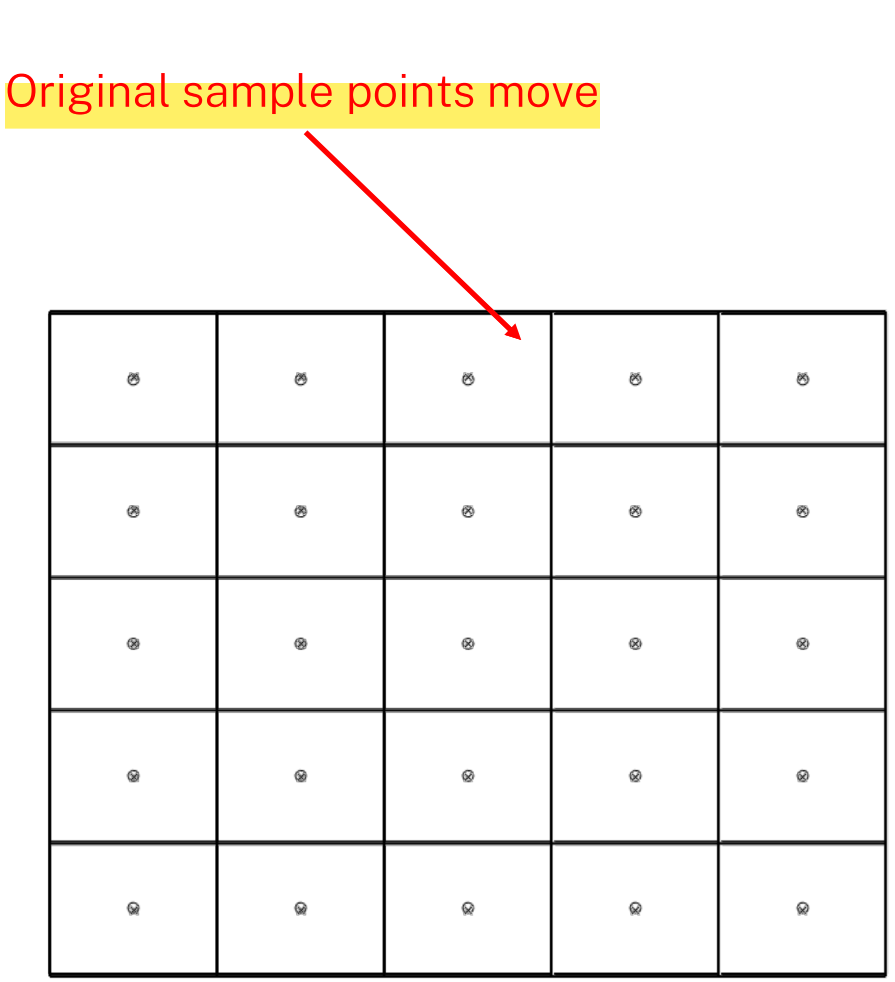
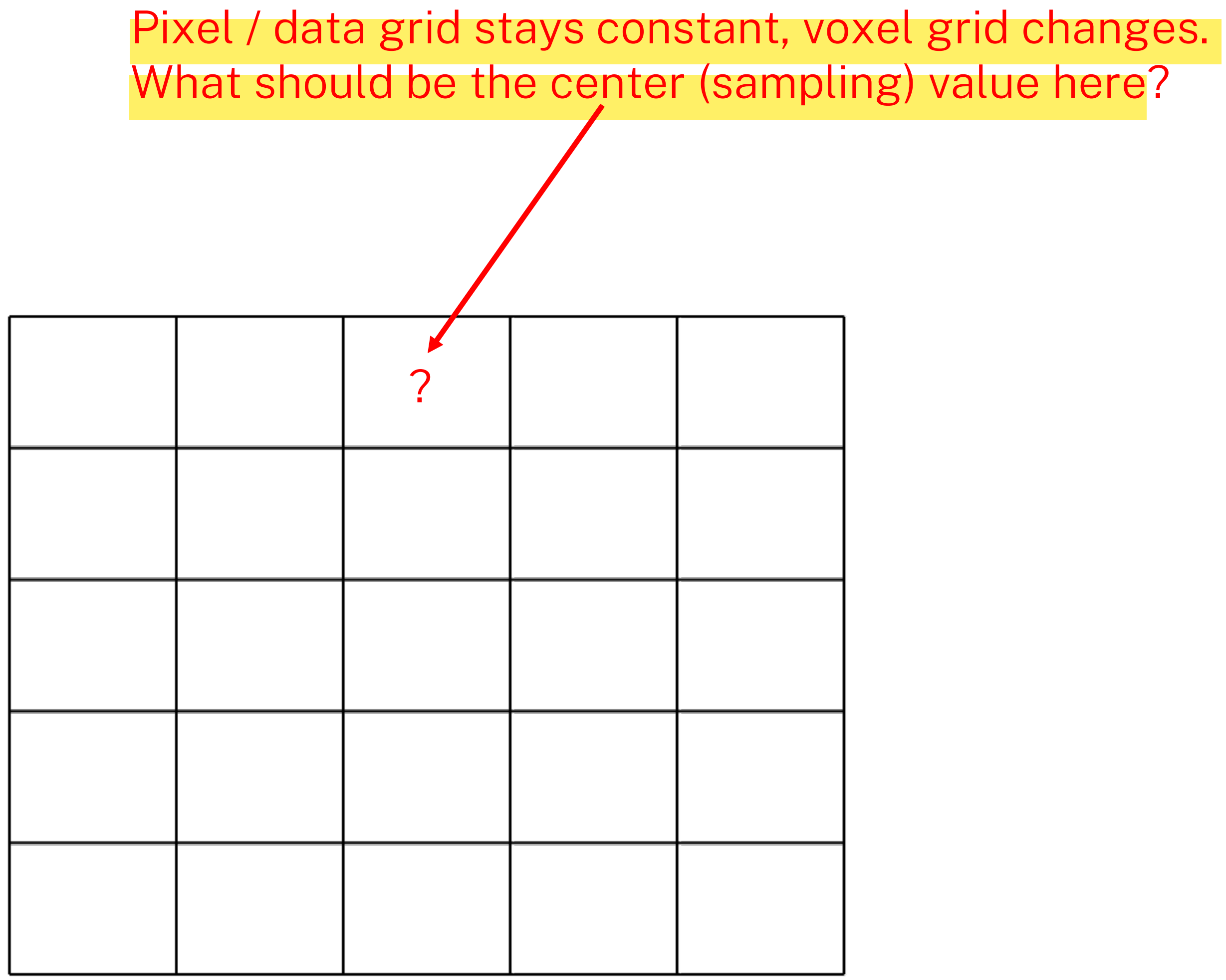
3.6 Interpolation

Interpolation

What happens when an image is rotated?



Input Image

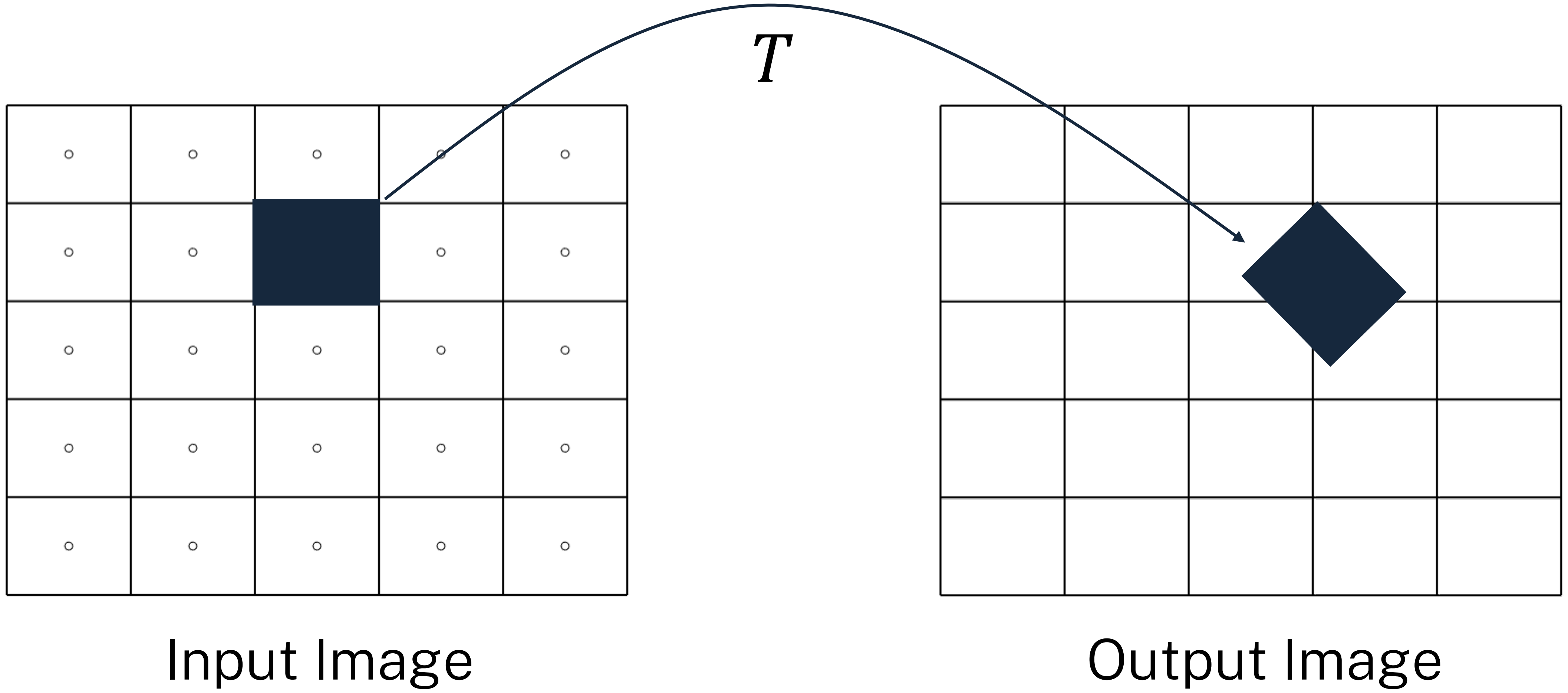


Output Image

Interpolation

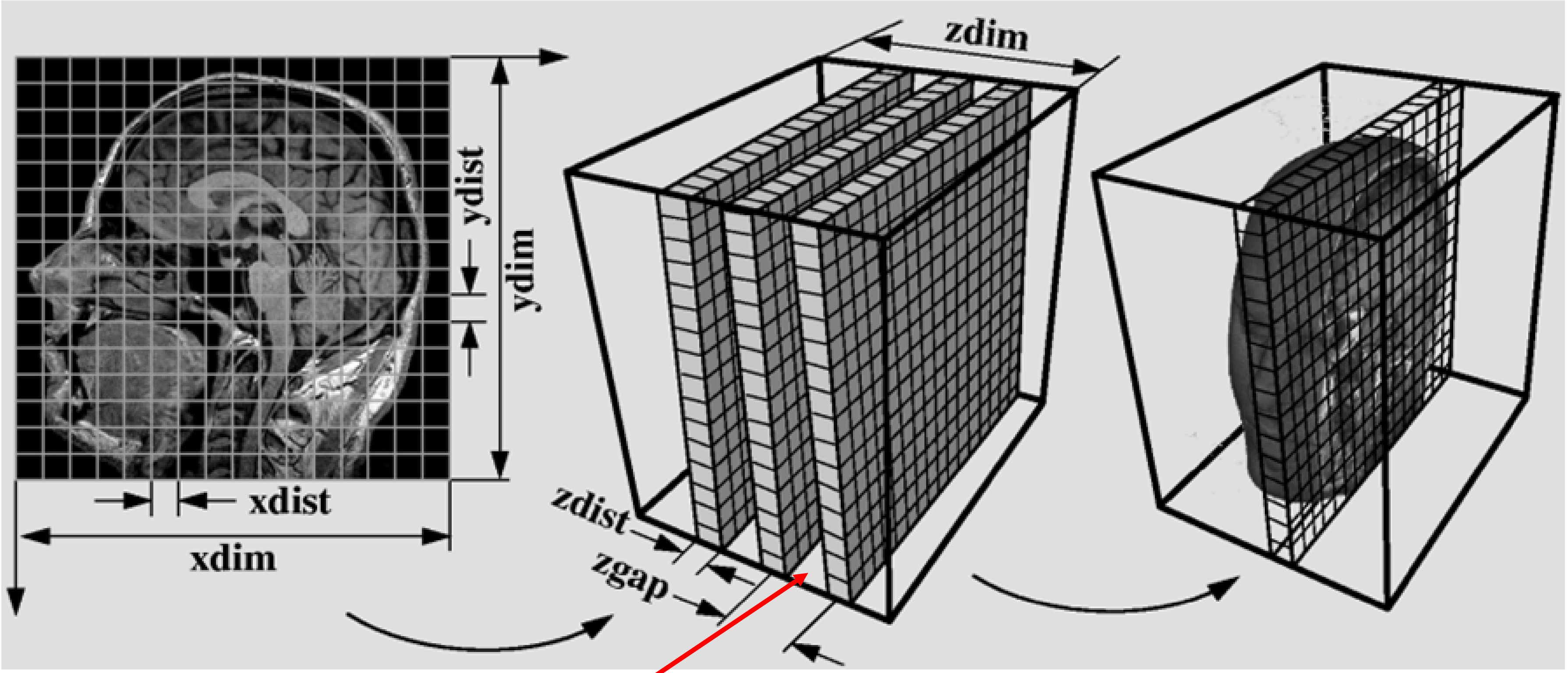
Resulting cells do not match the required sampling grid

- Resampling is required
  - Determine cells (and data values) on the output grid



Interpolation

- When constructing orthogonal or arbitrary orientations from a dataset there might be no data at all
  - This is also true when, e.g. enlarging an image or at the corners after rotation



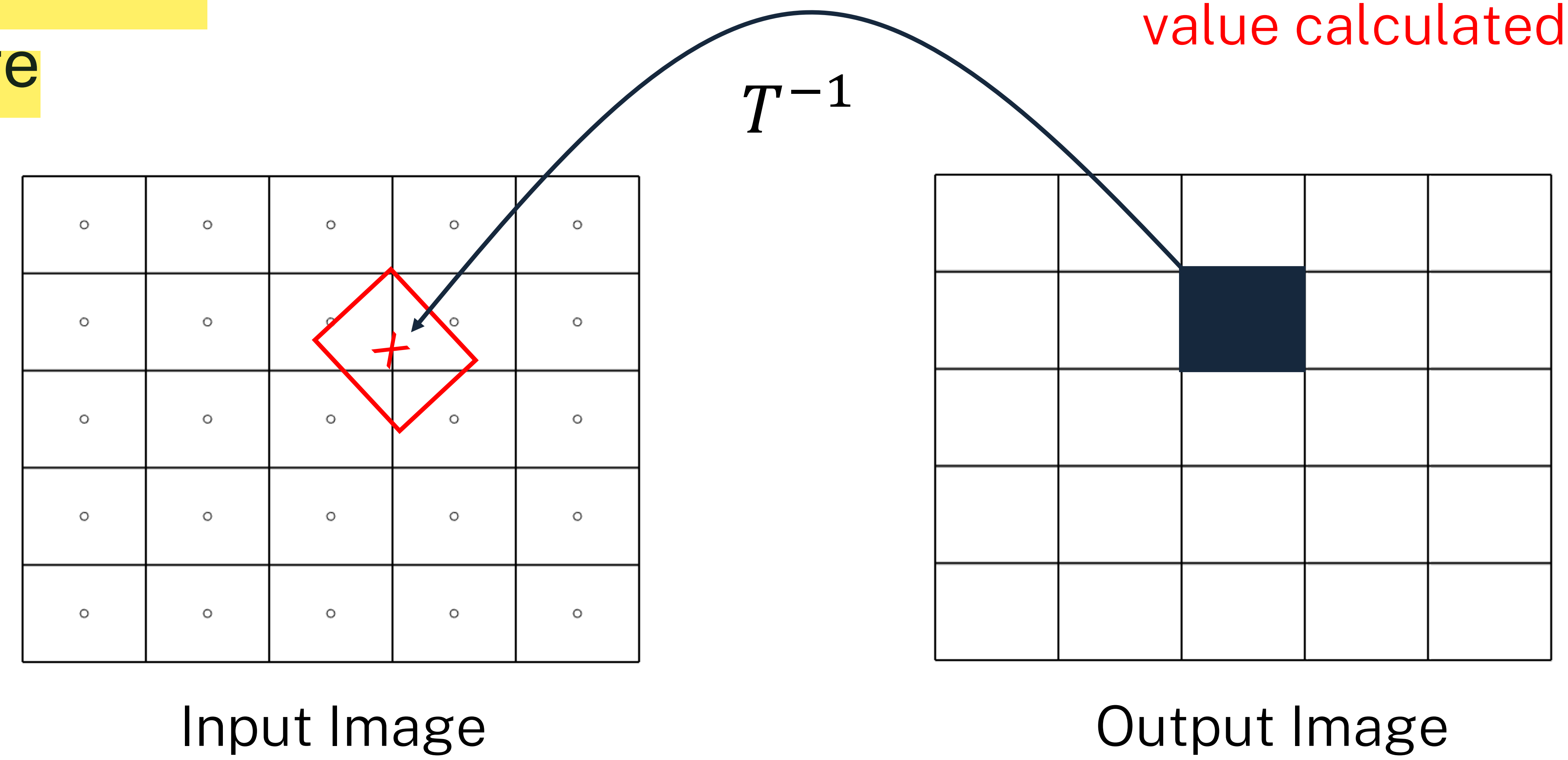
No sample data in the gap!

Interpolation

Resampling

- Traverse **output grid** and calculate cell location in original image by transforming with T^{-1}
  - Calculate cell center value
  - Apply as data value

How is the output value calculated?

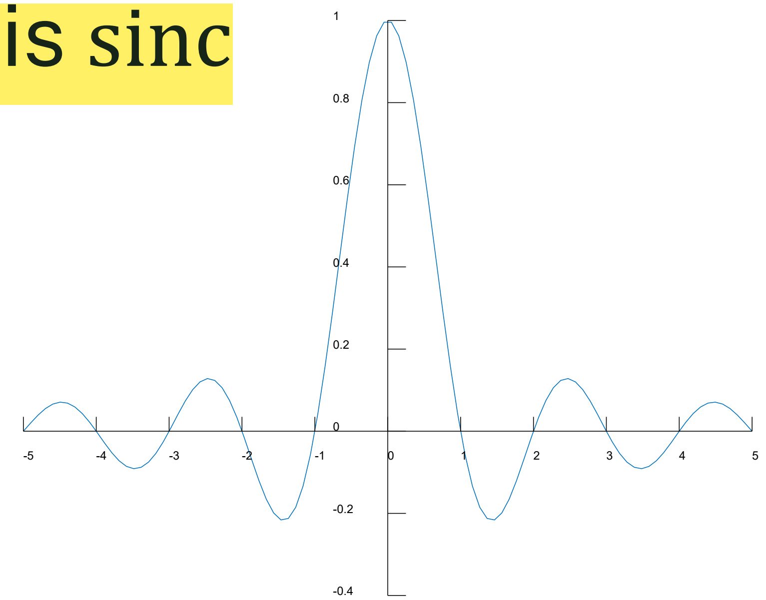


ohm

Interpolation

- Problem
  - (Image) function known only at discrete sampling points
  - Function values required in-between
  - Analytical description unknown / impossible to construct
- Interpolation rekonstruieren der Funktion aus diskreten sampling points
  - Estimate values of unknown function in-between given sampling points
  - The ideal interpolation function for a continuous signal is sinc

$$\mathrm{sinc}\,x = \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{-j\omega x} d\omega = \begin{cases} \frac{\sin \pi x}{\pi x} & , x \neq 0 \\ 1 & , x = 0 \end{cases}$$



  - Problem: sinc requires infinitely many samples for perfect interpolation

Interpolation

Classification

- Local interpolation methods (cell-wise)
  - Use information stored in the neighbor nodes (vertices)!

Nachbar, Ecken, Gitter, ... betrachten

- Global interpolation methods
  - Use information of all vertices or a *broader* neighborhood

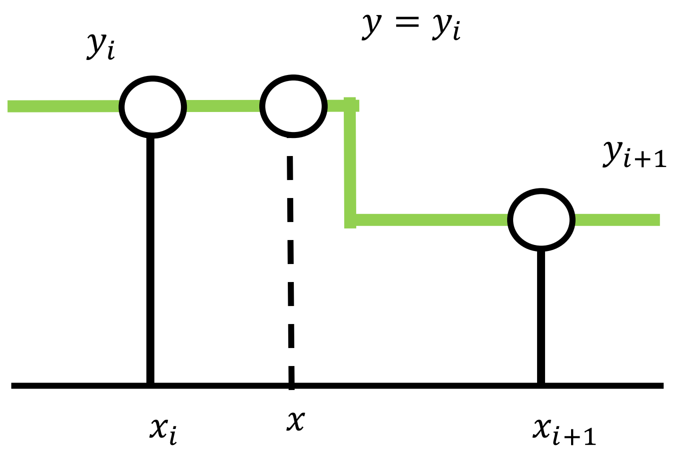
3.6.1 Local Interpolation

Local Interpolation

Simple approximations

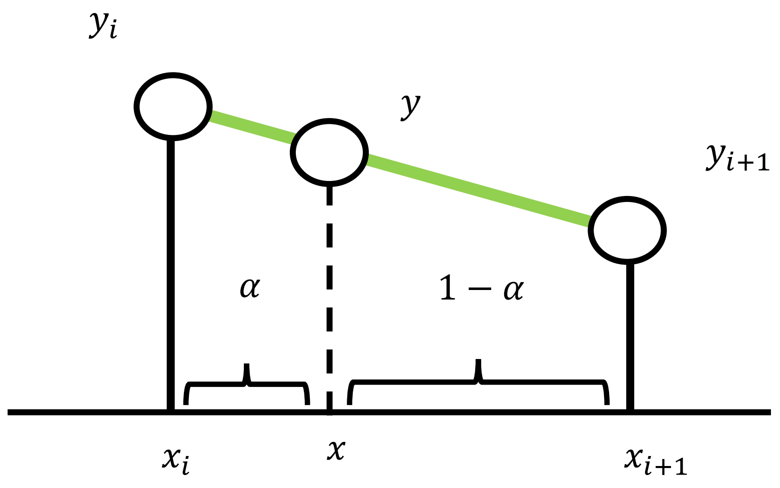
Abstand zum nahestehen Nachbar

- **Nearest-neighbour**
  - Very coarse approximation
  - Low quality interpolant
  - Fast calculation



$$f(x) = y = \begin{cases} y_i, & \dfrac{x - x_i}{x_{i+1} - x_i} < \dfrac{1}{2} \\ y_{i+1}, & \dfrac{x - x_i}{x_{i+1} - x_i} \geq \dfrac{1}{2} \end{cases}$$

- **Linear interpolation**
  - Assume linear relationship between samples
  - Quality ok, esp. on dense sampling grids
  - Fast calculation
  - Also known as linear blending
    - "Default" on graphics hardware
    - Shaders allow more sophisticated interpolation kernels



$$f(x) = y = \alpha y_{i+1} + (1 - \alpha) y_i$$

with $\alpha = \dfrac{x - x_i}{x_{i+1} - x_i}$

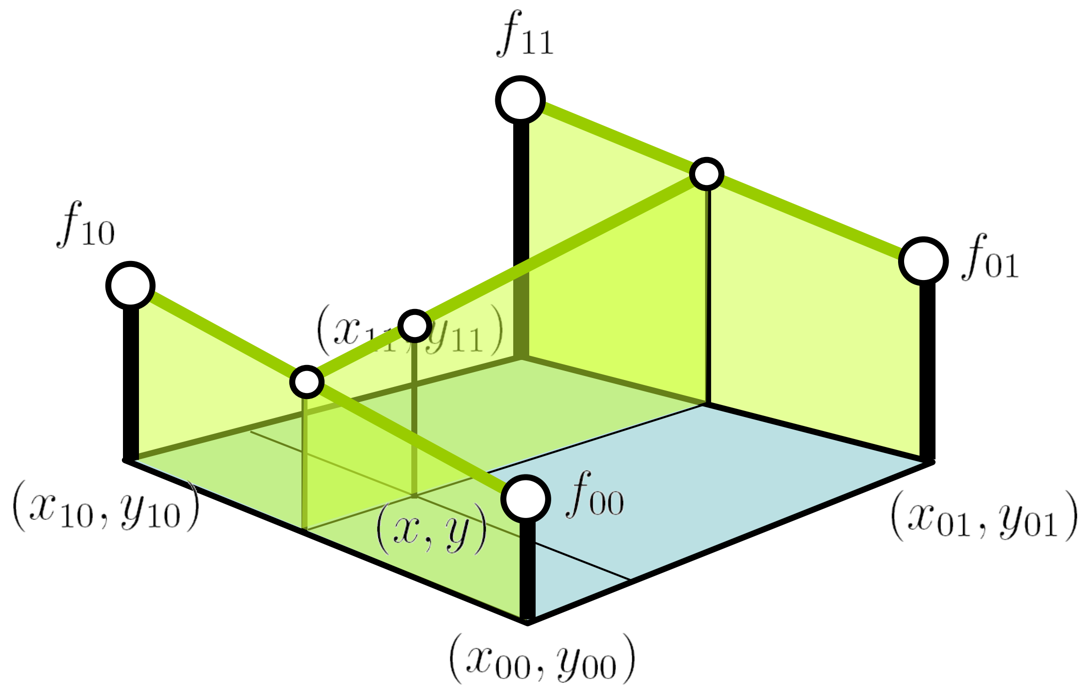
a := Abstand

ohm

Local Interpolation

Bi-linear interpolation (2D)

Zwei mal lineare Interpolation von x und y



- Most common interpolation function on 2D image grids
- Naive implementation requires 8 multiplications, optimized variant requires 3
- Fast data access with pointer arithmetic

```
dp = &data[y * NX + x];
f00 = dp[0];
f10 = dp[1];
dp += NX;
f01 = dp[0];
f11 = dp[1];
```

$$f_0 = \alpha f_{10} + (1 - \alpha) f_{00}$$

$$f_1 = \alpha f_{11} + (1 - \alpha) f_{01}$$

$$f(x, y) = \beta f_1 + (1 - \beta) f_0$$

$$\alpha = \frac{x - x_{00}}{x_{10} - x_{00}}$$

$$\beta = \frac{y - y_{00}}{y_{10} - y_{00}}$$

- Note: bi-linear interpolation is not a linear operation anymore!

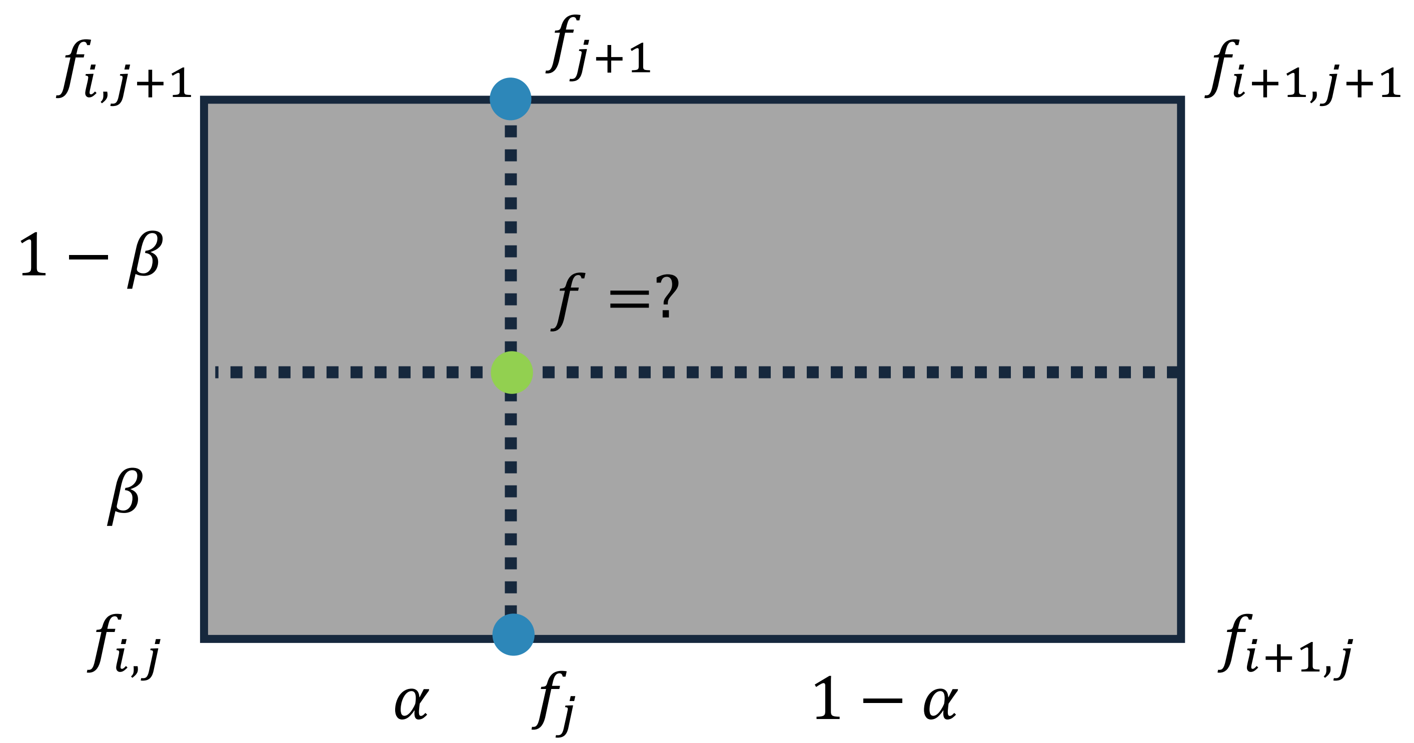
Local Interpolation

Bilinear interpolation on a rectangle

$$f_j = (1-\alpha) f_{i,j} + \alpha f_{i+1,j}$$

$$f_{j+1} = (1-\alpha) f_{i,j+1} + \alpha f_{i+1,j+1}$$

$$f = (1-\beta) f_j + \beta f_{j+1}$$



$$f(x,y) = (1-\beta)\left[(1-\alpha) f_{i,j} + \alpha f_{i+1,j}\right] + \beta\left[(1-\alpha) f_{i,j+1} + \alpha f_{i+1,j+1}\right]$$

$$\alpha = \frac{x - x_i}{x_{i+1} - x_i}, \qquad \beta = \frac{y - y_i}{y_{i+1} - y_i} \quad , \alpha, \beta \in [0,1]$$

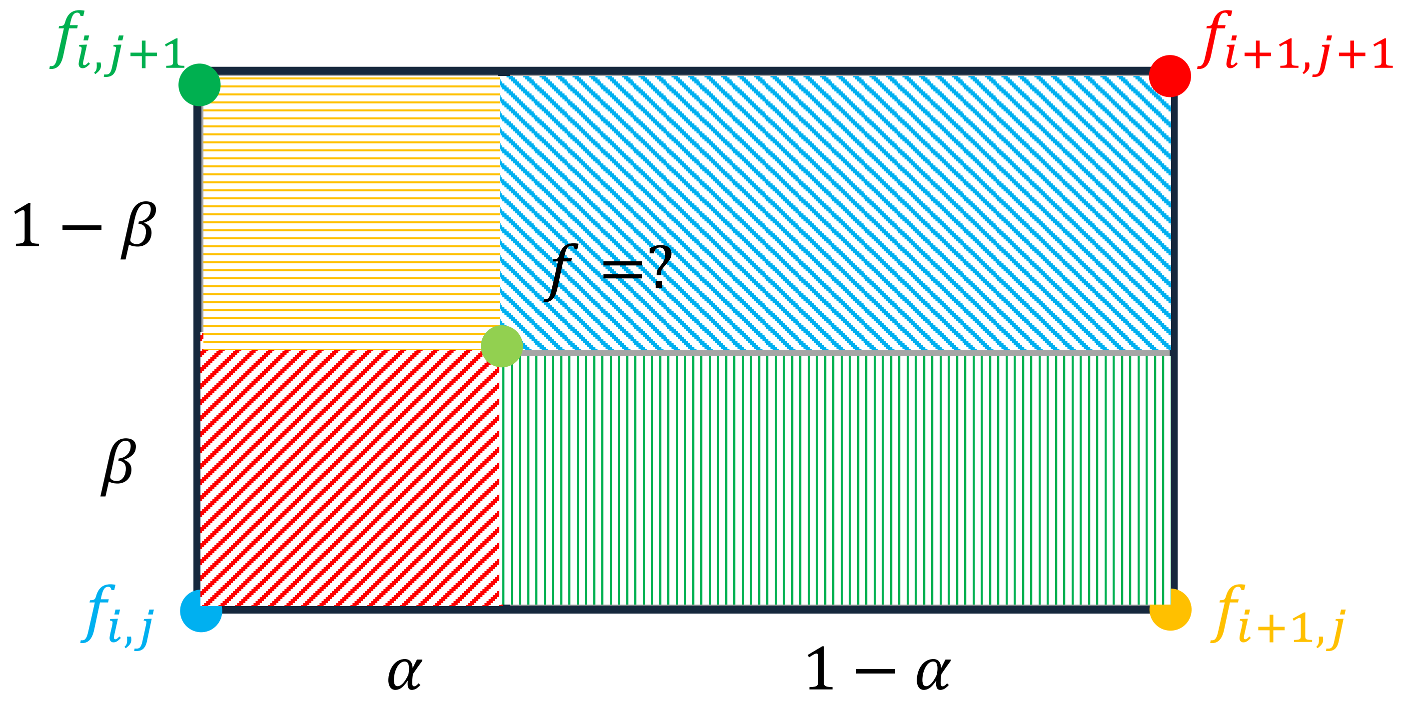
ohm

Local Interpolation

Bilinear interpolation on a rectangle

Gewichtete Summen vom gegenüberliegenden Punkt

$$
\begin{aligned}
f(x,y) = &(1-\alpha)(1-\beta) f_{i,j} \\
&+ \alpha(1-\beta) f_{i+1,j} \\
&+ (1-\alpha)\beta f_{i,j+1} \\
&+ \alpha\beta f_{i+1,j+1}
\end{aligned}
$$

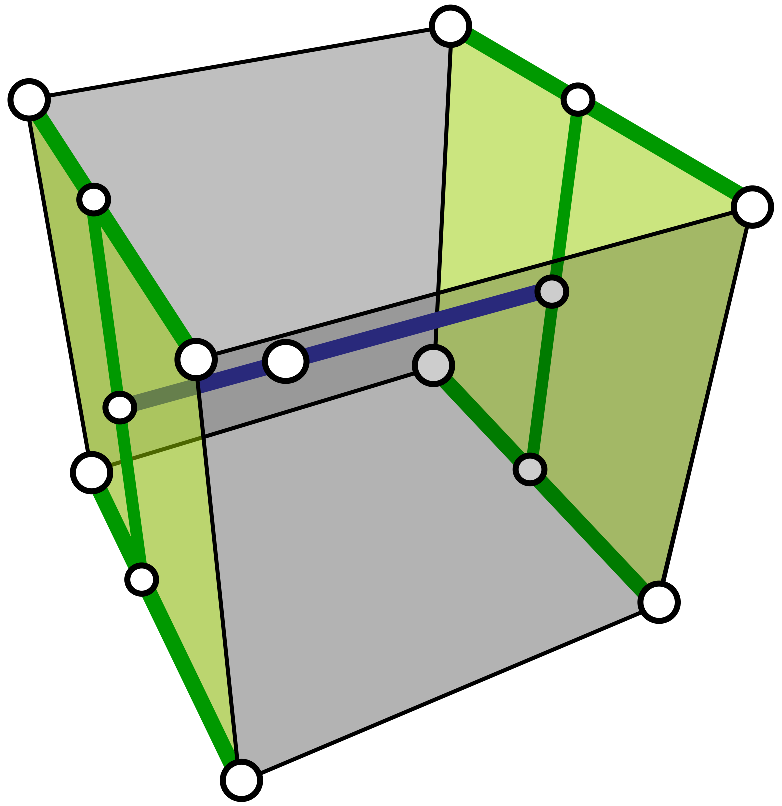


- Weighted by local areas of the opposite point
- Bilinear interpolation is not linear (but quadratic)!
  - Cannot be inverted easily!

Local Interpolation

Tri-linear interpolation (3D)

- Voxel-grids, non-linear operation
- 27 vs. 7 multiplications
- Computationally intensive on large voxel datasets



8 Nachtbarpunkte

4 linear interpolations

$$f_{00} = \alpha f_{001} + (1-\alpha) f_{000}$$

$$f_{01} = \alpha f_{011} + (1-\alpha) f_{010}$$

$$f_{10} = \alpha f_{101} + 1-\alpha) f_{100}$$

$$f_{11} = \alpha f_{111} + (1-\alpha) f_{110}$$

2 bi-linear interpolations

$$f_0 = \beta f_{10} + (1-\beta) f_{00}$$

$$f_1 = \beta f_{11} + (1-\beta) f_{01}$$

1 tri-linear interpolation

$$f(x,y,z) = \gamma f_1 + (1-\gamma) f_0$$

ohm

Local Interpolation

3D-case on a 3D uniform grid

- Trilinear interpolation of points (x,y,z)
  - Straightforward extension of bilinear interpolation
  - Three local coordinates α, β, γ
- Trilinear interpolation is not linear!
- Efficient evaluation (Horner): $f(\alpha,\beta,\gamma)=\alpha+\alpha(b+\beta(e+h\lambda))+\beta(c+f\gamma)+\gamma(d+g\alpha)$
  - Coefficients a, b, c, d, e, f, g from data at corner vertices

Local Interpolation

Linear interpolation on a triangle

- Point P can be expressed as

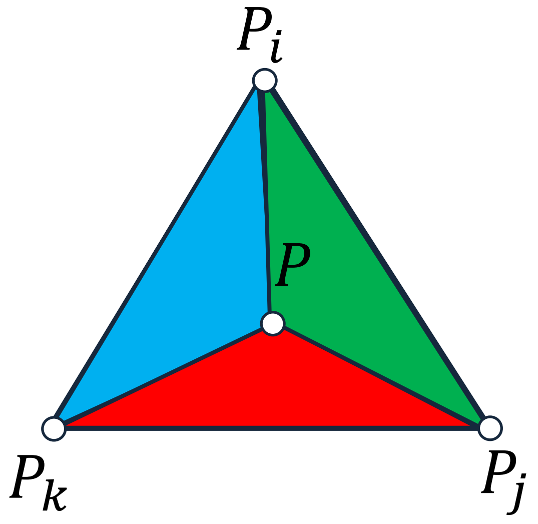
$$\boxed{P = a_i P_i + a_j P_j + a_k P_k}$$

- with $a_i + a_j + a_k = 1$ and

$$a_i = \frac{Area(\Delta P P_j P_k)}{Area(\Delta P_i P_j P_k)}$$

$$a_j = \frac{Area(\Delta P_i P P_k)}{Area(\Delta P_i P_j P_k)}$$

$$a_i = \frac{Area(\Delta P P_j P_k)}{Area(\Delta P_i P_j P_k)}$$



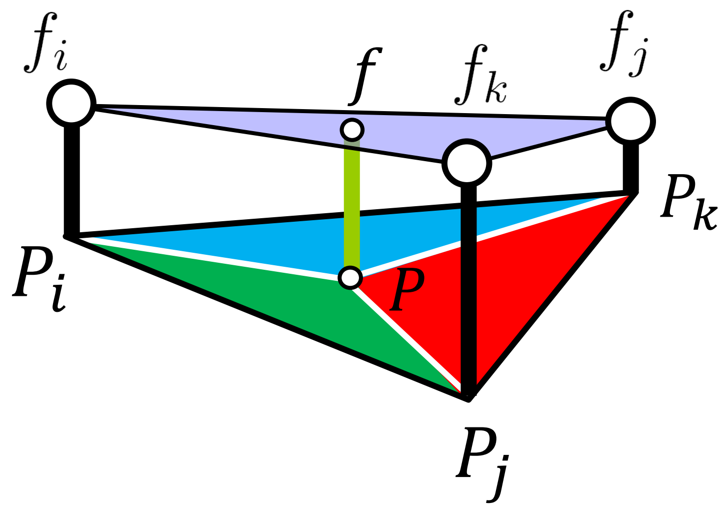
If $a_i = a_j = a_k = \frac{1}{3}$

$\Rightarrow P$ is ***barycenter*** of the triangle

a_i, a_j, a_k are ***barycentric*** coordinates
(Baryzentrische Koordinaten)

Interpolation

Barycentric interpolation



- Determine barycentric coordinates of point P

$$P = a_i P_i + a_j P_j + a_k P_k$$

- Then determine function f value using the same weights

$$f = a_i f_i + a_j f_j + a_k f_k$$

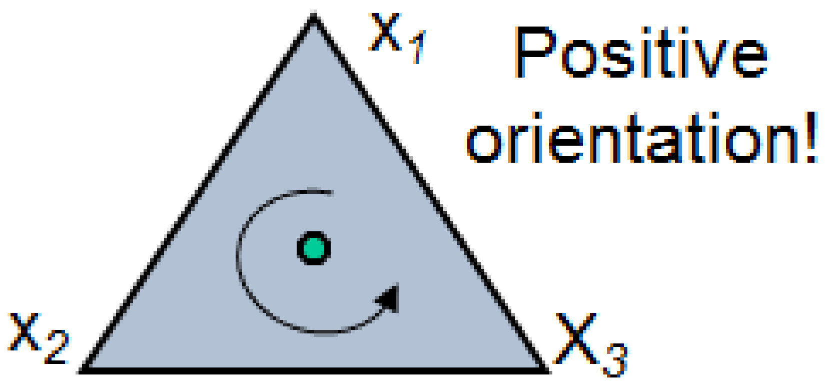
- Note
  - Points outside of the triangle can be expressed with barycentric coordinates
  - Then some of the weights are negative
  - But still: $a_i + a_j + a_k = 1$

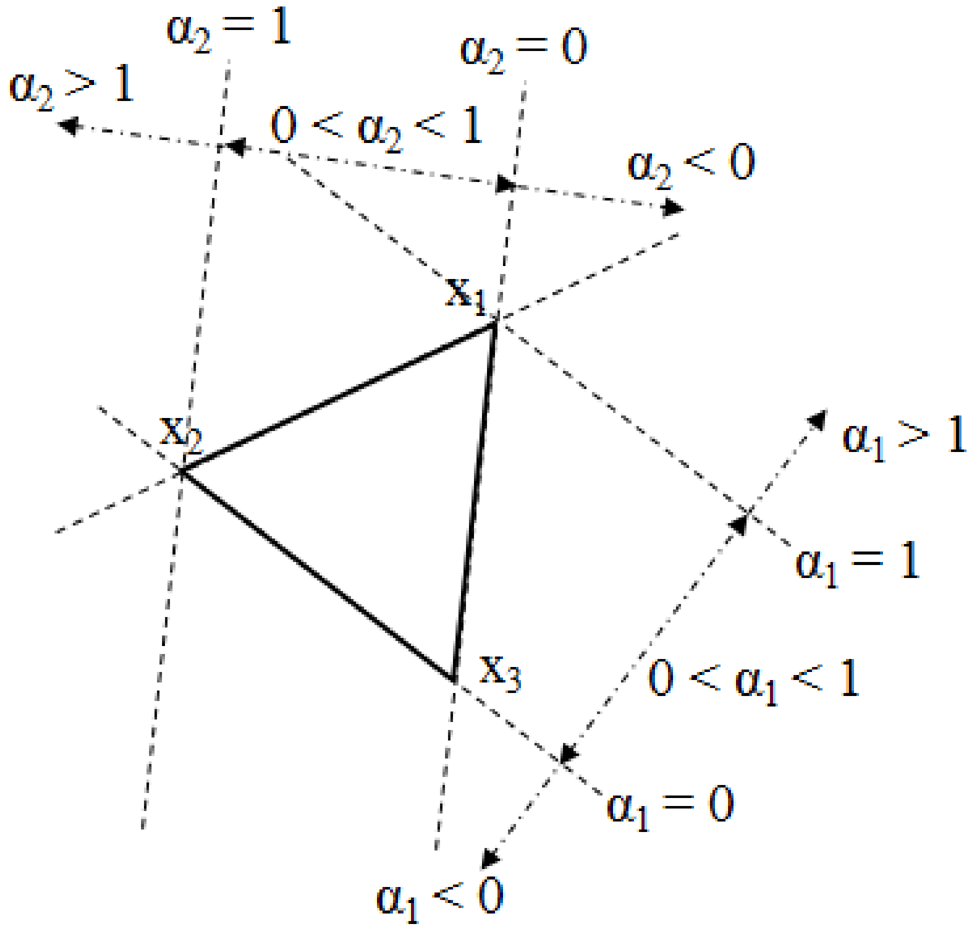
ohm

Local Interpolation

Localization with barycentric coordinates

- Use right-hand-rule for orientation



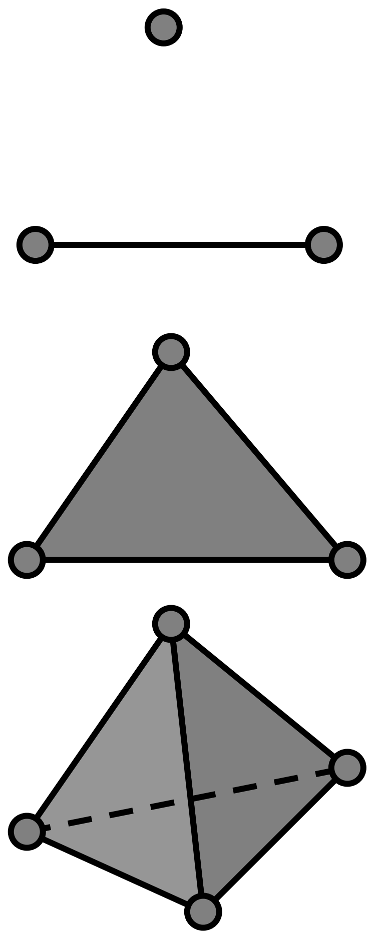


  - A point is inside the triangle if and only if for all weights: $0 < \alpha_i < 1$
  - Use for cell search

Local Interpolation

Barycentric coordinates in multiple dimensions

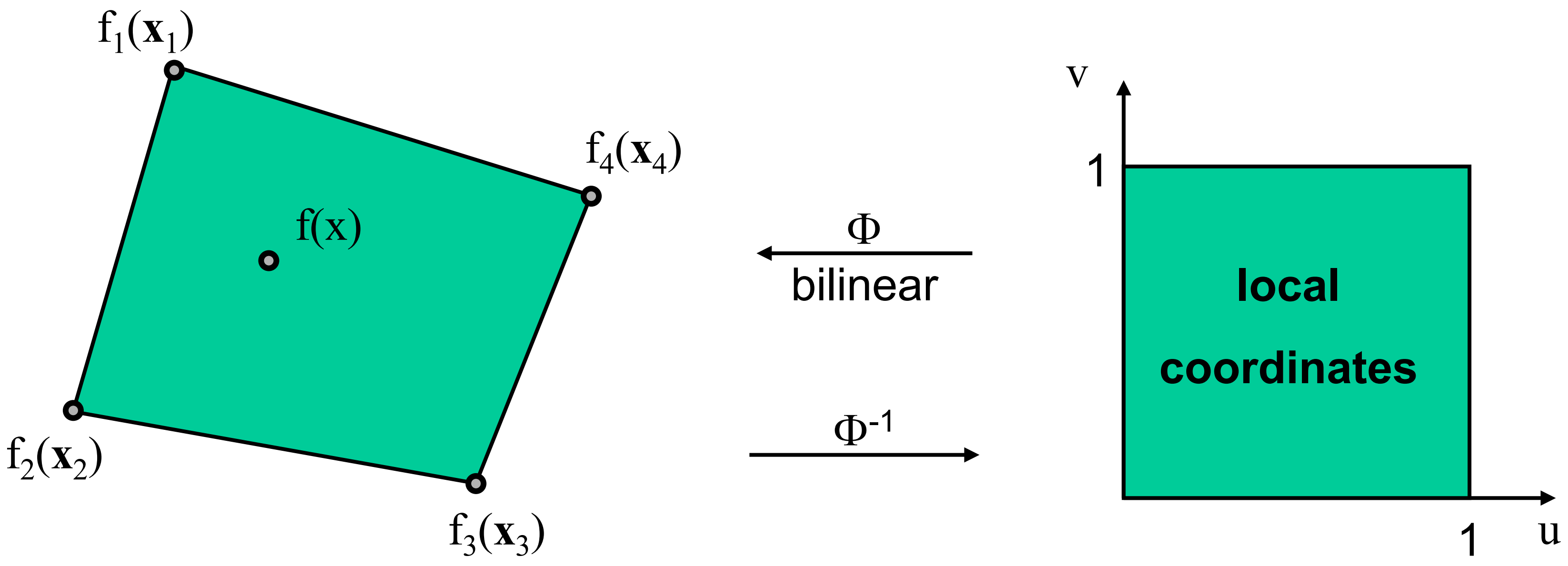
- Simplex in R^d → defined by its vertices
  - d+1 affinely independent points
    - 0D: point
    - 1D: line
    - 2D: triangle
    - 3D: tetrahedron



Local Interpolation

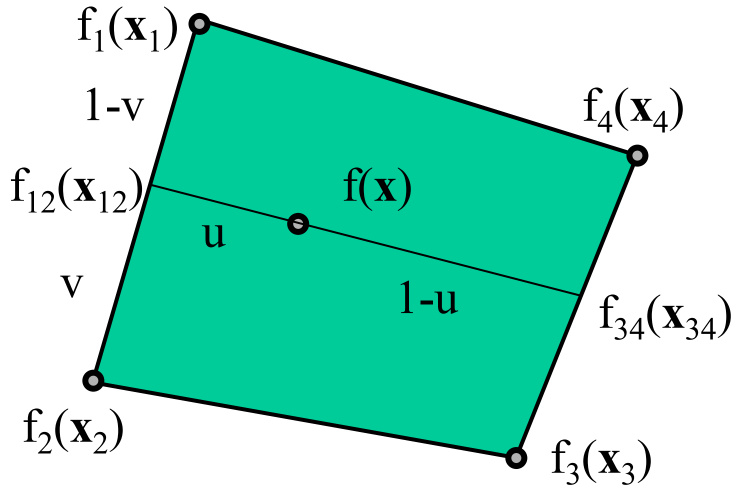
Interpolation in a generic quadrilateral

- Main application in curvilinear grids
  - Find a parameterization of arbitrary quadrilaterals
  - Use local coordinates for interpolation



Local Interpolation

- We know $\Phi(u,v) = (x,y)^T$
- Mapping from rectangular to quadratic domain
  - Bilinear interpolation on rectangle



$$\mathbf{x}_{12} = v\mathbf{x}_1 + (1-v)\mathbf{x}_2$$
$$\mathbf{x}_{34} = v\mathbf{x}_4 + (1-v)\mathbf{x}_3$$

$$\Phi(u,v) = \begin{pmatrix} x \\ y \end{pmatrix} = u\mathbf{x}_{34} + (1-u)\mathbf{x}_{12} \quad \text{with} \quad u,v \in [0,1]$$

- Final value $f = uvf_4 + u(1-v)f_3 + (1-u)vf_1 + (1-u)(1-v)f_2$

Local Interpolation

Computing the inverse $\Phi^{-1}(x, y) = (u, v)^T$ is more complicated

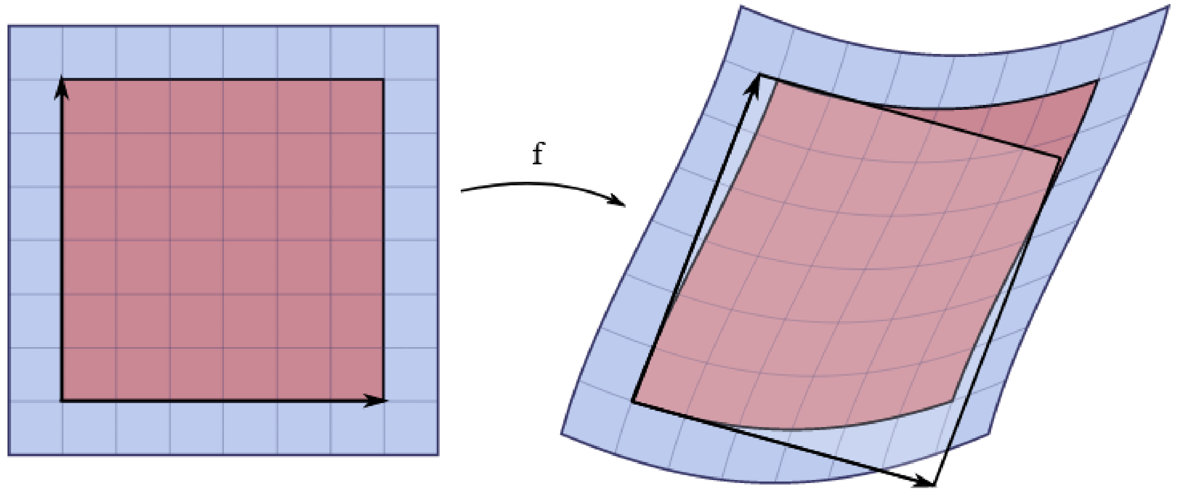
- Analytically, solve quadratic system for u, v
  - Use numerical solution by Newton iteration instead
- Stencil-walk algorithm [Bunnig '89]
  - In context of flow visualization
- Start with seed points $\Phi(u_0, v_0) = (x_0, y_0)^T$
  - Iteratively, move towards solution (x, y) by means of Jacobian $J_\Phi(u, v)$

Local Interpolation

Computing the inverse $\Phi^{-1}(x, y) = (u, v)^T$ is more complicated

- Determine start value
  - $(u_0, v_0) \in \{(0,0), (1,0), (0,1), (1,1)\}$ depending on closest vertex (x, y)
  - Alternatively, start from center: $u_0 = v_0 = {}^1/_2$
- Jacobi matrix $J_\Phi(u, v)$
  - $J_\Phi(a)_{i,j} = \dfrac{\partial \Phi_i}{\partial x_j}$, with $x = (u, v)$
  - The Jacobian describes direction and speed of position changes of Φ when the parameters are varied

Local Interpolation



solange Ableiten bis keine Änderung mehr

- From Taylor: $$\begin{pmatrix} x \\ y \end{pmatrix} - \Phi(u_0, v_0) \approx J_\Phi(u_0, v_0) \begin{pmatrix} u_1 - u_0 \\ v_1 - v_0 \end{pmatrix}$$
- Determine local coordinates (u_1, v_1) as follows:
  - Solve $J_\Phi(u_0, v_0) \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} - \Phi(u_0, v_0)$ and set $(u_1, v_1) = (u_0 + \Delta u, v_0 + \Delta v)$
- Newton iteration

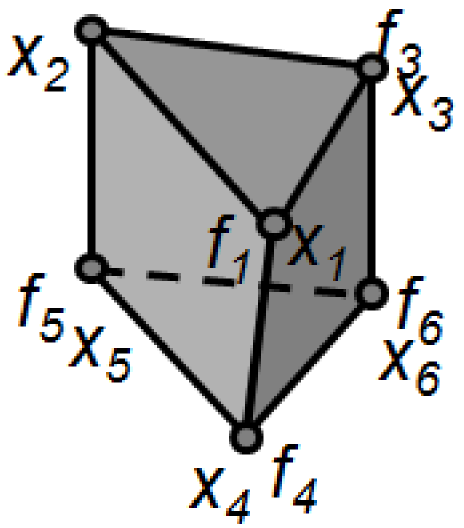
```
start with start configuration as seed points
  while (||x - Φ(u,v)|| > ε )
    compute J(Φ(u,v))
    transform x in coordinate system J(Φ):
      (Δu,Δv) = J(Φ(u,v))^-1·( x - Φ(u,v) )
    update (u_i+1,v_i+1) = (u_i,v_i) + (Δu,Δv)
```

Source: https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant

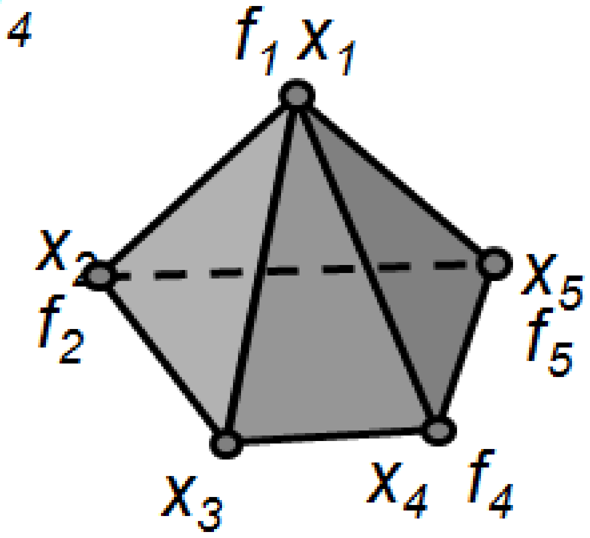
Local Interpolation

More 3D cases

- Tetrahedron → Barycentric
- Hexahedron (Box) → Trilinear
- Prism (over triangle)
  - Twice barycentric → once linear



- Pyramid (over quadrangle)
  - Bilinear on base face → then linear



3.6.2 Global Interpolation

Global Interpolation

More advanced interpolation functions

- Interpolation aims to estimate function values based on samples
  - Linear interpolation: very simple + acceptable quality
  - Considers only local neighborhood
- To improve quality find a better approximation function
  - Increases calculation cost (usually)
  - The right choice is often application dependent
- Might consider bigger neigborhood or all samples → global interpolation

Global Interpolation

Lanczos Filter

- sinc on a sliding window (neigborhood size)
  - Choose window size a (number of samples), then

$$L(x) = \begin{cases} 1, & x = 0 \\ \dfrac{a\sin(\pi x)\sin(\pi x/a)}{\pi^2 x^2}, & -a \le x < a \ \wedge x \ne 0 \\ 0, & sonst \end{cases}$$

  - and the interpolation function is $f(x) = \sum_{i=\lfloor x \rfloor - a + 1}^{\lfloor x \rfloor + a} x_i L(x-i)$

Global Interpolation

Splines

- In general, given $n + 1$ samples $\{x_0, x_1, \dots, x_n\}$ and corresponding values $\{y_0, y_1, \dots, y_n\}$, where $x_i \neq x_k$, if $i \neq k$ and $0 \leq i, k \leq n$ the set of polynomials

  $L_i(x) = \prod_{\substack{0 \leq k \leq n \\ i \neq k}} \frac{x - x_k}{x_i - x_k}$ of degree form a basis of a space where we can construct a

  function $f(x) = \sum_{i=0}^{n} y_i L_i(x)$, that interpolates the given samples (i.e. $f(x_i) = y_i$)

- The polynomials are called ***Lagrange***-Polynomials and there is a **unique** polynomial for any given set of points

Global Interpolation

Splines

- Problems
  - For n+1 data points we get a polynomial of degree n
  - High degree polynomials lead to oscillation at the edges of intervals (Runge's phenomenon) – hence they are infeasible for interpolation of large data
- Solution approach
  - Split the interval $x_0 \leq x_n$ into subintervals $[t_0, t_1], \dots, [t_i, t_{i+1}], \dots [t_{n-1}, t_n]$
  - Interpolate each subinterval with appropriate polynomial
  - Connect the individual segments → **Spline**

Global Interpolation

Splines

- Spline polynomials form a base for f: $f(x) = \sum_{i=0}^{n} B_i^n(x) b_i$ on the intervals
  - B_i^n are the basis functions, b_i the control points, forming a control polygon
  - b_i is usually not equal to x_i (depending on the spline)
  - b_i can be chosen such that $f(x_i) = y_i$
  - Continuity constraints at the interval borders depend on spline type
- There exist different Spline types with different properties
  - Cardinal Splines, Catmull-Rom Splines, Bézier-Splines, B-Splines, ...

Global Interpolation

Catmull-Rom Interpolation

- Compromise between "local linear" and "global B-Spline"
- Properties
  - Spline passes through all control points (i.e. $b_i = x_i$!)
  - Piecewise cubic, C^1-continous
    - There are *no discontinuities* in tangent direction and magnitude
  - Not C^2-continous
    - Second derivative is *linearly interpolated* within each segment
  - Error: $O(h^3)$

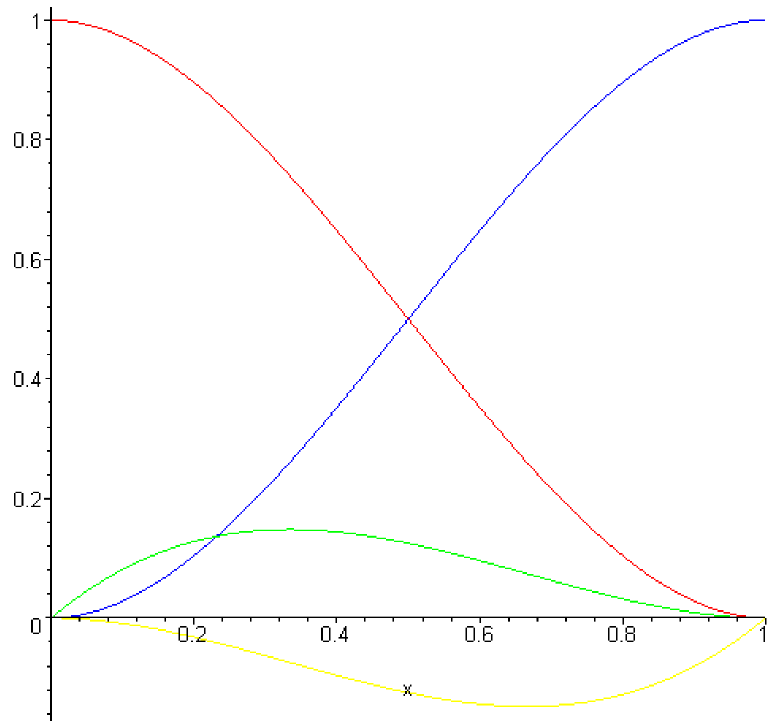
ohm

Global Interpolation

Hermite-Interpolation

- Given P_0, P_o', P_1, P_1' , then there exists a unique cubic polynomial f with
  - $f(t_o) = P_0, f'(t_o) = P_o', f(t_1) = P_1, f'(t_1) = P_1'$

Hermite polynoms	*f(0)*	*f'(0)*	*f'(1)*	*f(1)*
$H_0(t) = (1-t)^2(2t+1)$	1	0	0	0
$H_1(t) = t(1-t)^2$	0	1	0	0
$H_2(t) = -t^2(1-t)$	0	0	1	0
$H_3(t) = t^2(3-2t)$	0	0	0	1

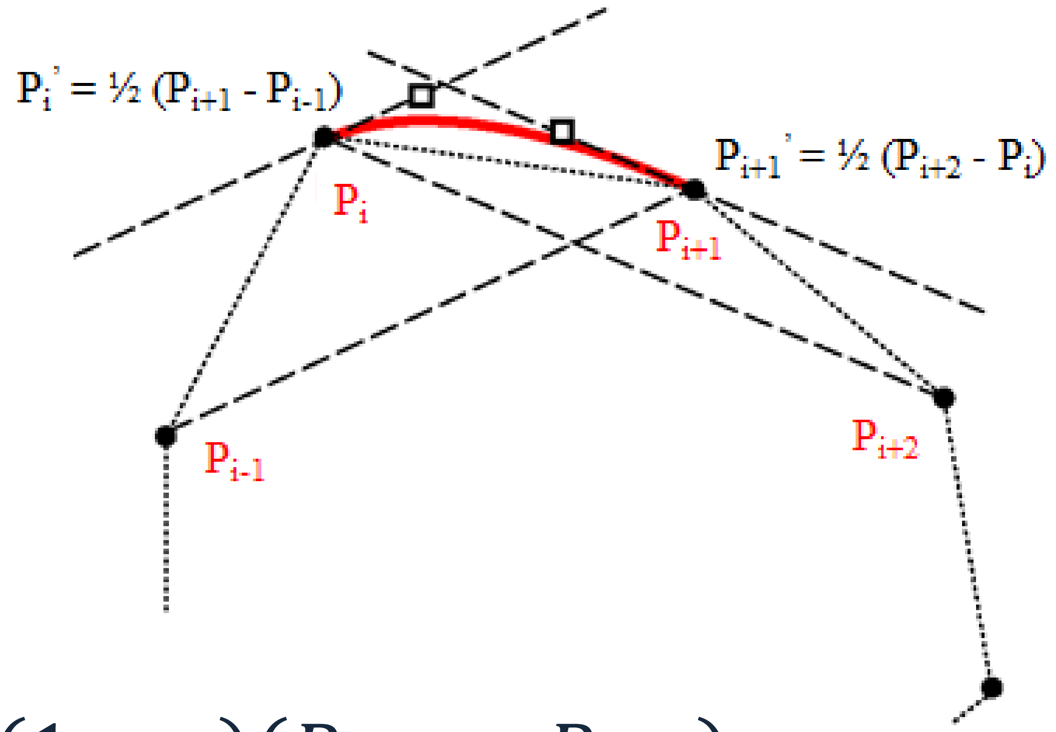


- Then $f(t) = P_0 H_0(t) + P_0' H_1(t) + P_1' H_2(t) + P_1 H_3(t)$ interpolates the positions P_0 and P_1 and the derivatives P_o' and P_1' !

ohm

Global Interpolation

Cardinal splines



- Subset of Hermite curves
  - Calculate tangent points from control points: $P_i' = \frac{1}{2}(1-c)(P_{i+1} - P_{i-1})$
  - c is a *tension* parameter

Ableitung am Punkt P_i

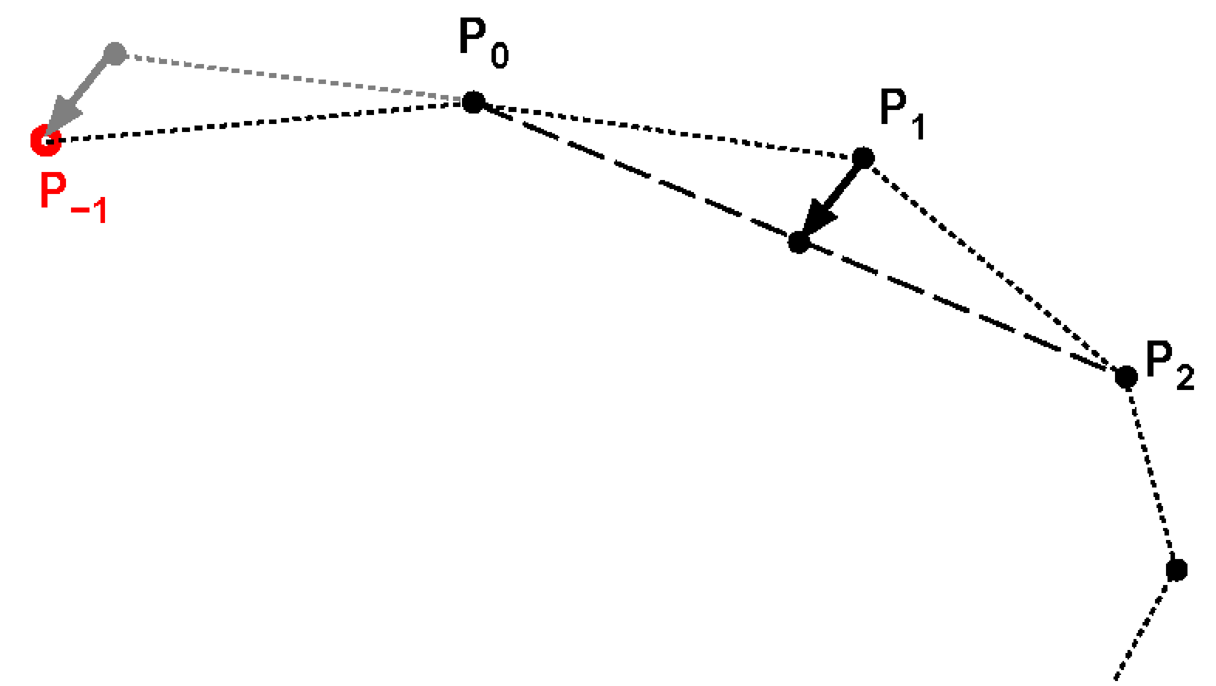
- **Catmull-Rom Splines**
  - Subset of cardinal splines where $c = 0$
  - Estimate derivatives by symmetric differences
    - Derivatives at a point P_i are parallel to the line $[P_{i-1}; P_{i+1}]$
  - We have the local property, i.e. if $t_i < t < t_{i+1}$, then $f(t)$ depends only on $P_{i-1}, P_i, P_{i+1}, P_{i-2}$

ohm

Global Interpolation

Estimate of derivatives in border points

- For this determine P_{-1} resp. P_{n+1}
  - I.e. estimate tangent along the lines $[P_{-1}; P_0]$, and $[P_n; P_{n+1}]$
  - This leads to the natural boundary condition $f''(t_0) = f''(t_n) = 0$



ohm

Global Interpolation

Further Reading and advanced topics

- Curves and Surfaces (Geometric Modelling)
  - Gerald Farin, Curves and Surfaces for CAGD, 5th Ed., Elsevier
  - Banchoff & Lovett, Differential Geometry of Curves and Surfaced, 3rd Ed., Chapman and Hall/CRC
  - Abate & Tovena, Curves and Surfaces, Spinger

- Advanced Topics: Scattered Data Interpolation using Radial Basis Functions
  - e.g. Iske, Scattered Data Modelling Using Radial Basis Functions, Springer