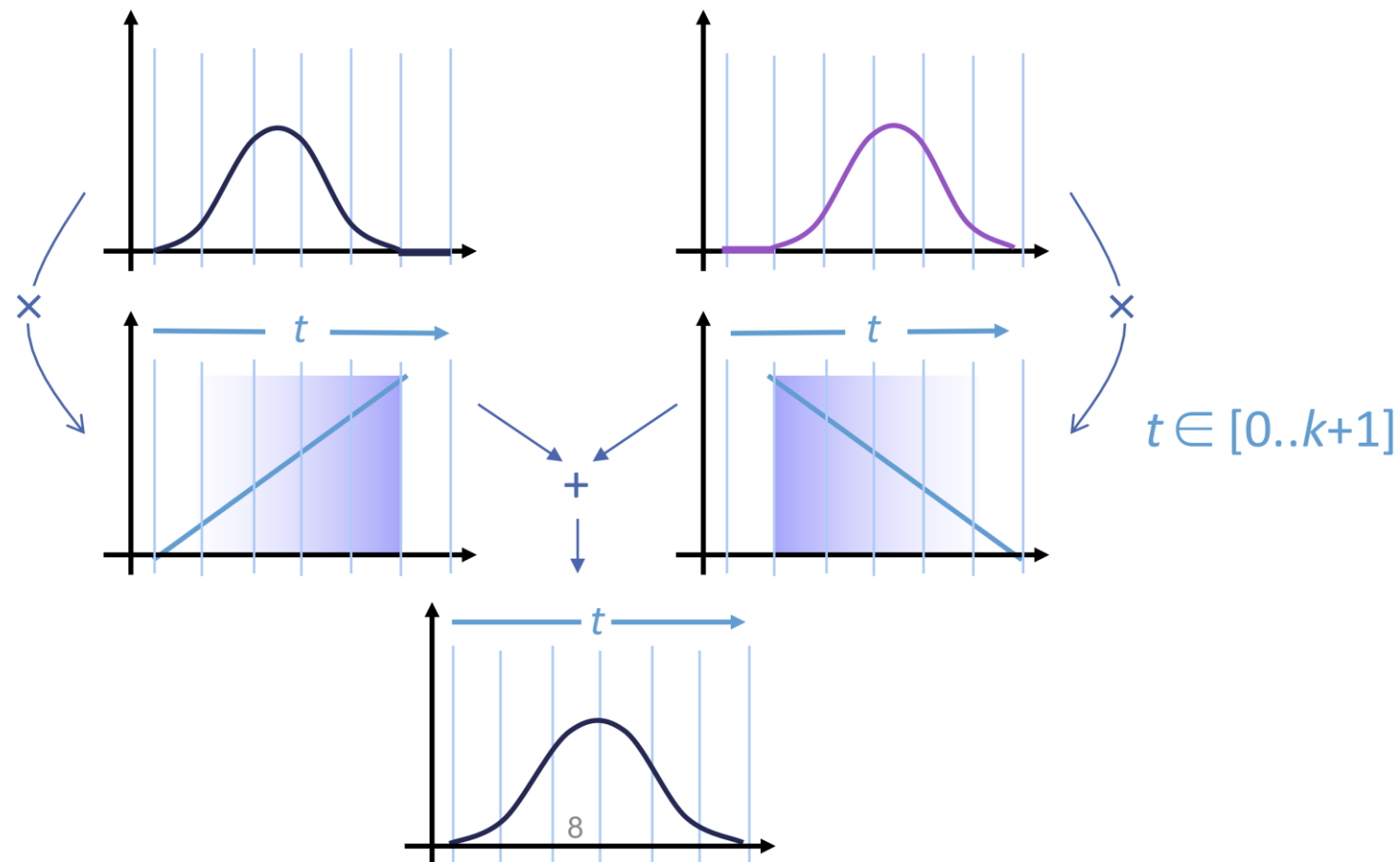
Repeated linear interpolation

- Another way to increase smoothness:



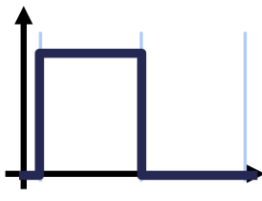
Repeated linear interpolation

- Another way to increase smoothness

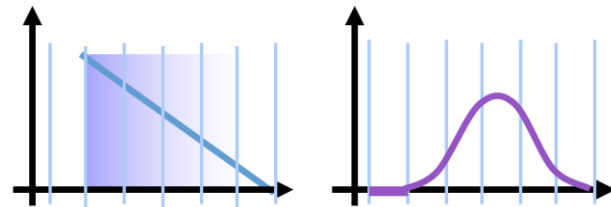
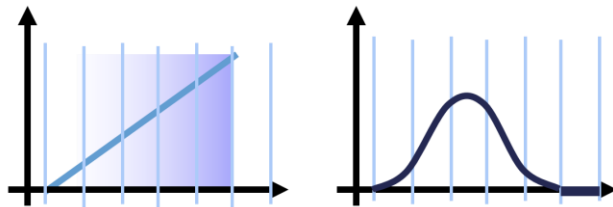


De Boor Recursion: uniform case

- The **uniform** B-spline basis of order k (degree $k - 1$) is given as

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


$$N_i^k(t) = \frac{t-i}{(i+k-1)-i} N_i^{k-1}(t) + \frac{(i+k)-t}{(i+k)-(i+1)} N_{i+1}^{k-1}(t)$$



$$= \frac{t-i}{k-1} N_i^{k-1}(t) + \frac{i+k-t}{k-1} N_{i+1}^{k-1}(t)$$

B-spline curves: general case

- Given: knot sequence $t_0 < t_1 < \dots < t_n < \dots < t_{n+k}$
($(t_0, t_1, \dots, t_{n+k})$ is called knot vector)
- Normalized B-spline functions $N_{i,k}$ of the order k (degree $k - 1$) are defined as:

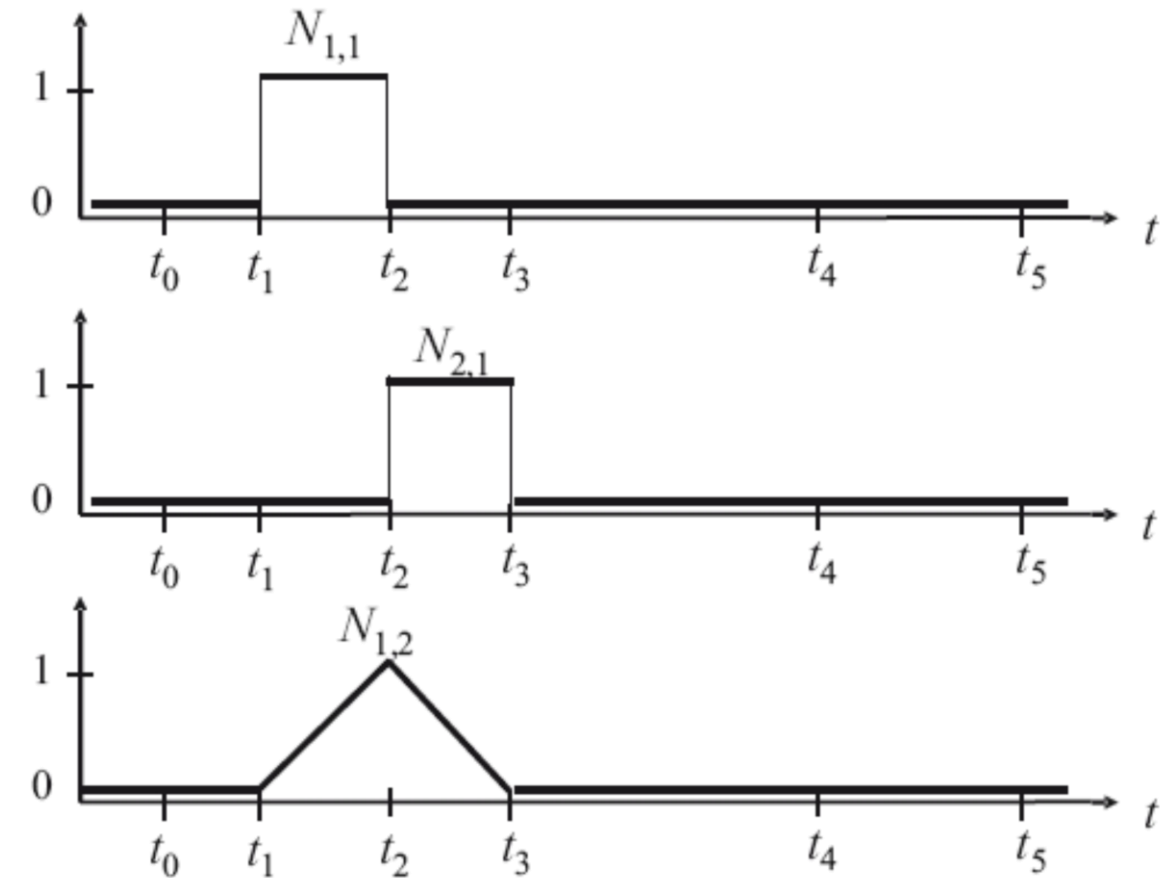
$$N_{i,1}(t) = \begin{cases} 1, & t_i \leq t < t_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

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

for $k > 1$ and $i = 0, \dots, n$

- **Remark:**
 - If a knot value is repeated k times, the denominator may vanish
 - In this case: The fraction is treated as a zero

Example

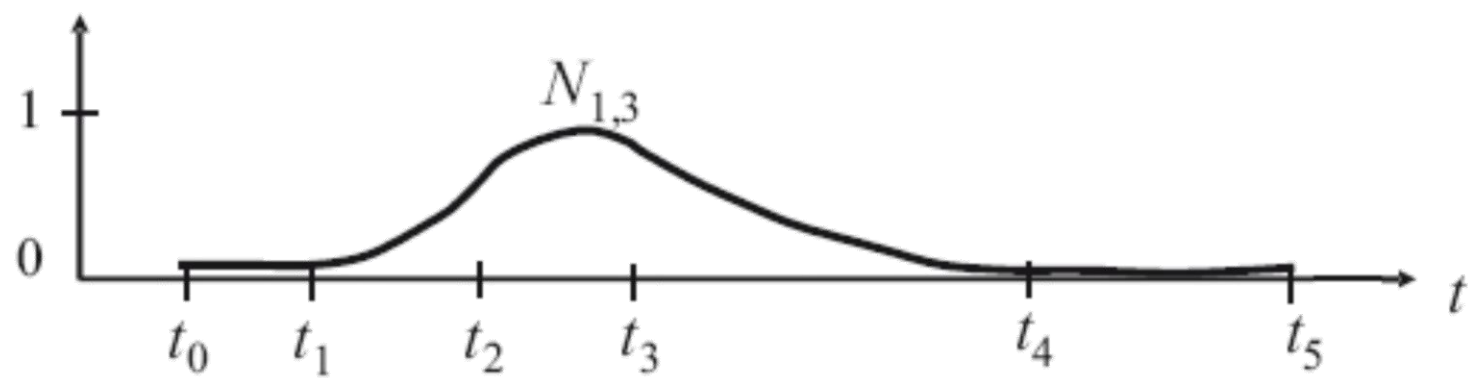
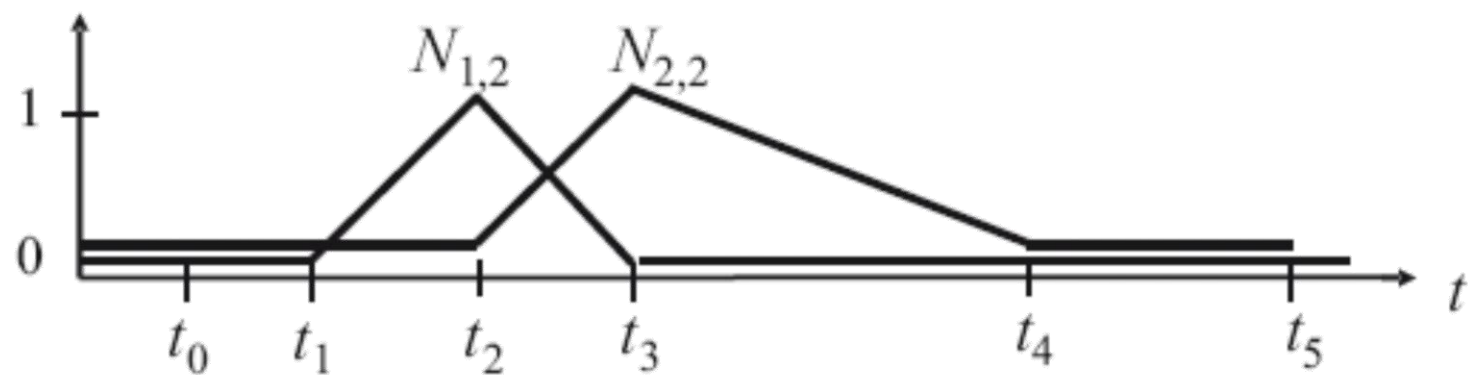


$$N_{i,1}(t) = \begin{cases} 1, & t_i \leq t < t_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

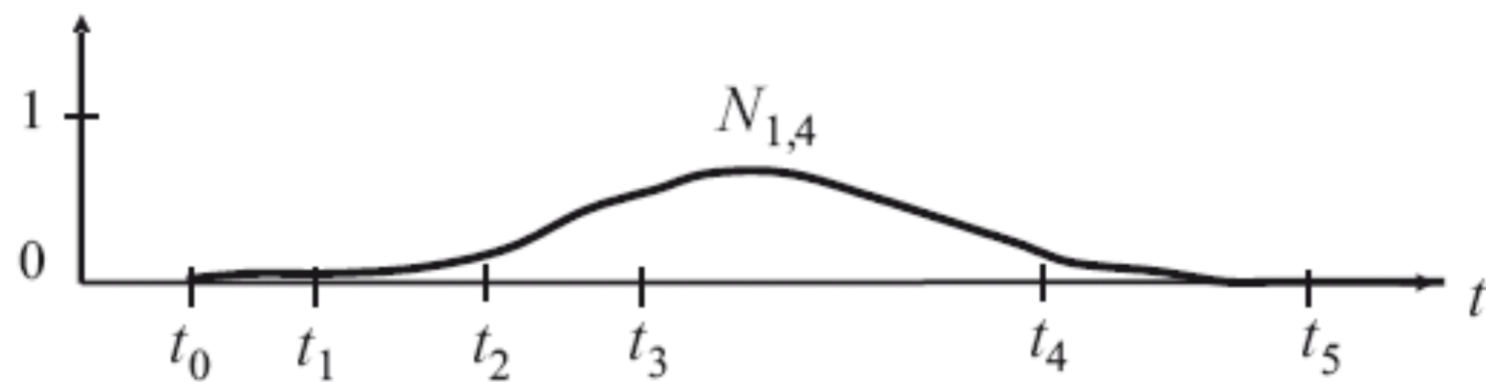
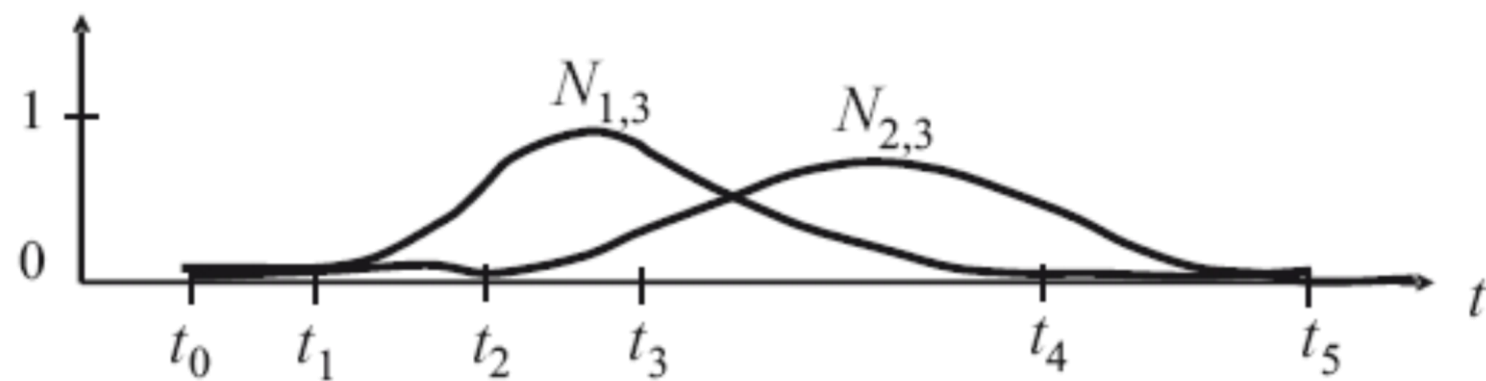
$$N_{i,k}(t) = \frac{t-t_i}{t_{i+k-1}-t_i} N_{i,k-1}(t) + \frac{t_{i+k}-t}{t_{i+k}-t_{i+1}} N_{i+1,k-1}(t)$$

for $k > 1$ and $i = 0, \dots, n$

Example

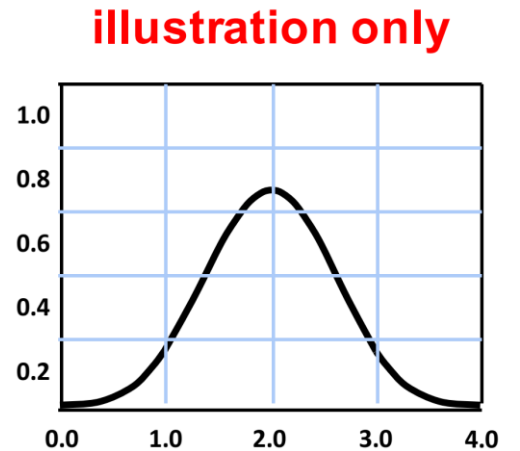


Example



Key Ideas

- We design one basis function $b(t)$
- Properties:
 - $b(t)$ is C^2 continuous
 - $b(t)$ is piecewise polynomial, degree 3 (cubic)
 - $b(t)$ has local support
 - Overlaying shifted $b(t + i)$ forms a partition of unity
 - $b(t) \geq 0$ for all t
- In short:
 - All desirable properties build into the basis
 - Linear combinations will inherit these



Shifted Basis Functions

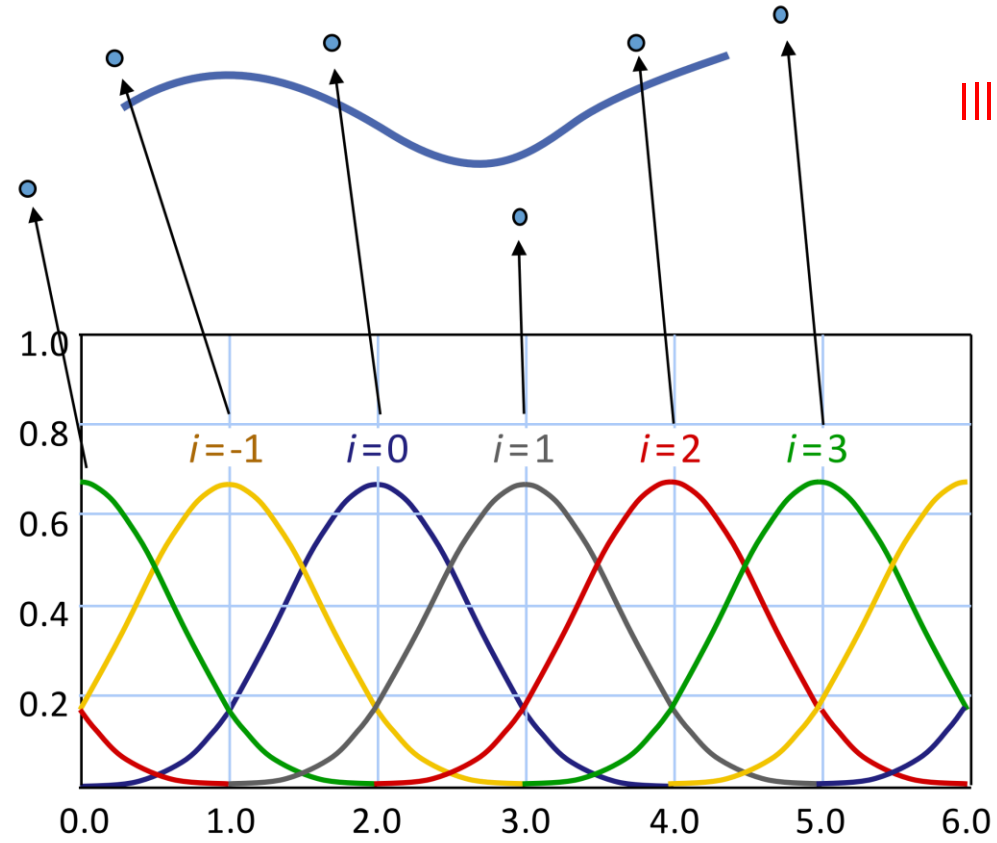


Illustration only

Shifted basis function $b(t)$

Basis properties

- For the so defined basis functions, the following properties can be shown:
 - $N_{i,k}(t) > 0$ for $t_i < t < t_{i+k}$
 - $N_{i,k}(t) = 0$ for $t_0 < t < t_i$ or $t_{i+k} < t < t_{n+k}$
 - $\sum_{i=0}^n N_{i,k}(t) = 1$ for $t_{k-1} \leq t \leq t_{n+1}$
- For $t_i \leq t_j \leq t_{i+k}$, the basis functions $N_{i,k}(t)$ are C^{k-2} at the knots t_j
- The interval $[t_i, t_{i+k}]$ is called support of $N_{i,k}$

B-spline curves

- Given: $n + 1$ control points $\mathbf{d}_0, \dots, \mathbf{d}_n \in \mathbb{R}^3$
knot vector $T = (t_0, \dots, t_n, \dots, t_{n+k})$
- Then, the B-spline curve $\mathbf{x}(t)$ of the order k is defined as

$$\mathbf{x}(t) = \sum_{i=0}^n N_{i,k}(t) \cdot \mathbf{d}_i$$

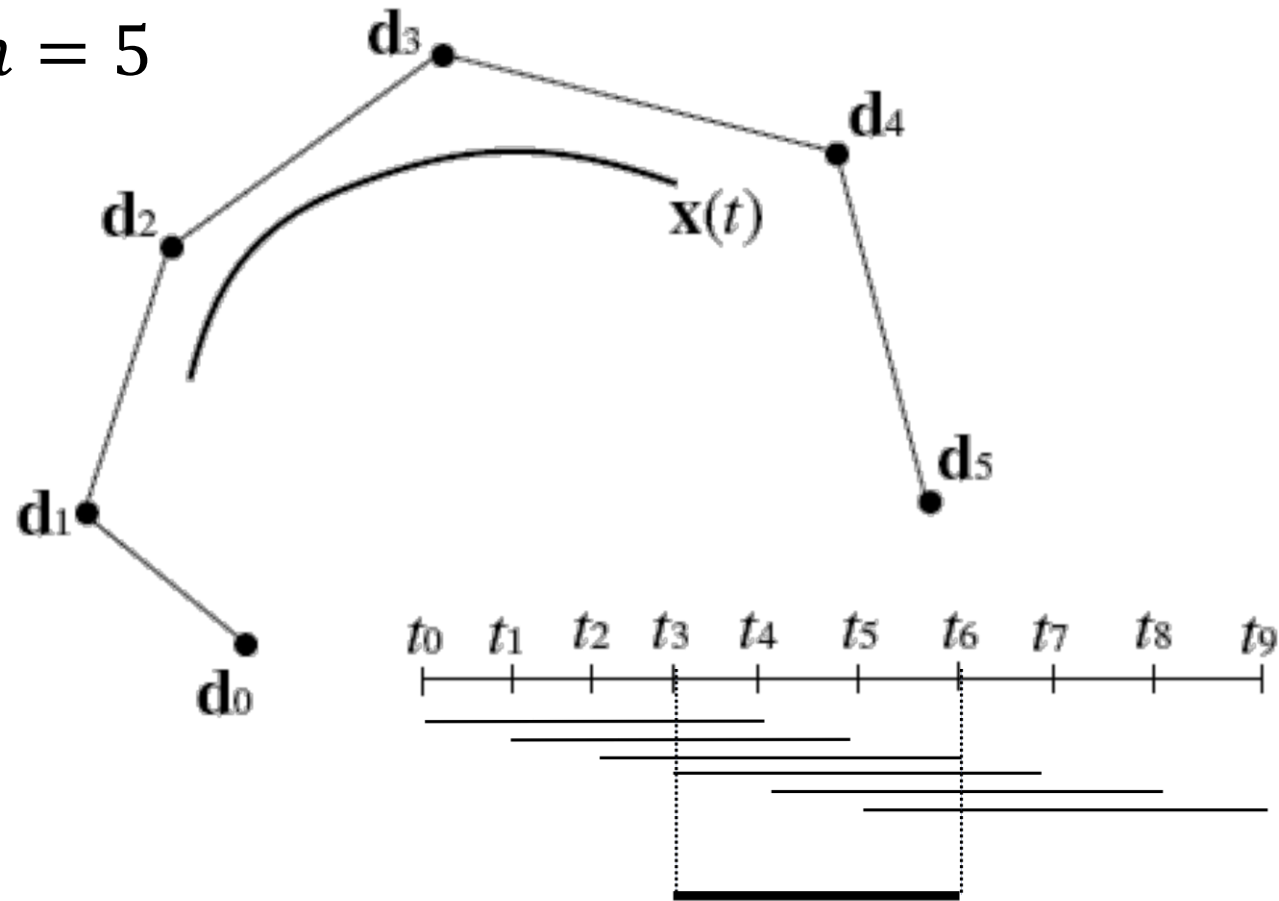
- The points \mathbf{d}_i are called *de Boor points*

Carl R. de Boor

German-American mathematician
University of Wisconsin-Madison

Example

- $k = 4, n = 5$



Support intervals of $N_{i,k}$

Curve defined in interval $t_3 \leq t \leq t_6$

B-spline curves

Multiple weighted knot vectors

- So far: $T = (t_0, \dots, t_n, \dots, t_{n+k})$ with $t_0 < t_1 < \dots < t_{n+k}$
- Now: also multiple knots allowed, i.e. with $t_0 \leq t_1 \leq \dots \leq t_{n+k}$
- The recursive definition of the B spline function $N_{i,k}$ ($i = 0, \dots, n$) works nonetheless, as long as no more than k knots coincide

B-spline curves

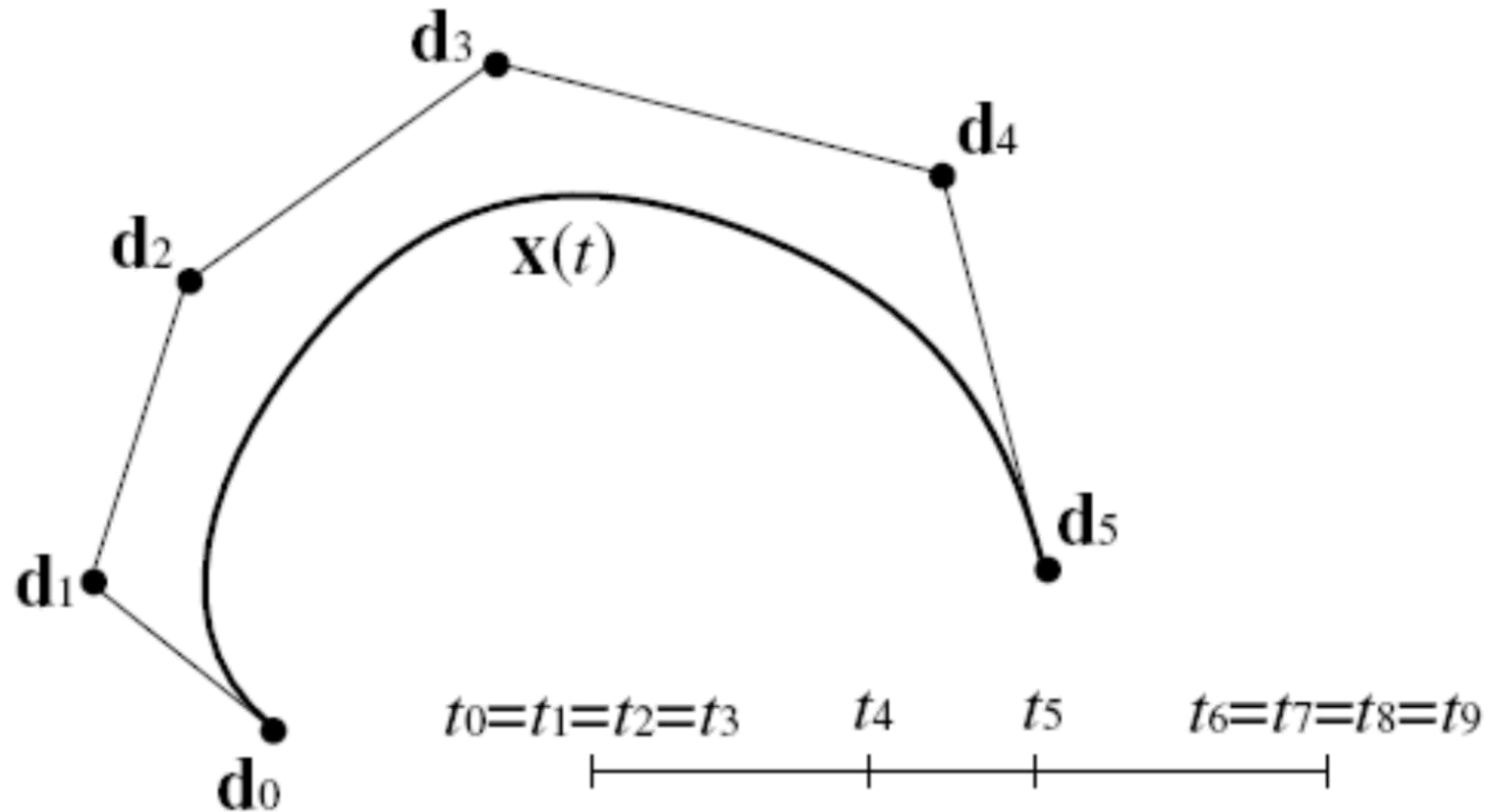
Effect of multiple knots:

- set: $t_0 = t_1 = \cdots = t_{k-1}$
- and $t_{n+1} = t_{n+2} = \cdots = t_{n+k}$

\mathbf{d}_0 and \mathbf{d}_n are interpolated

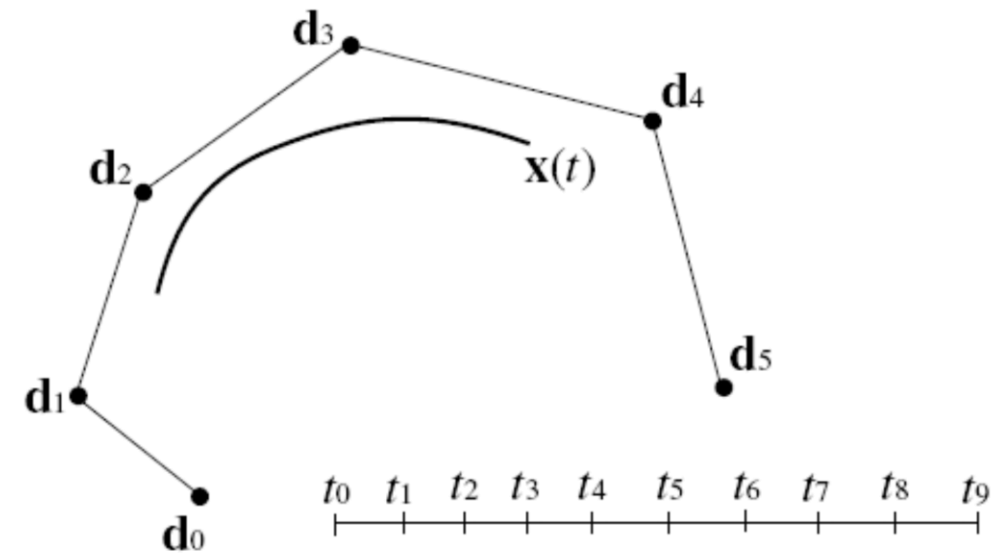
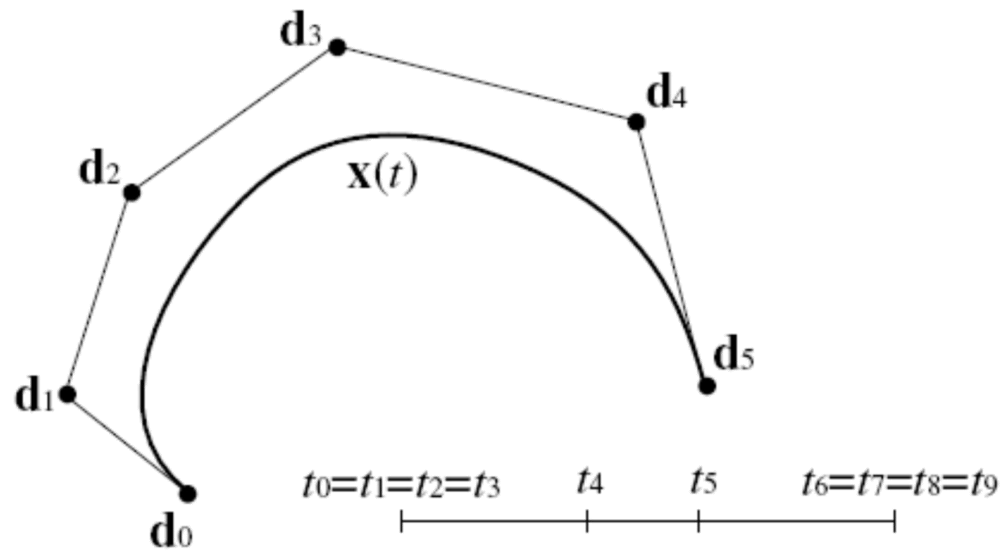
B-spline curves

- Example: $k = 4, n = 5$



B-spline curves

- Example: $k = 4, n = 5$



B-spline curves

- Further example



B-spline curves

Interesting property:

- B-spline functions $N_{i,k}$ ($i = 0, \dots, k - 1$) of the order k over the knot vector $T = (t_0, t_1, \dots, t_{2k-1}) = (\underbrace{0, \dots, 0}_{k \text{ times}}, \underbrace{1, \dots, 1}_{k \text{ times}})$

are Bernstein polynomials B_i^{k-1} of degree $k - 1$

B-spline curves properties

- Given:
 - $T = (\underbrace{t_0, \dots, t_0}_{k \text{ times}}, t_k, \dots, t_n, \underbrace{t_{n+1}, \dots, t_{n+1}}_{k \text{ times}})$
 - de Boor polygon $\mathbf{d}_0, \dots, \mathbf{d}_n$
- Then, the following applies for the related B-spline curve $\mathbf{x}(t)$:

B-spline curves properties

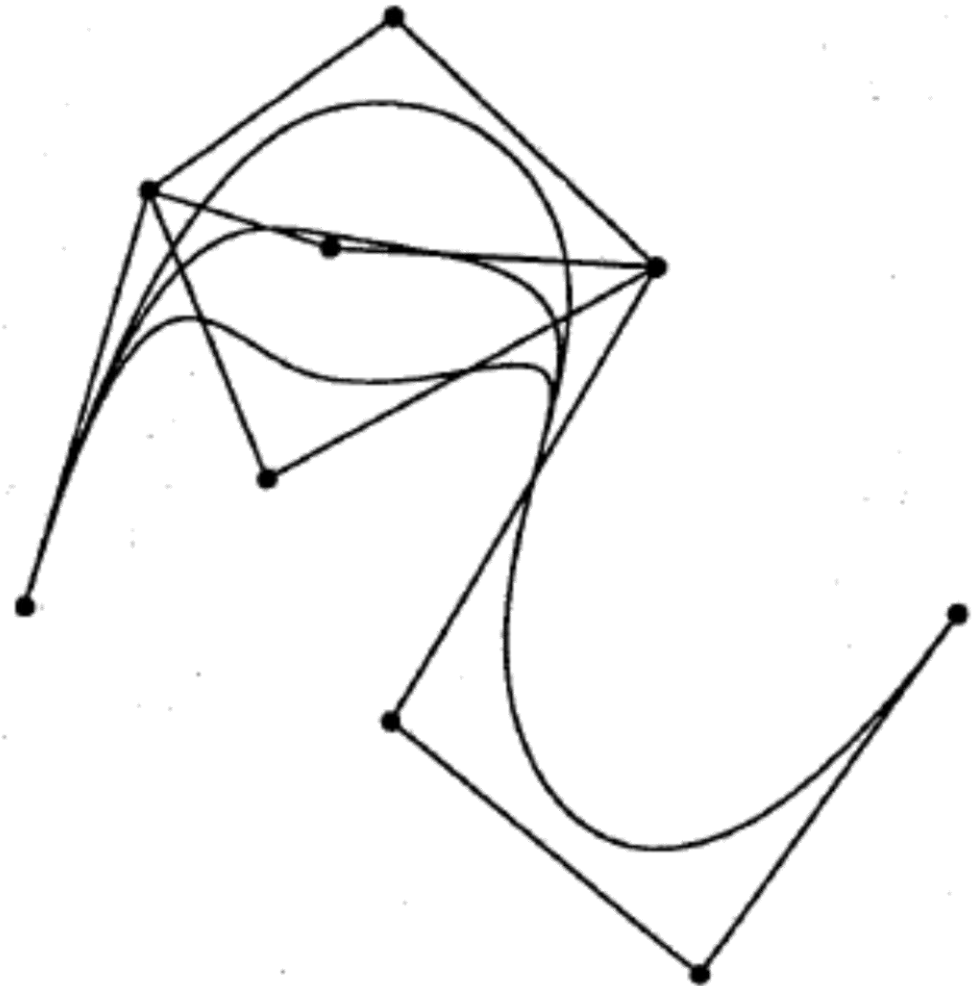
- $\mathbf{x}(t_0) = \mathbf{d}_0, \mathbf{x}(t_{n+1}) = \mathbf{d}_n$ (end point interpolation)
- $\mathbf{x}'(t_0) = \frac{k-1}{t_k - t_0} (\mathbf{d}_1 - \mathbf{d}_0)$ (tangent direction at \mathbf{d}_0 , similar in \mathbf{d}_n)
- $\mathbf{x}(t)$ consists of $n - k + 2$ polynomial curve segments of degree $k - 1$ (assuming no multiple inner knots)

B-spline curves properties

- Multiple inner knots \Rightarrow reduction of continuity of $x(t)$.
 l -times inner knot ($1 \leq l < k$) means
 C^{k-l-1} -continuity
- Local impact of the de Boor points: moving of d_i only changes the curve in the region $[t_i, t_{i+k}]$
- The insertion of new de Boor points does not change the polynomial degree of the curve segments

B-spline curves properties

Locality of B-spline curves



B-spline curves

Evaluation of B-spline curves

- Using B-spline functions
- Using the de Boor algorithm
Similar algorithm to the de Casteljau algorithm for Bézier curves;
consists of a number of linear interpolations on the de Boor polygon

The de Boor algorithm

- Given:

$\mathbf{d}_0, \dots, \mathbf{d}_n$: de Boor points

$(t_0, \dots, t_{k-1} = t_0, t_k, t_{k+1}, \dots, t_n, t_{n+1}, \dots, t_{n+k} = t_{n+1})$:

Knot vector

- wanted:

Curve point $\mathbf{x}(t)$ of the B-spline curve of the order k

The de Boor algorithm

1. Search index r with $t_r \leq t < t_{r+1}$

2. for $i = r - k + 1, \dots, r$

$$d_i^0 = d_i$$

- for $j = 1, \dots, k - 1$

for $i = r - k + 1 + j, \dots, r$

$$d_i^j = \left(1 - \alpha_i^j\right) \cdot d_{i-1}^{j-1} + \alpha_i^j \cdot d_i^{j-1}$$

$$\text{with } \alpha_i^j = \frac{t - t_i}{t_{i+k-j} - t_i}$$

Then: $d_r^{k-1} = x(t)$

B-spline curves

- The intermediate coefficients $d_i^j(t)$ can be placed into a triangular shaped matrix of points – the de Boor scheme:

$$d_{r-k+1} = d_{r-k+1}^0$$

$$d_{r-k+2} = d_{r-k+2}^0 \quad d_{r-k+2}^1$$

...

$$d_{r-1} = d_{r-1}^0 \quad d_{r-1}^1 \quad \dots \quad d_{r-1}^{k-2}$$

$$d_r = d_r^0 \quad d_r^1 \quad \dots \quad d_r^{k-2} \quad d_r^{k-1} = x(t)$$

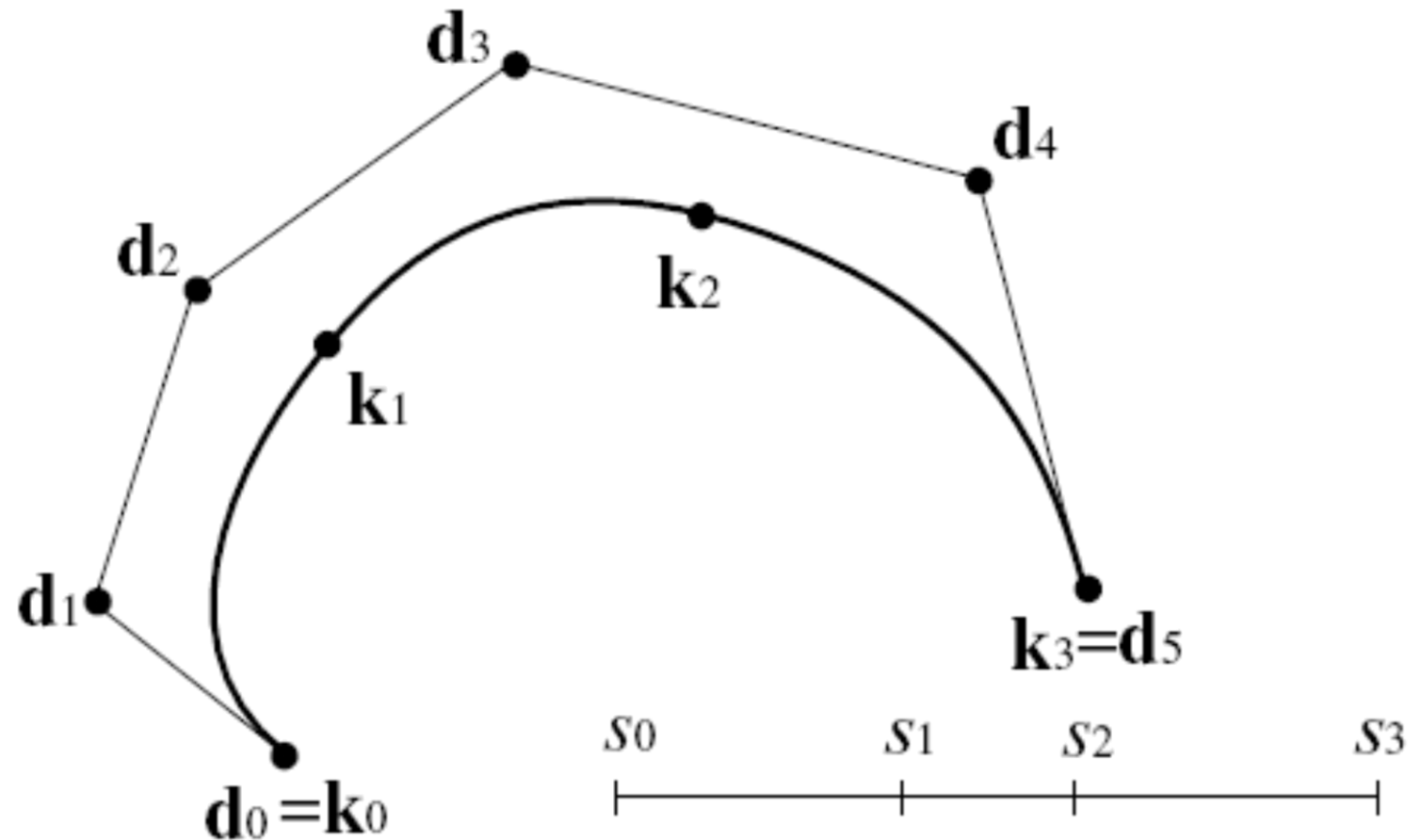
B-spline curves: interpolation

Interpolating B-spline curves

- Given: $n + 1$ control points $\mathbf{k}_0, \dots, \mathbf{k}_n$
knot sequence s_0, \dots, s_n
- Wanted: piecewise cubic interpolating B-spline curve \mathbf{x}
i.e., $\mathbf{x}(s_i) = \mathbf{k}_i$ for $i = 0, \dots, n$
- Approach: piecewise cubic $\Rightarrow k = 4$
 - $\mathbf{x}(t)$ consists of n segments $\Rightarrow n + 3$ de Boor points

B-spline curves: interpolation

- Example: $n = 3$



B-spline curves: interpolation

- We choose the knot vector
 - $T = (t_0, t_1, t_2, t_3, t_4, \dots, t_{n+2}, t_{n+3}, t_{n+4}, t_{n+5}, t_{n+6})$
 $= (s_0, s_0, s_0, s_0, s_1, \dots, s_{n-1}, s_n, s_n, s_n, s_n)$
- Then, the following conditions arise:
 - $\mathbf{x}(s_0) = \mathbf{k}_0 = \mathbf{d}_0$
 - $\mathbf{x}(s_i) = \mathbf{k}_i = N_{i,4}(s_i)\mathbf{d}_i + N_{i+1,4}(s_i)\mathbf{d}_{i+1} + N_{i+2,4}(s_i)\mathbf{d}_{i+2}$
for $i = 1, \dots, n - 1$
 - $\mathbf{x}(s_n) = \mathbf{k}_n = \mathbf{d}_{n+2}$
- Total: $n + 1$ conditions for $n + 3$ unknown de Boor points
→ 2 end conditions

B-spline curves: interpolation

- Here as example: natural end conditions

$$x''(s_0) = 0 \Leftrightarrow \frac{d_2 - d_1}{s_2 - s_0} = \frac{d_1 - d_0}{s_1 - s_0}$$

$$x''(s_n) = 0 \Leftrightarrow \frac{d_{n+2} - d_{n+1}}{s_n - s_{n-1}} = \frac{d_{n+1} - d_n}{s_n - s_{n-2}}$$

B-spline curves: interpolation

- This results in the following tridiagonal system of equations:

$$\begin{pmatrix}
 1 & & & & & \\
 \alpha_0 & \beta_0 & \gamma_0 & & & \\
 & \alpha_1 & \beta_1 & \gamma_1 & & \\
 & & & \ddots & & \\
 & & & & \ddots & \\
 & & & & & \ddots & \\
 & & & & & & \alpha_{n-1} & \beta_{n-1} & \gamma_{n-1} & \\
 & & & & & & & \alpha_n & \beta_n & \gamma_n \\
 & & & & & & & & & 1
 \end{pmatrix}
 \begin{pmatrix}
 d_0 \\
 d_1 \\
 d_2 \\
 \vdots \\
 \vdots \\
 \vdots \\
 d_n \\
 d_{n+1} \\
 d_{n+2}
 \end{pmatrix}
 =
 \begin{pmatrix}
 k_0 \\
 0 \\
 k_1 \\
 \vdots \\
 \vdots \\
 \vdots \\
 k_{n-1} \\
 0 \\
 k_n
 \end{pmatrix}$$

B-spline curves: interpolation

- with

$$\alpha_0 = s_2 - s_0$$

$$\beta_0 = -(s_2 - s_0) - (s_1 - s_0)$$

$$\gamma_0 = s_1 - s_0$$

$$\alpha_n = s_n - s_{n-1}$$

$$\beta_n = -(s_n - s_{n-1}) - (s_n - s_{n-2})$$

$$\gamma_n = s_n - s_{n-2}$$

$$\alpha_i = N_{i,4}(s_i)$$

$$\beta_i = N_{i+1,4}(s_i)$$

$$\gamma_i = N_{i+2,4}(s_i)$$

for $i = 1, \dots, n - 1$

Natural end conditions



B-spline curves: interpolation

- Solving a tridiagonal system of equations: Thomas-algorithm!
- $O(n)$
- Only for diagonally dominant matrices

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \cdot & \\ & & \cdot & \cdot & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \cdot \\ \cdot \\ d_n \end{bmatrix}$$

B-spline curves: interpolation

- Solving a tridiagonal system of equation: Thomas-algorithm!

Forward elimination phase

for $k = 2:n$

$$m = \frac{a_k}{b_{k-1}}$$

$$b_k = b_k - mc_{k-1}$$

$$d_k = d_k - md_{k-1}$$

end

Backward substitution phase

$$x_n = \frac{d_n}{b_n}$$

for $k = n - 1:-1:1$

$$x_k = \frac{d_k - c_k x_{k+1}}{b_k}$$

end

Bézier splines to B-splines

Conversion between cubic Bézier and B-spline curves

- Given:

$\mathbf{k}_0, \dots, \mathbf{k}_n$: control points

t_0, \dots, t_n : knot sequence

2 end conditions

b_0, \dots, b_{3n} : Bézier points for C^2 -continuous interpolating cubic Bézier spline curve

- Wanted: same curve in B-spline form

Bézier splines to B-splines

- Knot vector $T = (t_0, t_0, t_0, t_0, t_1, \dots, t_{n-1}, t_n, t_n, t_n, t_n)$

- $\mathbf{d}_0, \dots, \mathbf{d}_{n+2}$ are determined by

$$d_0 = b_0$$

$$d_1 = b_1$$

$$d_i = b_{3i-4} + \frac{\Delta_{i-1}}{\Delta_{i-2}} (b_{3i-4} - b_{3i-5}) \text{ for } i = 2, \dots, n$$

$$d_{n+1} = b_{3n-1}$$

$$d_{n+2} = b_{3n}$$

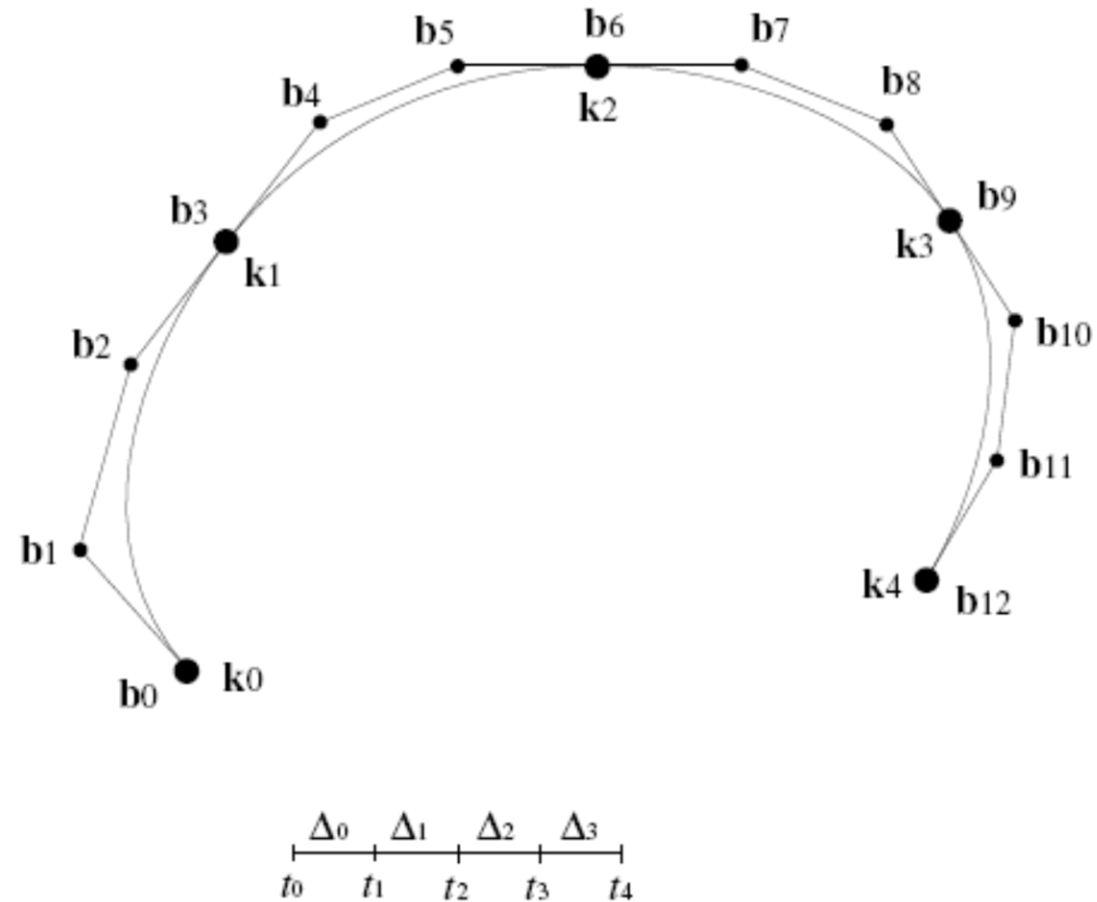
where $\Delta_i = t_{i+1} - t_i$ for $i = 0, \dots, n-1$

- The inverse problem is solvable as well

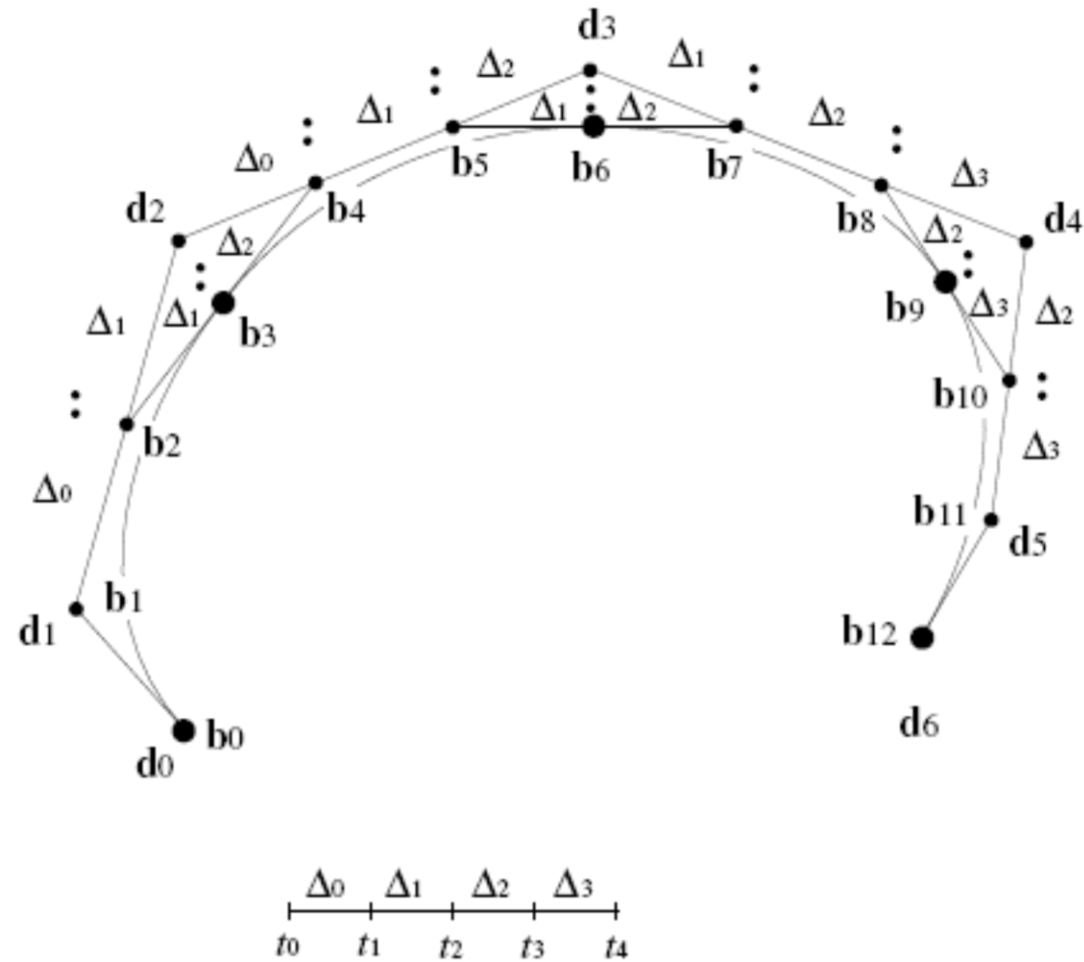
Remember the condition on d^- and d^+ for C^2 continuity of Bézier splines

Bézier splines to B-splines

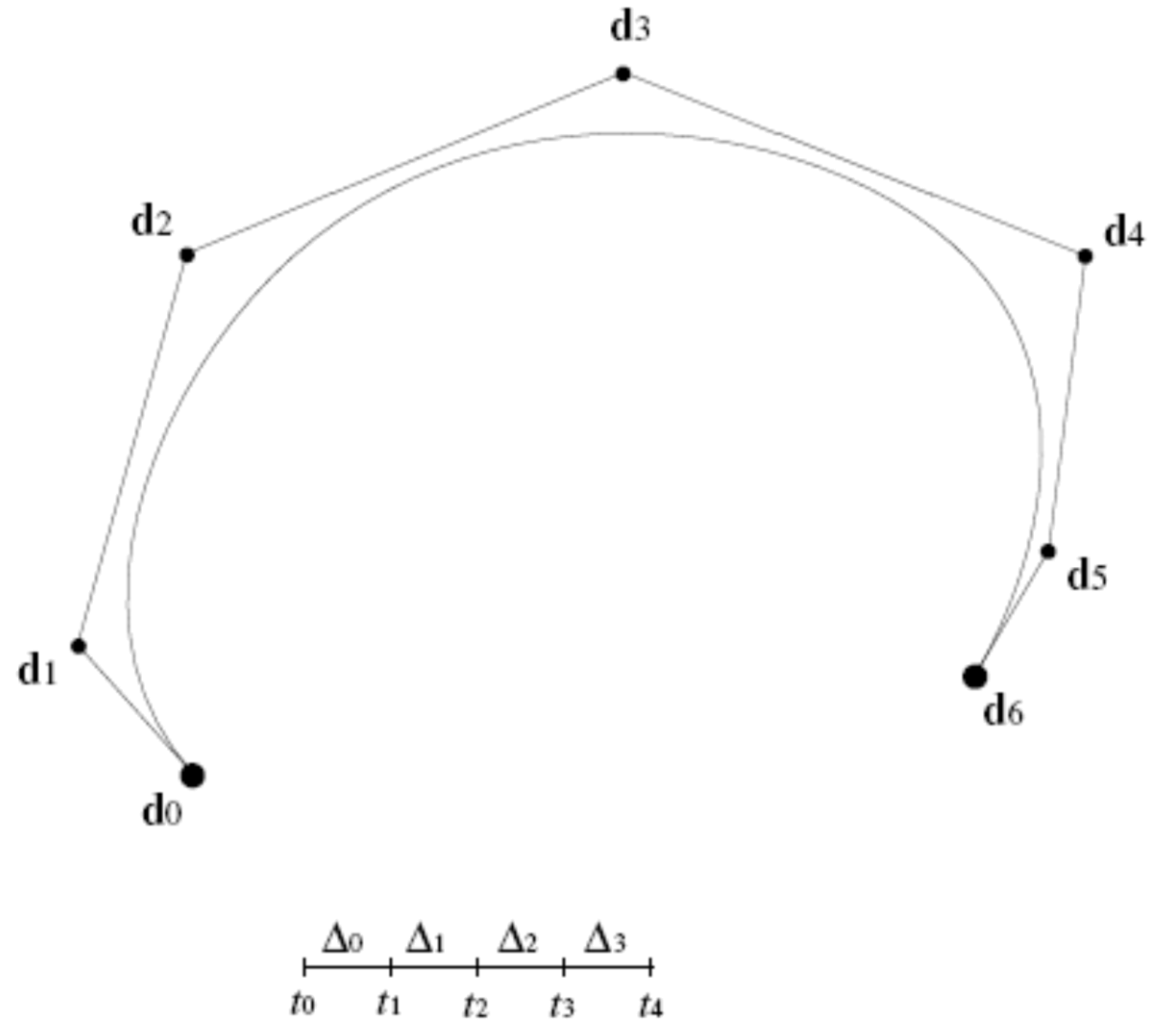
- Examples: $n = 4$



Bézier splines to B-splines



Bézier splines to B-splines



Summary of Bézier and B-spline curves

1. Bézier curve for $n + 1$ control points b_0, \dots, b_n :
 - Polynomial curve of degree n
 - Uniquely defined by control points
 - End point interpolation, remaining points are approximated
 - Pseudo-local impact of control points

Summary of Bézier and B-spline curves

2. Interpolating cubic Bézier-spline curves by control points k_0, \dots, k_n
 - Consists of n piecewise cubic curve segments
 - \mathcal{C}^2 -continuous at the control points
 - Uniquely defined by parameterization (i.e. knot sequence) and two end conditions
 - Interpolates all control points
 - Pseudo-local impact of the control points

Summary of Bézier and B-spline curves

3. Piecewise cubic B-spline curve for control points d_0, \dots, d_n and knot vector $T = (t_0, t_0, t_0, t_0, t_1, \dots, t_{n-1}, t_n, t_n, t_n, t_n)$
- Consists of $n - 2$ piecewise cubic curve segments which are C^2 at the knots
 - Uniquely defined by d_i and T
 - End point interpolation, the remaining points are approximated
 - Local impact of the de Boor points

Summary of Bézier and B-spline curves

4. Interpolating cubic B-spline through the control points k_0, \dots, k_n
 - Possible to formulate like (3) using 2 end conditions and solution of a tridiagonal system of equations for each x, y - and z - component
 - Identical curve to (2)

B-splines

detailed examples

B-spline curves: general case (reminder)

- Given: knot sequence $t_0 < t_1 < \dots < t_n < \dots < t_{n+k}$
($(t_0, t_1, \dots, t_{n+k})$ is called knot vector)
- Normalized B-spline functions $N_{i,k}$ of the order k (degree $k - 1$) are defined as:

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

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

for $k > 1$ and $i = 0, \dots, n$

- **Remark:**
 - If a knot value is repeated k times, the denominator may vanish
 - In this case: The fraction is treated as a zero

B-spline basis evaluation: ex. 1

- For order 4 and knot sequence

$$T = [t_0 \ t_1 \ t_2 \ t_3 \ t_4 \ t_5 \ t_6 \ t_7] = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1]$$

Evaluate the B-spline function $N_{0,4}(t)$, $N_{1,4}(t)$, $N_{2,4}(t)$, $N_{3,4}(t)$

B-spline basis evaluation: ex. 1

$$N_{0,1}(t) = N_{1,1}(t) = N_{2,1}(t) = N_{4,1}(t) = N_{5,1}(t) = N_{6,1}(t) = 0$$

$$N_{3,1}(t) = 1 \quad (0 \leq t < 1)$$

$$N_{0,2}(t) = \frac{t-t_0}{t_1-t_0} N_{0,1}(t) + \frac{t_2-t}{t_2-t_1} N_{1,1}(t) = 0$$

$$N_{1,2}(t) = \frac{t-t_1}{t_2-t_1} N_{1,1}(t) + \frac{t_3-t}{t_3-t_2} N_{2,1}(t) = 0$$

$$N_{2,2}(t) = \frac{t-t_2}{t_3-t_2} N_{2,1}(t) + \frac{t_4-t}{t_4-t_3} N_{3,1}(t) = (1-t)N_{3,1}(t)$$

$$N_{3,2}(t) = \frac{t-t_3}{t_4-t_3} N_{3,1}(t) + \frac{t_5-t}{t_5-t_4} N_{4,1}(t) = tN_{3,1}(t)$$

$$N_{4,2}(t) = \frac{t-t_4}{t_5-t_4} N_{4,1}(t) + \frac{t_6-t}{t_6-t_5} N_{5,1}(t) = 0$$

$$N_{5,2}(t) = \frac{t-t_5}{t_6-t_5} N_{5,1}(t) + \frac{t_7-t}{t_7-t_6} N_{6,1}(t) = 0$$

B-spline basis evaluation: ex. 1

$$N_{0,3}(t) = \frac{t-t_0}{t_2-t_0} N_{0,2}(t) + \frac{t_3-t}{t_3-t_1} N_{1,2}(t) = 0$$

$$N_{1,3}(t) = \frac{t-t_1}{t_3-t_1} N_{1,2}(t) + \frac{t_4-t}{t_4-t_2} N_{2,2}(t) = (1-t)^2 N_{3,1}(t)$$

$$N_{2,3}(t) = \frac{t-t_2}{t_4-t_2} N_{2,2}(t) + \frac{t_5-t}{t_5-t_3} N_{3,2}(t) = 2t(1-t) N_{3,1}(t)$$

$$N_{3,3}(t) = \frac{t-t_3}{t_5-t_3} N_{3,2}(t) + \frac{t_6-t}{t_6-t_4} N_{4,2}(t) = t^2 N_{3,1}(t)$$

$$N_{4,3}(t) = \frac{t-t_4}{t_6-t_4} N_{4,2}(t) + \frac{t_7-t}{t_7-t_5} N_{5,2}(t) = 0$$

B-spline basis evaluation: ex. 1

- Finally
$$N_{0,4}(t) = \frac{t-t_0}{t_3-t_0} N_{0,3}(t) + \frac{t_4-t}{t_4-t_1} N_{1,3}(t) = (1-t)^3 N_{3,1}(t)$$
$$N_{1,4}(t) = \frac{t-t_1}{t_4-t_1} N_{1,3}(t) + \frac{t_5-t}{t_5-t_2} N_{2,3}(t) = 3(1-t)^2 t N_{3,1}(t)$$
$$N_{2,4}(t) = \frac{t-t_2}{t_5-t_2} N_{2,3}(t) + \frac{t_6-t}{t_6-t_3} N_{3,3}(t) = 3(1-t) t^2 N_{3,1}(t)$$
$$N_{3,4}(t) = \frac{t-t_3}{t_6-t_3} N_{3,3}(t) + \frac{t_7-t}{t_7-t_4} N_{4,3}(t) = t^3 N_{3,1}(t)$$

B-spline basis evaluation: ex. 1

- Finally
$$N_{0,4}(t) = \frac{t-t_0}{t_3-t_0} N_{0,3}(t) + \frac{t_4-t}{t_4-t_1} N_{1,3}(t) = (1-t)^3 N_{3,1}(t)$$
$$N_{1,4}(t) = \frac{t-t_1}{t_4-t_1} N_{1,3}(t) + \frac{t_5-t}{t_5-t_2} N_{2,3}(t) = 3(1-t)^2 t N_{3,1}(t)$$
$$N_{2,4}(t) = \frac{t-t_2}{t_5-t_2} N_{2,3}(t) + \frac{t_6-t}{t_6-t_3} N_{3,3}(t) = 3(1-t) t^2 N_{3,1}(t)$$
$$N_{3,4}(t) = \frac{t-t_3}{t_6-t_3} N_{3,3}(t) + \frac{t_7-t}{t_7-t_4} N_{4,3}(t) = t^3 N_{3,1}(t)$$
- We clearly get the Bernstein basis function as mentioned earlier

B-spline basis evaluation: ex. 2

- For order 4 and not sequence

$$T = [t_0 \quad t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7] = [-3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4]$$

Evaluate the corresponding basis

B-spline basis evaluation: ex. 2

$$N_{0,2}(t) = (t + 3)N_{0,1}(t) + (-1 - t)N_{1,1}(t)$$

$$N_{1,2}(t) = (t + 2)N_{1,1}(t) + (-t)N_{2,1}(t)$$

$$N_{2,2}(t) = (t + 1)N_{2,1}(t) + (1 - t)N_{3,1}(t)$$

$$N_{3,2}(t) = tN_{3,1}(t) + (2 - t)N_{4,1}(t)$$

$$N_{4,2}(t) = (t - 1)N_{4,1}(t) + (3 - t)N_{5,1}(t)$$

$$N_{5,2}(t) = (t - 2)N_{5,1}(t) + (4 - t)N_{6,1}(t)$$

B-spline basis evaluation: ex. 2

$$N_{0,3}(t) = \frac{1}{2}(t+2)N_{0,2}(t) + \frac{1}{2}(0-t)N_{1,2}(t)$$

$$N_{1,3}(t) = \frac{1}{2}(t+1)N_{1,2}(t) + \frac{1}{2}(1-t)N_{2,2}(t)$$

$$N_{2,3}(t) = \frac{1}{2}(t+0)N_{2,2}(t) + \frac{1}{2}(2-t)N_{3,2}(t)$$

$$N_{3,3}(t) = \frac{1}{2}(t-1)N_{3,2}(t) + \frac{1}{2}(3-t)N_{4,2}(t)$$

B-spline basis evaluation: ex. 2

- Finally

$$N_{0,4}(t) = \frac{1}{3}(t+3)N_{0,3}(t) + \frac{1}{3}(1-t)N_{1,3}(t)$$

$$N_{1,4}(t) = \frac{1}{3}(t+2)N_{1,3}(t) + \frac{1}{3}(2-t)N_{2,3}(t)$$

$$N_{2,4}(t) = \frac{1}{3}(t+1)N_{2,3}(t) + \frac{1}{3}(3-t)N_{3,3}(t)$$

$$N_{3,4}(t) = \frac{1}{3}tN_{3,3}(t) + \frac{1}{3}(4-t)N_{4,3}(t)$$

B-spline basis evaluation: ex. 2

- Then substituting

$$N_{0,4}(t) = \frac{1}{6}(t+3)^3 N_{0,1}(t) + \left\{ -(t+1)^3 + \frac{2}{3}t^3 - \frac{1}{6}(t-1)^3 \right\} N_{1,1}(t) + \left\{ \frac{2}{3}t^3 - \frac{1}{t}(t-1)^3 \right\} N_{2,1}(t) - \frac{1}{6}(t-1)^3 N_{3,1}(t)$$

$$N_{1,4}(t) = \frac{1}{6}(t+2)^3 N_{1,1}(t) + \left\{ -t^3 + \frac{2}{3}(t-1)^3 - \frac{1}{6}(t-2)^3 \right\} N_{2,1}(t) + \left\{ \frac{2}{3}(t-1)^3 - \frac{1}{t}(t-2)^3 \right\} N_{3,1}(t) - \frac{1}{6}(t-2)^3 N_{4,1}(t)$$

$$N_{2,4}(t) = \frac{1}{6}(t+1)^3 N_{2,1}(t) + \left\{ -(t-1)^3 + \frac{2}{3}(t-2)^3 - \frac{1}{6}(t-3)^3 \right\} N_{3,1}(t) + \left\{ \frac{2}{3}(t-2)^3 - \frac{1}{t}(t-3)^3 \right\} N_{4,1}(t) - \frac{1}{6}(t-3)^3 N_{5,1}(t)$$

$$N_{3,4}(t) = \frac{1}{6}t^3 N_{3,1}(t) + \left\{ -(t-2)^3 + \frac{2}{3}(t-3)^3 - \frac{1}{6}(t-4)^3 \right\} N_{4,1}(t) + \left\{ \frac{2}{3}(t-3)^3 - \frac{1}{t}(t-4)^3 \right\} N_{5,1}(t) - \frac{1}{6}(t-4)^3 N_{6,1}(t)$$

de Boor algorithm (reminder)

1. Search index r with $t_r \leq t < t_{r+1}$

2. for $i = r - k + 1, \dots, r$

$$d_i^0 = d_i \quad \text{sometimes noted as } d_i^0(t) = d_i$$

- for $j = 1, \dots, k - 1$

for $i = r - k + 1 + j, \dots, r$

$$d_i^j = \left(1 - \alpha_i^j\right) \cdot d_{i-1}^{j-1} + \alpha_i^j \cdot d_i^{j-1}$$

$$\text{with } \alpha_i^j = \frac{t - t_i}{t_{i+k-j} - t_i}$$

Then: $d_r^{k-1} = x(t)$

Order k
 $n + 1$ points
 $n + k + 1$ knots

de Boor algorithm: ex. 1

- For order 4, de Boor points Q_0, Q_1, \dots, Q_8 and knot sequence
$$T = [t_0 \quad t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7 \quad t_8 \quad t_9 \quad t_{10} \quad t_{11} \quad t_{12}]$$
$$= [0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 6 \quad 6 \quad 6]$$

Evaluate the B-spline curve at $t = 4.75$

de Boor algorithm: ex. 1

- Since $t_7 \leq 4.75 < t_8$, $r = 7$, therefore $i = 7 - 4 + 1 = 4$

$$\begin{aligned} Q_5^{[1]}(4.75) &= (1 - \lambda)Q_4^{[0]}(4.75) + \lambda Q_5^{[0]}(4.75) \\ &= (1 - \lambda)Q_4 + \lambda Q_5 = 0.083Q_4 + 0.917Q_5 \end{aligned} \quad \left(\lambda = \frac{4.75 - t_5}{t_8 - t_5} = 0.917 \right)$$

$$\begin{aligned} Q_6^{[1]}(4.75) &= (1 - \lambda)Q_5^{[0]}(4.75) + \lambda Q_6^{[0]}(4.75) \\ &= (1 - \lambda)Q_5 + \lambda Q_6 = 0.417Q_5 + 0.583Q_6 \end{aligned} \quad \left(\lambda = \frac{4.75 - t_6}{t_9 - t_6} = 0.583 \right)$$

$$\begin{aligned} Q_7^{[1]}(4.75) &= (1 - \lambda)Q_6^{[0]}(4.75) + \lambda Q_7^{[0]}(4.75) \\ &= (1 - \lambda)Q_6 + \lambda Q_7 = 0.625Q_6 + 0.375Q_7 \end{aligned} \quad \left(\lambda = \frac{4.75 - t_7}{t_{10} - t_7} = 0.375 \right)$$

de Boor algorithm: ex. 1

- Then

$$\begin{aligned}Q_6^{[2]}(4.75) &= (1 - \lambda)Q_5^{[1]}(4.75) + \lambda Q_6^{[1]}(4.75) \\&= 0.125(0.083Q_4 + 0.917Q_5) + 0.875(0.417Q_5 + 0.583Q_6) \\&= 0.01Q_4 + 0.479Q_5 + 0.510Q_6\end{aligned}$$

$$\left(\lambda = \frac{4.75 - t_6}{t_8 - t_6} = 0.875\right)$$

$$\begin{aligned}Q_7^{[1]}(4.75) &= (1 - \lambda)Q_6^{[1]}(4.75) + \lambda Q_7^{[1]}(4.75) \\&= 0.625(0.417Q_5 + 0.583Q_6) + 0.375(0.625Q_6 + 0.375Q_7) \\&= 0.261Q_5 + 0.598Q_6 + 0.141Q_7\end{aligned}$$

$$\left(\lambda = \frac{4.75 - t_7}{t_9 - t_7} = 0.375\right)$$

de Boor algorithm: ex. 1

- Then

$$\begin{aligned} Q_7^{[3]}(4.75) &= (1 - \lambda)Q_6^{[2]}(4.75) + \lambda Q_7^{[2]}(4.75) \\ &= 0.25(0.01Q_4 + 0.479Q_5 + 0.510Q_6) \\ &\quad + 0.75(0.261Q_5 + 0.598Q_6 + 0.141Q_7) \\ &= 0.0025Q_4 + 0.316Q_5 + 0.576Q_6 + 0.106Q_7 \end{aligned} \quad \left(\lambda = \frac{4.75 - t_7}{t_8 - t_7} = 0.75 \right)$$

Order k
 $n + 1$ points
 $n + k + 1$ knots

de Boor algorithm: ex. 2

- For order 4, de Boor points Q_0, Q_1, \dots, Q_6 and knot sequence
$$T = [t_0 \quad t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7 \quad t_8 \quad t_9 \quad t_{10}]$$
$$= [-3 \quad -2 \quad -1 \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7]$$

Evaluate the B-spline curve at $t = 3.5$

de Boor algorithm: ex. 1

- Since $t_6 \leq 3.5 < t_7$, $r = 7$, therefore $i = 6 - 4 + 1 = 3$

$$Q_4^{[1]}(3.5) = (1 - \lambda)Q_3^{[0]} + \lambda Q_4^{[0]} \quad \left(\lambda = \frac{3.5 - t_4}{4 - 1} = 0.833 \right)$$

$$= 0.167Q_3 + 0.833Q_4$$

$$Q_5^{[2]}(3.5) = (1 - \lambda)Q_4^{[1]} + \lambda Q_5^{[1]} \quad \left(\lambda = \frac{3.5 - t_5}{4 - 2} = 0.75 \right)$$

$$= 0.25(0.167Q_3 + 0.833Q_4) + 0.75(0.5Q_4 + 0.5Q_5)$$

$$= 0.042Q_3 + 0.583Q_4 + 0.375Q_5$$

$$Q_5^{[1]}(3.5) = (1 - \lambda)Q_4^{[0]} + \lambda Q_5^{[0]} \quad \left(\lambda = \frac{3.5 - t_5}{4 - 1} = 0.5 \right)$$

$$= 0.5Q_4 + 0.5Q_5$$

$$Q_6^{[2]}(3.5) = (1 - \lambda)Q_5^{[1]} + \lambda Q_6^{[1]} \quad \left(\lambda = \frac{3.5 - t_6}{4 - 2} = 0.25 \right)$$

$$= 0.75(0.5Q_4 + 0.5Q_5) + 0.25(0.833Q_5 + 0.167Q_6)$$

$$= 0.375Q_4 + 0.583Q_5 + 0.042Q_6$$

$$Q_6^{[1]}(3.5) = (1 - \lambda)Q_5^{[0]} + \lambda Q_6^{[0]} \quad \left(\lambda = \frac{3.5 - t_6}{4 - 1} = 0.167 \right)$$

$$= 0.833Q_5 + 0.167Q_6$$

$$Q_6^{[3]}(3.5) = (1 - \lambda)Q_5^{[2]} + \lambda Q_6^{[2]} \quad \left(\lambda = \frac{3.5 - t_6}{4 - 3} = 0.5 \right)$$

$$= 0.5(0.042Q_3 + 0.583Q_4 + 0.375Q_5) + 0.5(0.375Q_4 + 0.583Q_5 + 0.042Q_6)$$

$$= 0.021Q_3 + 0.479Q_4 + 0.479Q_5 + 0.021Q_6$$

Computer Aided Geometric Design

Fall Semester 2024

Blossoming and Polar Forms
Bézier Splines and B-Splines Revisited

陈仁杰

renjiec@ustc.edu.cn

<http://staff.ustc.edu.cn/~renjiec>

A Short Step Back

Bézier & Monomials

Matrix Form

Matrix Notation: Bézier \rightarrow Monomials

$$f(t) = (1 \quad t \quad t^2) \begin{pmatrix} 1 & 0 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \end{pmatrix} \quad (\text{quadratic case})$$

$$f(t) = (1 \quad t \quad t^2 \quad t^3) \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{pmatrix} \quad (\text{cubic case})$$

Format Conversion

Conversion: Compute Bézier coefficients from monomial coefficients

$$\begin{pmatrix} \mathbf{c}_0^{(Bez.)} \\ \mathbf{c}_1^{(Bez.)} \\ \mathbf{c}_2^{(Bez.)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{c}_0 \\ \mathbf{c}_1 \\ \mathbf{c}_2 \end{pmatrix} \quad \text{(quadratic case)}$$

$$\begin{pmatrix} \mathbf{c}_0^{(Bez.)} \\ \mathbf{c}_1^{(Bez.)} \\ \mathbf{c}_2^{(Bez.)} \\ \mathbf{c}_3^{(Bez.)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{c}_0 \\ \mathbf{c}_1 \\ \mathbf{c}_2 \\ \mathbf{c}_3 \end{pmatrix} \quad \text{(cubic case)}$$

Format Conversion

Conversion: quadratic to cubic

$$\begin{pmatrix} \mathbf{c}_0^{(3)} \\ \mathbf{c}_1^{(3)} \\ \mathbf{c}_2^{(3)} \\ \mathbf{c}_3^{(3)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 0 & 0 & 0 \\ -2 & 2 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{c}_0^{(2)} \\ \mathbf{c}_1^{(2)} \\ \mathbf{c}_2^{(2)} \\ 0 \end{pmatrix}$$

Convert to monomials and back to Bézier coefficients. (other degrees similar)

Example application: Output of TrueType fonts in Postscript.

Polar Forms & Blossoms

Idea & Definition

Affine Combinations

Definition (reminder):

- An *affine combination* of n points $\in \mathbb{R}^d$ is given by:

$$P_\alpha = \sum_{i=1}^n \alpha_i p_i \text{ with } \sum_{i=1}^n \alpha_i = 1$$

- A function f is said to be *affine* in its parameter x_i , if:

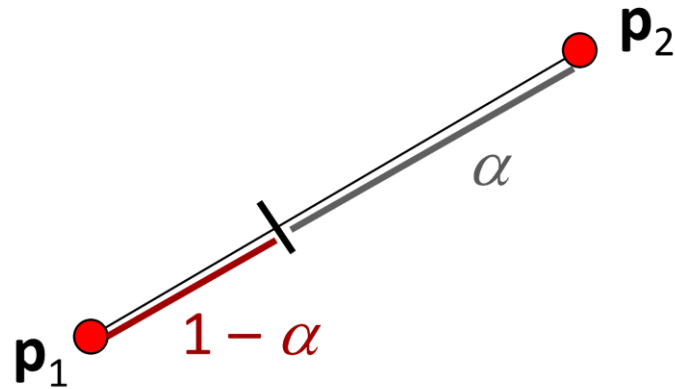
$$f\left(x_1, \dots, \sum_{i=1}^n \alpha_i x_i^{(k)}, \dots, x_m\right) = \sum_{i=1}^n \alpha_i f\left(x_1, \dots, x_i^{(k)}, \dots, x_m\right) \text{ for } \sum_{i=1}^n \alpha_i = 1$$

Affine Combinations

Examples:

- Linear (affine) interpolation of 2 points:

$$\mathbf{p}_\alpha = \alpha \mathbf{p}_1 + (1 - \alpha) \mathbf{p}_2$$

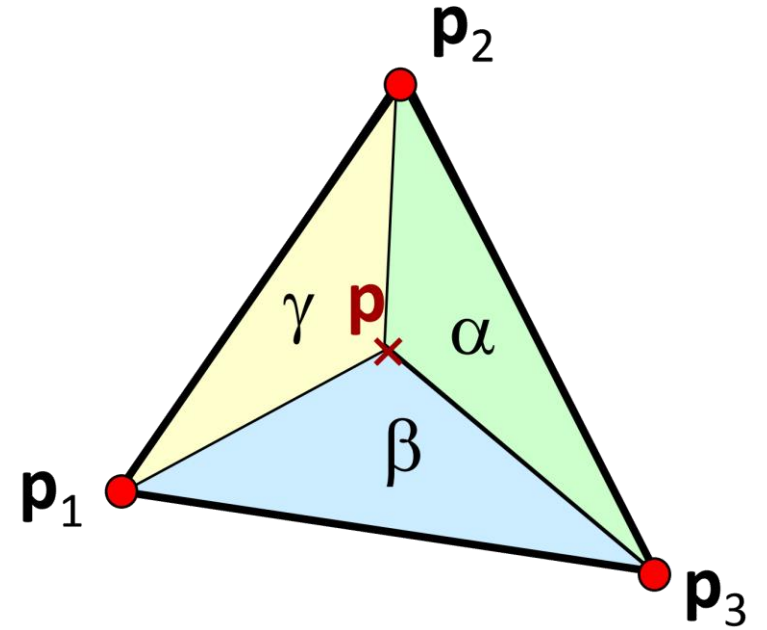


Affine Combinations

Examples:

- Barycentric combinations of 3 points
("barycentric coordinates")

$$p = \alpha p_1 + \beta p_2 + \gamma p_3, \text{ with } \alpha + \beta + \gamma = 1$$



Properties:

$$\gamma = 1 - \alpha - \beta$$

$$\alpha = \frac{\Delta(p_2, p_3, p)}{\Delta(p_1, p_2, p_3)},$$

$$\beta = \frac{\Delta(p_1, p_3, p)}{\Delta(p_1, p_2, p_3)},$$

$$\gamma = \frac{\Delta(p_1, p_2, p)}{\Delta(p_1, p_2, p_3)}$$

Transformation to barycentric coordinates is a linear map
(heights in triangles)

Formalizing the Idea

- **Idea:** Express (piecewise) polynomial curves as iterated linear (affine) interpolations
- **First steps:**
 - A polynomials: $P(t) = at^3 + bt^2 + ct + d$
 - Can be written as: $P(t) = a \cdot t \cdot t \cdot t + b \cdot t \cdot t + c \cdot t + d$
 - Interpret each variable t as a separate parameter:

$$p(t_1, t_2, t_3) = a \cdot t_1 \cdot t_2 \cdot t_3 + b \cdot t_1 \cdot t_2 + c \cdot t_1 + d$$

new function



Polar Forms

Solution: Polar Forms / Blossoms

A *polar form* or *blossom* f of a polynomial F of degree d is a function in d variables:

$$F: \mathbb{R} \rightarrow \mathbb{R}$$

$$f: \mathbb{R}^d \rightarrow \mathbb{R}$$

with the following properties:

- *Diagonality*: $f(t, t, \dots, t) = F(t)$
- *Symmetry*: $f(t_1, t_2, \dots, t_d) = f(t_{\pi(1)}, t_{\pi(2)}, \dots, t_{\pi(d)})$
for all permutations of indices π
- *Multi-affine*: $\sum \alpha_k = 1$
 $\Rightarrow f(t_1, t_2, \dots, \sum \alpha_k t_i^{(k)}, \dots, t_d)$
 $= \alpha_1 f(t_1, t_2, \dots, t_i^{(1)}, \dots, t_d) + \dots + \alpha_n f(t_1, t_2, \dots, t_i^{(n)}, \dots, t_d)$

Polar Forms

Rationale:

- Model polynomial as multi-affine function
 - (multi-affine property)
- Plug in a common parameter to obtain the original polynomial
 - (diagonal of the blossom)
- Symmetry property – makes the solution unique
 - There is exactly one polar form for each polynomial
 - This standardization makes different polars “compatible”, we can compare them with each other
 - We will see how this works in a few slides ...

Properties

The mapping from polynomials to their corresponding polar forms is one-to-one

- For each polar form $f(t_1, t_2, \dots, t_n)$,
a unique polynomial $F(t)$ exists
- For each polynomial $F(t)$,
a unique polar form $f(t_1, t_2, \dots, t_n)$ exists

Properties

Polar forms of monomials:

- Degree 0: $1 \rightarrow ?$
- Degree 1:
- Degree 2:
- Degree 3:

Properties

Polar forms of monomials:

- Degree 0: $1 \rightarrow 1$
- Degree 1: $1 \rightarrow 1, t \rightarrow t$
- Degree 2: $1 \rightarrow 1, t \rightarrow \frac{t_1+t_2}{2}, t^2 \rightarrow t_1t_2$
- Degree 3: $1 \rightarrow 1, t^2 \rightarrow \frac{t_1t_2+t_2t_3+t_1t_3}{3}$
 $t \rightarrow \frac{t_1+t_2+t_3}{3}, t^3 \rightarrow t_1t_2t_3$

Properties

Polar forms of monomials:

- Degree 0: $f = c_0$
- Degree 1: $f(t_1) = c_0 + c_1 t_1$
- Degree 2: $f(t_1, t_2) = c_0 + c_1 \frac{t_1 + t_2}{2} + c_2 t_1 t_2$
- Degree 3: $f(t_1, t_2, t_3) = c_0 + c_1 \frac{t_1 + t_2 + t_3}{3} + c_2 \frac{t_1 t_2 + t_2 t_3 + t_1 t_3}{3} + c_3 t_1 t_2 t_3$

Properties

General Case:

$$f(t_1, \dots, t_n) = \sum_{i=0}^n c_i \binom{n}{i}^{-1} \sum_{\substack{S \subseteq \{1 \dots n\}, \\ |S|=i}} \prod_{j \in S} t_j$$

- The c_i are the monomial coefficients
- Idea: use all possible subsets of t_i to make it symmetric
- This solution is unique
- Without the symmetry property, there would be a large number of solutions

$|S|$ denotes the cardinality of the set S

Generalizations

Blossoms for polynomials curves (points as output):

- Polar form of a polynomial curve of degree d :

$$\begin{array}{lcl} F: & \mathbb{R} & \rightarrow \mathbb{R}^{\textcolor{red}{n}} \\ f: & \mathbb{R}^{\textcolor{blue}{d}} & \rightarrow \mathbb{R}^{\textcolor{red}{n}} \end{array} \quad \begin{array}{l} \swarrow \\ \text{new} \\ \searrow \end{array}$$

- Required Properties:

- Diagonality: $f(t, t, \dots, t) = F(t)$
- Symmetry: $f(t_1, t_2, \dots, t_d) = f(t_{\pi(1)}, t_{\pi(2)}, \dots, t_{\pi(d)})$
for all permutations of indices π
- Multi-affine: $\sum \alpha_k = 1$

$$\begin{aligned} &\Rightarrow f\left(t_1, t_2, \dots, \sum \alpha_k t_i^{(k)}, \dots, t_d\right) \\ &= \alpha_1 f\left(t_1, t_2, \dots, t_i^{(1)}, \dots, t_d\right) + \dots + \alpha_n f\left(t_1, t_2, \dots, t_i^{(n)}, \dots, t_d\right) \end{aligned}$$

Generalizations

Blossoms with points as arguments:

- Polar form of degree d with points as input and output:

$$\begin{array}{lcl} F: & \mathbb{R}^m & \xrightarrow{\quad} \mathbb{R}^n \\ f: & \mathbb{R}^{d \times m} & \xrightarrow{\quad} \mathbb{R}^n \end{array} \quad \text{new}$$

- Required Properties:

- Diagonality: $f(\mathbf{t}, \mathbf{t}, \dots, \mathbf{t}) = F(\mathbf{t})$
- Symmetry: $f(\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_d) = f(\mathbf{t}_{\pi(1)}, \mathbf{t}_{\pi(2)}, \dots, \mathbf{t}_{\pi(d)})$
for all permutations of indices π
- Multi-affine: $\sum \alpha_k = 1$

$$\begin{aligned} &\Rightarrow f\left(\mathbf{t}_1, \mathbf{t}_2, \dots, \sum \alpha_k \mathbf{t}_i^{(k)}, \dots, \mathbf{t}_d\right) \\ &= \alpha_1 f\left(\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_i^{(1)}, \dots, \mathbf{t}_d\right) + \dots + \alpha_n f\left(\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_i^{(n)}, \dots, \mathbf{t}_d\right) \end{aligned}$$

Generalizations

Vector arguments

- We will have to distinguish between *points* and *vectors* (differences of points)
- Use “hat” notation $\hat{v} = p - q$ to denote vectors
- Also defined in the one dimensional case (vectors in \mathbb{R})
- One vector: $\hat{1} = 1 - 0$, $\hat{\mathbf{1}} = [1, \dots 1]^T - \mathbf{0}$
- Define shorthand notation (recursive) to define vectors in polar form:

$$\underbrace{f(t_1, \dots, t_{n-k})}_{n-k} \underbrace{(\hat{v}_1, \dots, \hat{v}_k)}_k := \underbrace{f(t_1, \dots, t_{n-k}, p_1)}_{n-k} \underbrace{(\hat{v}_2, \dots, \hat{v}_k)}_{k-1} - \underbrace{f(t_1, \dots, t_{n-k}, q_1)}_{n-k} \underbrace{(\hat{v}_2, \dots, \hat{v}_k)}_{k-1}$$

Properties

Derivatives of blossoms:

$$f(t_1, \dots, t_n) = \sum_{i=0}^n c_i \binom{n}{i}^{-1} \sum_{\substack{S \subseteq \{1 \dots n\}, \\ |S|=i}} \prod_{j \in S} t_j$$

- The c_i are related to the derivatives at $t = 0$
- Hence: $c_k = \frac{1}{k!} \frac{d^k}{dt^k} F(0) = \binom{n}{k} f(\underbrace{0, \dots, 0}_{n-k}, \underbrace{\hat{1}, \dots, \hat{1}}_k)$
- In general: $\frac{d^k}{dt^k} F(t) = \frac{n!}{(n-k)!} f(\underbrace{t, \dots, t}_{n-k}, \underbrace{\hat{1}, \dots, \hat{1}}_k)$

Example

$$f(t_1, t_2, t_3) = c_0 + c_1 \frac{t_1 + t_2 + t_3}{3} + c_2 \frac{t_1 t_2 + t_1 t_3 + t_2 t_3}{3} + c_3 t_1 t_2 t_3$$

$$f'(t) = \frac{3!}{2!} \left[\left(c_0 + c_1 \frac{1 + t + t}{3} + c_2 \frac{1t + tt + 1t}{3} + c_3 1tt \right) - \left(c_0 + c_1 \frac{0 + t + t}{3} + c_2 \frac{tt}{3} \right) \right]$$

$$= 3 \left(c_1 \frac{1}{3} + c_2 \frac{2t}{3} + c_3 tt \right)$$

$$= 3c_3 t^2 + 2c_2 t + c_1$$

Continuity Condition

Theorem: continuity condition for polynomials

The following statements are equivalent:

1. F and G are C^k -continuous at t

$$2. \quad \forall t_1, \dots, t_k: f(t, \dots, t, t_1, \dots, t_k) = g(t, \dots, t, t_1, \dots, t_k)$$

$$3. \quad f(\underbrace{t, \dots, t}_{k\text{-times}}, \hat{1}, \dots, \hat{1}) = g(\underbrace{t, \dots, t}_{k\text{-times}}, \hat{1}, \dots, \hat{1})$$

$$\begin{aligned} 2 \Leftrightarrow 3: \quad f(t, \dots, t, t_1) &= f(t, \dots, t, t_1 - 0) \\ &= t_1 (f(t, \dots, t, 1) - f(t, \dots, t, 0)) \\ &= t_1 f(t, \dots, t, \hat{1}) \end{aligned}$$

Continuity Condition

Examples:


- $\forall t_1, t_2, t_3: f(t_1, t_2, t_3) = g(t_1, t_2, t_3) \Rightarrow$ same curve
- $\forall t_1, t_2: f(t_1, t_2, t) = g(t_1, t_2, t) \Rightarrow C^2$ at t
- $\forall t_1: f(t_1, t, t) = g(t_1, t, t) \Rightarrow C^1$ at t
- $f(t, t, t) = g(t, t, t) \Rightarrow C^0$ at t

Raising the Degree

Raising the degree of a blossom:

- Can we directly construct a polar form with degree elevated by one from a lower degree one, without changing the polynomial?
- [Other than transforming to monomials, adding $0 \cdot t^{d+1}$, and transforming back?]

Solution:

- Given: $f(t_1, \dots, t_d)$
- We obtain: $f^{(+1)}(t_1, \dots, t_{d+1}) = \frac{1}{d+1} \sum_{i=1}^{d+1} f(t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_{d+1})$
 leave out t_i

Raising the Degree

Proof:

$$\begin{aligned}\forall t: f^{(+1)}(t, \dots, t) &= \frac{1}{d+1} \sum_{i=1}^{d+1} f(t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_{d+1}) \Big|_{t_1=\dots=t_{d+1}=t} \\ &= \frac{1}{d+1} \sum_{i=1}^{d+1} f(t, \dots, t) \\ &= f(t, \dots, t)\end{aligned}$$

$$\Rightarrow F^{(+1)}(t) = F(t)$$

Polars and Control Points

Interpretation (Examples):

- Multi-variate function: $f(t_1, t_2, t_3)$
 - Describes a point depending on three parameters
 - Where $f(t_1, t_2, t_3)$ moves for changing (t_1, t_2, t_3)
(think of storing monomial coefficients inside)
- Polynomial value: $f(1.5, 1.5, 1.5)$
 - One value of the polynomial curve: $F(1.5)$
- Off-curve points: $f(1, 2, 3)$
 - Evaluate points not necessarily on the polynomial curve
 - Question: what meaning do various off-curve points have?
 - We will use off-curve points as control points

Polars and Control Points

Interpretation (Examples):

- Specifying: $f(t_1, t_2, t_3)$
 - Assume, f is not known yet
 - We want to determine a polar (i.e. a polynomial)
- On-curve points:
 $\{f(0,0,0) = \mathbf{x}_0, f(1,1,1) = \mathbf{x}_1, f(2,2,2) = \mathbf{x}_2, f(3,3,3) = \mathbf{x}_3\}$
 - Degree d polynomial has $d + 1$ degrees of freedom
 - We know already how to do polynomial interpolation
- Off-curve points:
 $\{f(1, 2, 3) = \mathbf{x}_{123}, f(2,3,4) = \mathbf{x}_{234}\}$
 - We can also use off-curve points to specify the polar/polynomial
 - This is the main motivation for the whole formalism

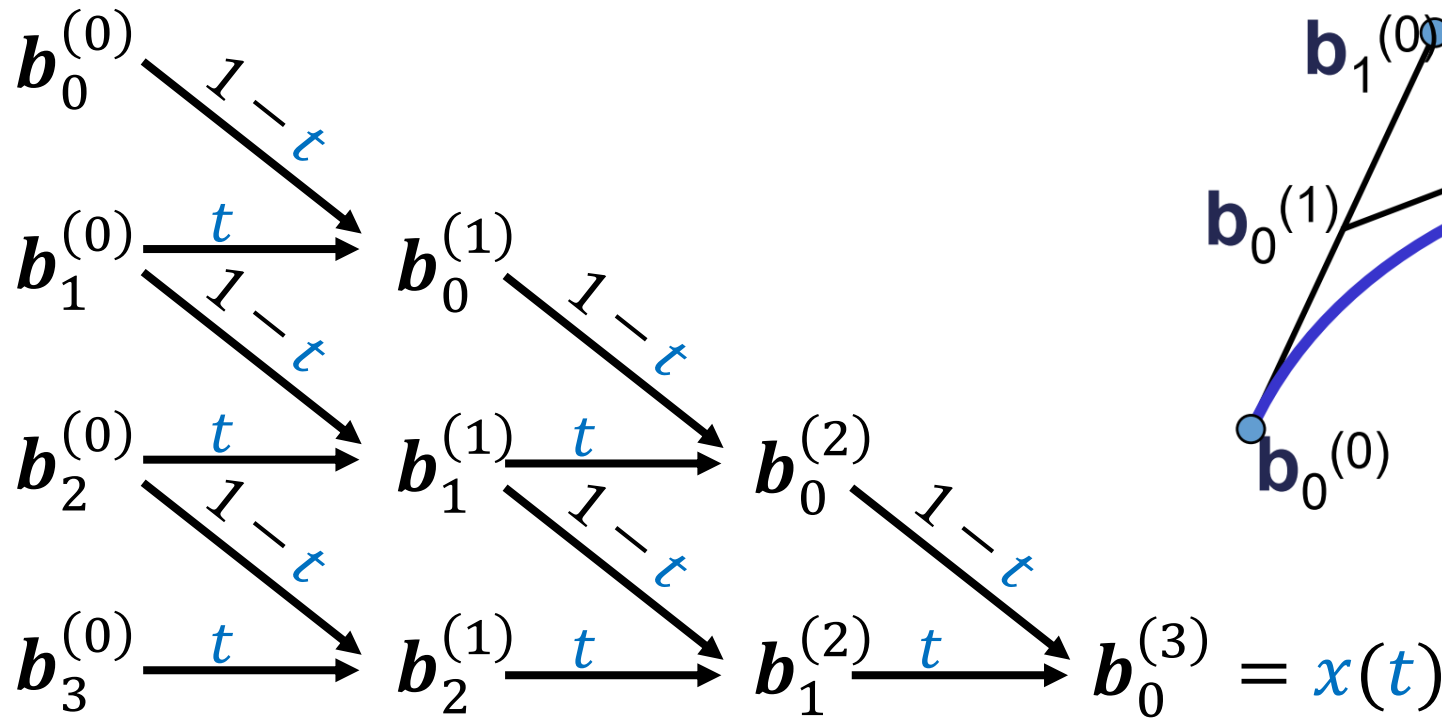
Polar Forms & Blossoms

Bézier Splines

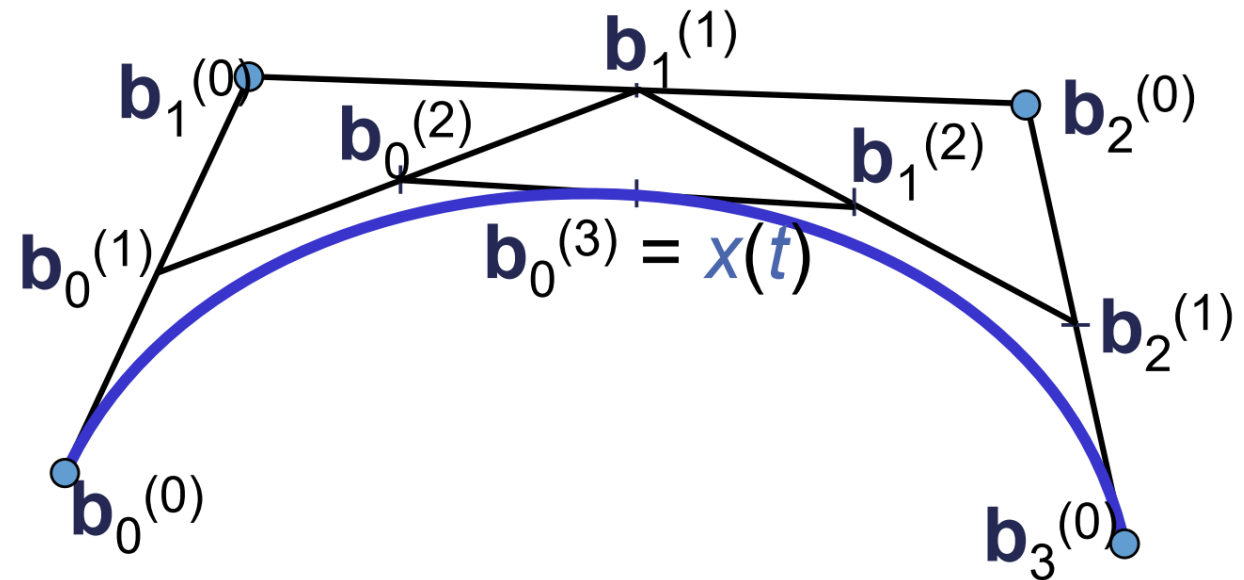
De Casteljau algorithm (from earlier)

Repeated convex combination of control points

$$\mathbf{b}_i^{(r)} = (1 - t)\mathbf{b}_i^{(r-1)} + t\mathbf{b}_{i+1}^{(r-1)}$$



de Casteljau scheme



De Casteljau Algorithm

The de Casteljau algorithm is simple to state with blossoms:

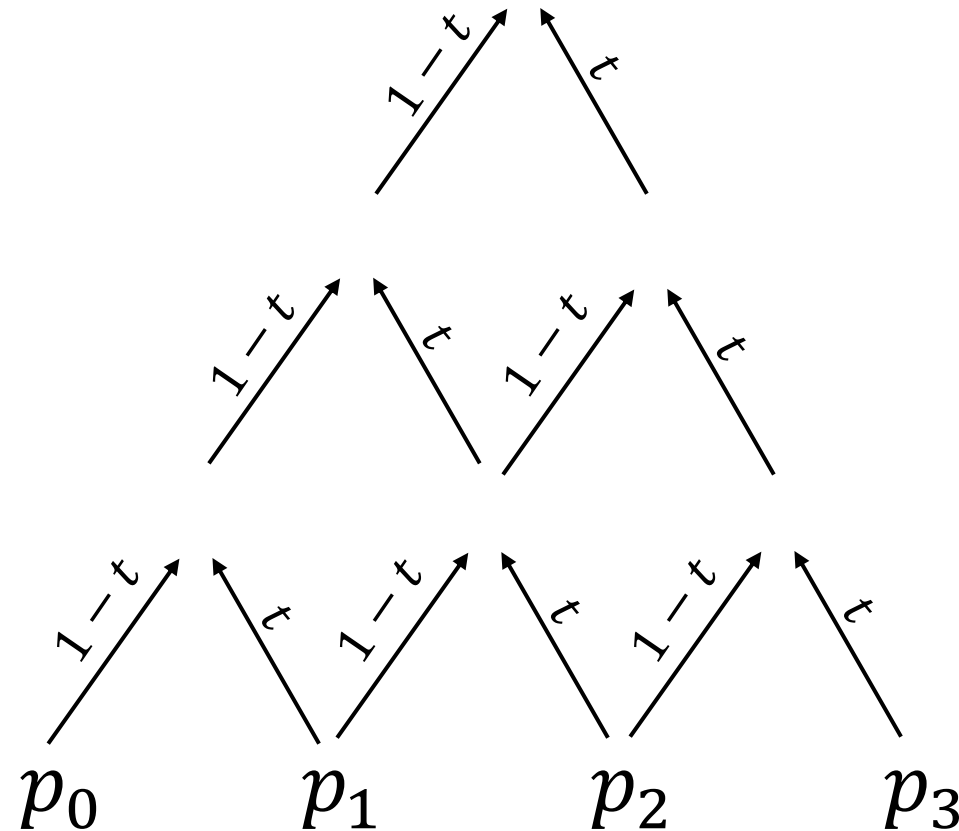
- We just have to exchange the labels
- Then use the multi-affinity property in order to compute intermediate points
- With this view, we can easily show that the de Casteljau algorithm is equivalent to the formulation based on Bernstein polynomials

Key observation

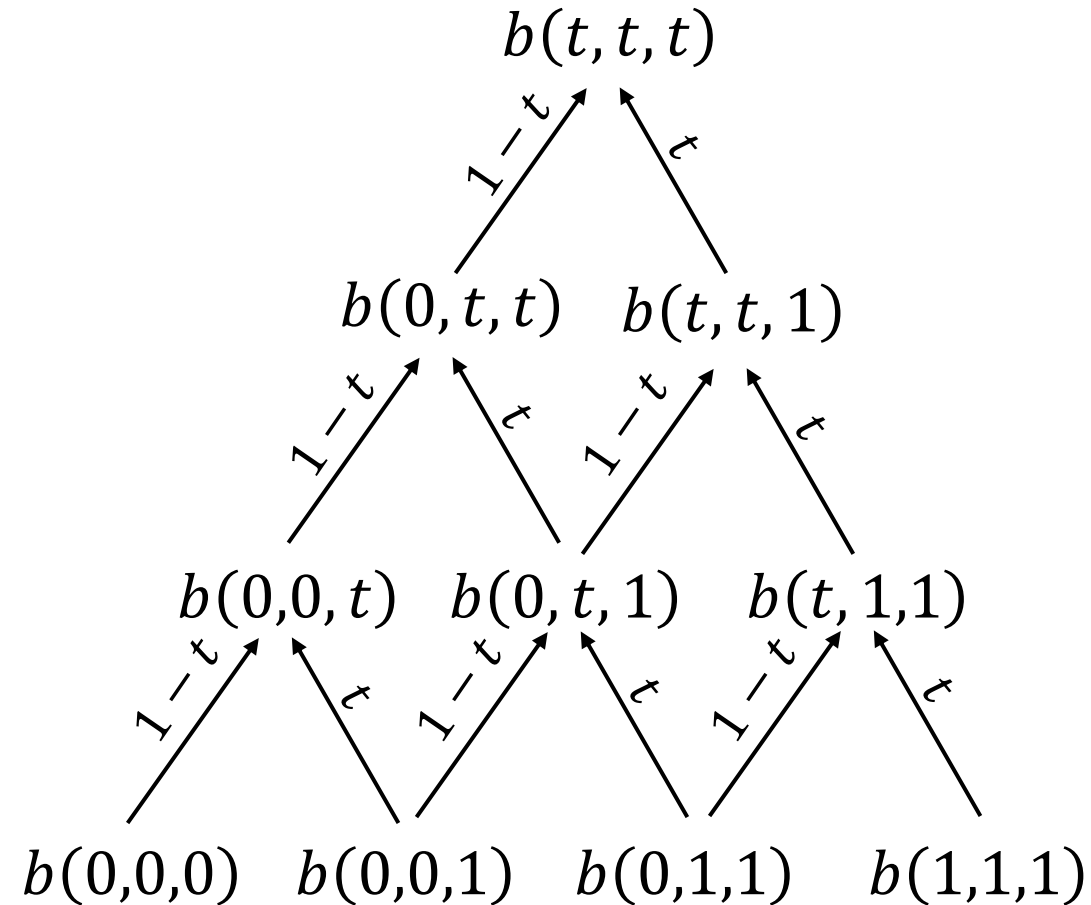
$$\begin{array}{ccc} & b(0, 0(1 - t) + 1t, 1) & \\ \nearrow^{1-t} & & \nwarrow_t \\ b(0, 0, 1) & & b(0, 1, 1) \end{array}$$

De Castljam Algorithms for Bézier Curves

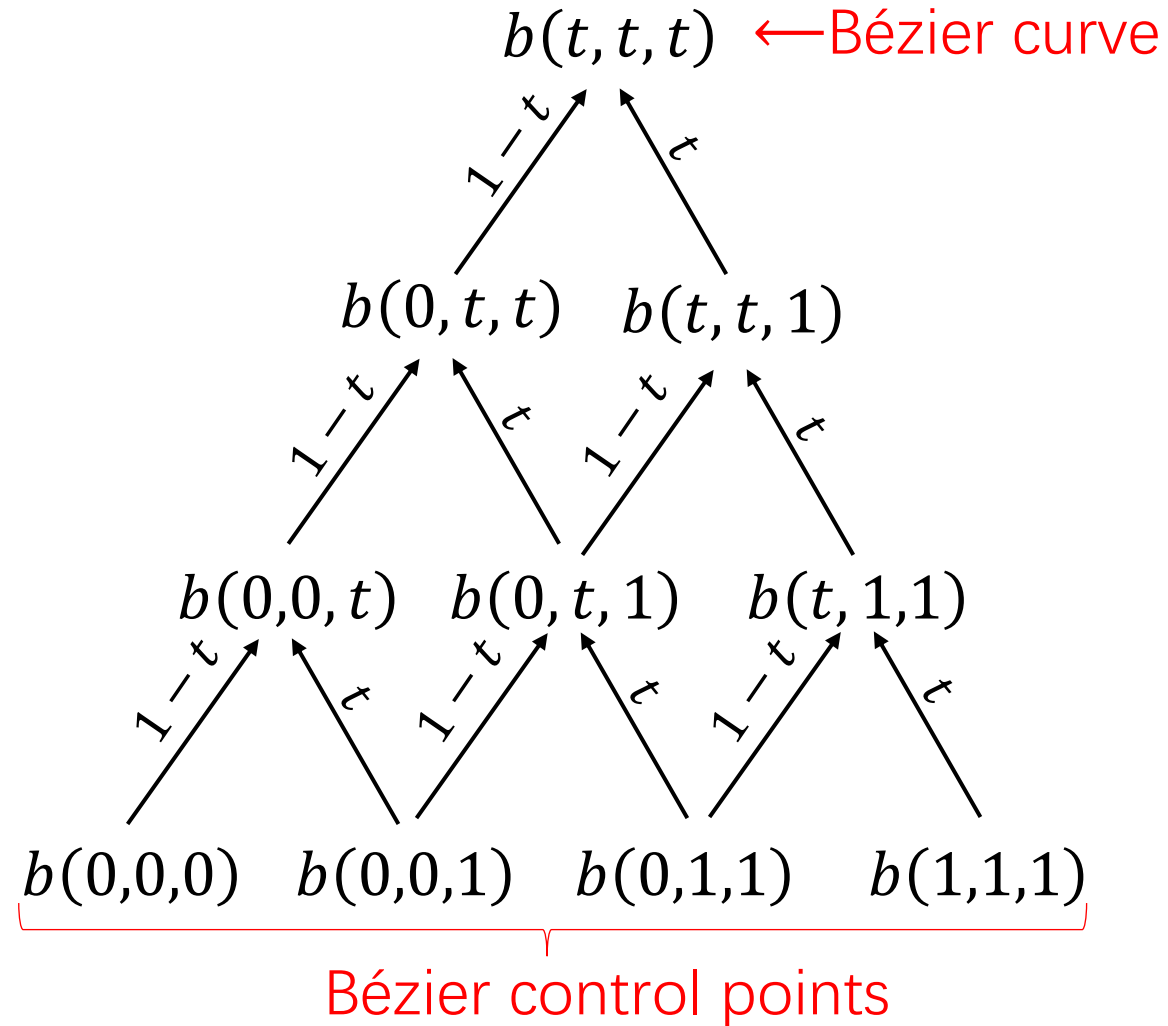
$$(1 - t)^3 p_0 + 3(1 - t)^2 t p_1 + 3(1 - t) t^2 p_2 + t^3 p_3$$



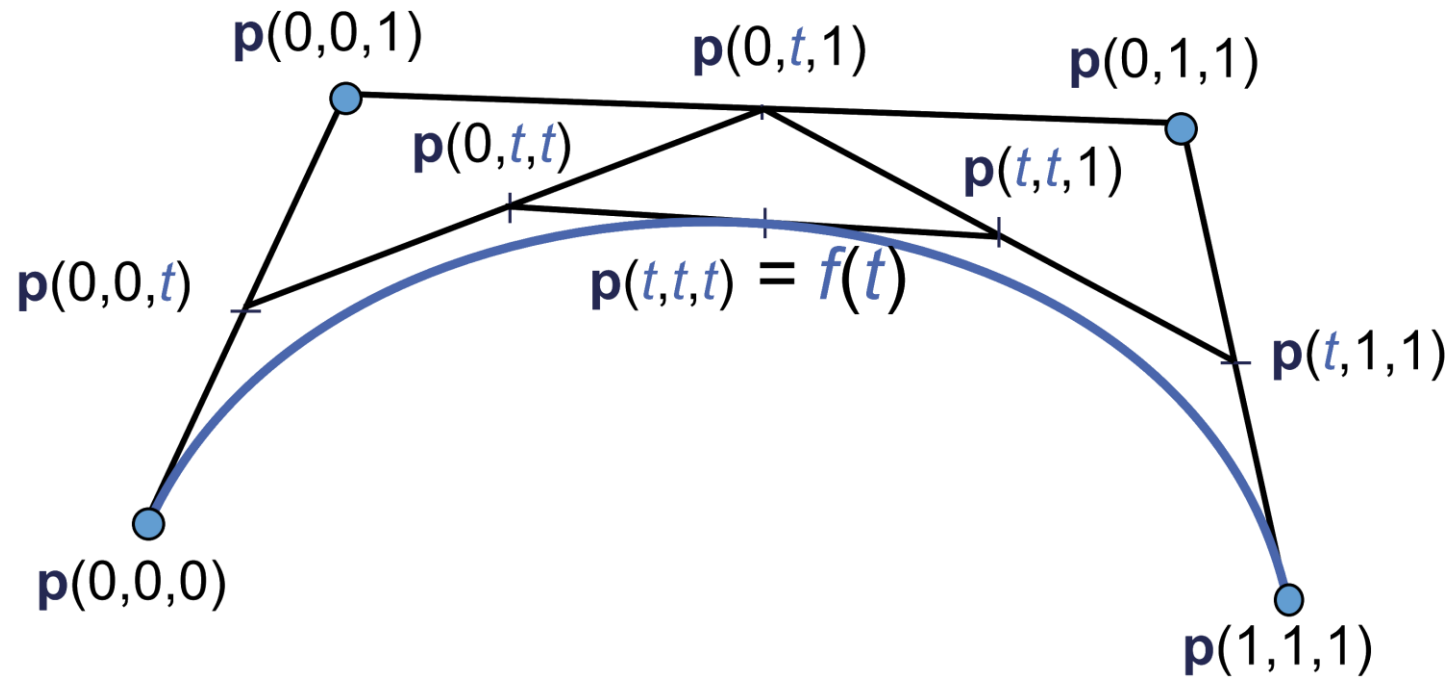
De Castljam Algorithms for Bézier Curves



De Castljam Algorithms for Bézier Curves



De Casteljau (Polar forms)



Bézier control points: $p(0,0,0)$, $p(0,0,1)$, $p(0,1,1)$, $p(1,1,1)$

Analysis

Transforming a polar to the Bernstein basis:

$$f(t, \dots, t) = (1 - t)f(t, \dots, t, 0) + tf(t, \dots, t, 1)$$

Analysis

Transforming a polar to the Bernstein basis:

$$\begin{aligned} f(t, \dots, t) &= (1 - t)f(t, \dots, t, 0) + tf(t, \dots, t, 1) \\ &= (1 - t)[(1 - t)f(t, \dots, t, 0, 0) + tf(t, \dots, 0, 1)] + t[(1 - t)f(t, \dots, t, 1, 0) + tf(t, \dots, t, 1, 1)] \end{aligned}$$

Analysis

Transforming a polar to the Bernstein basis:

$$\begin{aligned} f(t, \dots, t) &= (1 - t)f(t, \dots, t, 0) + tf(t, \dots, t, 1) \\ &= (1 - t)[(1 - t)f(t, \dots, t, 0, 0) + tf(t, \dots, 0, 1)] + t[(1 - t)f(t, \dots, t, 1, 0) + tf(t, \dots, t, 1, 1)] \\ &= (1 - t)^2 f(t, \dots, t, 0, 0) + 2t(1 - t)tf(t, \dots, 0, 1) + t^2 f(t, \dots, t, 1, 1) \end{aligned}$$

Analysis

Transforming a polar to the Bernstein basis:

$$\begin{aligned}f(t, \dots, t) &= (1 - t)f(t, \dots, t, 0) + tf(t, \dots, t, 1) \\&= (1 - t)[(1 - t)f(t, \dots, t, 0, 0) + tf(t, \dots, 0, 1)] + t[(1 - t)f(t, \dots, t, 1, 0) + tf(t, \dots, t, 1, 1)] \\&= (1 - t)^2 f(t, \dots, t, 0, 0) + 2t(1 - t)tf(t, \dots, 0, 1) + t^2 f(t, \dots, t, 1, 1) \\&= \dots\end{aligned}$$

$$= \sum_{i=0}^n \binom{n}{i} t^i (1 - t)^{n-i} f(\underbrace{0, \dots, 0}_{n-i}, \underbrace{1, \dots, 1}_i)$$

Analysis

De Castlejau Algorithm: Perform this in reverse order

- Bézier points:
$$p_i^{(0)}(t) = f(\underbrace{0, \dots, 0}_{d-i}, \underbrace{1, \dots, 1}_i)$$
- Intermediate points:
$$p_i^{(j)}(t) = f(\underbrace{0, \dots, 0}_{d-i-j}, \underbrace{1, \dots, 1}_i, \underbrace{t, \dots, t}_j)$$
- Recursive computation:
$$\begin{aligned} p_i^{(j)}(t) &= f(\underbrace{0, \dots, 0}_{d-i-j}, \underbrace{t, \dots, t}_j, \underbrace{1, \dots, 1}_i) \\ &= (1-t)f(\underbrace{0, \dots, 0}_{d-i-j+1}, \underbrace{t, \dots, t}_{j-1}, \underbrace{1, \dots, 1}_i) + tf(\underbrace{0, \dots, 0}_{d-i-j}, \underbrace{t, \dots, t}_{j-1}, \underbrace{1, \dots, 1}_{i+1}) \\ &= (1-t)p_i^{(j-1)}(t) + tp_{i+1}^{(j-1)}(t) \end{aligned}$$

Consequence: Bernstein / de Casteljau lead to the same result

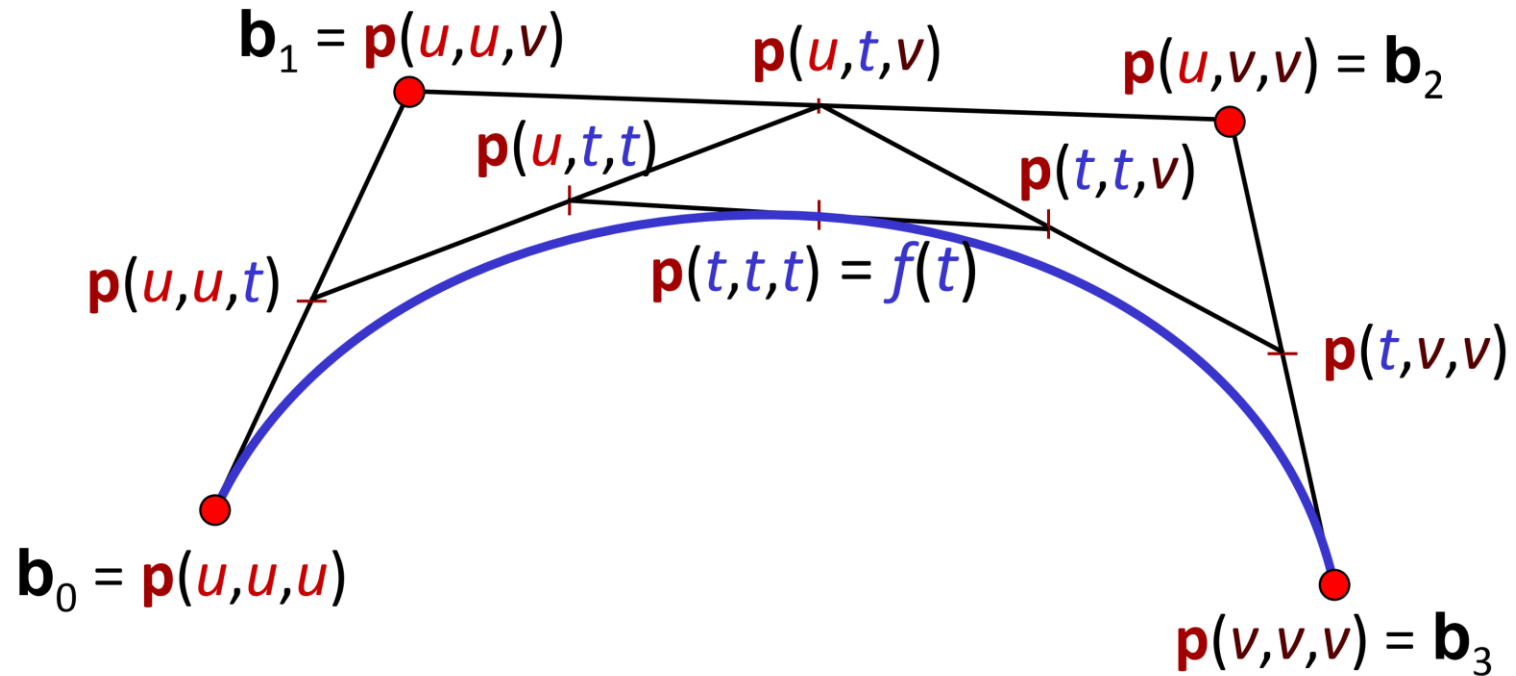
Generalized Parameter Intervals

- Let $f(t)$ be a Bézier curve of degree d over the domain $t \in [u, v]$
- Let p be the polar form of f
- Then the Bézier points of f are given in polar form as:
 - $b_i = p(u, \dots, u, v, \dots, v)$

Example for a cubic Bézier curve:

$$p(u, u, u), p(u, u, v), p(u, v, v), p(v, v, v)$$

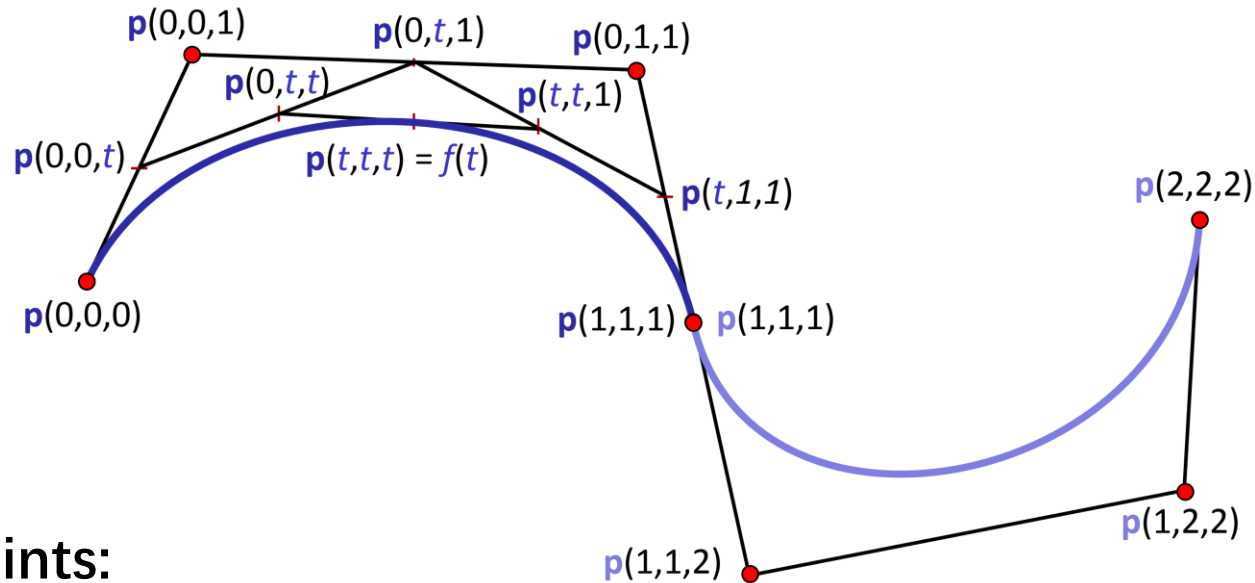
Generalized Parameter Intervals



Example for a cubic Bézier curve:

$$\mathbf{p}(u, u, u), \mathbf{p}(u, u, v), \mathbf{p}(u, v, v), \mathbf{p}(v, v, v)$$

Multiple Segments



Bézier Control points:

$$p(0,0,0), p(0,0,1), p(0,1,1), p(1,1,1) = p(1,1,1), p(1,1,2), p(1,2,2), p(2,2,2)$$

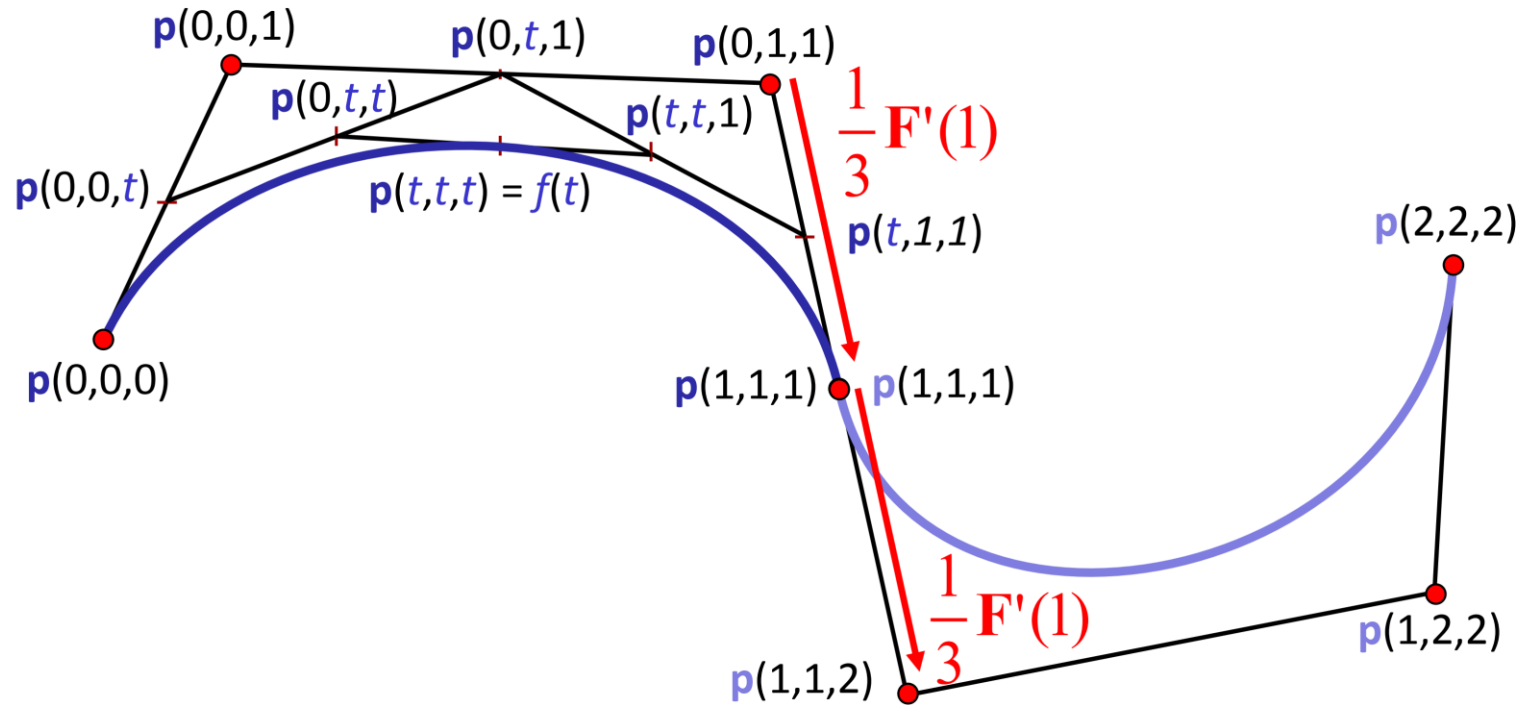
Two Curve Segments:

$$\{p(0,0,0), p(0,0,1), p(0,1,1), p(1,1,1)\}, \{p(1,1,1), p(1,1,2), p(1,2,2), p(2,2,2)\}$$

Remark: no intersection between different segments

(e.g.: combination of $p(0,1,1)$ and $p(2,1,1)$ is not defined)

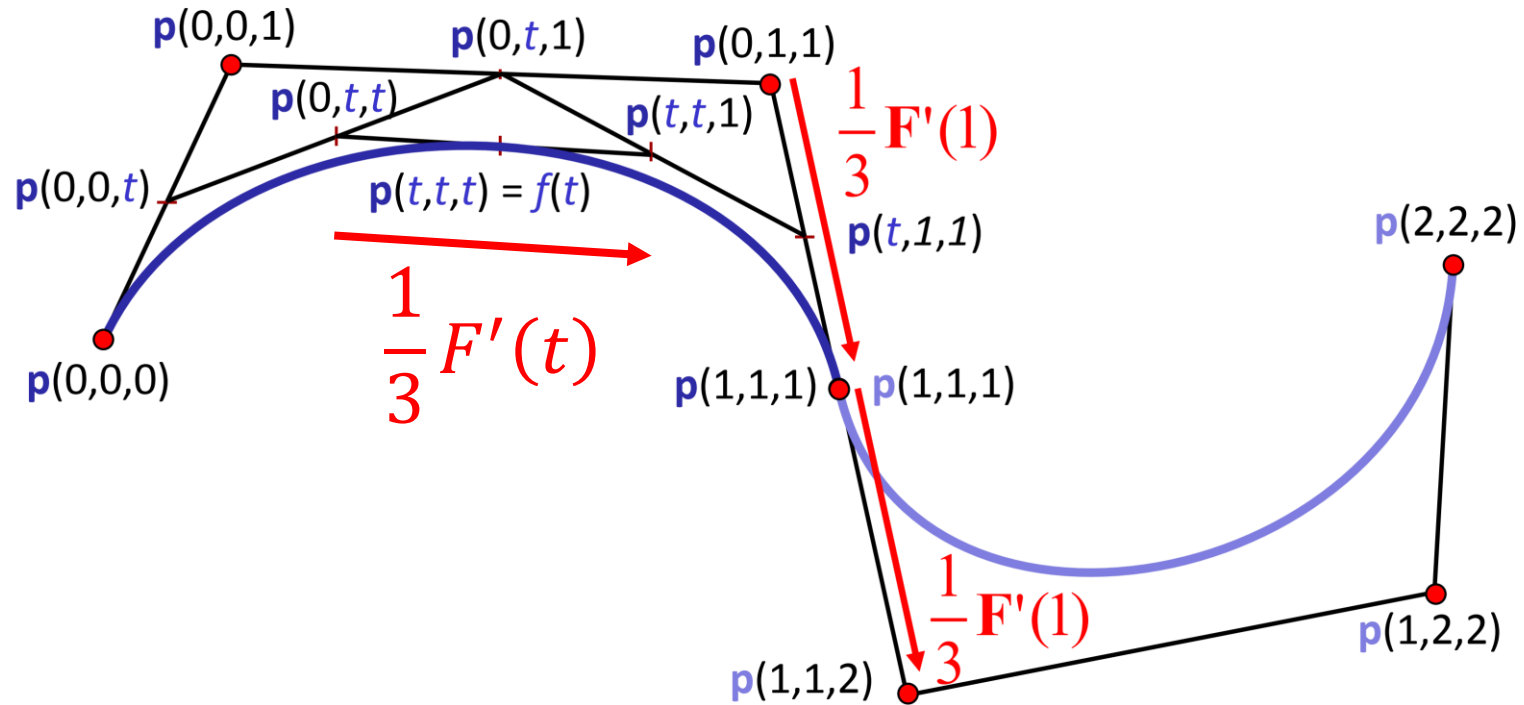
More Observations



Derivatives:

- $\frac{d}{dt} F(t) = df(t, \dots, t, \hat{1}) = d(f(t, \dots, t, 1) - f(t, \dots, t, 0))$ (degree d)
- C^1 continuity condition follows

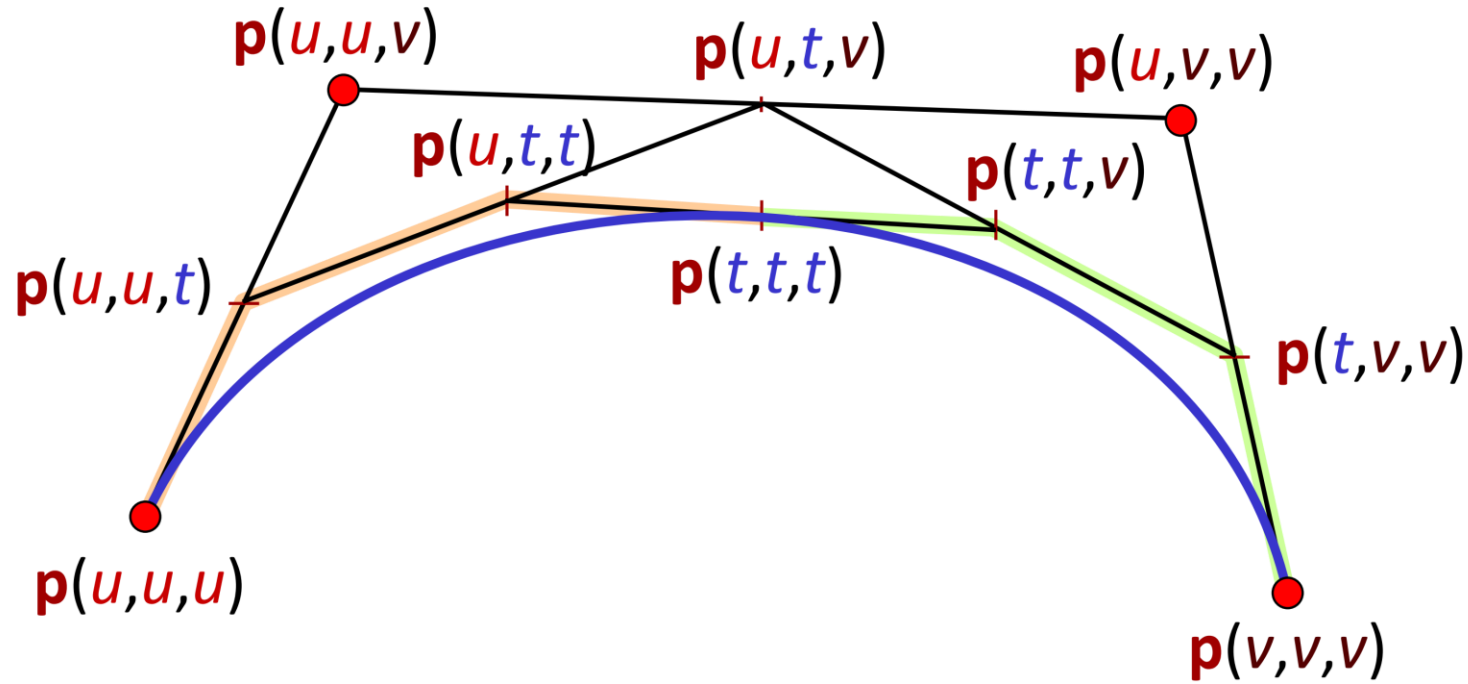
More Observations



Derivatives:

- de Casteljau algorithm computes tangent vectors at any points as a byproduct
- Proportional to last line segment that is bisected

More Observations



Subdivision:

- After each de Casteljau step, we obtain two new control polygons left and right of $f(t)$ describing the same curve.
- We can divide a segment into two
- Recursive subdivision can be used for rendering

Observations

Remark: The de Casteljau algorithm for computing

- Derivatives
 - At end points
 - At inner points t
- Subdivisions

Hold for Bézier curves of arbitrary degree $d \geq 1$


(General degree derivatives: $F'(t)/d$)

More Bézier Curve Properties...

General degree elevation

- Increase the degree of a Bézier curve segment by one
- What are the new control points?

Polar forms:

- Old curve: $b(t_1, \dots, t_d)$
- New curve: $b^{(+1)}(t_1, \dots, t_{d+1}) = \frac{1}{d+1} \sum_{i=1}^{d+1} b(t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_{d+1})$
 leave out t_i

Degree Elevation

$$\mathbf{b}^{(+1)}(0, \dots, 0) = \frac{1}{d+1} \sum_{i=1}^{d+1} \mathbf{b}(0, \dots, 0) = \mathbf{b}(0, \dots, 0)$$

$$\mathbf{b}^{(+1)}(1, \dots, 0) = \frac{1}{d+1} \mathbf{b}(0, \dots, 0) + \frac{d}{d+1} \mathbf{b}(1, 0, \dots, 0)$$

$$\mathbf{b}^{(+1)}(1, 1, 0, \dots, 0) = \frac{2}{d+1} \mathbf{b}(0, \dots, 0) + \frac{d-1}{d+1} \mathbf{b}(1, 1, 0, \dots, 0)$$

$$\mathbf{b}^{(+1)}(1, 1, 1, \dots, 1, 0) = \frac{d}{d+1} \mathbf{b}(1, \dots, 1, 0) + \frac{1}{d+1} \mathbf{b}(1, \dots, 1)$$

$$\mathbf{b}^{(+1)}(1, \dots, 1) = \mathbf{b}(1, \dots, 1)$$

Degree Elevation

Result: new control points

$$\mathbf{p}_i^{(+1)} = \frac{i}{n+1} \mathbf{p}_{i-1} + \left(1 - \frac{i}{n+1}\right) \mathbf{p}_i, \quad i = 0, \dots, n+1 \text{ (zero points if out of range)}$$

Repeated degree elevation:

$$\mathbf{p}_i^{(+k)} = \sum_{j=1}^{d+1} \mathbf{p}_j \frac{\binom{d}{j} \binom{k}{i-j}}{\binom{d+k}{j}} \text{ (proof by induction)}$$

Repeating degree elevation let the control point converge to the Bézier curve in the limit

Change of basis, the easy way

- Given: Polynomial $p(t)$ of degree n in monomial form
- Wanted: coefficients of the same Bézier curve

Change of basis, the easy way

- Given: Polynomial $p(t)$ of degree n in monomial form
- Wanted: coefficients of the same Bézier curve
- Solution:

$$p(t) \rightarrow b(t_1, \dots, t_n)$$

$$\text{coefficients: } b(\underbrace{0, \dots, 0}_{n+1-k}, \underbrace{1, \dots, 1}_k)$$

- This is a direct implication of de Casteljau in polar form

Example

- Example: Bézier coefficients of $p(t) = 1 + 2t + 3t^2 - t^3$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ 2 \\ 3 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \\ 5/3 \\ 10/3 \\ 5 \end{pmatrix}$$

Example

- Using polar forms

$$b(t_0, t_1, t_2) = 1 + 2 \frac{t_0 + t_1 + t_2}{3} + 3 \frac{t_0 t_1 + t_1 t_2 + t_0 t_2}{3} - t_0 t_1 t_2$$


$$b(0,0,0) = 1, \quad b(0,0,1) = \frac{5}{3}, \quad b(0,1,1) = \frac{10}{3}, \quad b(1,1,1) = 5$$

Polar Forms & Blossoms

B-Splines

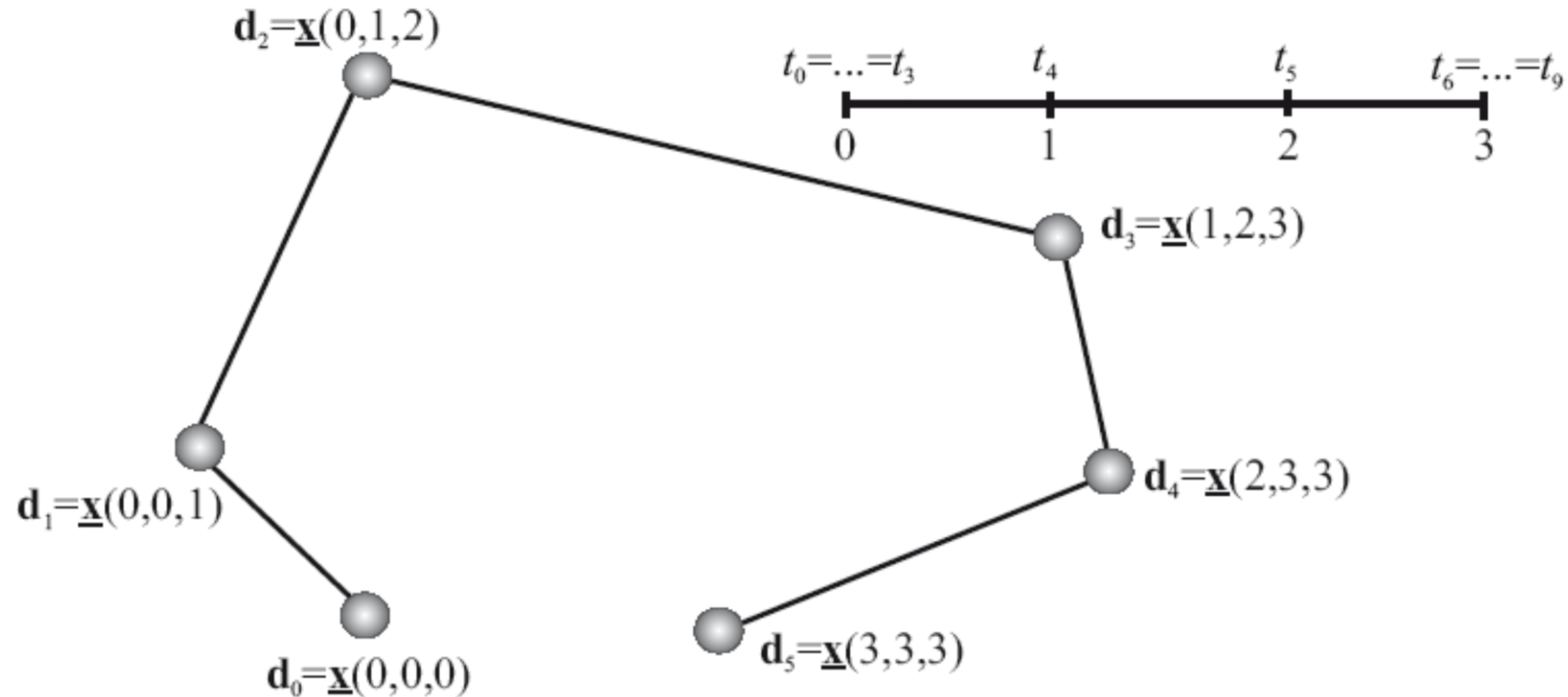
B-Spline Curves in Polar Form

An unique description in polar form exists for piecewise polynomial curves as well

- Given: B-Spline curve \mathbf{x} of order k
 - with knot vector $T = (t_0, \dots, t_{n+k})$
 - and de Boor points $\mathbf{d}_0, \dots, \mathbf{d}_n$
- Let $\underline{\mathbf{x}}$ be the polar form of \mathbf{x}
- Then the de Boor points of \mathbf{x} are given as:
 - $\mathbf{d}_i = \underline{\mathbf{x}}(t_{i+1}, \dots, t_{i+k-1})$
 *we just use consecutive knot values as blossom arguments*

B-Spline Curves in Polar form

- Example: $k = 4, n = 5$



De Boor Algorithm in Polar Form

de Boor algorithm in polar form

- To Define the curve value at t , we look for the relevant part of the de Boor polygon, i.e.,

→ the (partial) knot sequence

$$r_{k-1} \leq \cdots \leq r_1 < s_1 \leq \cdots \leq s_{k-1}$$

with $r_1 \leq t < s_1$

De Boor Algorithm in Polar Form

de Boor algorithm in polar form

- Then the intermediate points of the de Boor algorithm result are

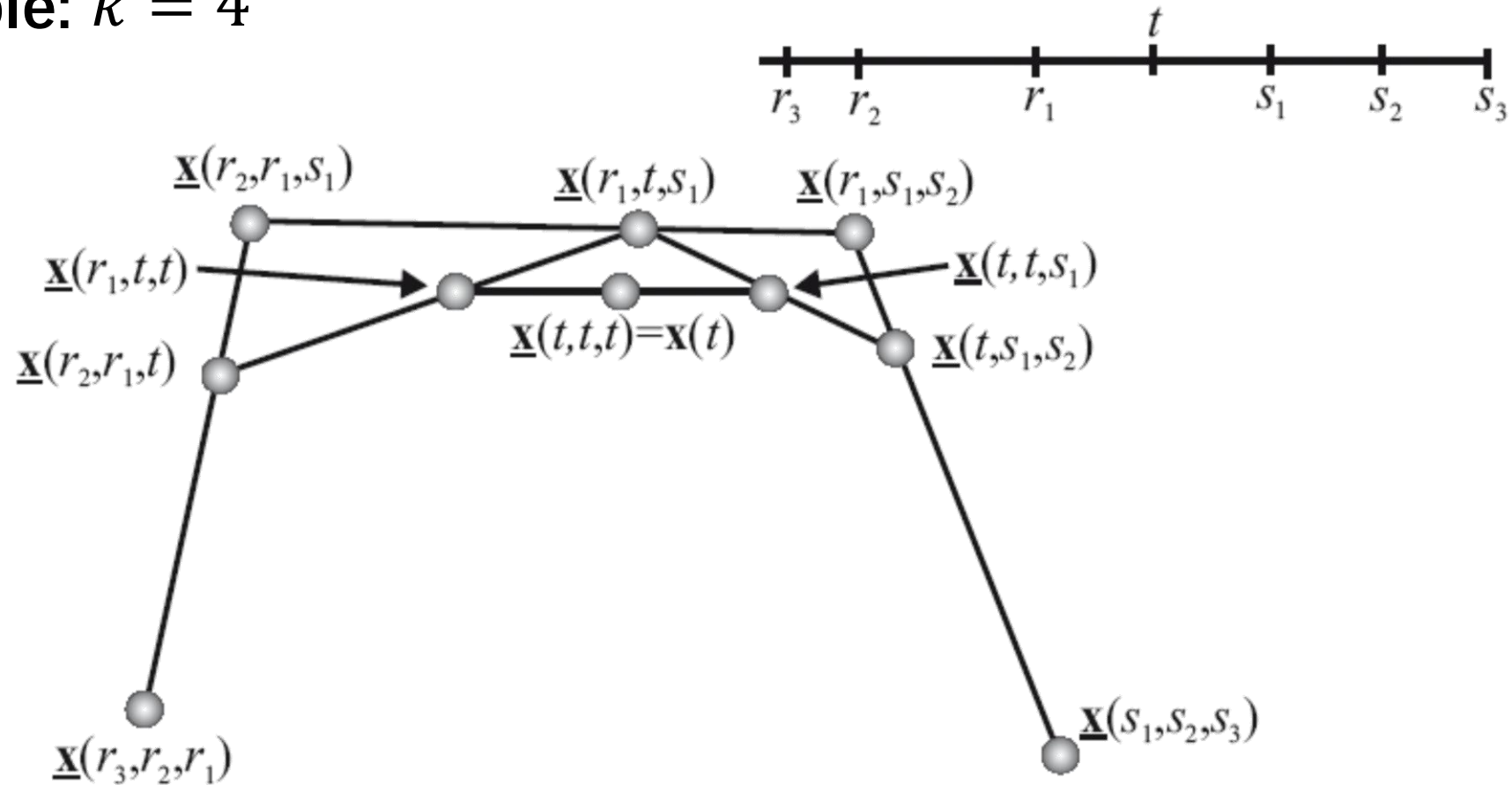
$$d_j^l(t) = \underline{x}(r_1, \dots, r_{k-1-l-j}, \underbrace{t, \dots, t}_l, s_1, \dots, s_j)$$

and the desired curve point is

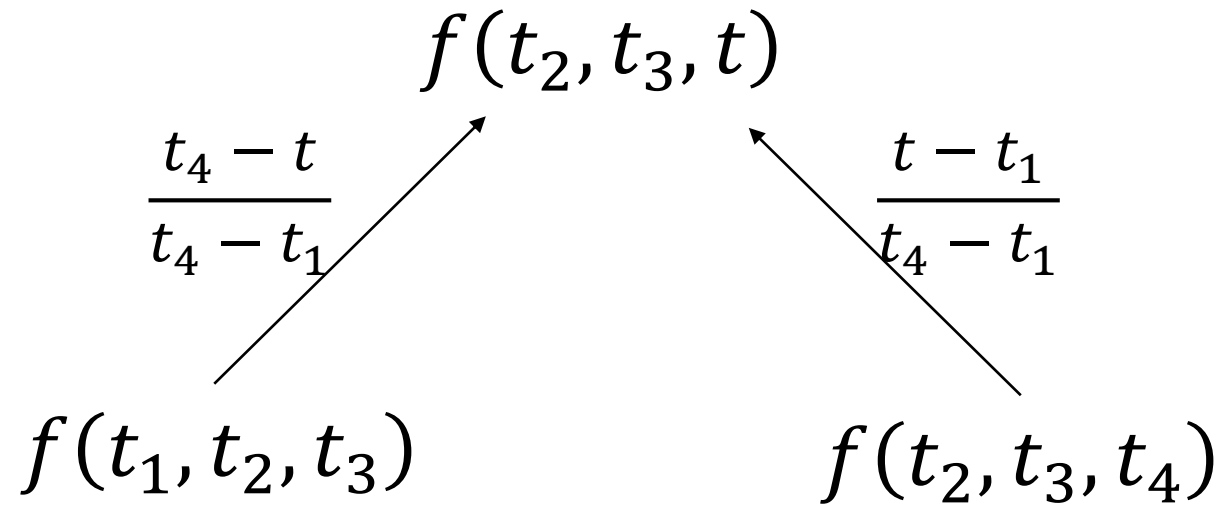
$$x(t) = \underline{x}(\underbrace{t, \dots, t}_{k-1})$$

De Boor Algorithm in Polar form

Example: $k = 4$

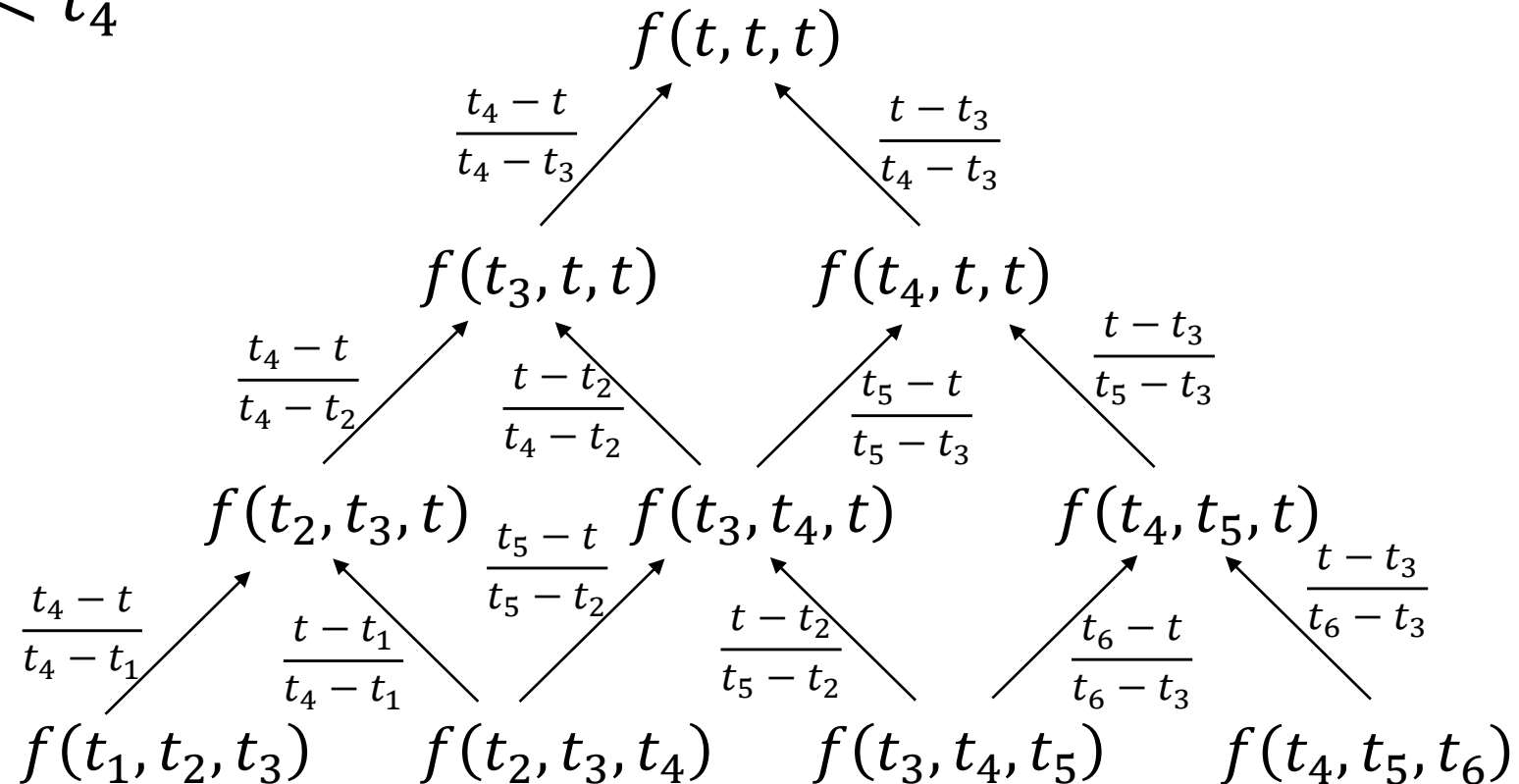


Key observation


$$\begin{array}{ccc} & f(t_2, t_3, t) & \\ \frac{t_4 - t}{t_4 - t_1} \nearrow & & \nwarrow \frac{t - t_1}{t_4 - t_1} \\ f(t_1, t_2, t_3) & & f(t_2, t_3, t_4) \end{array}$$

De Boor Alg. In Polar form

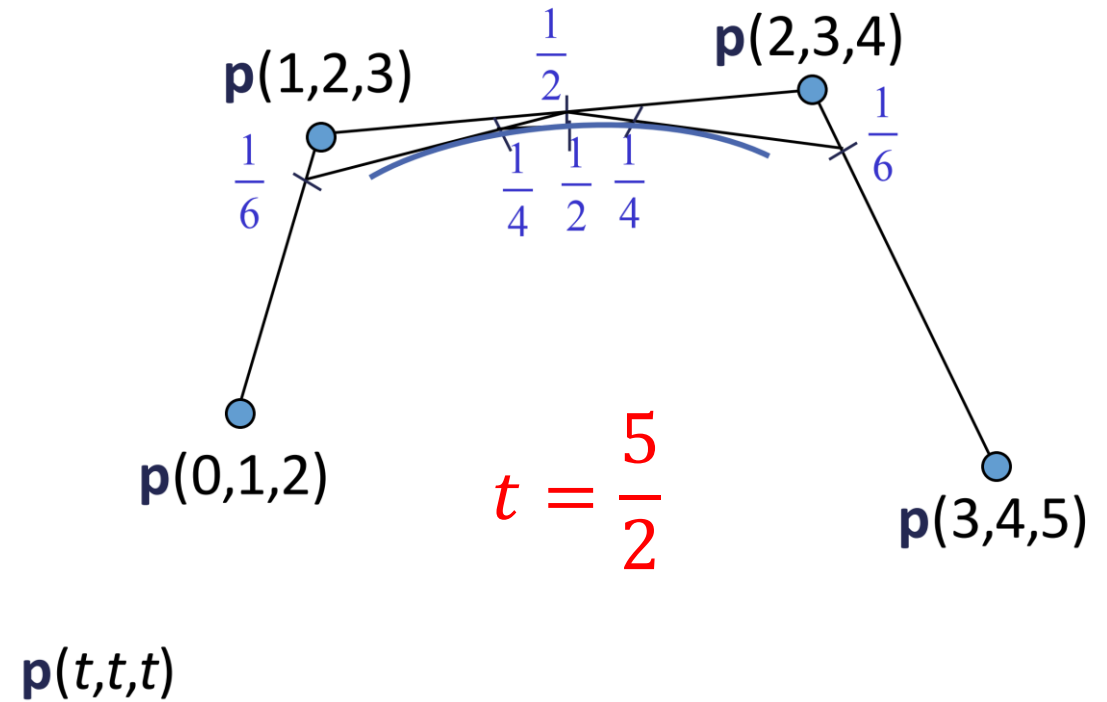
- For $t_3 \leq t < t_4$



De Boor Algorithm in Polar Form

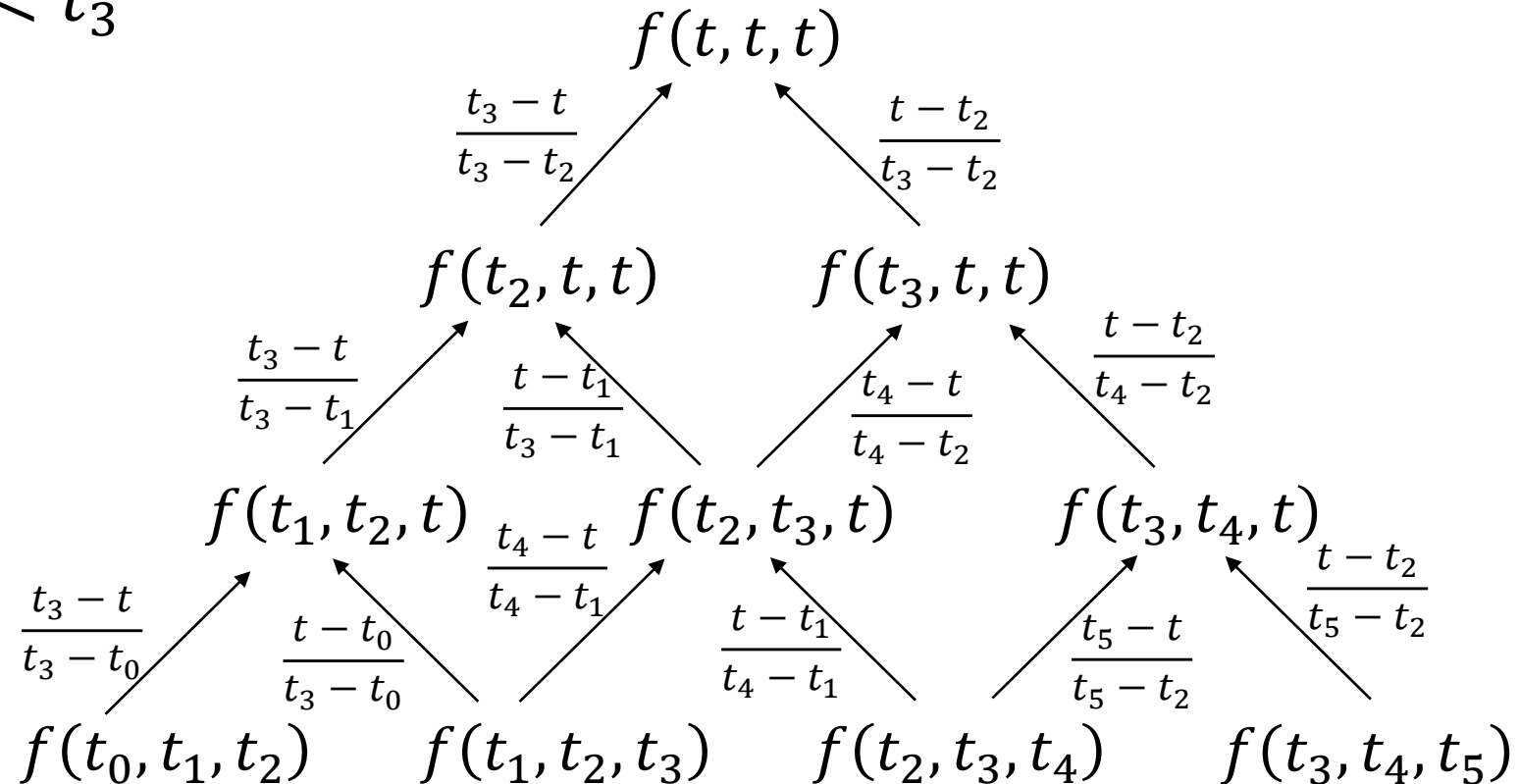
for $2 \leq t < 3$

$$\begin{aligned}
 & \frac{3-t}{3} \mathbf{p}(0,1,2) + \left[1 - \frac{3-t}{3}\right] \mathbf{p}(3,1,2) \\
 & \frac{4-t}{3} \mathbf{p}(1,2,3) + \left[1 - \frac{4-t}{3}\right] \mathbf{p}(4,2,3) \\
 & \frac{5-t}{3} \mathbf{p}(2,3,4) + \left[1 - \frac{5-t}{3}\right] \mathbf{p}(5,3,4) \\
 & \frac{3-t}{2} \mathbf{p}(t,1,2) + \left[1 - \frac{3-t}{2}\right] \mathbf{p}(t,3,2) \\
 & \frac{4-t}{2} \mathbf{p}(t,3,2) + \left[1 - \frac{4-t}{2}\right] \mathbf{p}(t,3,4) \\
 & \frac{3-t}{1} \mathbf{p}(t,t,2) + \left[1 - \frac{3-t}{1}\right] \mathbf{p}(t,t,3)
 \end{aligned}$$



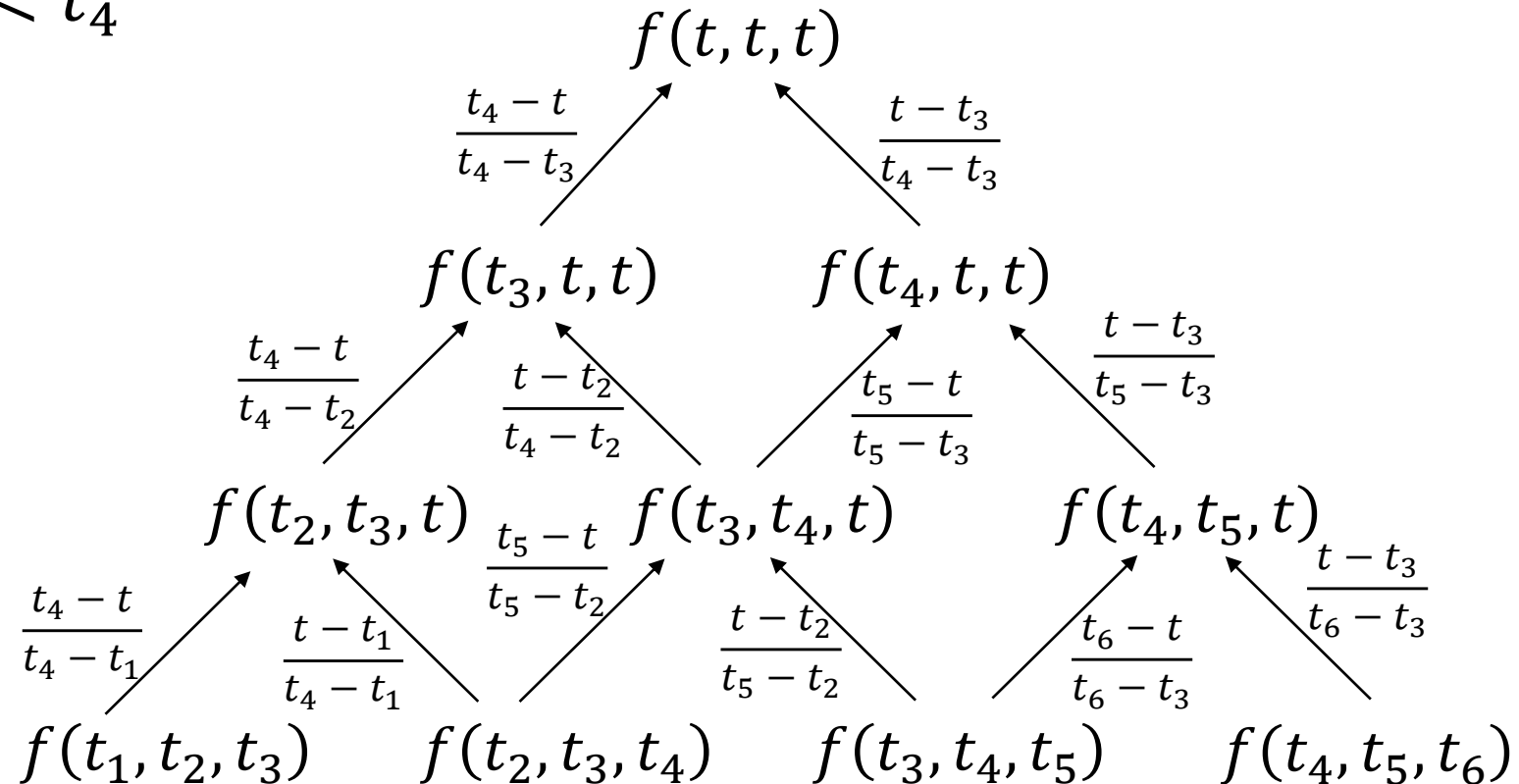
B-Splines in Polar form

- For $t_2 \leq t < t_3$

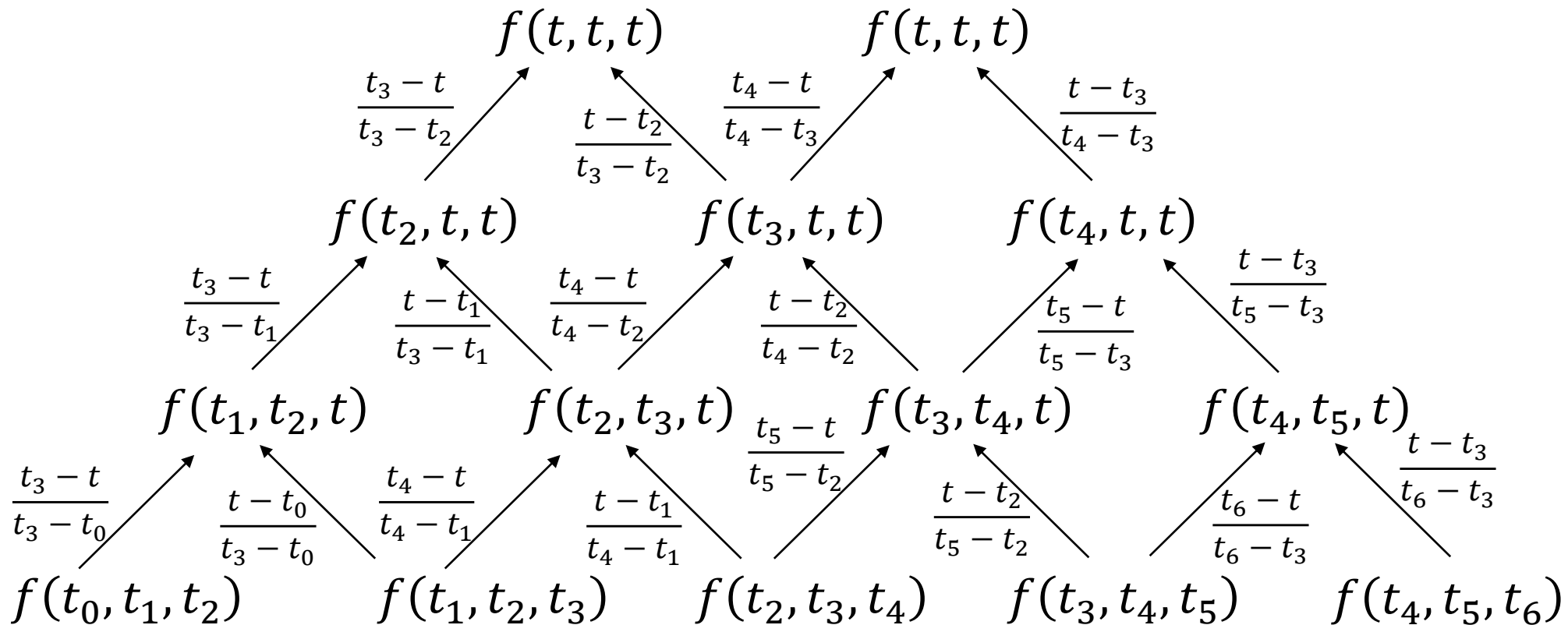


B-Splines in Polar form

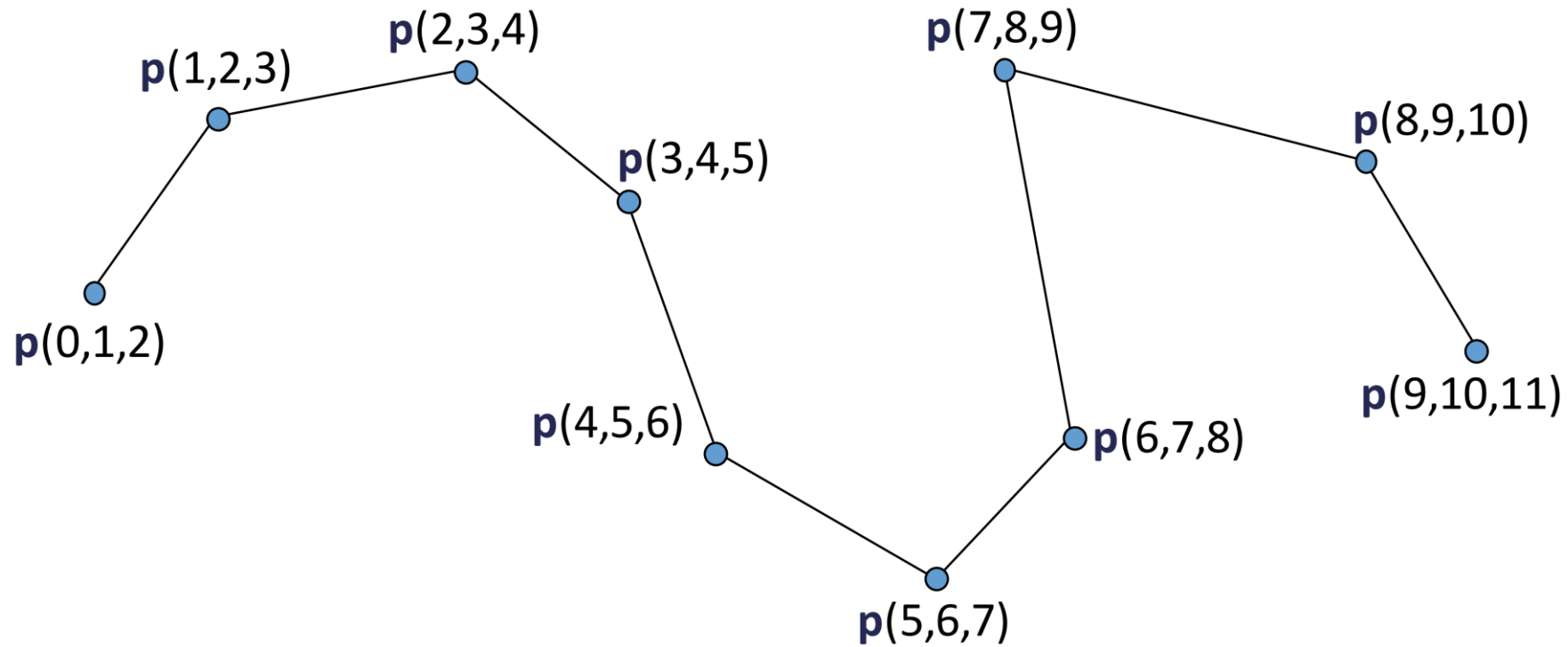
- For $t_3 \leq t < t_4$



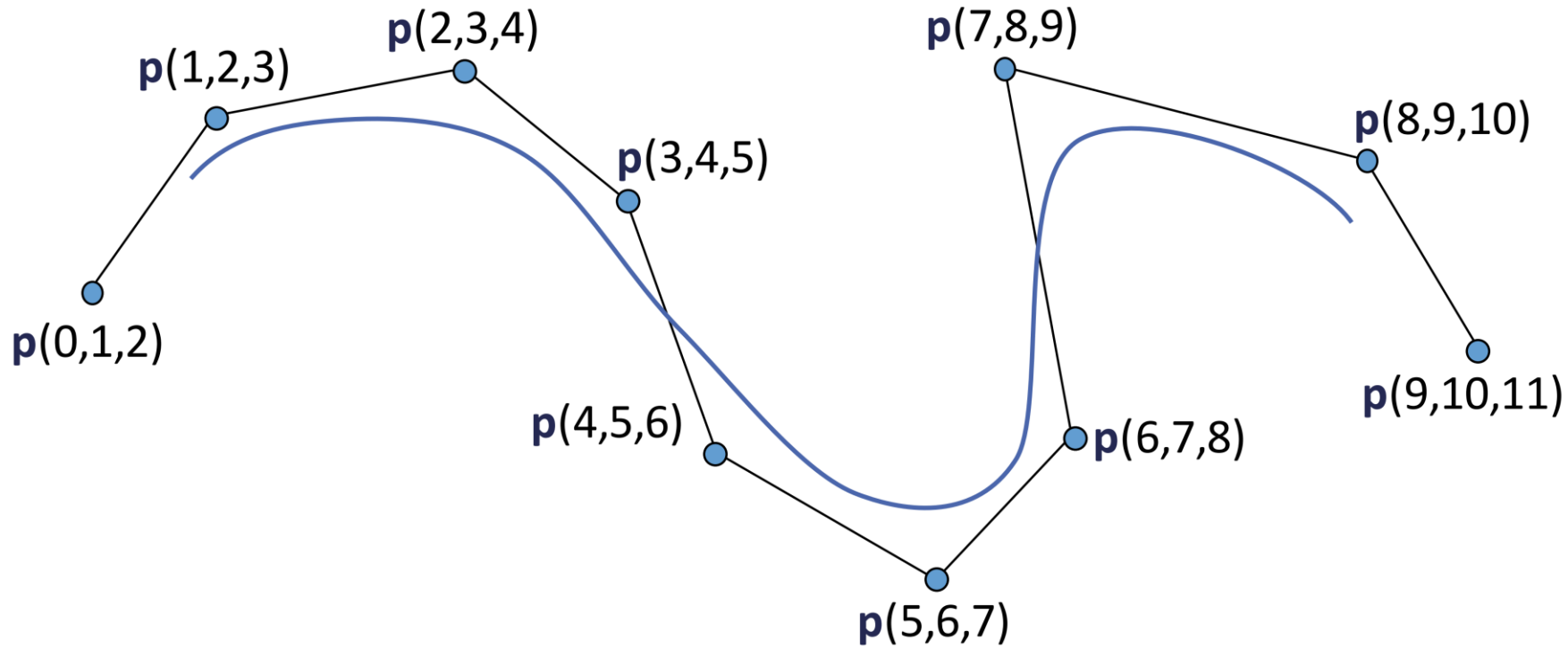
B-Splines in Polar form



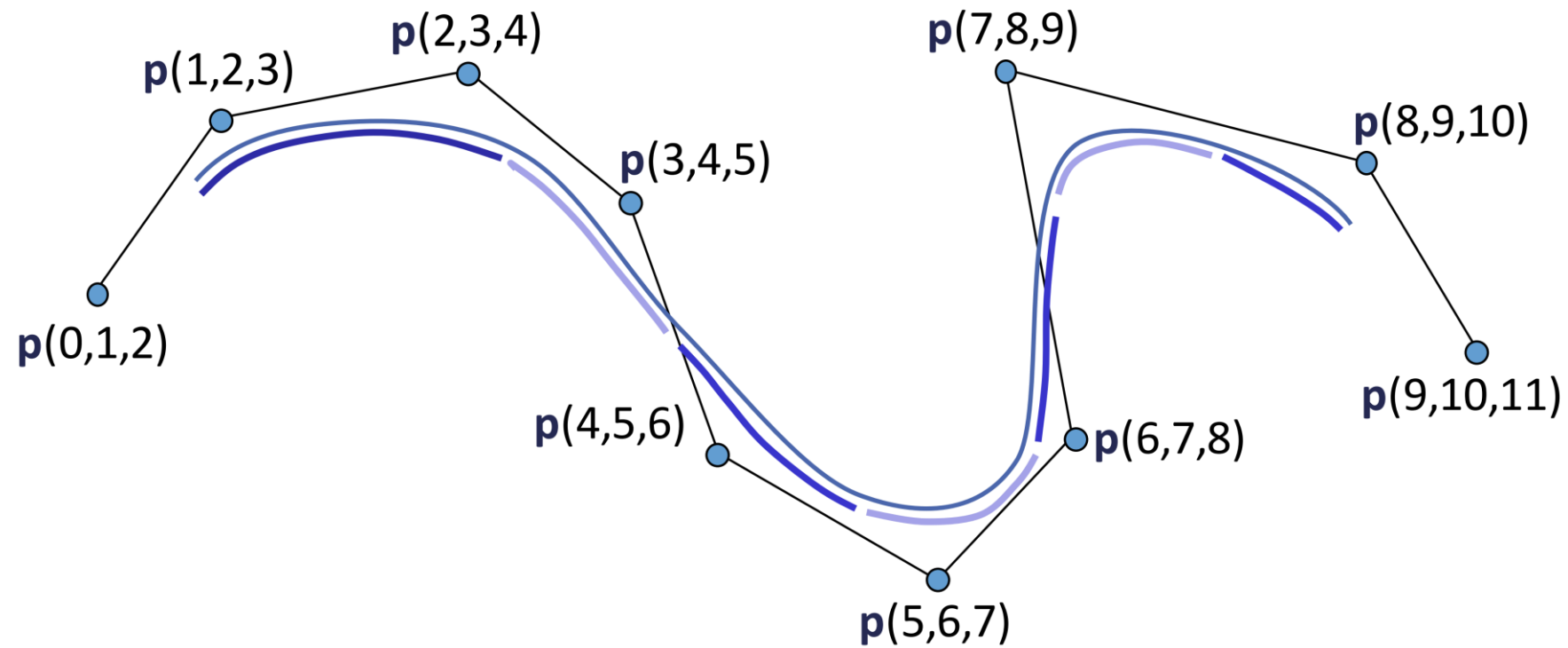
Example: General Case



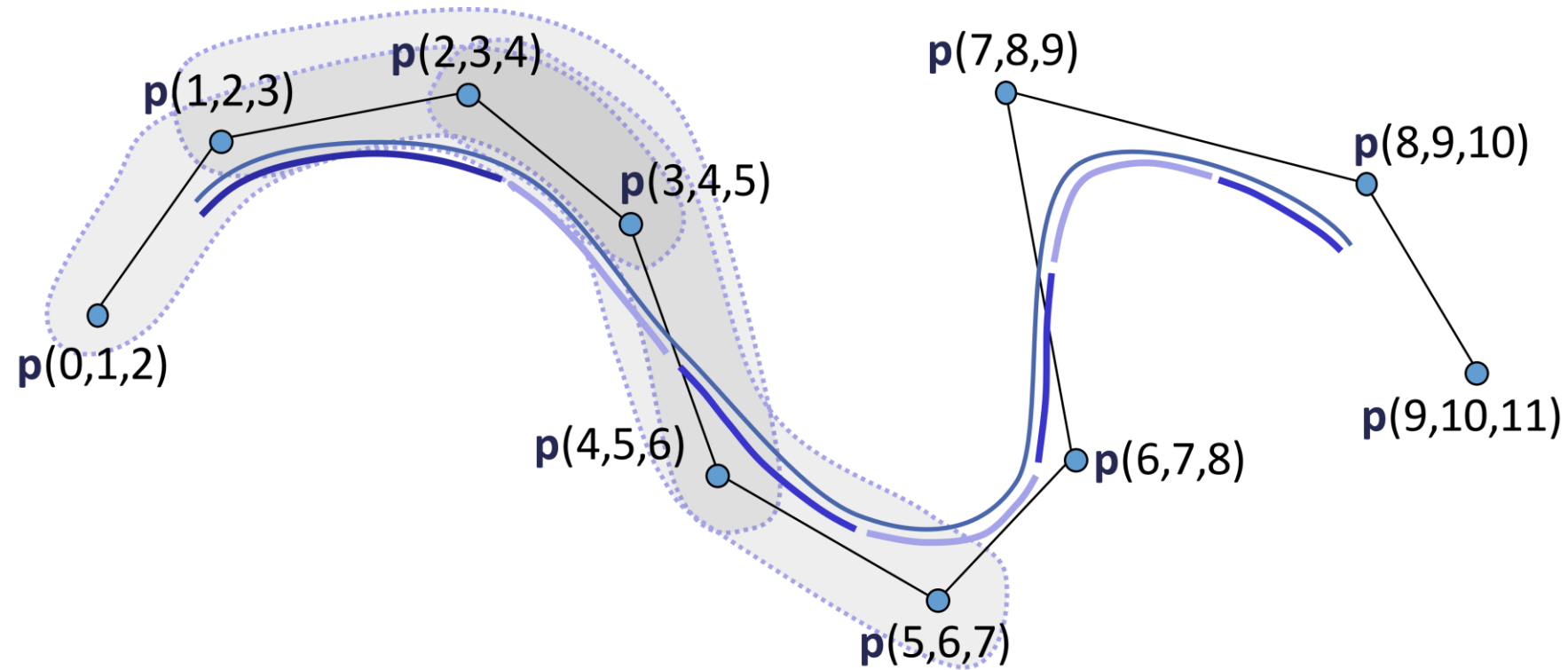
Example: General Case



Example: General Case



Example: General Case



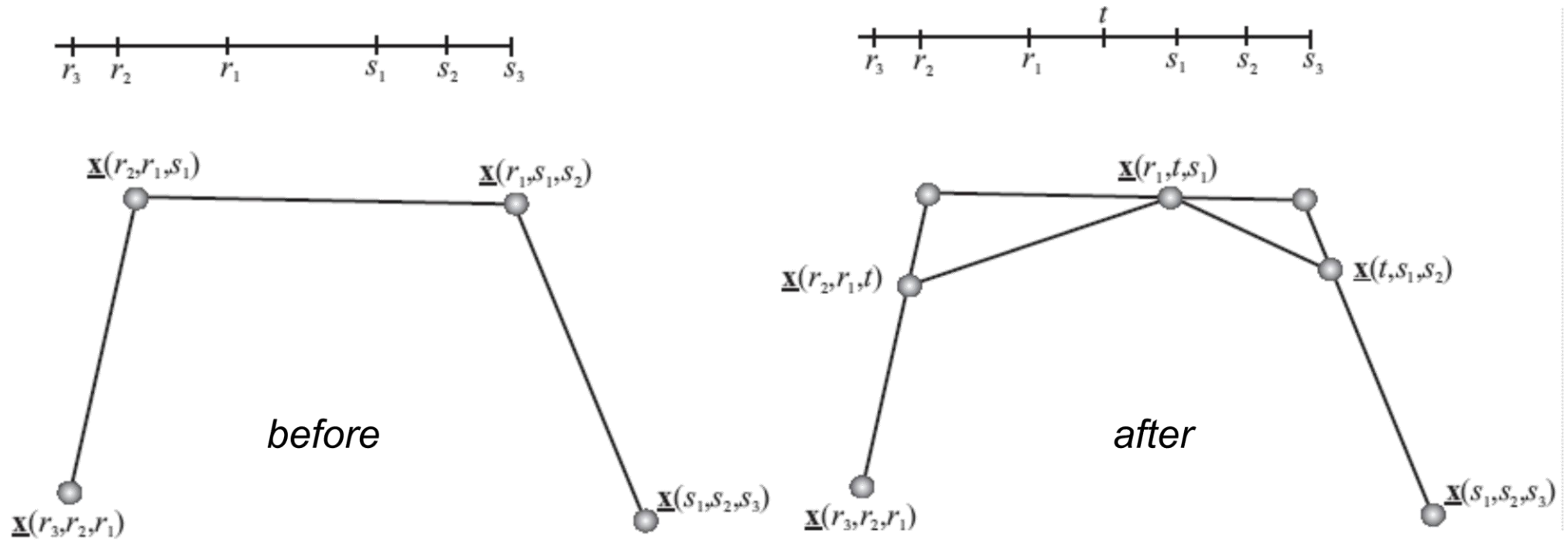
Knot Insertion

Knot insertion

- Increases the number of curve segments, but not the polynomial degree
- Insertion at t : First step of the de Boor algorithm!

Knot Insertion

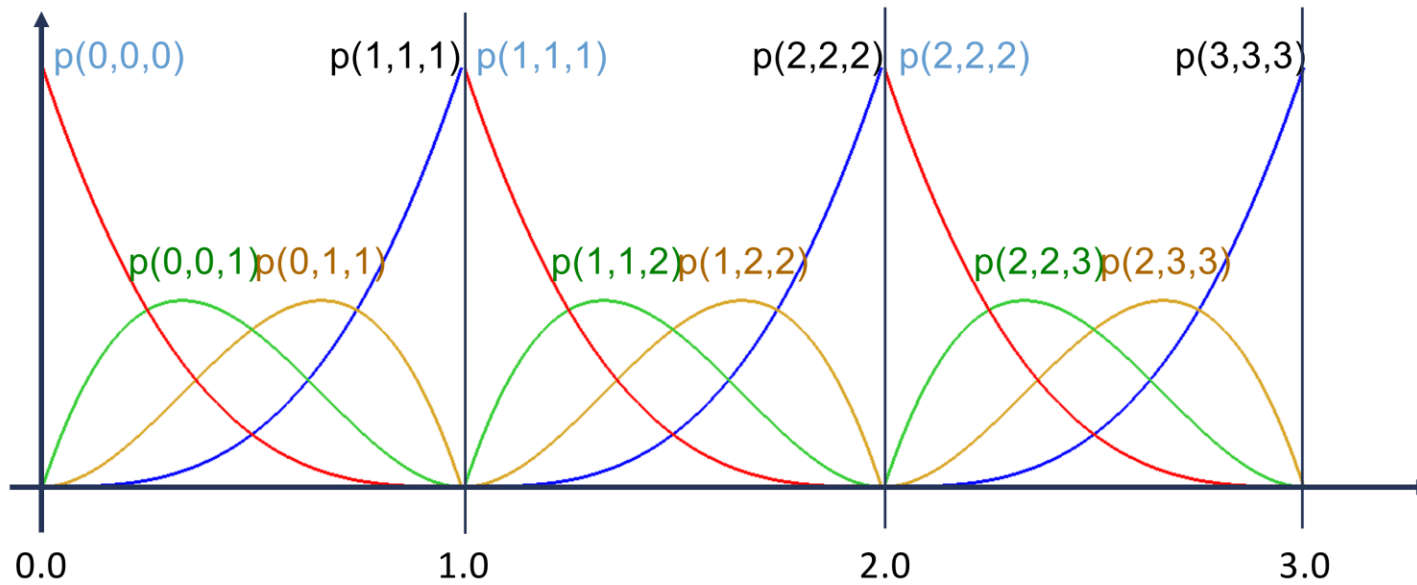
- Insertion of knots Example



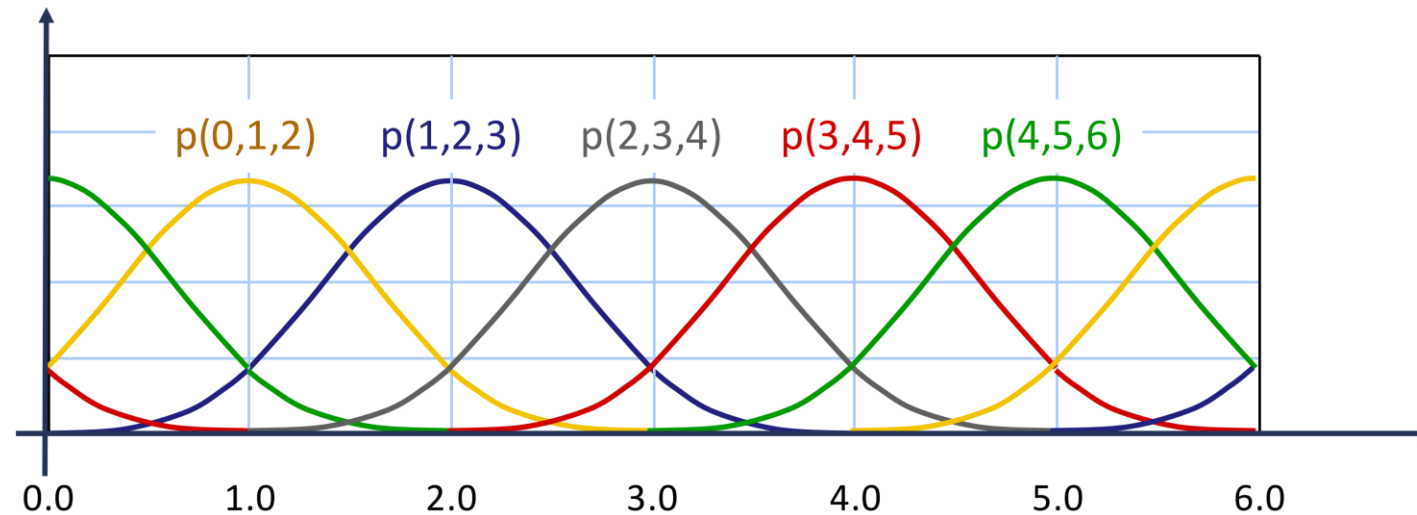
Polar Forms & Blossoms

Illustrations

Structure

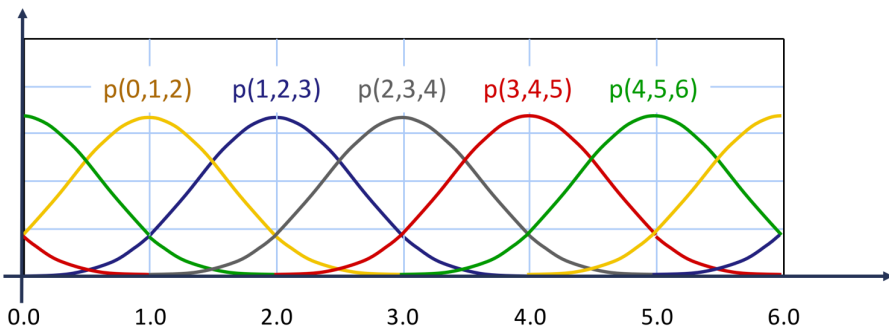
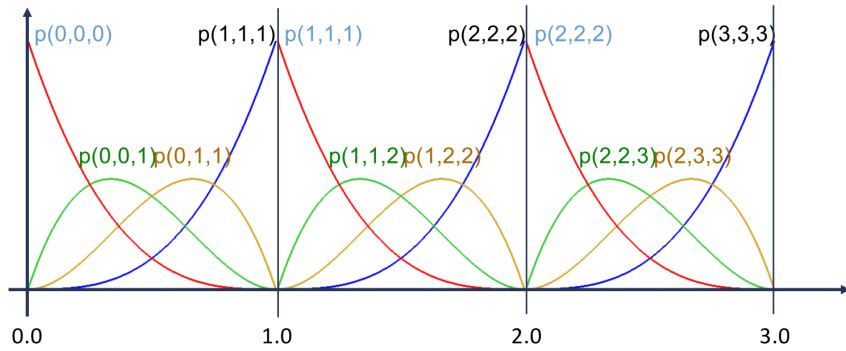


Bézier Spline

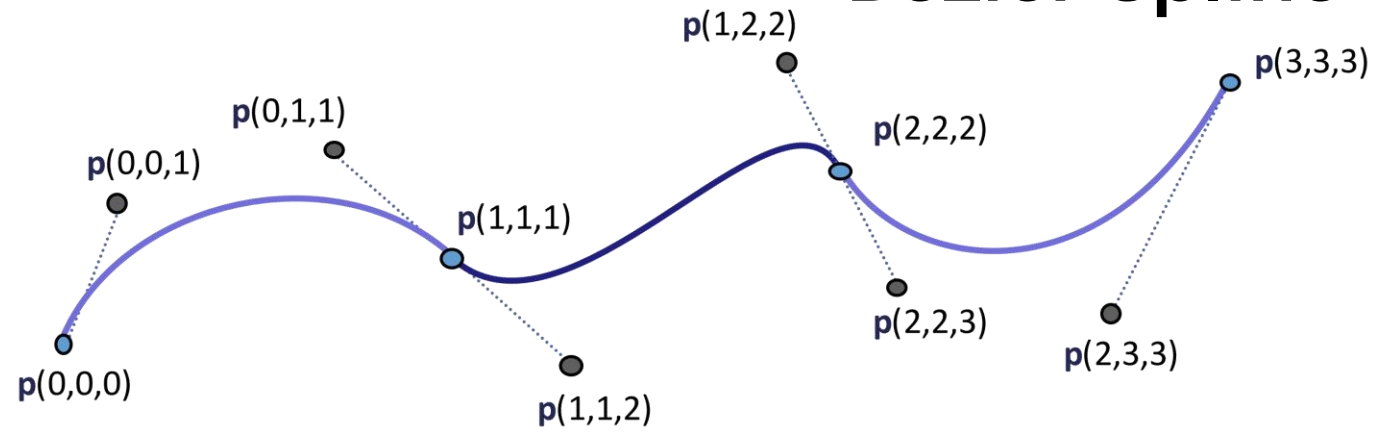


B-Spline

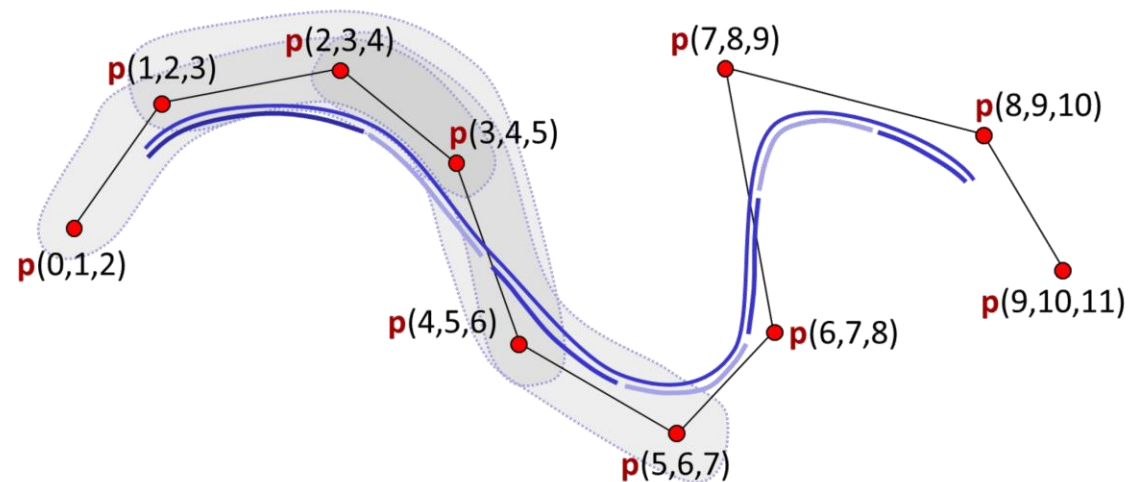
Structure



Bézier Spline



B-Spline



Computer Aided Geometric Design

Fall Semester 2024

Rational Spline Curves

Projective Geometry · Rational Bézier Curves · NURBS

陈仁杰

renjiec@ustc.edu.cn

<http://staff.ustc.edu.cn/~renjiec>

Some Projective Geometry

Projective Geometry

- **A very short overview of projective geometry**
 - The computer graphics perspective
 - Formal definition

Homogeneous Coordinates

- **Problem**

- Linear maps (matrix multiplication in \mathbb{R}^d) can represent ...
 - Rotations
 - Scaling
 - Shearing
 - Orthogonal projection
- ...but not:
 - Translations
 - Perspective projections
- This is a problem in computer graphics:
 - We would like to represent compound operations in a single closed representation

Translations

- **“Quick Hack” #1: Translations**

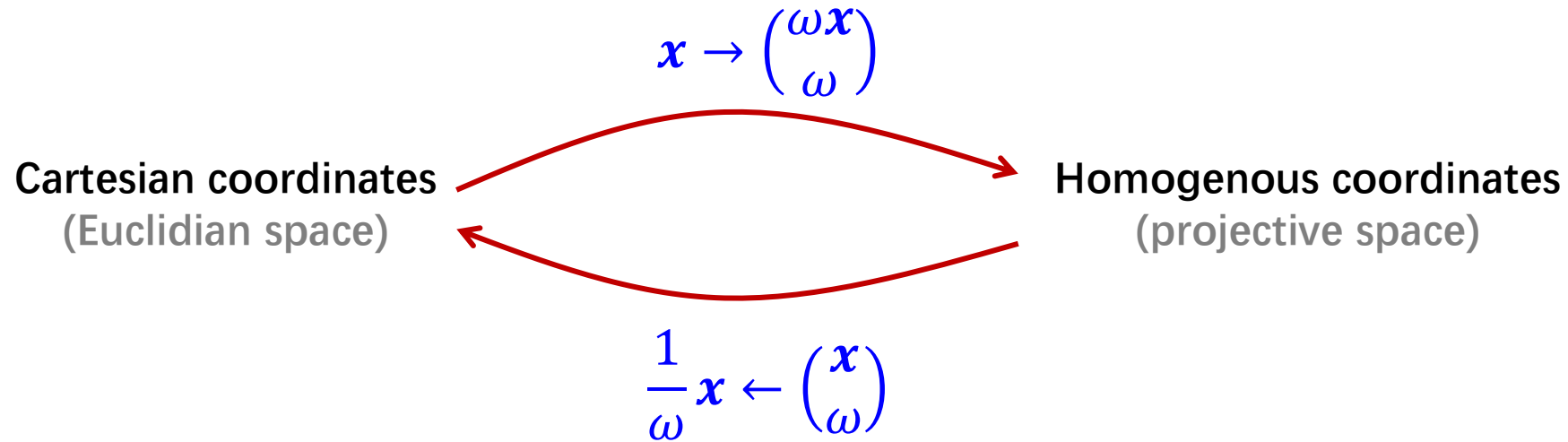
- Linear maps cannot represent translations:
 - Every linear map maps the zero vector to zero $M\mathbf{0} = \mathbf{0}$
 - Thus, non-trivial translations are non-linear
- Solution:
 - Add one dimension to each vector
 - Fill in a one
 - Now we can do translations by adding multiples of the one:

$$Mx = \begin{pmatrix} r_{11} & r_{21} & t_x \\ r_{12} & r_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{21} \\ r_{12} & r_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

Normalization

Problem: what if the last entry is not 1?

- It's not a bug, it's a feature...
- If the last component is not 1, divide everything by it before using the result



Notation

Notation:

- The extra component is called the *homogenous component* of the vector.
- It is usually denoted by ω :

■ 2D case:

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} \omega x \\ \omega y \\ \omega \end{pmatrix}$$

■ 3D case:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} \omega x \\ \omega y \\ \omega z \\ \omega \end{pmatrix}$$

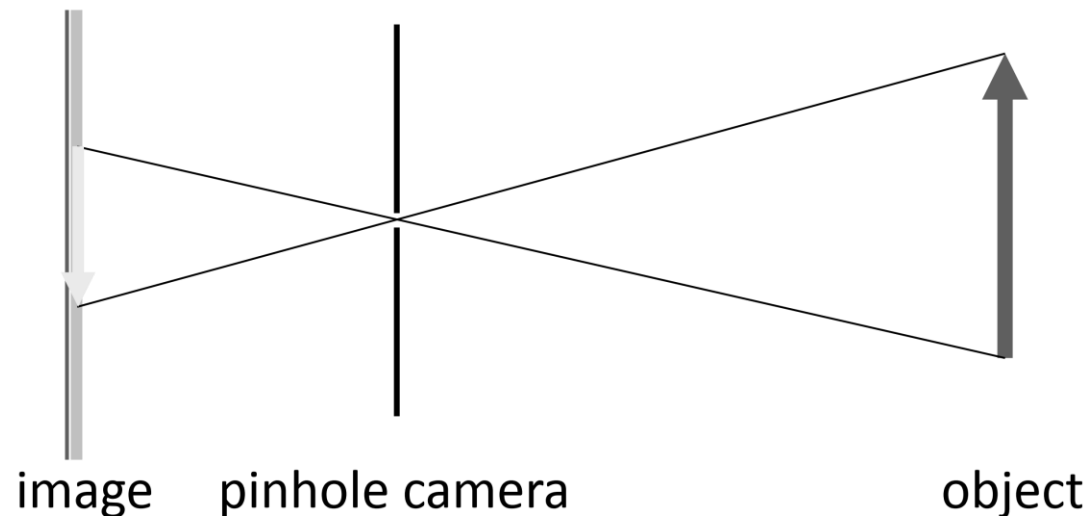
■ General case:

$$\mathbf{x} \rightarrow \begin{pmatrix} \omega \mathbf{x} \\ \omega \end{pmatrix}$$

Perspective Projections

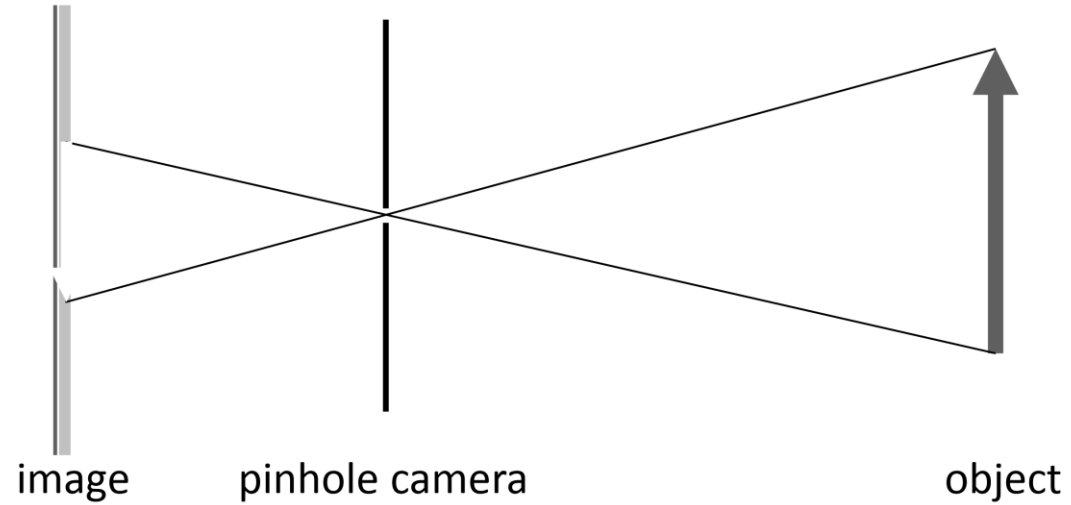
New Feature: Perspective projections

- Very useful for 3D computer graphics
- Perspective projection (central projection)
 - involves divisions
 - can be packed into homogeneous component

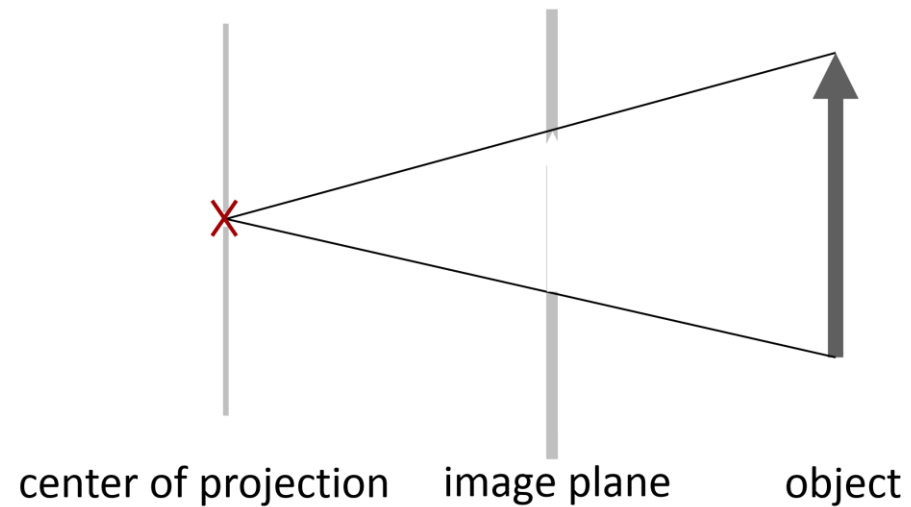


Perspective Projection

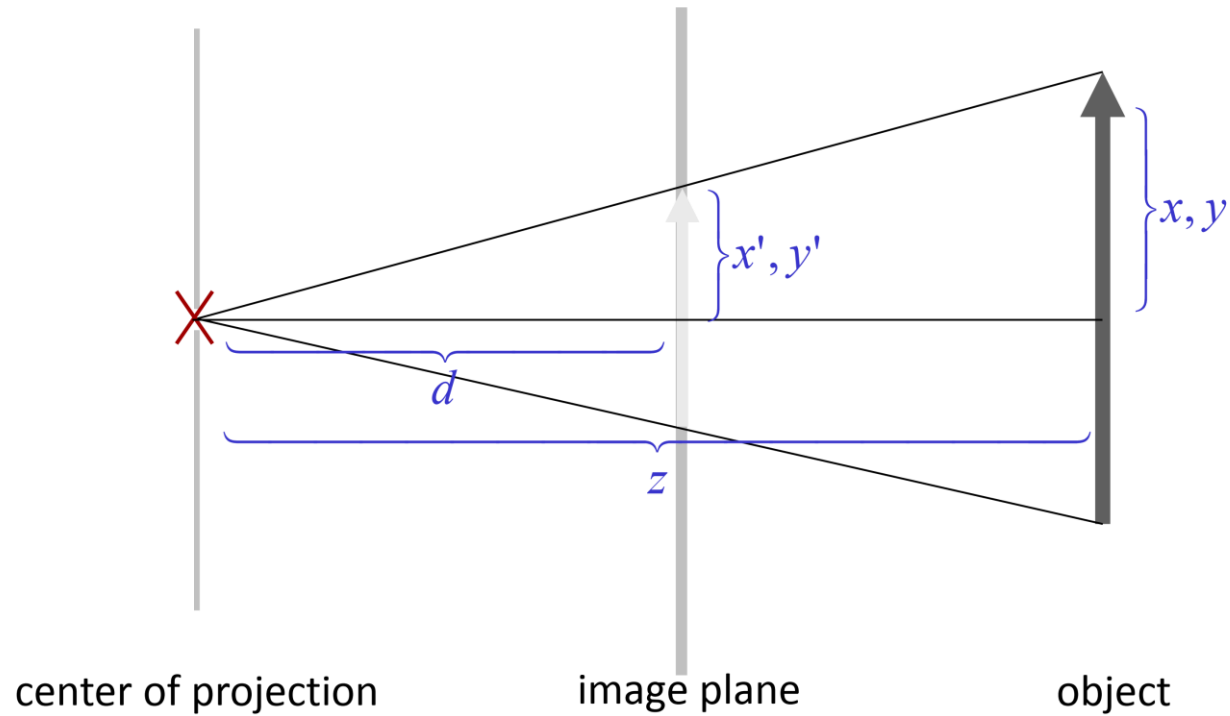
Physical camera:



Virtual camera:



Perspective Projection



Perspective projection: $x' = d \frac{x}{z}, y' = d \frac{y}{z}$

Homogenous Transformation

- **Projection as linear transformation in homogenous coordinates:**
 - Trick: Put the denominator into the ω component

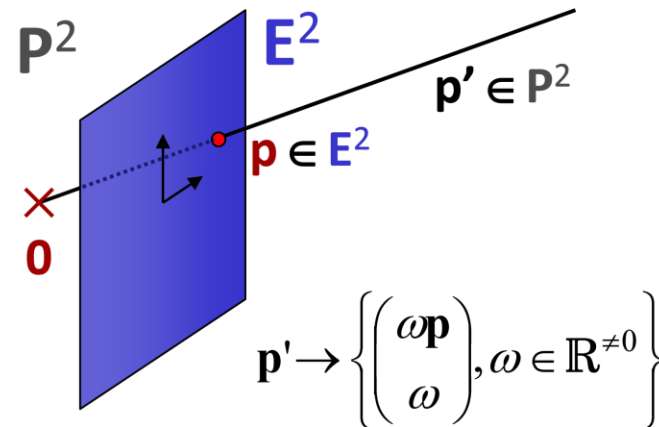
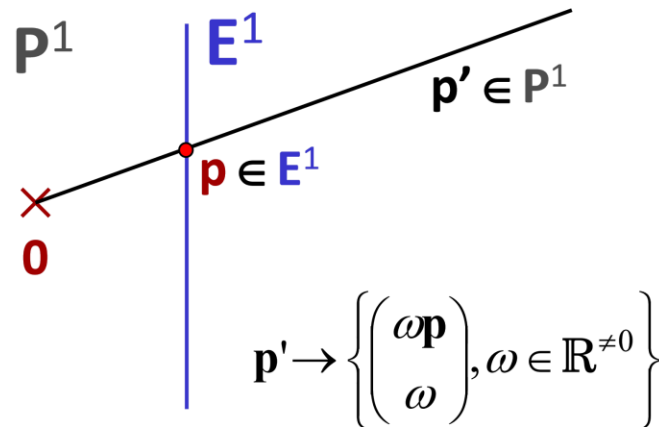
$$x' = d \frac{x}{z}, \quad y' = d \frac{y}{z}$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ \omega' \end{pmatrix} = \begin{pmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Formal Definition

Projective Space P^d

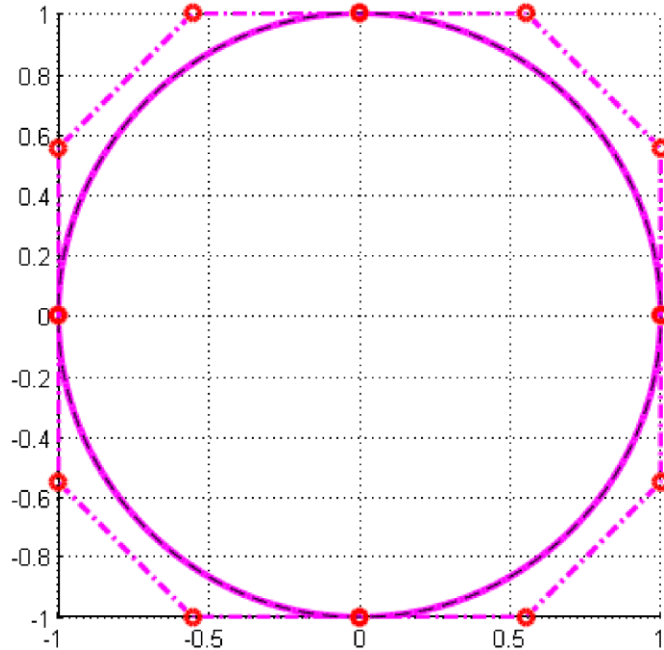
- Embed Euclidian space E^d
 - Into $d + 1$ dimensional Euclidian space at $\omega = 1$
 - Additional dimension usually named ω
- Identify all points on lines through the origin
 - *Representing* the same Euclidian point



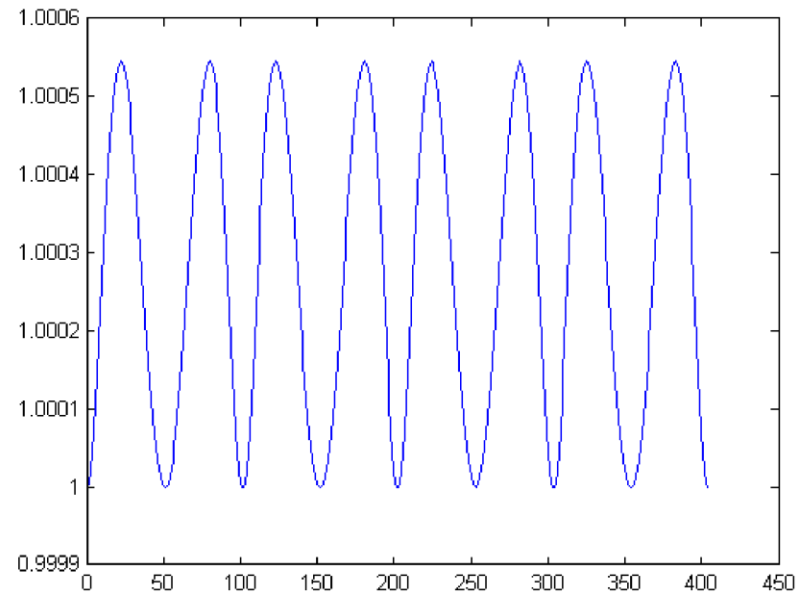
Question

Can we represent a circle arc using a Bézier curve?

Approximation of Circle using Cubic Bézier



Evaluation of $(x^2 + y^2)$ for points on the Bézier curve



Rational Curves

- **Motivation**

- Bézier and B-spline curves **cannot** represent conic sections (circles, hyperbolas, etc.)
- But we require those for some tasks

- **Goal**

- Uniform and easily manageable description of polynomial curves and conic sections

- **Idea**

- Control points are equipped with weights...but not any weights!



Planetarium of the St. Louis Science Center



Tycho Brahe Planetarium, Copenhagen

Quadrics and Conics

Modeling Wish List

We want to model:

- Circles (surfaces: Spheres)
- Ellipses (surfaces: Ellipsoids)
- And segments of those
- Surfaces: Objects with circular cross section
 - Cylinders
 - Cones
 - Surfaces of revolution (lathing)

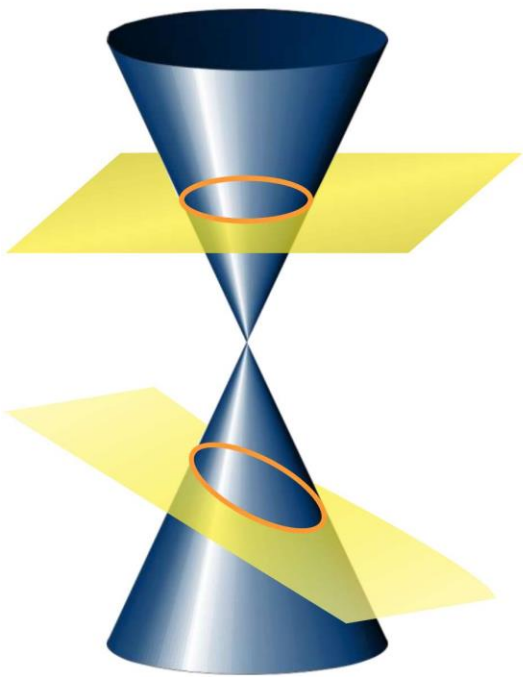
These objects cannot be represented exactly by piecewise polynomials (they are only approximated)

Conical Sections

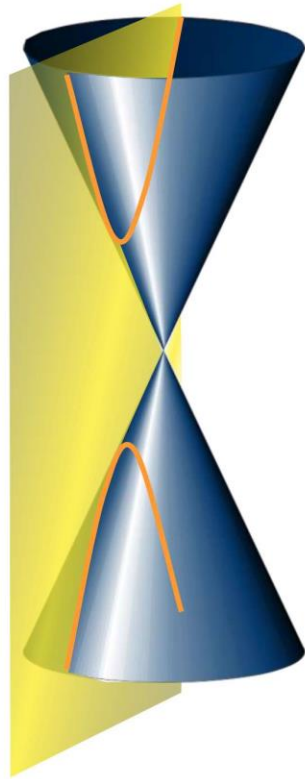
Classic description of such objects:

- Conical sections (conics)
- Intersections of a cone and a plane
- Resulting Objects:
 - Circles
 - Ellipses
 - Hyperbolas
 - Parabolas
 - Points
 - Lines

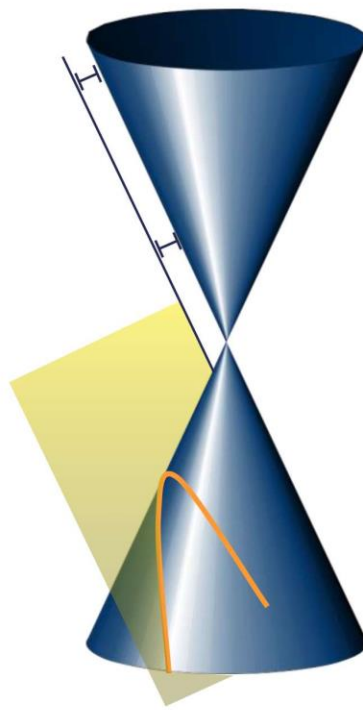
Conic Sections



**Circle,
Ellipse**



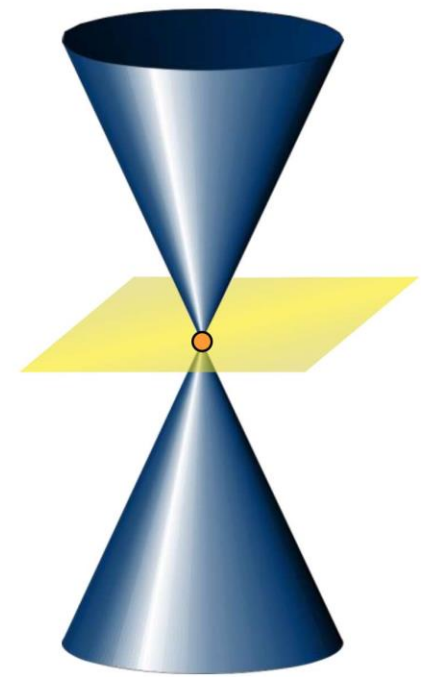
Hyperbola



Parabola



Line
(degenerate case)



Point
(degenerate case)

Implicit Form

Implicit quadrics:

- Conic sections can be expressed as zero set of a quadratic function:

$$ax^2 + bxy + cy^2 + dx + ey + f = 0$$
$$\Leftrightarrow \mathbf{x}^T \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix} \mathbf{x} + [d \quad e]\mathbf{x} + f = 0$$

- Easy to see why:

Implicit eq. for a cone: $Ax^2 + by^2 = z^2$

Explicit eq. for a plane: $z = Dx + Ey + F$

Conical Section: $Ax^2 + By^2 = (Dx + Ey + F)^2$

Quadrics & Conics

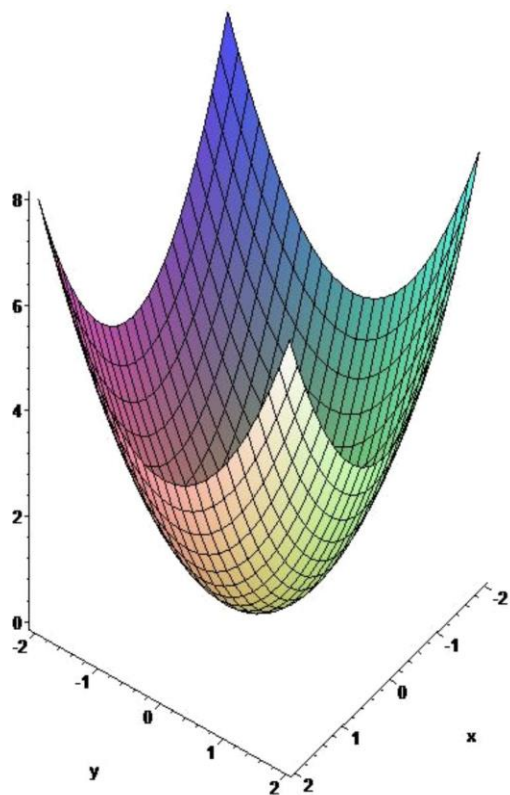
Quadrics:

- Zero sets of quadratic functions (any dimension) are called *quadrics*:

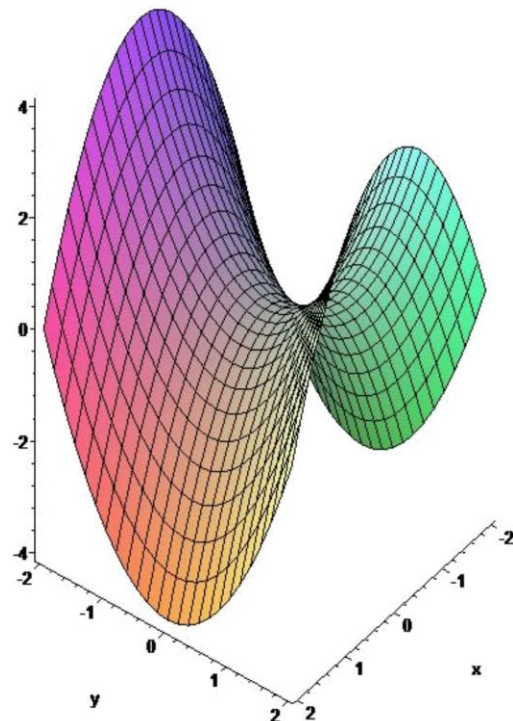
$$\{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{x}^T \mathbf{M} \mathbf{x} + \mathbf{b}^T \mathbf{x} + c = 0\}$$

- *Conics* are the special case for $d = 2$

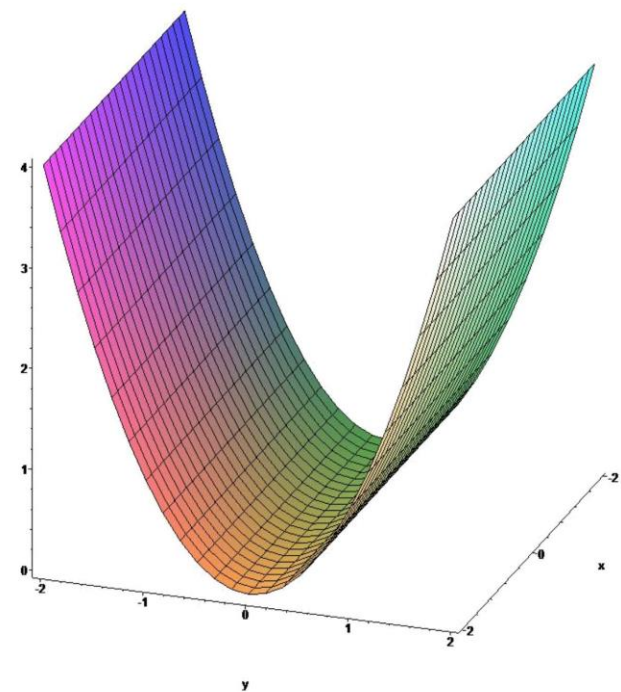
Shapes of Quadratic Polynomials



$$\lambda_1 = 1, \lambda_2 = 1$$



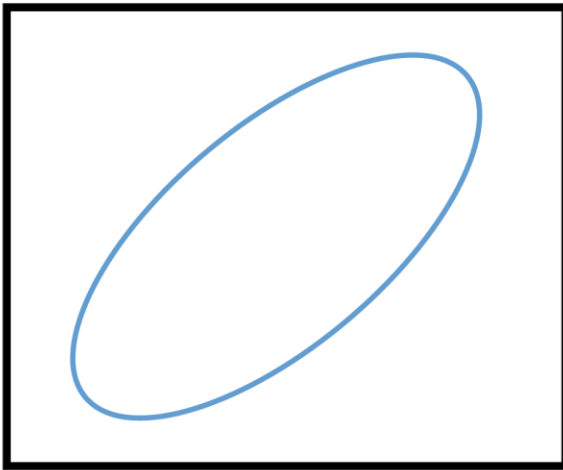
$$\lambda_1 = 1, \lambda_2 = -1$$



$$\lambda_1 = 1, \lambda_2 = 0$$

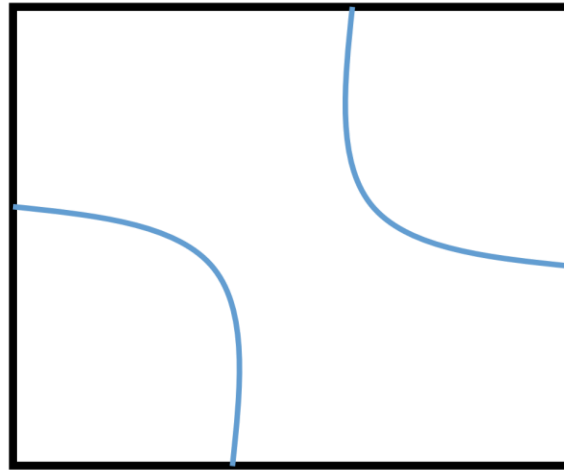
The Iso-Lines: Quadrics

Elliptic



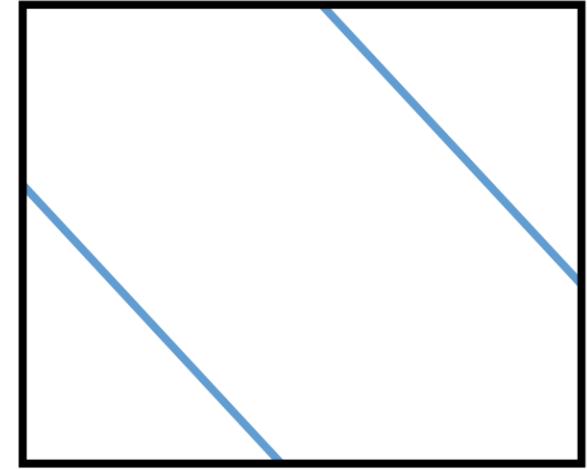
$$\lambda_1 > 0, \lambda_2 > 0$$

hyperbolic



$$\lambda_1 < 0, \lambda_2 > 0$$

degenerate case



$$\lambda_1 = 0, \lambda_2 > 0$$

Characterization

Determining the type of Conic from the implicit form:

- Implicit function: quadratic polynomial

$$ax^2 + bxy + cy^2 + dx + ey + f = 0$$

$$\Leftrightarrow \mathbf{x}^T \underbrace{\begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix}}_M \mathbf{x} + [d \quad e]\mathbf{x} + f = 0$$

- Eigenvalues of M

$$\lambda_{1,2} = \frac{a+c}{2} \pm \frac{1}{2} \sqrt{(a-c)^2 + b^2}$$

Cases

We obtain the following cases:

- Ellipse: $b^2 < 4ac$
 - Circle: $b = 0, a = c$
 - Otherwise: general ellipse
- Parabola: $b^2 = 4ac$ (border case)
- Hyperbola: $b^2 > 4ac$

Implicit function:
 $ax^2 + bxy + cy^2 + dx + ey + f = 0$

Cases

Implicit function:

$$ax^2 + bxy + cy^2 + dx + ey + f = 0$$

Explanation:

$$\begin{aligned} b^2 = 4ac \Rightarrow \lambda_{1,2} &= \frac{a+c}{2} \pm \frac{1}{2} \sqrt{(a-c)^2 + 4ac} \\ &= \frac{a+c}{2} \pm \frac{1}{2} \sqrt{a^2 - 2ac + c^2 + 4ac} \\ &= \frac{a+c}{2} \pm \frac{1}{2} \sqrt{a^2 + 2ac + c^2} \\ &= \frac{a+c}{2} \pm \frac{1}{2} \sqrt{(a+c)^2} \\ &= \frac{a+c}{2} \pm \frac{a+c}{2} \\ &= \{0, a+c\} \end{aligned}$$

Polynomial Curves & Conics

We want to represent conics with parametric curves:

- How can we represent (pieces) of conics as parametric curves?
- How can we generalize our framework of piecewise polynomial curves to include conical sections?

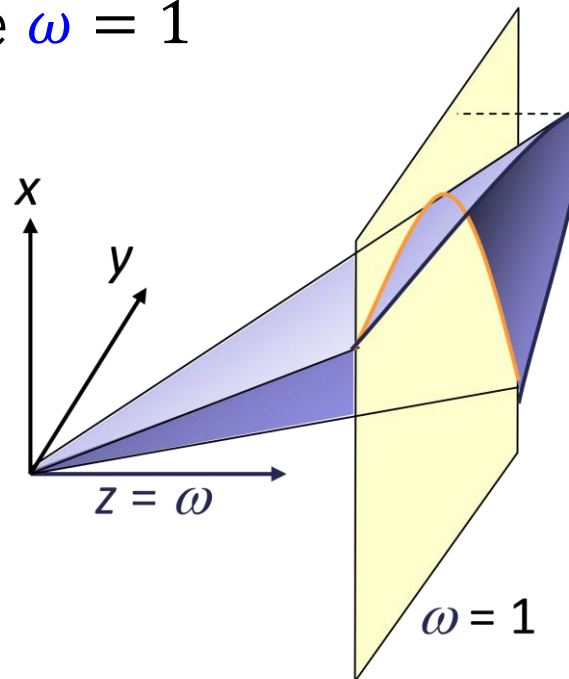
Projections of Parabolas:

- We will look at a certain class of parametric functions – projections of parabolas
- This class turns out to be general enough
- And can be expressed easily with the tools we know.

Projections of Parabolas

Definition: Projection of a Parabola

- We start with a quadratic space curve
- Interpret the z -coordinate as homogenous component ω
- Project the curve on the plane $\omega = 1$



Projected Parabola

Formal Definition:

- Quadratic polynomial curve in three space
- Project by dividing by the third coordinates

$$\mathbf{f}^{(hom)}(t) = \mathbf{p}_0 + t\mathbf{p}_1 + t^2\mathbf{p}_2 = \begin{pmatrix} \mathbf{p}_0 \cdot x \\ \mathbf{p}_0 \cdot y \\ \mathbf{p}_0 \cdot \omega \end{pmatrix} + t \begin{pmatrix} \mathbf{p}_1 \cdot x \\ \mathbf{p}_1 \cdot y \\ \mathbf{p}_1 \cdot \omega \end{pmatrix} + t^2 \begin{pmatrix} \mathbf{p}_2 \cdot x \\ \mathbf{p}_2 \cdot y \\ \mathbf{p}_2 \cdot \omega \end{pmatrix}$$

$$\mathbf{f}^{(eucl)}(t) = \frac{\begin{pmatrix} \mathbf{p}_0 \cdot x \\ \mathbf{p}_0 \cdot y \end{pmatrix} + t \begin{pmatrix} \mathbf{p}_1 \cdot x \\ \mathbf{p}_1 \cdot y \end{pmatrix} + t^2 \begin{pmatrix} \mathbf{p}_2 \cdot x \\ \mathbf{p}_2 \cdot y \end{pmatrix}}{\mathbf{p}_0 \cdot \omega + t\mathbf{p}_1 \cdot \omega + t^2\mathbf{p}_2 \cdot \omega}$$

Parameterizing Conics

Conics can be parameterized using projected parabolas:

- We show that we can represent (piecewise):
 - Points and lines (obvious ✓)
 - A unit parabola
 - A unit circle
 - A unit hyperbola
- General cases (ellipses etc.) can be obtained by affine mappings of the control points (which leads to affine maps of the curve)

Parameterizing Parabolas

Parabolas as rational parametric curves:

$$f^{(eucl)}(t) = \frac{\begin{pmatrix} 0 \\ 0 \end{pmatrix} + t \begin{pmatrix} 1 \\ 0 \end{pmatrix} + t^2 \begin{pmatrix} 0 \\ 1 \end{pmatrix}}{1 + 0t + 0t^2} \quad \begin{pmatrix} x(t) = t \\ y(t) = t^2 \end{pmatrix} \checkmark \text{ (obvious)}$$

Circle

Let's try to find a rational parameterization of a (**piece of a**) unit circle:

$$f^{(eucl)}(\varphi) = \begin{pmatrix} \cos \varphi \\ \sin \varphi \end{pmatrix}$$

Circle

Let's try to find a rational parameterization of a (**piece of a**) unit circle:

$$f^{(eucl)}(\varphi) = \begin{pmatrix} \cos \varphi \\ \sin \varphi \end{pmatrix}$$

$$\cos \varphi = \frac{1 - \tan^2 \frac{\varphi}{2}}{1 + \tan^2 \frac{\varphi}{2}}, \quad \sin \varphi = \frac{2 \tan \frac{\varphi}{2}}{1 + \tan^2 \frac{\varphi}{2}} \text{ (tangent half-angle formula)}$$

$$t := \tan \frac{\varphi}{2} \Rightarrow f^{(eucl)}(\varphi) = \begin{pmatrix} \frac{1-t^2}{1+t^2} \\ \frac{2t}{1+t^2} \end{pmatrix}$$

Circle

Let's try to find a rational parameterization of a (**piece of a**) unit circle:

$$\mathbf{f}^{(eucl)}(\varphi) = \begin{pmatrix} \cos \varphi \\ \sin \varphi \end{pmatrix} = \begin{pmatrix} \frac{1-t^2}{1+t^2} \\ \frac{2t}{1+t^2} \end{pmatrix} \text{ with } t := \tan \frac{\varphi}{2}$$

$$\Rightarrow \mathbf{f}^{(hom)}(t) = \begin{pmatrix} 1 - t^2 \\ 2t \\ 1 + t^2 \end{pmatrix}$$

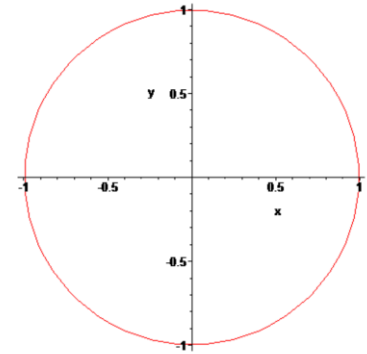
parameterization for $\varphi \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$

\Rightarrow we need at least three segments to parametrize a full circle

Hyperbolas

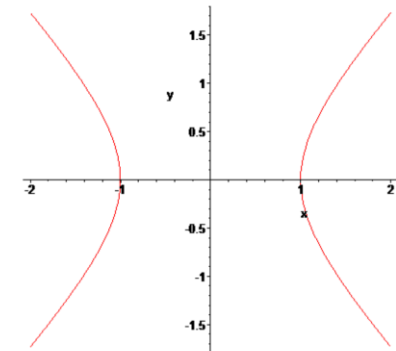
Unit Circle: $x^2 + y^2 = 1$

$$\Rightarrow x(t) = \frac{1 - t^2}{1 + t^2}, \quad y(t) = \frac{2t}{1 + t^2} \quad (t \in \mathbb{R})$$



Unit Hyperbola: $x^2 - y^2 = 1$

$$\Rightarrow x(t) = \frac{1 + t^2}{1 - t^2}, \quad y(t) = \frac{2t}{1 - t^2}, \quad (t \in [0, 1])$$



Rational Bézier Curves

Rational Bézier Curves

Rational Bézier curves in \mathbb{R}^n of degree d :

- Form a Bézier curve of degree d in $n + 1$ dimensional space
- Interpret last coordinates as homogenous component
- Euclidean coordinates are obtained by projection

$$\mathbf{f}^{(hom)}(t) = \sum_{i=0}^n B_i^{(d)}(t) \mathbf{p}_i, \quad \mathbf{p}_i \in \mathbb{R}^{n+1}$$

$$\mathbf{f}^{(eucl)}(t) = \frac{\sum_{i=0}^n B_i^{(d)}(t) \begin{pmatrix} p_i^{(1)} \\ \vdots \\ p_i^{(n)} \end{pmatrix}}{\sum_{i=0}^n B_i^{(d)}(t) p_i^{(n+1)}}$$

More Convenient Notation

The curve can be written in “weighted points” form:

$$\mathbf{f}^{(eucl)}(t) = \frac{\sum_{i=0}^n B_i^{(d)}(t) \omega_i \begin{pmatrix} p_i^{(1)} \\ \vdots \\ p_i^{(n)} \end{pmatrix}}{\sum_{i=0}^n B_i^{(d)}(t) \omega_i}$$

Interpretation:

- Points are weighted by weights ω_i
- Normalized by interpolated weights in the denominator
- Large weights \rightarrow more influence of that point

Properties

What about affine invariance, convex hull prop.?

$$\mathbf{f}^{(eucl)}(t) = \sum_{i=0}^n \mathbf{p}_i \frac{B_i^{(d)}(t)\omega_i}{\sum_{j=0}^n B_j^{(d)}(t)\omega_j} = \sum_{i=0}^n q_i(t)\mathbf{p}_i \quad \text{with } \sum_{i=0}^n q_i(t) = 1$$

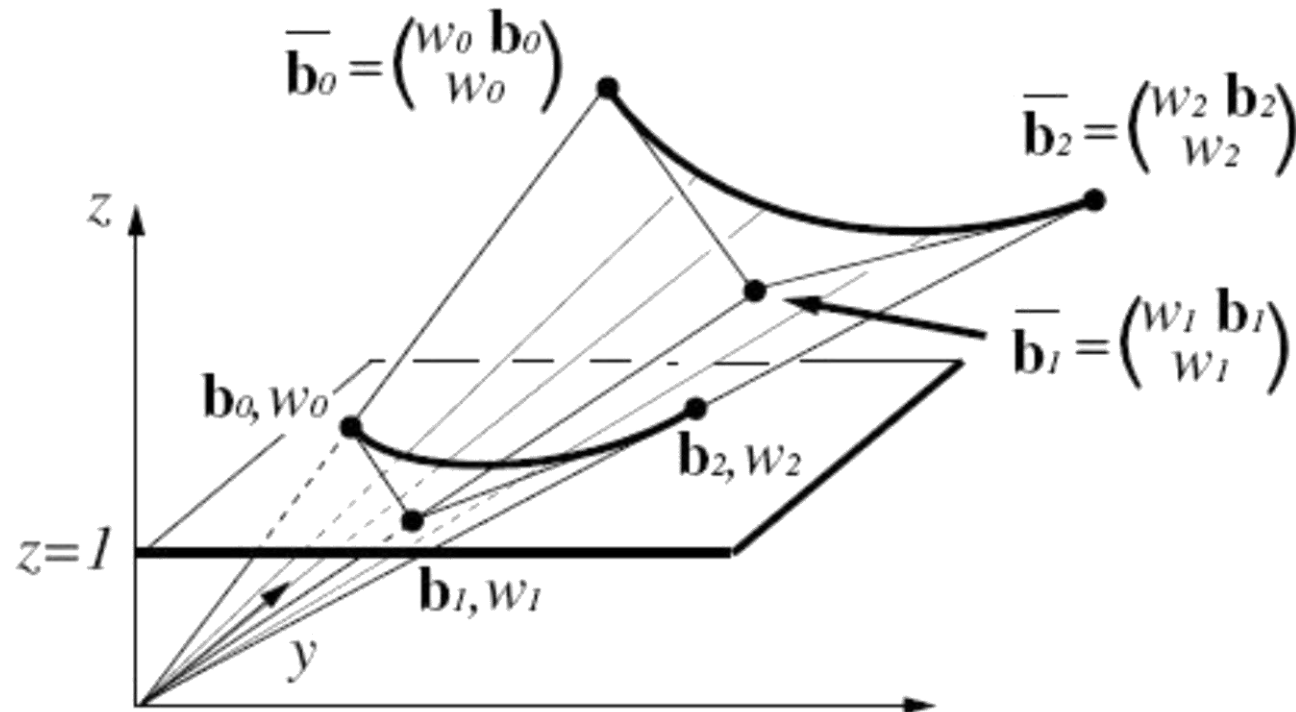
Consequences:

- Affine invariance still holds
- For strictly positive weights:
 - Convex hull property still holds
 - This is not a big restriction (potential singularities otherwise)
- Projective invariance (projective maps, hom. coord's)

Rational Bézier Curves

Geometric interpretation of rational Bézier curves:

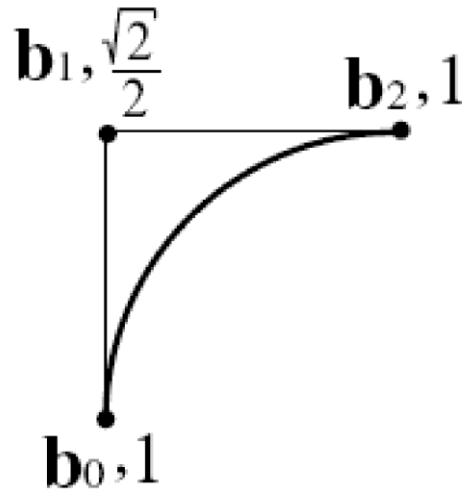
- Rational Bézier curves are obtained by central projection of “normal” Bézier curves



Rational Bézier Curves

Examples:

- $\omega_i = 1$ ($i = 0, \dots, n$): “normal” Bézier curves
- Generally:
 - Each conic section can be described as rational Bézier curve of degree two
 - Each rational Bézier curve of degree two is a conic section
- Example: Circular arc



Rational de Casteljau Algorithm

Evaluation with de Casteljau Algorithm

- Three variants:
 - Compute in $n + 1$ dimensional space, then project
 - Compute numerator and denominator separately, then divide
 - Divide in each intermediate step (“rational de Casteljau”)
- **Non-rational** de Casteljau algorithm:

$$\mathbf{b}_i^{(r)}(t) = (1 - t)\mathbf{b}_i^{(r-1)}(t) + t\mathbf{b}_{i+1}^{(r-1)}(t)$$

- **Rational** de Casteljau algorithm

$$\mathbf{b}_i^{(r)}(t) = (1 - t) \frac{\omega_i^{(r-1)}(t)}{\omega_i^{(r)}(t)} \mathbf{b}_i^{(r-1)}(t) + t \frac{\omega_{i+1}^{(r-1)}(t)}{\omega_i^{(r)}(t)} \mathbf{b}_{i+1}^{(r-1)}(t)$$

$$\text{with } \omega_i^{(r)}(t) = (1 - t)\omega_i^{(r-1)}(t) + t\omega_{i+1}^{(r-1)}(t)$$

Rational de Casteljau Algorithm

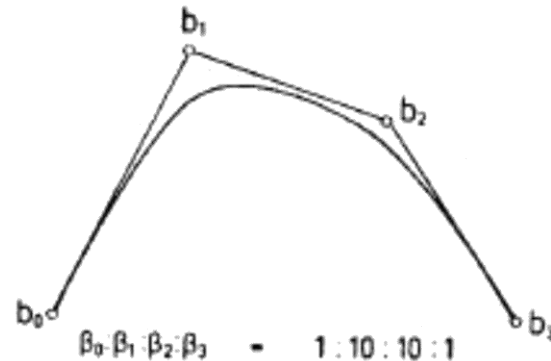
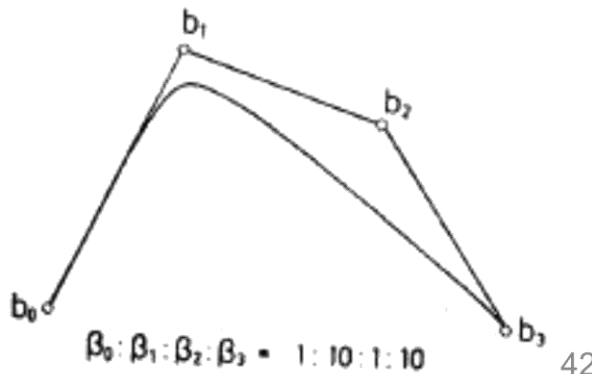
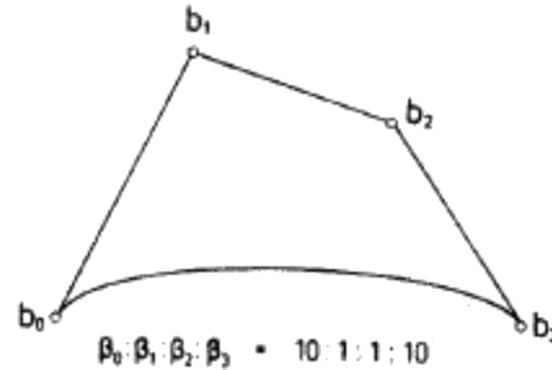
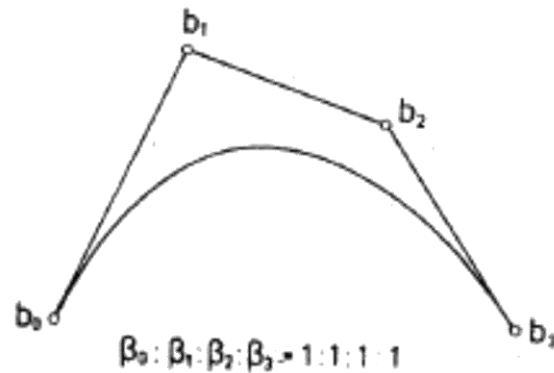
Advantages:

- More intuitive (repeated weighted linear interpolation of points and weights)
- Numerically more stable (only convex combinations for the standard case of positive weights, $t \in [0,1]$)

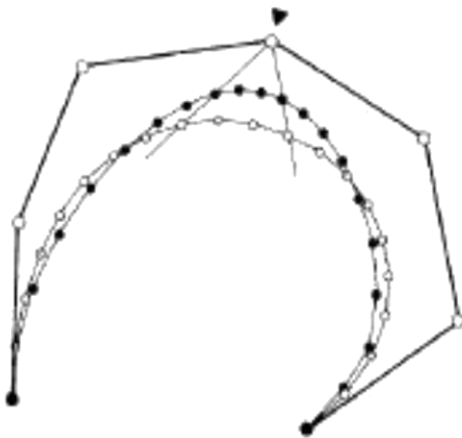
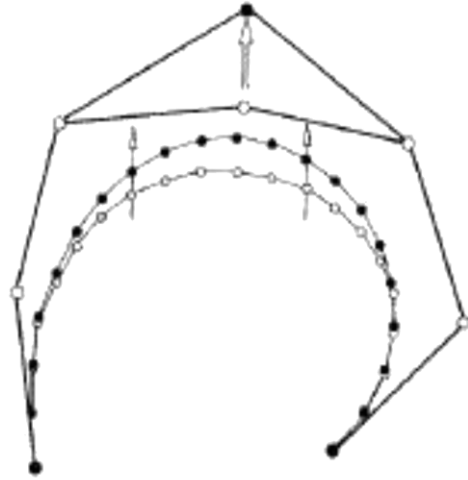
Influence of the Weights

Influence of the weights on the curve shape:

- Increasing ω_i moves the curve towards the Bézier point b_i
- Examples:



Influence of the Weights



Moving a control point

Not the same!

Increasing the weight of a control point

Quadratic Bézier Curves

- Quadratic curves:
 - Necessary and sufficient to represent conics
 - Therefore, we will examine them closer ...
- Quadratic rational Bézier curve:

$$\mathbf{f}^{(eucl)}(t) = \frac{B_0^{(2)}(t)\omega_0\mathbf{p}_0 + B_1^{(2)}(t)\omega_1\mathbf{p}_1 + B_2^{(2)}(t)\omega_2\mathbf{p}_2}{B_0^{(2)}(t)\omega_0 + B_1^{(2)}(t)\omega_1 + B_2^{(2)}(t)\omega_2}, \quad \mathbf{p}_i \in \mathbb{R}^n, \omega_i \in \mathbb{R}$$

Standard Form (or Normal Form)

How many degrees of freedom are in the weights?

- Quadratic rational Bézier curve:

$$f^{(eucl)}(t) = \frac{B_0^{(2)}(t)\omega_0\mathbf{p}_0 + B_1^{(2)}(t)\omega_1\mathbf{p}_1 + B_2^{(2)}(t)\omega_2\mathbf{p}_2}{B_0^{(2)}(t)\omega_0 + B_1^{(2)}(t)\omega_1 + B_2^{(2)}(t)\omega_2}$$

If one of the weights is $\neq 0$ (which must be the case), we can divide numerator and denominator by this weight and thus remove one degree of freedom. *No impact on the curve.*

If we are only interested in the *shape of the curve*, we can remove one more degree of freedom by a *reparameterization* ... *No impact on shape of the curve*

Standard Form

How many degrees of freedom are in the weights?

- Concerning the shape of the curve, the parameterization does not matter
- We have

$$f^{(eucl)}(t) = \frac{(1-t)^2\omega_0\mathbf{p}_0 + 2t(1-t)\omega_1\mathbf{p}_1 + t^2\omega_2\mathbf{p}_2}{(1-t)^2\omega_0 + 2t(1-t)\omega_1 + t^2\omega_2}$$

- We set: (with α to be determined later)

$$t \leftarrow \frac{\tilde{t}}{\alpha(1-\tilde{t})+\tilde{t}}, \quad \text{i.e., } (1-t) \leftarrow \frac{\alpha(1-\tilde{t})}{\alpha(1-\tilde{t})+\tilde{t}}$$

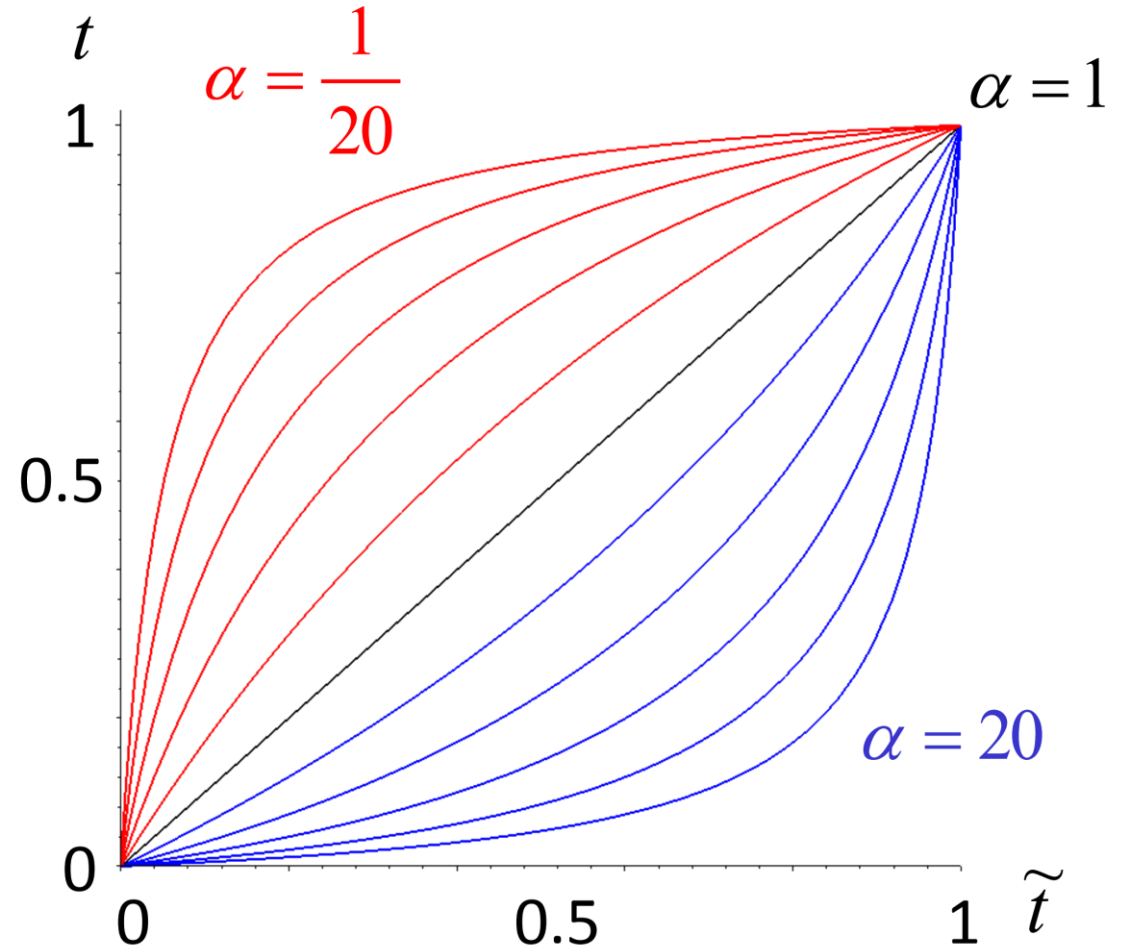
Remark: Why this reparameterization?

Reparameterization:

$$t \leftarrow \frac{\tilde{t}}{\alpha(1 - \tilde{t}) + \tilde{t}}$$

Properties:

- $0 \rightarrow 0, 1 \rightarrow 1$,
monotonic in between
- Shape determined by
parameter α



Standard Form

$$t \leftarrow \frac{\tilde{t}}{\alpha(1-\tilde{t})+\tilde{t}}, \text{ i.e., } (1-t) \leftarrow \frac{\alpha(1-\tilde{t})}{\alpha(1-\tilde{t})+\tilde{t}}$$

Standard Form

$$t \leftarrow \frac{\tilde{t}}{\alpha(1-\tilde{t})+\tilde{t}}, \text{ i.e., } (1-t) \leftarrow \frac{\alpha(1-\tilde{t})}{\alpha(1-\tilde{t})+\tilde{t}}$$

$$\begin{aligned} \mathbf{f}^{(eucl)}(t) &= \frac{\left(\frac{\alpha(1-\tilde{t})}{\alpha(1-\tilde{t})+\tilde{t}}\right)^2 \omega_0 \mathbf{p}_0 + 2 \left(\frac{\alpha(1-\tilde{t})}{\alpha(1-\tilde{t})+\tilde{t}}\right) \left(\frac{\tilde{t}}{\alpha(1-\tilde{t})+\tilde{t}}\right) \omega_1 \mathbf{p}_1 + \left(\frac{\tilde{t}}{\alpha(1-\tilde{t})+\tilde{t}}\right)^2 \omega_2 \mathbf{p}_2}{\left(\frac{\alpha(1-\tilde{t})}{\alpha(1-\tilde{t})+\tilde{t}}\right)^2 \omega_0 + 2 \left(\frac{\alpha(1-\tilde{t})}{\alpha(1-\tilde{t})+\tilde{t}}\right) \left(\frac{\tilde{t}}{\alpha(1-\tilde{t})+\tilde{t}}\right) \omega_1 + \left(\frac{\tilde{t}}{\alpha(1-\tilde{t})+\tilde{t}}\right)^2 \omega_2} \\ &= \frac{\alpha^2(1-\tilde{t})^2 \omega_0 \mathbf{p}_0 + 2\alpha(1-\tilde{t})\tilde{t} \omega_1 \mathbf{p}_1 + \tilde{t}^2 \omega_2 \mathbf{p}_2}{\alpha^2(1-\tilde{t})^2 \omega_0 + 2\alpha(1-\tilde{t})\tilde{t} \omega_1 + \tilde{t}^2 \omega_2} \end{aligned}$$

Standard Form

$$t \leftarrow \frac{\tilde{t}}{\alpha(1-\tilde{t})+\tilde{t}}, \text{ i.e., } (1-t) \leftarrow \frac{\alpha(1-\tilde{t})}{\alpha(1-\tilde{t})+\tilde{t}}$$

$$\begin{aligned} \mathbf{f}^{(eucl)}(t) &= \frac{\left(\frac{\alpha(1-\tilde{t})}{\alpha(1-\tilde{t})+\tilde{t}}\right)^2 \omega_0 \mathbf{p}_0 + 2 \left(\frac{\alpha(1-\tilde{t})}{\alpha(1-\tilde{t})+\tilde{t}}\right) \left(\frac{\tilde{t}}{\alpha(1-\tilde{t})+\tilde{t}}\right) \omega_1 \mathbf{p}_1 + \left(\frac{\tilde{t}}{\alpha(1-\tilde{t})+\tilde{t}}\right)^2 \omega_2 \mathbf{p}_2}{\left(\frac{\alpha(1-\tilde{t})}{\alpha(1-\tilde{t})+\tilde{t}}\right)^2 \omega_0 + 2 \left(\frac{\alpha(1-\tilde{t})}{\alpha(1-\tilde{t})+\tilde{t}}\right) \left(\frac{\tilde{t}}{\alpha(1-\tilde{t})+\tilde{t}}\right) \omega_1 + \left(\frac{\tilde{t}}{\alpha(1-\tilde{t})+\tilde{t}}\right)^2 \omega_2} \\ &= \frac{\alpha^2(1-\tilde{t})^2 \omega_0 \mathbf{p}_0 + 2\alpha(1-\tilde{t})\tilde{t} \omega_1 \mathbf{p}_1 + \tilde{t}^2 \omega_2 \mathbf{p}_2}{\alpha^2(1-\tilde{t})^2 \omega_0 + 2\alpha(1-\tilde{t})\tilde{t} \omega_1 + \tilde{t}^2 \omega_2} \\ &= \frac{\alpha^2 B_0^{(2)}(\tilde{t}) \omega_0 \mathbf{p}_0 + \alpha B_1^{(2)}(\tilde{t}) \omega_1 \mathbf{p}_1 + B_2^{(2)}(\tilde{t}) \omega_2 \mathbf{p}_2}{\alpha^2 B_0^{(2)}(\tilde{t}) \omega_0 + \alpha B_1^{(2)}(\tilde{t}) \omega_1 + B_2^{(2)}(\tilde{t}) \omega_2} \end{aligned}$$

Standard Form

$$\mathbf{f}^{(eucl)}(t) = \frac{\alpha^2 B_0^{(2)}(\tilde{t})\omega_0 \mathbf{p}_0 + \alpha B_1^{(2)}(\tilde{t})\omega_1 \mathbf{p}_1 + B_2^{(2)}(\tilde{t})\omega_2 \mathbf{p}_2}{\alpha^2 B_0^{(2)}(\tilde{t})\omega_0 + \alpha B_1^{(2)}(\tilde{t})\omega_1 + B_2^{(2)}(\tilde{t})\omega_2}$$

$$\text{let } \alpha = \sqrt{\frac{\omega_2}{\omega_0}} \text{ (assume } 0 \leq \frac{\omega_2}{\omega_0} < \infty)$$

Standard Form

$$\mathbf{f}^{(eucl)}(t) = \frac{\alpha^2 B_0^{(2)}(\tilde{t}) \omega_0 \mathbf{p}_0 + \alpha B_1^{(2)}(\tilde{t}) \omega_1 \mathbf{p}_1 + B_2^{(2)}(\tilde{t}) \omega_2 \mathbf{p}_2}{\alpha^2 B_0^{(2)}(\tilde{t}) \omega_0 + \alpha B_1^{(2)}(\tilde{t}) \omega_1 + B_2^{(2)}(\tilde{t}) \omega_2}$$

let $\alpha = \sqrt{\frac{\omega_2}{\omega_0}}$ (assume $0 \leq \frac{\omega_2}{\omega_0} < \infty$)

$$\begin{aligned} \mathbf{f}^{(eucl)}(t) &= \frac{B_0^{(2)}(\tilde{t}) \left(\sqrt{\frac{\omega_2}{\omega_0}} \right)^2 \omega_0 \mathbf{p}_0 + B_1^{(2)}(\tilde{t}) \sqrt{\frac{\omega_2}{\omega_0}} \omega_1 \mathbf{p}_1 + B_2^{(2)}(\tilde{t}) \omega_2 \mathbf{p}_2}{B_0^{(2)}(\tilde{t}) \left(\sqrt{\frac{\omega_2}{\omega_0}} \right)^2 \omega_0 + B_1^{(2)}(\tilde{t}) \sqrt{\frac{\omega_2}{\omega_0}} \omega_1 + B_2^{(2)}(\tilde{t}) \omega_2} \\ &= \frac{B_0^{(2)}(\tilde{t}) \omega_2 \mathbf{p}_0 + B_1^{(2)}(\tilde{t}) \sqrt{\frac{\omega_2}{\omega_0}} \omega_1 \mathbf{p}_1 + B_2^{(2)}(\tilde{t}) \omega_2 \mathbf{p}_2}{B_0^{(2)}(\tilde{t}) \omega_2 + B_1^{(2)}(\tilde{t}) \sqrt{\frac{\omega_2}{\omega_0}} \omega_1 + B_2^{(2)}(\tilde{t}) \omega_2} \end{aligned}$$

Standard Form

$$\mathbf{f}^{(eucl)}(t) = \frac{B_0^{(2)}(\tilde{t})\omega_2\mathbf{p}_0 + B_1^{(2)}(\tilde{t})\sqrt{\frac{\omega_2}{\omega_0}}\omega_1\mathbf{p}_1 + B_2^{(2)}(\tilde{t})\omega_2\mathbf{p}_2}{B_0^{(2)}(\tilde{t})\omega_2 + B_1^{(2)}(\tilde{t})\sqrt{\frac{\omega_2}{\omega_0}}\omega_1 + B_2^{(2)}(\tilde{t})\omega_2}$$

Standard Form

$$\begin{aligned} \mathbf{f}^{(eucl)}(t) &= \frac{B_0^{(2)}(\tilde{t})\omega_2\mathbf{p}_0 + B_1^{(2)}(\tilde{t})\sqrt{\frac{\omega_2}{\omega_0}}\omega_1\mathbf{p}_1 + B_2^{(2)}(\tilde{t})\omega_2\mathbf{p}_2}{B_0^{(2)}(\tilde{t})\omega_2 + B_1^{(2)}(\tilde{t})\sqrt{\frac{\omega_2}{\omega_0}}\omega_1 + B_2^{(2)}(\tilde{t})\omega_2} \\ &= \frac{B_0^{(2)}(\tilde{t})\mathbf{p}_0 + B_1^{(2)}(\tilde{t})\sqrt{\frac{1}{\omega_0\omega_2}}\omega_1\mathbf{p}_1 + B_2^{(2)}(\tilde{t})\mathbf{p}_2}{B_0^{(2)}(\tilde{t}) + B_1^{(2)}(\tilde{t})\sqrt{\frac{1}{\omega_0\omega_2}}\omega_1 + B_2^{(2)}(\tilde{t})} \\ &= \frac{B_0^{(2)}(\tilde{t})\mathbf{p}_0 + B_1^{(2)}(\tilde{t})\omega\mathbf{p}_1 + B_2^{(2)}(\tilde{t})\mathbf{p}_2}{B_0^{(2)}(\tilde{t}) + B_1^{(2)}(\tilde{t})\omega + B_2^{(2)}(\tilde{t})} \quad \text{with } \omega := \sqrt{\frac{1}{\omega_0\omega_2}}\omega_1 \end{aligned}$$

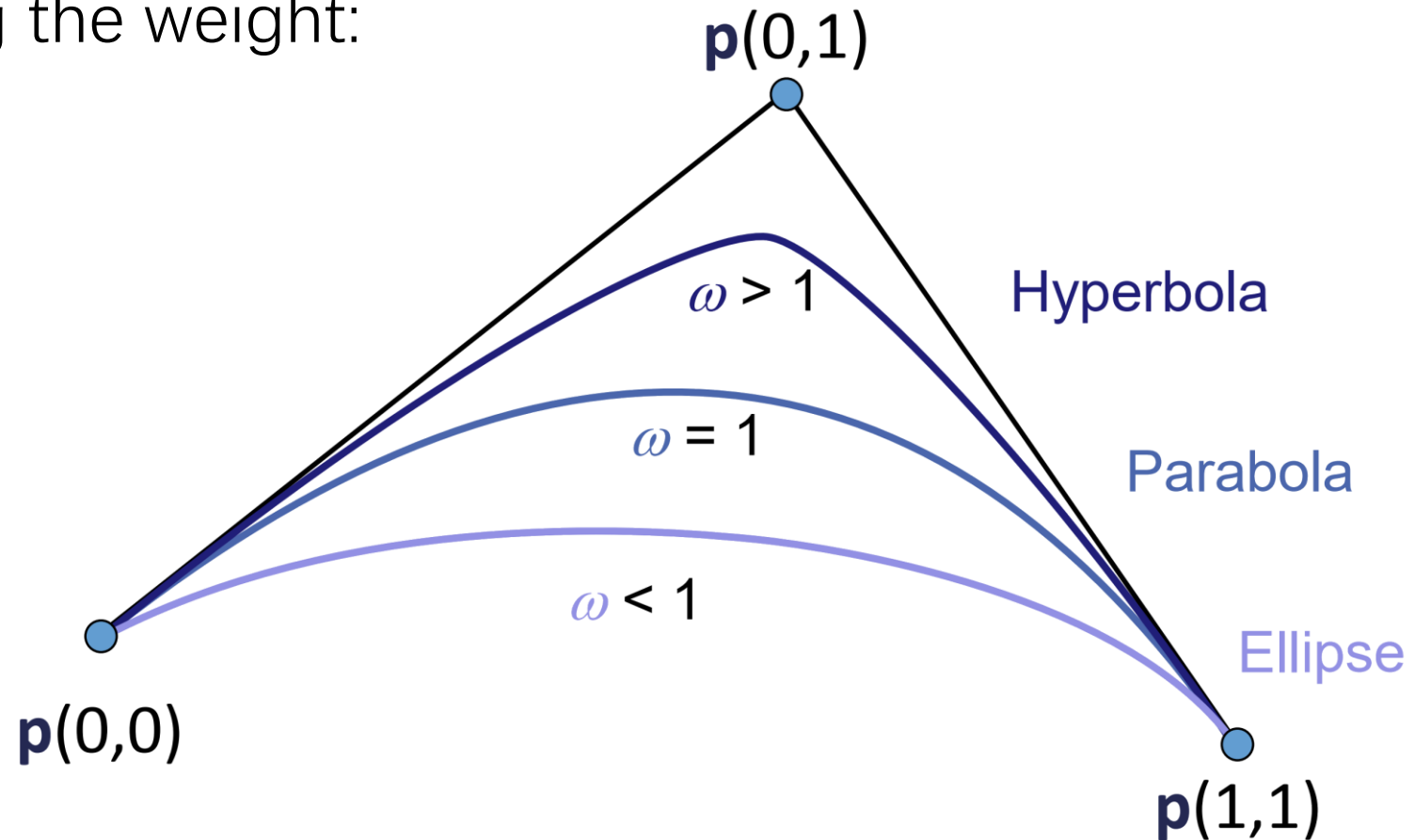
Standard Form

Consequence:

- It is sufficient to specify the weight of the inner point
- We can w.l.o.g. set $\omega_0 = \omega_2 = 1$, $\omega_1 = \omega$
- This form of a quadratic Bézier curve is called the *standard form* or the *normal form*
- Choices:
 - $\omega < 1$: ellipse segment
 - $\omega = 1$: parabola segment (non-rational curve)
 - $\omega > 1$: hyperbola segment

Illustration

- Changing the weight:



Conversion to Implicit Form

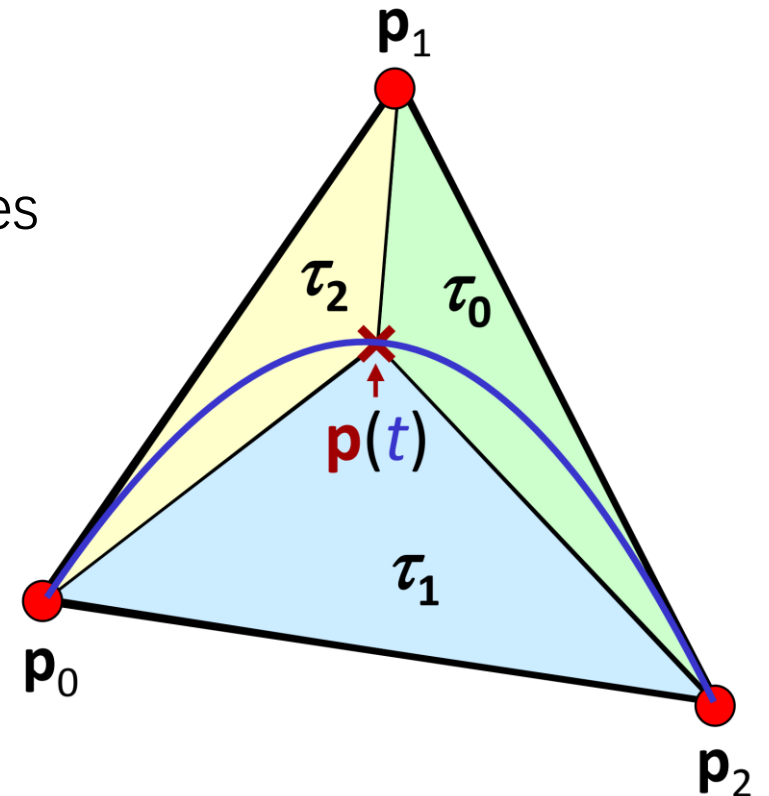
Convert parametric to implicit form

- In order to show the shape condition
- For distance computation / inside-outside tests

Express curve in barycentric coordinates

- Curve can be expressed in barycentric coordinates (linear transform)

$$f(t) = \tau_0(t)\mathbf{p}_0 + \tau_1(t)\mathbf{p}_1 + \tau_2(t)\mathbf{p}_2$$



Conversion to Implicit Form

Compare the coefficients

$$f(t) = \tau_0(t)\mathbf{p}_0 + \tau_1(t)\mathbf{p}_1 + \tau_2(t)\mathbf{p}_2$$

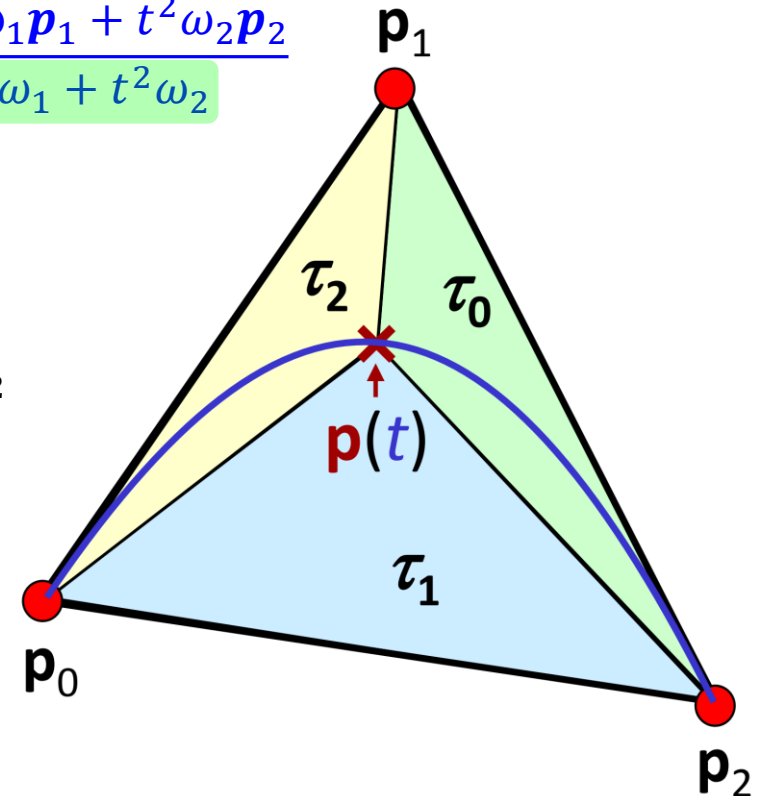
$$f^{(eucl)}(t) = \frac{(1-t)^2\omega_0\mathbf{p}_0 + 2t(1-t)\omega_1\mathbf{p}_1 + t^2\omega_2\mathbf{p}_2}{(1-t)^2\omega_0 + 2t(1-t)\omega_1 + t^2\omega_2}$$

$$\tau_0 = \frac{\omega_0(1-t)^2}{\omega(t)}$$

$$\tau_1 = \frac{2\omega_1 t(1-t)}{\omega(t)}$$

$$\tau_2 = \frac{\omega_2 t^2}{\omega(t)}$$

$$\omega(t) = (1-t)^2 \omega_0 + 2t(1-t)\omega_1 + t^2 \omega_2$$



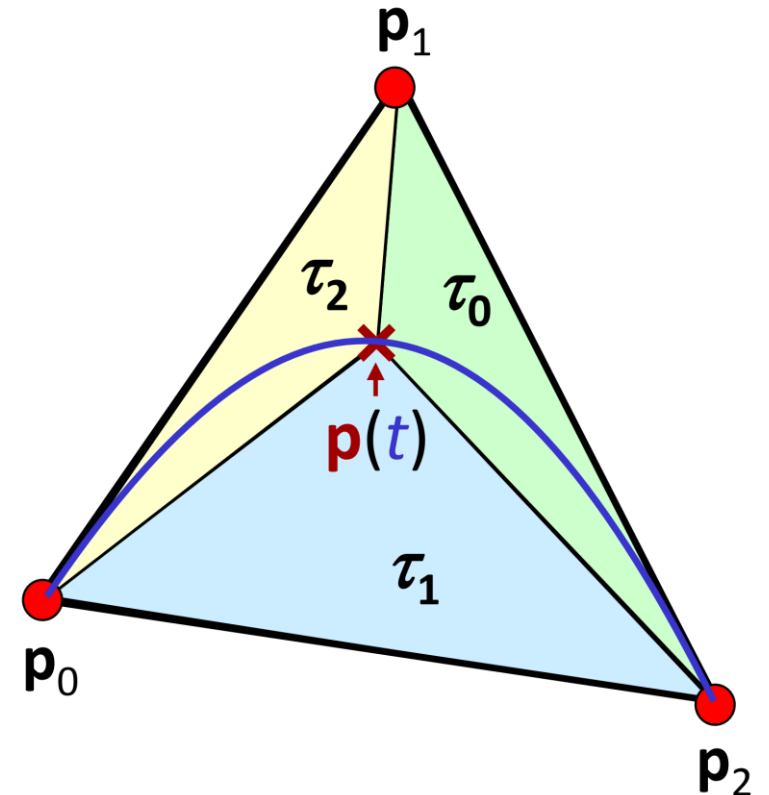
Conversion to Implicit Form

Solving for t , $1 - t$

$$\tau_0 = \frac{\omega_0(1-t)^2}{\omega(t)} \Rightarrow 1 - t = \sqrt{\frac{\tau_0(t)\omega(t)}{\omega_0}}$$

$$\tau_1 = \frac{2\omega_1 t(1-t)}{\omega(t)}$$

$$\tau_2 = \frac{\omega_2 t^2}{\omega(t)} \Rightarrow t = \sqrt{\frac{\tau_2(t)\omega(t)}{\omega_2}}$$



Conversion to Implicit Form

Solving for t , $1 - t$

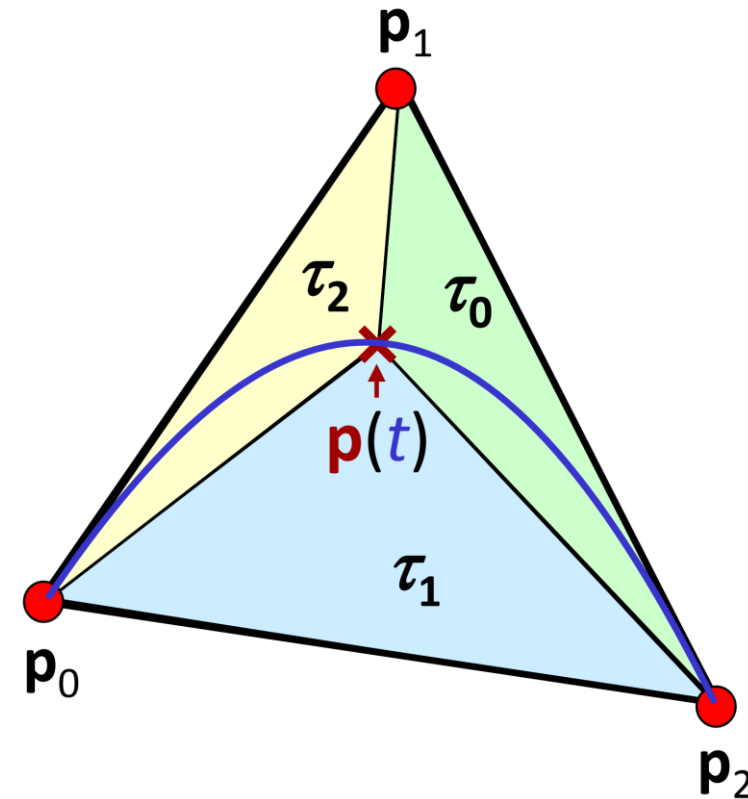
$$\tau_0 = \frac{\omega_0(1-t)^2}{\omega(t)} \Rightarrow 1 - t = \sqrt{\frac{\tau_0(t)\omega(t)}{\omega_0}}$$

$$\tau_1 = \frac{2\omega_1 t(1-t)}{\omega(t)}$$

$$\tau_2 = \frac{\omega_2 t^2}{\omega(t)} \Rightarrow t = \sqrt{\frac{\tau_2(t)\omega(t)}{\omega_2}}$$

$$\tau_1 = \frac{2\omega_1 t(1-t)}{\omega(t)} = 2 \frac{\omega_1}{\omega(t)} \sqrt{\frac{\tau_0(t)\omega(t)}{\omega_0} \frac{\tau_2(t)\omega(t)}{\omega_2}} = 2\omega_1 \sqrt{\frac{\tau_0(t)\tau_2(t)}{\omega_0\omega_2}}$$

$$\Rightarrow \frac{\tau_1^2(t)}{\tau_2(t)\tau_0(t)} = \frac{4\omega_1^2}{\omega_0\omega_2}$$



Conversion to Implicit Form

More algebra ...

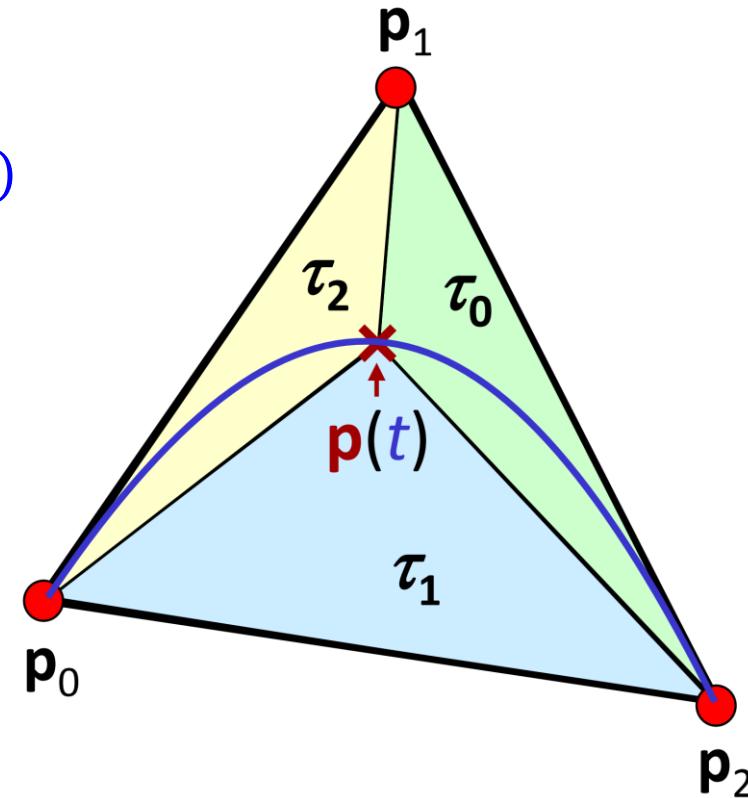
$$\frac{\tau_1^2(t)}{\tau_2(t)\tau_0(t)} = \frac{4\omega_1^2}{\omega_0\omega_2}$$

Using $\tau_2(t) = 1 - \tau_0(t) - \tau_1(t)$, we get

$$(\omega_0\omega_2)\tau_1^2(t) = 4\omega_1^2\tau_2(t)\tau_0(t) = 4\omega_1^2(1 - \tau_0(t) - \tau_1(t))\tau_0(t)$$

$$= 4\omega_1^2(\tau_0(t) - \tau_0^2(t) - \tau_0(t)\tau_1(t))$$

$$\Rightarrow (\omega_0\omega_2)\tau_1^2(t) + 4\omega_1^2\tau_1(t)\tau_0(t) + 4\omega_1^2\tau_0^2(t) - 4\omega_1^2\tau_0(t) = 0$$



Conversion to Implicit Form

More algebra ...

$$\frac{\tau_1^2(t)}{\tau_2(t)\tau_0(t)} = \frac{4\omega_1^2}{\omega_0\omega_2}$$

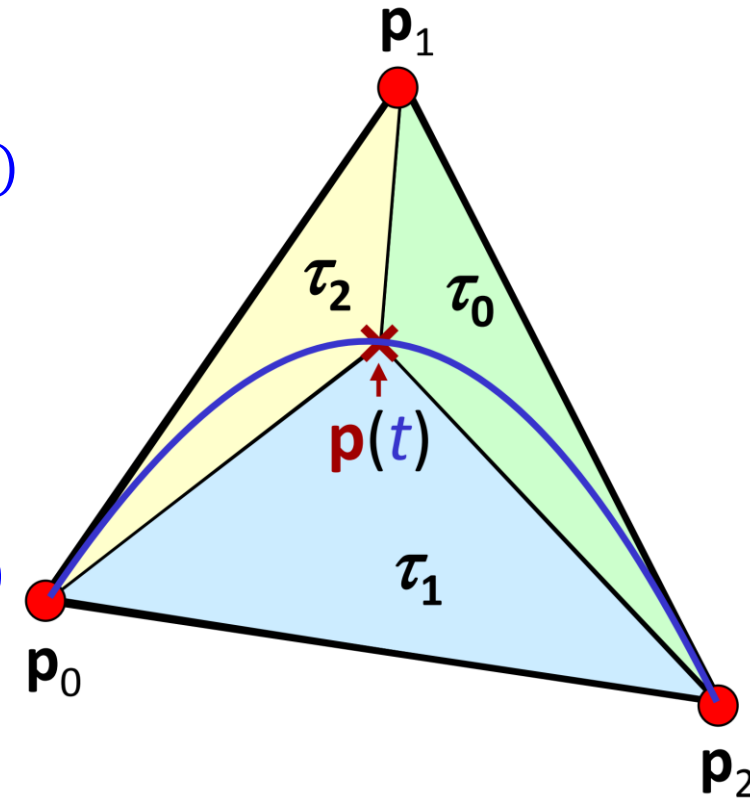
Using $\tau_2(t) = 1 - \tau_0(t) - \tau_1(t)$, we get

$$(\omega_0\omega_2)\tau_1^2(t) = 4\omega_1^2\tau_2(t)\tau_0(t) = 4\omega_1^2(1 - \tau_0(t) - \tau_1(t))\tau_0(t)$$

$$= 4\omega_1^2(\tau_0(t) - \tau_0^2(t) - \tau_0(t)\tau_1(t))$$

$$\Rightarrow (\omega_0\omega_2)\tau_1^2(t) + 4\omega_1^2\tau_1(t)\tau_0(t) + 4\omega_1^2\tau_0^2(t) - 4\omega_1^2\tau_0(t) = 0$$

$$ax^2 + bxy + cy^2 + 0x + ey + 0 = 0$$



Classification

Eigenvalue argument led to:

- Parabola requires $b^2 = 4ac$ in $ax^2 + bxy + cy^2 + dx + ey + f = 0$
- In our case:

$$(\omega_0\omega_2)\tau_1^2(t) + 4\omega_1^2\tau_1(t)\tau_0(t) + 4\omega_1^2\tau_0^2(t) - 4\omega_1^2\tau_0(t) = 0$$

i.e.

$$4(\omega_0\omega_2)(4\omega_1^2) = (4\omega_1^2)^2$$

$$\Leftrightarrow \omega_0\omega_2 = \omega_1^2$$

- Standard form: $\omega_0 = \omega_2 = 1$
 $\Rightarrow \omega_1 = 1$

Classification

Similarly, it follows that

$\omega_1 < 1 \rightarrow$ Ellipse

$\omega_1 = 1 \rightarrow$ Parabola

$\omega_1 > 1 \rightarrow$ Hyperbola

Towards Dual Conic Sections

Rational quadratic curves – conic sections

- Consider a rational quadratic curve in normal form for $t \in [0,1]$:

$$\mathbf{x}(t) = \frac{(1-t)^2 \cdot \mathbf{b}_0 + 2 \cdot t \cdot (1-t) \cdot \omega \cdot \mathbf{b}_1 + t^2 \cdot \mathbf{b}_2}{(1-t)^2 + 2 \cdot t \cdot (1-t) \cdot \omega + t^2}$$

Dual Conic Sections

Rational quadratic curves – conic sections

- Dual conic section $t \in \mathbb{R} \setminus [0,1]$
- Choice of reparameterization

$$s(t) = \hat{t} = \frac{t}{2 \cdot t - 1} \Rightarrow (1 - \hat{t}) = \frac{t - 1}{2 \cdot t - 1}$$

\hat{t} changes from 0 to $-\infty \Leftrightarrow t$ changes from 0 to $\frac{1}{2}$

\hat{t} changes from ∞ to 1 $\Leftrightarrow t$ changes from $\frac{1}{2}$ to 1

Dual Conic Sections

The following applies:

$$\boldsymbol{x}(s(t)) = \boldsymbol{x}(\hat{t})$$

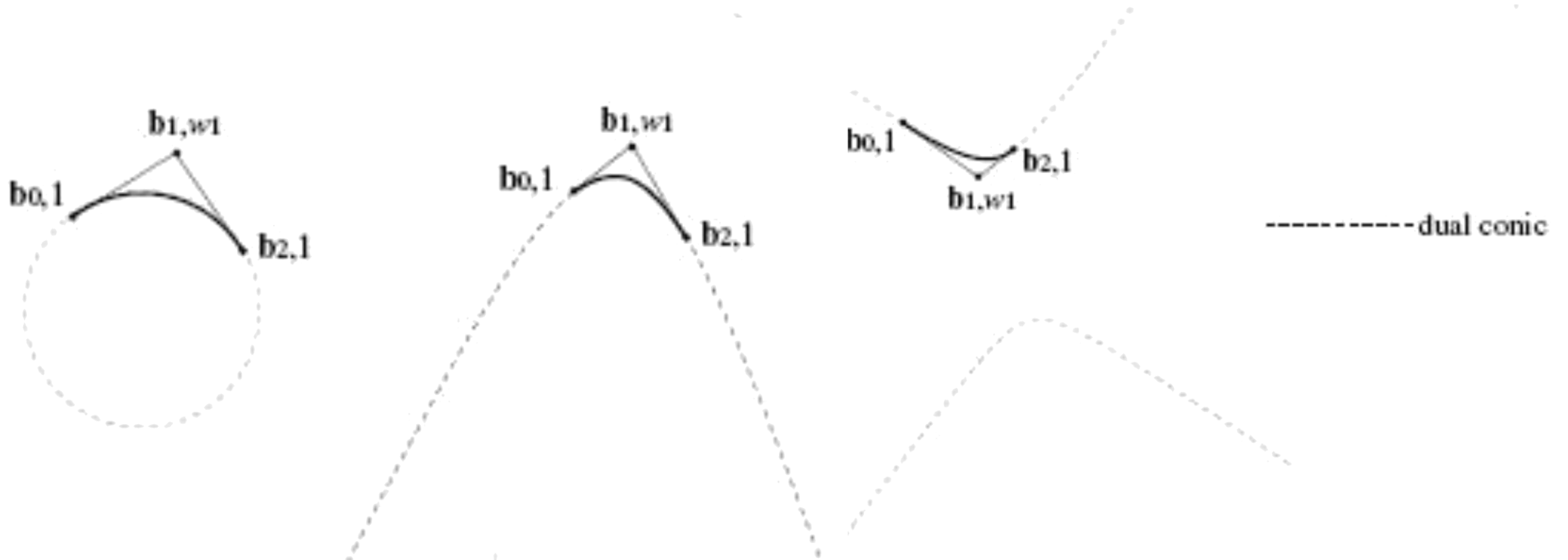
$$= \frac{(1 - \hat{t})^2 \cdot b_0 + 2 \cdot \hat{t} \cdot (1 - \hat{t}) \cdot \omega \cdot b_1 + \hat{t}^2 \cdot b_2}{(1 - \hat{t})^2 + 2 \cdot \hat{t} \cdot (1 - \hat{t}) \cdot \omega + \hat{t}^2}$$

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

- → Dual conic section arises in **Normal form** by negation of ω

Dual Conic Sections

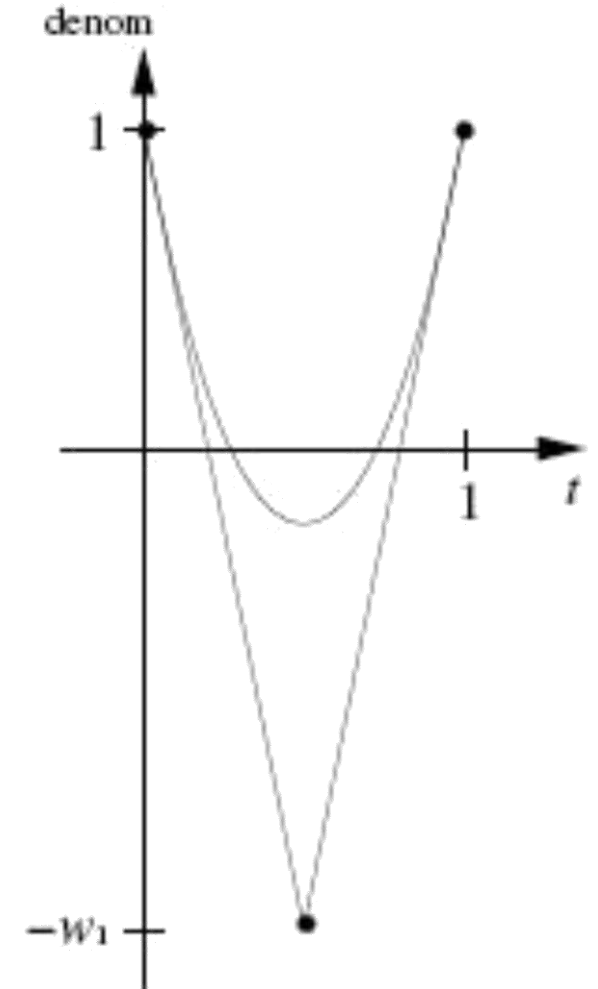
Examples:



Dual Conic Sections

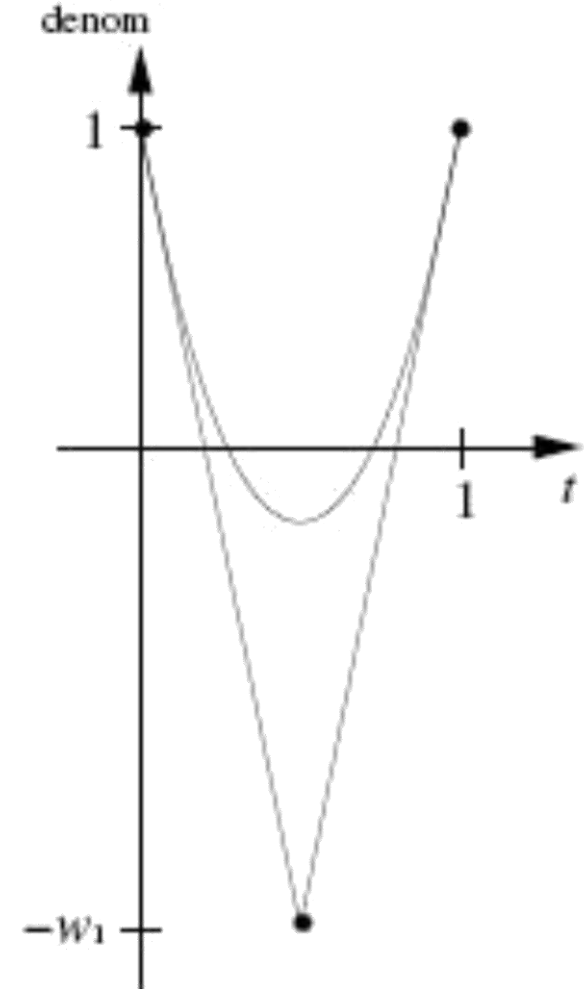
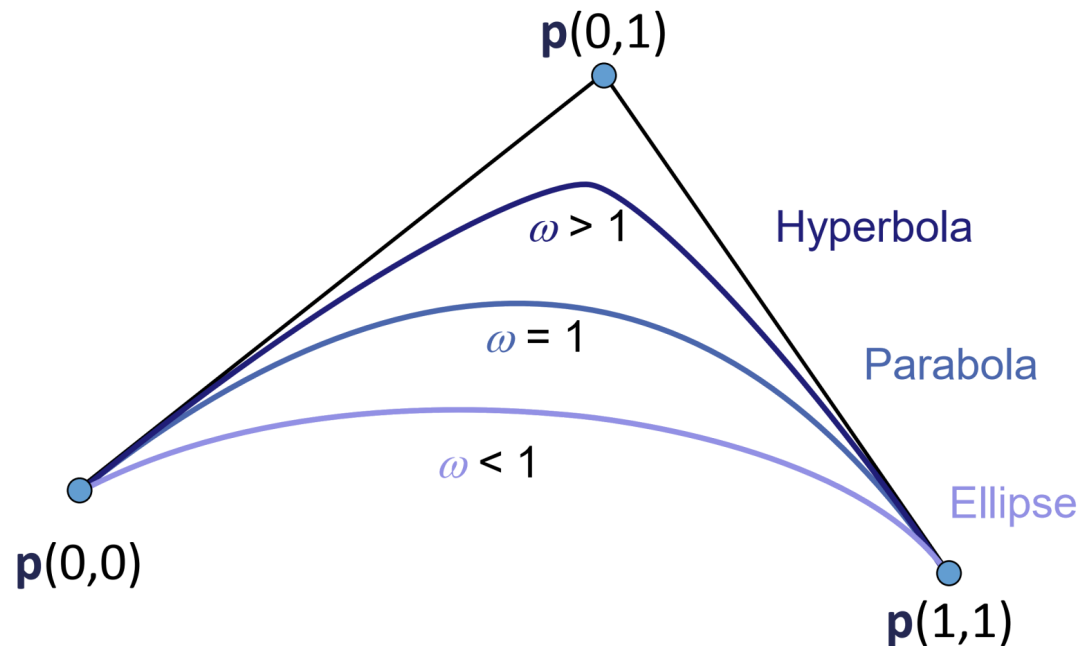
Classification of conic sections:

- By means of the dual conic section
- Consider singularities of the denominator function $(1 - t)^2 - 2 \cdot t \cdot (1 - t) \cdot \omega + t^2$ in $[0,1]$



Rational Bézier curves

- $\omega < 1 \rightarrow$ no singularities \rightarrow ellipse
- $\omega = 1 \rightarrow$ one singularities \rightarrow parabola
- $\omega > 1 \rightarrow$ two singularities \rightarrow hyperbola



Circle in Bézier Form

- Quadratic rational polynomial:

$$f(t) = \frac{1}{1+t^2} \begin{pmatrix} 1-t^2 \\ 2t \end{pmatrix}, \quad t = \tan \frac{\varphi}{2}, \quad \varphi \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$$

- Conversion to Bézier basis

$$B_0^{(2)} = (1-t)^2 = 1 - 2t + t^2 := [1 \quad -2 \quad 1]^T$$

$$B_1^{(2)} = 2t(1-t) = 2t - 2t^2 := [0 \quad 2 \quad -2]^T$$

$$B_2^{(2)} = t^2 := [0 \quad 0 \quad 1]^T$$

$$1 - t^2 := [1 \quad 0 \quad -1]^T$$

$$2t := [0 \quad 2 \quad 0]^T$$

$$1 + t^2 := [1 \quad 0 \quad 1]^T$$

Circle in Bézier Form

Conversion to Bézier basis: Method 1

$$B_0^{(2)} = (1 - t)^2 = 1 - 2t + t^2 := [1 \quad -2 \quad 1]^T$$

$$B_1^{(2)} = 2t(1 - t) = 2t - 2t^2 := [0 \quad 2 \quad -2]^T$$

$$B_2^{(2)} = t^2 := [0 \quad 0 \quad 1]^T$$

$$1 - t^2 := [1 \quad 0 \quad -1]^T$$

$$2t := [0 \quad 2 \quad 0]^T$$

$$1 + t^2 := [1 \quad 0 \quad 1]^T$$

Comparison yields:

$$1 - t^2 = B_0^{(2)} + B_1^{(2)}$$

$$2t = B_1^{(2)} + 2B_2^{(2)}$$

$$1 + t^2 = B_0^{(2)} + B_1^{(2)} + 2B_2^{(2)}$$

$$\mathbf{f}^{(hom)}(t) = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} B_0^{(2)} + \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} B_1^{(2)} + \begin{pmatrix} 0 \\ 2 \\ 2 \end{pmatrix} B_2^{(2)}$$

Circle in Bézier Form

Conversion to Bézier basis: Method 2

Use polar forms:

$$1 - t^2 \Rightarrow f_0 = 1 - t_1 t_2$$

$$2t \Rightarrow f_1 = t_1 + t_2$$

$$1 + t^2 \Rightarrow f_2 = 1 + t_1 t_2$$

And then evaluate at (0,0), (0,1), (1,1)

Circle in Bézier Form

- Result:

$$f(t) = \frac{\binom{1}{0} B_0^{(2)}(t) + \binom{1}{1} B_1^{(2)}(t) + \binom{0}{2} B_2^{(2)}(t)}{B_0^{(2)}(t) + B_1^{(2)}(t) + 2B_2^{(2)}(t)}$$

- Parameters:

$$t = \tan \frac{\varphi}{2} \Rightarrow \varphi = 2 \arctan t$$

$$t \in [0,1] \rightarrow \varphi \in \left[0, \frac{\pi}{2}\right]$$

Circle in Bézier Form

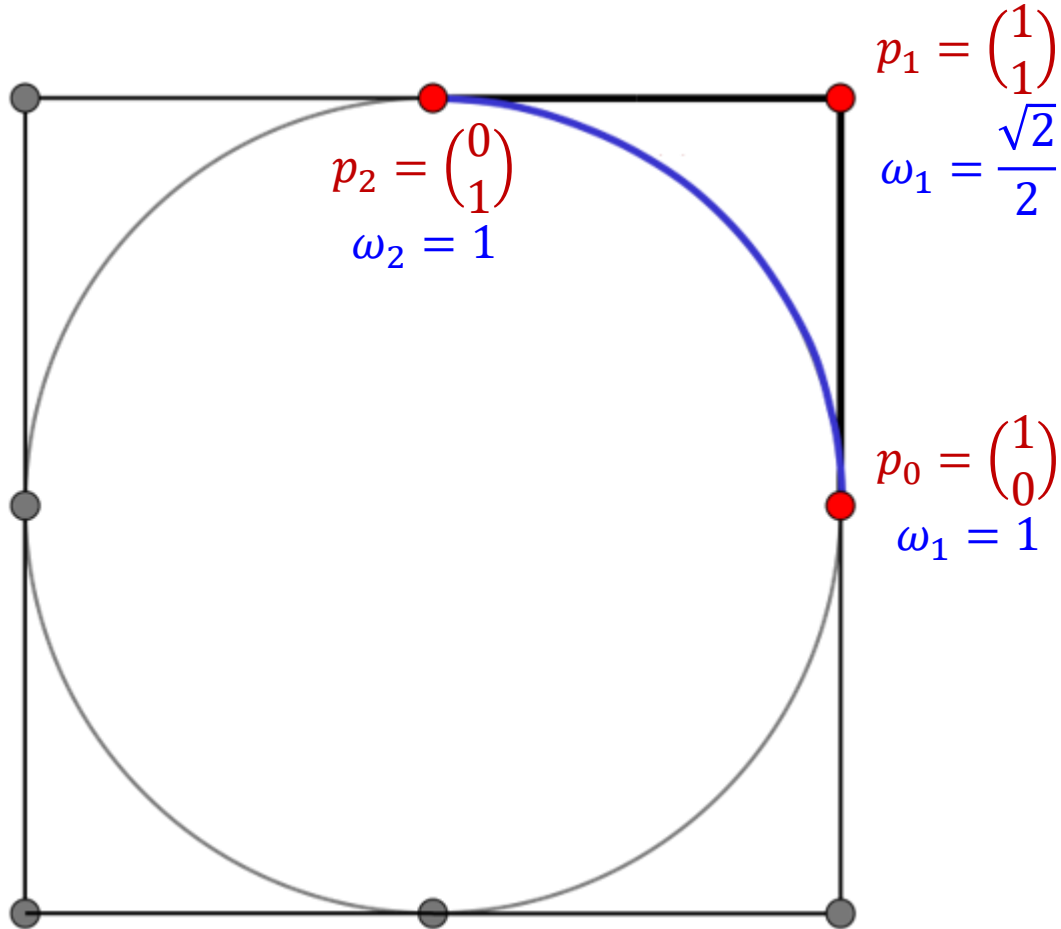
Standard Form:

$$\mathbf{f}(t) = \frac{B_0^{(2)}(\tilde{t})\mathbf{p}_0 + B_1^{(2)}(\tilde{t})\omega\mathbf{p}_1 + B_2^{(2)}(\tilde{t})\mathbf{p}_2}{B_0^{(2)}(\tilde{t}) + B_1^{(2)}(\tilde{t})\omega + B_2^{(2)}(\tilde{t})} \quad \text{with } \omega := \sqrt{\frac{1}{\omega_0\omega_2}}\omega_1$$

$$\mathbf{f}(t) = \frac{B_0^{(2)} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \frac{\sqrt{2}}{2} B_1^{(2)} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + B_2^{(2)} \begin{pmatrix} 0 \\ 1 \end{pmatrix}}{B_0^{(2)} + \frac{\sqrt{2}}{2} B_1^{(2)} + B_2^{(2)}}$$

Result: Circle in Bézier Form

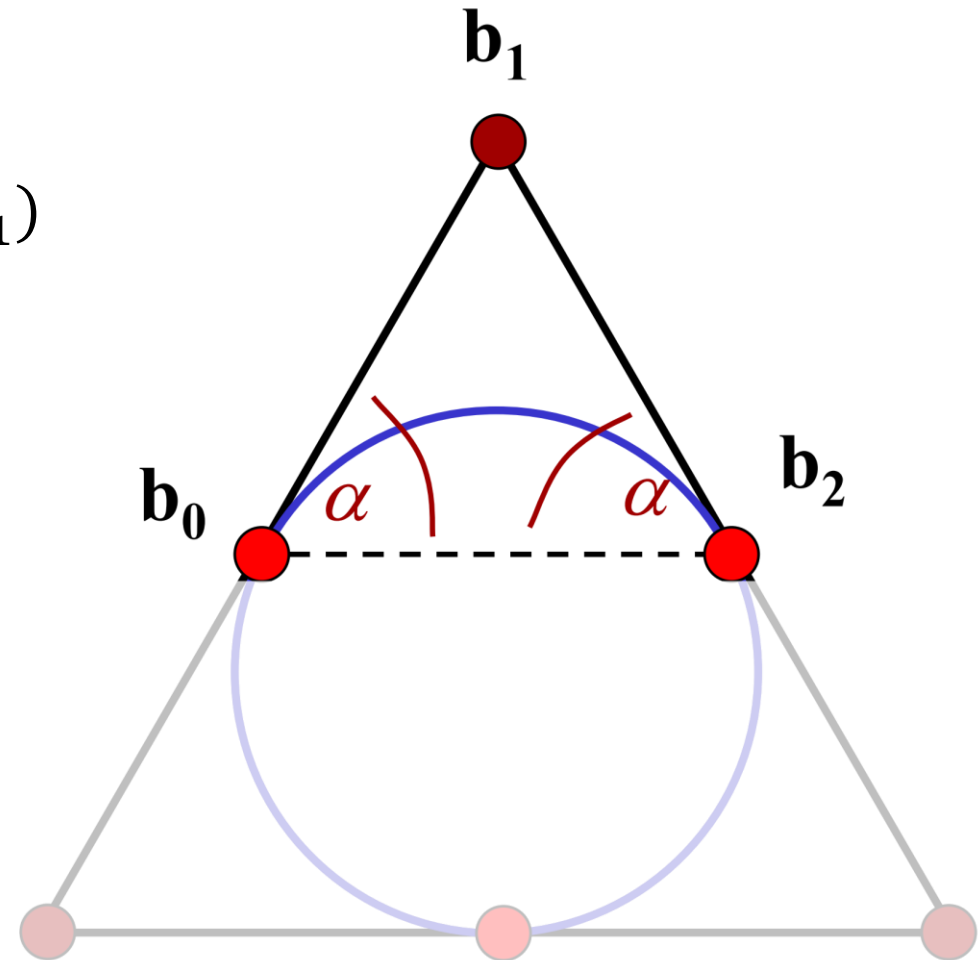
Final Result:



General Circle Segments

Circular arcs:

- Let $\text{dist}(\mathbf{b}_0, \mathbf{b}_1) = \text{dist}(\mathbf{b}_1, \mathbf{b}_2)$
and $\alpha = \text{angle}(\mathbf{b}_0, \mathbf{b}_2, \mathbf{b}_1) = \text{angle}(\mathbf{b}_2, \mathbf{b}_0, \mathbf{b}_1)$
- Then, $\mathbf{x}(t)$ is the circular arc for
$$\omega = \cos \alpha$$
- $\mathbf{x}(t)$ is not arc length parameterized!



Properties, Remarks

Continuity:

- The parameterization is only C^1 , but G^∞
- No arc length parameterization possible
- *Even stronger:* No rational curve other than a straight line can have arc-length parameterization.

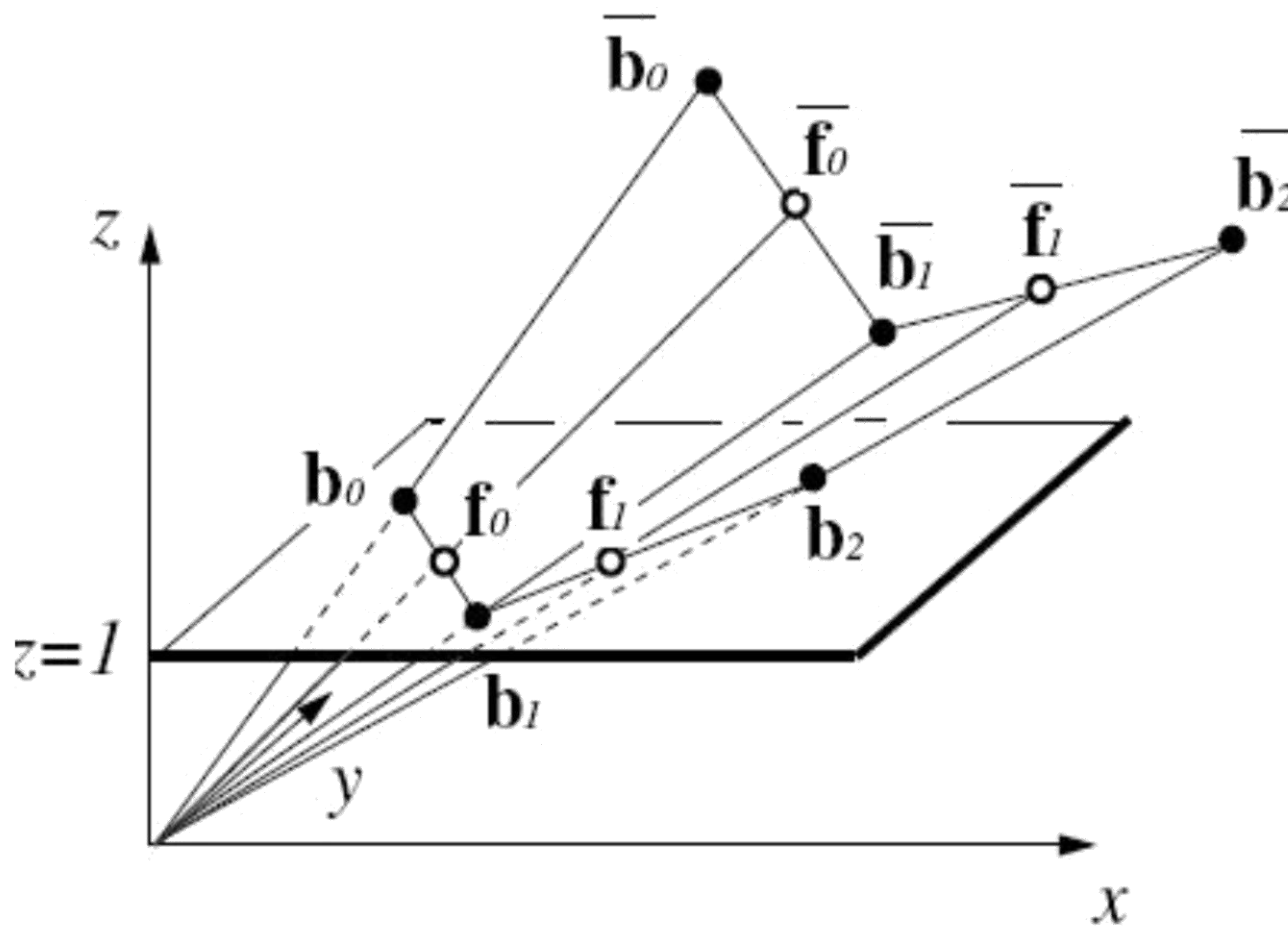
Circles in general degree Bézier splines:

- Simplest solution:
 - Form quadratic circle (segments)
 - Apply degree elevation to obtain the desired degree

Farin Points

$$\bar{f}_i = \frac{1}{2} \cdot (\bar{b}_i + \bar{b}_{i+1})$$

$$f_i = \frac{\omega_i \cdot b_i + \omega_{i+1} \cdot b_{i+1}}{\omega_i + \omega_{i+1}}$$

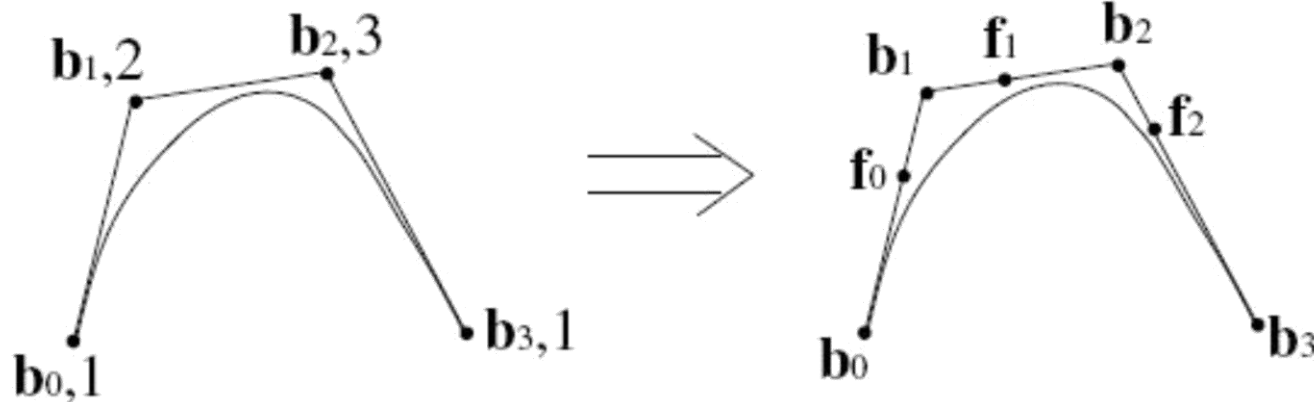


Farin Points

Not the weights themselves determine the curve shape, but the relation of the weights among each other!

The ratio $\frac{\omega_{i+1}}{\omega_i}$ is expressed by point f_i , at line segment $b_i \rightarrow b_{i+1}$ of the Bézier polygon. The following applies:

$$\frac{\omega_{i+1}}{\omega_i} = \frac{\|b_i - f_i\|}{\|b_{i+1} - f_i\|}$$



Farin Points

Alternative technique to specify weights:

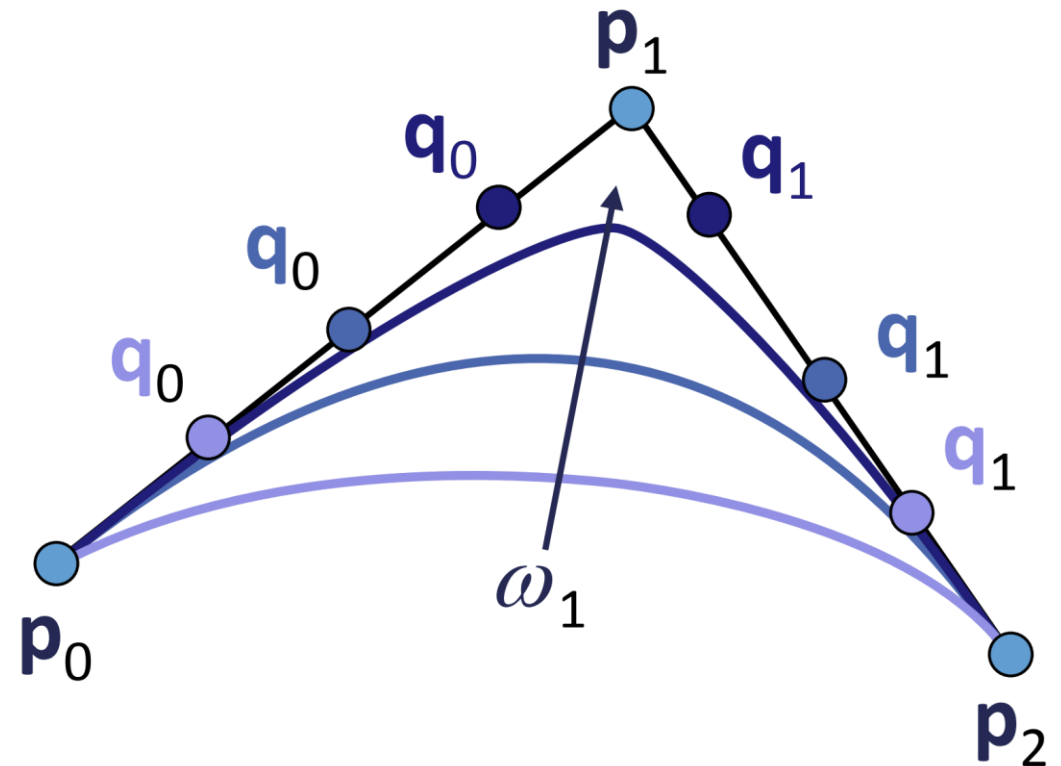
- Farin points or Weight points
- User interface: More intuitive in interactive design

Farin Points:

$$q_0 = \frac{\omega_0 p_0 + \omega_1 p_1}{\omega_0 + \omega_1}, q_1 = \frac{\omega_1 p_1 + \omega_2 p_2}{\omega_1 + \omega_2}$$

Standard Form

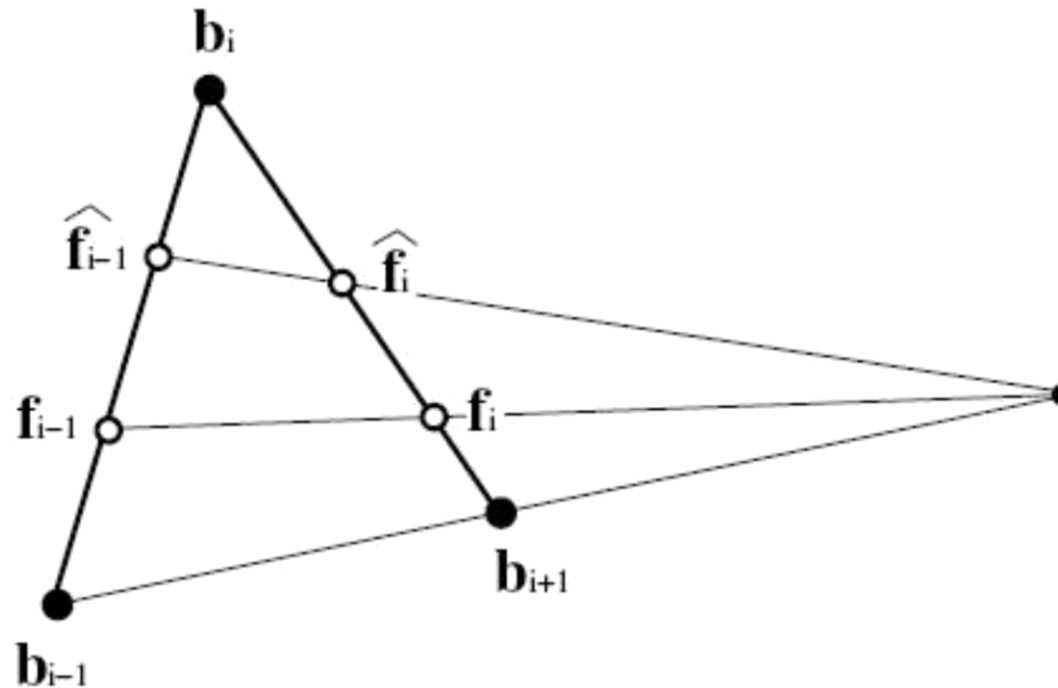
$$q_0 = \frac{p_0 + \omega_1 p_1}{1 + \omega_1}, q_1 = \frac{p_1 + \omega_1 p_2}{1 + \omega_1}$$



Farin Points

Farin Points and changing of weight:

- The change of the weight ω_i into $\hat{\omega}_i$ under preservation of the other weights only changes the Farin points f_{i-1}, f_i to \hat{f}_{i-1}, \hat{f}_i



Rational Curves: Rational Bézier Curves

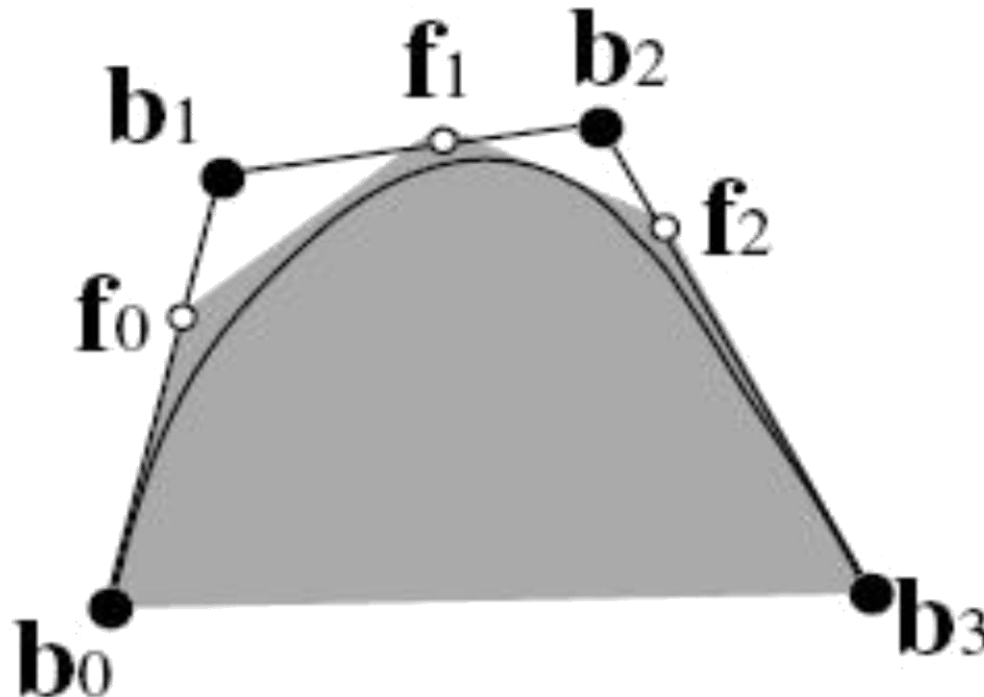
Properties of rational Bézier curves:

- (Let $\omega_i > 0$ for $i = 0, \dots, n$)
- End point interpolation
- Tangent direction in the boundary points corresponds with the direction of the control polygon
- Variation diminishing property

Rational Curves: Rational Bézier Curves

Convex hull properties:

Tightened convex hull properties: the curve lies in the convex hull of $(b_0, f_0, \dots, f_{n-1}, b_n)$



Derivatives

Computing derivatives of rational Bézier curves:

- Straightforward: Apply quotient rule
- A simpler expression can be derived using an algebraic trick:

$$\mathbf{f}(t) = \frac{\sum_{i=0}^n B_i^{(d)}(t) \omega_i \mathbf{p}_i}{\sum_{i=0}^n B_i^{(d)}(t) \omega_i} =: \frac{\mathbf{p}(t)}{\omega(t)}$$

$$\mathbf{f}(t) = \frac{\mathbf{p}(t)}{\omega(t)} \Rightarrow \mathbf{p}(t) = \mathbf{f}(t) \omega(t) \Rightarrow \mathbf{p}'(t) = \mathbf{f}'(t) \omega(t) + \mathbf{f}(t) \omega'(t)$$

$$\Rightarrow \mathbf{f}'(t) \omega(t) = \mathbf{p}'(t) - \mathbf{f}(t) \omega'(t) \Rightarrow \mathbf{f}'(t) = \frac{\mathbf{p}'(t) - \mathbf{f}(t) \omega'(t)}{\omega(t)}$$

Derivatives

At the end points:

$$f'(t) = \frac{\mathbf{p}'(t) - \omega'(t)\mathbf{f}(t)}{\omega(t)}$$

$$f'(0) = \frac{\mathbf{p}'(0) - \omega'(0)\mathbf{f}(0)}{\omega(0)}$$

$$\begin{aligned} &= \frac{d(\omega_1\mathbf{p}_1 - \omega_0\mathbf{p}_0) - d(\omega_1 - \omega_0)\mathbf{p}_0}{\omega_0} = \frac{d}{\omega_0}(\omega_1\mathbf{p}_1 - \omega_0\mathbf{p}_0 - \omega_1\mathbf{p}_0 + \omega_0\mathbf{p}_0) \\ &= d \frac{\omega_1}{\omega_0} (\mathbf{p}_1 - \mathbf{p}_0) \end{aligned}$$

$$f'(1) = d \frac{\omega_{d-1}}{\omega_d} (\mathbf{p}_d - \mathbf{p}_{d-1})$$

NURBS

Non-Uniform Rational B-Splines

NURBS

NURBS: Rational B-Splines

- Same idea:
 - Control points in homogenous coordinates
 - Evaluate curve in $(d + 1)$ -dimensional space (same as before)
 - For display, divide by ω -component
 - (we can divide anytime)

NURBS

NURBS: Rational B-Splines

- Formally: ($N_i^{(d)}$): B-spline basis function i of degree d)

$$f(t) = \frac{\sum_{i=1}^n N_i^{(d)}(t) \omega_i \mathbf{p}_i}{\sum_{i=1}^n N_i^{(d)}(t) \omega_i}$$

- Knot sequences etc. all remain the same
- de Boor algorithm – similar to rational de Casteljau alg.
 - option 1. – apply separately to numerator, denominator
 - option 2. – normalize weights in each intermediate result
 - the second option is numerically more stable

Computer Aided Geometric Design

Fall Semester 2024

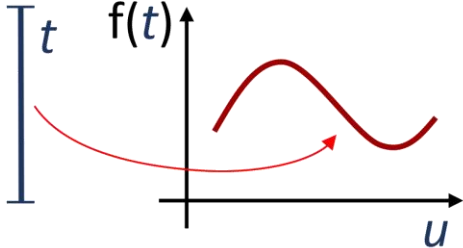
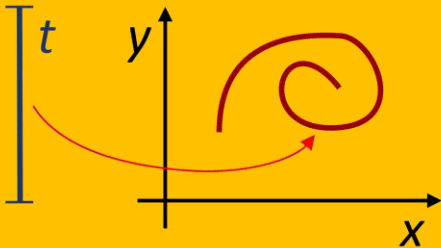
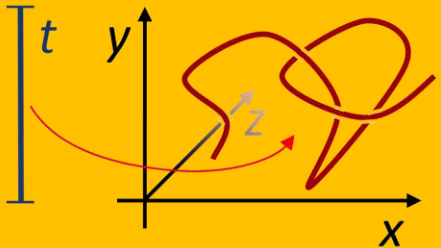
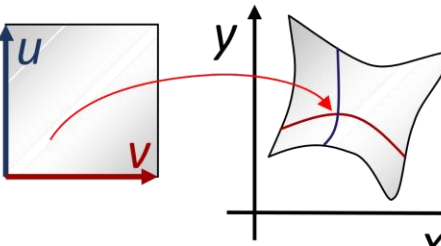
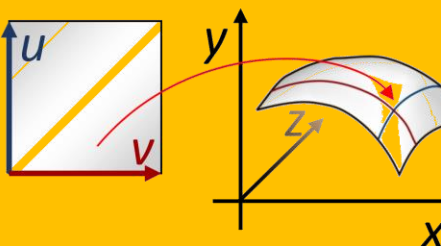
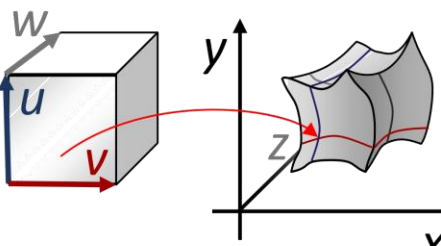
Spline Surfaces

Tensor Product Surfaces · Total Degree Surfaces

陈仁杰

renjiec@ustc.edu.cn

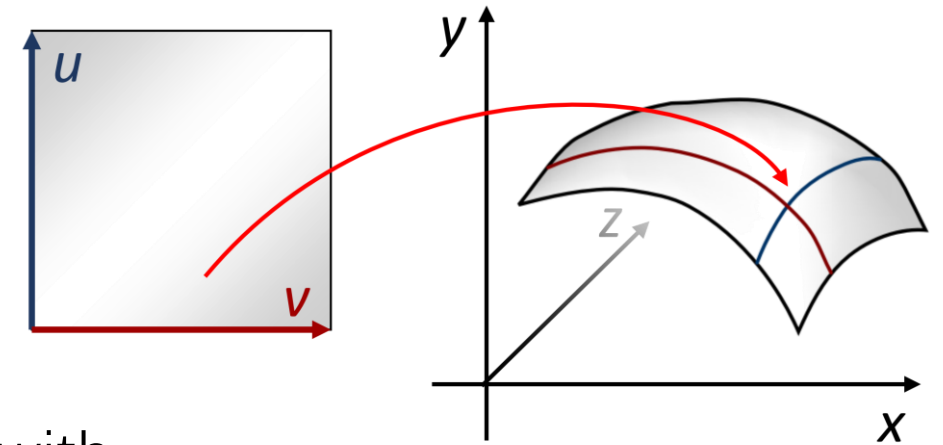
<http://staff.ustc.edu.cn/~renjiec>

	Output: 1D	Output: 2D	Output: 3D
Input: 1D	 <p>Function graph</p>	 <p>Plane curve</p>	 <p>Space curve</p>
Input: 2D		 <p>Plane warp</p>	 <p>Surface</p>
Input: 3D			 <p>Space warp</p>

Spline Surfaces

Parametric spline surfaces:

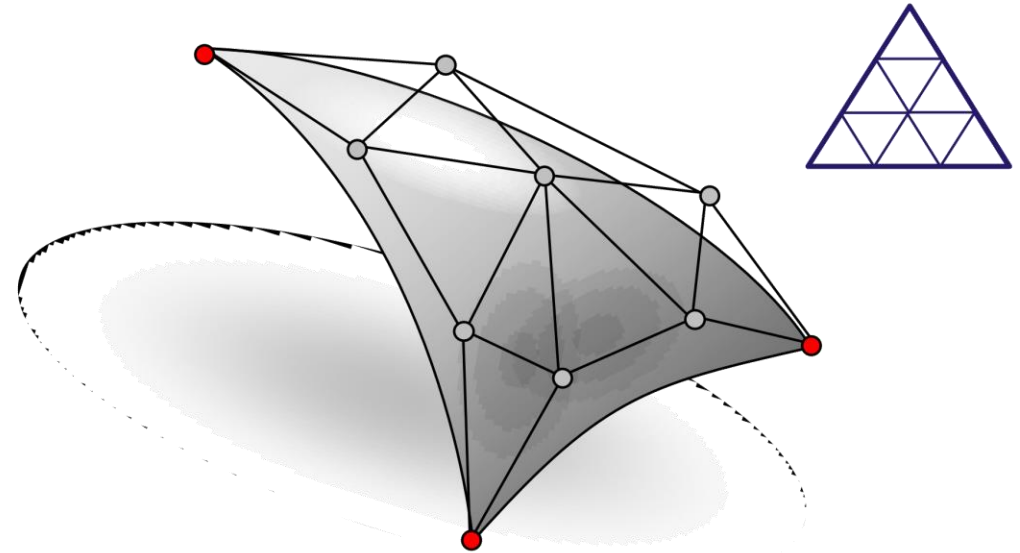
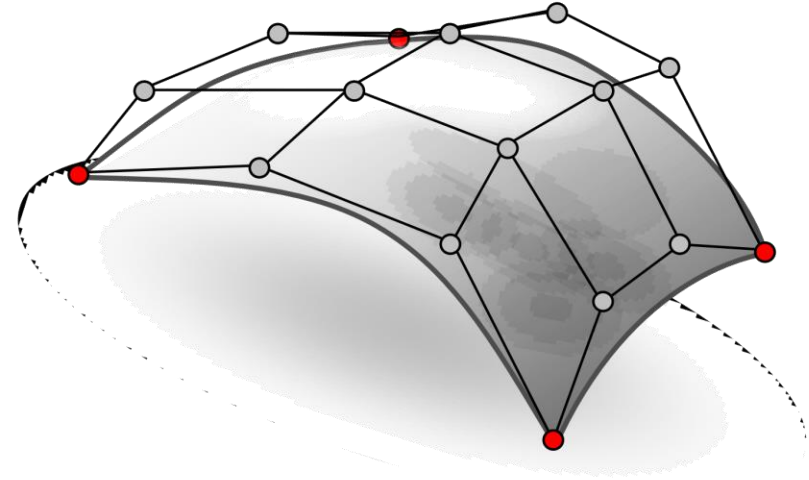
- Two parameter coordinates (u, v)
- Piecewise bivariate polynomials
(rational surfaces \rightarrow homogeneous coords)
- Assemble multiple pieces to form a surface with continuity
- Each piece is called *spline patch*



Spline Surfaces

Two different approaches

- Tensor product surfaces
 - Simple construction
 - Everything carries over from curve case
 - Quad patches
 - Degree anisotropy
- Total degree surfaces
 - Not as straightforward (blossoming will help)
 - Isotropic degree
 - Triangle patches
 - “natural” generalization of curves



Tensor Product Surfaces

Tensor Product Surfaces

Simple Idea

- Given a basis for a one dimensional function space on the interval $t \in [t_0, t_1] \rightarrow \mathbb{R}^d$:

$$\mathbf{B}^{(curv)} := \{b_1(t), \dots, b_n(t)\}$$

- Build a new basis with two parameters by taking all possible products:

$$\mathbf{B}^{(surf)} := \{b_1(u)b_1(v), b_1(u)b_2(v), \dots, b_n(u)b_n(v)\}$$

Tensor Product Surfaces

Tensor product basis

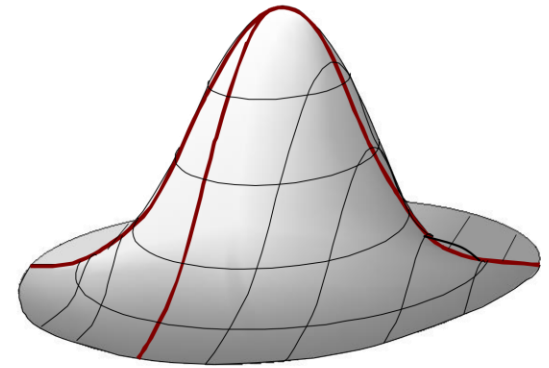
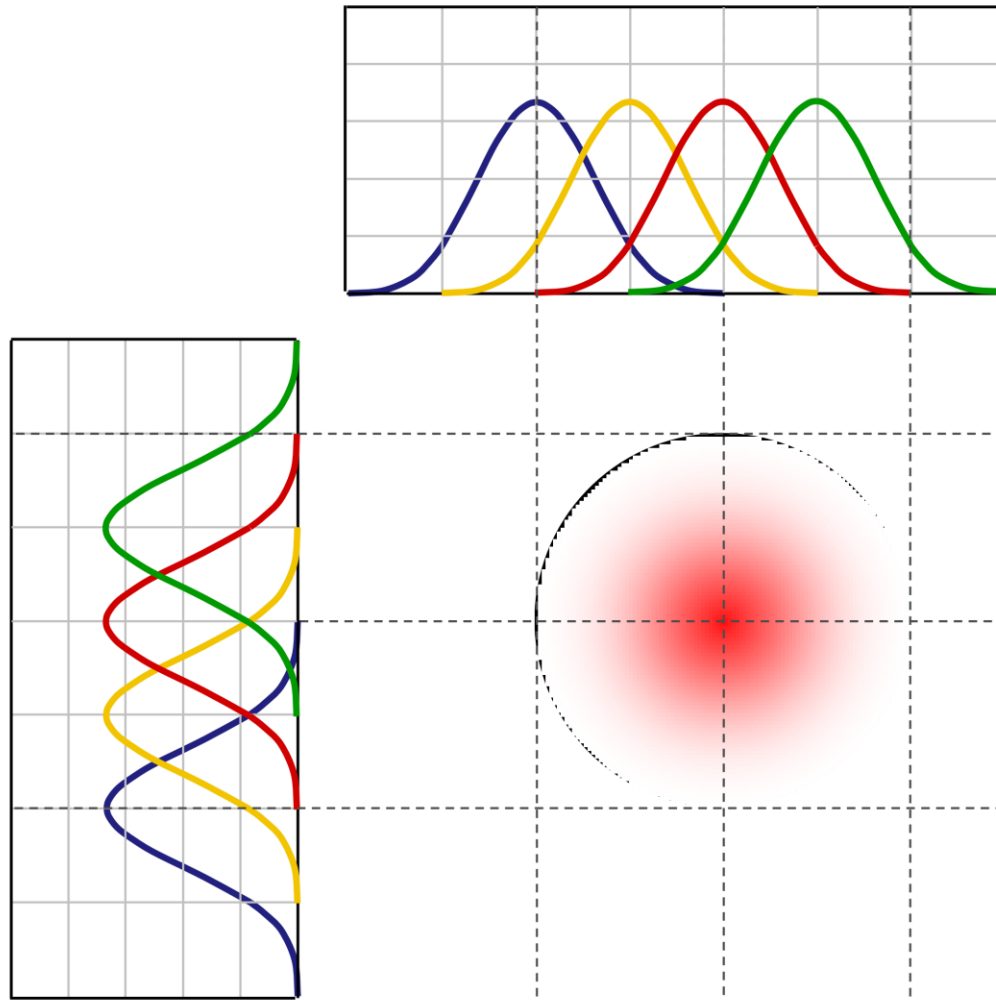
	$b_1(u)$	$b_2(u)$	$b_3(u)$	$b_4(u)$
$b_1(v)$	$b_1(v)b_1(u)$	$b_1(v)b_2(u)$	$b_1(v)b_3(u)$	$b_1(v)b_4(u)$
$b_2(v)$	$b_2(v)b_1(u)$	$b_2(v)b_2(u)$	$b_2(v)b_3(u)$	$b_2(v)b_4(u)$
$b_3(v)$	$b_3(v)b_1(u)$	$b_3(v)b_2(u)$	$b_3(v)b_3(u)$	$b_3(v)b_4(u)$
$b_4(v)$	$b_4(v)b_1(u)$	$b_4(v)b_2(u)$	$b_4(v)b_3(u)$	$b_4(v)b_4(u)$

Monomial Example

Tensor product basis of cubic monomials

	1	u	u^2	u^3
1	1	u	u^2	u^3
v	v	vu	vu^2	vu^3
v^2	v^2	v^2u	v^2u^2	v^2u^3
v^3	v^3	v^3u	v^3u^2	v^3u^3

Example

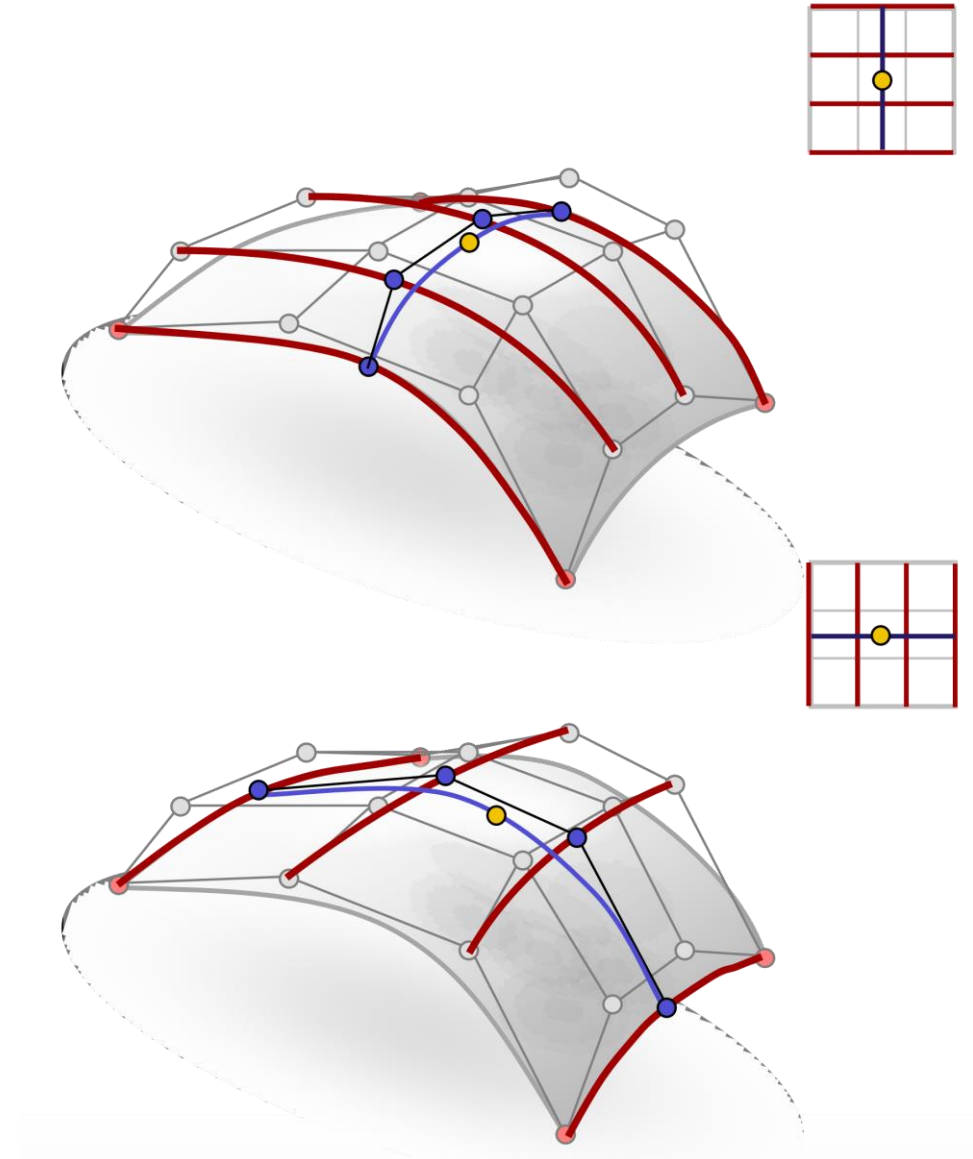


Tensor Product Surfaces

Tensor Product Surfaces

$$\begin{aligned} f(u, v) &= \sum_{i=1}^n \sum_{j=1}^n b_i(u) b_j(v) \mathbf{p}_{i,j} \\ &= \sum_{i=1}^n b_i(u) \sum_{j=1}^n b_j(v) \mathbf{p}_{i,j} \\ &= \sum_{j=1}^n b_j(v) \sum_{i=1}^n b_i(u) \mathbf{p}_{i,j} \end{aligned}$$

- “Curves of Curves”
- Order does not matter



Properties

Properties of tensor product surfaces:

- Linear invariance: Obvious
- Affine invariance?
 - Needs partition of unity property
 - Assume basis $B^{(curv)} := \{b_1(t), \dots, b_n(t)\}$ forms a partition of unity, i.e.: $\sum_{i=1}^n b_i(v) = 1$
 - Then we get:

$$\sum_{i=1}^n \sum_{j=1}^n b_i(u) b_j(v) = \sum_{i=1}^n b_i(u) \sum_{j=1}^n b_j(v) = \sum_{j=1}^n b_j(v) \cdot 1 = 1$$

- Affine invariance for tensor product surfaces is induced by the corresponding property of the employed curve basis

Properties

Properties of tensor product surfaces:

- Convex Hull?
 - Assume basis $\mathbf{B}^{(curv)} := \{b_1(t), \dots, b_n(t)\}$ forms a partition of unity and it is nonnegative (≥ 0) on $t \in [t_0, t_1]$
 - Obviously, we then have:

$$\sum_{i=1}^n \sum_{j=1}^n \underbrace{b_i(u)}_{\geq 0} \underbrace{b_j(v)}_{\geq 0} \geq 0$$

- So we have the convex hull property on $[t_0, t_1]^2$
- The convex hull property for tensor product surface is induced by the property of the employed curve basis

Partial Derivatives

Computing partial derivatives:

- First derivatives:

$$\frac{\partial}{\partial u} \sum_{i=1}^n \sum_{j=1}^n b_i(u) b_j(v) \mathbf{p}_{i,j} = \sum_{j=1}^n b_j(v) \sum_{i=1}^n \left(\frac{d}{du} b_i \right) (u) \mathbf{p}_{i,j}$$

$$\frac{\partial}{\partial v} \sum_{i=1}^n \sum_{j=1}^n b_i(u) b_j(v) \mathbf{p}_{i,j} = \sum_{i=1}^n b_i(u) \sum_{j=1}^n \left(\frac{d}{dv} b_j \right) (v) \mathbf{p}_{i,j}$$

- Just spline-curve combinations of curve derivatives

Partial Derivatives

Computing partial derivatives:

- Second derivatives:

$$\frac{\partial}{\partial u^2} \sum_{i=1}^n \sum_{j=1}^n b_i(u) b_j(v) \mathbf{p}_{i,j} = \sum_{j=1}^n b_j(v) \sum_{i=1}^n \left(\frac{d^2}{du^2} b_i \right) (u) \mathbf{p}_{i,j}$$

$$\begin{aligned} \frac{\partial^2}{\partial u \partial v} \sum_{i=1}^n \sum_{j=1}^n b_i(u) b_j(v) \mathbf{p}_{i,j} &= \frac{\partial}{\partial v} \sum_{j=1}^n b_j(v) \sum_{i=1}^n \left(\frac{d}{du} b_i \right) (u) \mathbf{p}_{i,j} \\ &= \sum_{j=1}^n \left(\frac{d}{dv} b_j \right) (v) \sum_{i=1}^n \left(\frac{d}{du} b_i \right) (u) \mathbf{p}_{i,j} \end{aligned}$$

Partial Derivatives

Computing partial derivatives:

- General derivatives:

$$\begin{aligned}\frac{\partial^{r+s}}{\partial u^r \partial v^s} \sum_{i=1}^n \sum_{j=1}^n b_i(u) b_j(v) \mathbf{p}_{i,j} &= \sum_{j=1}^n \left(\frac{d^s}{dv^s} b_j \right) (v) \sum_{i=1}^n \left(\frac{d^r}{du^r} b_i \right) (u) \mathbf{p}_{i,j} \\ &= \sum_{i=1}^n \left(\frac{d^r}{du^r} b_i \right) (u) \sum_{j=1}^n \left(\frac{d^s}{dv^s} b_j \right) (v) \mathbf{p}_{i,j}\end{aligned}$$

Normal Vectors

We can compute normal vectors from partial derivatives:

$$\mathbf{n}(u, v) = \frac{\left(\sum_{j=1}^n b_j(v) \sum_{i=1}^n \frac{d}{du} b_i(u) \mathbf{p}_{i,j} \right) \times \left(\sum_{j=1}^n \frac{d}{dv} b_j(v) \sum_{i=1}^n b_i(u) \mathbf{p}_{i,j} \right)}{\left\| \left(\sum_{j=1}^n b_j(v) \sum_{i=1}^n \frac{d}{du} b_i(u) \mathbf{p}_{i,j} \right) \times \left(\sum_{j=1}^n \frac{d}{dv} b_j(v) \sum_{i=1}^n b_i(u) \mathbf{p}_{i,j} \right) \right\|}$$

- Problem: degenerate cases
 - Collinear tangents
 - Irregular parametrization
- Need extra code to handle special cases

Tensor Product Surfaces

Tensor Product Bézier Surfaces

Tensor Product Bézier Spline Surfaces

Tensor Product Bézier Surfaces

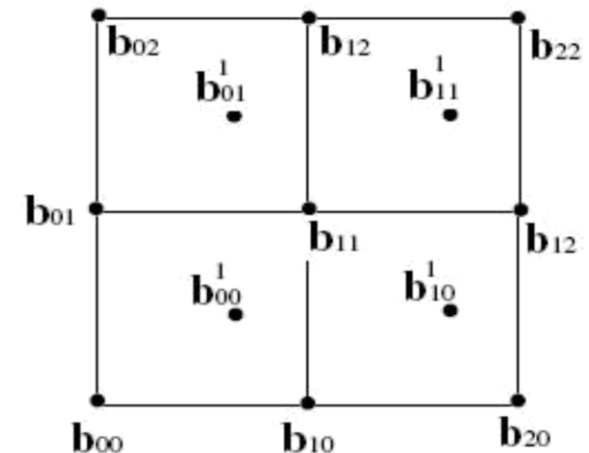
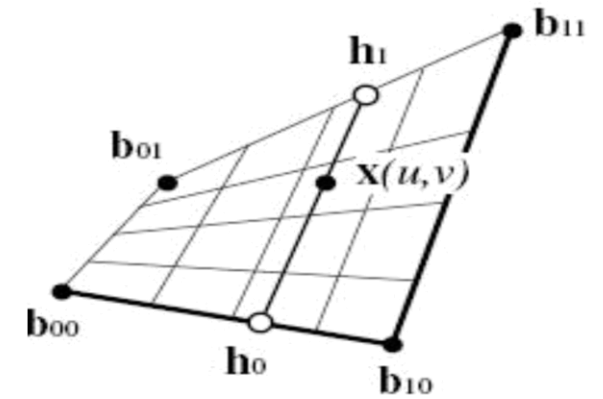
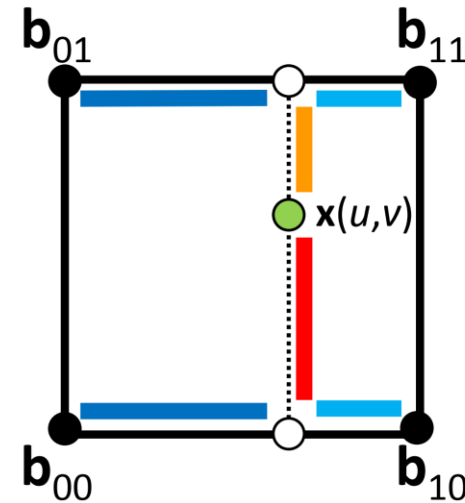
Bézier curves:
repeated linear interpolation

now a different setup:

4 points \mathbf{b}_{00} , \mathbf{b}_{10} , \mathbf{b}_{11} , \mathbf{b}_{01}
Parameter area $[0,1] \times [0,1]$

→ bilinear interpolation:
repeated linear interpolation

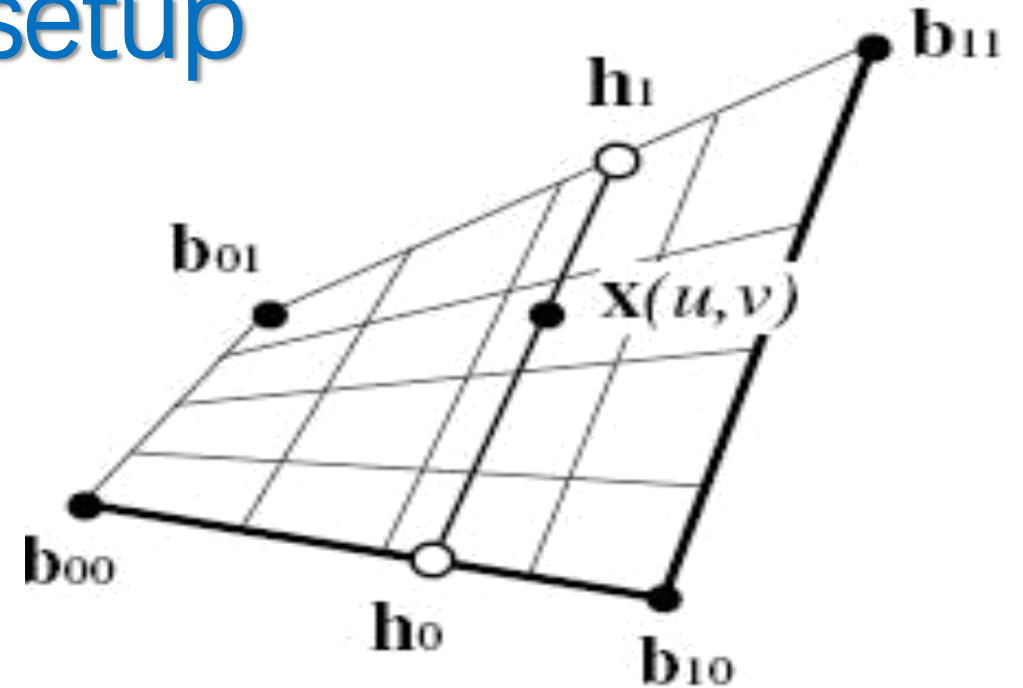
→ repeated bilinear interpolation:
Gives us tensor product Bézier surfaces
(example) shows quadratic Bézier Surfaces)



Some formulas for this setup

$$\mathbf{h}_0 = (1 - u)\mathbf{b}_{00} + u\mathbf{b}_{10}$$

$$\mathbf{h}_1 = (1 - u)\mathbf{b}_{01} + u\mathbf{b}_{11}$$



$$\mathbf{x}(u, v) = (1 - v)\mathbf{h}_0 + v\mathbf{h}_1$$

$$= (1 - v)[(1 - u)\mathbf{b}_{00} + u\mathbf{b}_{10}] + v[(1 - u)\mathbf{b}_{01} + u\mathbf{b}_{11}]$$

$$\mathbf{x}(u, v) = (1 - u)(1 - v)\mathbf{b}_{00} + u(1 - v)\mathbf{b}_{10} + (1 - u)v\mathbf{b}_{01} + uv\mathbf{b}_{11}$$

Some formulas for this setup

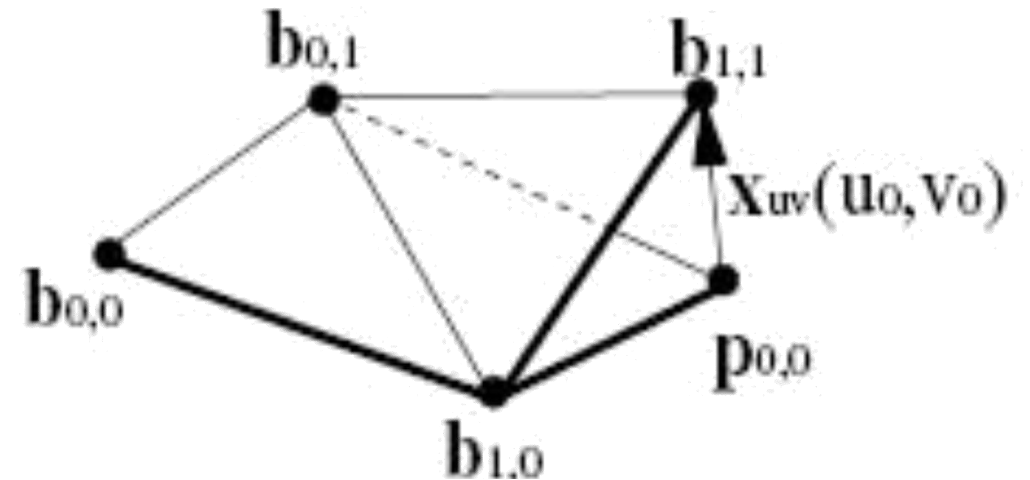
Derivatives of bilinear surfaces

$$\mathbf{x}_u(u, v) = (1 - v)(\mathbf{b}_{10} - \mathbf{b}_{00}) + v(\mathbf{b}_{11} - \mathbf{b}_{01})$$

$$\mathbf{x}_v(u, v) = (1 - u)(\mathbf{b}_{01} - \mathbf{b}_{00}) + u(\mathbf{b}_{11} - \mathbf{b}_{10})$$

$$\mathbf{x}_{uu}(u, v) = \mathbf{x}_{vv}(u, v) = 0$$

$$\mathbf{x}_{uv}(u, v) = \mathbf{b}_{00} - \mathbf{b}_{10} - \mathbf{b}_{01} + \mathbf{b}_{11}$$



Some formulas for this setup

Biquadratic surfaces

$$\mathbf{b}_{00}^1 = (1-u)(1-v)\mathbf{b}_{00} + u(1-v)\mathbf{b}_{10} + (1-u)v\mathbf{b}_{01} + uv\mathbf{b}_{11}$$

$$\mathbf{b}_{10}^1 = (1-u)(1-v)\mathbf{b}_{10} + u(1-v)\mathbf{b}_{20} + (1-u)v\mathbf{b}_{11} + uv\mathbf{b}_{21}$$

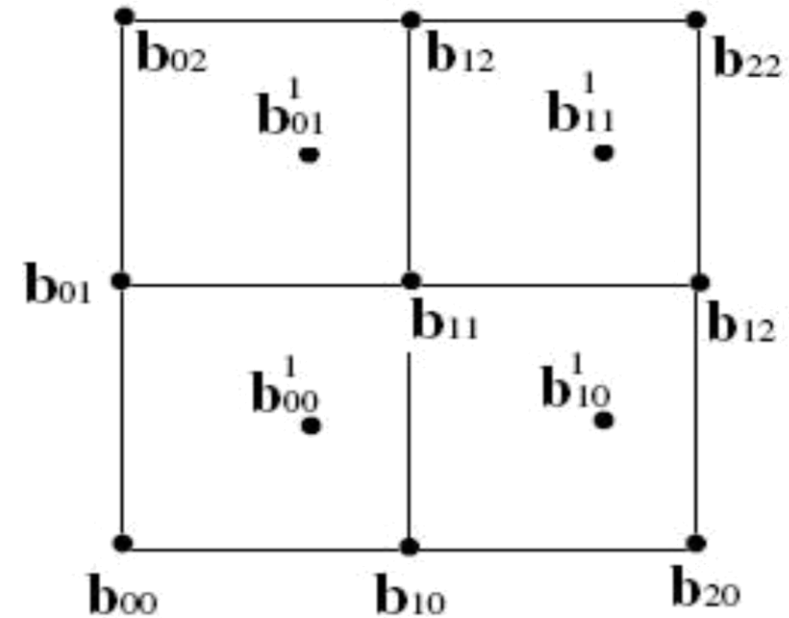
$$\mathbf{b}_{01}^1 = (1-u)(1-v)\mathbf{b}_{01} + u(1-v)\mathbf{b}_{11} + (1-u)v\mathbf{b}_{02} + uv\mathbf{b}_{12}$$

$$\mathbf{b}_{11}^1 = (1-u)(1-v)\mathbf{b}_{11} + u(1-v)\mathbf{b}_{21} + (1-u)v\mathbf{b}_{12} + uv\mathbf{b}_{22}$$

$$\mathbf{x}(u, v)$$

$$= (1-u)(1-v)\mathbf{b}_{00}^1 + u(1-v)\mathbf{b}_{10}^1 + (1-u)v\mathbf{b}_{01}^1 + uv\mathbf{b}_{11}^1$$

$$= \sum_{i=0}^2 \sum_{j=0}^2 B_i^2(u) B_j^2(v) \mathbf{b}_{i,j}$$



Bézier Patches

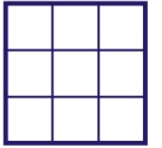
Bézier Patches:

- Use tensor product Bernstein basis

$$f(u, v) = \sum_{i=0}^d \sum_{j=0}^d B_i^{(d)}(u) B_j^{(d)}(v) \mathbf{p}_{i,j}$$

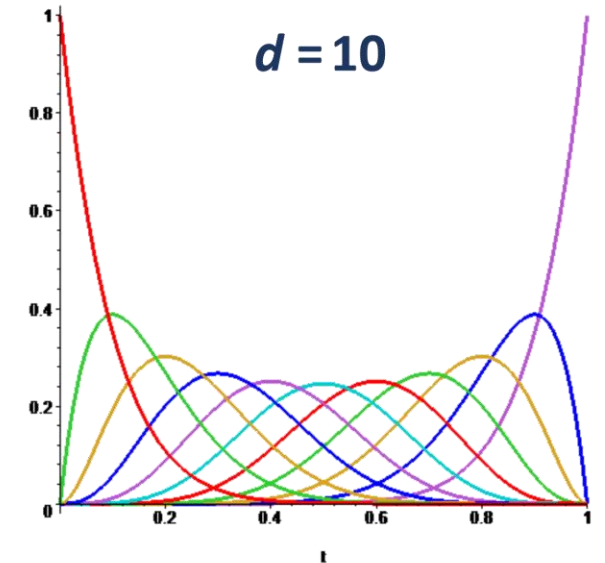
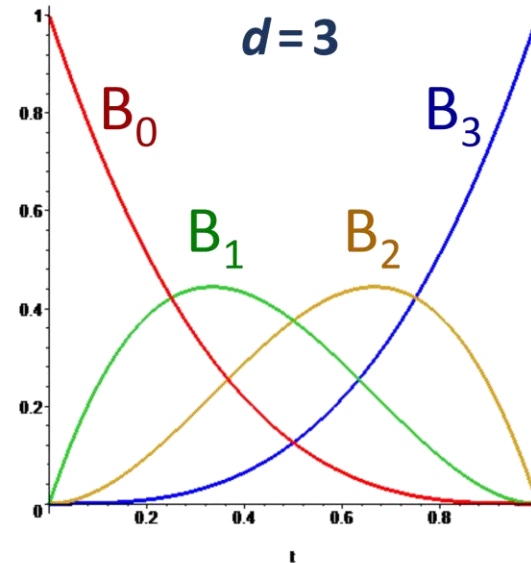
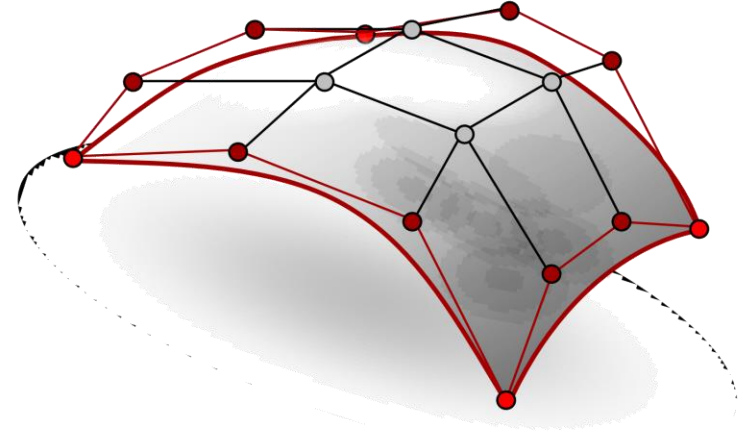
- We get automatically:
 - Affine invariance
 - Convex hull property

Bézier Patches



Bézier Patches:

- Remember endpoint interpolation:
 - Boundary curves are Bézier curves of the boundary control points



Bézier Patches

Bézier Patches:

- Tangent vectors:
 - First derivatives at boundary points are proportional to differences of control points:

$$\begin{aligned}\frac{\partial}{\partial u} f(u, v) \Big|_{u=0} &= \sum_{i=0}^d \sum_{j=0}^d B_i^{(d)}(v) B_j'^{(d)}(0) \mathbf{p}_{i,j} \\ &= d \sum_{j=0}^d B_j^{(d)}(v) (\mathbf{p}_{1,j} - \mathbf{p}_{0,j})\end{aligned}$$

$$\frac{\partial}{\partial u} f(u, v) \Big|_{u=1} = d \sum_{j=0}^d B_j^{(d)}(v) (\mathbf{p}_{d,j} - \mathbf{p}_{d-1,j})$$

Continuity Conditions

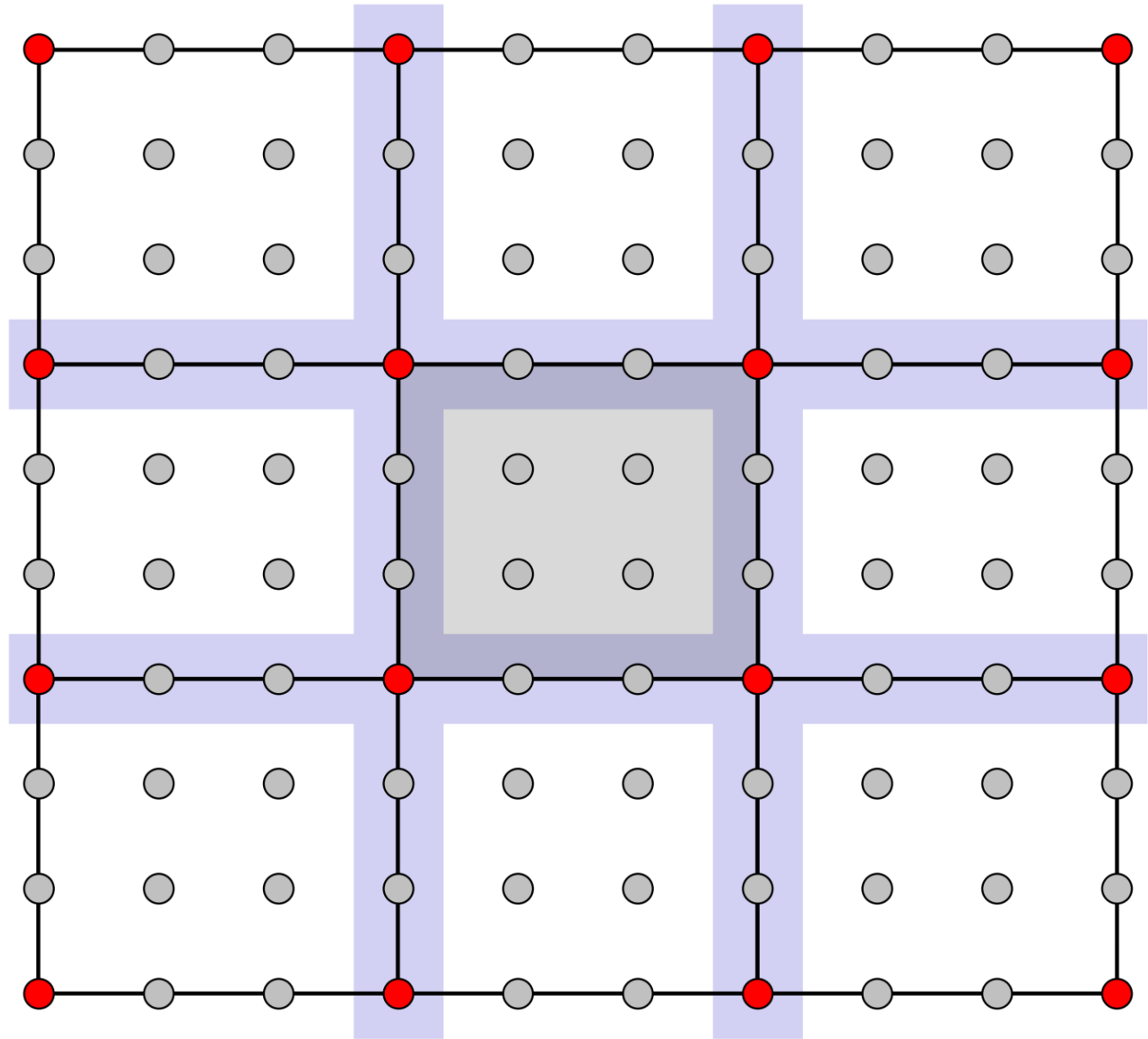
For C^0 continuity:

- Boundary control points must match

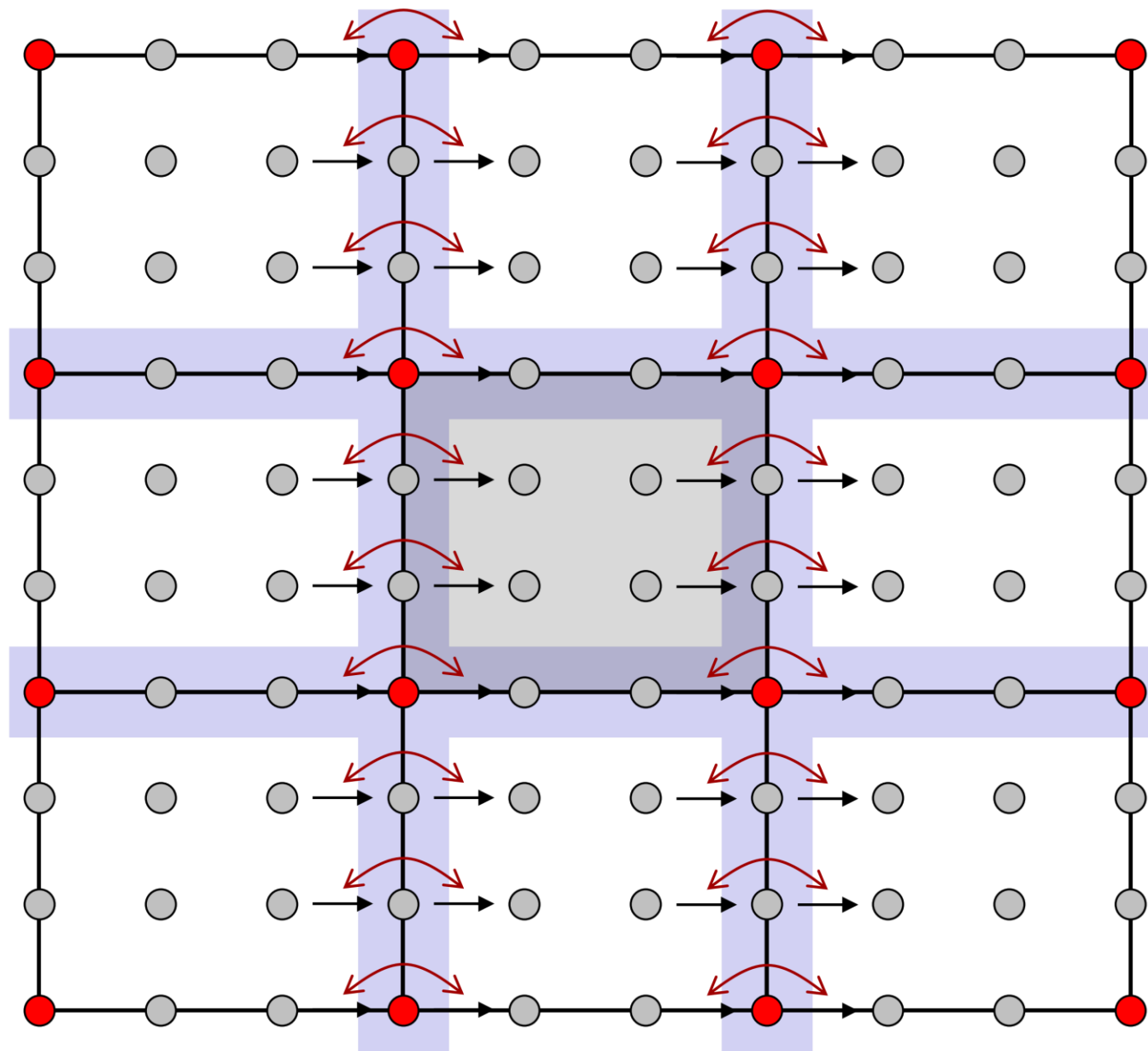
For C^1 continuity:

- Difference vectors must match at the boundary

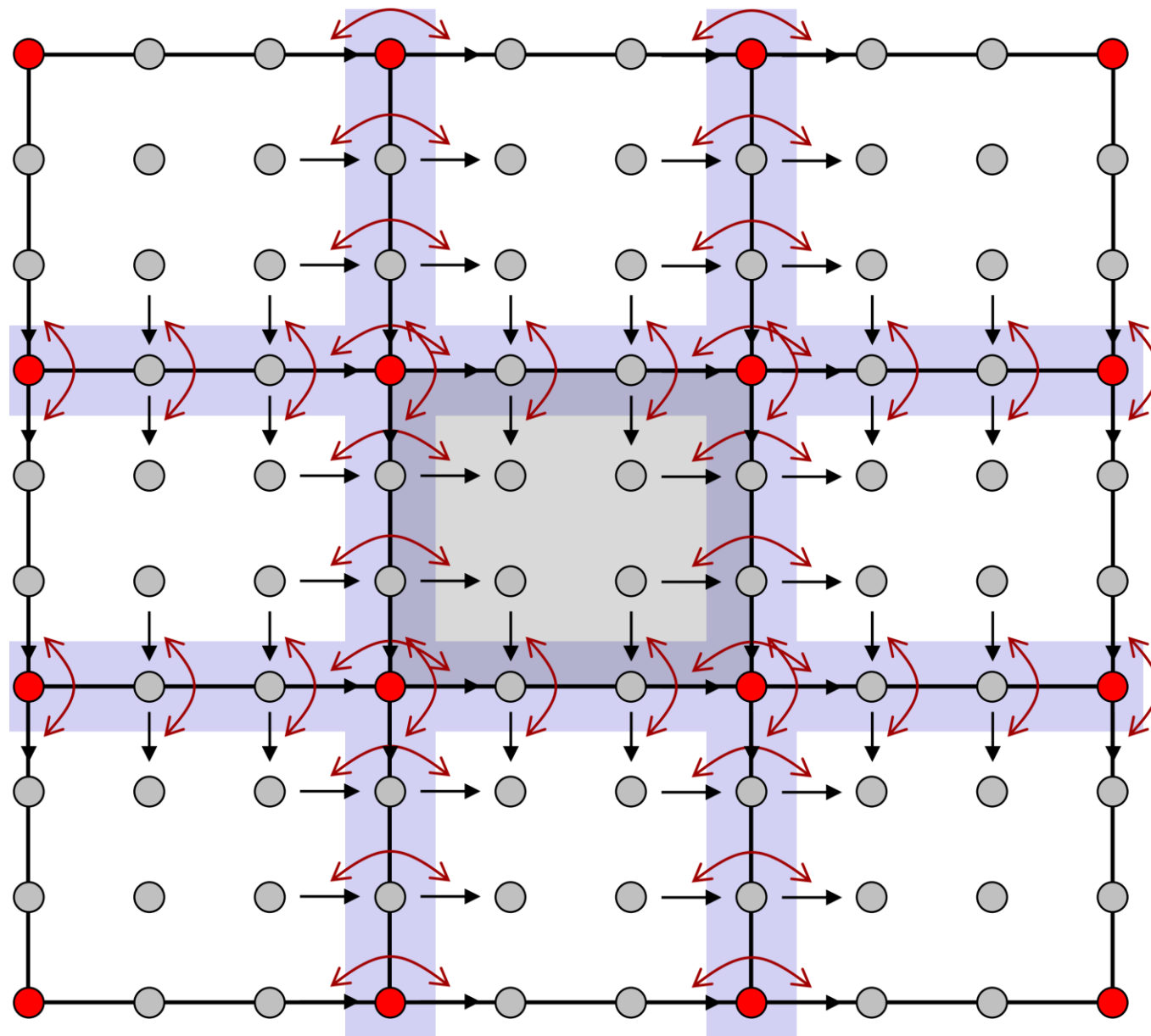
C^0 Continuity



C^1 Continuity



C^1 Continuity



Polars & Blossoms

Blossoms for tensor product surfaces:

- Polar form of a polynomial tensor product surfaces of degree d :

$$F: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}^n \quad F(u, v)$$

$$\mathbf{f}: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^n \quad \mathbf{f}(u_1, \dots, u_d; v_1, \dots, v_d)$$

- Required properties:

- **Diagonality:** $\mathbf{f}(u, \dots, u; v, \dots, v) = F(u, v)$

- **Symmetry:** $\mathbf{f}(u_1, \dots, u_d; v_1, \dots, v_d) = \mathbf{f}(u_{\pi(1)}, \dots, u_{\pi(d)}; v_{\mu(1)}, \dots, v_{\mu(d)})$
for all permutations of indices π, μ

- **Multi-affine:** $\sum \alpha_k = 1$

$$\Rightarrow \mathbf{f}\left(u_1, \dots, \sum \alpha_k u_i^{(k)}, \dots, u_d; v_1, \dots, v_d\right)$$

$$= \alpha_1 \mathbf{f}\left(u_1, \dots, u_i^{(1)}, \dots, u_d; v_1, \dots, v_d\right) + \dots + \alpha_n \mathbf{f}\left(u_1, \dots, u_i^{(n)}, \dots, u_d; v_1, \dots, v_d\right)$$

$$\text{and } \mathbf{f}\left(u_1, \dots, u_d; v_1, \dots, \sum \alpha_k v_i^{(k)}, \dots, v_d\right)$$

$$= \alpha_1 \mathbf{f}\left(u_1, \dots, u_d; v_1, \dots, v_i^{(1)}, \dots, v_d\right) + \dots + \alpha_n \mathbf{f}\left(u_1, \dots, u_d; v_1, \dots, v_i^{(n)}, \dots, v_d\right)$$

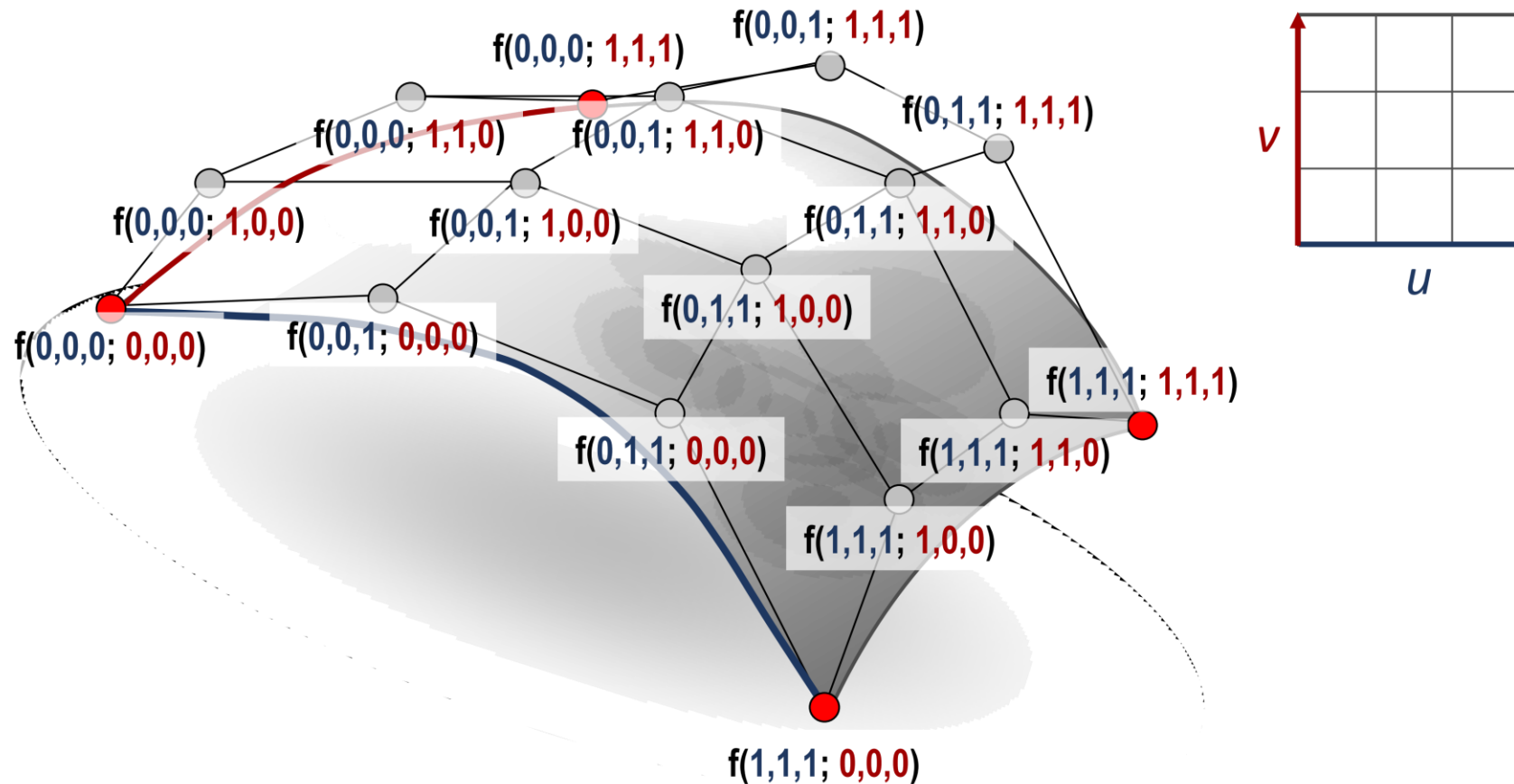
Short Summary

Polar forms for tensor product surfaces:

- Polar separately in u and v
- Notation: $f(\underbrace{u_1, \dots, u_d}_{u\text{-parameters}}; \underbrace{v_1, \dots, v_d}_{v\text{-parameters}})$
- Can be used to derive properties/algorithms similar to the curve case

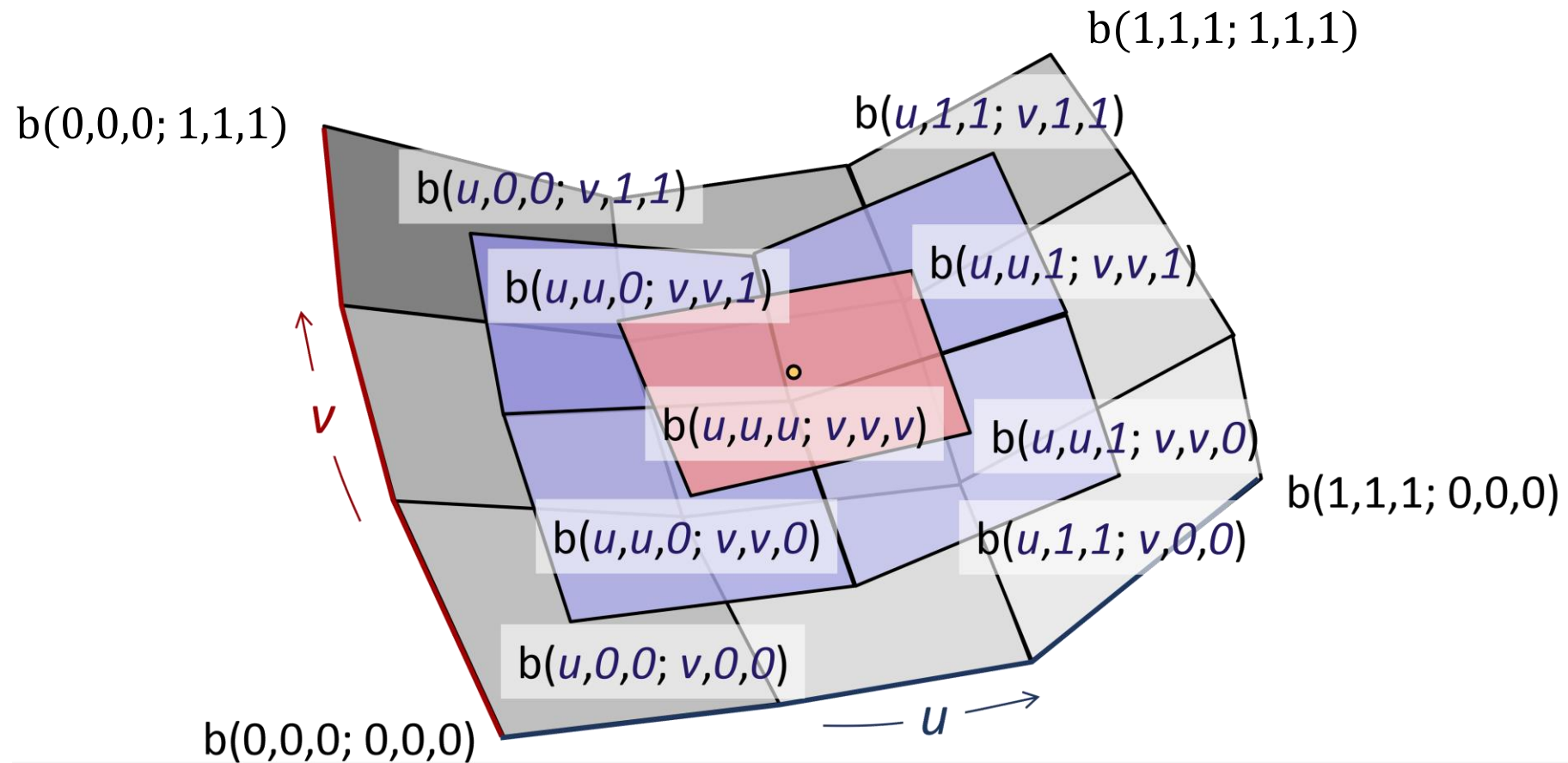
Bézier Control Points

Bézier control points in blossom notation:



de Casteljau Algorithm

de Casteljau algorithm for tensor product surfaces



Tensor Product Surfaces

Tensor Product B-Spline Surfaces

B-Spline Patches

B-Spline Patches

- More general than Bézier patches
(we get Bézier patches as a special case)
- First, we fix a degree d
- Then, we need knot sequences in u and v direction:
 $(u_1, \dots, u_n), (v_1, \dots, v_m)$
- And a corresponding array of control points

$$\begin{array}{ccccc} d_{0,0} & & \dots & & d_{n-d+1,0} \\ & & \dots & & \dots \\ d_{0,m-d+1} & & \dots & & d_{n-d+1,m-d+1} \end{array}$$

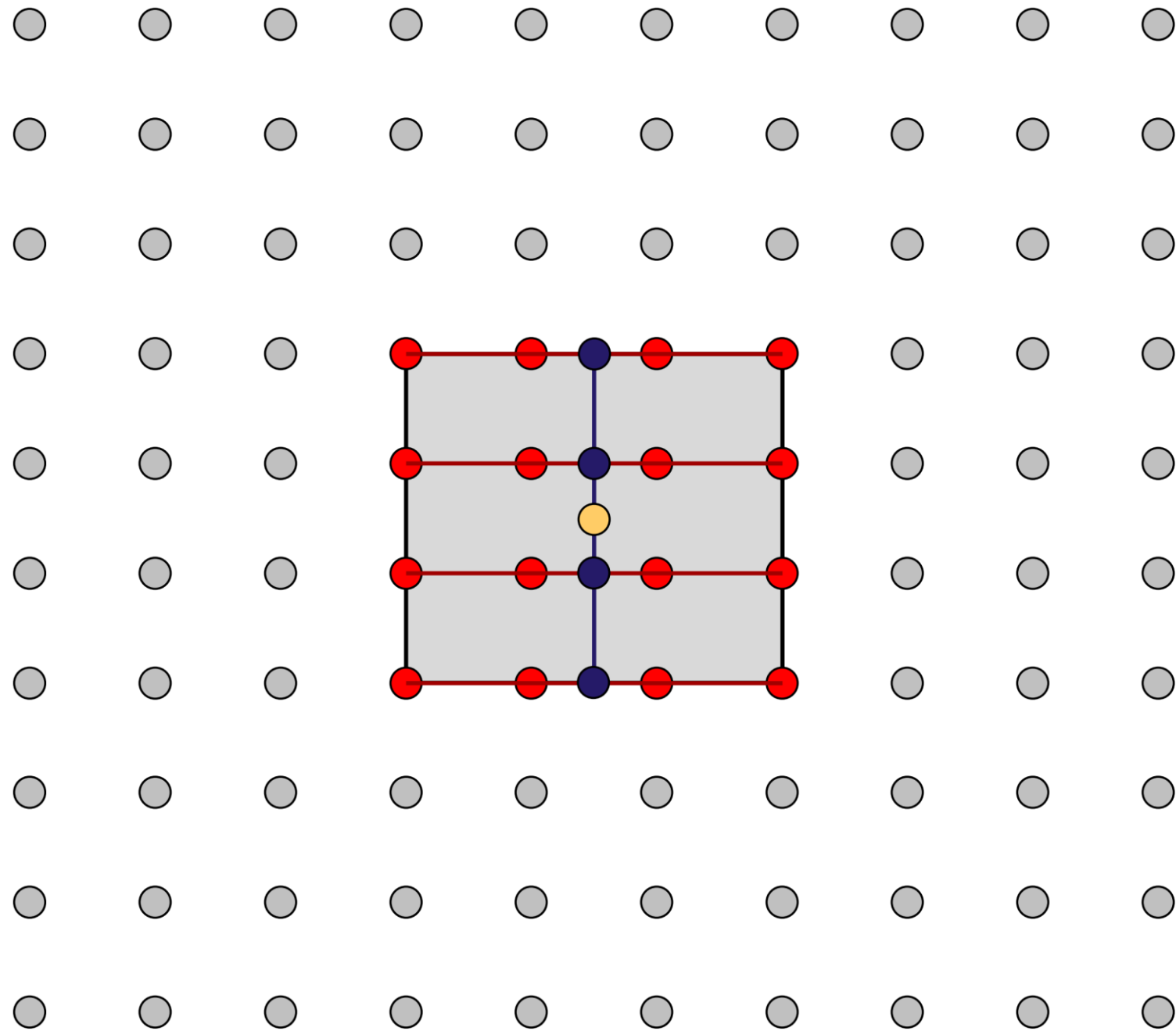
B-Spline Patches

Then, obtain a parametric B-spline patch as:

$$f(u, v) = \sum_{i=0}^d \sum_{j=0}^d N_i^{(d)}(u) N_j^{(d)}(v) \mathbf{p}_{i,j}$$

- We can evaluate the patches using the de Boor Algorithm:
 - “Curves of curves” idea
 - Determine the knots/control points influencing (u, v) ,
These will be no more than $(d + 1) \times (d + 1)$ points
 - Compute $(d + 1)$ v -direction control points along u -direction,
Performing $(d + 1)$ curve evaluations
 - Then evaluate the curve in v -direction
 - (or the other way around, interchanging u, v -directions)

Illustration



B-Spline Patches

Alternative:

- 2D de Boor algorithm
- Works similar to the 2D de Casteljau algorithm but with different weights (we can use tensor-product blossoming to derive the weights)

Tensor Product Surfaces

Rational Patches

Rational Patches

Rational Patches

- We can use rational Bézier/B-splines to create the patches (“rational Bézier patches” / “NURBS-patches”)
- Idea:
 - Form a parametric surface in 4D, homogenous space
 - Then project to $\omega = 1$ to obtain the surface in Euclidian 3D space
- In short: Just use homogeneous coordinates everywhere

Rational Patch

Rational Bézier Patch:

$$\mathbf{f}^{(hom)}(u, v) = \sum_{i=0}^d \sum_{j=0}^d B_i^{(d)}(u) B_j^{(d)}(v) \begin{pmatrix} \omega_{i,j} \mathbf{p}_{i,j} \\ \omega_{i,j} \end{pmatrix}$$

$$\mathbf{f}^{(Eucl)}(u, v) = \frac{\sum_{i=0}^d \sum_{j=0}^d B_i^{(d)}(u) B_j^{(d)}(v) \omega_{i,j} \mathbf{p}_{i,j}}{\sum_{i=0}^d \sum_{j=0}^d B_i^{(d)}(u) B_j^{(d)}(v) \omega_{i,j}}$$

Rational Patch

Rational B-Spline Patch:

$$\mathbf{f}^{(hom)}(u, v) = \sum_{i=0}^d \sum_{j=0}^d N_i^{(d)}(u) N_j^{(d)}(v) \begin{pmatrix} \omega_{i,j} \mathbf{p}_{i,j} \\ \omega_{i,j} \end{pmatrix}$$

$$\mathbf{f}^{(Eucl)}(u, v) = \frac{\sum_{i=0}^d \sum_{j=0}^d N_i^{(d)}(u) N_j^{(d)}(v) \omega_{i,j} \mathbf{p}_{i,j}}{\sum_{i=0}^d \sum_{j=0}^d N_i^{(d)}(u) N_j^{(d)}(v) \omega_{i,j}}$$

Remark: Rational Patches

Observation:

- Euclidian surface is not a tensor product surface
 - Denominator depends on both u and v
- Homogeneous space: 4D surface is a tensor product surface.

$$\mathbf{f}^{(Eucl)}(u, v) = \frac{\sum_{i=0}^d \sum_{j=0}^d B_i^{(d)}(u) B_j^{(d)}(v) \omega_{i,j} \mathbf{p}_{i,j}}{\sum_{i=0}^d \sum_{j=0}^d B_i^{(d)}(u) B_j^{(d)}(v) \omega_{i,j}}$$

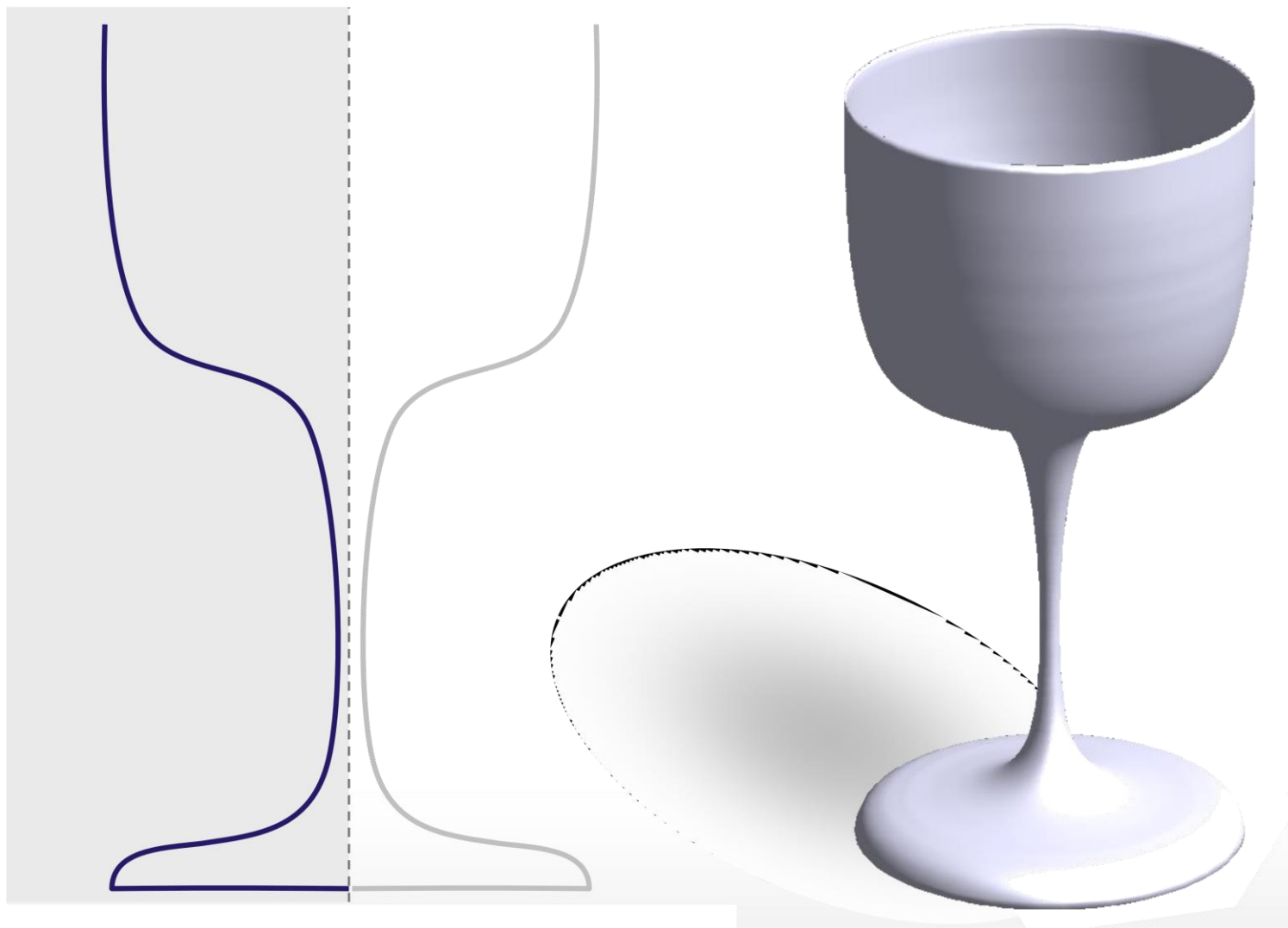
$$\mathbf{f}^{(Eucl)}(u, v) = \frac{\sum_{i=0}^d \sum_{j=0}^d N_i^{(d)}(u) N_j^{(d)}(v) \omega_{i,j} \mathbf{p}_{i,j}}{\sum_{i=0}^d \sum_{j=0}^d N_i^{(d)}(u) N_j^{(d)}(v) \omega_{i,j}}$$

Surfaces of Revolution

Advantages of rational patches:

- Rational patches can represent surfaces of revolution exactly.
- Examples:
 - Cylinders
 - Cones
 - Spheres
 - Ellipsoids
 - Tori
- Question: given a cross section curve, how do we get the control points for the 3D surface?

Surfaces of Revolution



Surfaces of Revolution

Given:

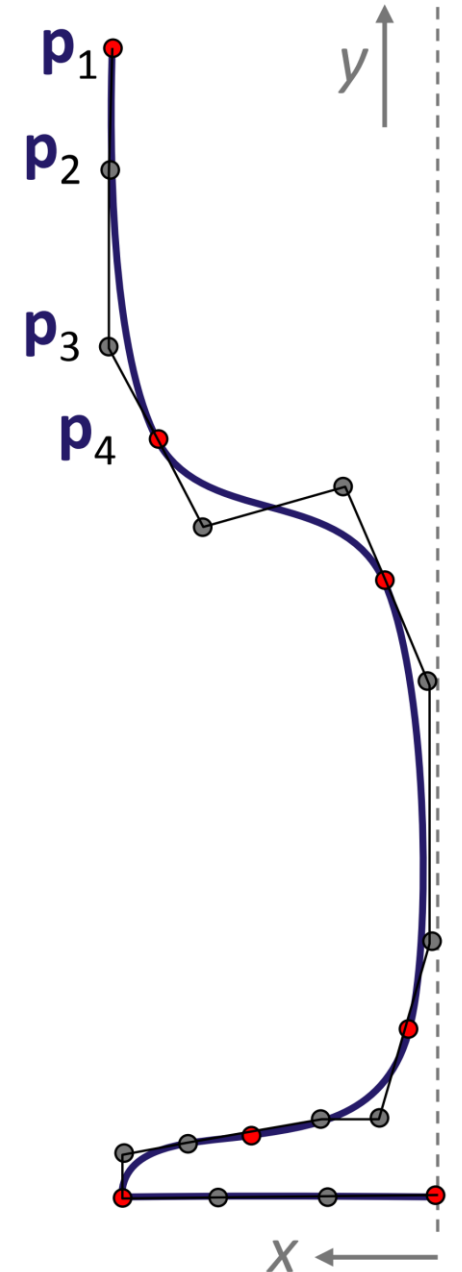
- Control points p_1, \dots, p_n of curve (“generatrix”)

We want to compute:

- Control points $p_{i,j}$ of a rational surface

Such that:

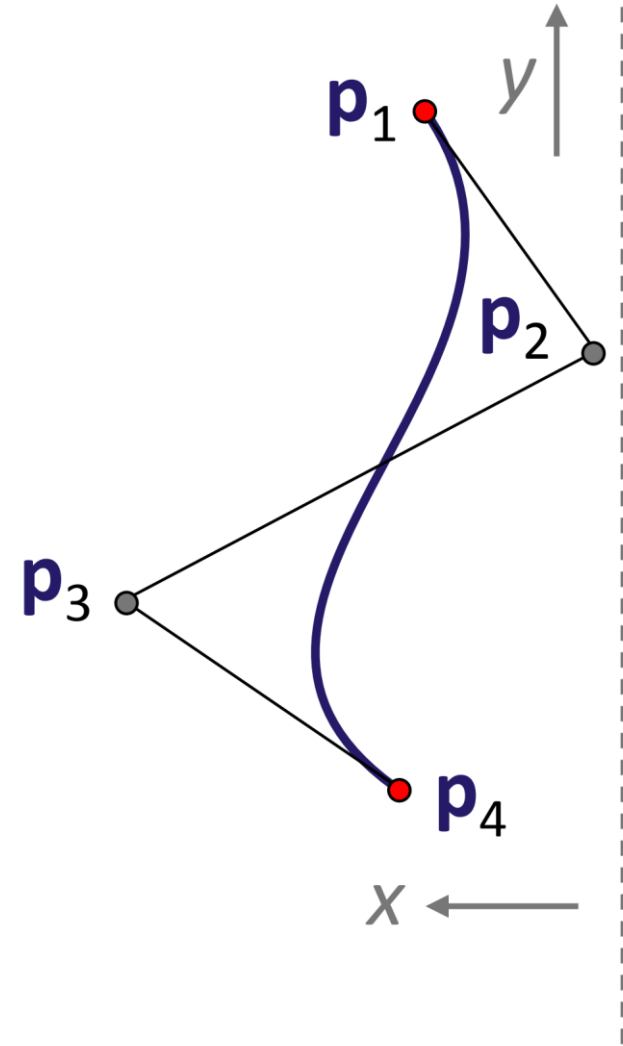
- The surface describes the surface of revolution that we obtain by rotating the curve around the y axis (w.l.o.g.)



Surfaces of Revolution

Simplification:

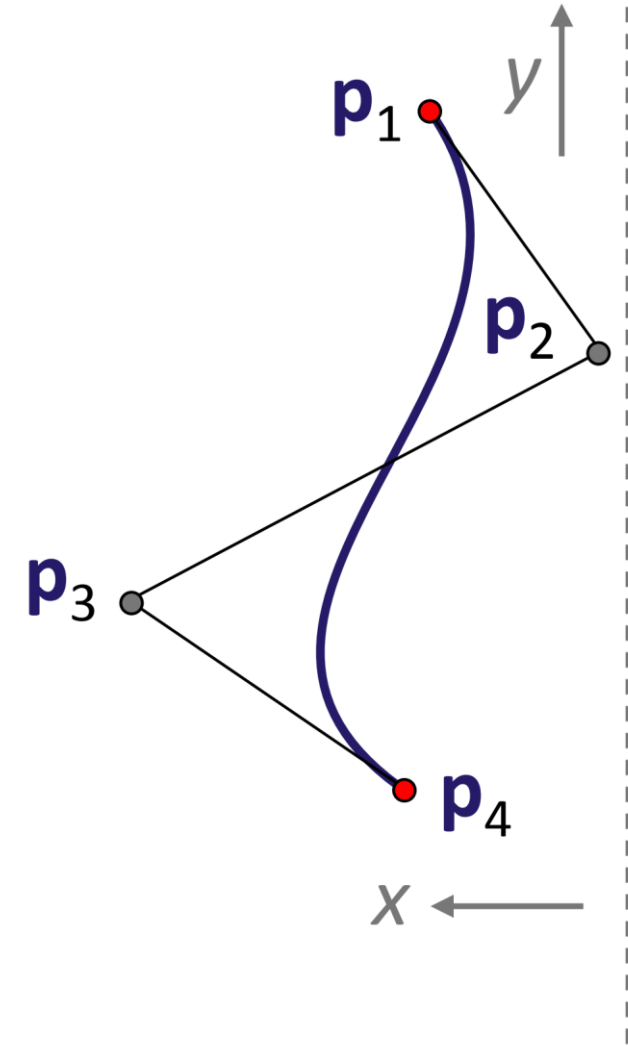
- We look only at a single rational Bézier segment
- Applying the scheme to multiple segments together is straightforward
- The same idea also works for B-splines



Surfaces of Revolution

Construction:

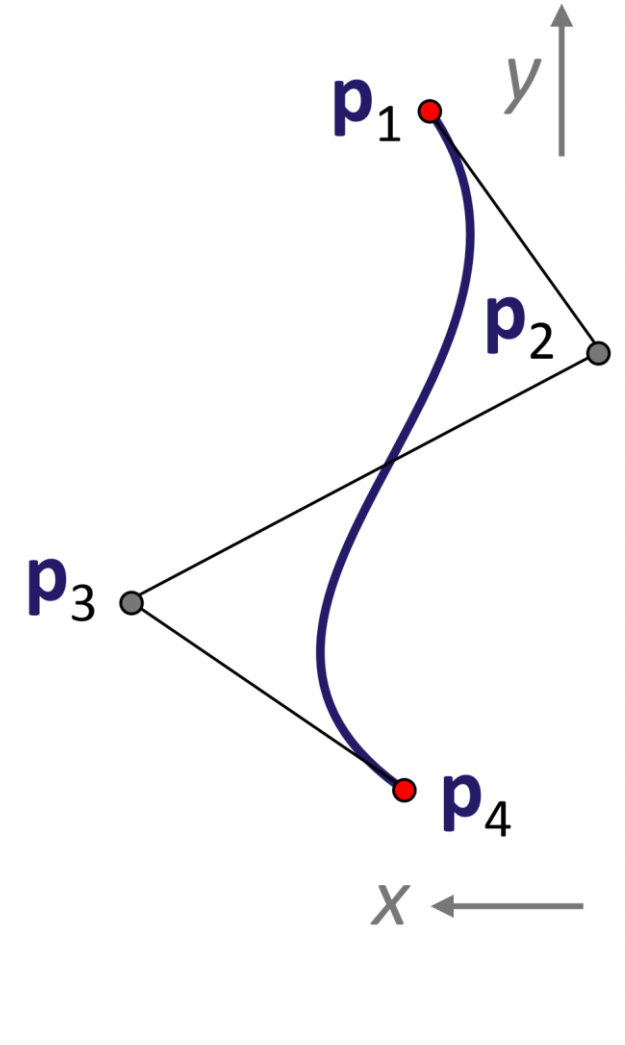
- We are given control points $\mathbf{p}_1, \dots, \mathbf{p}_{d+1}$
(d is the degree in u direction)
- We introduce a new parameter v
- In v direction, we use quadratic Bézier curves (2nd degree basis in v -direction)



Surfaces of Revolution

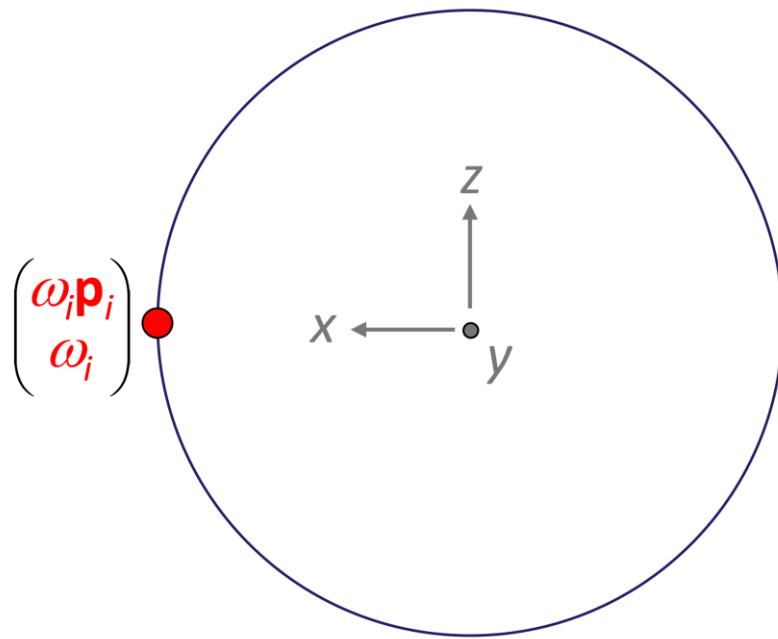
Key Idea:

- For u -direction curves: control points (and thus the curves) must move on circles around the y -axis
- Circles must have the same parametrization (this is easy)
- This means, the control points rotate around the y -axis
- Affine invariance will make the whole curve rotate, we get the desired surface of revolution



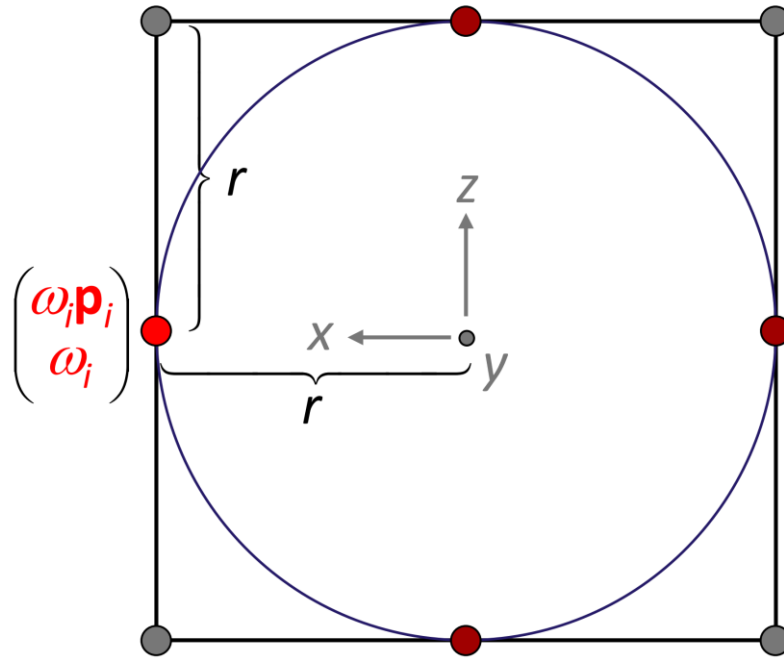
Surface of Revolution

Making one point rotate around the y-axis:



Surface of Revolution

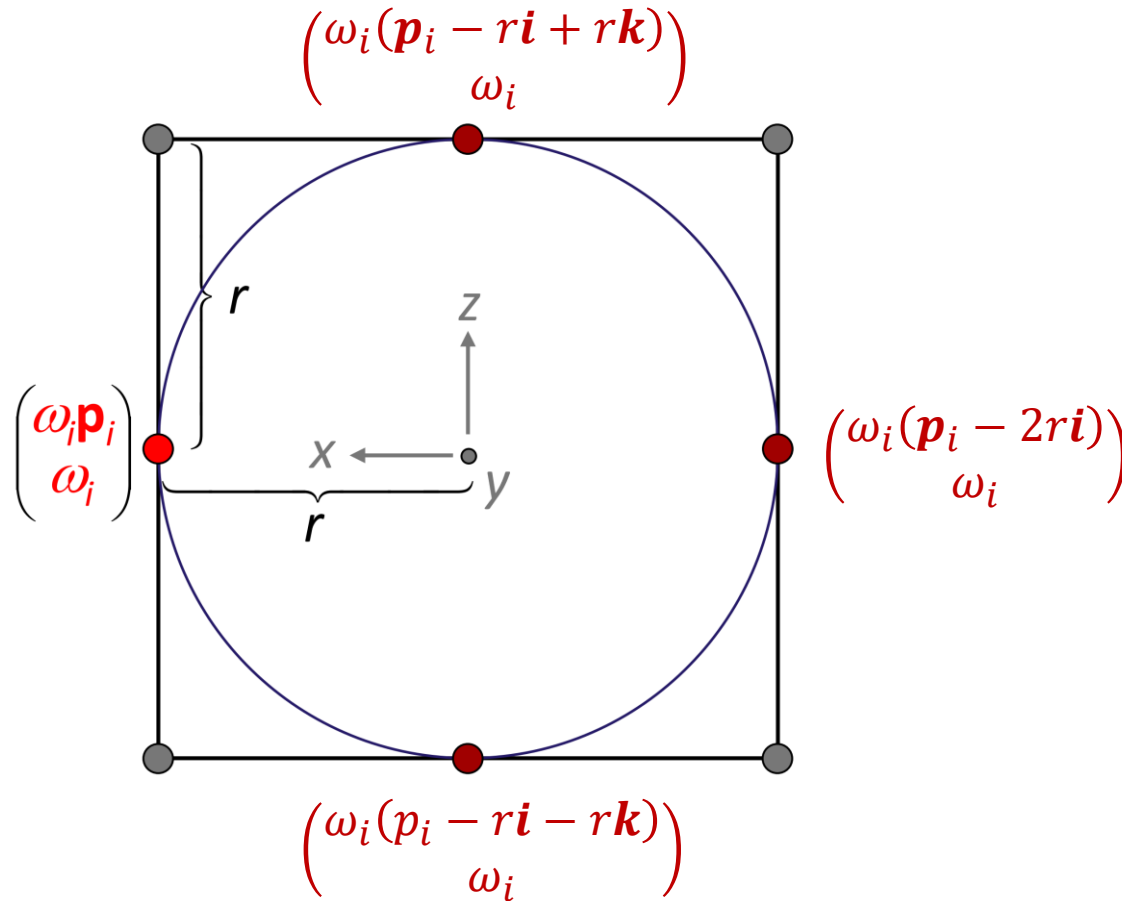
Making one point rotate around the y-axis:



Surface of Revolution

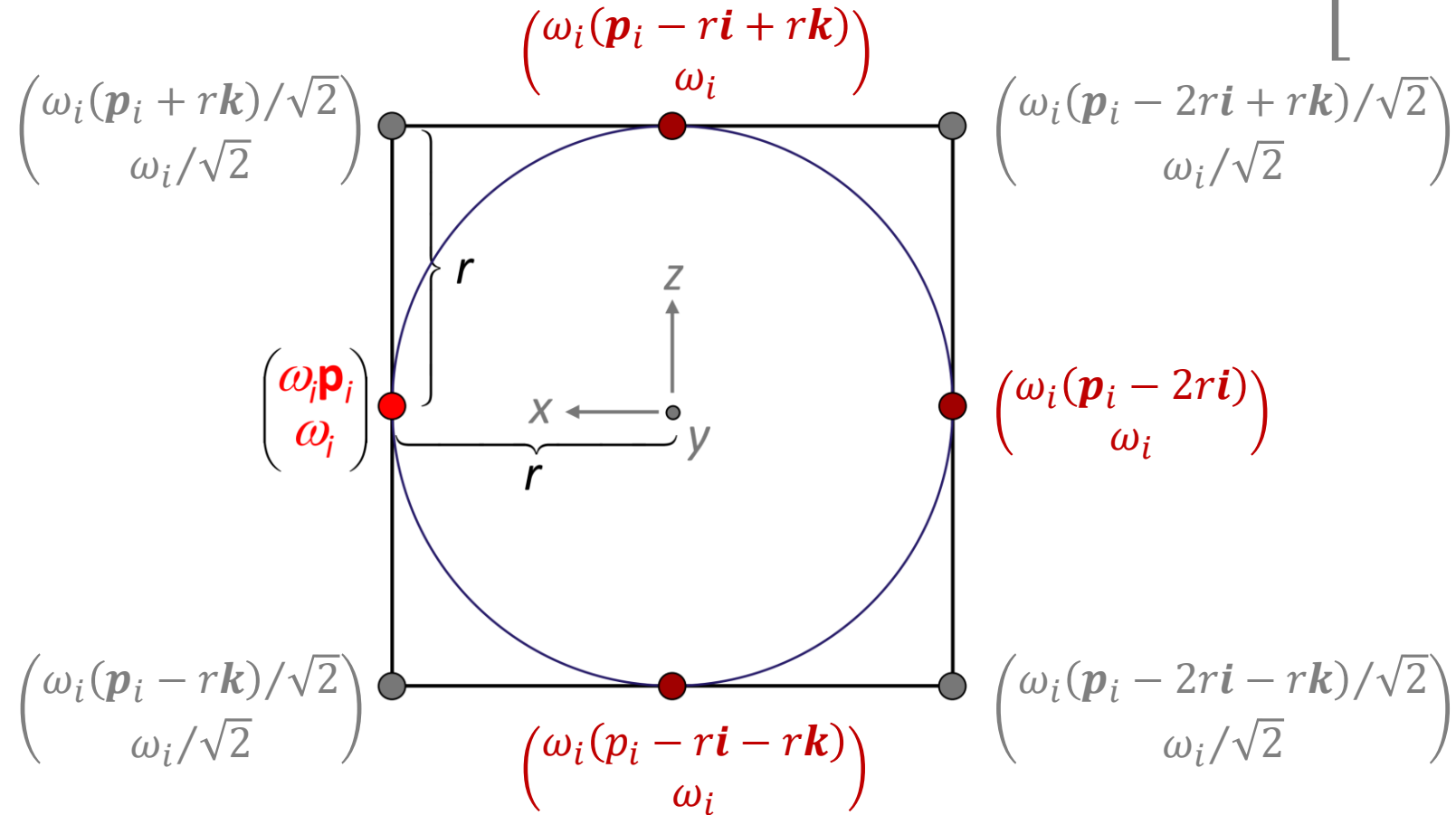
Making one point rotate around the y-axis:

$$\left[\mathbf{i} := \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \mathbf{k} := \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right]$$



Surface of Revolution

Making one point rotate around the y-axis: $\left[\mathbf{i} := \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \mathbf{k} := \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right]$



Remark

What we get:

- We obtain 4 segments, i.e. 4 patches for each Bézier segment
- A similar construction with 3 segments exists as well

Does the scheme yield a circle for weights $\neq 1$ in the generatrix curve?

- Common factors in weights cancel out
- Therefore, we still obtain a circle at these points
- Parametrization does not change either

Benefit

With this construction, it is straightforward to create:

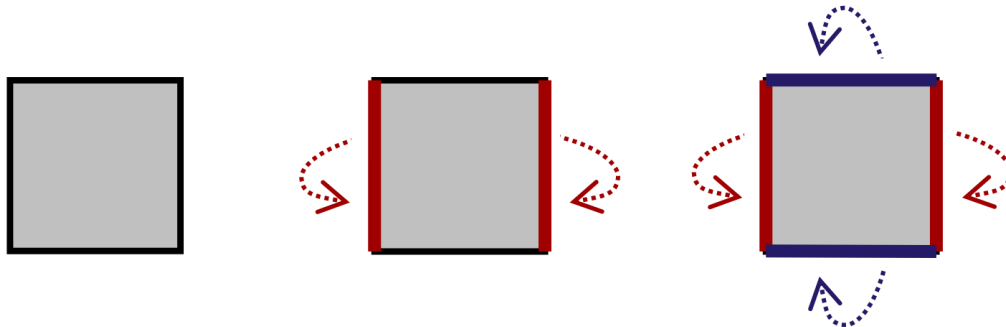
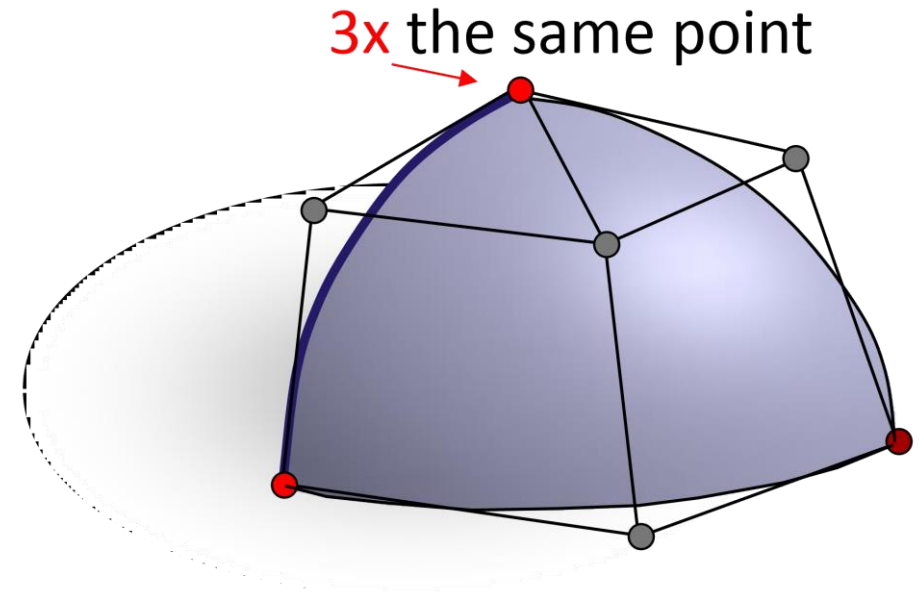
- Spheres
- Tori
- Cylinders
- Cones

And affine transformations of these (e.g. ellipsoids)

Parametrization Restrictions

Remaining problem:

- The sphere and the cone are not regularly parametrized (double control points)
- Might cause trouble (normal computation, tessellation)
- In general: no sphere, or n -tori ($n > 1$) can be parametrized without degeneracies
- What works: open surfaces, cylinders, tori



Curves on Surfaces, trimmed NURBS

Quad patch problem:

- All of our shapes are parameterized over rectangular regions
- General boundary curves are hard to create
- Topology fixed to a disc (or cylinder, torus)
- No holes in the middle
- Assembling complicated shapes is painful
 - Lots of pieces
 - Continuity conditions for assembling pieces become complicated
 - Cannot use C^2 B-splines continuity along boundaries when using multiple pieces

Curves on Surfaces, trimmed NURBS

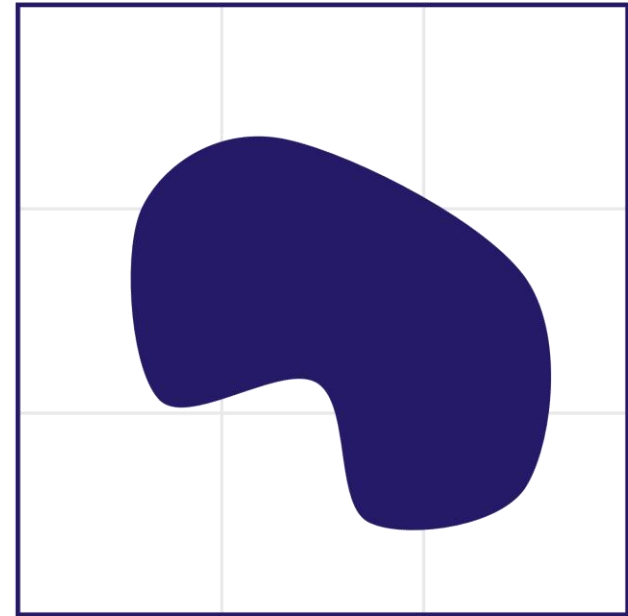
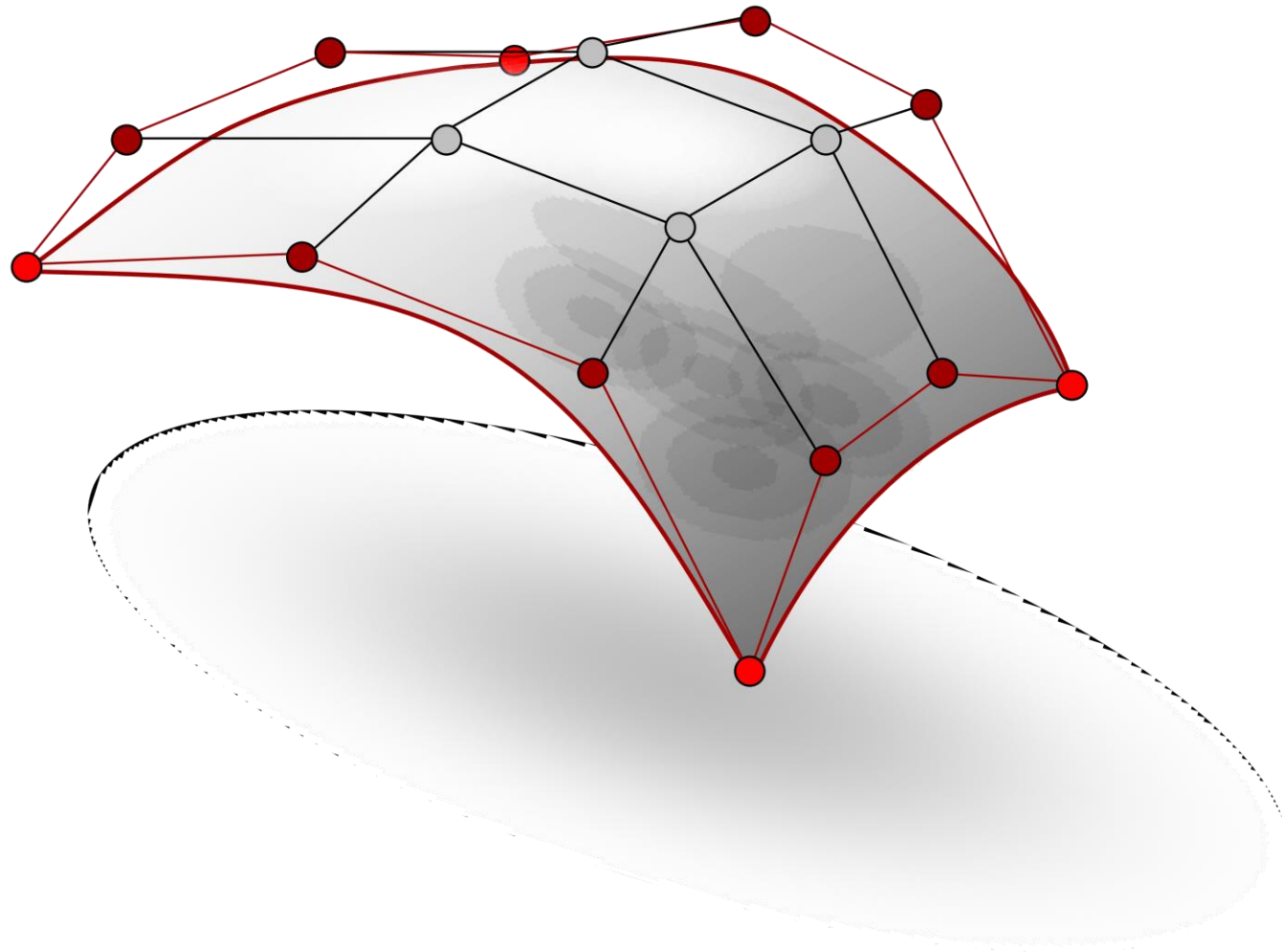
Consequence:

- We need more control over the parameter domain
- One solution is *trimming* using *curves on surfaces (CONS)*
- Standard tool in CAD: *trimmed NURBS*

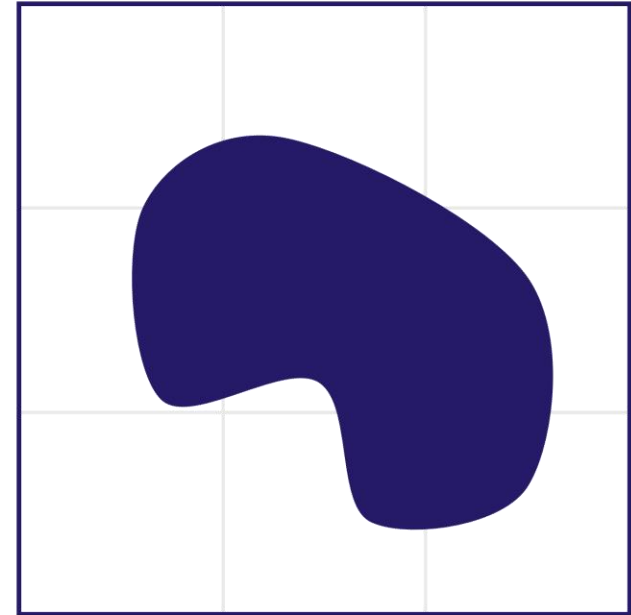
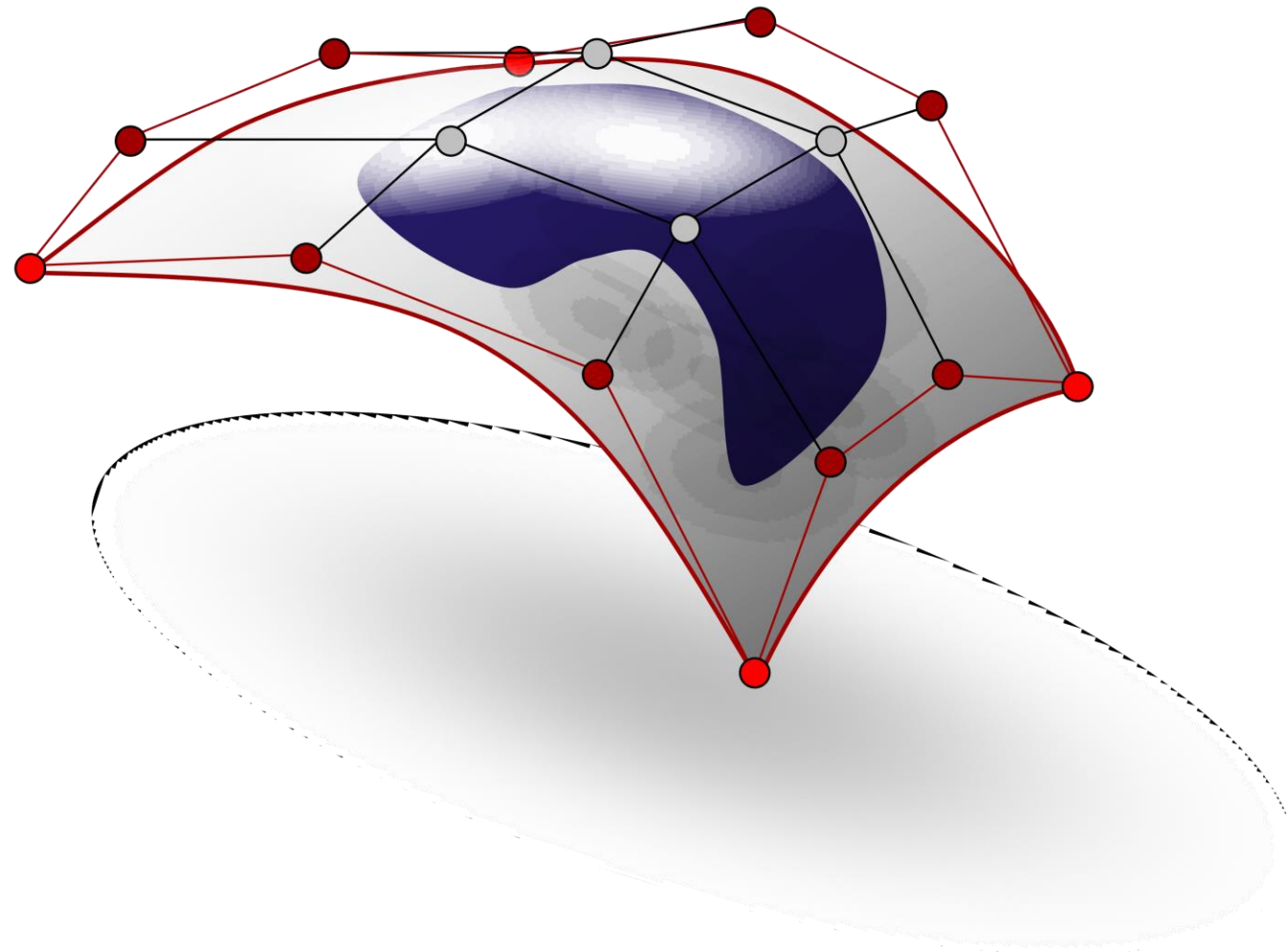
Basic idea:

- Specify a curve in the parameter domain that encapsulates one (or more) pieces of area
- Tessellate the parameter domain accordingly to cut out the trimmed piece (rendering)

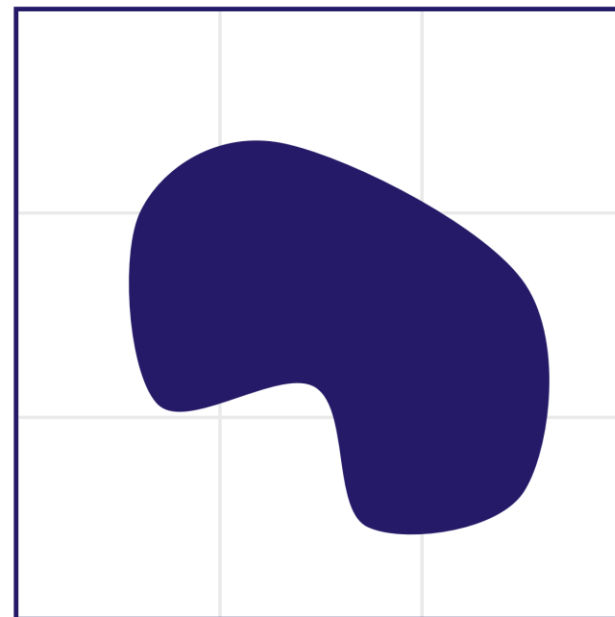
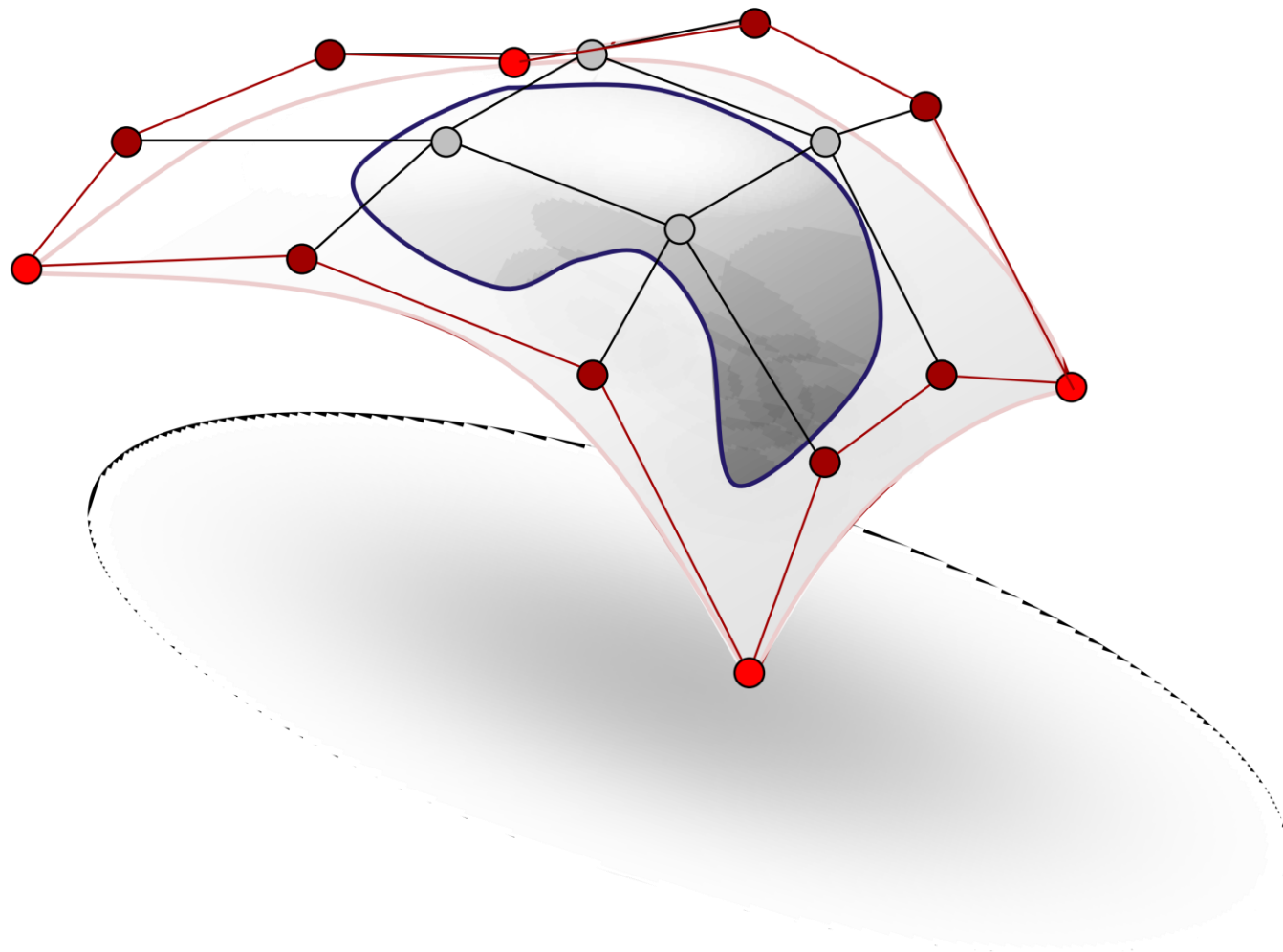
Curves-on-Surfaces (CONS)



Curves-on-Surfaces (CONS)



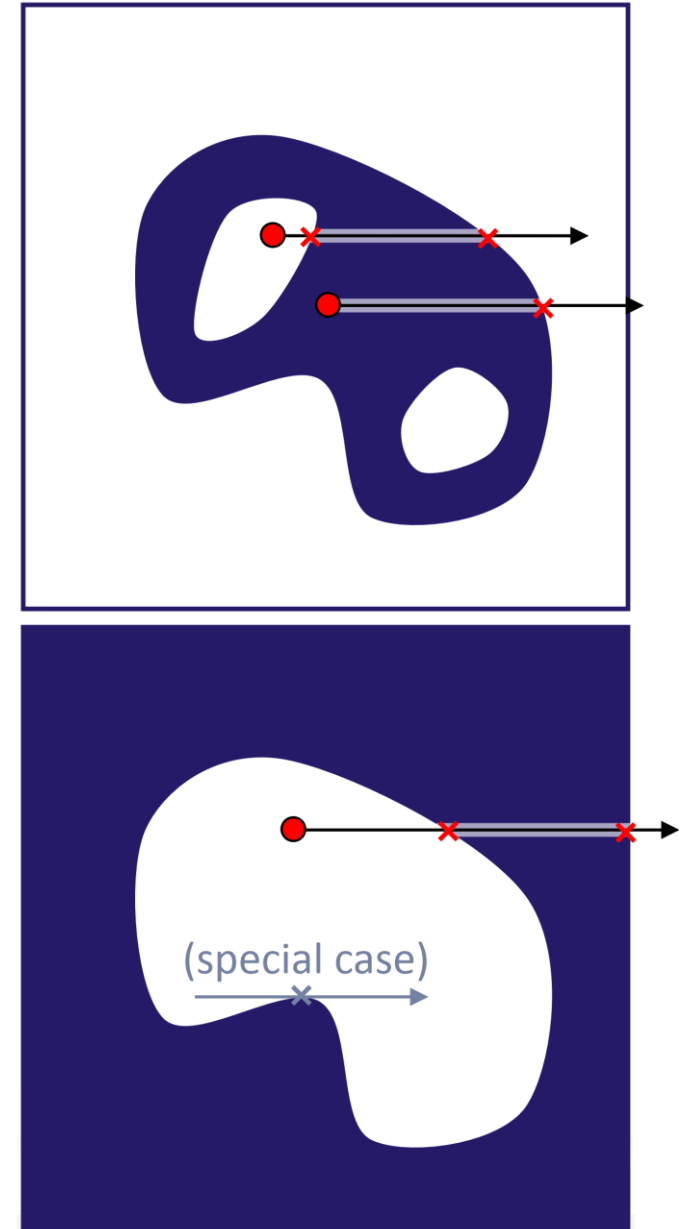
Curves-on-Surfaces (CONS)



General Shapes

General shapes with holes:

- Draw multiple curves
- Inside / outside test:
 - If any ray in the parameter domain intersects the boundary curves an odd number of times, the point is inside
 - Outside otherwise
 - Implementation needs to take care of special cases (critical points with respect to normal of the ray)
 - Nasty, but doable

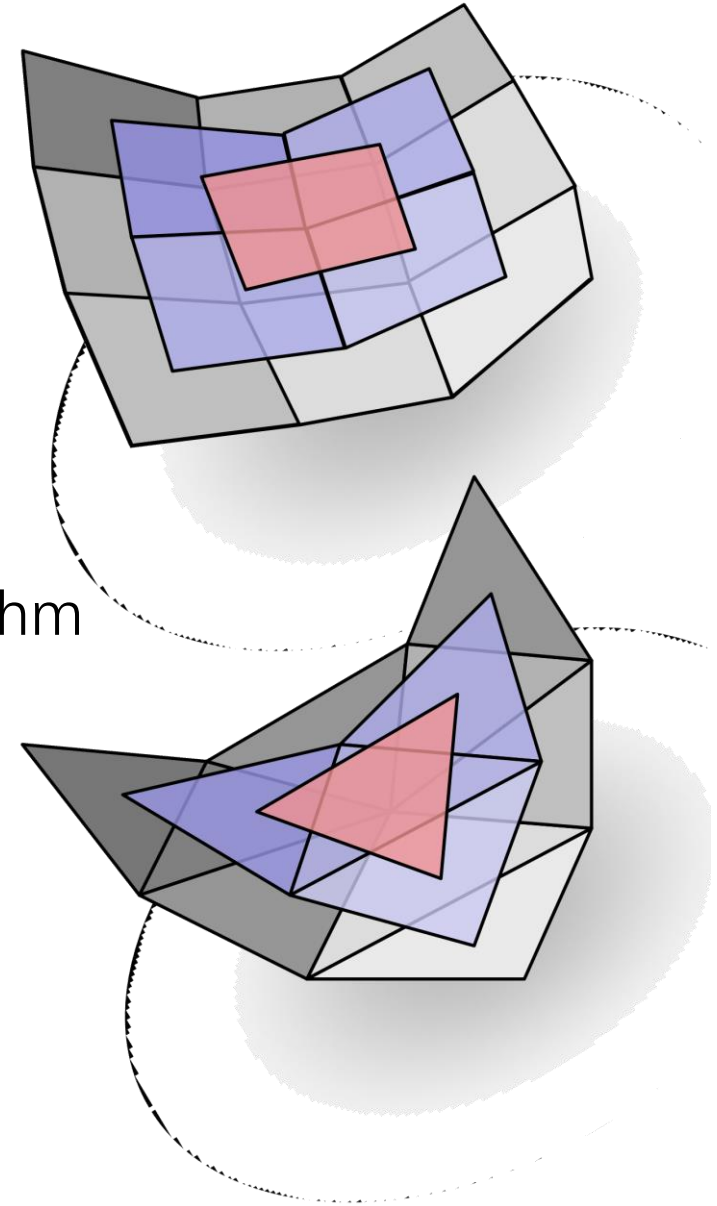


Total Degree Surfaces

Bézier Triangles

Alternative surface definition: Bézier triangles

- Constructed according to given total degree
 - Completely symmetric: degree anisotropy
- Can be derived using a triangular de Casteljau algorithm
 - Blossoming formalism is very helpful for defining Bézier Triangles
 - Barycentric interpolation of blossom values



Blossoms for Total Degree Surfaces

Blossom with points as arguments:

- Polar form degree d with points as input and output:

$$\begin{array}{l} \mathbf{F}: \mathbb{R}^n \rightarrow \mathbb{R}^m \\ \mathbf{f}: \mathbb{R}^{d \times n} \rightarrow \mathbb{R}^m \end{array} \quad \text{points as arguments}$$

- Required Properties:

- Diagonality: $\mathbf{f}(\mathbf{t}, \mathbf{t}, \dots, \mathbf{t}) = \mathbf{F}(\mathbf{t})$
- Symmetry: $\mathbf{f}(\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_d) = \mathbf{f}(\mathbf{t}_{\pi(1)}, \mathbf{t}_{\pi(2)}, \dots, \mathbf{t}_{\pi(d)})$
for all permutations of indices π

- Multi-affine: $\sum \alpha_k = 1$

$$\begin{aligned} \Rightarrow \mathbf{f}(\mathbf{t}_1, \dots, \sum \alpha_k \mathbf{t}_i^{(k)}, \dots, \mathbf{t}_d) \\ = \alpha_1 \mathbf{f}(\mathbf{t}_1, \dots, \mathbf{t}_i^{(1)}, \dots, \mathbf{t}_d) + \dots + \alpha_n \mathbf{f}(\mathbf{t}_1, \dots, \mathbf{t}_i^{(n)}, \dots, \mathbf{t}_d) \end{aligned}$$

Example

Example: bivariate monomial basis

- In powers of (u, v) :

$$B = \{1, u, v, u^2, uv, v^2\}$$

- Blossom form: multilinear in (u_1, u_2, v_1, v_2)

$$B = \{1, \frac{1}{2}(u_1 + u_2), \frac{1}{2}(v_1 + v_2), u_1u_2, \frac{1}{4}(u_1v_1 + u_1v_2 + u_2v_1 + u_2v_2), v_1v_2\}$$

Barycentric Coordinates

Barycentric Coordinates:

- Planar case:

Barycentric combinations of 3 points

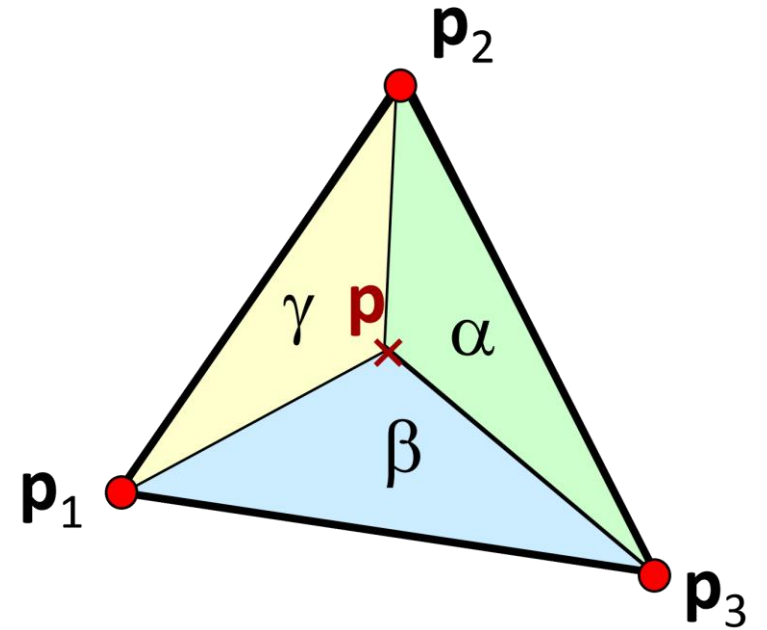
$$\mathbf{p} = \alpha \mathbf{p}_1 + \beta \mathbf{p}_2 + \gamma \mathbf{p}_3, \text{ with } \alpha + \beta + \gamma = 1$$

$$\gamma = 1 - \alpha - \beta$$

- Area formulation

$$\gamma = 1 - \alpha - \beta$$

$$\alpha = \frac{\text{area}(\Delta(\mathbf{p}_2, \mathbf{p}_3, \mathbf{p}))}{\text{area}(\Delta(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3))}, \beta = \frac{\text{area}(\Delta(\mathbf{p}_1, \mathbf{p}_3, \mathbf{p}))}{\text{area}(\Delta(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3))}, \gamma = \frac{\text{area}(\Delta(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}))}{\text{area}(\Delta(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3))}$$

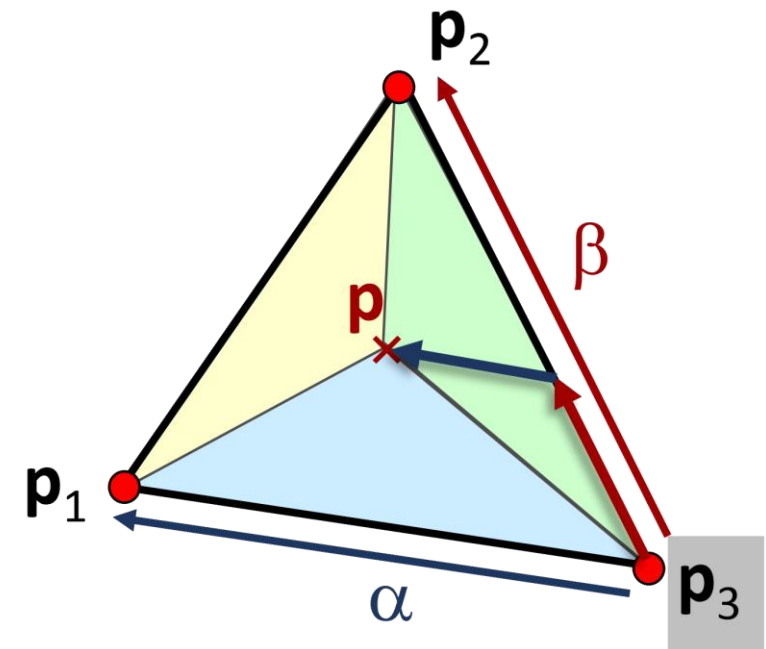
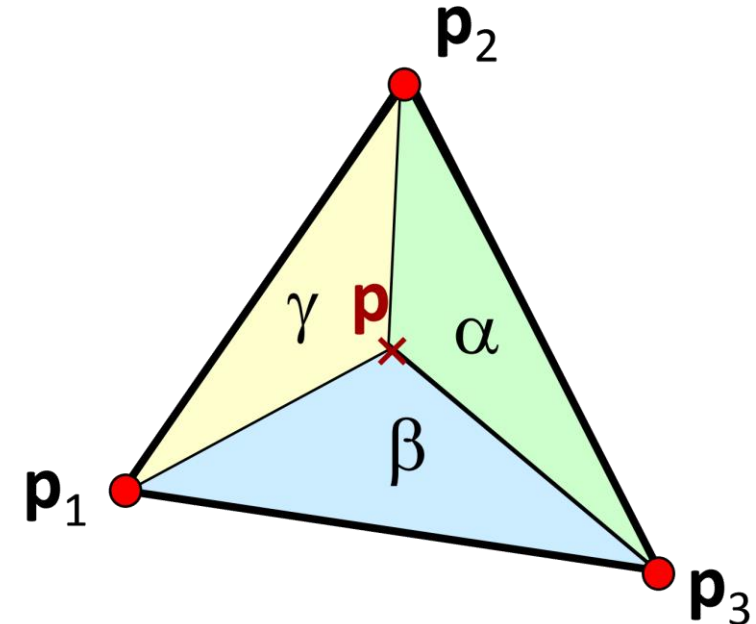


Barycentric Coordinates

Barycentric Coordinates:

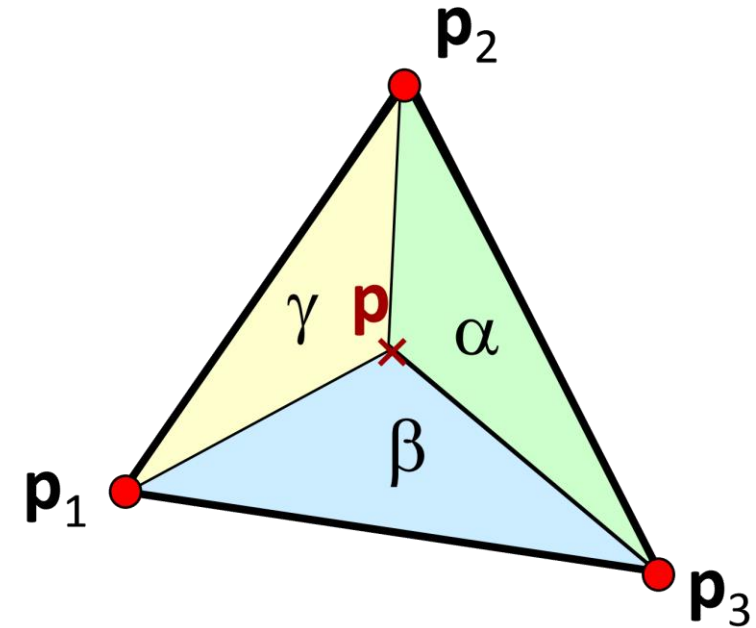
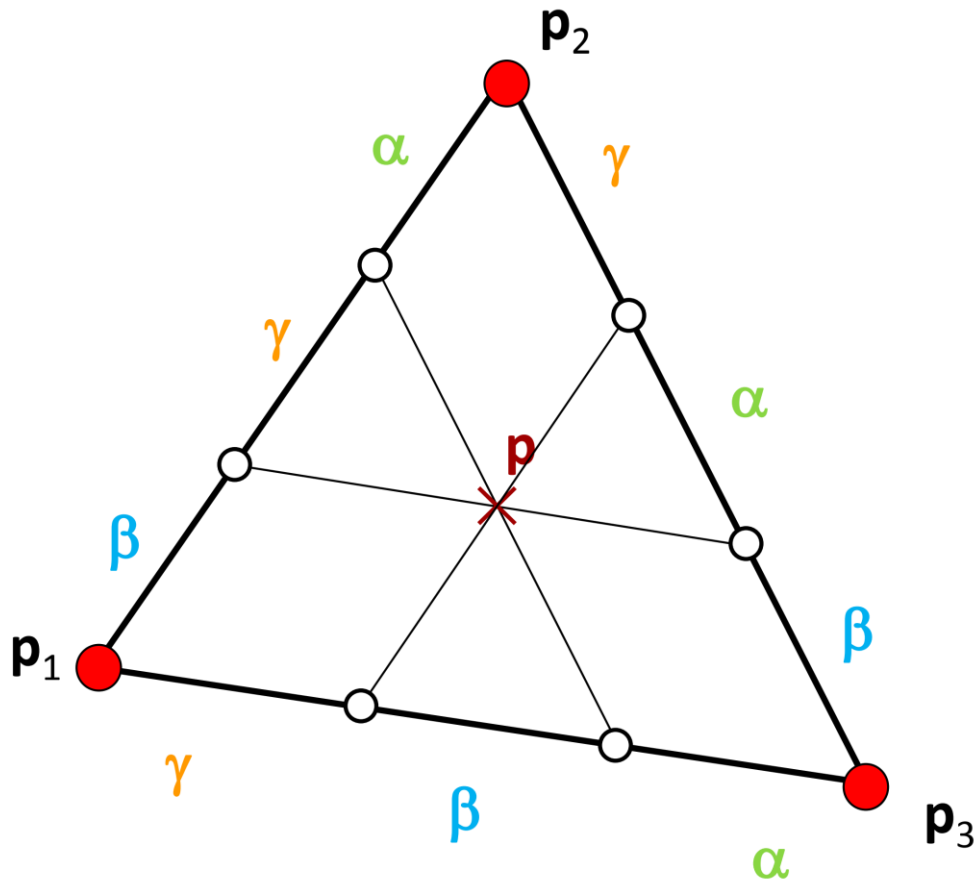
- Linear formulation:

$$\begin{aligned}\mathbf{p} &= \alpha \mathbf{p}_1 + \beta \mathbf{p}_2 + \gamma \mathbf{p}_3 \\ &= \alpha \mathbf{p}_1 + \beta \mathbf{p}_2 + (1 - \alpha - \beta) \mathbf{p}_3 \\ &= \alpha \mathbf{p}_1 + \beta \mathbf{p}_2 + \mathbf{p}_3 - \alpha \mathbf{p}_3 - \beta \mathbf{p}_3 \\ &= \mathbf{p}_3 + \alpha(\mathbf{p}_1 - \mathbf{p}_3) + \beta(\mathbf{p}_2 - \mathbf{p}_3)\end{aligned}$$



Barycentric Coordinates

$$\mathbf{p} = \alpha \mathbf{p}_1 + \beta \mathbf{p}_2 + \gamma \mathbf{p}_3, \text{ with } \alpha + \beta + \gamma = 1$$



Bézier Triangles: Overview

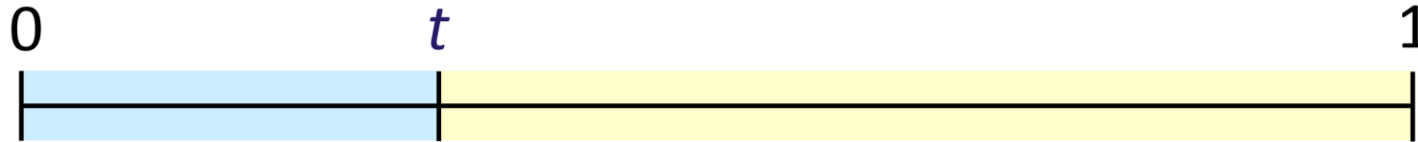
Bézier Triangles: Main Ideas

- Use 3D points as inputs to the blossoms
- These are Barycentric coordinates of a parameter triangle $\{a, b, c\}$
- Use 3D points as outputs
- Form control points by multiplying parameter points, just as in the curve case: $p(\underbrace{a, \dots, a}_i, \underbrace{b, \dots, b}_j, \underbrace{c, \dots, c}_k)$
- De Casteljau Algorithm: compute polynomial values $p(x, \dots, x)$ by barycentric interpolation

Plugging in the Barycentric Coord's

Analog: 2D curves in barycentric coordinates

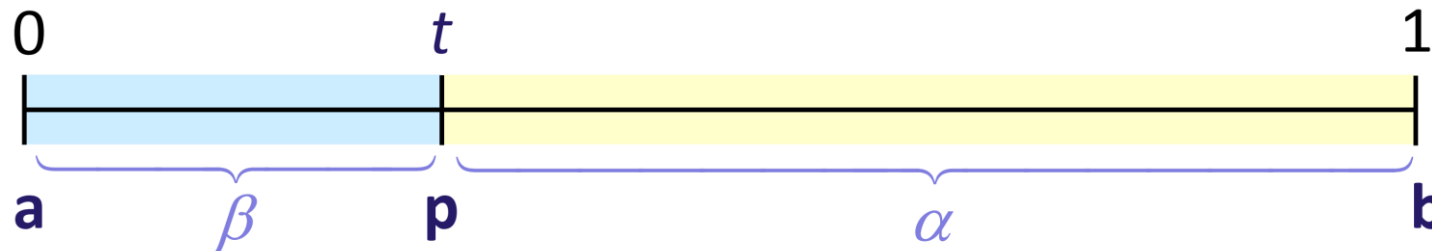
- Barycentric coordinates for 2D curves:



Plugging in the Barycentric Coord's

Analog: 2D curves in barycentric coordinates

- Barycentric coordinates for 2D curves:



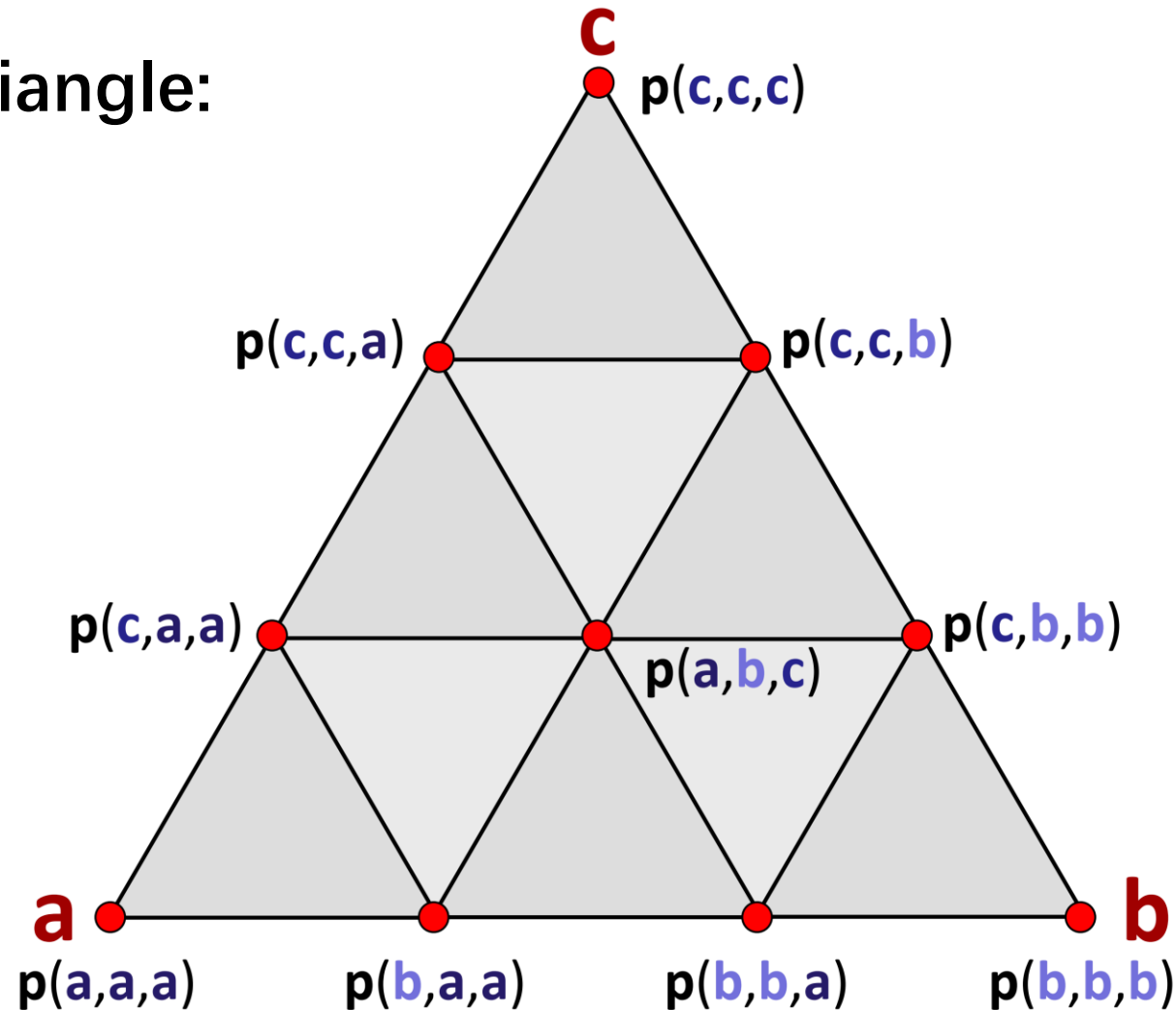
- $p = \alpha a + \beta b, \quad \alpha + \beta = 1$
- Bézier splines:

$$F(t) = \sum_{i=0}^d \binom{d}{i} (1-t)^i t^{d-i} f(\underbrace{a, \dots, a}_i, \underbrace{b, \dots, b}_{d-i}) \quad (\text{standard form})$$

$$F(p) = \sum_{\substack{i+j=d \\ i \geq 0, j \geq 0}} \frac{d!}{i!j!} \alpha^i \beta^j f(\underbrace{a, \dots, a}_i, \underbrace{b, \dots, b}_j) \quad (\text{barycentric form})$$

Example

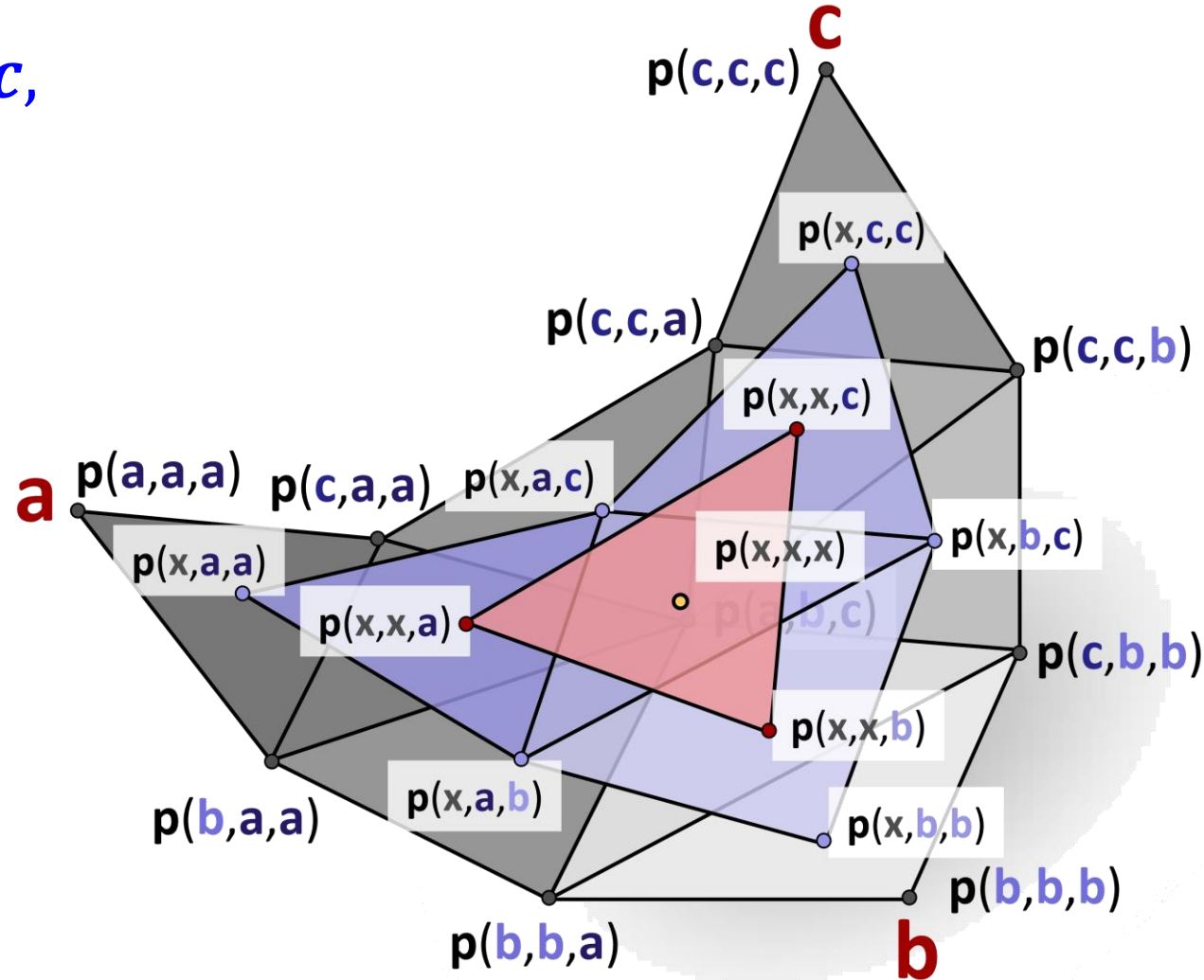
Cubic Bézier Triangle:



De Casteljau Algorithm

$$x = \alpha a + \beta b + \gamma c,$$

$$\alpha + \beta + \gamma = 1$$



Bernstein Form

Writing this recursion out, we obtain:

$$F(\mathbf{x}) = \sum_{\substack{i+j+k=d \\ i,j,k \geq 0}} \frac{d!}{i!j!k!} \alpha^i \beta^j \gamma^k f(\underbrace{a, \dots, a}_i, \underbrace{b, \dots, b}_j, \underbrace{c, \dots, c}_k)$$

$$\mathbf{x} = \alpha \mathbf{a} + \beta \mathbf{b} + \gamma \mathbf{c},$$

$$\alpha + \beta + \gamma = 1$$

- This is the *Bernstein form* of a Bézier triangle surface
- (Proof by induction)

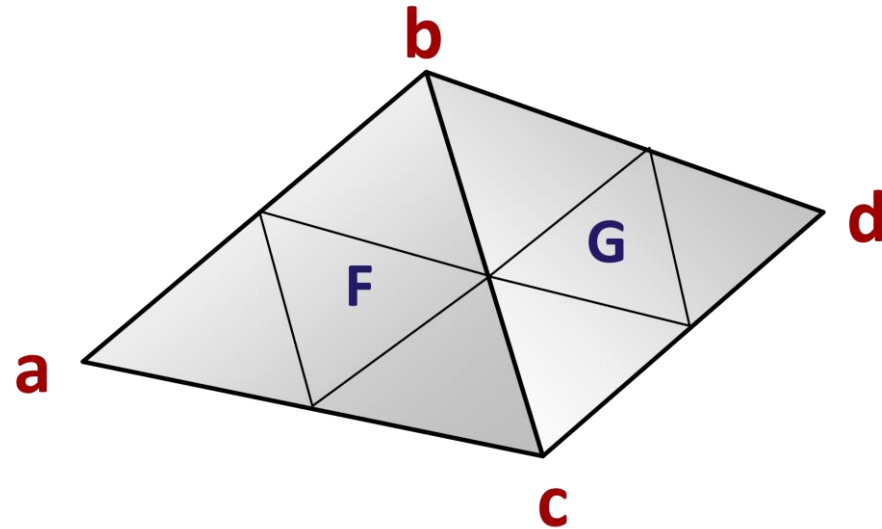
Continuity

We need to assemble Bézier triangles continuously:

- What are the conditions for \mathcal{C}^0 , \mathcal{C}^1 continuity?
- As an example, we will look at the quadratic case...
- (Try the cubic case as an exercise)

Continuity

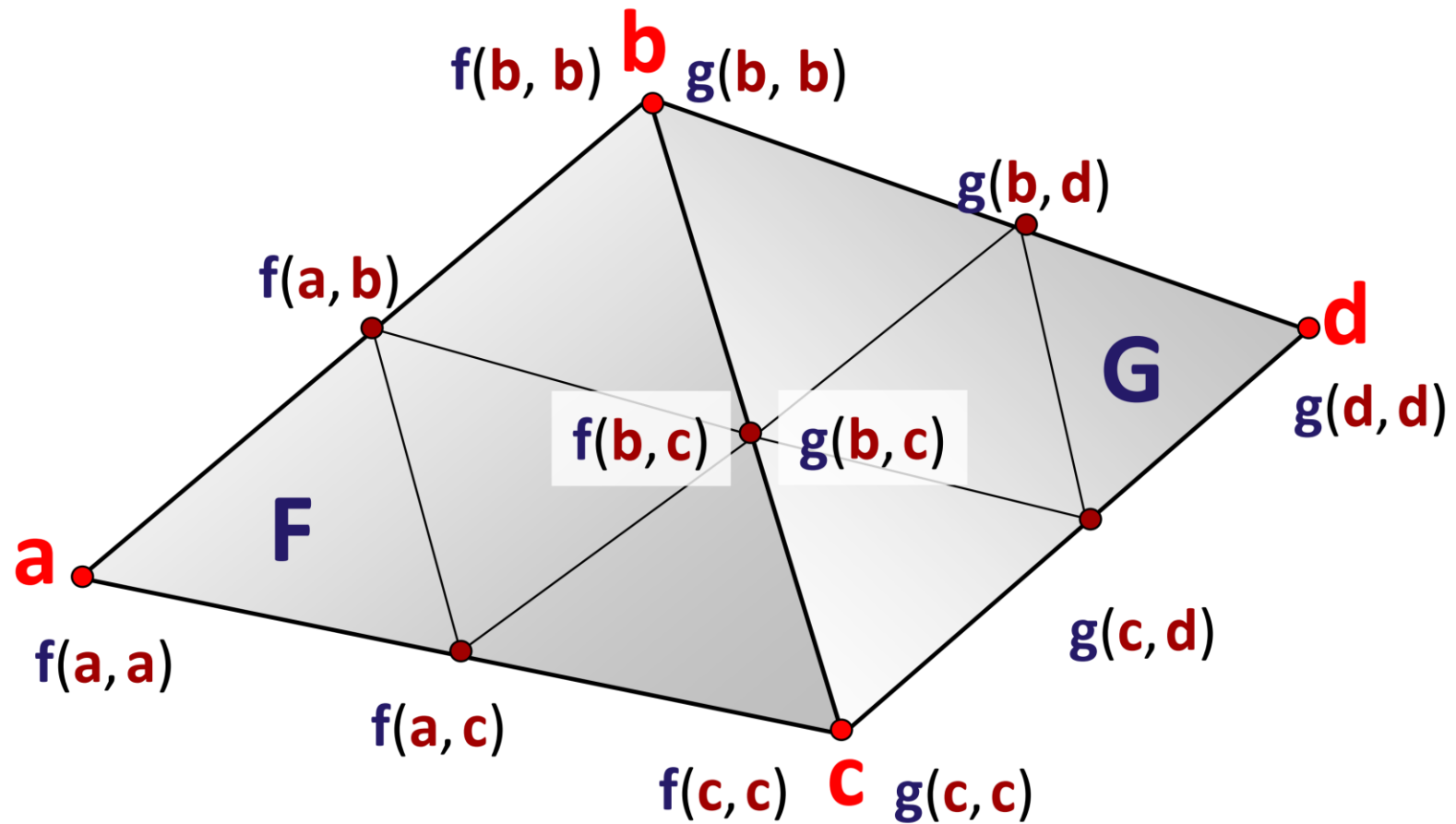
Situation:



- Two Bézier triangles meet along a common edge.
 - Parametrization: $T_1 = \{a, b, c\}, T_2 = \{c, b, d\}$
 - Polynomial surfaces $F(T_1), G(T_2)$
 - Control points:
 - $F(T_1)$: $f(a, a), f(a, b), f(b, b), f(a, c), f(c, c), f(b, c)$
 - $G(T_2)$: $g(d, d), g(d, b), g(b, b), g(d, c), g(c, c), g(b, c)$

Continuity

Situation:



Continuity

C^0 continuity:

- The points on the boundary have to agree:

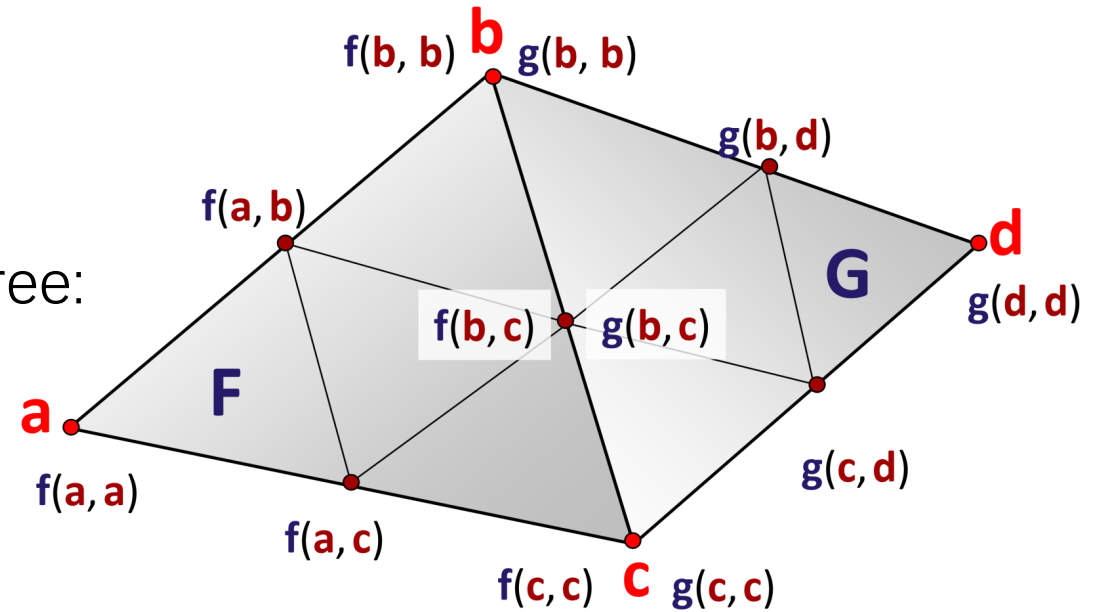
$$f(\mathbf{b}, \mathbf{b}) = g(\mathbf{b}, \mathbf{b})$$

$$f(\mathbf{b}, \mathbf{c}) = g(\mathbf{b}, \mathbf{c})$$

$$f(\mathbf{c}, \mathbf{c}) = g(\mathbf{c}, \mathbf{c})$$

- Proof: Let $\mathbf{x} := \beta \mathbf{b} + \gamma \mathbf{c}$, $\beta + \gamma = 1$

$$\begin{aligned} f(\mathbf{x}, \mathbf{x}) &= \beta f(\mathbf{b}, \mathbf{x}) + \gamma f(\mathbf{c}, \mathbf{x}) \\ &= \beta^2 f(\mathbf{b}, \mathbf{b}) + 2\beta\gamma f(\mathbf{b}, \mathbf{c}) + \gamma^2 f(\mathbf{c}, \mathbf{c}) \\ &\quad \parallel \quad \parallel \quad \parallel \\ &\quad g(\mathbf{b}, \mathbf{b}) \quad g(\mathbf{b}, \mathbf{c}) \quad g(\mathbf{c}, \mathbf{c}) \\ &= \beta^2 g(\mathbf{b}, \mathbf{b}) + 2\beta\gamma g(\mathbf{b}, \mathbf{c}) + \gamma^2 g(\mathbf{c}, \mathbf{c}) \\ &= \beta g(\mathbf{b}, \mathbf{x}) + \gamma g(\mathbf{c}, \mathbf{x}) = g(\mathbf{x}, \mathbf{x}) \end{aligned}$$



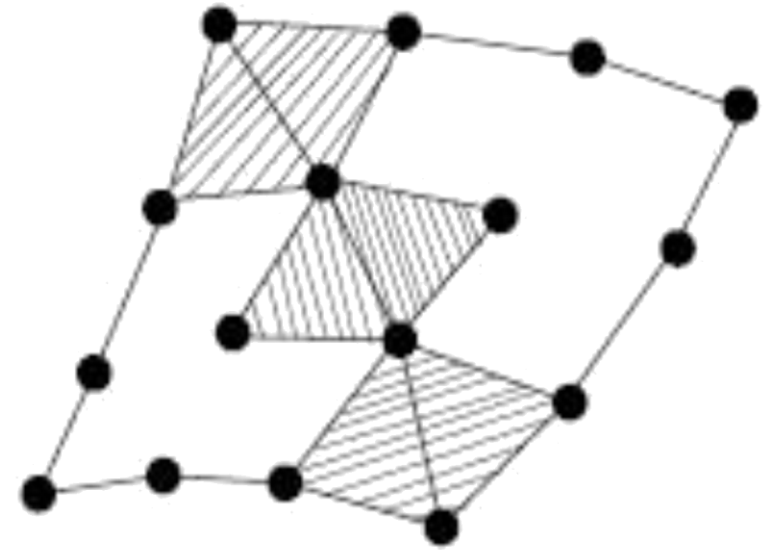
Continuity

C^1 continuity:

- We need C^0 continuity.

In addition:

- Points at hatched quadrilaterals are coplanar
- Hatched quadrilaterals are an affine image of the same parameter quadrilateral



Continuity

C^1 continuity:

- We need C^0 continuity.

In addition:

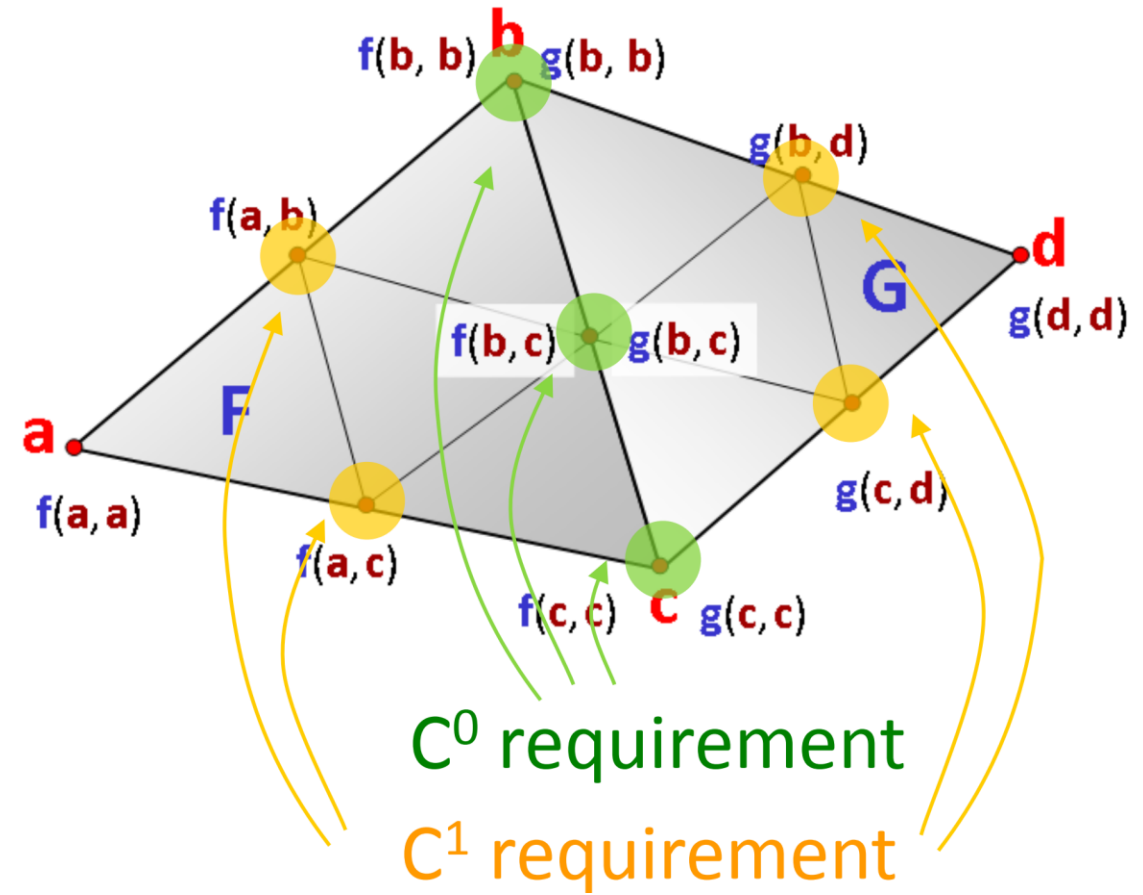
The blossoms have to agree partially:

$$f(a, b) = g(a, b)$$

$$f(b, d) = g(b, d)$$

$$f(a, c) = g(a, c)$$

$$f(c, d) = g(c, d)$$



Continuity

\mathcal{C}^1 continuity: Proof

- Derivatives:

$$\frac{\partial}{\partial \hat{d}} F(x)|_{x=p} = f(p, \hat{d})$$

(similar to the curve case)

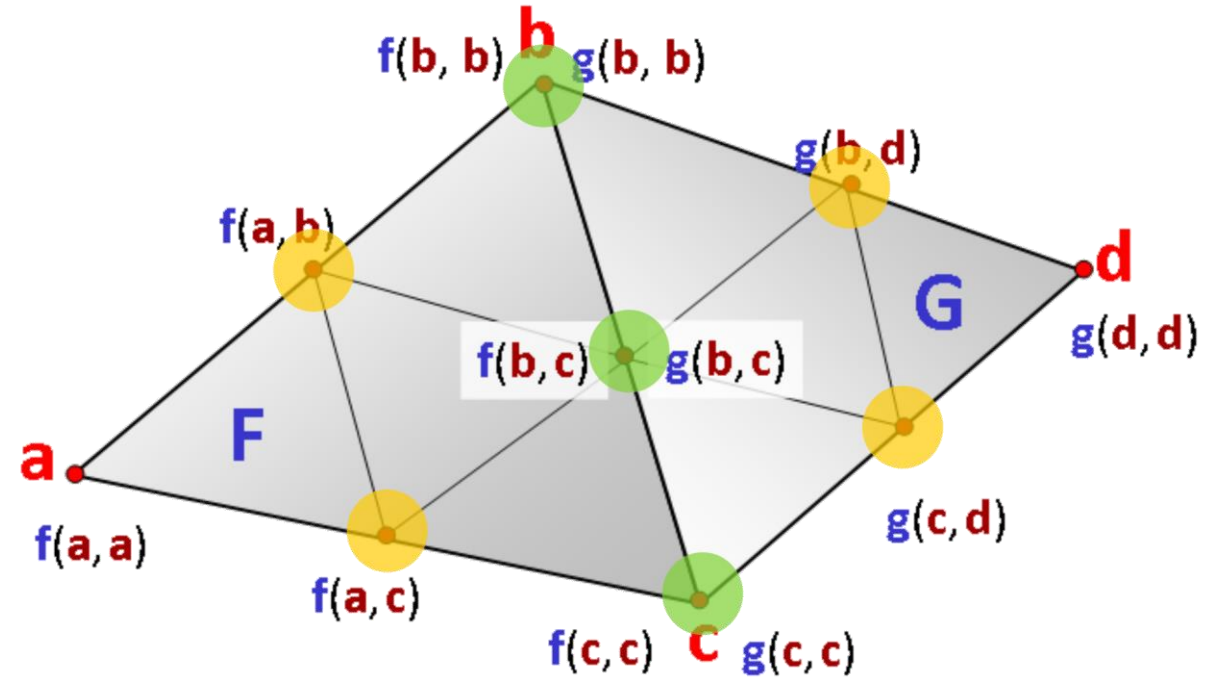
- \mathcal{C}^1 -Continuity:

$$\forall x \in \mathbb{R}^3: f(p, x) = g(p, x)$$

- We have to show

$$\forall x \in \mathbb{R}^3: \begin{cases} f(b, x) = g(b, x) \\ f(c, x) = g(c, x) \end{cases}$$

- $\Rightarrow \mathcal{C}^1$ continuity follows for all boundary points (by interp.)



Continuity

C^1 continuity: Proof

- So we have to show

$$\forall x \in \mathbb{R}^3: \begin{cases} f(b, x) = g(b, x) \\ f(c, x) = g(c, x) \end{cases}$$

- Proof:

Write $x = \alpha \mathbf{a} + \beta \mathbf{b} + \gamma \mathbf{c}$ (coordinate system)

$$f(b, x) = \alpha f(a, b) + \beta f(b, b) + \gamma f(b, c)$$

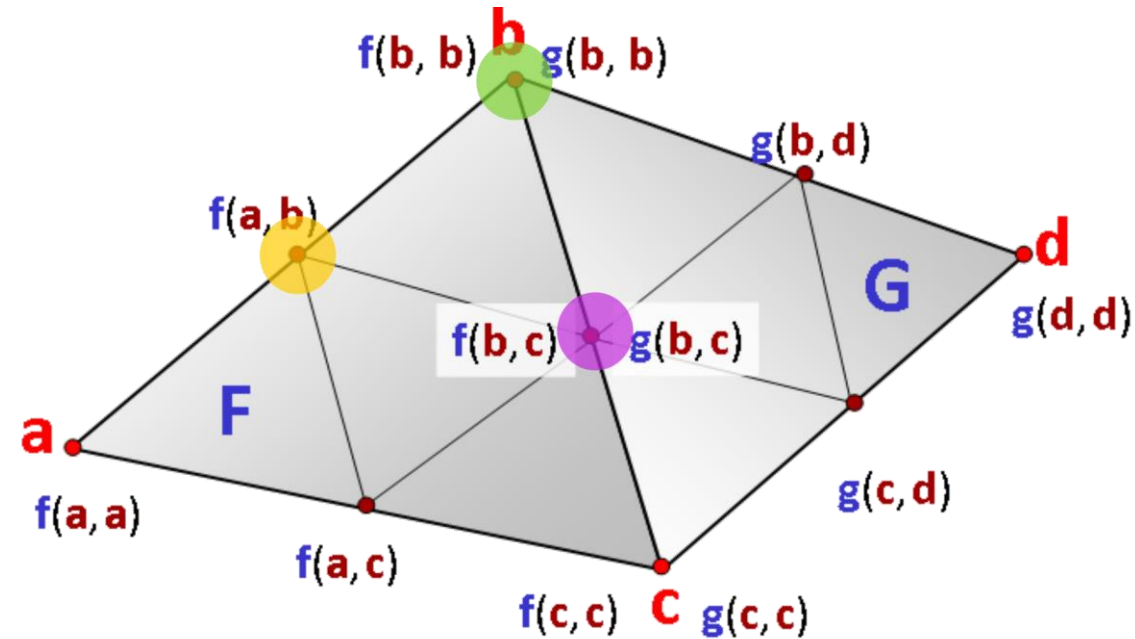
$$g(b, x) = \alpha g(a, b) + \beta g(b, b) + \gamma g(b, c)$$

$$f(b, x) = g(b, x) \Leftrightarrow \alpha f(a, b) + \beta f(b, b) + \gamma f(b, c)$$

$$= \alpha g(a, b) + \beta g(b, b) + \gamma g(b, c)$$

$$\Leftrightarrow f(a, b) = g(a, b)$$

(same for the other conditions)



Continuity

So what does this mean?

- The blossoms have to agree partially:

$$f(a, b) = g(a, b)$$

$$f(b, d) = g(b, d)$$

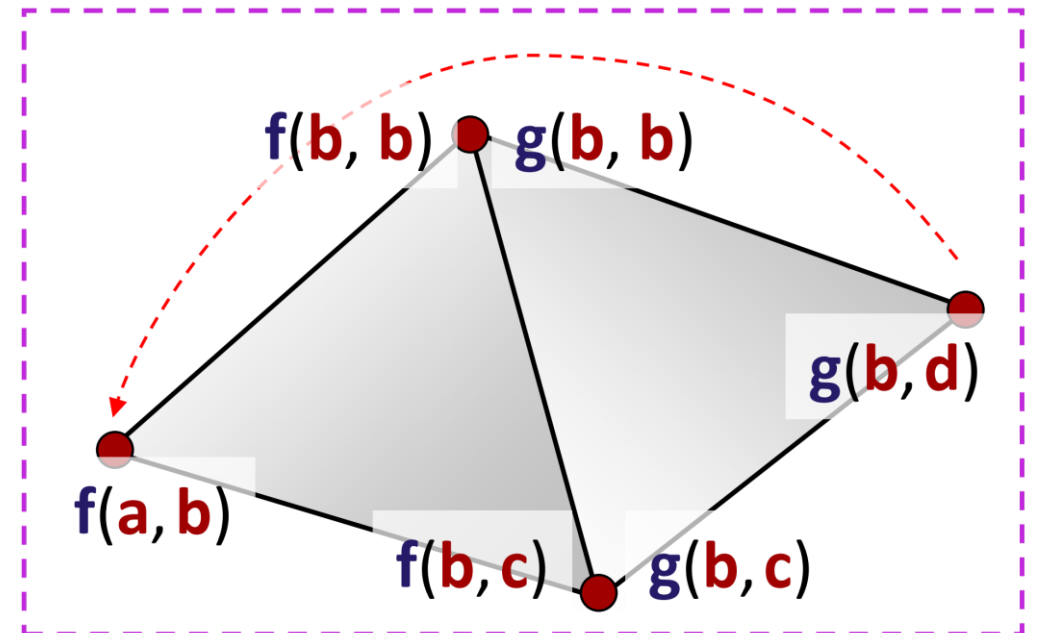
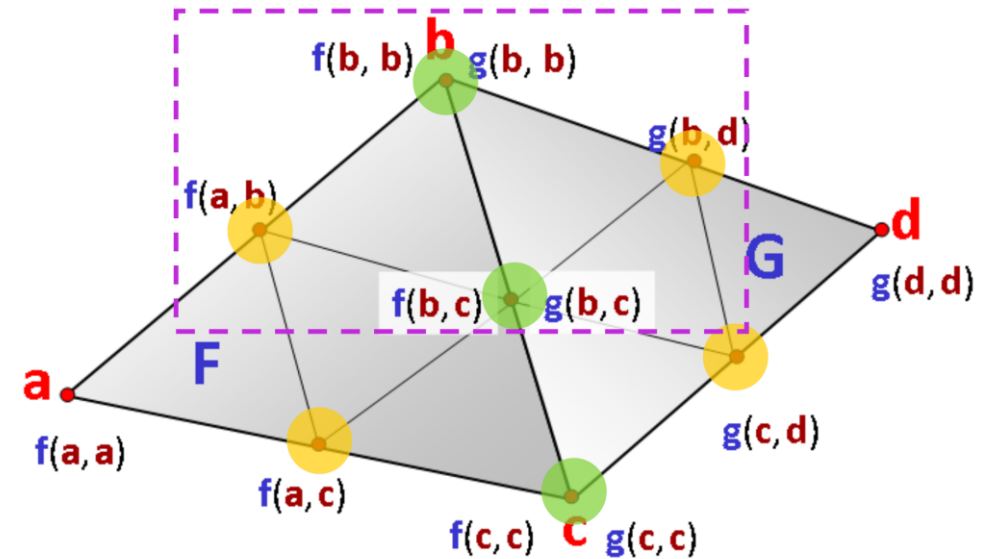
$$f(a, c) = g(a, c)$$

$$f(c, d) = g(c, d)$$

- The points must be coplanar
(with edge points):

$$f(a, b), g(b, d), g(b, b), g(b, c)$$

- The points in **F** must be affine images of the points in **G**



Computer Aided Geometric Design

Fall Semester 2024

Subdivision Curves and Surfaces

陈仁杰

renjiec@ustc.edu.cn

<http://staff.ustc.edu.cn/~renjiec>

Subdivision Surfaces

Problem with Spline Patches

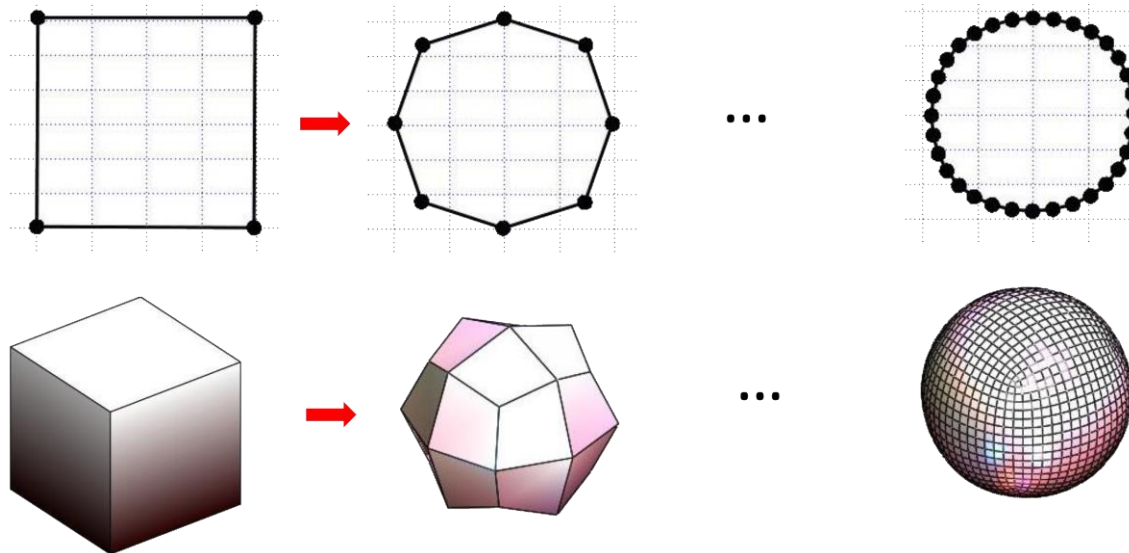
- A continuous tensor product spline surface is only defined on a regular grid of quads as parametrization domain
- Thus, the topology of the object is restricted
- Assembling multiple parameter domains to a single surface is tedious, hard to get continuity guarantees
- Handling trimming curves is not that straightforward

Question: can we do better?

Subdivision Surfaces

Wish list:

- Provide a very coarse representation of the geometry
- Obtain a fine and smooth representation
- Preferably by means of a simple set of rules which can be recursively applied (subdivision rules or subdivision scheme)



Subdivision Surfaces

Bigger goals:

- Simplify the creation of smooth refined geometric models (especially in feature film industry)

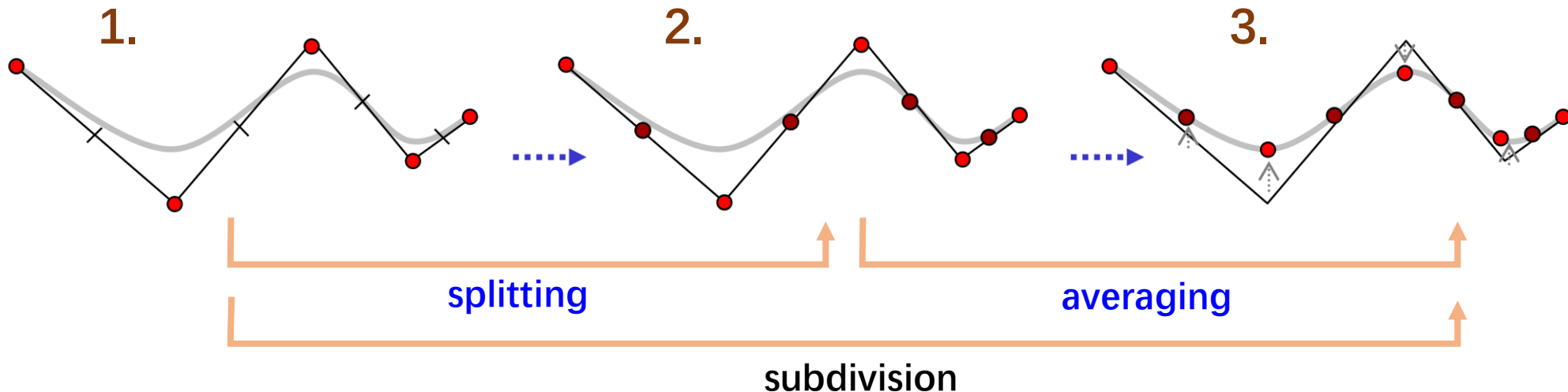


- What's lost? Parametric representation ...

Basic Scheme

Subdivision Curves & Surfaces: Three Steps

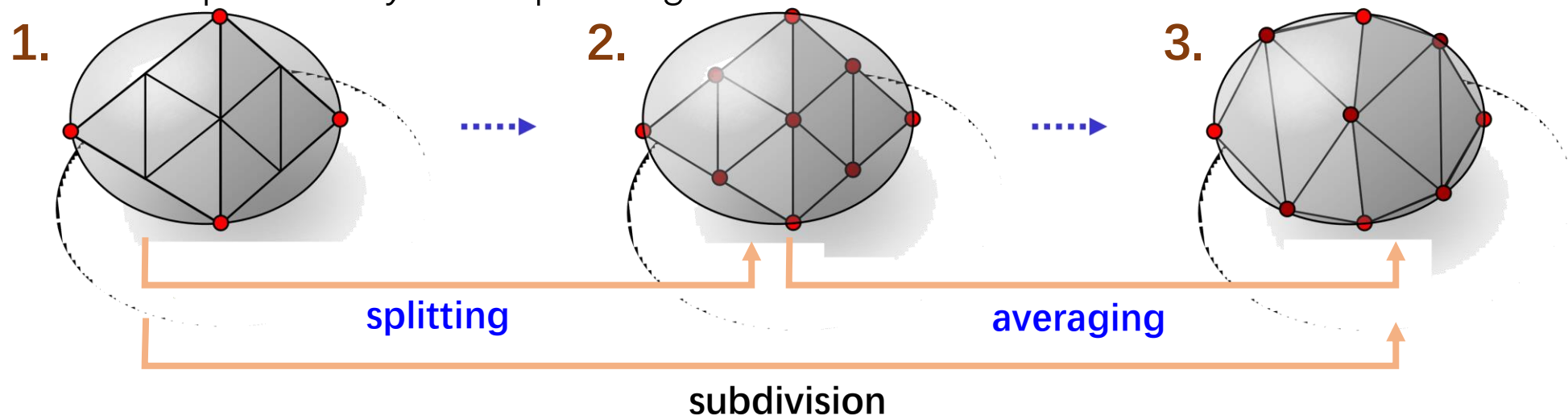
- Subdivide current polygon
- Insert linearly interpolated points (*splitting*)
- Move points: local weighted average (*averaging*)
 - To all points – approximating scheme
 - To new points only – interpolating scheme



Basic Scheme

Subdivision Curves & Surfaces: Three Steps

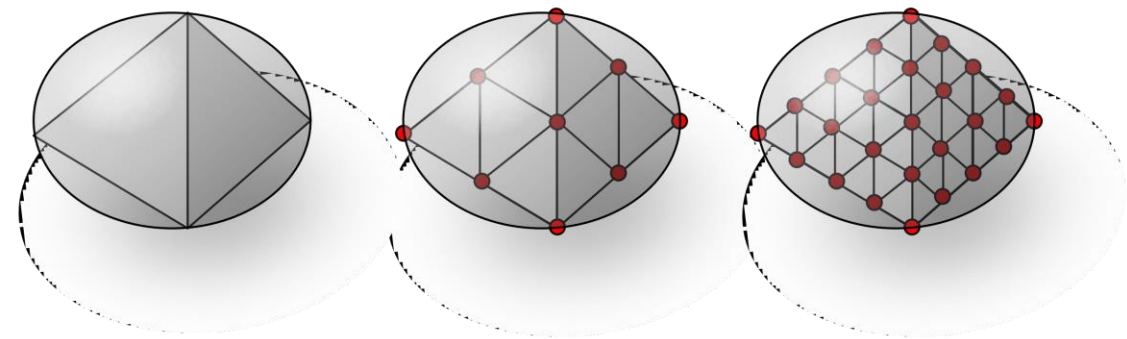
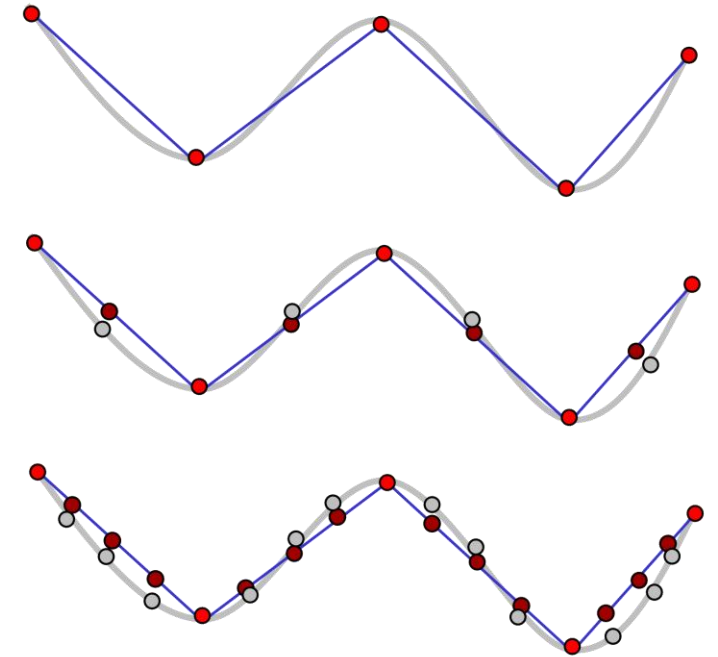
- Subdivide current mesh
- Insert linearly interpolated points (*splitting*)
- Move points: local weighted average (*averaging*)
 - To all points – approximating scheme
 - To new points only – interpolating scheme



Subdivision Surfaces

The main question is:

- How should we place the new points to create a smooth surface?
(interpolating scheme)
- Respectively: how should we alter the points in each subdivision step to create a smooth surface?
(approximating scheme)



Subdivision Schemes

More precisely

- What are good *averaging masks*?
- The averaging mask determines the weights by which new point positions are computed

Interesting observation:

- Most averaging schemes do not converge
(in particular interpolating schemes)
- We need to be very careful to design a good averaging mask
- How can we guarantee C^1 , C^2 surfaces?

Subdivision Surfaces – History

de Rahm described a 2D (curve) subdivision scheme in 1947;
rediscovered in 1974 by Chaikin

Concept extended to 3D (surface) schemes by two separate groups in 1978:

- Doo and Sabin found a biquadratic surface
- Catmull and Clark found a bicubic surface

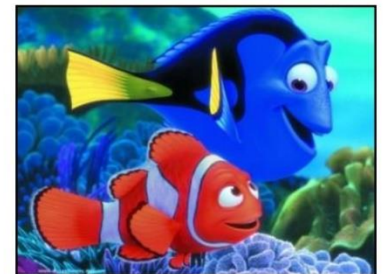
Subsequent work in the 1980s (Loop 1987, Dyn [Butterfly subdivision] 1990) led to tools suitable for CAD/CAM and animation

Subdivision Surfaces and the Movies

Pixar first demonstrated subdivision surfaces in 1997 with Geri's Game

- Up until then they'd done everything in NURBS (Toy Story, a Bug's Life)
- From 1999 onwards, everything they did was with subdivision surfaces (Toy Story 2, Monsters Inc, Finding Nemo...)

It's not clear what Dreamworks uses, but they have recent patents on subdivision techniques

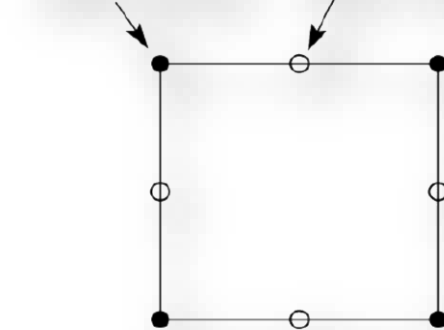


Curves Revisited

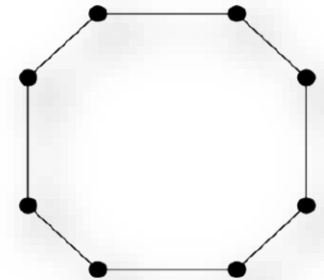
Corner Cutting Splines [Chaikin 1974]:

1. Split each line segment in half
 2. Average every point with its next neighbor (clock-wise)
 3. Repeat
- Converges to quadratic B-Spline curve

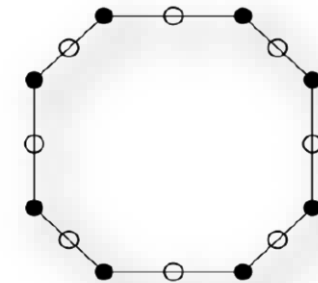
Old vertex New vertex



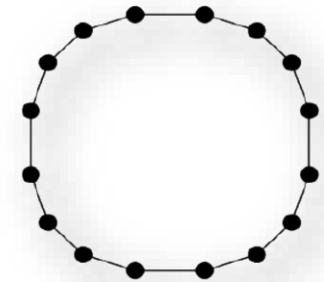
1. Split



2. Average



3. Split

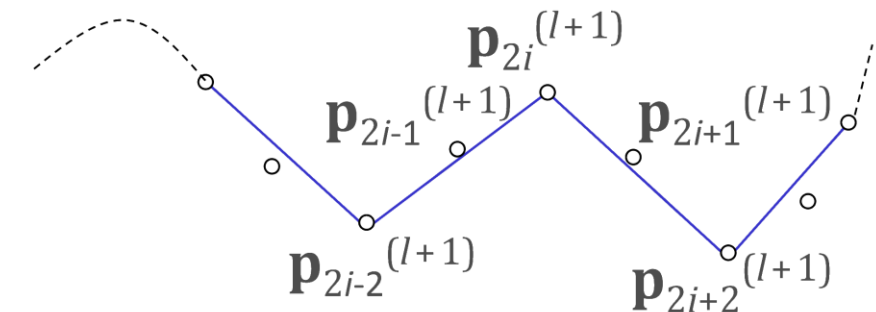
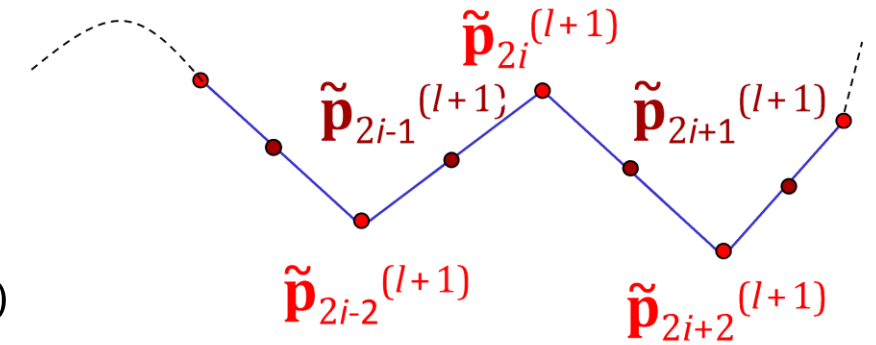
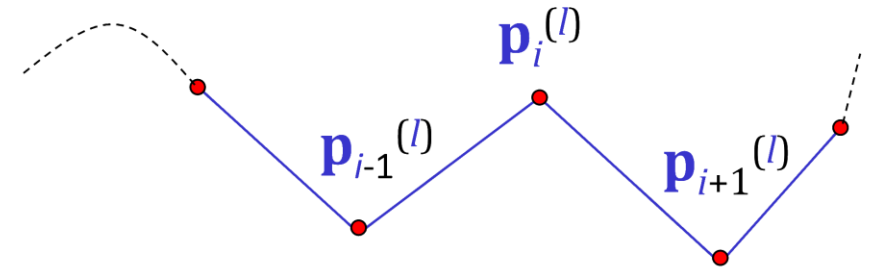


4. Average

Matrix Notation

Curve Subdivision in matrix notation:

- Control points at level l : $\mathbf{p}_i^{(l)}$
- “Splitted” points at level $l + 1$: $\tilde{\mathbf{p}}_i^{(l+1)}$
- “Averaged” control points at level $l + 1$: $\mathbf{p}_i^{(l+1)}$



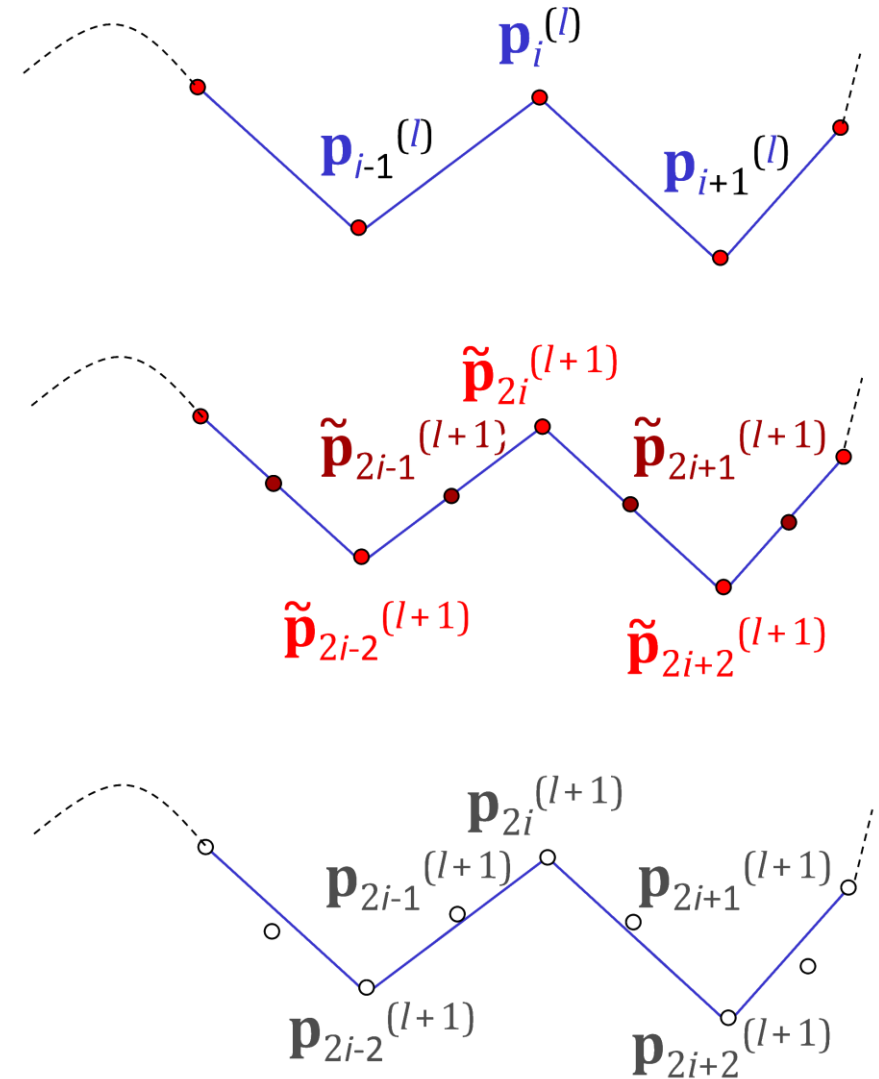
Matrix Notation

Splitting in matrix notation

$$2n \left\{ \begin{pmatrix} \vdots \\ \tilde{x}_{2i}^{(l+1)} \\ \tilde{x}_{2i+1}^{(l+1)} \\ \vdots \end{pmatrix} \right\} = 2n \left\{ \underbrace{\begin{pmatrix} \ddots & & & \\ & 1 & & \\ & 1/2 & 1/2 & \\ & & 1 & \\ & & 1/2 & 1/2 & \\ & & & \ddots \end{pmatrix}}_n \begin{pmatrix} \vdots \\ x_i^{(l)} \\ x_{i+1}^{(l)} \\ \vdots \end{pmatrix} \right\}_n$$

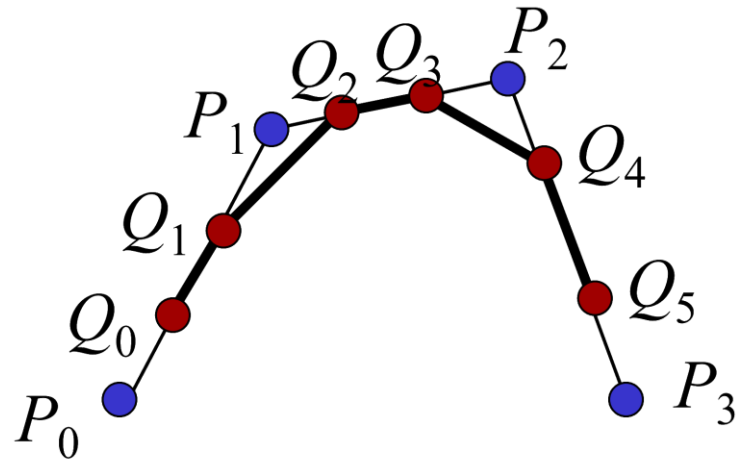
Averaging in matrix notation

$$2n \left\{ \begin{pmatrix} \vdots \\ x_{2i}^{(l+1)} \\ x_{2i+1}^{(l+1)} \\ \vdots \end{pmatrix} \right\} = 2n \left\{ \underbrace{\begin{pmatrix} \ddots & & & \\ & 1/2 & 1/2 & \\ & & 1/2 & 1/2 & \\ & & & \ddots \end{pmatrix}}_{2n} \begin{pmatrix} \vdots \\ \tilde{x}_{2i}^{(l+1)} \\ \tilde{x}_{2i+1}^{(l+1)} \\ \vdots \end{pmatrix} \right\}_{2n}$$



a different view on the same algorithm...

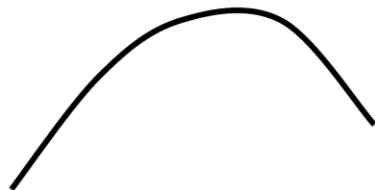
Chaikin's Corner Cutting



Apply
Iterated
Function
System

$$Q_{2i} = \frac{3}{4}P_i + \frac{1}{4}P_{i+1}$$

$$Q_{2i+1} = \frac{1}{4}P_i + \frac{3}{4}P_{i+1}$$



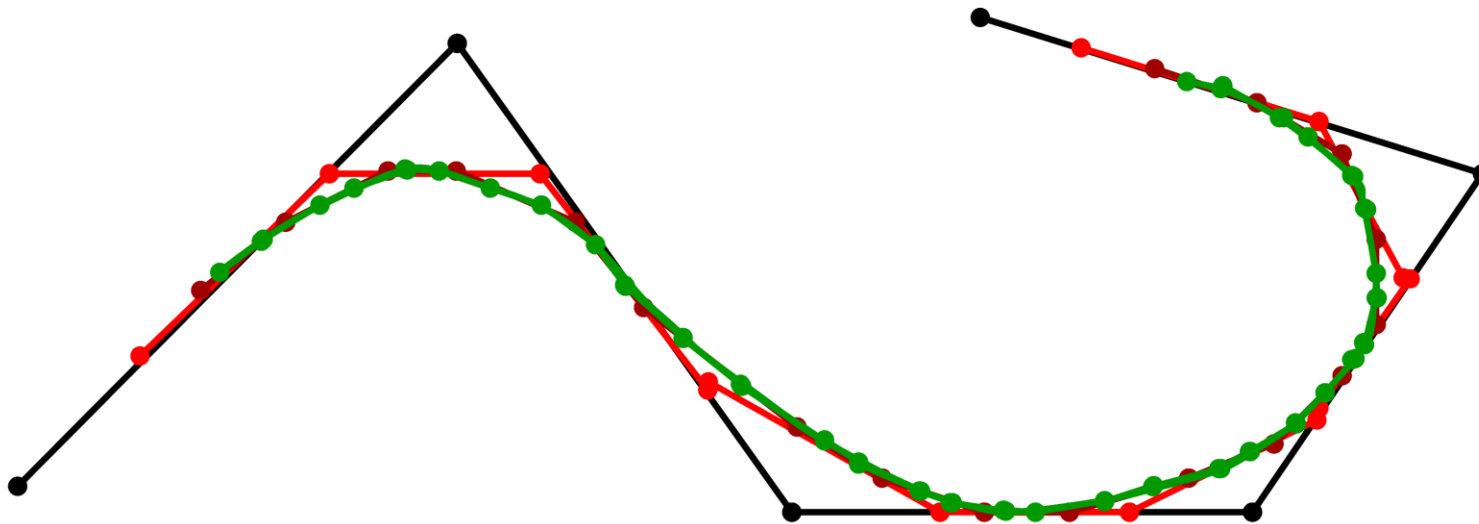
Limit Curve/Surface

- $Q_0 = \frac{3}{4}P_0 + \frac{1}{4}P_1$
- $Q_1 = \frac{1}{4}P_0 + \frac{3}{4}P_1$
- $Q_2 = \frac{3}{4}P_1 + \frac{1}{4}P_2$
- $Q_3 = \frac{1}{4}P_1 + \frac{3}{4}P_2$
- $Q_4 = \frac{3}{4}P_2 + \frac{1}{4}P_3$
- $Q_5 = \frac{1}{4}P_2 + \frac{3}{4}P_3$

Chaikin's Corner Cutting

Chaikin curve subdivision (2D)

- On each edge, insert new control points at $1/4$ and $3/4$ between old vertices; delete old points
- The *limit curve* is C^1 everywhere



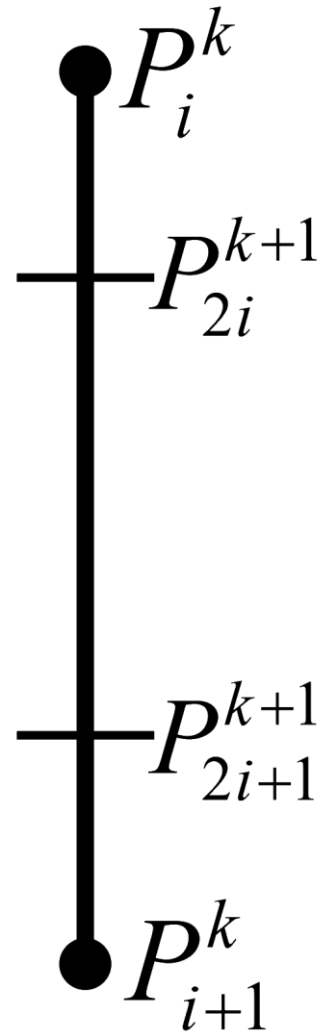
Chaikin's Corner Cutting

Chaikin can be written programmatically as

$$P_{2i}^{k+1} = (3/4)P_i^k + (1/4)P_{i+1}^k \quad \leftarrow \text{Even}$$

$$P_{2i+1}^{k+1} = (1/4)P_i^k + (3/4)P_{i+1}^k \quad \leftarrow \text{Odd}$$

- ...where k is the 'generation'; each generation will have twice as many control points as before
- Notice the different treatment of generating odd and even points
- Borders (terminal points) are a special case



Chaikin's Corner Cutting

Chaikin can be written in matrix/vector notation as:

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

Chaikin's Corner Cutting

The standard notation compresses the scheme to a *kernel*:

- $h = (1/4)[\dots, 0, 0, 1, 3, 3, 1, 0, 0, \dots]$

The kernel interlaces the odd and even rules

It also makes matrix analysis possible: eigen-analysis of the matrix form can be used to prove the continuity of the subdivision limit surface

The limit curve of Chaikin is a quadratic B-spline!

Cubic B-Spline Subdivision Scheme

Lane-Riesenfeld subdivision

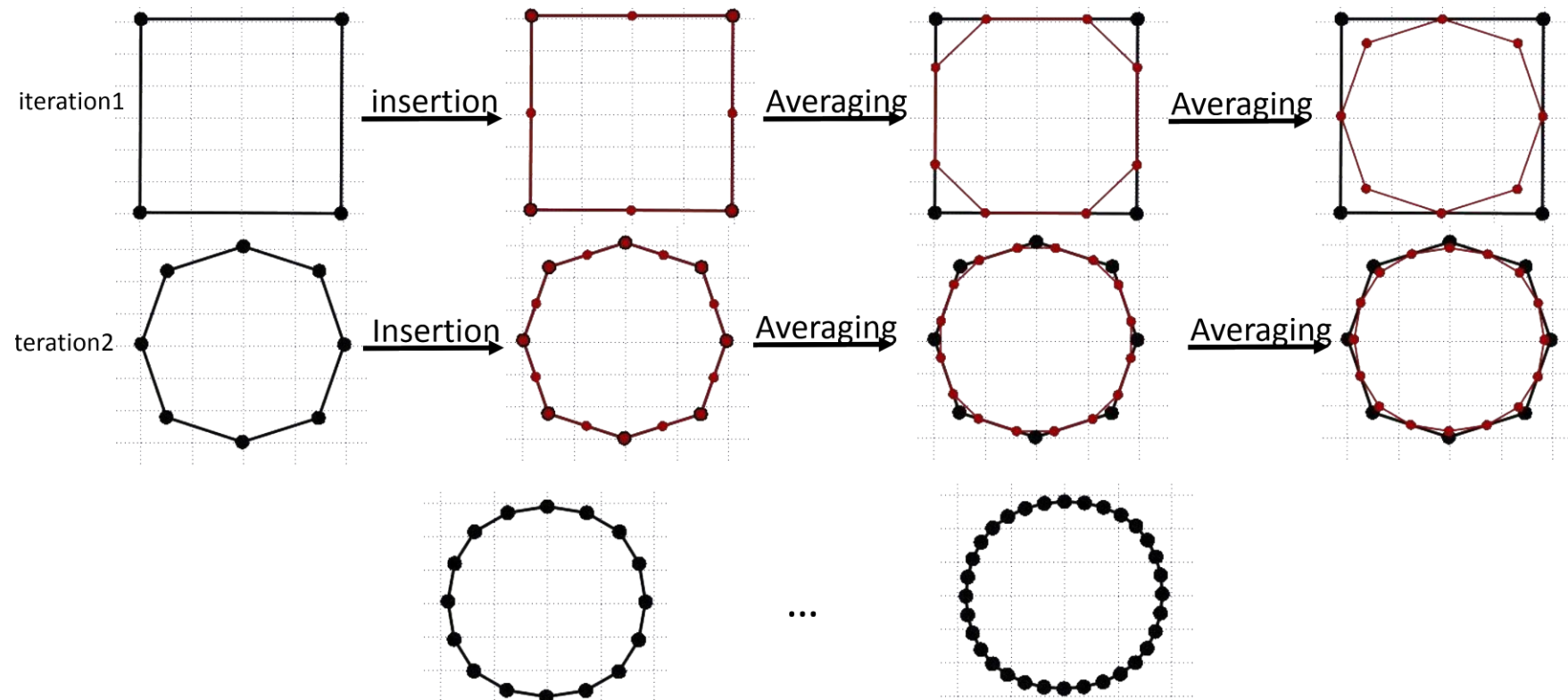
Algorithm:

- Linearly subdivide the curve by inserting the midpoint on each edge
 - Perform Averaging by replacing each edge by its midpoint d times
-
- Let's examine the case of $d = 2$

Lane-Riesenfeld subdivision

Examples:

- Closed curve



Lane-Riesenfeld subdivision

Close examination

- Step by step

$$\begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} = \frac{1}{8} \begin{pmatrix} 4 & 4 & 0 \\ 1 & 6 & 1 \\ 0 & 4 & 4 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \end{pmatrix}$$

$$a_1 = \frac{A + B}{2}$$

$$c_1 = \frac{B + C}{2}$$

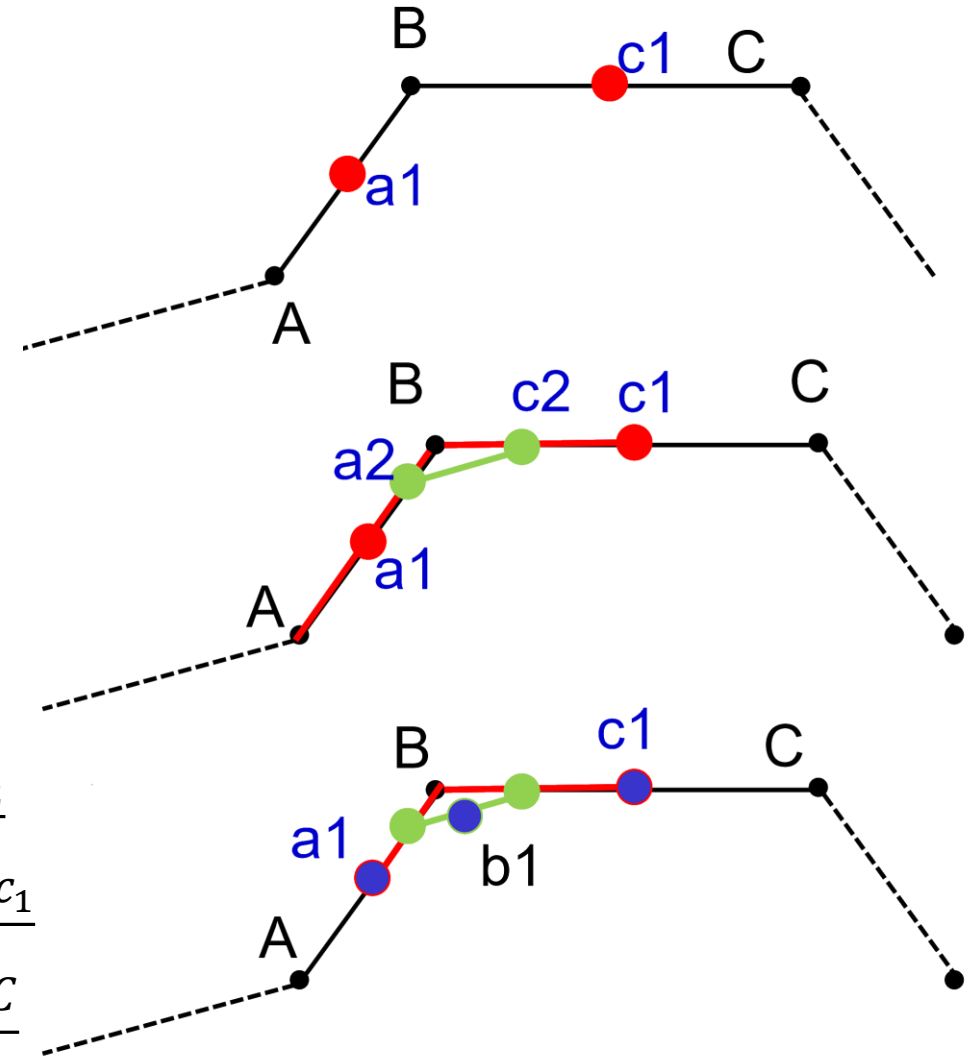
$$a_2 = \frac{B + a_1}{2}$$

$$c_2 = \frac{B + c_1}{2}$$

$$b_1 = \frac{a_2 + c_2}{2}$$

$$= \frac{a_1 + 2B + c_1}{4}$$

$$= \frac{A + 6B + C}{8}$$

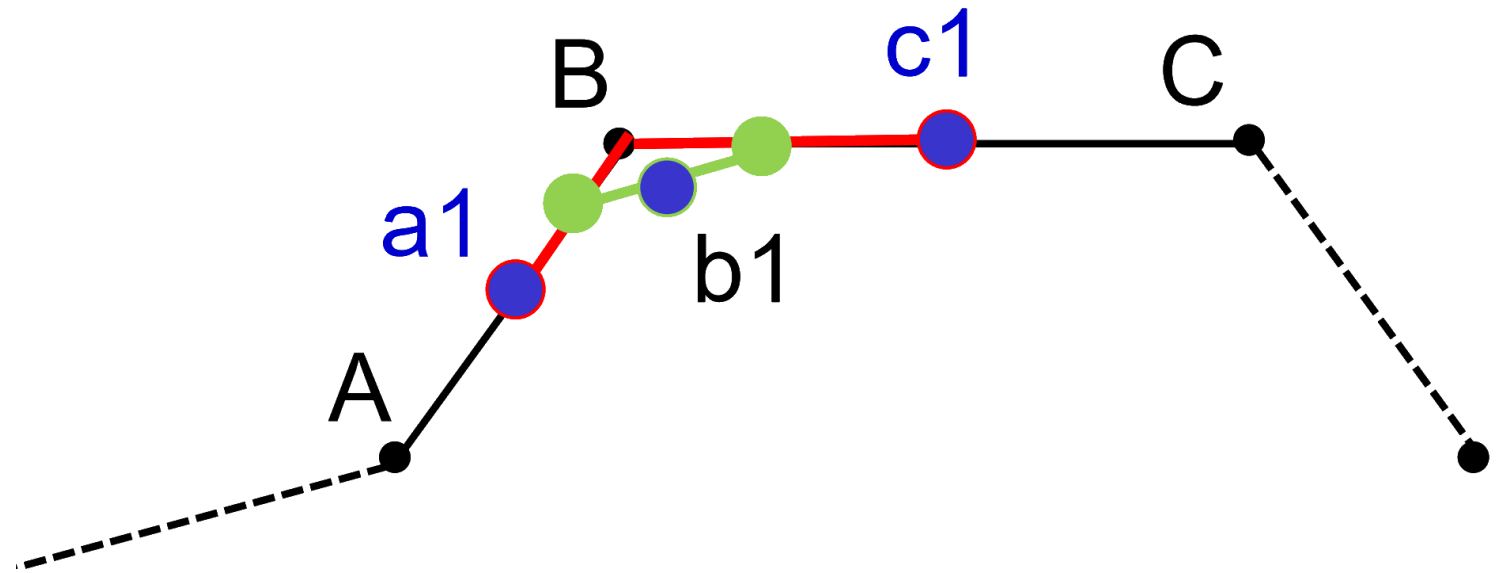


Lane-Riesenfeld subdivision

Close examination:

- In matrix form

$$\begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} = \frac{1}{8} \begin{pmatrix} 4 & 4 & 0 \\ 1 & 6 & 1 \\ 0 & 4 & 4 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \end{pmatrix}$$



Separate Splitting Step

Using a separate splitting matrix

$$\begin{pmatrix} \vdots \\ \mathbf{p}_{2i}^{(l+1)} \\ \mathbf{p}_{2i+1}^{(l+1)} \\ \vdots \end{pmatrix} = \underbrace{\begin{pmatrix} \ddots & & & & \\ \frac{1}{4} & & & & \\ & \frac{1}{2} & & & \\ & & \frac{1}{4} & & \\ & & & \frac{1}{2} & \\ & & & & \frac{1}{4} & \\ & & & & & \frac{1}{2} & \\ & & & & & & \frac{1}{4} & \\ & & & & & & & \ddots \end{pmatrix}}_{2n \times 2n \text{ averaging}} \underbrace{\begin{pmatrix} \ddots & & & \\ & 1 & & \\ & \frac{1}{2} & & \\ & & \frac{1}{2} & \\ & & & 1 & \\ & & & & \frac{1}{2} & \\ & & & & & \frac{1}{2} & \\ & & & & & & \ddots \end{pmatrix}}_{2n \times n \text{ splitting}} \begin{pmatrix} \vdots \\ \mathbf{p}_i^{(l)} \\ \mathbf{p}_{i+1}^{(l)} \\ \vdots \end{pmatrix}$$

$$\begin{pmatrix} \vdots \\ \mathbf{p}_{2i}^{(l+1)} \\ \mathbf{p}_{2i+1}^{(l+1)} \\ \vdots \end{pmatrix} = \underbrace{\begin{pmatrix} \ddots & & & & \\ & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & \\ & & \frac{1}{2} & \frac{1}{2} & \\ & & & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ & & & & \frac{1}{2} & \frac{1}{2} \\ & & & & & \ddots \end{pmatrix}}_{\text{One step}} \begin{pmatrix} \vdots \\ \mathbf{p}_i^{(l)} \\ \mathbf{p}_{i+1}^{(l)} \\ \vdots \end{pmatrix}$$

$$\mathbf{p}_{2i}^{[l+1]} = \frac{1}{4}\mathbf{p}_i^{[l]} + \frac{1}{2}\left(\frac{1}{2}\mathbf{p}_i^{[l]} + \frac{1}{2}\mathbf{p}_{i+1}^{[l]}\right) + \frac{1}{4}\mathbf{p}_{i+1}^{[l]} = \frac{1}{2}\mathbf{p}_i^{[l]} + \frac{1}{2}\mathbf{p}_{i+1}^{[l]}$$

$$\mathbf{p}_{2i+1}^{[l+1]} = \frac{1}{4}\left(\frac{1}{2}\mathbf{p}_i^{[l]} + \frac{1}{2}\mathbf{p}_{i+1}^{[l]}\right) + \frac{1}{2}\mathbf{p}_{i+1}^{[l]} + \frac{1}{4}\left(\frac{1}{2}\mathbf{p}_{i+1}^{[l]} + \frac{1}{2}\mathbf{p}_{i+2}^{[l]}\right) = \frac{1}{8}\mathbf{p}_i^{[l]} + \frac{6}{8}\mathbf{p}_{i+1}^{[l]} + \frac{1}{8}\mathbf{p}_{i+2}^{[l]}$$

Separate Splitting Step

Using a separate splitting matrix

$$\begin{pmatrix} \vdots \\ \mathbf{p}_{2i}^{(l+1)} \\ \mathbf{p}_{2i+1}^{(l+1)} \\ \vdots \end{pmatrix} = \underbrace{\begin{pmatrix} \ddots & & & & \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & & \\ & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & \\ & & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ & & & \frac{1}{4} & \frac{1}{2} \\ & & & & \frac{1}{4} \\ & & & & & \ddots \end{pmatrix}}_{2n \times 2n \text{ averaging}} \underbrace{\begin{pmatrix} \ddots & & & & \\ & \frac{1}{2} & & & \\ & \frac{1}{2} & \frac{1}{2} & & \\ & & \frac{1}{2} & \frac{1}{2} & \\ & & & \frac{1}{2} & \frac{1}{2} \\ & & & & \ddots \end{pmatrix}}_{2n \times n \text{ splitting}} \begin{pmatrix} \vdots \\ \mathbf{p}_i^{(l)} \\ \mathbf{p}_{i+1}^{(l)} \\ \vdots \end{pmatrix}$$

Cubic Subdivision

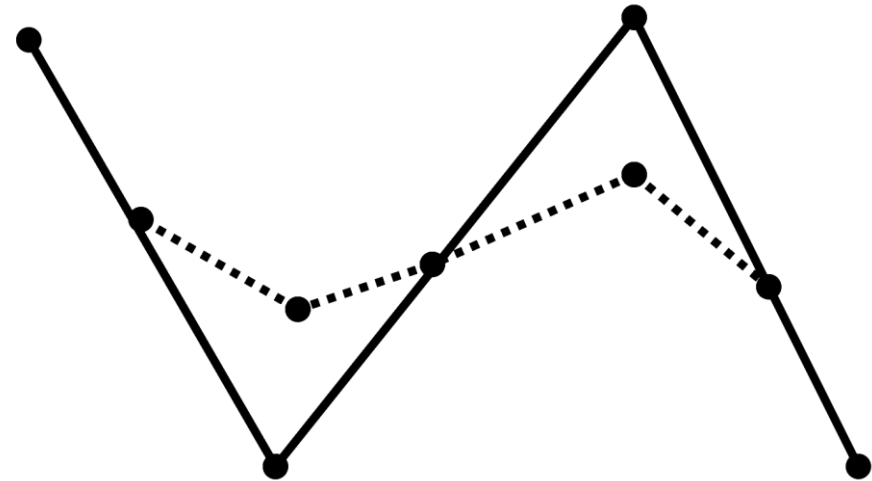
Consider the Kernel

- $h = \left(\frac{1}{8}\right) [\dots, 0, 0, 1, 4, 6, 4, 1, 0, 0, \dots]$

You would read this as

- $P_{2i}^{k+1} = (1/8)(P_{i-1}^k + 6P_i^k + P_{i+1}^k)$
- $P_{2i+1}^{k+1} = (1/8)(4P_i^k + 4P_{i+1}^k)$

The limit curve is provably C^2 continuous



General Formula:

B-spline curve subdivision:

- Splitting step as usual (insert midpoints on lines)
- Averaging mask is stationary (constant everywhere):

$$\frac{1}{2^{d-1}} \left(\binom{d-1}{0}, \binom{d-1}{1}, \dots, \binom{d-1}{d-1} \right)$$

for B-splines of degree d

Approximating the curve

- Infinite subdivision will create a dense point set that converges to the curve

Spectral Convergence Analysis

of the cubic B-Spline Subdivision Scheme

The Spectral Limit Trick

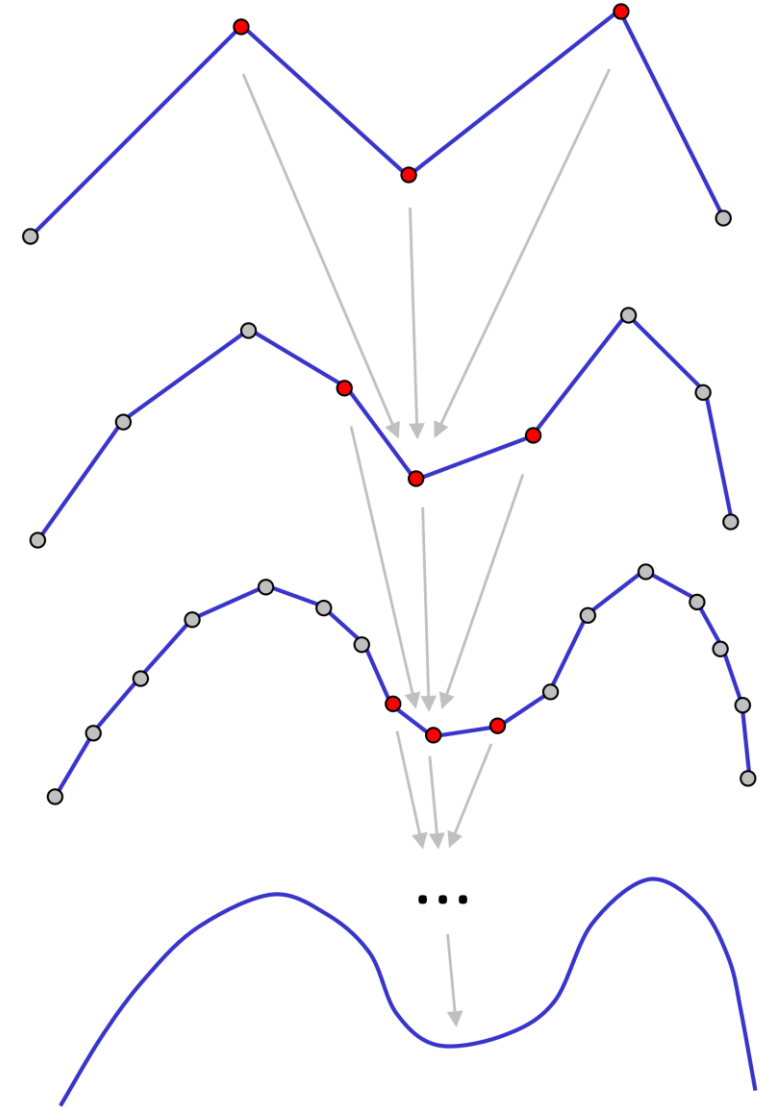
Problem:

- We need to subdivide several times to obtain a good approximation
- This might yield more control points than necessary
(think of adaptive rendering with low level of detail)
- Can we directly compute the limit position for a control points?

Computing the Limit

Observations:

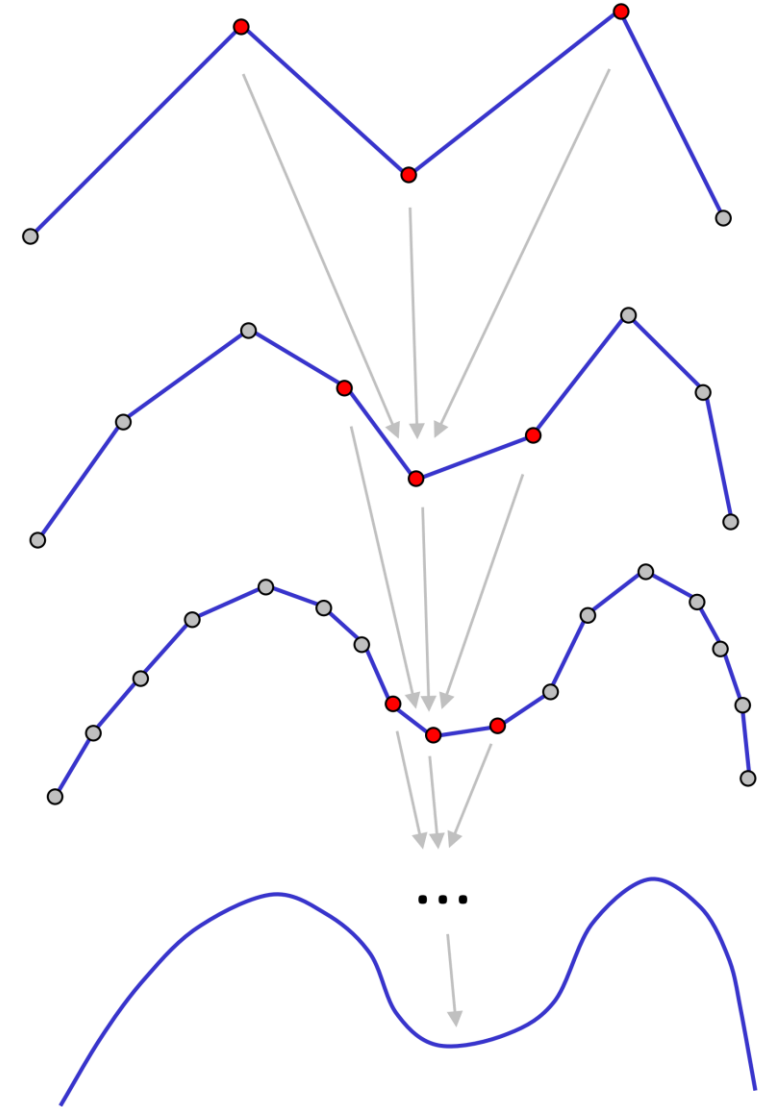
- Every curve point is influenced only by a fixed number of control points
- Even stronger : Every point $p^{[l+1]}$ is only influenced by a small neighborhood of points in $p^{[l]}$
- To each neighborhood, the same subdivision matrix is applied (splitting & averaging)



The Local Subdivision Matrix

Invariant Neighborhood

- Example: Cubic B-splines
 - A single point lies in one of two adjacent spline segments
 - So at most 5 control points are influencing each point on the curve
 - A closer look at the subdivision rule reveals that limit properties can actually be computed from 3 points (two direct neighbors)



Local Subdivision Matrix

Local subdivision matrix:

- Transforms a neighborhood of points

Example: cubic B-spline

- Only the two direct neighbors influence the point in the next level
- The local subdivision matrix is

x_- = left neighbor

x = point ($x/y/z$ -coordinate)

x_+ = right neighbor

$$\begin{pmatrix} x_-^{[l+1]} \\ x^{[l+1]} \\ x_+^{[l+1]} \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}}_{= M_{subdiv}} \begin{pmatrix} x_-^{[l]} \\ x^{[l]} \\ x_+^{[l]} \end{pmatrix}$$

To the Limit...

This means:

- At any recursion depth of the subdivision, we can send a point to the limit by evaluating:

$$\begin{pmatrix} x_-^{[\infty]} \\ x^{[\infty]} \\ x_+^{[\infty]} \end{pmatrix} = \lim_{k \rightarrow \infty} \mathbf{M}_{subdiv}^k \begin{pmatrix} x_-^{[l]} \\ x^{[l]} \\ x_+^{[l]} \end{pmatrix} = \lim_{k \rightarrow \infty} \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}^k \begin{pmatrix} x_-^{[l]} \\ x^{[l]} \\ x_+^{[l]} \end{pmatrix}$$

To the Limit...

Spectral power:

- Assuming the matrix \mathbf{M}_{subdiv} is diagonalizable, we get:

$$\begin{aligned} \begin{pmatrix} x_-^{[\infty]} \\ x^{[\infty]} \\ x_+^{[\infty]} \end{pmatrix} &= \lim_{k \rightarrow \infty} \mathbf{U} \mathbf{D}^k \mathbf{U}^{-1} \begin{pmatrix} x_-^{[l]} \\ x^{[l]} \\ x_+^{[l]} \end{pmatrix} = \mathbf{U} \left(\lim_{k \rightarrow \infty} \mathbf{D}^k \right) \mathbf{U}^{-1} \begin{pmatrix} x_-^{[l]} \\ x^{[l]} \\ x_+^{[l]} \end{pmatrix} \\ &= \begin{pmatrix} 1 & -1 & -2 \\ 1 & 0 & 1 \\ 1 & 1 & -2 \end{pmatrix} \lim_{k \rightarrow \infty} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{4} \end{pmatrix}^k \begin{pmatrix} \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ -\frac{1}{2} & 0 & \frac{1}{2} \\ -\frac{1}{6} & \frac{1}{3} & -\frac{1}{6} \end{pmatrix} \begin{pmatrix} x_-^{[l]} \\ x^{[l]} \\ x_+^{[l]} \end{pmatrix} \end{aligned}$$

To the Limit...

Spectral power:

- For cubic B-splines:

$$\begin{pmatrix} x_{-}^{[\infty]} \\ x^{[\infty]} \\ x_{+}^{[\infty]} \end{pmatrix} = \begin{pmatrix} 1 & -1 & -2 \\ 1 & 0 & 1 \\ 1 & 1 & -2 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ -\frac{1}{2} & 0 & \frac{1}{2} \\ -\frac{1}{6} & \frac{1}{3} & -\frac{1}{6} \end{pmatrix} \begin{pmatrix} x_{-}^{[1]} \\ x^{[1]} \\ x_{+}^{[1]} \end{pmatrix} = \begin{pmatrix} \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{pmatrix} \begin{pmatrix} x_{-}^{[1]} \\ x^{[1]} \\ x_{+}^{[1]} \end{pmatrix}$$

- and hence

$$x^{[\infty]} = \begin{bmatrix} \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{bmatrix} \begin{pmatrix} x_{-}^{[1]} \\ x^{[1]} \\ x_{+}^{[1]} \end{pmatrix}$$

To the Limit, in General

- **In general:**
 - The dominant eigenvalue / eigenvector of the subdivision scheme determines the limit mask

Necessary Condition

Necessary condition for convergence:

- 1 must be the largest eigenvalue (in absolute value)
- Otherwise the subdivision either explodes (>1) or shrinks to the origin (<1)

$$\begin{pmatrix} x_{-n}^{[l+k]} \\ \vdots \\ x_0^{[l+k]} \\ \vdots \\ x_{+n}^{[l+k]} \end{pmatrix} = \mathbf{M}_{subdiv}^k \begin{pmatrix} x_{-n}^{[l]} \\ \vdots \\ x_0^{[l]} \\ \vdots \\ x_{+n}^{[l]} \end{pmatrix} = \mathbf{U} \mathbf{D}^k \mathbf{U}^{-1} \begin{pmatrix} x_{-n}^{[l]} \\ \vdots \\ x_0^{[l]} \\ \vdots \\ x_{+n}^{[l]} \end{pmatrix}$$

Affine Invariance

Affine Invariance

- The limit curve should be independent of the choice of a coordinate system
- We get this, if the intermediate subdivision points are affine invariant
- For this, the rows of the (local) subdivision matrix must sum to one:

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Affine Invariance

Affine Invariance

- For this, the rows of the (local) subdivision matrix must sum to one:

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

- This means: The one-vector **1** must be an eigenvector with eigenvalue 1:
 - $M_{subdiv} \mathbf{1} = \mathbf{1}$
 - This must also be the largest eigenvalue / vector pair
 - One can show: it must be the only eigenvector with eigenvalue 1, otherwise the scheme does not converge

Summary

For a reasonable subdivision scheme, we need at least:

- **1** must be an eigenvector with eigenvalue 1.
- This must be the largest eigenvalue.
- The second eigenvalue should be smaller than 1
- All other eigenvalues should be smaller than the second one

(This is assuming a diagonalizable subdivision matrix.)

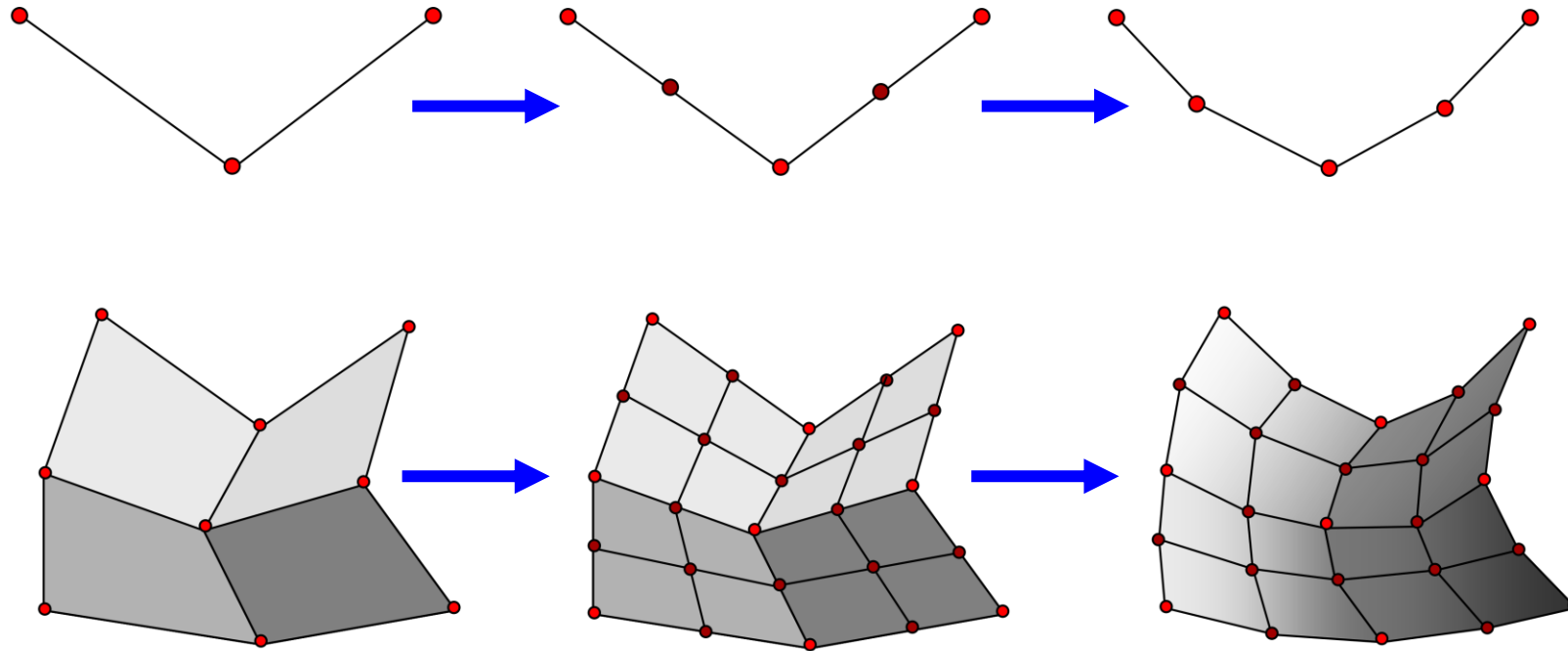
More details: Zorin, Schroder – Subdivision for Modeling and Animation, Siggraph 2000 course

B-Spline Subdivision Surfaces

B-Spline Subdivision Surfaces

B-Spline Subdivision Surfaces

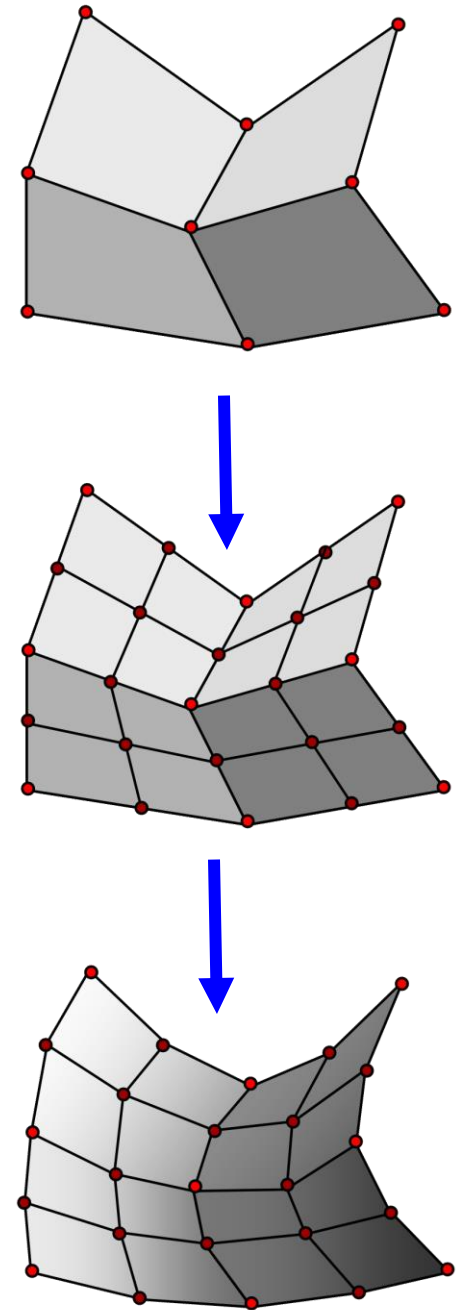
- We can apply the tensor product construction to obtain subdivision surfaces



B-Spline Subdivision Surfaces

Tensor Product B-Spline Subdivision Surfaces

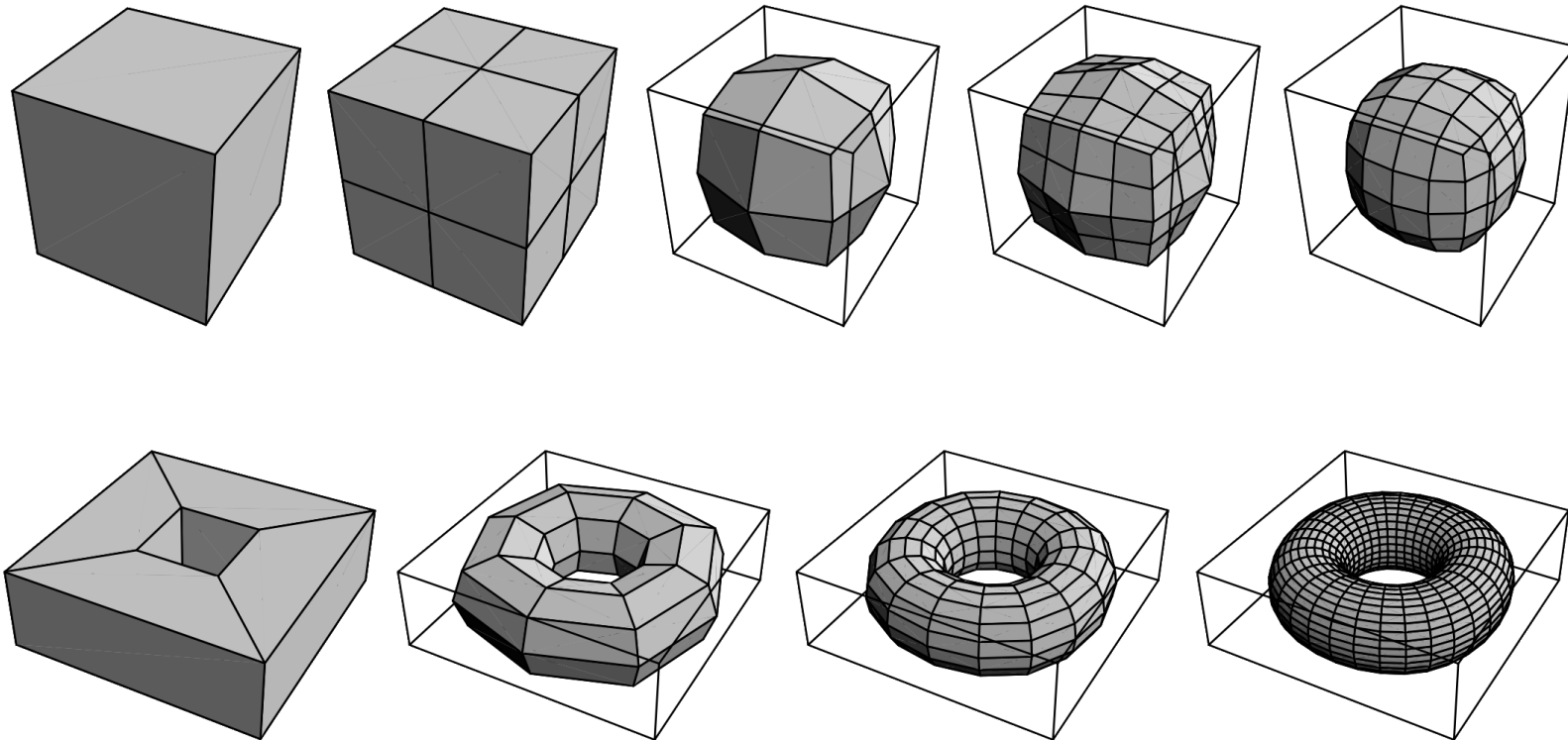
- Start with a regular quad mesh (will be relaxed later)
- In each subdivision step:
 - Divide each quad in four (quadtree subdivision)
 - Place linearly interpolated vertices
 - Apply 2-dimensional averaging mask



B-Spline Subdivision Surfaces

Bilinear Subdivision Surfaces + quad averaging:

- Quad averaging : reposition each vertex at the centroid of its adjacent quads



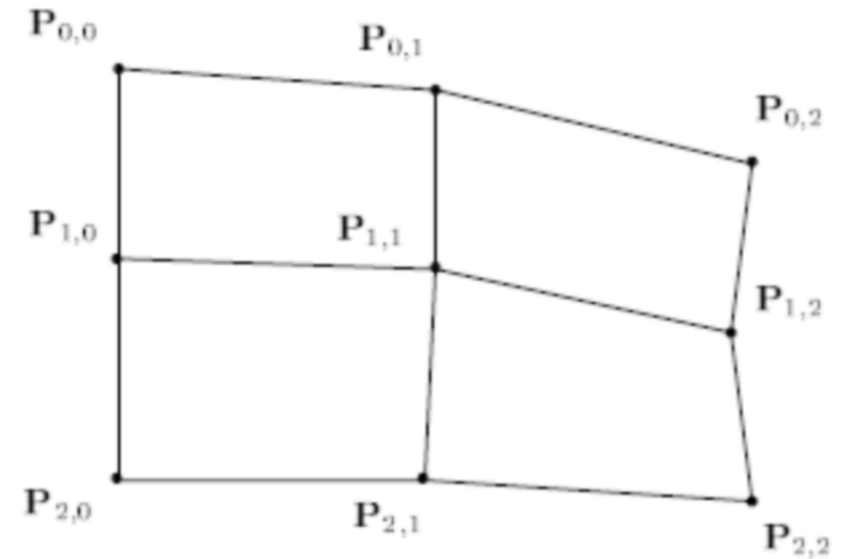
B-Spline Subdivision Surfaces

Biquadratic case:

- Recall the matrix B-spline patch representation

$$P(u, v) = [1 \quad u \quad u^2] M P M^T \begin{bmatrix} 1 \\ v \\ v^2 \end{bmatrix}$$

$$M = \frac{1}{2} \begin{bmatrix} 1 & 1 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{bmatrix}, \quad P = \begin{bmatrix} P_{0,0} & P_{0,1} & P_{0,2} \\ P_{1,0} & P_{1,1} & P_{1,2} \\ P_{2,0} & P_{2,1} & P_{2,2} \end{bmatrix}$$



B-Spline Subdivision Surfaces

Biquadratic case:

- By restricting to only one quadrant of the 2×2 patch, i.e. $u, v \in [0, \frac{1}{2}]$. We consider the new surface patch P' defined by re-parameterization $u' = \frac{u}{2}, v' = \frac{v}{2}$

$$\begin{aligned} P'(u, v) &= P\left(\frac{u}{2}, \frac{v}{2}\right) = \begin{bmatrix} 1 & u/2 & u^2/4 \end{bmatrix} M P M^T \begin{bmatrix} 1 \\ v/2 \\ v^2/4 \end{bmatrix} \\ &= \cdots = \begin{bmatrix} 1 & u & u^2 \end{bmatrix} M P' M^T \begin{bmatrix} 1 \\ v \\ v^2 \end{bmatrix} \end{aligned}$$

B-Spline Subdivision Surfaces

Biquadratic case:

- By restricting to only one quadrant of the 2×2 patch, i.e. $u, v \in [0, \frac{1}{2}]$. We consider the new surface patch P' defined by re-parameterization $u' = \frac{u}{2}, v' = \frac{v}{2}$

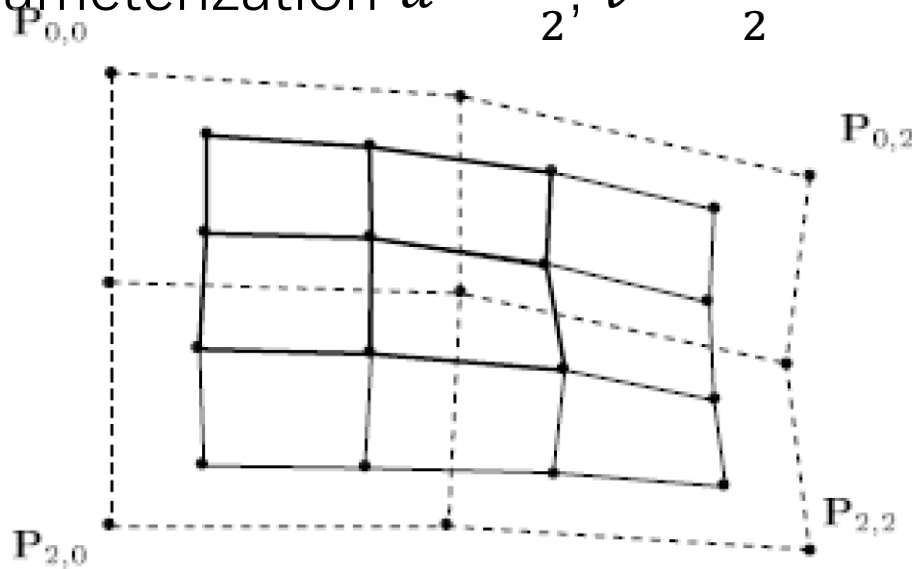
$$P' = SPST^T$$

$$S = M^{-1} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{4} \end{bmatrix} M$$

B-Spline Subdivision Surfaces

Biquadratic case:

- By restricting to only one quadrant of the 2×2 patch, i.e. $u, v \in [0, \frac{1}{2}]$. We consider the new surface patch P' defined by re-parameterization $u' = \frac{u}{2}, v' = \frac{v}{2}$



$$\begin{aligned}
 P'_{00} &= \frac{1}{16} (9P_{00} + 3P_{10} + 3P_{01} + P_{11}) \\
 P'_{01} &= \frac{1}{16} (3P_{00} + P_{10} + 9P_{01} + 3P_{11}) \\
 P'_{02} &= \frac{1}{16} (9P_{01} + 3P_{11} + 3P_{02} + 2P_{12}) \\
 P'_{11} &= \frac{1}{16} (3P_{00} + 9P_{10} + P_{01} + 3P_{11}) \\
 P'_{11} &= \frac{1}{16} (P_{00} + 3P_{10} + 3P_{01} + 9P_{11}) \\
 P'_{12} &= \frac{1}{16} (3P_{01} + 9P_{11} + P_{02} + 3P_{12}) \\
 P'_{20} &= \frac{1}{16} (9P_{10} + 3P_{20} + 3P_{11} + P_{21}) \\
 P'_{21} &= \frac{1}{16} (3P_{10} + P_{20} + 9P_{11} + 3P_{21}) \\
 P'_{22} &= \frac{1}{16} (9P_{11} + 3P_{21} + 3P_{12} + P_{22})
 \end{aligned}$$

B-Spline Subdivision Surfaces

Bicubic case:

- Recall the matrix B-spline patch representation

$$P(u, v) = [u^3 \quad u^2 \quad u \quad 1] M P M^T \begin{bmatrix} w^3 \\ w^2 \\ w \\ 1 \end{bmatrix}$$

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

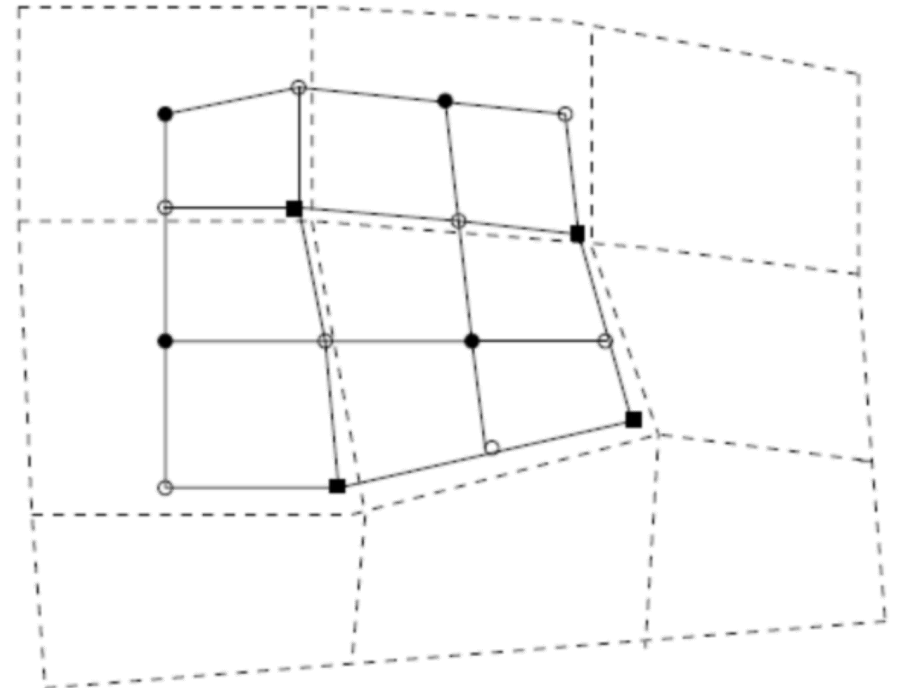
B-Spline Subdivision Surfaces

Bicubic case:

- By restricting to only one quadrant of the 3×3 patch, i.e. $u, v \in [0, \frac{1}{2}]$. We consider the new surface patch P' defined by re-parameterization $u' = \frac{u}{2}$, $v' = \frac{v}{2}$
- We obtain similarly (by matrix manipulation)

$$P' = SPST^T$$

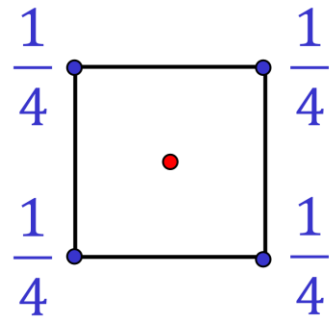
$$S = \frac{1}{8} \begin{bmatrix} 4 & 4 & 0 & 0 \\ 1 & 6 & 1 & 0 \\ 0 & 4 & 4 & 0 \\ 0 & 1 & 6 & 1 \end{bmatrix}$$



Subdivision and Averaging Masks

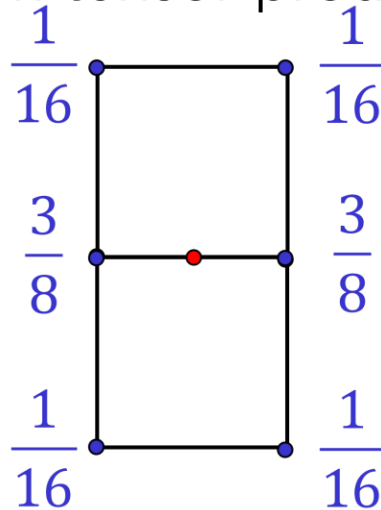
What is the subdivision mask?

- Can be derived from tensor product construction:



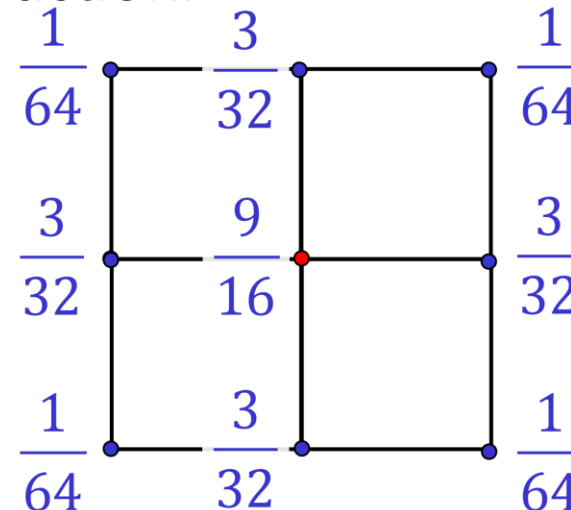
face midpoint
(odd/odd)

$$\begin{pmatrix} 1 \\ 2 \\ 1 \\ 2 \end{pmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}$$



edge midpoint
(even/odd)

$$\begin{pmatrix} 1 \\ 2 \\ 1 \\ 2 \end{pmatrix} \cdot \begin{bmatrix} 1 & 3 & 1 \\ 8 & 4 & 8 \end{bmatrix}$$



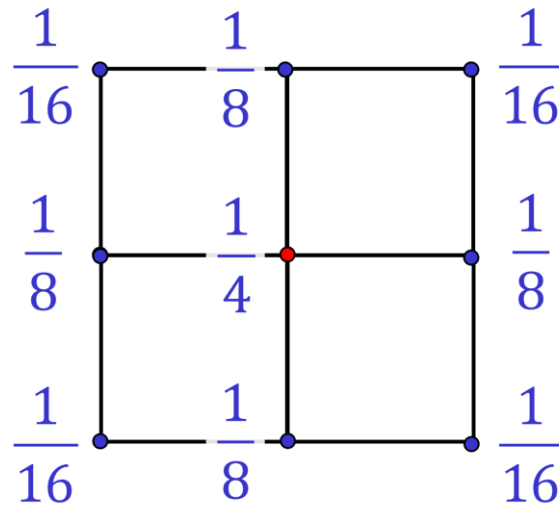
original vertex
(even/even)

$$\begin{pmatrix} 1 \\ 8 \\ 3 \\ 4 \\ 1 \\ 8 \end{pmatrix} \cdot \begin{bmatrix} 1 & 3 & 1 \\ 8 & 4 & 8 \end{bmatrix}$$

Subdivision and Averaging Masks

What is the averaging mask?

- Can be derived from tensor product construction, too



Any (split) vertex

$$\begin{pmatrix} \frac{1}{4} \\ \frac{1}{2} \\ \frac{1}{4} \end{pmatrix} \cdot \left[\frac{1}{4}, \frac{1}{2}, \frac{1}{4} \right]$$

Remaining Problems

Remaining Problems:

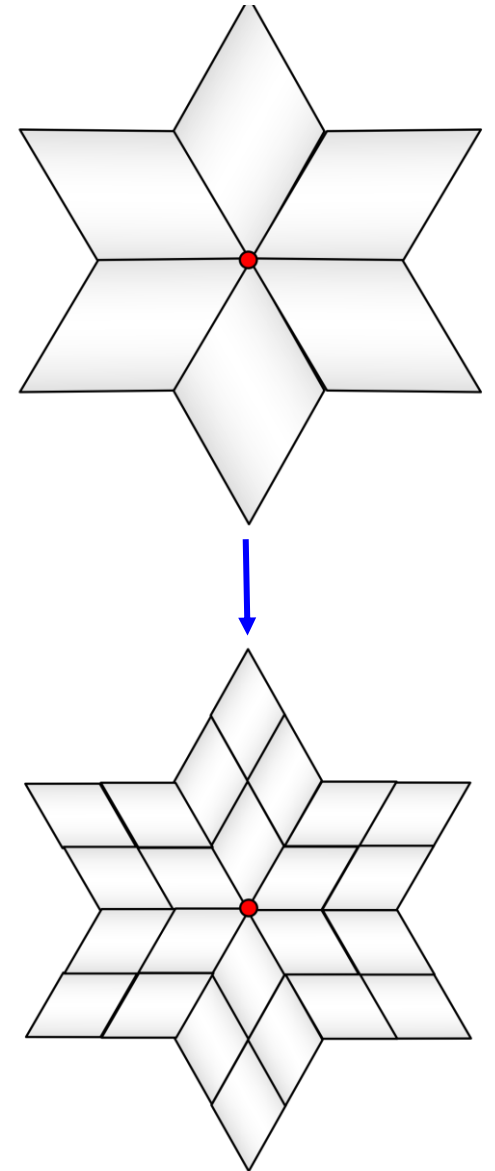
- The derived rules work only in the interior or a regular quad mesh
- We did not really gain any flexibility over the standard B-spline construction
- We still need to figure out, how to ...
 - ...handle quad meshes of arbitrary topology
 - ...handle boundary regions
 - Placing boundaries in the interior of objects will allow us to model sharp C^0 creases
 - So we also have some continuity control (despite the uniform B-Spline scheme)

Here is the answer...

Answer: Catmull-Clark subdivision scheme at extraordinary vertices

Observation:

- The recursive subdivision rule always creates regular grids
- Problems can only occur at “extraordinary” vertices
 - These are vertices where the base has degree > 4
 - Extraordinary vertices are maintained by quadtree-like-subdivision
 - All new vertices are ordinary

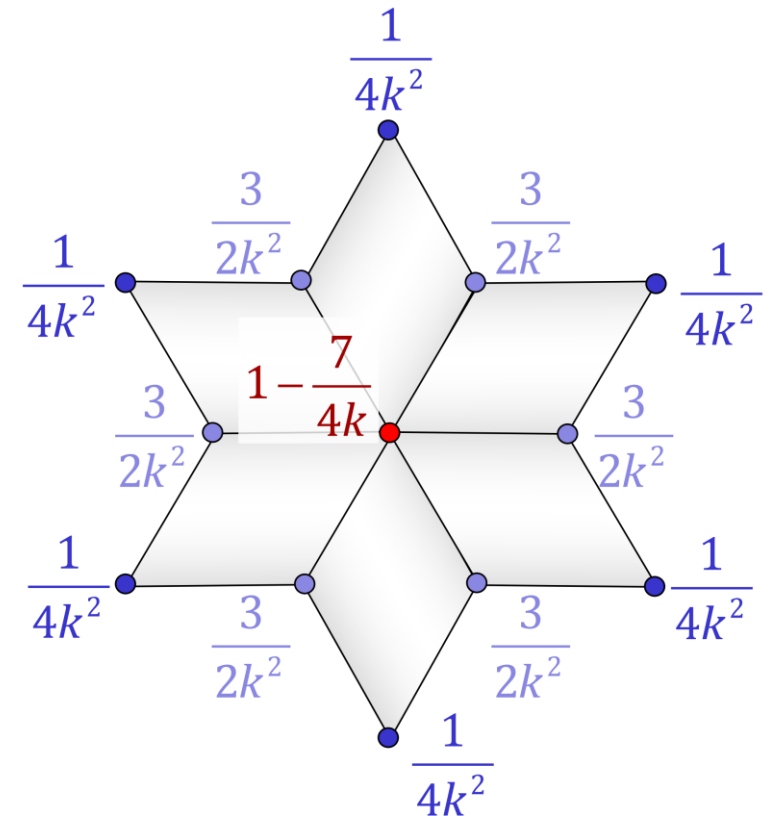


Here is the answer...

Answer: Catmull-Clark subdivision scheme at extraordinary vertices

Subdivision mask at extraordinary vertex:

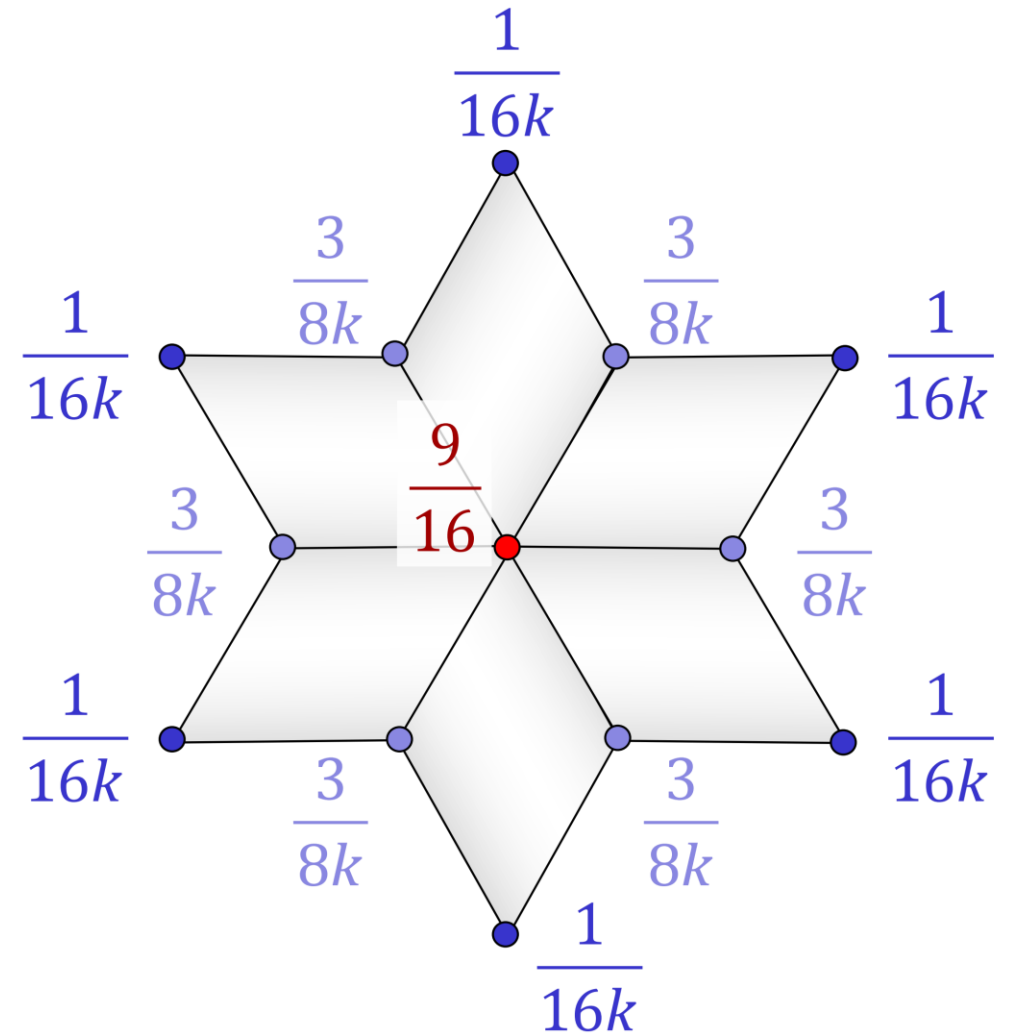
- Vertex degree k (number of incident faces)
- The surface is C^1 at extraordinary vertices



Here is the answer...

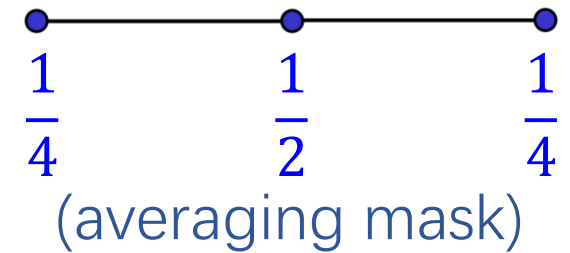
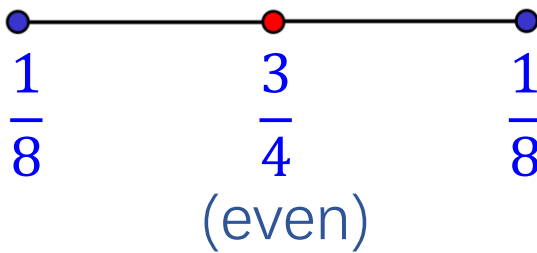
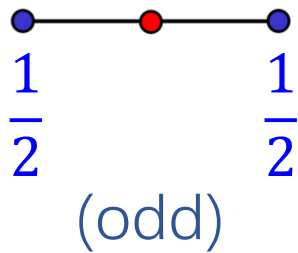
Averaging mask:

- Use after bilinear splitting



Boundary Rules

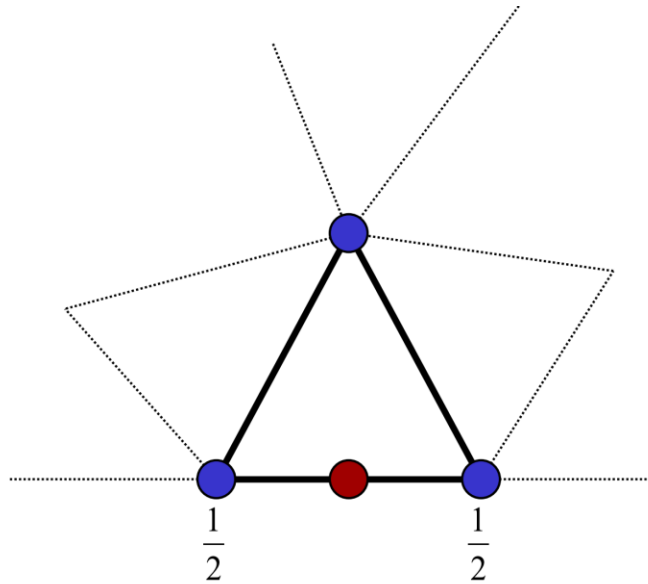
Subdivision mask at boundaries / sharp creases:



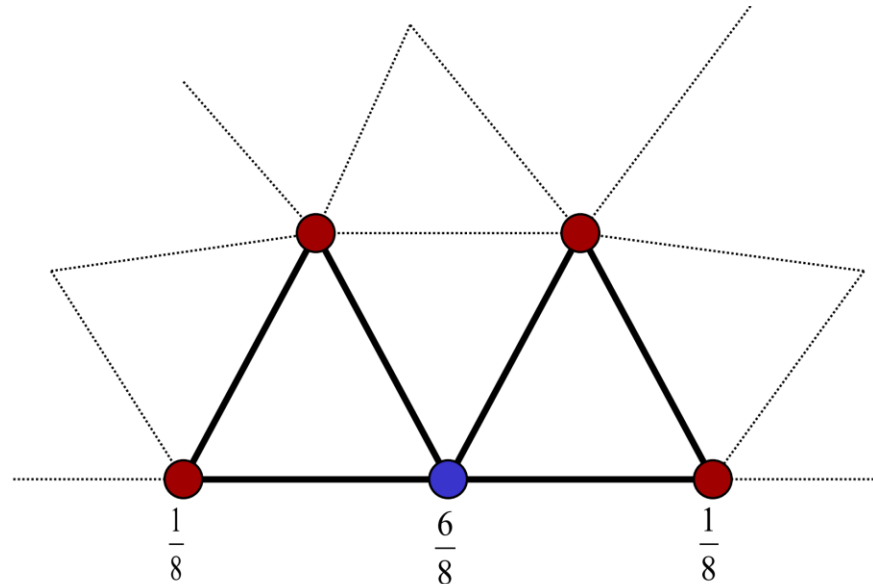
- Just use the normal spline curve rules
- This gives visually good results
- However, the surface is not strictly \mathcal{C}^1 at the boundary
- There is a modified weighting scheme that creates half-sided \mathcal{C}^1 -continuous surfaces at the boundary curves

Boundary Rules

Subdivision Mask for Boundary Conditions



↑
Edge Rule (odd)



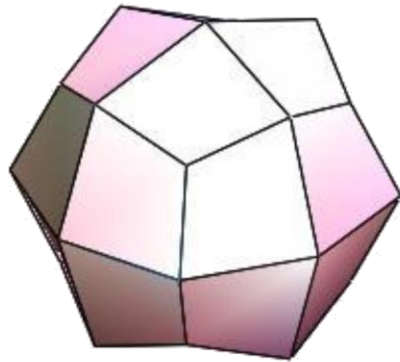
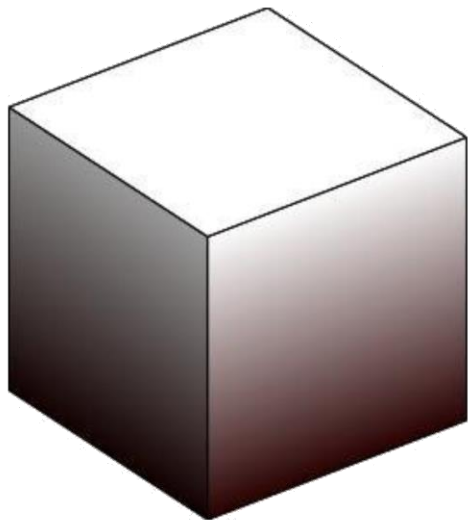
↑
Vertex Rule (even)

Catmull-Clark in short

Face, edge, vertex points:

1. Introduce a **face point** for each face of the original mesh. The point is simply the average of all the points that bound the face.
2. An **edge point** is created for each interior edge of the polygonal surface. The point is the average of the midpoint of the edge and the two face points on both sides of the edge
3. A **vertex point** is generated for each interior vertex P of the original mesh. The point is the average of Q , $2R$, and $\frac{(n-3)S}{n}$, where Q is the average of the face points on all the faces adjacent to P , R is the average of the midpoints of all the edges incident on P , and S is simply P itself

Catmull-Clark scheme



Other Subdivision Schemes

Loop, Butterfly, ...

Subdivision Zoo

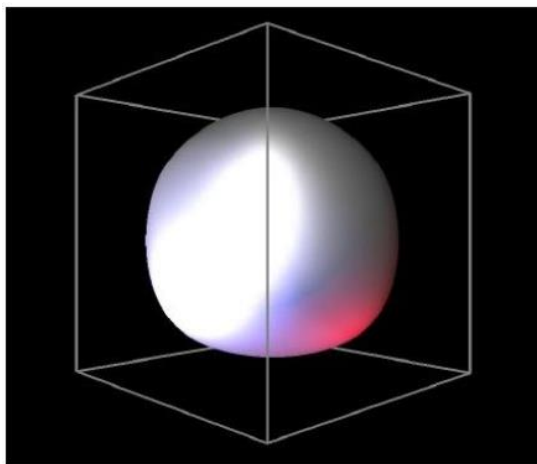
A large number of subdivision scheme exists. The most popular are:

- Catmull-Clark subdivision
(*quad-mesh, approximating, C^2 surfaces, C^1 at extraordinary vertices*)
- Loop subdivision
(*triangular, approximating, C^2 surfaces, C^1 at extraordinary vertices*)
- Butterfly subdivision
(*triangular, interpolation, C^1 surfaces, C^1 at extraordinary vertices*)

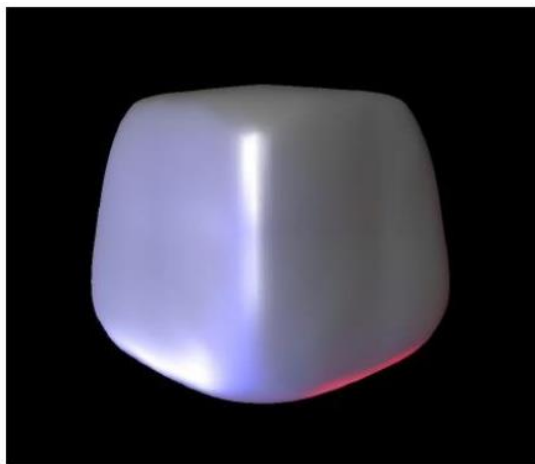
Examples of other schemes:

- $\sqrt{3}$ -subdivision (level of detail increases more slowly)
- Circular subdivision (used e.g. for surfaces of revolution)

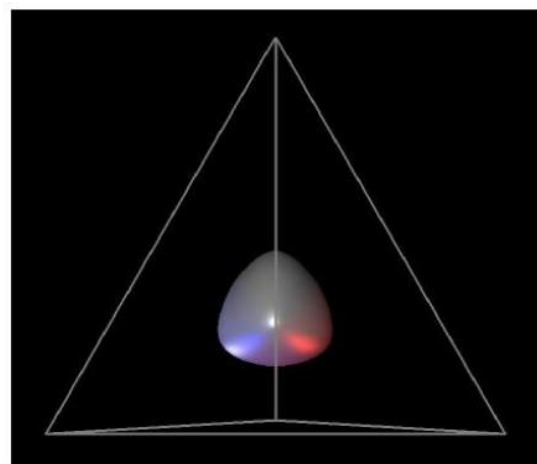
Comparisons



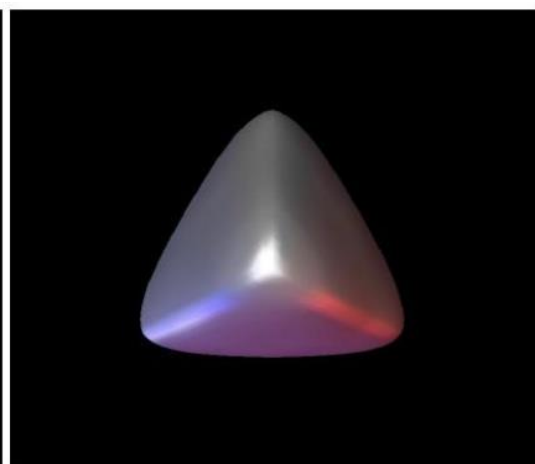
Loop



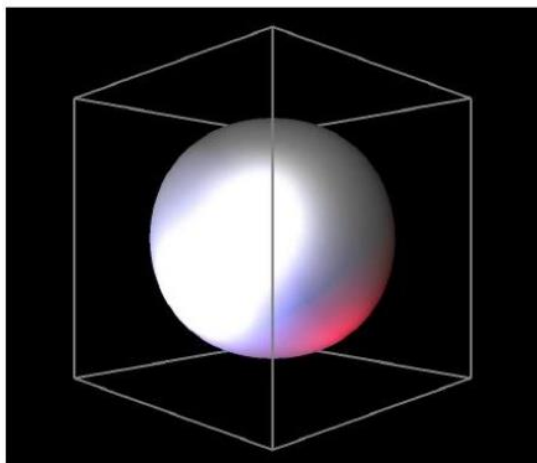
Butterfly



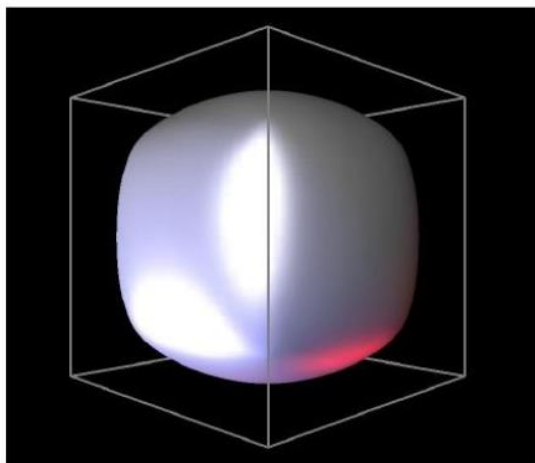
Loop



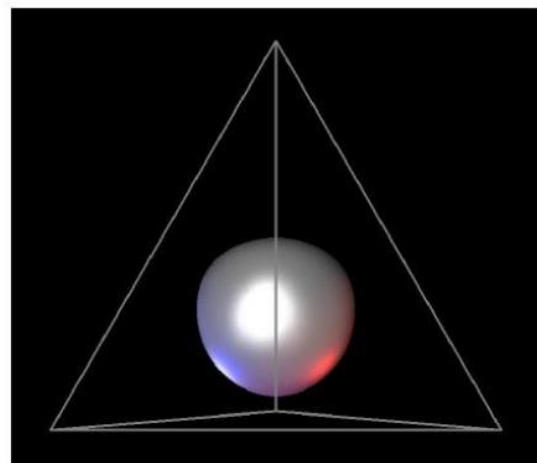
Butterfly



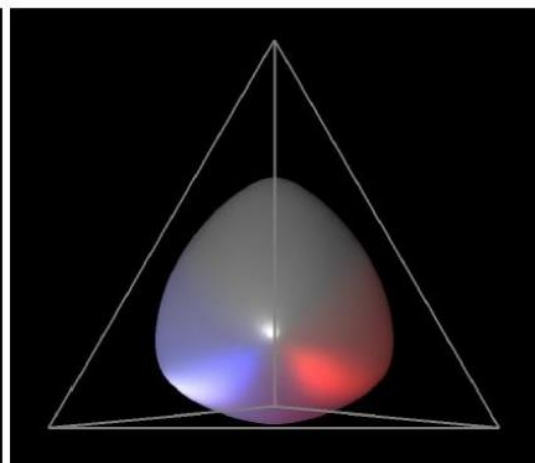
Catmull-Clark



Doo-Sabin



Catmull-Clark

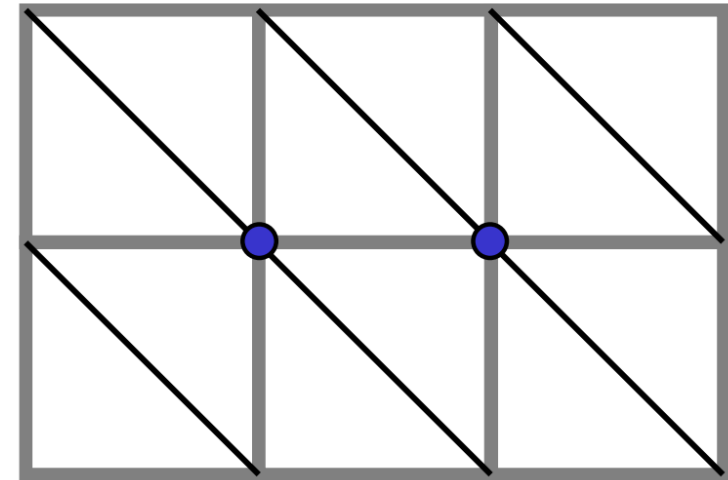


Doo-Sabin

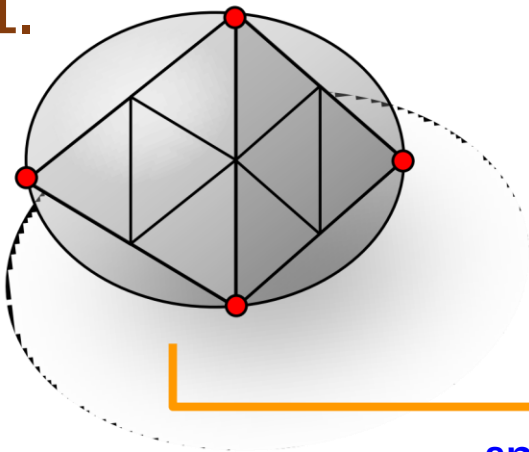
Triangular Subdivision

Triangular Subdivision:

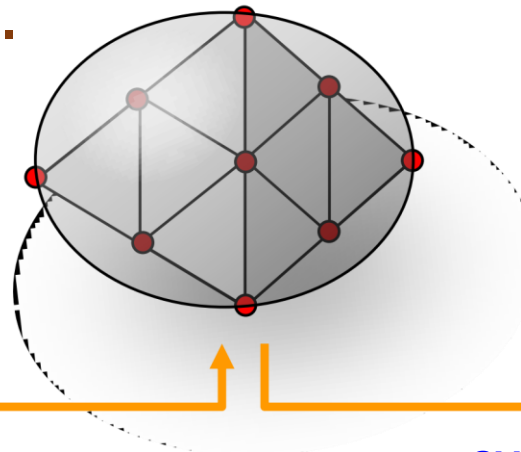
- Uses 1:4 triangular splits
 - Extraordinary vertices: valence $\neq 6$
- Again:
 - Splitting with linear interpolation
 - Then apply averaging mask



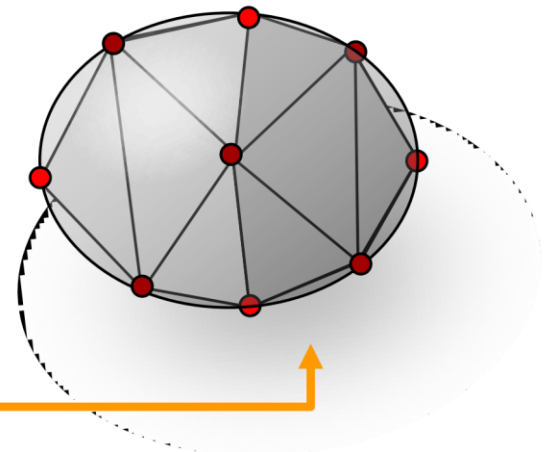
1.



2.



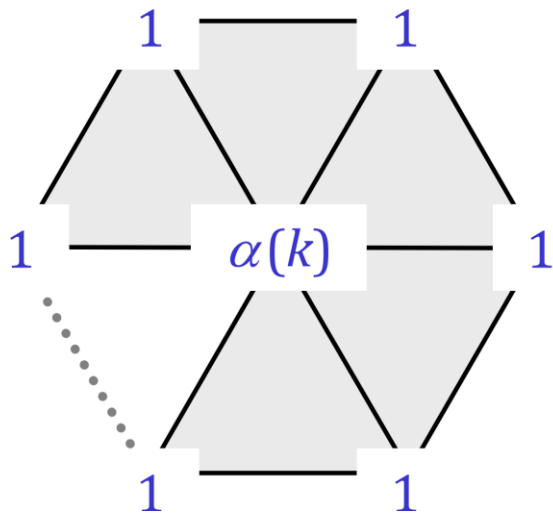
3.



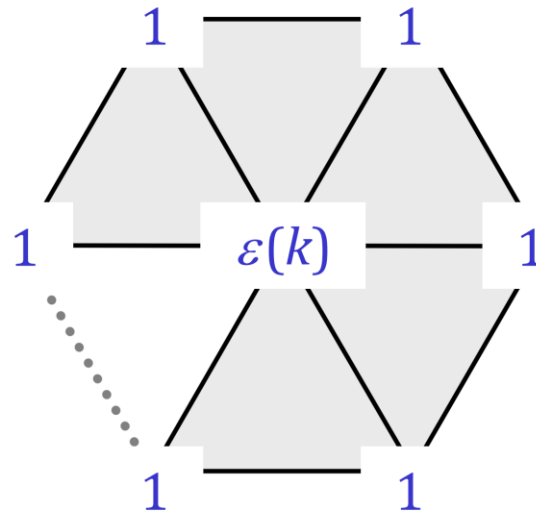
splitting

averaging

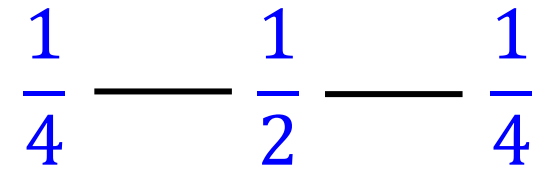
Loop Subdivision



averaging mask



evaluation (limit) mask



boundary/sharp
crease mask

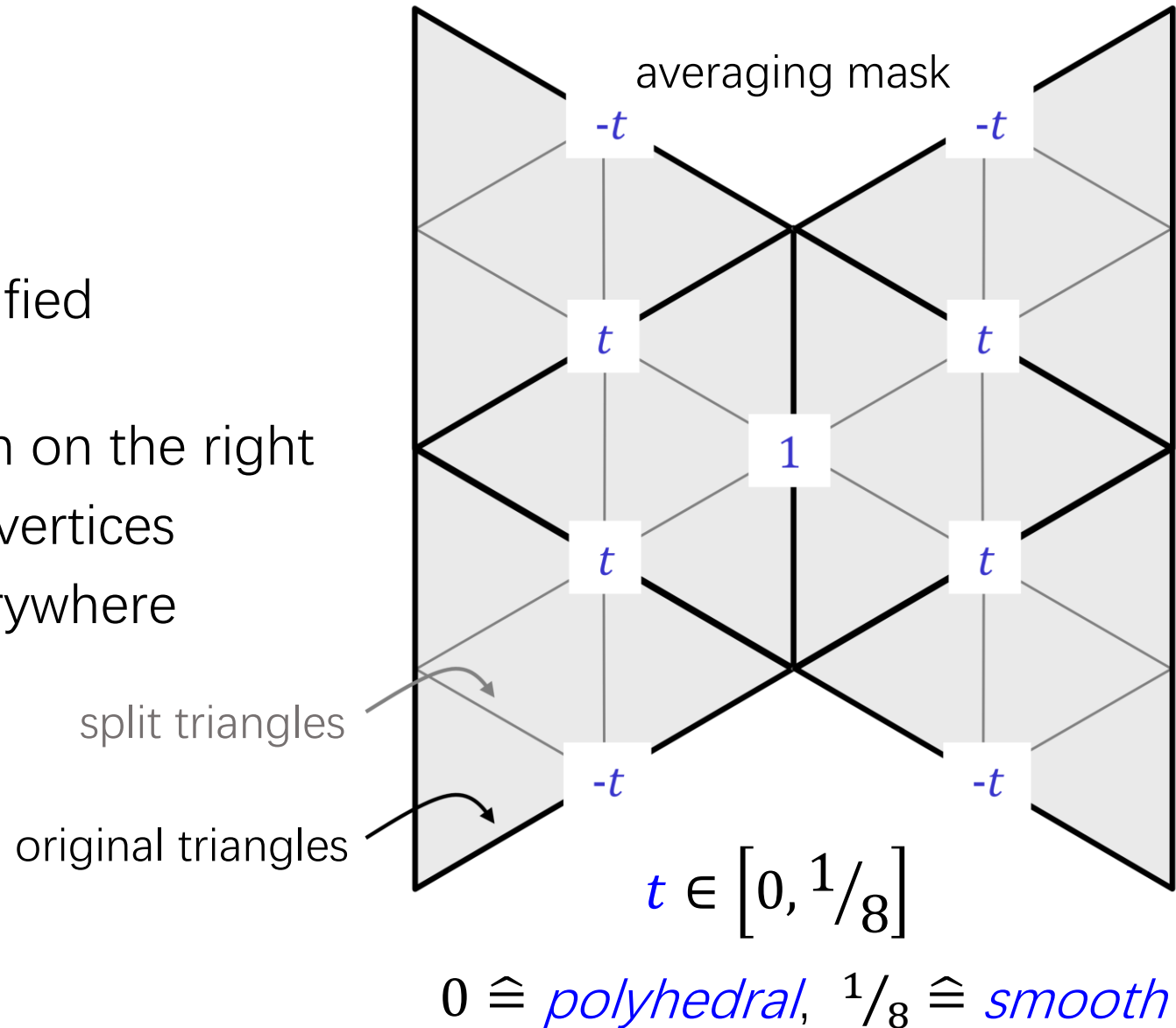
$$\alpha(k) = \frac{k(1 - \beta(k))}{\beta(k)} \quad \varepsilon(k) = \frac{3k}{(4\beta(k))}$$

$$\beta(k) = \frac{5}{4} - \frac{(3 + 2 \cos(2\pi/k))^2}{32}$$

Butterfly Scheme

Butterfly scheme:

- Original points remain unmodified (interpolating scheme)
- New points averaged as shown on the right
- C^1 , except from extraordinary vertices
- Can be modified to be C^1 everywhere



Computer Aided Geometric Design

Fall Semester 2024

Implicit Surfaces

陈仁杰

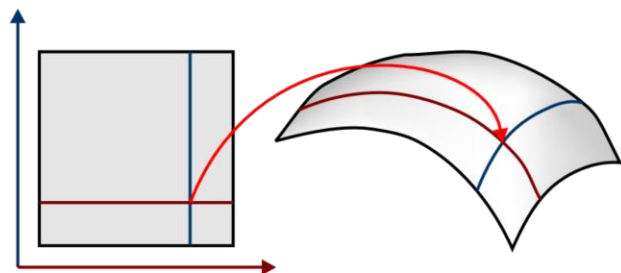
renjiec@ustc.edu.cn

<http://staff.ustc.edu.cn/~renjiec>

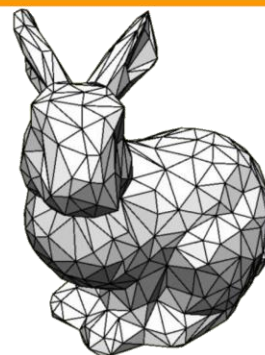
Implicit Surfaces

Introduction

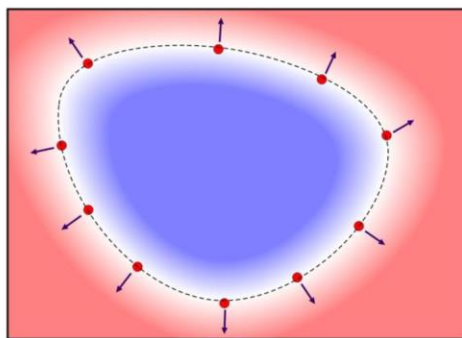
Modeling Zoo



Parametric Models



Primitive Models



Implicit Models



Particle Models

Implicit Functions

Basic Idea:

- We describe an object $S \subseteq \mathbb{R}^d$ by an implicit equation:
 - $S = \{x \in \mathbb{R}^d \mid f(x) = 0\}$
 - The function f describes the shapes of the object.
- Applications:
 - In general, we could describe arbitrary objects
 - Most common case: surfaces in \mathbb{R}^3
 - This means, f is zero on an infinitesimally thin sheet only

The Implicit Function Theorem

Implicit Function Theorem:

- Given a *differentiable* function

$$f: \mathbb{R}^n \supseteq D \rightarrow \mathbb{R}, f(\mathbf{x}^{(0)}) = 0, \frac{\partial}{\partial x_n} f(\mathbf{x}^{(0)}) = \frac{\partial}{\partial x_n} f(x_1^{(0)}, \dots, x_n^{(0)}) \neq 0$$

- Within an ε -neighborhood of $\mathbf{x}^{(0)}$ we can represent the zero level set of f completely as a heightfield function g

$g: \mathbb{R}^{n-1} \rightarrow \mathbb{R}$ such that for $\mathbf{x} - \mathbf{x}^{(0)} < \varepsilon$ we have:

$$f(x_1, \dots, x_{n-1}, g(x_1, \dots, x_{n-1})) = 0 \text{ and}$$

$$f(x_1, \dots, x_n) \neq 0 \text{ everywhere else}$$

- The heightfield is a differentiable $(n - 1)$ -manifold and its surface normal is colinear to the gradient of f .

This means

If we want to model surfaces, we are on the safe side if:

- We use a smooth (differentiable) function f in \mathbb{R}^3
- The gradient of f does not vanish

This gives us the following guarantees:

- The zero-level set is actually a surface
 - We obtained a closed 2-manifold without boundary
 - We have a well defined interior / exterior.

Sufficient:

- We need smoothness / non-vanishing gradient only close to the zero-crossing.

Implicit Functions Types

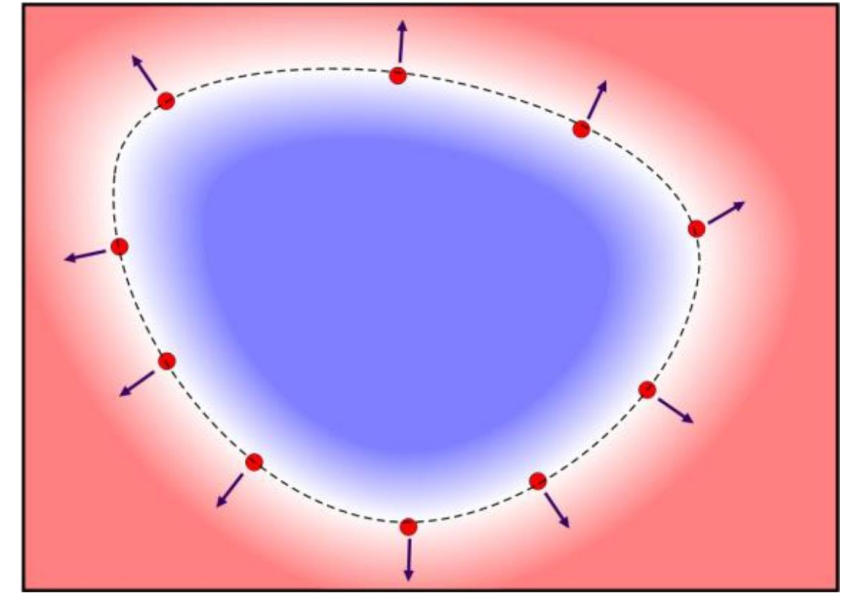
Function Types:

- General case
 - Non-zero gradient at zero crossings
 - Otherwise arbitrary
- Signed implicit function:
 - $\text{sign}(f)$: negative inside and positive outside the object
(or the other way round, but we assume this orientation here)
- Signed distance field (SDF)
 - $|f|$ = distance to the surface
 - $\text{sign}(f)$: negative inside, positive outside
- Squared distance function
 - $f = (\text{distance to the surface})^2$

Implicit Functions Types

Use depends on application:

- Signed implicit function
 - Solid modelling
 - Interior well defined
- Signed distance function (SDF)
 - Most frequently used representation
 - Constant gradient \rightarrow numerically stable surface definition
 - Availability of distance value useful for many applications
- Squared distance function
 - This representation is useful for statistical optimization
 - Minimize sum of squared distances \rightarrow least squares optimization
 - Useful for surface defined up to some insecurity / noise
 - Direct surface extraction more difficult (*gradient vanishes!*)

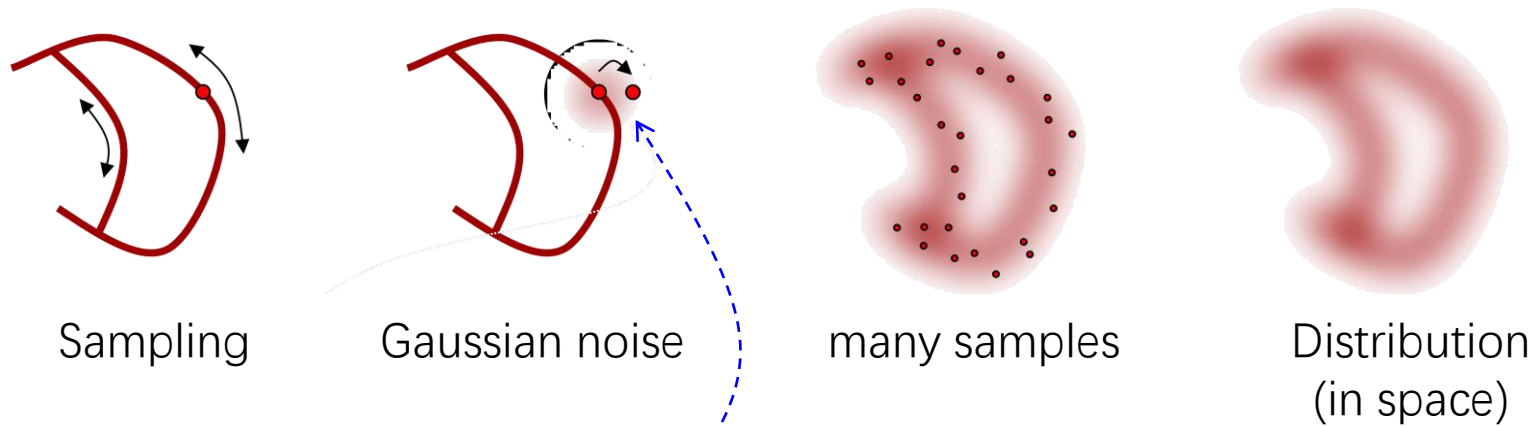


signed distance

Squared Distance Function

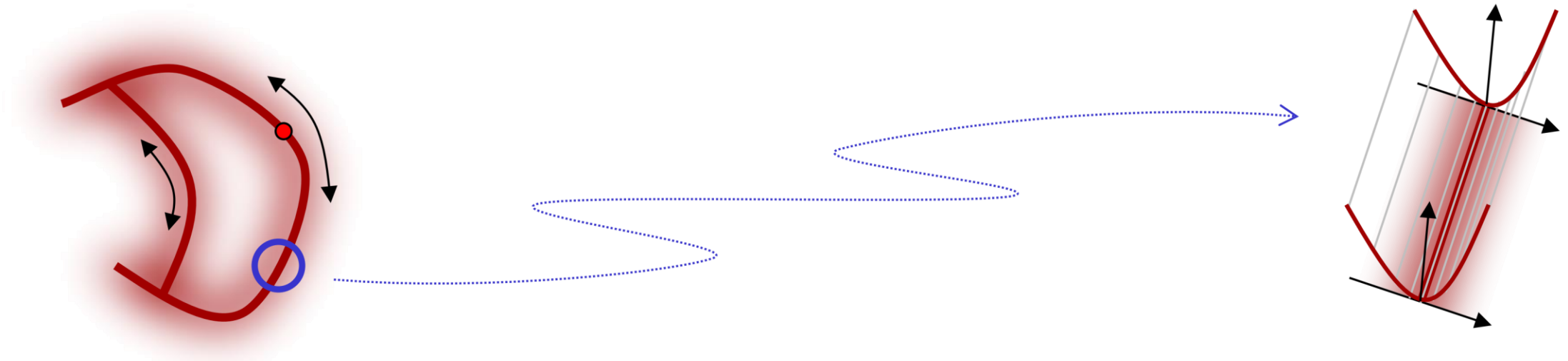
Example: Surface from random samples

1. Determine sample point (uniform)
2. Add noise (Gaussian)



$$p_{\mu, \Sigma}(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left(-\frac{1}{2} (\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu) \right)$$

Squared Distance Function



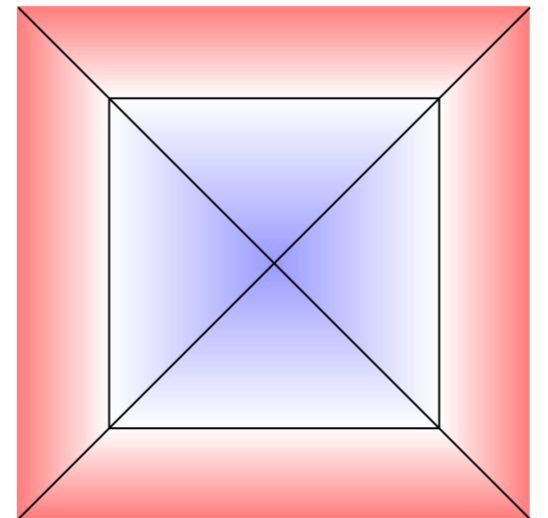
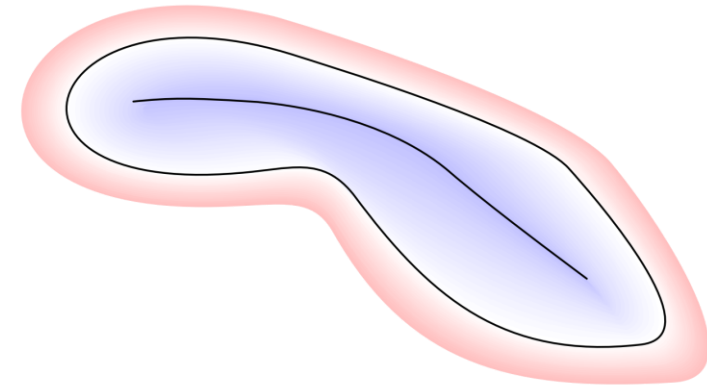
Square Distance Function:

- Sampling a surface with uniform sampling and Gaussian noise:
⇒ Probability density is a convolution of the object with a Gaussian kernel
- Smooth surfaces: The log-likelihood can be approximated by a squared distance function

Smoothness

Smoothness of signed distance function:

- Any distance function (signed, unsigned, squared) in general cannot be globally smooth
- The distance function is non-differentiable at the medial axis
 - Media axis = set of points that have the same distances to two or more different surface points
 - For sharp corners, the medial axis touches the surfaces
 - This means: f non-differentiable on the surface itself
 - Usually, this is no problem in practice



Differential Properties

Some useful differential properties:

- We look at a surface point \mathbf{x} , i.e. $f(\mathbf{x}) = 0$.
- We assume $\nabla f(\mathbf{x}) \neq 0$.
- The unit normal of the implicit surface is given by:

$$\mathbf{n}(\mathbf{x}) = \frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}$$

- For signed functions, the normal is pointing outward
- For signed distance functions, this simplifies to $\mathbf{n}(\mathbf{x}) = \nabla f(\mathbf{x})$

Differential Properties

Some useful differential properties:

- The mean curvature of the surface is proportional to the divergence of the unit normal:

$$-2H(\mathbf{x}) = \nabla \cdot \mathbf{n}(\mathbf{x}) = \frac{\partial}{\partial x} n_x(\mathbf{x}) + \frac{\partial}{\partial y} n_y(\mathbf{x}) + \frac{\partial}{\partial z} n_z(\mathbf{x}) = \nabla \cdot \frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}$$

- For a signed distance function, the formula simplifies to:

$$-2H(\mathbf{x}) = \nabla \cdot \nabla f(\mathbf{x}) = \frac{\partial^2}{\partial x^2} f(\mathbf{x}) + \frac{\partial^2}{\partial y^2} f(\mathbf{x}) + \frac{\partial^2}{\partial z^2} f(\mathbf{x}) = \Delta f(\mathbf{x})$$

Computing Volume Integrals

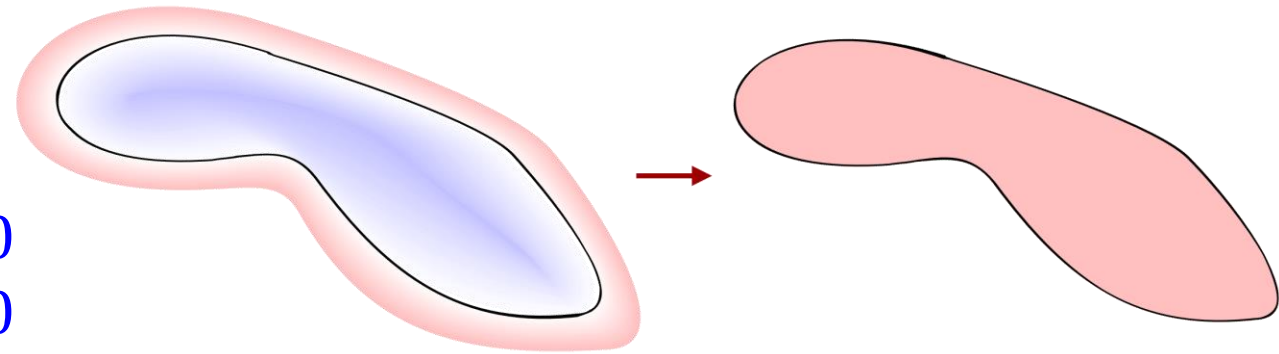
Computing volume integrals

- Heavyside function

$$\text{step}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

- Volume integral over interior volume Ω_f of some function $g(\mathbf{x})$ (assuming negative interior values):

$$\int_{\Omega_f} g(\mathbf{x}) d\mathbf{x} = \int_{\mathbb{R}^3} g(\mathbf{x}) \left(1 - \text{step}(f(\mathbf{x}))\right) d\mathbf{x}$$

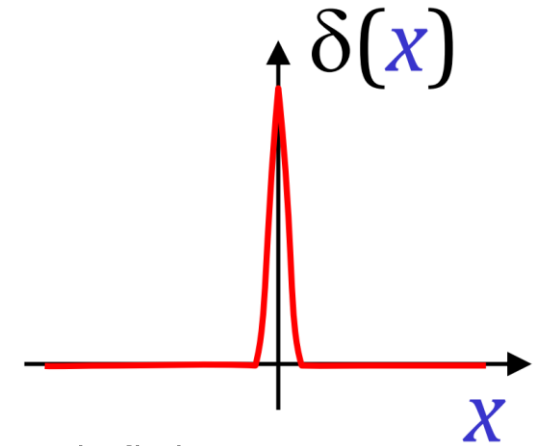


Computing Surface Integrals

Computing surface integrals:

- Dirac delta functions:
 - Idealized function (distribution)
 - Zero everywhere ($\delta(\mathbf{x}) = 0$), except at $\mathbf{x} = 0$, where it is positive, infinitely large
 - The integral of $\delta(\mathbf{x})$ over \mathbf{x} is one
- Dirac delta function on the surface: directional derivative of $\text{step}(\mathbf{x})$ in normal direction:

$$\begin{aligned}\hat{\delta} &= \nabla[\text{step}(f(\mathbf{x}))] \cdot \mathbf{n}(\mathbf{x}) = [\nabla \text{step}](f(\mathbf{x})) \nabla f(\mathbf{x}) \cdot \frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|} \\ &= \delta(f(\mathbf{x})) \cdot \|\nabla f(\mathbf{x})\|\end{aligned}$$



Surface Integral

Computing surface integrals:

- Surface integral over the surface $\partial\Omega_f = \{\mathbf{x} | f(\mathbf{x}) = 0\}$ of some function $g(\mathbf{x})$:

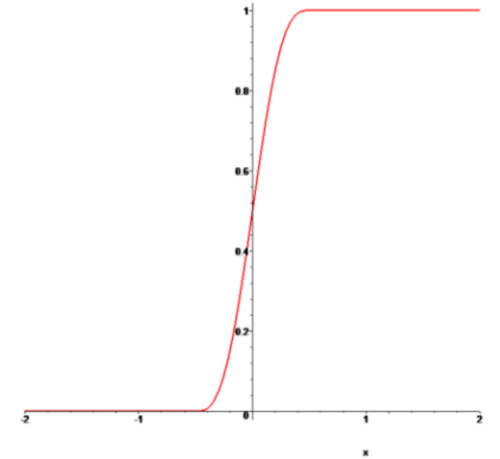
$$\int_{\Omega_f} g(\mathbf{x}) d\mathbf{x} = \int_{\mathbb{R}^3} g(\mathbf{x}) \delta(f(\mathbf{x})) |\nabla f(\mathbf{x})| d\mathbf{x}$$

- This looks nice, but is numerically intractable.
- We can fix this using smoothed out Dirac/Heavyside functions...

Smoothed Functions

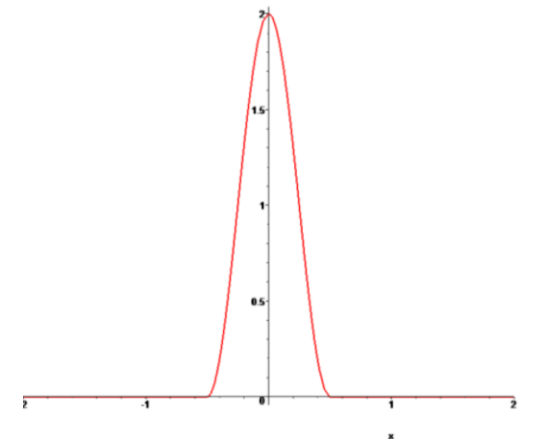
Smooth-step function

$$\text{smoothstep}(x) = \begin{cases} 0 & x < -\varepsilon \\ \frac{1}{2} + \frac{x}{2\varepsilon} + \frac{1}{2\pi} \sin\left(\frac{\pi x}{\varepsilon}\right) & -\varepsilon \leq x \leq \varepsilon \\ 1 & \varepsilon < x \end{cases}$$



Smoothed Dirac delta function

$$\text{smoothdelta}(x) = \begin{cases} 0 & x < -\varepsilon \\ \frac{1}{2\varepsilon} + \frac{1}{2\varepsilon} \cos\left(\frac{\pi x}{\varepsilon}\right) & -\varepsilon \leq x \leq \varepsilon \\ 0 & \varepsilon < x \end{cases}$$



Implicit Surfaces

Numerical Discretization

Representing Implicit Functions

Representation: Two basic techniques

- Discretization on grids
 - Simple finite differencing (FD) grids
 - Grids of basis functions (finite elements FE)
 - Hierarchical / adaptive grids (FE)
- Discretization with radial basis functions
(particle FE methods)

Discretization

Discretization examples

- In the following, we will look at 2D examples
- The 3D (d -dimensional) case is similar

Regular Grids

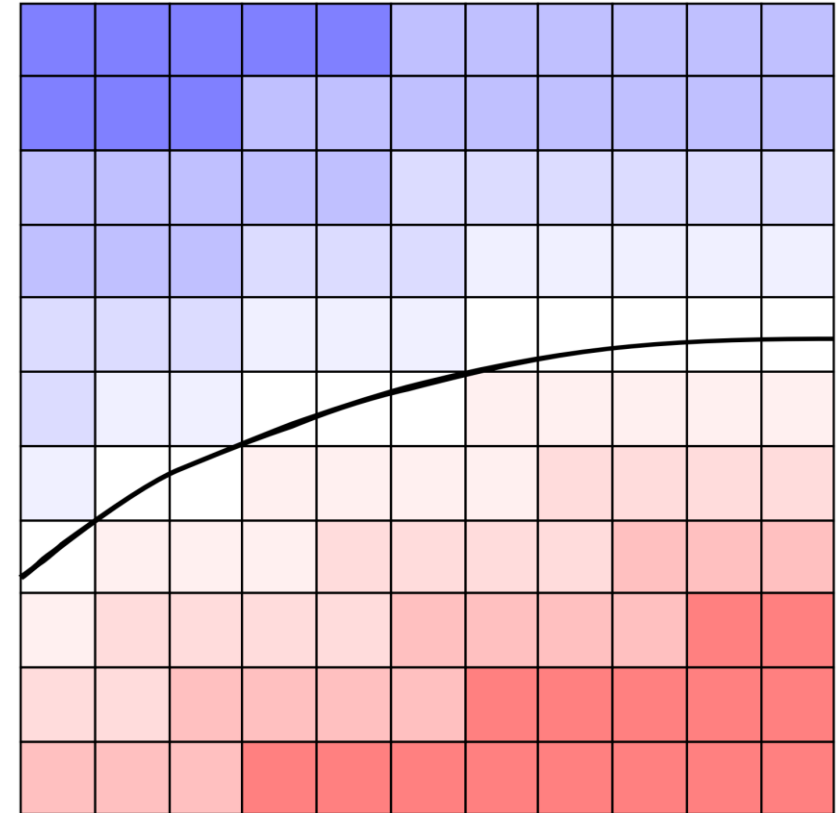
Discretization:

- Regular grid of values $f_{i,j}$
- Grid spacing h
- Differential properties can be approximated by finite differences:

- For example

$$\frac{\partial}{\partial x} f(\mathbf{x}) = \frac{1}{h} (f_{i(x),j(x)} - f_{i(x)-1,j(x)}) + O(h)$$

$$\frac{\partial}{\partial x} f(\mathbf{x}) = \frac{1}{2h} (f_{i(x)+1,j(x)} - f_{i(x)-1,j(x)}) + O(h^2)$$

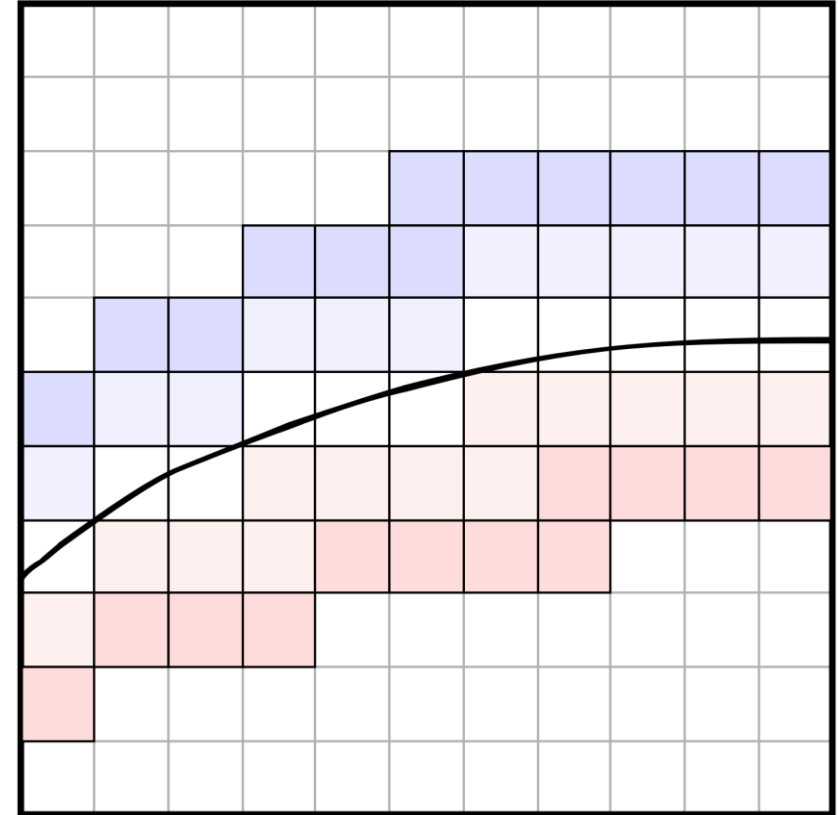


Regular Grids

Variant:

- Use only cells near the surface
- Saves storage & computation time
- However: we need to know an estimate on where the surface is located to setup the representation
- Propagate to the rest of the volume (if necessary):

fast marching method



Fast Marching Method

Problem statement:

- Assume we are given the surface and signed distance value in a narrow band
- Now we want to compute distance values everywhere on the grid

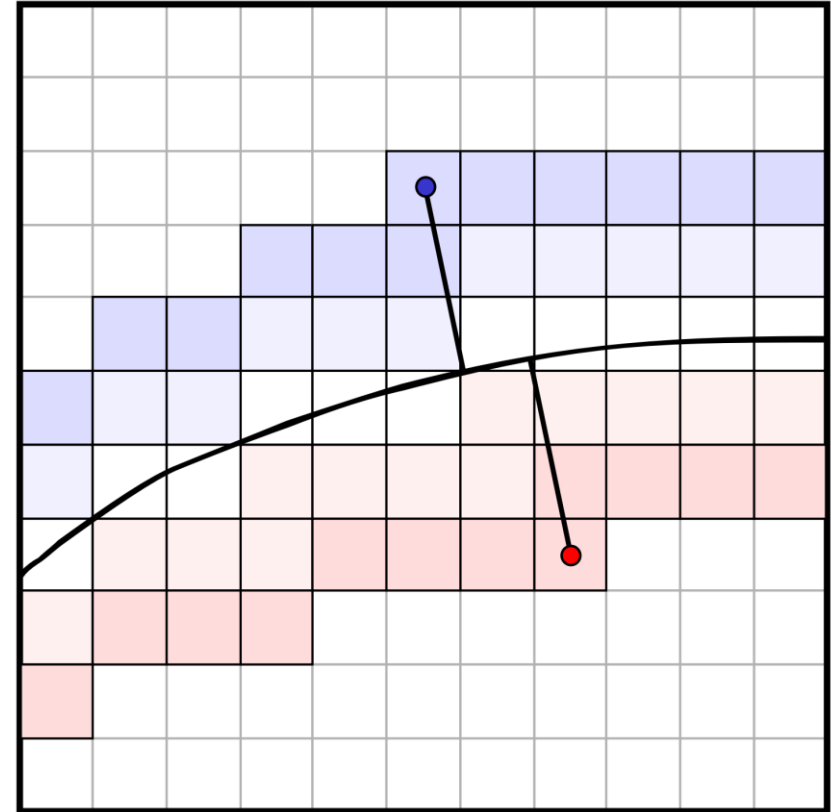
Three Solutions:

- Nearest neighbor queries
- Eikonal equation
- Fast marching

Nearest Neighbors

Algorithm:

- For each grid cell:
 - Compute nearest point on the surface
 - Enter distance
- Approximate nearest neighbor computation:
 - Look for nearest grid cell with zero crossing first
 - Then compute distance curve \leftrightarrow zero level set using a Newton-like algorithm (repeated point-to-plane distance)
- Costs: $O(n)$ kNN queries (n empty cells)

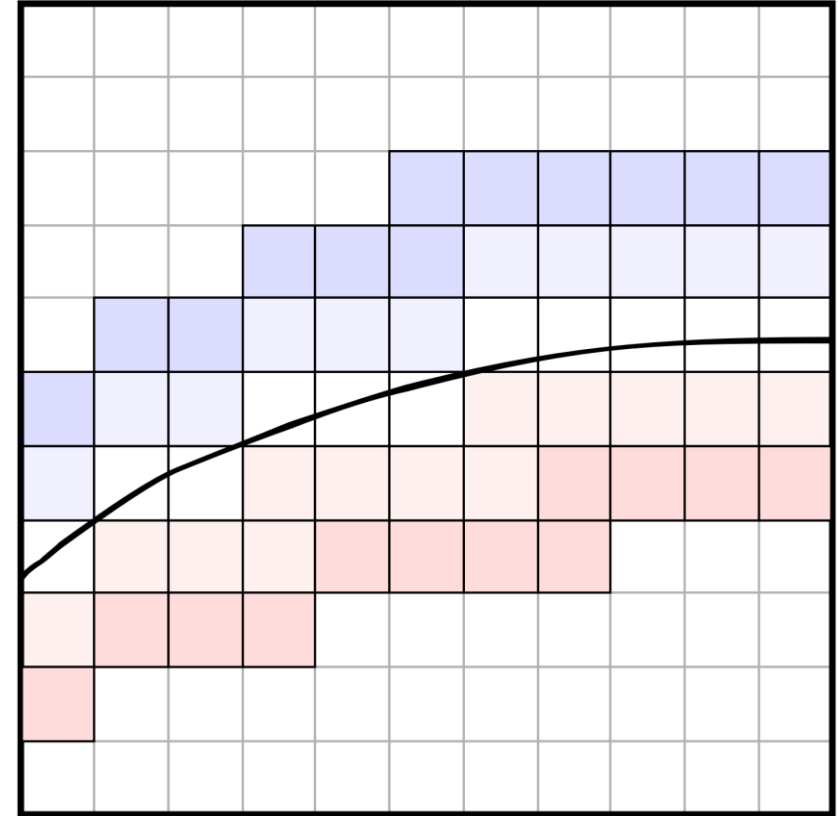


Eikonal Equation

Eikonal Equation

- Place variables in empty cells
- Fixed values in known cells
- Then solve the following PDE:
$$\|\nabla f\| = 1$$

subject to $f(\mathbf{x}) = f_{\text{known}}(\mathbf{x})$
on the known area $\mathbf{x} \in A_{\text{known}}$
- This is a (non-linear) boundary value problem



Fast Marching

Solving the Equation:

- The Eikonal equation can be solved efficiently by a region growing algorithm:
 - Start with the initial known values
 - Compute new distances at immediate neighbors solving a local Eikonal equation (*)
 - The smallest of these values must be correct (similar to Dijkstra's algorithm)
 - Fix this value and update the neighbors again
 - Growing front, $O(n \log n)$ time

Regular Grids of Basis Functions

Discretization (2D):

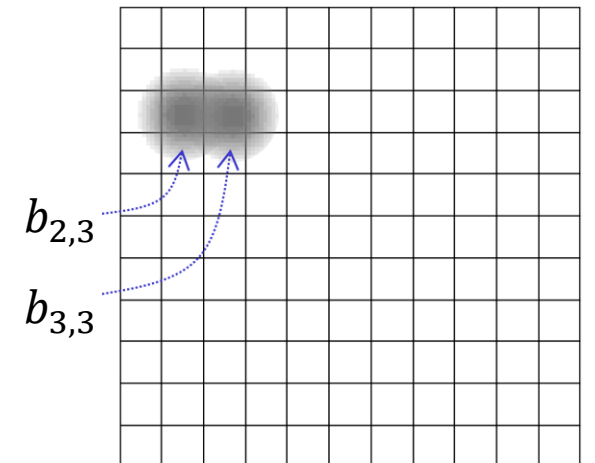
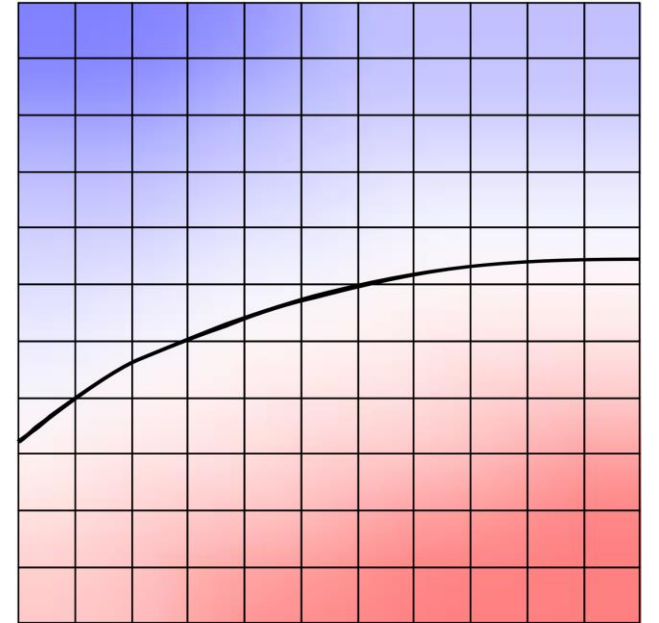
- Place a basis function in each grid cell:

$$b_{i,j} = b(x - i, y - j)$$

- Typical choices:
 - Bivariate uniform cubic B-splines (tensor product)
 - $b(x, y) = \exp[-\lambda(x^2 + y^2)]$
- The implicit function is then represented as

$$f(x, y) = \sum_0^{n_i} \sum_0^{n_j} \lambda_{i,j} b_{i,j}(x, y)$$

- The $\lambda_{i,j}$ describe different f



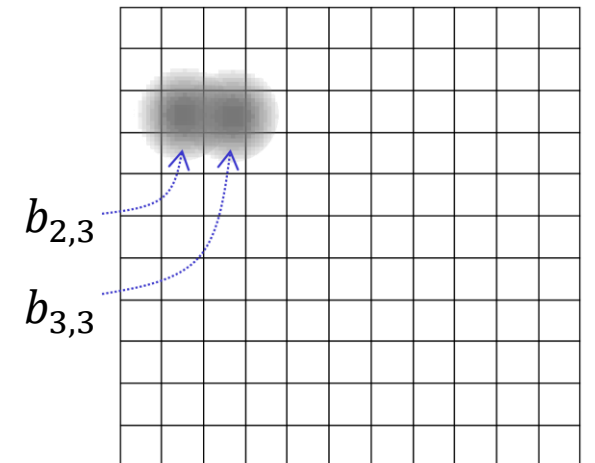
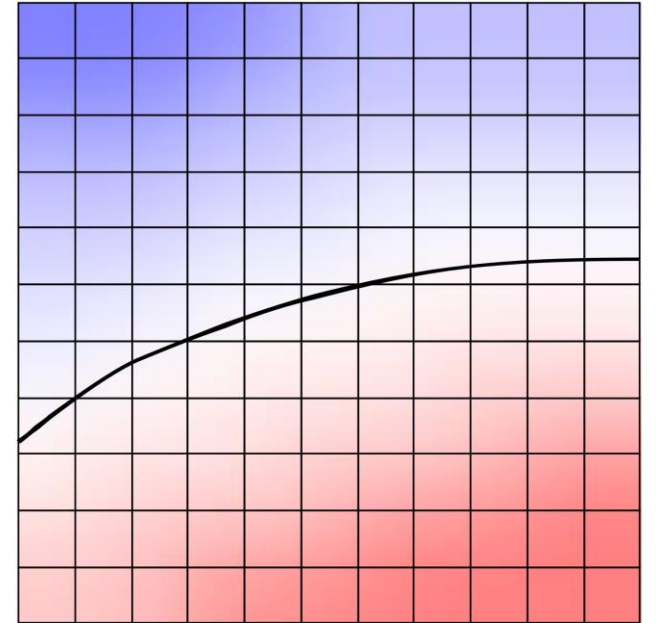
Regular Grids of Basis Functions

Differential Properties:

- Derivatives:

$$\frac{\partial}{\partial x_{k1} \dots \partial x_{km}} f(x, y) = \sum_{i=0}^{n_i} \sum_{j=0}^{n_j} \lambda_{i,j} \left(\frac{\partial}{\partial x_{k1} \dots \partial x_{km}} b \right) (x, y)$$

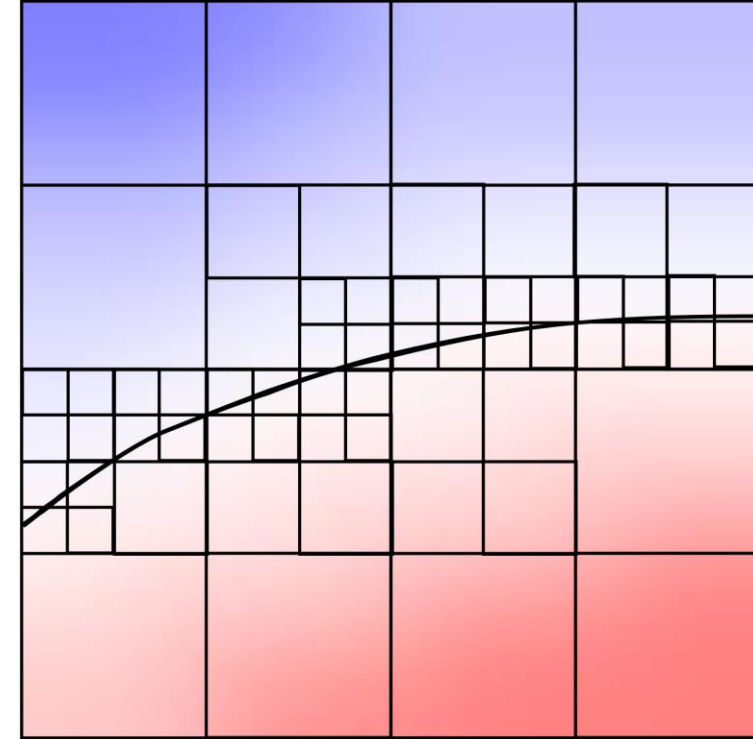
- Derivatives are linear combinations of the derivatives of the basis function
- In particular: we again get a linear expression in $\lambda_{i,j}$



Adaptive Grids

Adaptive / hierarchical grids:

- Perform a quadtree / octree tessellation of the domain (or any other partition into elements)
- Refine where more precision is necessary (near surface, maybe curvature dependent)
- Associate basis functions with each cell (constant or higher order)



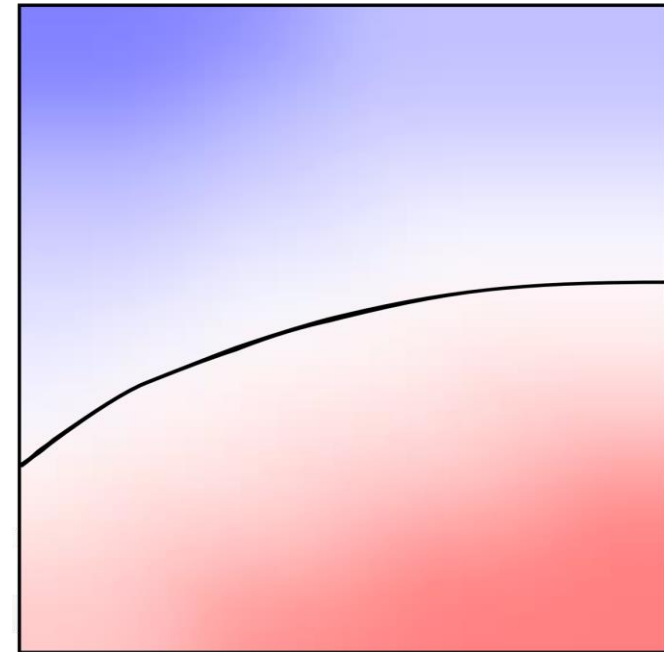
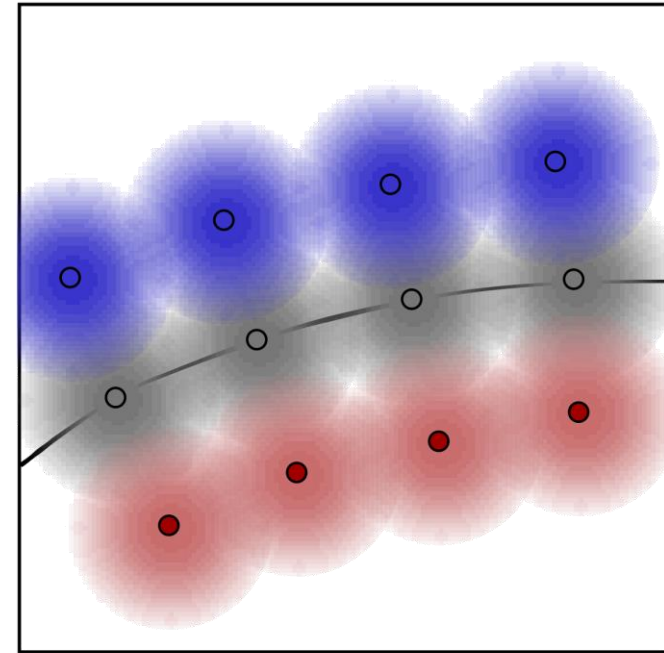
Particle Methods

Particle methods / radial basis function:

- Place a set of “particles” in space at positions \mathbf{x}_i
- Associate each with a radial basis function $b(\mathbf{x} - \mathbf{x}_i)$
- The discretization is then given by:

$$f(\mathbf{x}) = \sum_{i=0}^n \lambda_i b(\mathbf{x} - \mathbf{x}_i)$$

- The λ_i encode f .



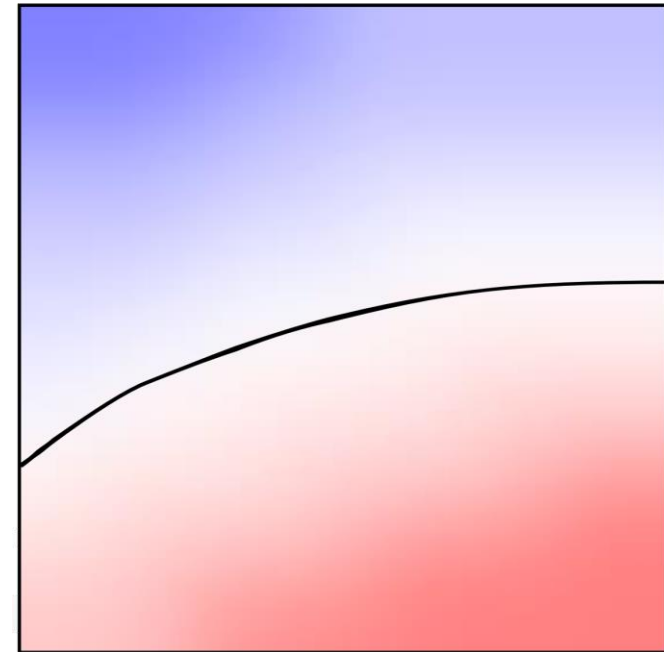
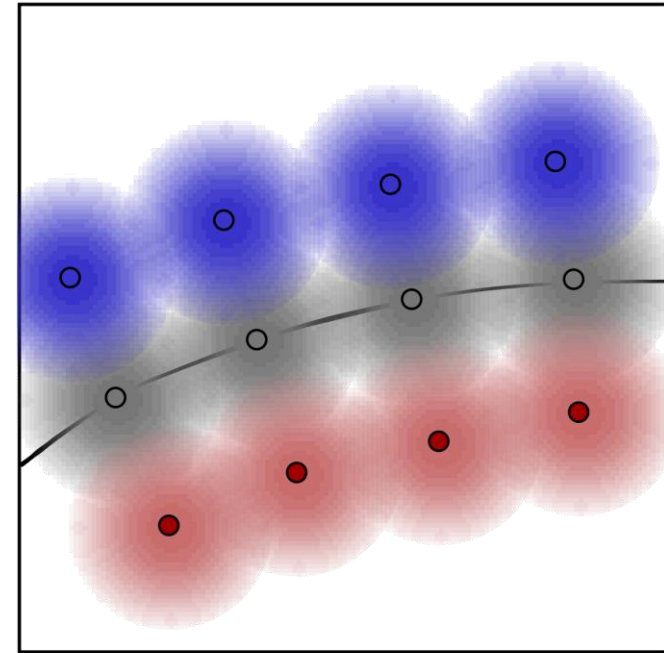
Particle Methods

Particle methods / radial basis function:

- Obviously, derivatives are again linear in λ_i :

$$\frac{\partial}{\partial x_{k1} \dots \partial x_{km}} f(\mathbf{x}) = \sum_{i=0}^{n_j} \lambda_{i,j} \left(\frac{\partial}{\partial x_{k1} \dots \partial x_{km}} b \right) (\mathbf{x} - \mathbf{x}_i)$$

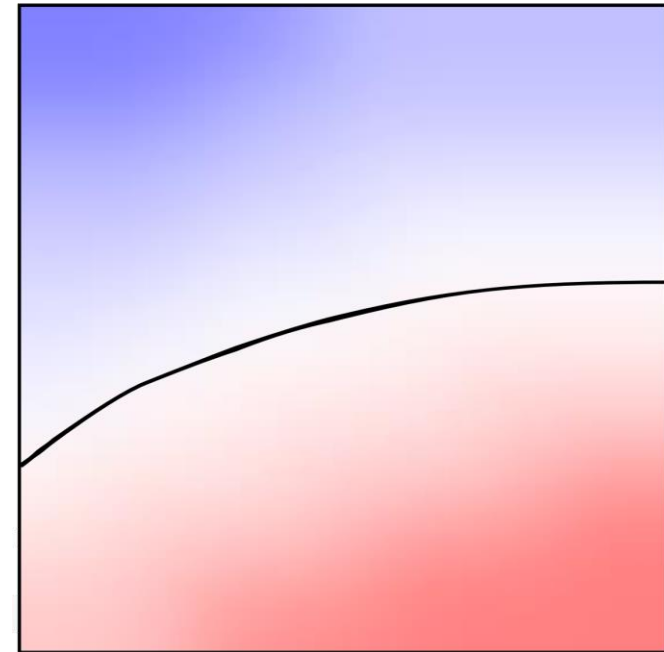
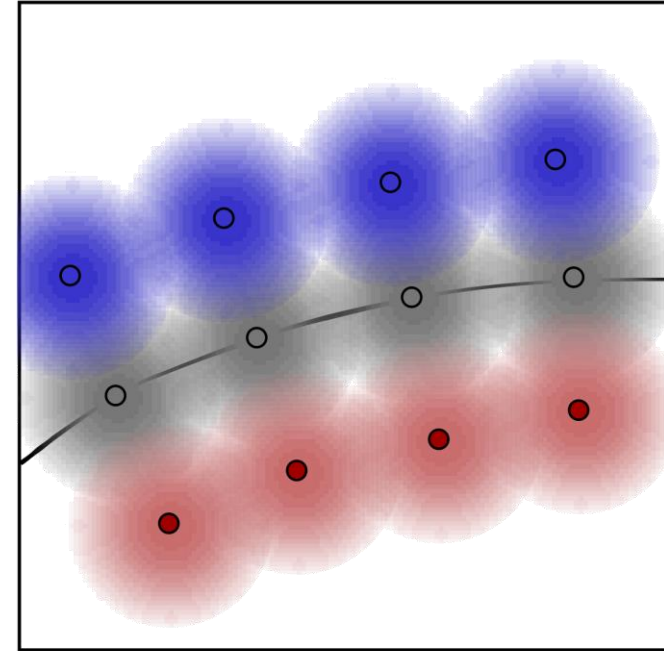
- The radial basis functions can also have different size (support) for adaptive refinement
- Placement: near the expected surface



Particle Methods

Particle methods / radial basis function:

- Where should we place the radial basis functions?
 - If we have an initial guess for the surface shape:
 - Put some on the surface
 - And some in +/- normal direction
 - Otherwise:
 - Uniform placement in lowres
 - Solve for surface
 - Refine near lowres-surface, iterate



Types of Radial Basis Functions

Typical choices for radial basis functions:

- (Quasi-) compactly supported functions:
 - Exponentials / normal distribution densities: $\exp(-\lambda x^2)$
 - Uniform (cubic) tensor product B-Splines
 - Moving-least squares finite element basis functions (will be discussed later)
- Globally supported functions:
 - Thin plate spline basis functions:
 $\|x - x_0\|^2 \ln \|x - x_0\|$ (2D), $\|x - x_0\|^3$ (3D)
 - These functions guarantee minimal integral second derivatives.

Pros & Cons

Why use globally supported basis functions?

- They come with smoothness guarantees
- However: computations might become expensive (we will see later how to device efficient algorithms for globally supported radial basis functions)

Locally supported functions:

- Easy to use
- Additional regularization might become necessary to compute a “nice” surface

Implicit Surfaces

Level Set Extraction

Iso-Surface Extraction

New task:

- Assume we have defined an implicit function
- Now we want to extract the surface
- I.e. convert it to an explicit, piecewise parametric representation, typically a triangle mesh
- For this we need an iso-surface extraction algorithm
 - a.k.a. level set extraction
 - a.k.a. contouring

Algorithms

Algorithms:

- Marching Cubes
 - This is the standard technique
 - We will also discuss some problems / modifications
- Particle methods
 - Just to show an alternative
 - Not used that frequently in practice

Marching Cubes

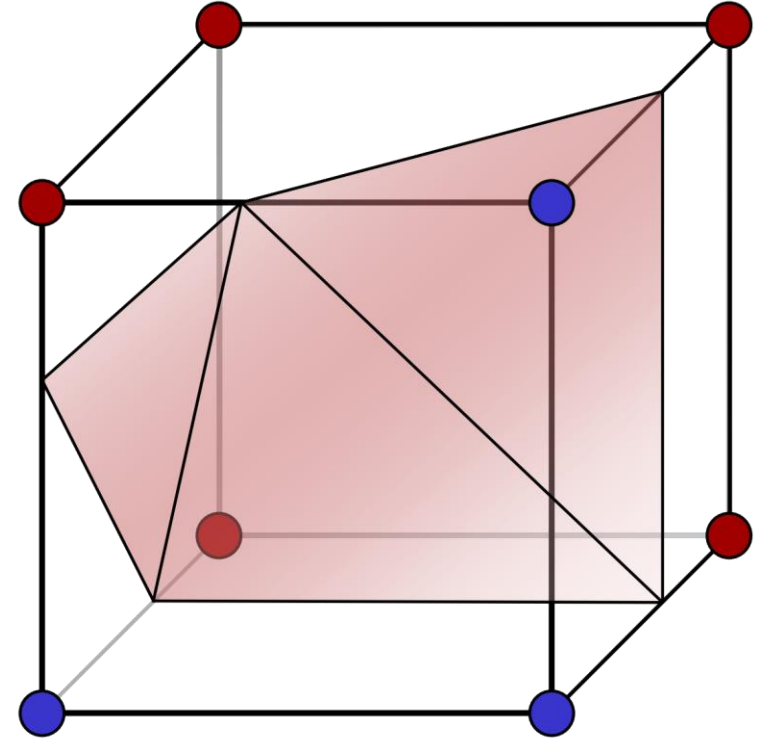
Marching Cubes:

- The most frequently used iso surface extraction algorithm
 - Creates a triangle mesh from an iso-value surface of a scalar volume
 - The algorithm is also used frequently to visualize CT scanner data and other volume data
- Simple idea:
 - Define and solve a fixed complexity, local problem
 - Compute a full solution by solving many such local problems incrementally

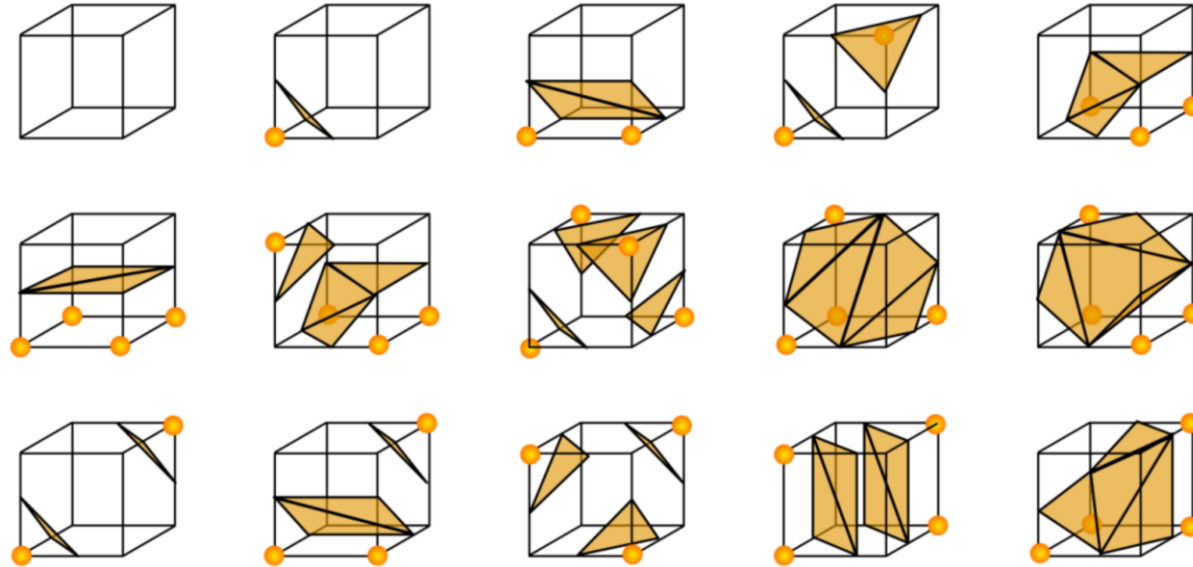
Marching Cubes

Marching Cubes:

- Here is the local problem:
 - We have a cube with 8 vertices
 - Each vertex is either inside or outside the volume (i.e. $f(\mathbf{x}) < 0$ or $f(\mathbf{x}) \geq 0$)
 - How should we triangulate this cube?
 - How should we place the vertices?



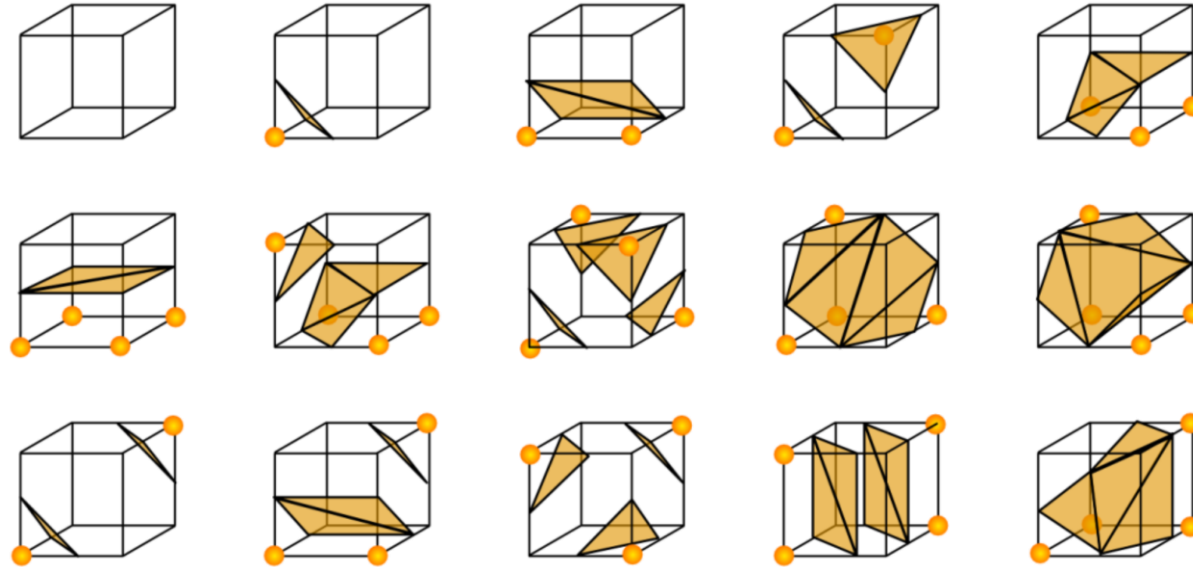
Triangulation



Triangulation:

- We have 256 different cases – each of the 8 vertices can be in or out
- By symmetry, this can be reduced to 15 cases
 - Symmetry: reflection, rotation, and bit inversion
- This means, we can compute the topology of the mesh

Vertex Placement



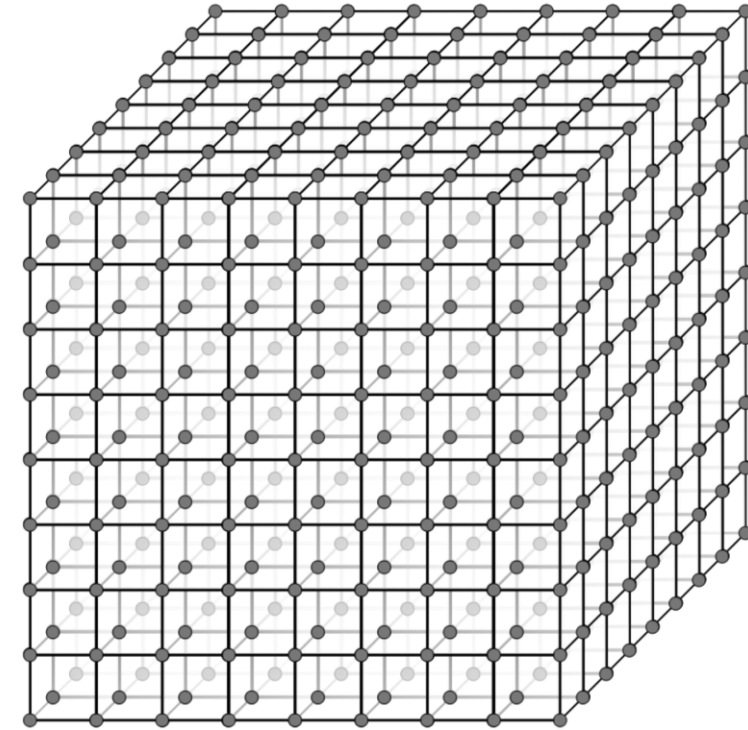
How to place the vertices?

- Zero-th order accuracy: Place vertices at edge midpoints
- First order accuracy: Linearly interpolate vertices along edges.
- Example: for scalar values $f(x) = -0.1$ and $f(y) = 0.2$, place the vertex at ratio 1:2 between x and y

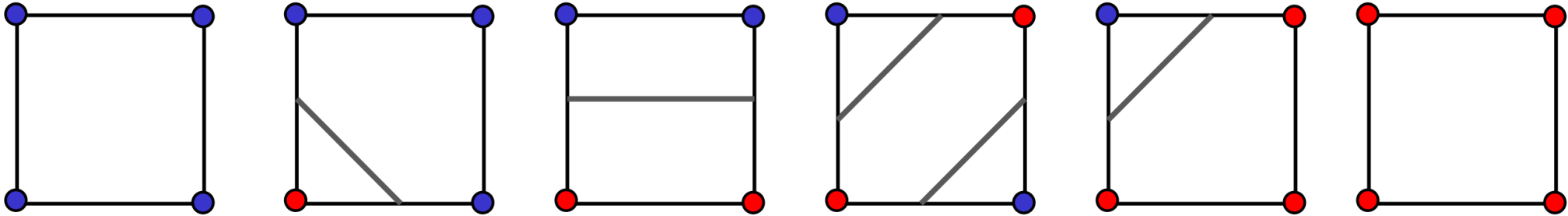
Outer Loop

Outer Loop:

- Compute a bounding box of the domain of the implicit function
- Divide it into cubes of the same size (regular cube grid)
- Execute “marching cube” algorithm in each subcube
- Output the union of all triangles generated
- Optionally: Use a vertex hash table to make the mesh consistent (remove double vertices)



Marching Squares



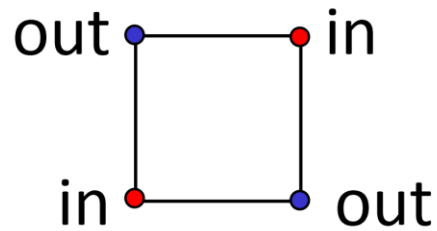
Marching Squares:

- There is also a 2D version of the algorithm, called marching squares
- Same idea, but fewer cases

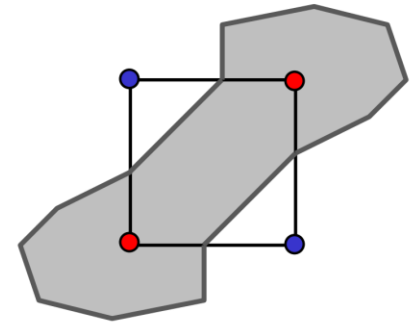
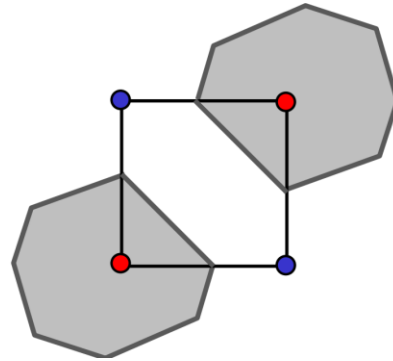
Ambiguities

There is a (minor) technical problem remaining:

- The triangulation can be ambiguous
- In some cases, different topologies are possible which are all locally plausible:



?



- This is an *undersampling artifact*. At a sufficiently high resolution, this cannot occur.
- Problem: Inconsistent application can lead to holes in the surface (non-manifold solutions)

Ambiguities

Solution

- Always use the same solution pattern in ambiguous situations
- For example: Always *connect* diagonally
 - This might yield topologically wrong results.
 - But the surface is guaranteed to be a triangulated 2-manifold without holes and with well-defined interior / exterior
- Better solution:
 - Use higher resolution sampling (if possible)
- All of this (problem and solutions) also applies to the 3D case.

MC Variations

Empty space skipping:

- Marching cube uses an n^3 voxel grid, which can become pretty expensive
- The surface intersects typically only $O(n^2)$ voxels.
- If we roughly know where the surface might appear, we can restrict the execution of the algorithm (and the evaluations of f at the corners) to a narrow band around the surface.
- Example: Particle methods – only extract within the support of the radial basis functions.

MC Variations

Hierarchical marching cubes algorithm:

- One can use a hierarchical version of the marching cubes algorithms using a balanced octree instead of a regular grid
 - We need some refinement criterion to judge on where to subdivide
 - This is application dependent (depends on the definition of f).
- However, we obtain many more cases to consider (which is painful to derive).

Simple solution (common in practice):

- Extract high-resolution triangle mesh
- Then run mesh simplification (slower, but better quality).

Particle-Based Extraction

Particle-based method:

- This technique creates a set of points as output, which cover the iso-surface.
- Algorithm:
 - Start with a random point cloud (n points in a bounding volume)
 - Now define forces that attract particles to the zero-level set.
 - Also add some (weak) tangential repulsion to make them distribute uniformly

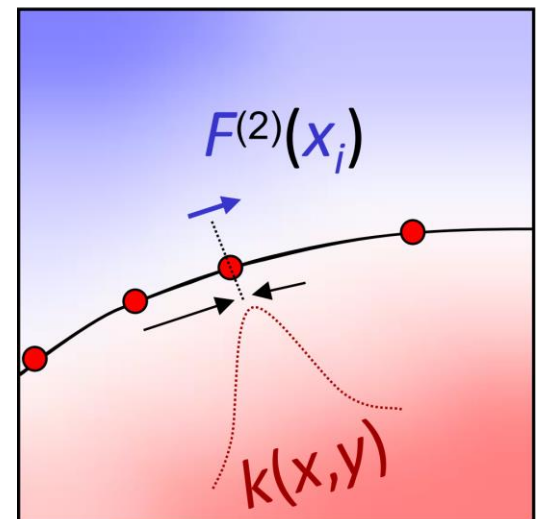
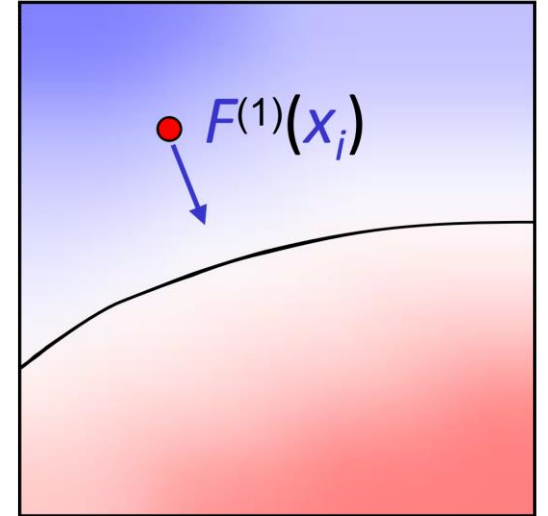
Forces

Attraction “force”:

$$F^{(1)}(x_i) = m_i \|\nabla f(x_i)\|^2$$

Tangential repulsion force:

$$F^{(2)}(x_i) = \left(\sum_{j \neq i} k(x_i, x_j) \frac{x_i - x_j}{\|x_i - x_j\|^2} \right) \left(I - \left[\frac{\nabla f(x_i)}{\|\nabla f(x_i)\|} \right] \cdot \left[\frac{\nabla f(x_i)}{\|\nabla f(x_i)\|} \right]^T \right)$$



Solution

Solution:

- We obtain a system of ordinary differential equations
- The ODE can be solved numerically
- Simplest technique: gradient decent (explicit Euler)
 - Move every point by a fraction of the force vector
 - Recalculate forces
 - Iterate
- We have the solution if the system reaches a steady state (nothing moves anymore, numerically)

Implicit Surfaces

Solid Modeling

Solid Modeling

We want to:

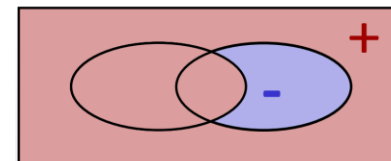
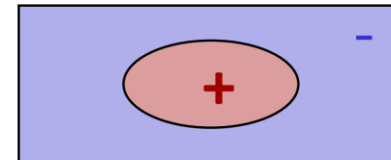
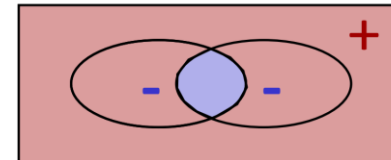
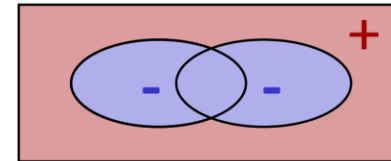
- Form basic volumetric primitives (spheres, cubes, cylinders) as implicit functions (this is easy, no details)
- Compute Boolean combinations of these primitives:
Intersection, union, etc...
- Derive an implicit function from these operations

Boolean Operations

Actually, Boolean operations with implicit functions are simple:

- Given two signed implicit functions (negative inside) f_A, f_B for objects A, B
- The Boolean combinations are given by:

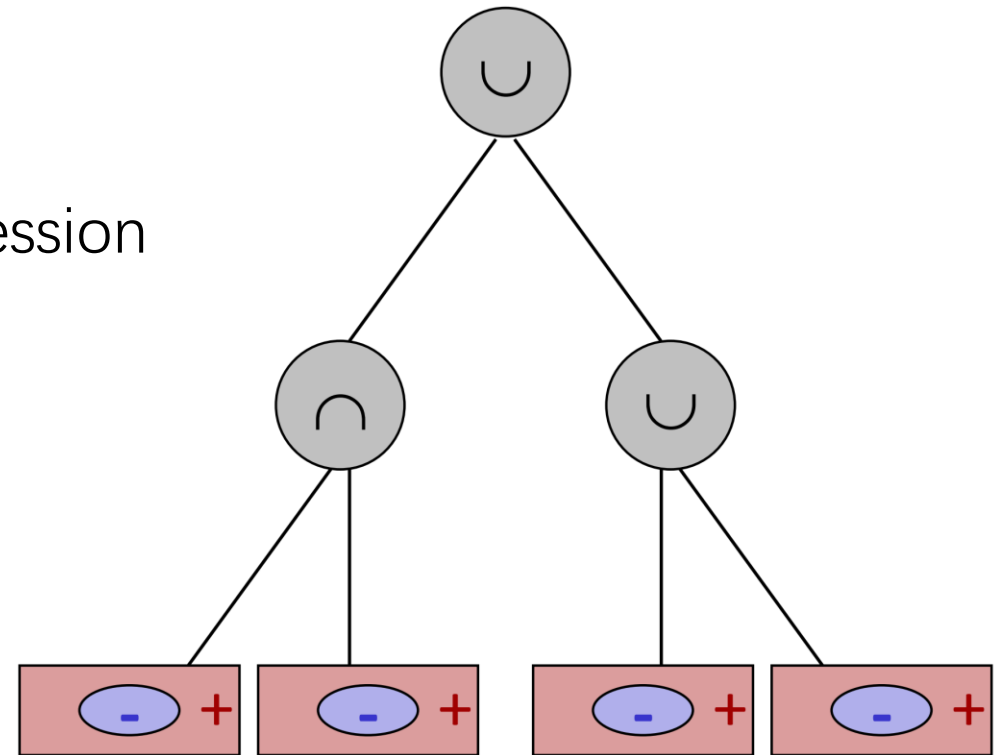
- Union $A \cup B$: $f_{A \cup B} = \min(f_A, f_B)$
- Intersection $A \cap B$: $f_{A \cap B} = \max(f_A, f_B)$
- Complement $\neg A$: $f_{\neg A} = -f_A$
- Difference $A \setminus B$: $f_{A \setminus B} = \min(f_A, -f_B)$



Hierarchical Modeling

This can be models as a CSG tree (constructive solid geometry):

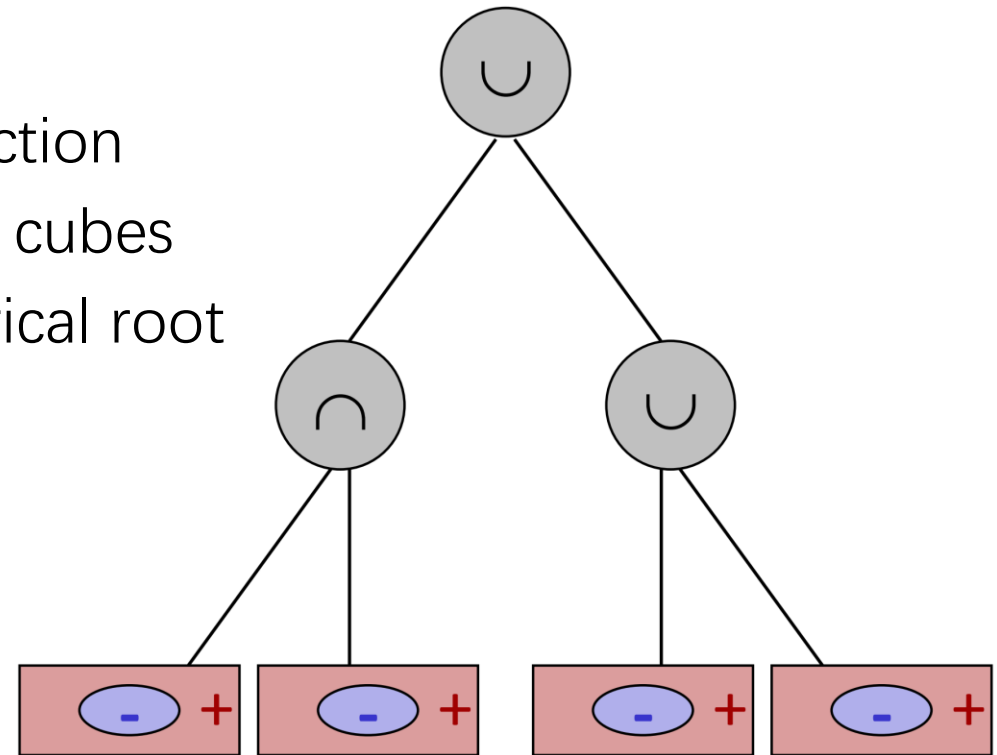
- Leaf nodes are signed distance functions
- Inner nodes are Boolean operations
- Evaluation translates to an arithmetic expression
- Other operations:
 - Deformation (apply vector field)
 - Blending (combine surface smoothly)



Hierarchical Modeling

Rendering CSG hierarchies:

- Rendering is simple
- We get one compound signed implicit function
- We can extract the surface using marching cubes
- We can raytrace the surface using a numerical root finding algorithm
 - For example:
Newton scheme with voxel-based initialization



Implicit Surfaces

Data Fitting

Constructing Implicit Surfaces

Question: How to construct implicit surfaces?

- Basic primitives: Spheres, boxes etc ... are (almost) trivial.
- We can construct implicit spline schemes by using 3D tensor product (or tetrahedral) constructions of 3D Bezier or B-Spline functions
- Another option: Variational modeling
- In this chapter of this lecture: Fitting to data

Data Fitting

Data Fitting Problem:

- We are given a set of points
- We want to find an implicit surface that interpolates or approximates these points
- This problem is ill-defined
- We need additional assumptions to make it well-defined
- We will look at three variants:
 - Hoppe's method / plane blending
 - Thin-plate spline data matching
 - MPU Implicits (multi-level partition of unity implicits)

Plane Blending Method

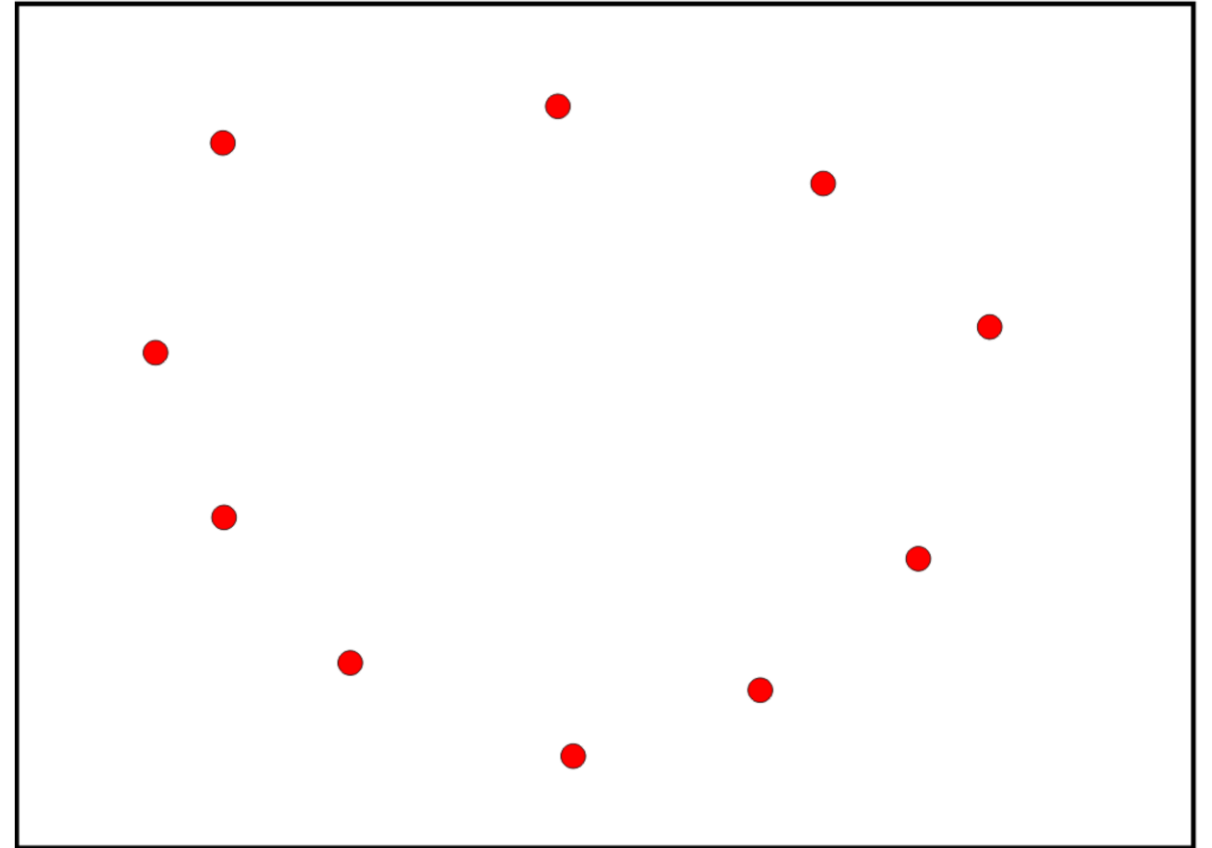
Initial data

Estimate normal

Signed distance func.

Marching cubes

Final mesh



Plane Blending Method

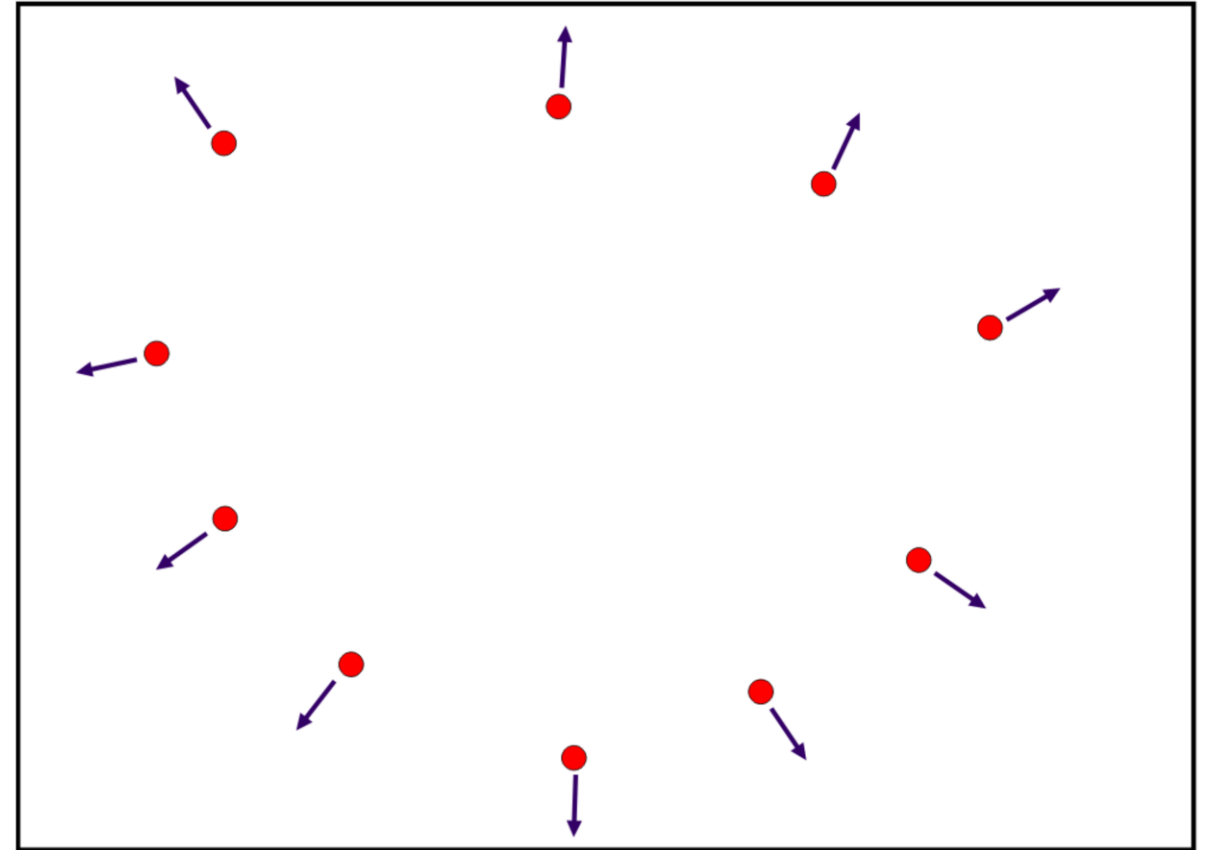
Initial data

Estimate normal

Signed distance func.

Marching cubes

Final mesh



unoriented normal:

total least squares plane fit (PCA)
in a k -nearest neighbors neighborhood

Plane Blending Method

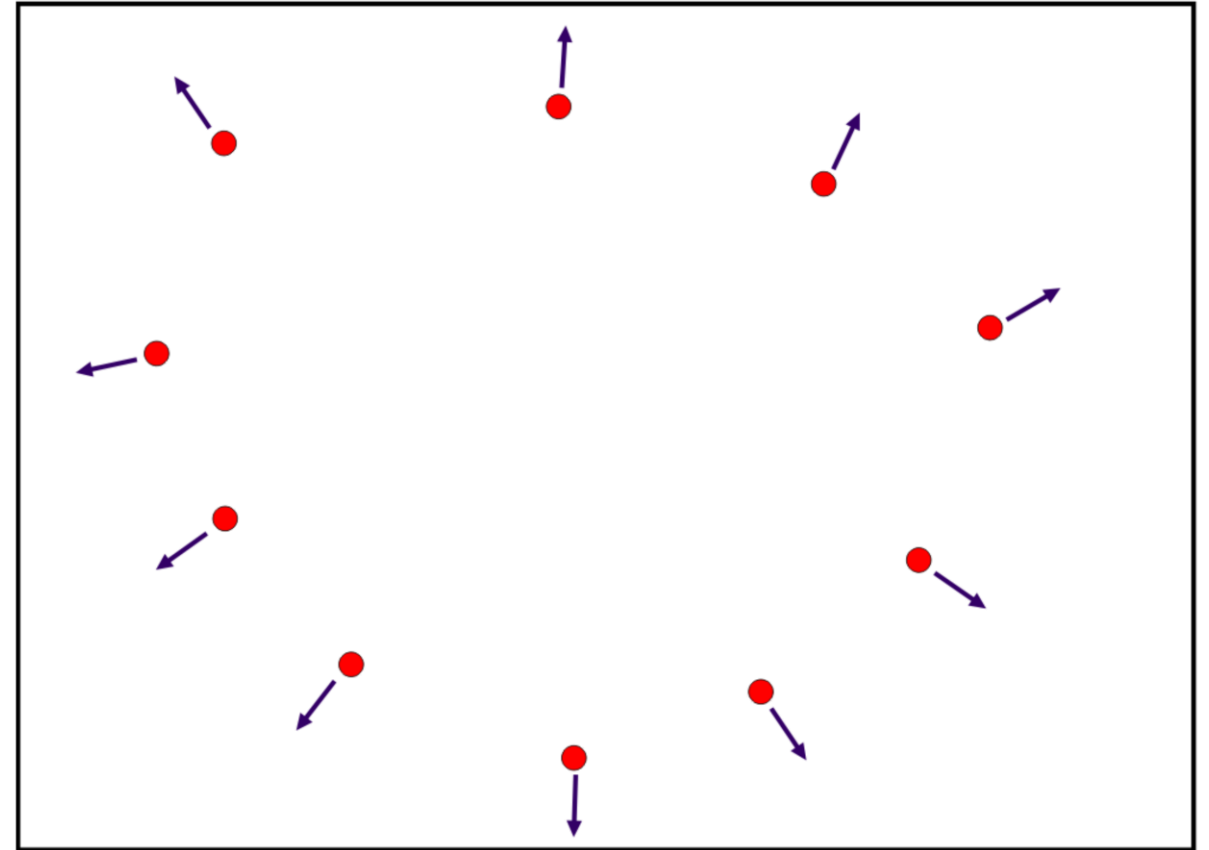
Initial data

Estimate normal

Signed distance func.

Marching cubes

Final mesh



consistent orientation:

region growing, flip normal if angle $> 180^\circ$
pick most similar normal next in each step

Plane Blending Method

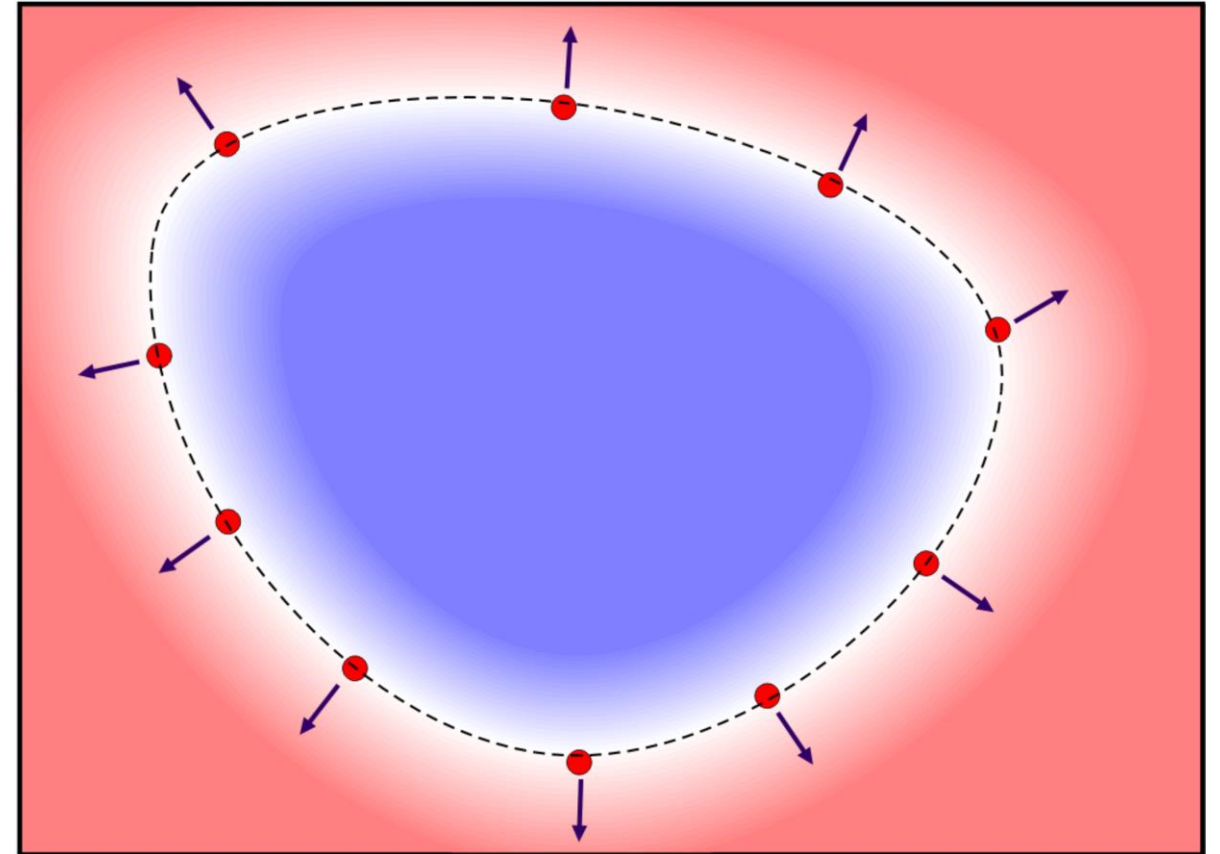
Initial data

Estimate normal

Signed distance func.

Marching cubes

Final mesh



consistent orientation:

blend between signed distance functions of
planes associated with each point

Plane Blending Method

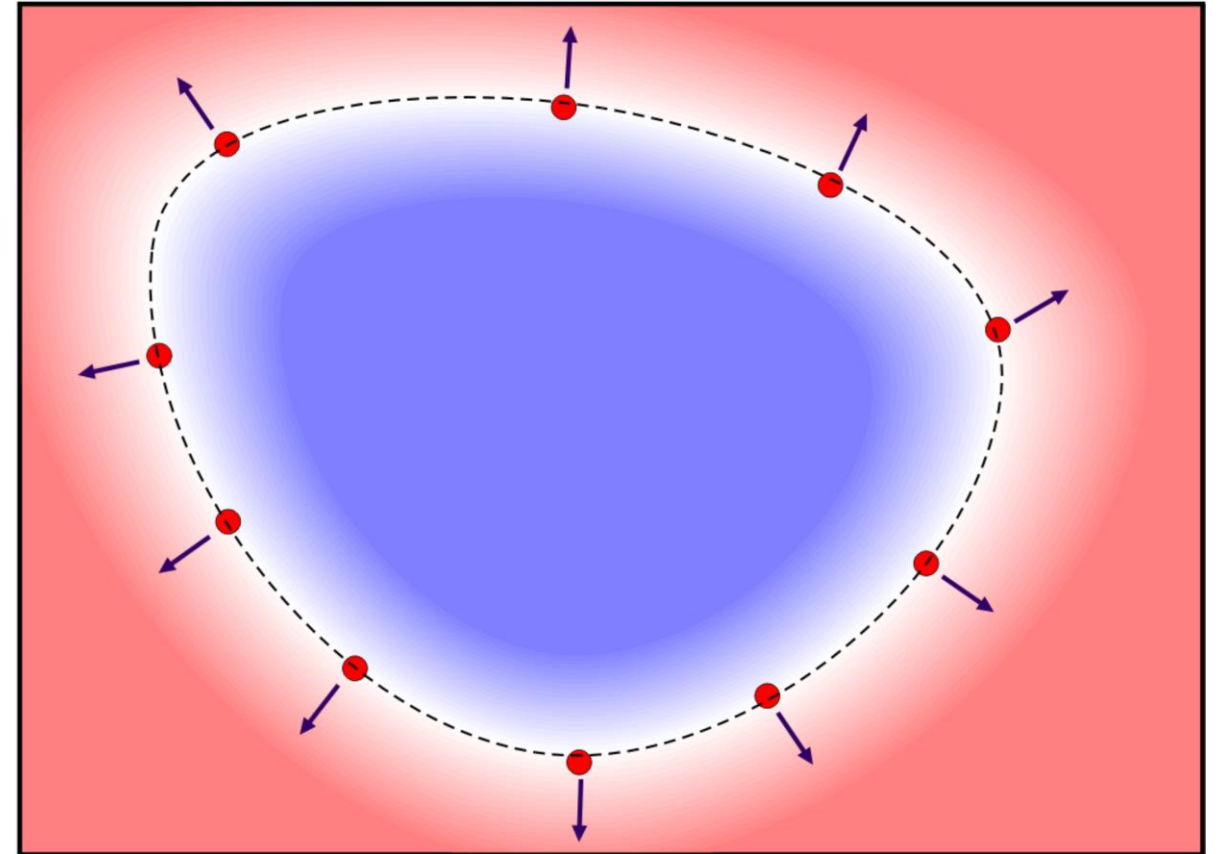
Initial data

Estimate normal

Signed distance func.

Marching cubes

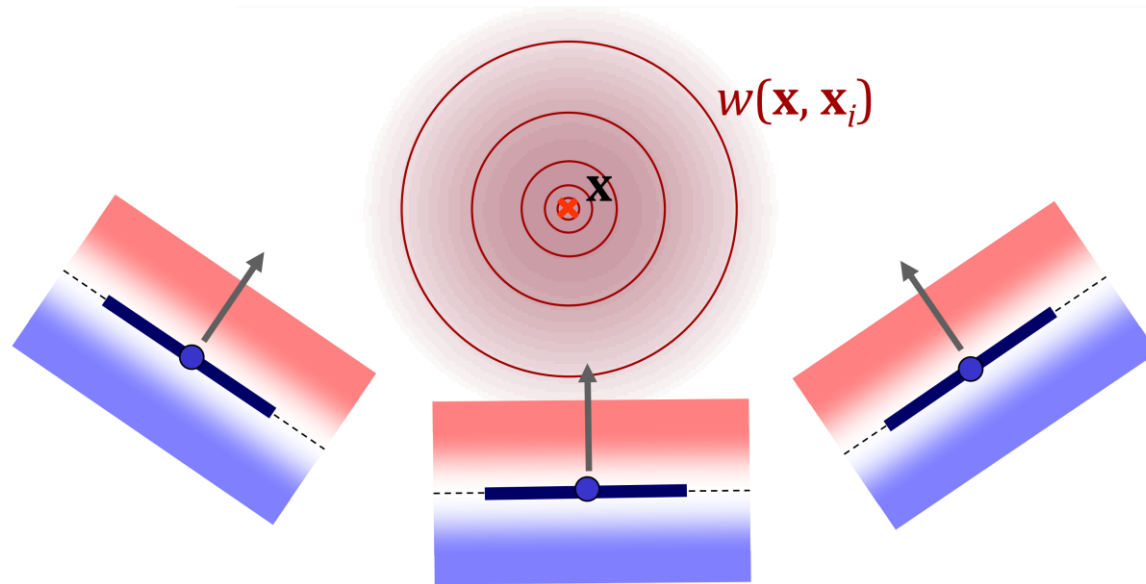
Final mesh



signed distance function:
plane blending (next slide)

Normal Constraints

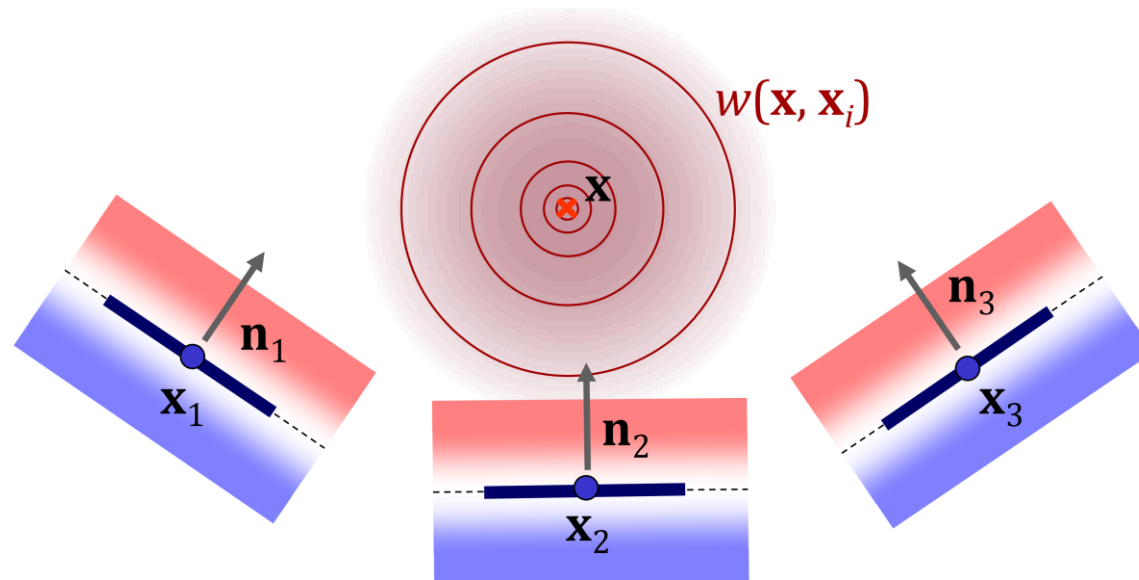
Basic Idea:



- Each point defines an oriented plane and a signed distance function
- To obtain a composite distance field in space:
Blend these distance functions with weights from a kernel (Gaussian, or uniform B-Spline)

Normal Constraints

Basic Idea:



$$f(\mathbf{x}) = \frac{\sum_{i=1}^n \langle \mathbf{n}_i, \mathbf{x} - \mathbf{x}_i \rangle w(\|\mathbf{x} - \mathbf{x}_i\|)}{\sum_{i=1}^n w(\|\mathbf{x} - \mathbf{x}_i\|)} \quad (\text{partition of unity weights})$$

Plane Blending Method

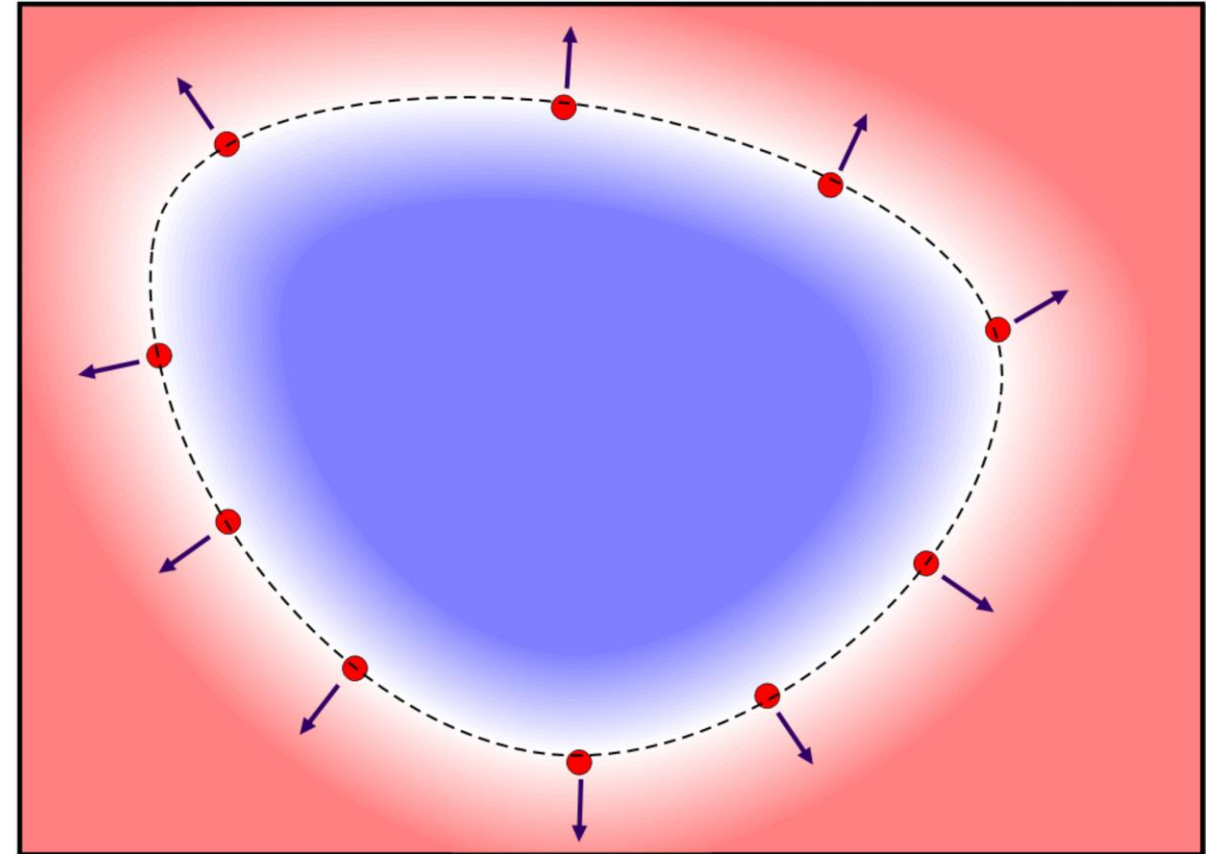
Initial data

Estimate normal

Signed distance func.

Marching cubes

Final mesh



Plane Blending Method

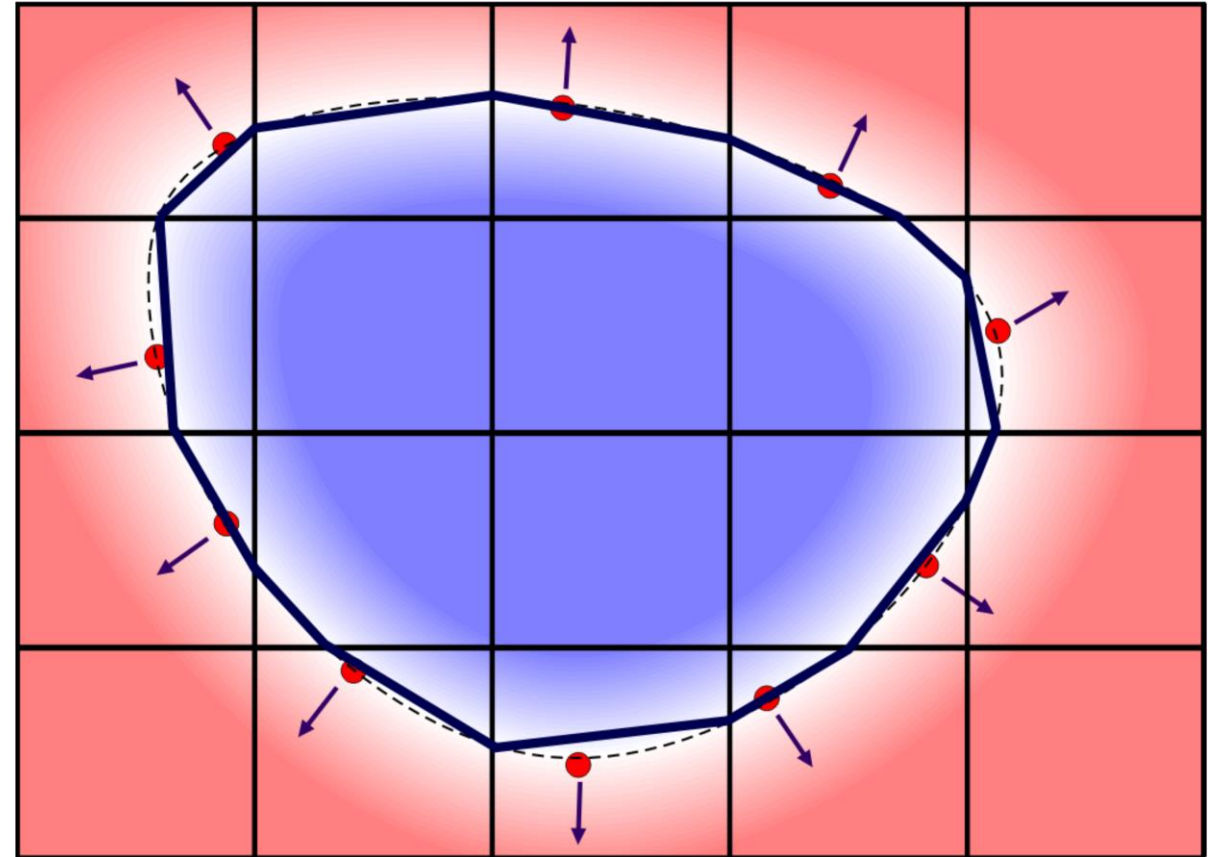
Initial data

Estimate normal

Signed distance func.

Marching cubes

Final mesh



Plane Blending Method

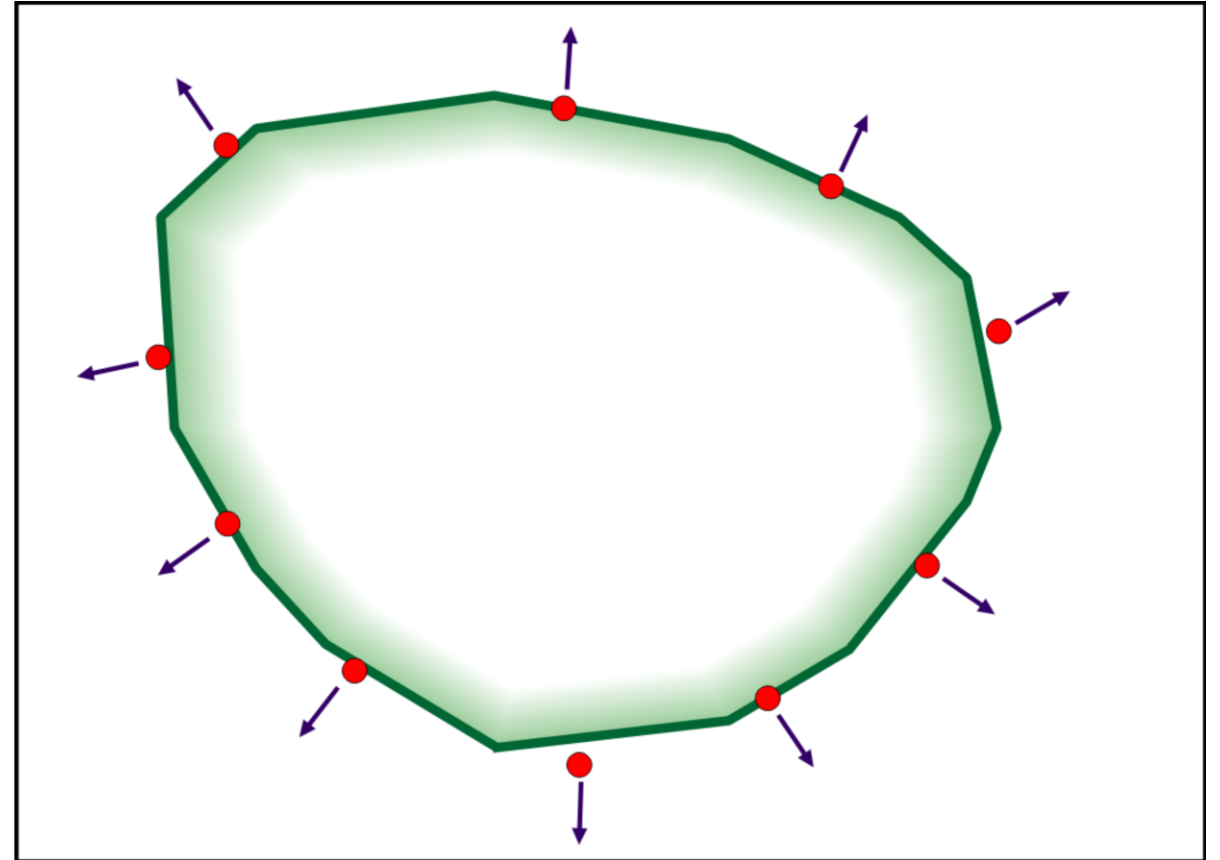
Initial data

Estimate normal

Signed distance func.

Marching cubes

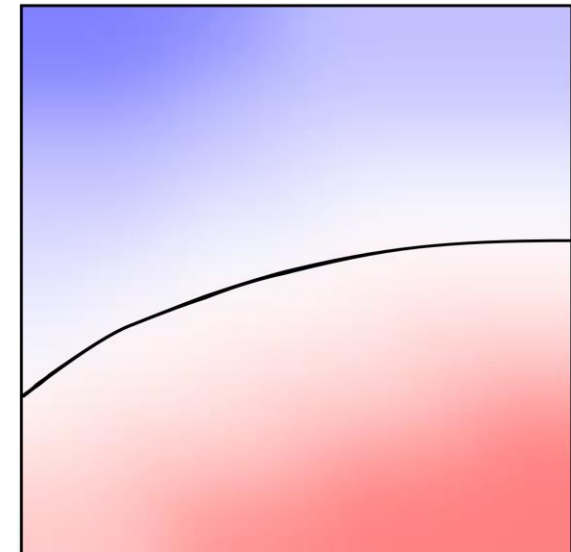
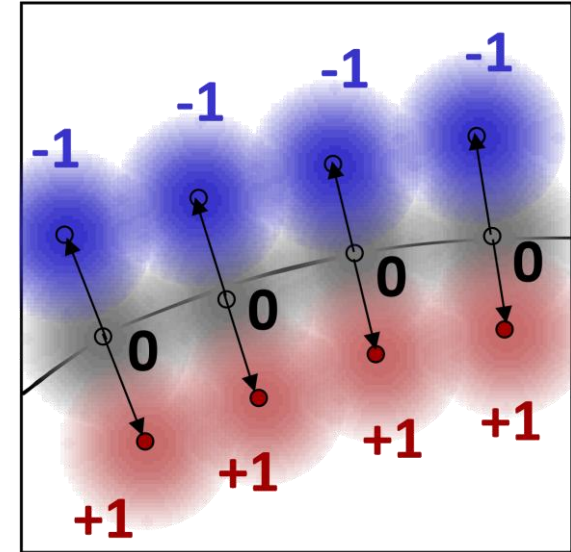
Final mesh



Thin-Plate Spline Data Matching

Agenda:

- Use radial basis functions
- Use a globally supported basis that guarantees smoothness
- Place radial basis functions at the input points
- Place two more in normal and negative normal direction
- Prescribe values $+1, 0, -1$
- Solve a linear system to meet these constraints

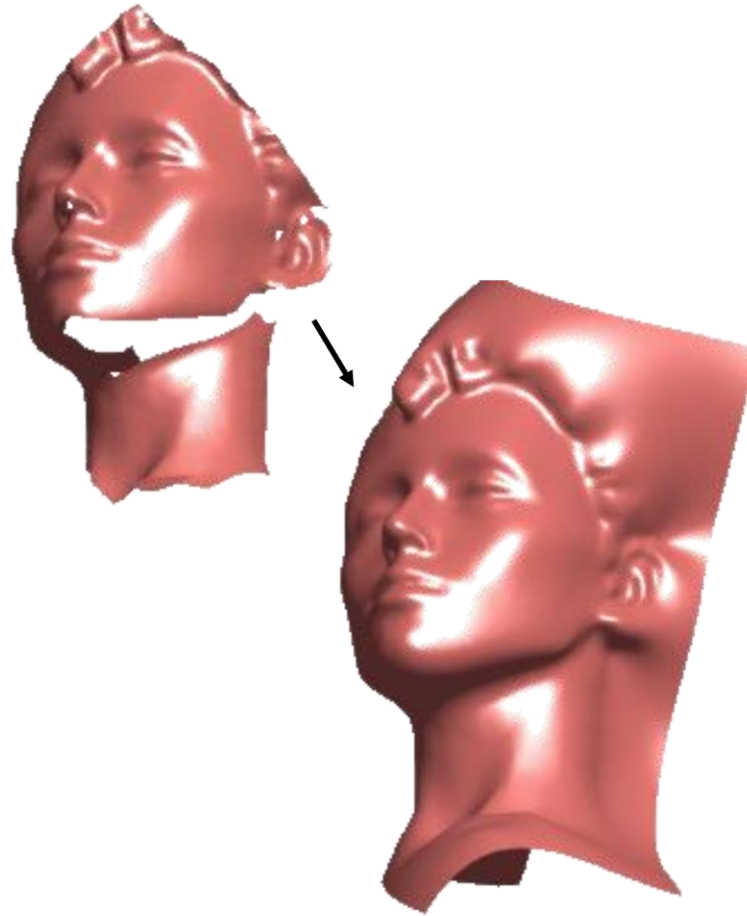
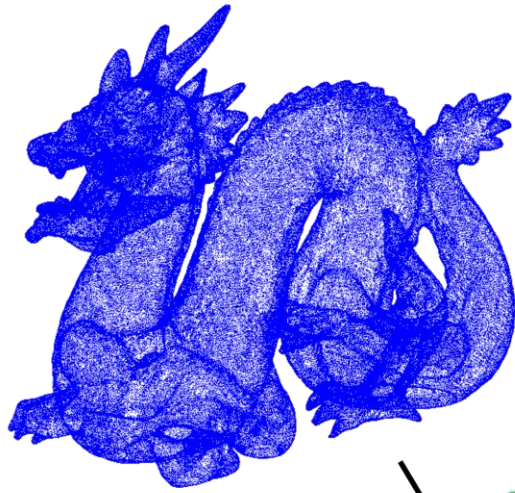


Types of Radial Basis Functions

Typical choices for radial basis functions:

- Globally supported functions:
 - Thin plate spline basis functions:
 $\|x - x_0\|^2 \ln \|x - x_0\|$ (2D), $\|x - x_0\|^3$ (3D)
 - These functions guarantee minimal integral second derivatives
- Problem: evaluation
 - Every basis function interacts with each other one
 - This creates a dense $n \times n$ linear system
 - One can use a fast multi pole method that clusters far away nodes in bigger octree boxes
 - This gives $O(\log n)$ interactions per particle, overall $O(n \log n)$ interactions

Examples



Carr et al. Reconstruction and representation of 3D objects with Radial Basis Functions, SIGGRAPH 2001

Alternative

Alternative:

- Use locally supported basis functions (e.g. B-Splines)
- Employ an additional regularization term to make the solution smooth.
- Optimize the energy function

$$E(\lambda) = \sum_{i=1}^n f(\mathbf{x}_i)^2 + \mu \int_{\Omega} \left(\left[\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} + \frac{2\partial^2}{\partial x \partial y} + \frac{2\partial^2}{\partial y \partial z} + \frac{2\partial^2}{\partial x \partial z} \right] f(\mathbf{x}) \right)^2 d\mathbf{x}$$

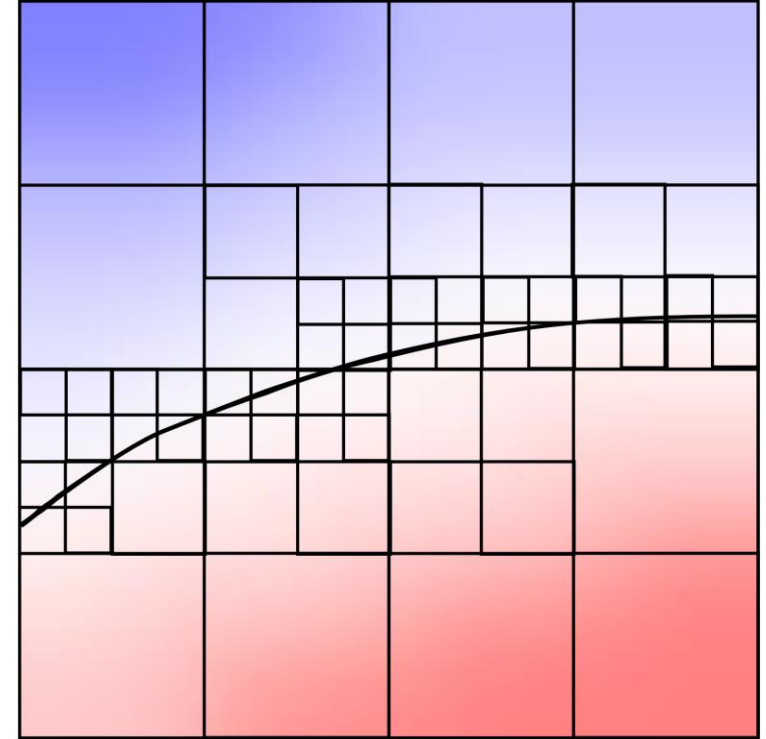
with $f(\mathbf{x}) = \sum_{i=1}^m \lambda_i b(\mathbf{x} - \mathbf{x}_i)$

- The critical point is the solution to a linear system

MPU Implicits

Multi-level partition of unity implicits:

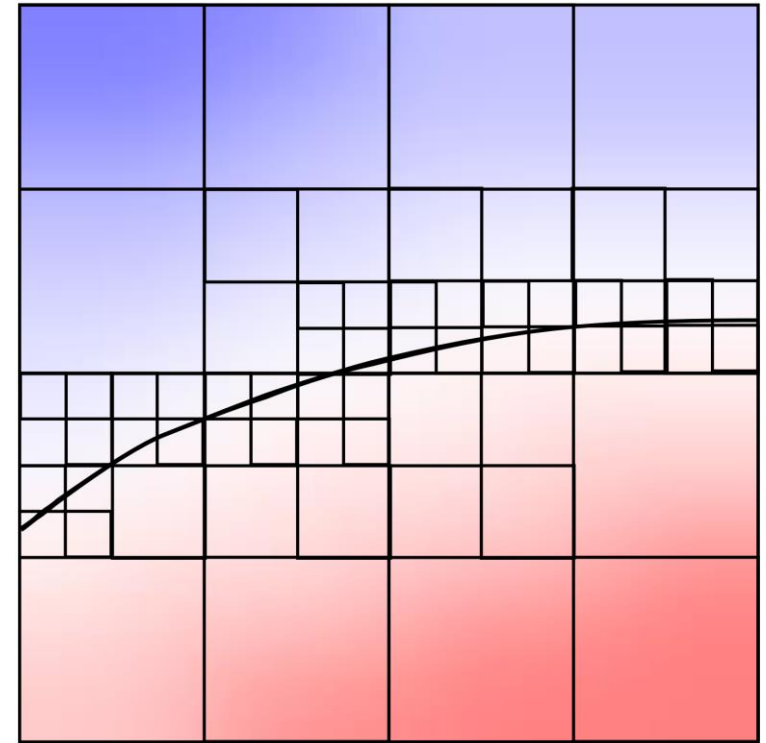
- Hierarchical implicit function approximation
 - Given: data points with normal
 - Computes: hierarchical approximation of the signed distance function



MPU Implicits

Multi-level partition of unity implicits:

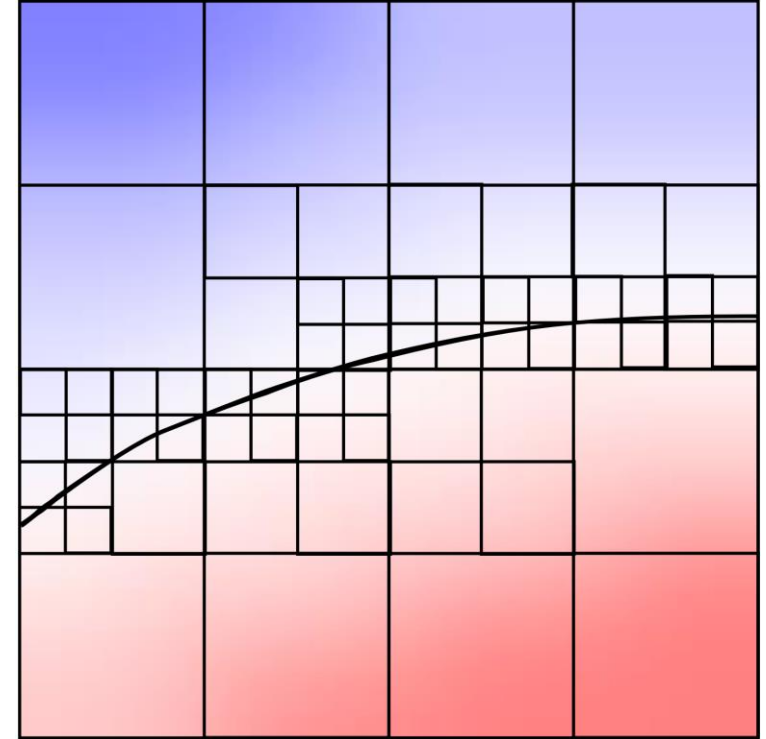
- Octree decomposition of space
- In each octree cell, fit an implicit quadratic function to points
 - $f(\mathbf{x}_i) = 0$ at data points
 - Additional normal constraints
- Stopping criterion:
 - Sufficient approximation accuracy
(evaluate f at data points to calculate distance)
 - At least 15 points per cell.



MPU Implicits

Multi-level partition of unity implicits:

- This gives an adaptive grid of local implicit function approximations
- Problem: How to define a global implicit function?
- Idea: Just blend between local approximants using a windowing function

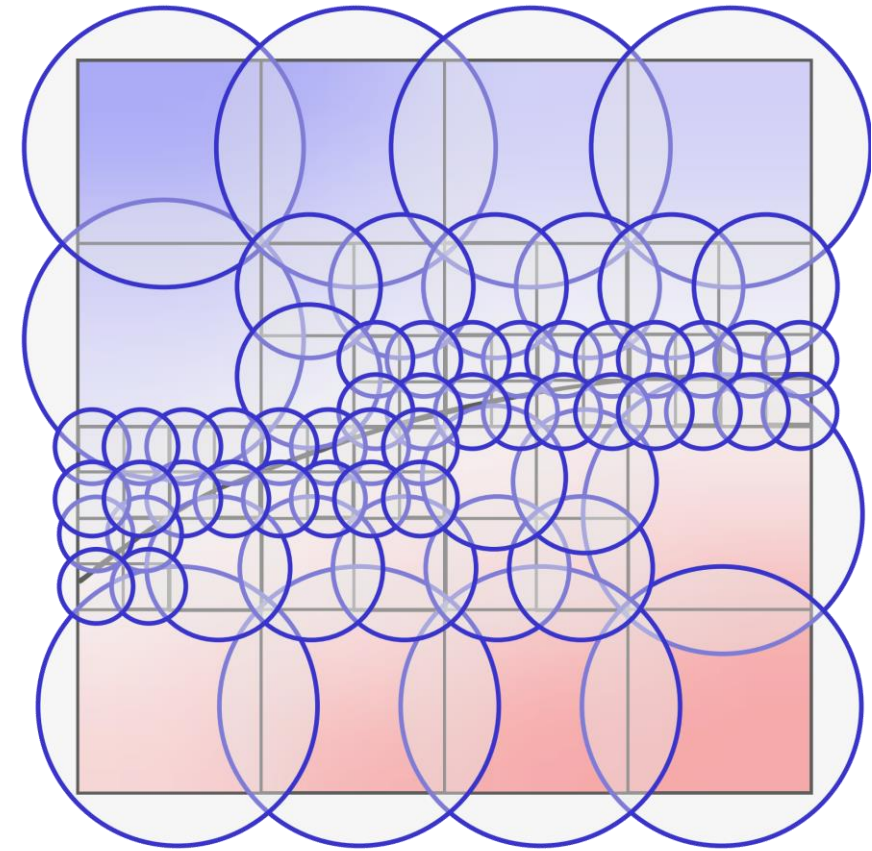


MPU Implicits

Multi-level partition of unity implicits:

- Windowing function:
 - Use smooth windowing function w
 - B-splines / normal distribution
 - Original formulation: quadratic tensor product B-spline function, support = $1.5 \times$ cell diagonal
 - Renormalize to form partition of unity:

$$f(\mathbf{x}) = \frac{\sum_{i=1}^n w(\mathbf{x} - \mathbf{x}_i) f_i(\mathbf{x})}{\sum_{i=1}^n w(\mathbf{x} - \mathbf{x}_i)}$$

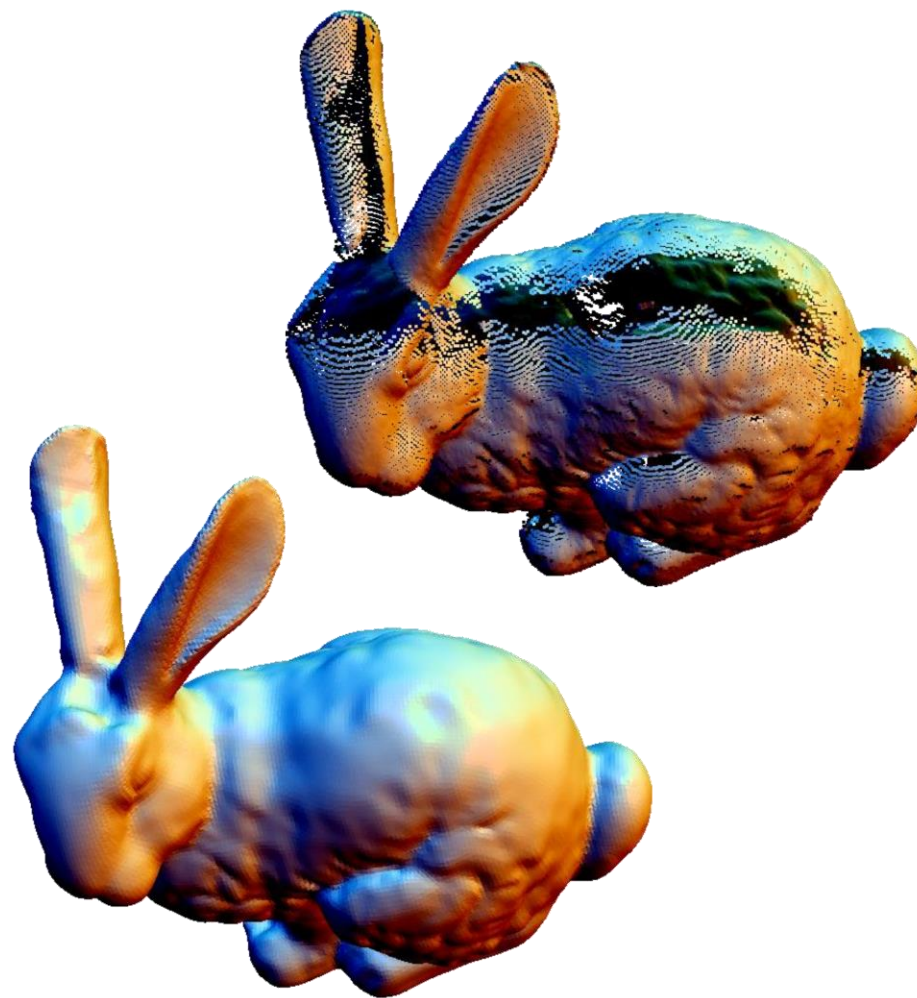


MPU Implicits

Multi-level partition of unity implicits:

- Sharp features:
 - If a leaf cell with a few points has strongly varying normal, this might be a sharp feature.
 - Multiple functions can be fitted to parts of the data
 - Boolean operations to obtain composite distance field

Examples



Ohtake et al. Multi-level Partition of
Unity Implicits, SIGGRAPH 2003

Computer Aided Geometric Design

Fall Semester 2024

Parameterization

陈仁杰

renjiec@ustc.edu.cn

<http://staff.ustc.edu.cn/~renjiec>

Subdivision Surfaces

Problem with Spline Patches

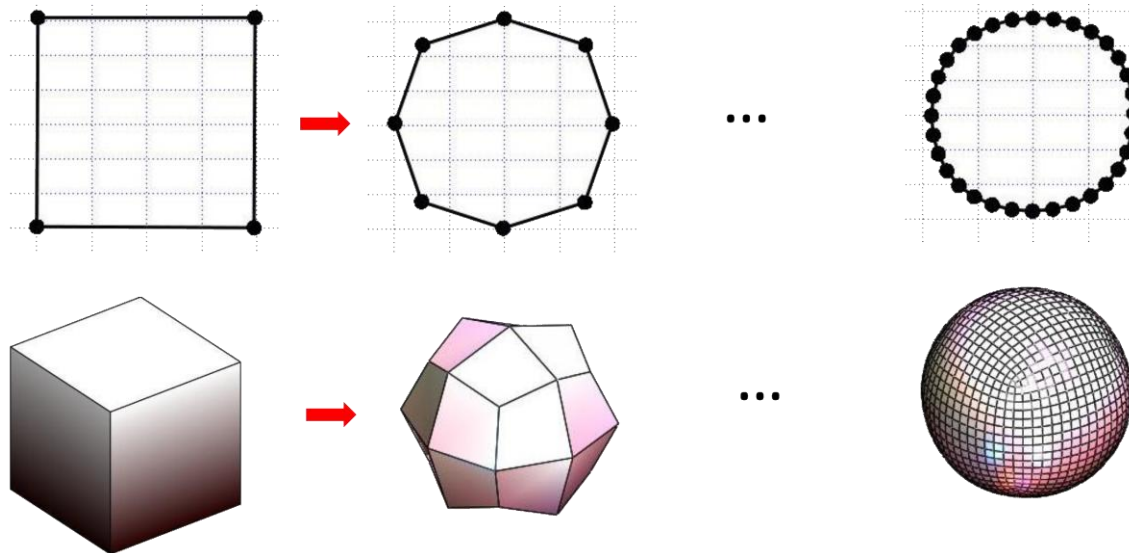
- A continuous tensor product spline surface is only defined on a regular grid of quads as parametrization domain
- Thus, the topology of the object is restricted
- Assembling multiple parameter domains to a single surface is tedious, hard to get continuity guarantees
- Handling trimming curves is not that straightforward

Question: can we do better?

Subdivision Surfaces

Wish list:

- Provide a very coarse representation of the geometry
- Obtain a fine and smooth representation
- Preferably by means of a simple set of rules which can be recursively applied (subdivision rules or subdivision scheme)



Subdivision Surfaces

Bigger goals:

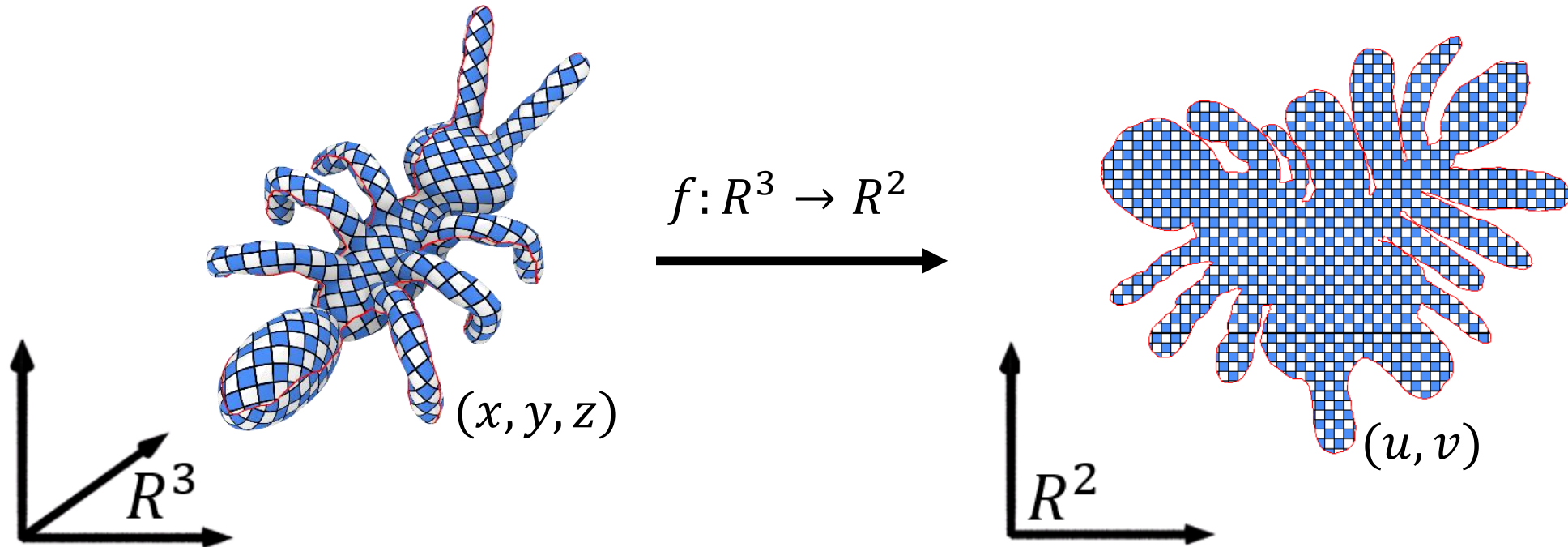
- Simplify the creation of smooth refined geometric models (especially in feature film industry)



- What's lost? Parametric representation ...

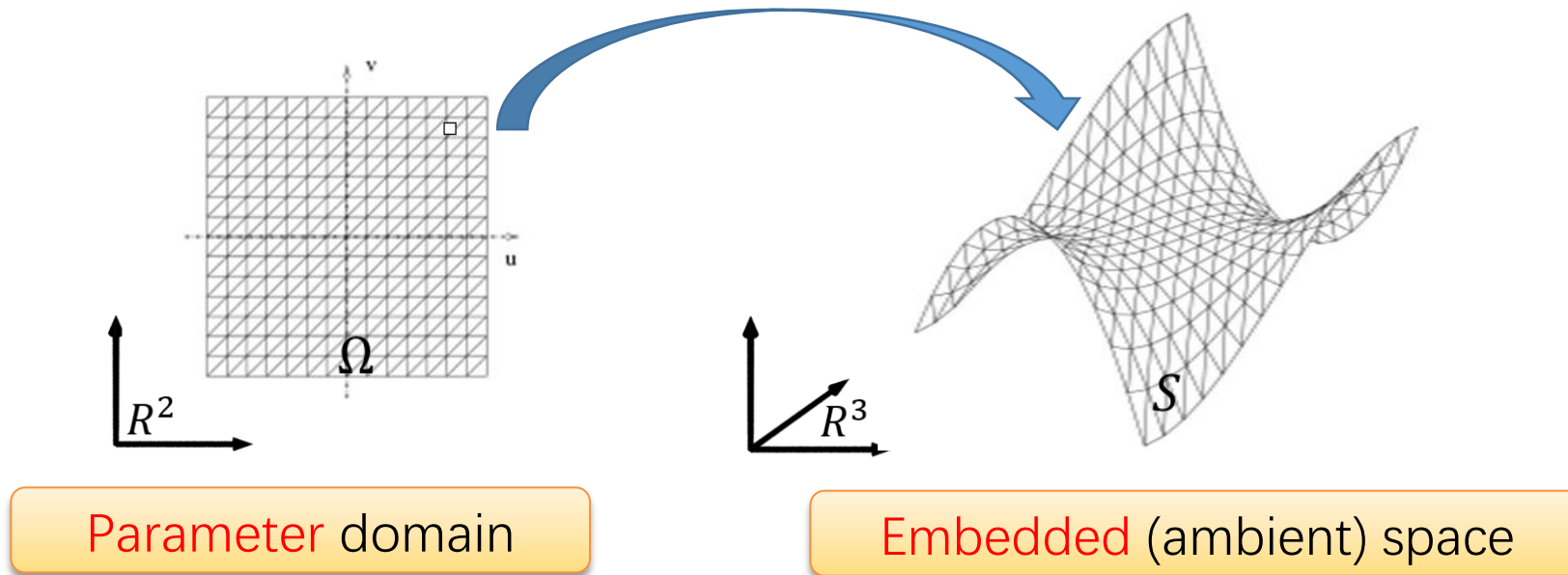
Surface Parameterization

- Geometric intuition: 3D surface unfolds into 2D plane
- Mathematical essence: embedding/mapping of 3D surface in 2D plane
 - Construct one-to-one correspondence between surface and plane area
 - 3D surface is essentially 2D: 2D manifold

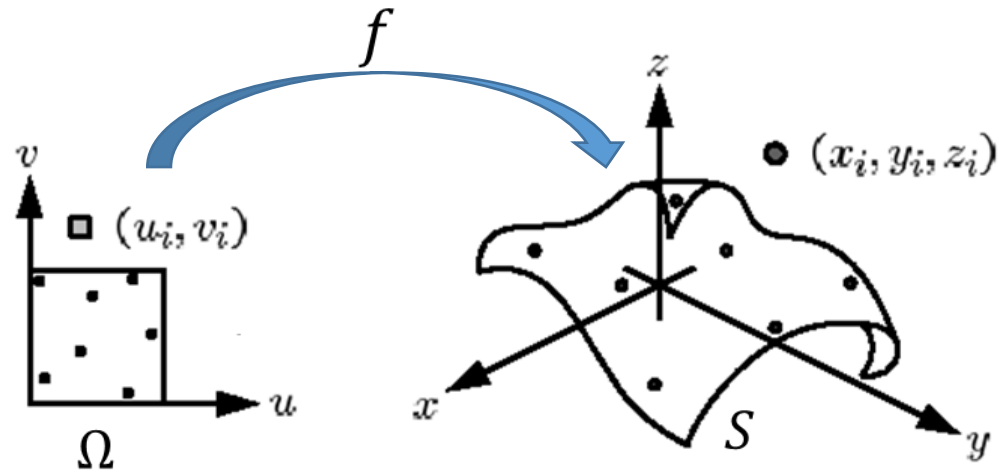


2D Manifold Surface in \mathbb{R}^3

A surface S in \mathbb{R}^3 has an **intrinsic dimension** of 2D – a patch $\Omega \in \mathbb{R}^2$ is embedded into \mathbb{R}^3 (each point in Ω is assigned a position in \mathbb{R}^3)



Parametric Surfaces

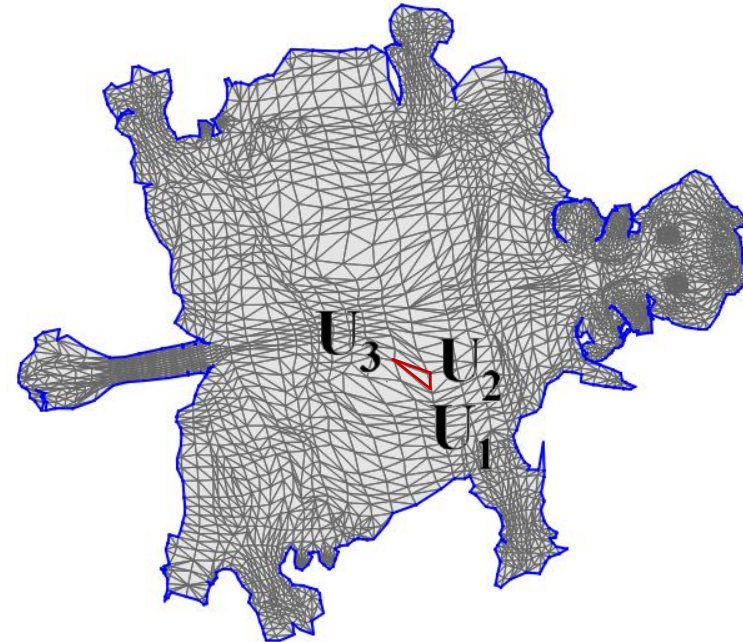
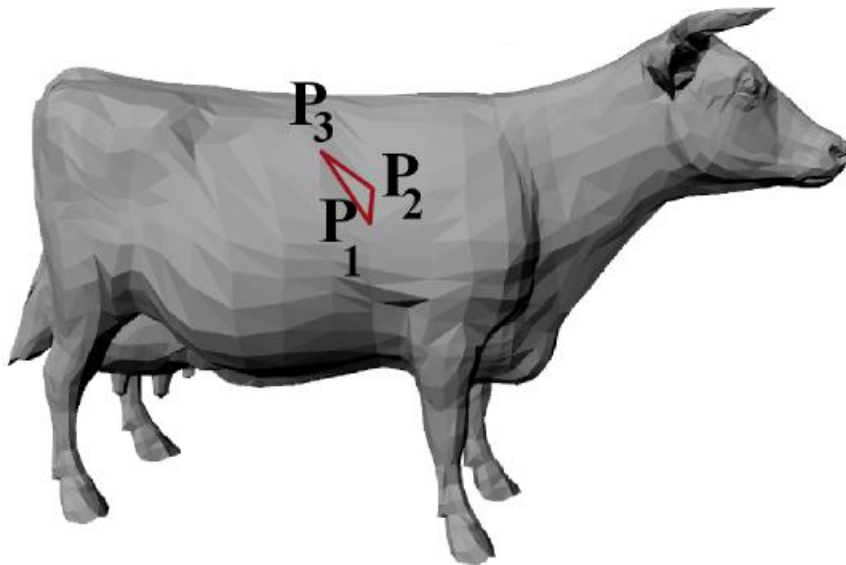


$$f: \Omega \rightarrow S$$

$$(u, v) \mapsto \begin{cases} x = x(u, v), \\ y = y(u, v), \\ z = z(u, v), \end{cases}$$

Parameterization: flatten the surface into a plane

- Each 3D vertex (x,y,z) corresponds to a 2D point (u,v)
 - (u,v) is called the parameter of (x,y,z) (intrinsic dimension of the 2D manifold surface)

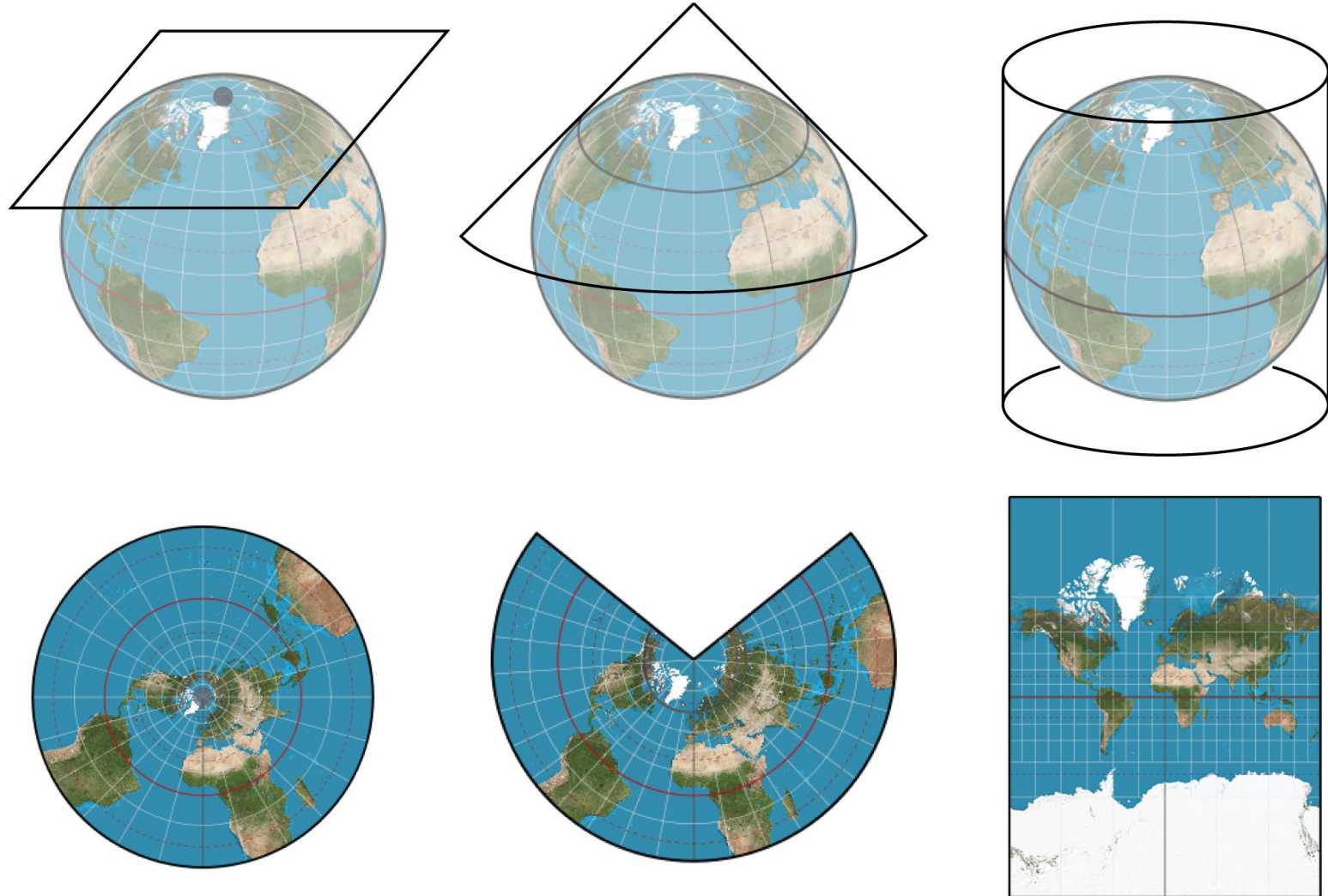


Parameterization is the most basic problem in geometric processing

- **Provides a 2D parameter for each point on a 3D surface**
 - Intrinsic dimension parameter
- **Processing high-dimensional problems in low dimensions reduces complexity**
 - Dimension reduction
- **Related problems between 3D surfaces can be processed through parameterized space**

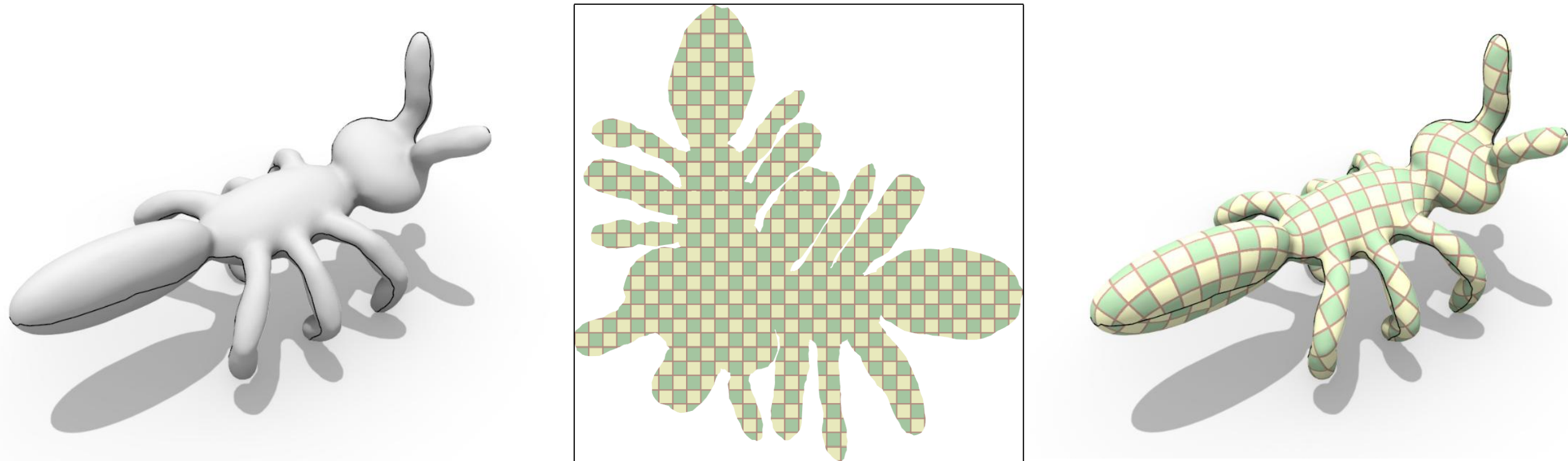
Application of surface parameterization-1

- Map making



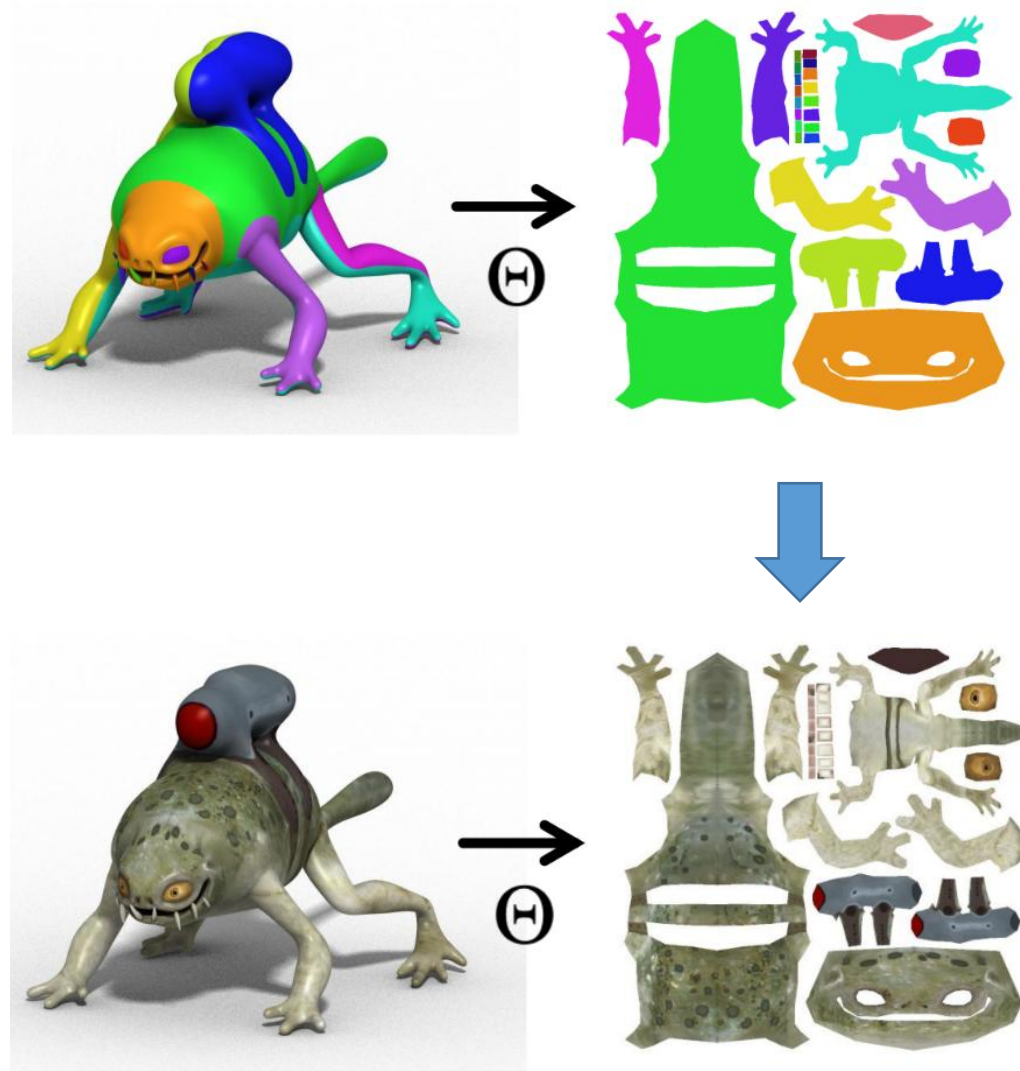
Application of surface parameterization-2

- **Surface mapping: storing and expressing various information on the surface**
 - texture mapping, normal mapping, displacement mapping, color (albedo), material



Application of surface parameterization-3

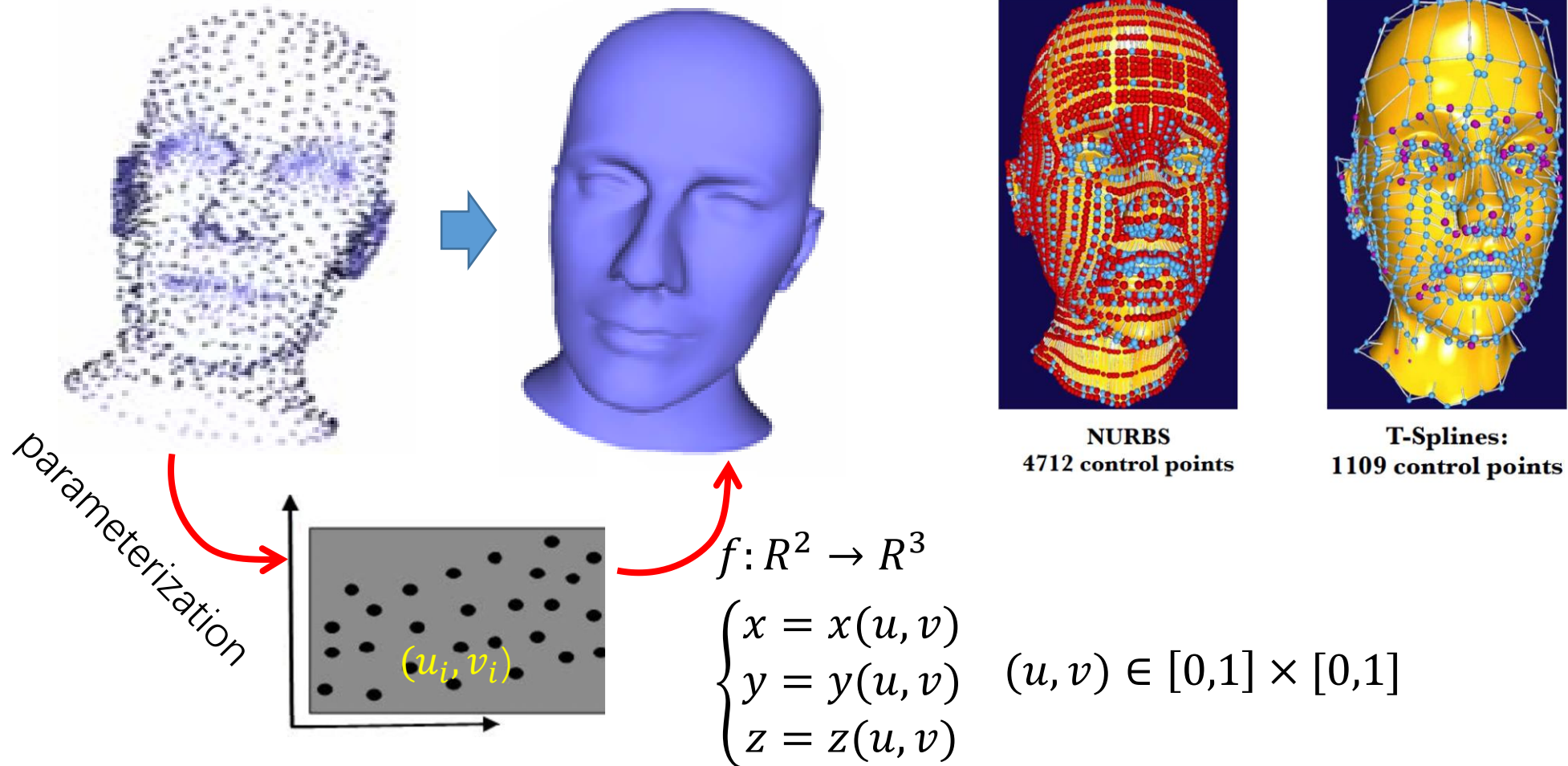
- **Texture Atlas**
 - Surface painting



Application of surface parameterization-4

- **Surface Fitting**

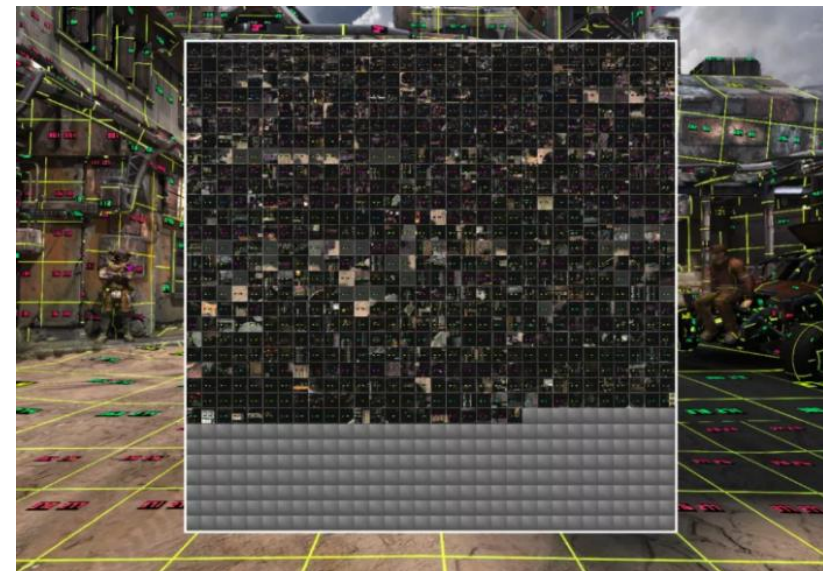
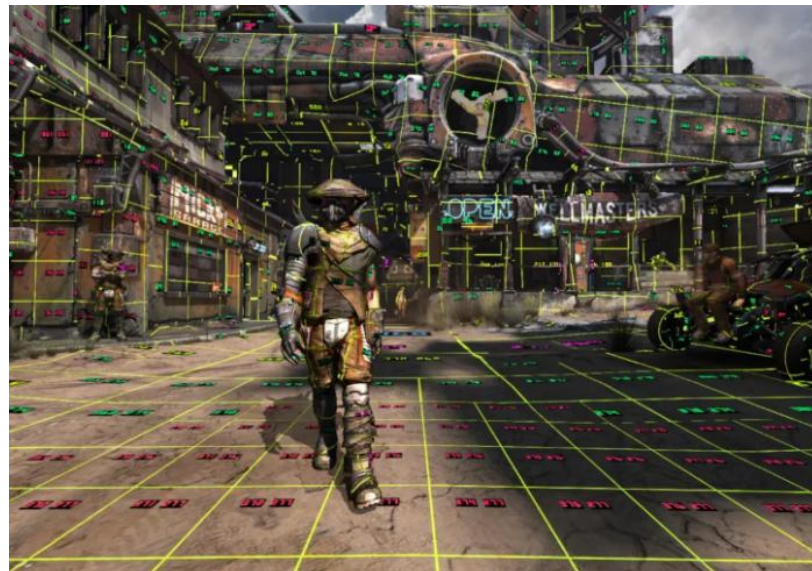
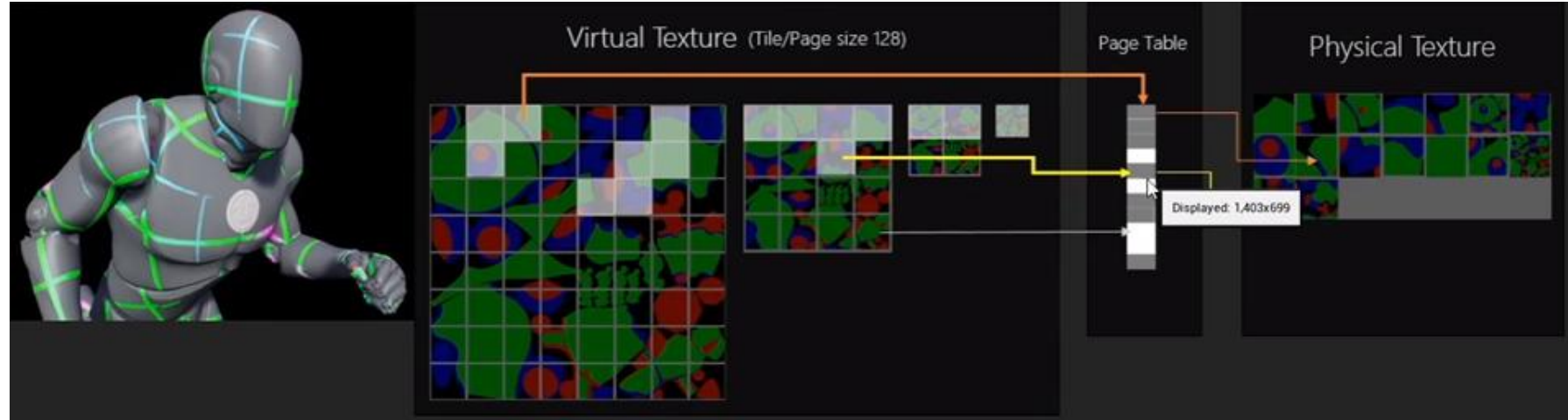
B-spline (NURBS/T-spline) surface fitting of 3D point cloud



Application of surface parameterization-5

- **Surface rendering**

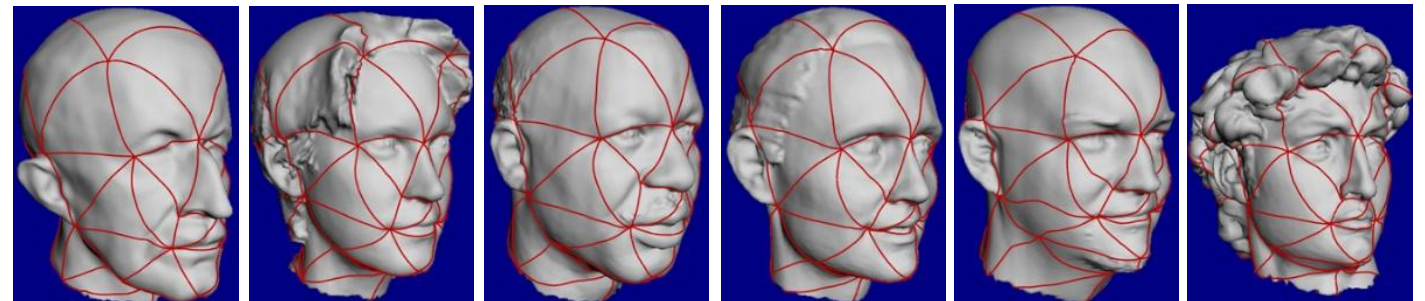
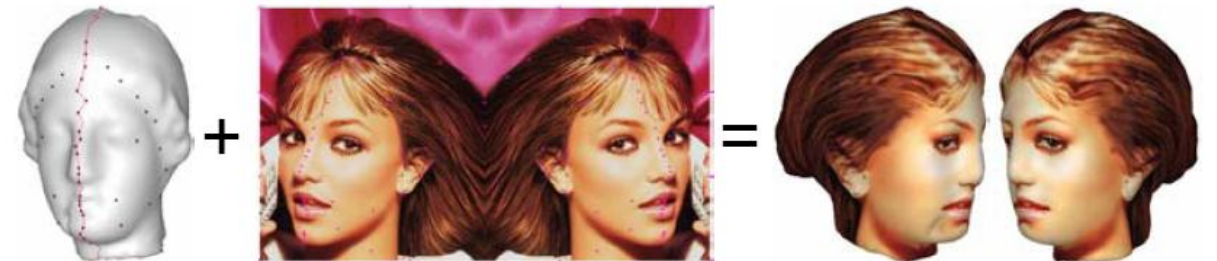
- Virtual textures
- Virtual geometry
- Mipmap
- LOD



Application of surface parameterization-6

- The basis for most geometry processing (basic problems)

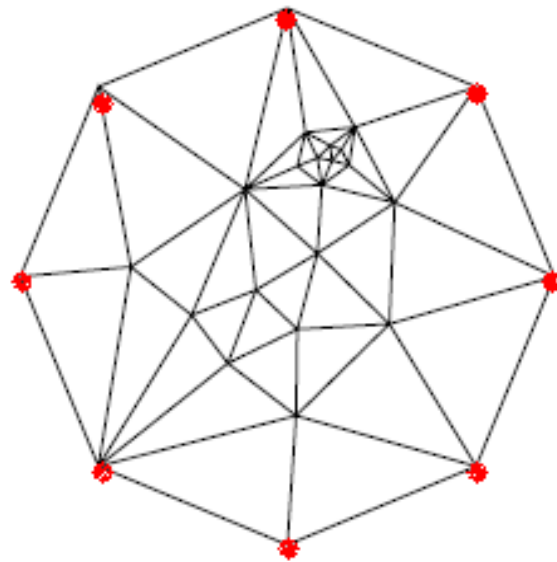
- Visualization
- Compression
- Transmission
- Simplification
- Matching
- Remeshing
- Reconstruction
- Repairing
- Texture synthesis
- Rendering
- Animation
- Morphing
- ...



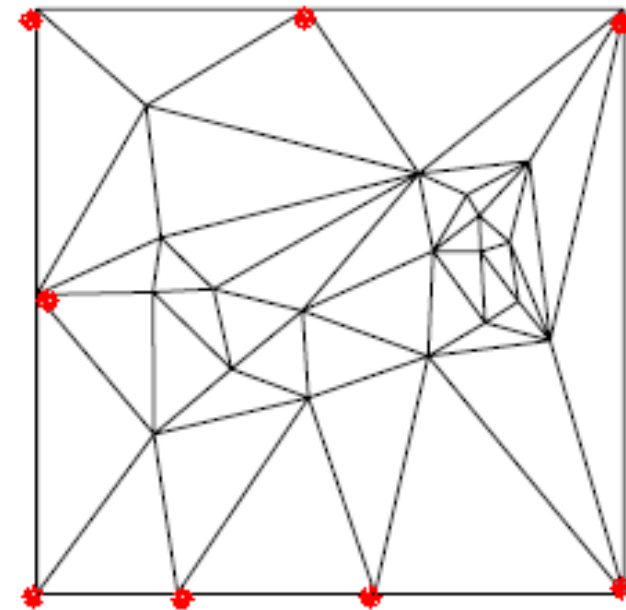
Mapping the boundary onto a **convex** polygon in the plane

[Floater97]

Fixing the boundary of the mesh onto



an unit circle

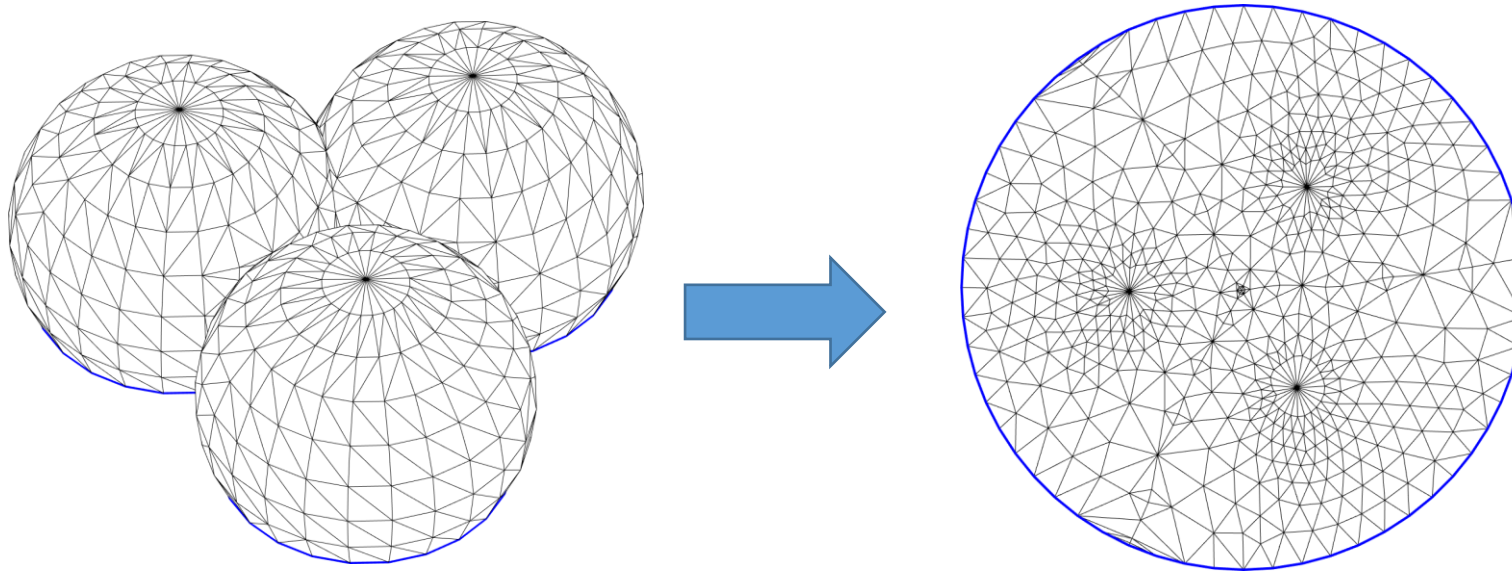


an unit square

Tutte's Method: Why it Works

Theorem [**Tutte**63], [Maxwel1864] :

- If $G = \langle V, E \rangle$ is a 3-connected planar graph (triangular mesh) then any barycentric embedding provides a **valid** parameterization



If the boundary lies on a convex polygon, the triangles in the Tutte Embedding must not be flipped!

Floater parametrization

[Floater 97']

- Uniform parametrization
- Shape-preserving parametrization
- How to judge which parameterization method is better?

GAMES Course 301: Surface parameterization

GAMES 301



曲面参数化



刘利刚 & 陈仁杰
傅孝明 & 方 清

中国科学技术大学

2022年10月8日起 | 北京时间每周六、周日上午10:00-11:30 | WEBINAR.GAMES-CN.ORG

▲ GAMES301: 曲面参数化 

<https://www.bilibili.com/video/BV18T411P7hT>