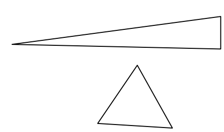
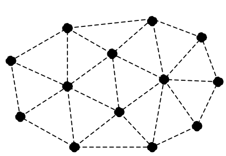
3. Interpolation and Filtering

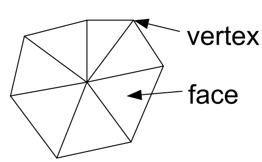
- Data is often discretized in space and / or time
- Finite number of samples
  - The continuous signal is usually known only at a few points (data points)
  - In general, data is needed in between these points
- By interpolation we obtain a representation that matches the function at the data points
  - Evaluation at any other point possible
- Reconstruction of signal at points that are not sampled
- Assumptions needed for reconstruction
  - Often smooth functions

3.1. Voronoi Diagrams and Delaunay Triangulation

- Given irregularly distributed positions without connectivity information
- Problem: obtain connectivity to find a “good” triangulation
- For a set of points there are many possible triangulations
  - A measure for the quality of a triangulation is the aspect ratio of the so-defined triangles
  - Avoid long, thin ones







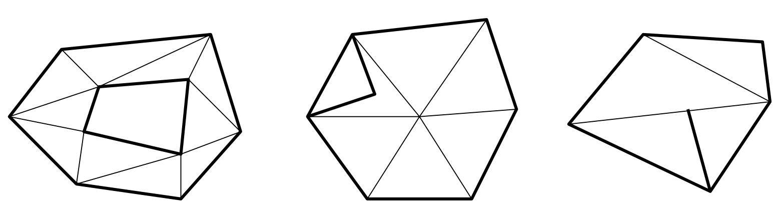
3.1. Voronoi Diagrams and Delaunay Triangulation

- Scattered data triangulation
  - A triangulation of data points $S = s_0, s_1, \ldots, s_m \in R^2$ consists of
    - Vertices (0D) = S
    - Edges (1D) connecting two vertices
    - Faces (2D) connecting three vertices
- A triangulation must satisfy the following criteria
  - $\cup$ faces = *conv*(S), i.e. the union of all faces including the boundary is the convex hull of all vertices
  - The intersection of two triangles is either empty, or a common vertex, or a common edge, or a common face (tetrahedra)

3.1. Voronoi Diagrams and Delaunay Triangulation

- Triangulation with

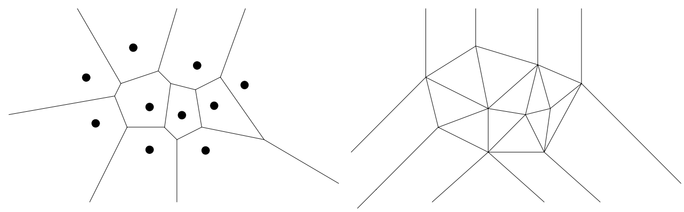
holes, faces overlap, T-vertices



are not valid

3.1. Voronoi Diagrams and Delaunay Triangulation

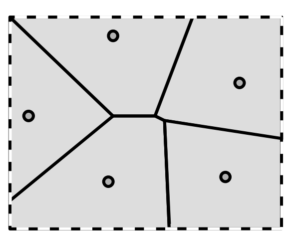
- How to get connectivity/triangulation from scattered data ?
  - Voronoi diagram
  - Delaunay triangulation



3.1. Voronoi Diagrams and Delaunay Triangulation

- Voronoi diagram
  - For each sample every point within a Voronoi region is closer to it than to every other sample
  - Given: a set of points $X=\{x_1,\ldots,x_n\}$ from $\boldsymbol{R}^d$ and a distance function $dist(x,y)$
  - The Voronoi diagram $Vor(X)$ contains for each point x_i a cell $V(x_i)$ with

$$V(x_i)=\{x \mid dist(x, x_i) < dist(x, x_j) \ \forall j \neq i\}$$

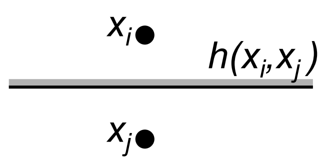


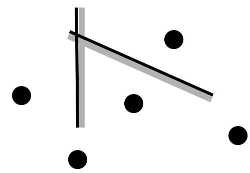
3.1. Voronoi Diagrams and Delaunay Triangulation

- Voronoi cells
  - The half space $h(x_i,x_j)$ is separated by the perpendicular bisector between x_i and x_j
  - $h(x_i,x_j)$ contains x_i
  - Voronoi cell:

$$V(x_i) = \cap_{j \neq i} h(x_i,x_j)$$

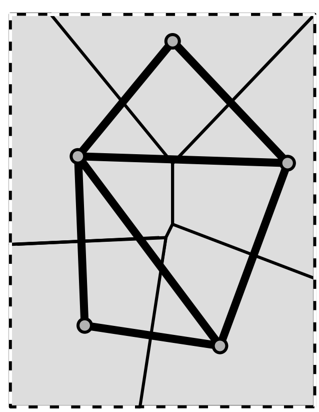
  - Voronoi cells are convex





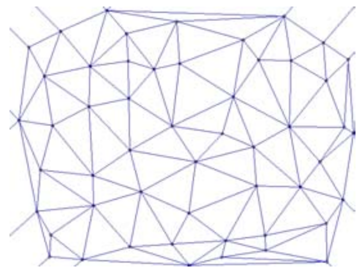
3.1. Voronoi Diagrams and Delaunay Triangulation

- Delaunay graph *Del(X)*
  - The geometric dual (topologically equal) of the Voronoi diagram *Vor(X)*
  - Points in X are nodes
  - Two nodes x_i and x_j are connected iff the Voronoi cells $V(x_i)$ and $V(x_j)$ share a same edge
- Delaunay cells are convex
- Delaunay triangulation = triangulation of the Delaunay graph



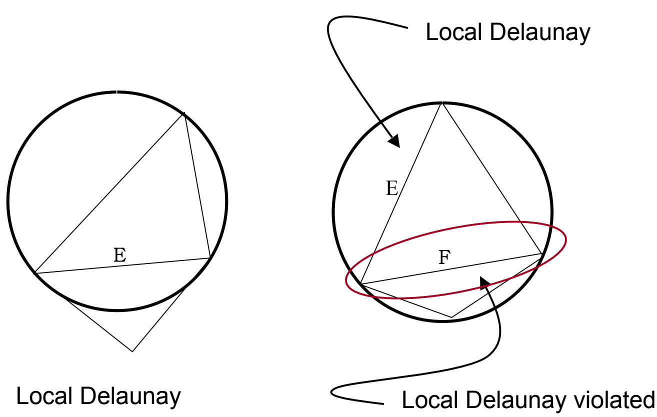
3.1. Voronoi Diagrams and Delaunay Triangulation

- Delaunay triangulation in 2D
  - Three points x_i, x_j, x_k in X belong to a face from *Del(X)* iff no further point lies inside the circle around x_i, x_j, x_k
  - Two points x_i, x_j form an edge iff there is a circle around x_i, x_j that does not contain a third point from X
  - For each triangle the circumcircle does not contain any other sample
  - Maximizes the smallest angle
  - Maximizes the ratio of (radius of incircle)/(radius of circumcircle)
  - It is unique (independent of the order of samples) for all but some very specific cases



3.1. Voronoi Diagrams and Delaunay Triangulation

- Local Delaunay property



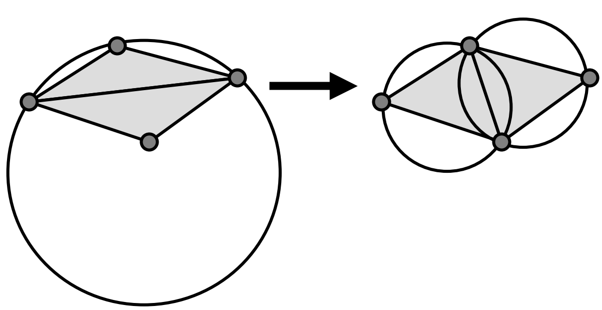
3.1. Voronoi Diagrams and Delaunay Triangulation

- Algorithms for Delaunay triangulations
  - Edge flip algorithm

```
find an initial (valid) triangulation
find all edges where local Delaunay property is violated
mark these edges and push them onto the stack
while (stack not empty)
    pop edge from stack
    if (edge does not satisfy Delaunay property)
        flip this edge
        flip all adjacent edges for which the Delaunay
          property is violated due to the flip
```

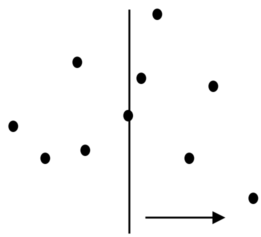

3.1. Voronoi Diagrams and Delaunay Triangulation

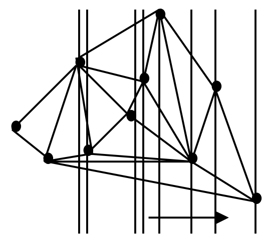
- Edge flip algorithm



3.1. Voronoi Diagrams and Delaunay Triangulation

- Plane-sweep algorithm for finding an initial triangulation
  - Imaginary vertical sweepline passes from left to right
  - As the sweepline moves:
    - Problem has been solved for the data to the left of the sweepline
    - Is currently being solved for the data at or near the sweepline and
    - Is going to be solved sometime later for the data to right of the sweep-line
  - Reduces a problem in 2D space to a series of problems in 1D space





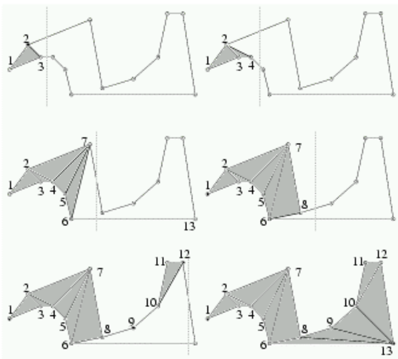
3.1. Voronoi Diagrams and Delaunay Triangulation

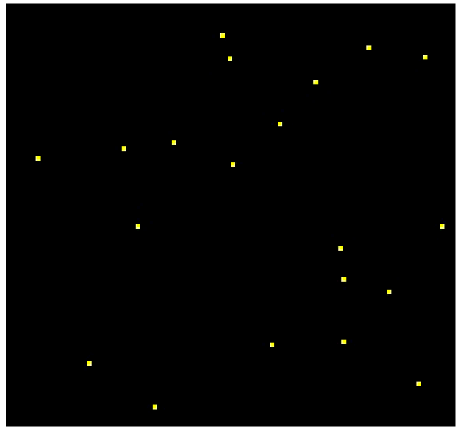
- Plane-sweep algorithm for finding an initial triangulation

```
sort points from left to right
construct initial triangle using first three vertices
for i=4 to n do
    use last inserted p_{i-1} as starting point
    walk counterclockwise along convex polygon (hull) of
      triangulation until the tangent points,
      inserting edges between p_i and polygon points
    walk clockwise along convex polygon of triangulation
      until the tangent points,
      inserting edges between p_i and polygon points
    update convex hull
endfor
```

3.1. Voronoi Diagrams and Delaunay Triangulation

- Plane Sweep algorithm
  - Also for triangulating polygons

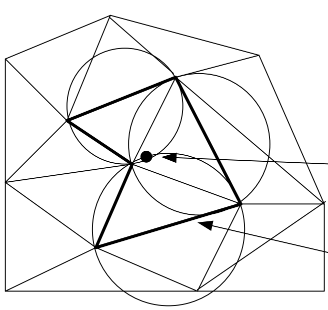


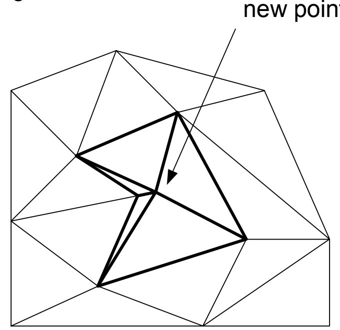


3.1. Voronoi Diagrams and Delaunay Triangulation

- Bowyer-Watson algorithm

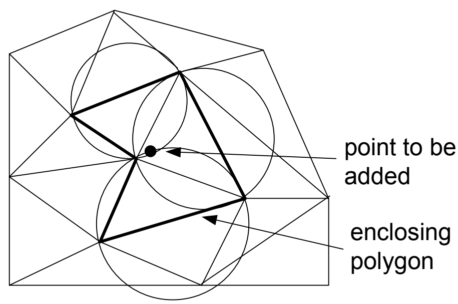
  [D.F. Watson. Computing the -Dimensional Delaunay Tessellation with Application to Voronoi Polytopes. The Computer Journal, 24(2):167-172, 1981]
  [A. Bowyer.Computing Dirichlet Tessellations. The Computer Journal, 24(2):162-166, 1981]
  - Incremental insertion of points into the triangulation





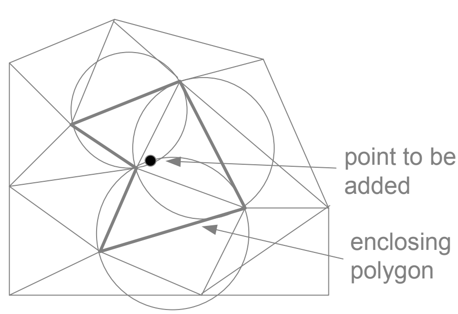
3.1. Voronoi Diagrams and Delaunay Triangulation

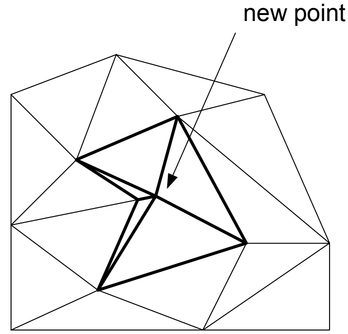
- Bowyer-Watson algorithm
  - Start with initial triangulation which covers the domain (e.g. two triangles of bounding box)
  - Incremental insertion of points into the triangulation
  - All triangles whose circumcircles contain the inserted point are removed
  - *… cont.*



3.1. Voronoi Diagrams and Delaunay Triangulation

- Bowyer-Watson algorithm *cont.*
  - The resulting cavity is triangulated by linking the inserted point to all vertices of the cavity boundary
  - The cavity is star-shaped: Edges from the location of the newly inserted point





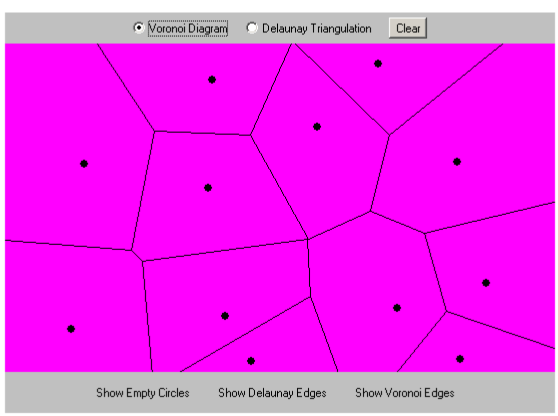
3.1. Voronoi Diagrams and Delaunay Triangulation

- Bowyer-Watson algorithm

```
determine the super triangle that encompasses all vertices
add super triangle vertices to the end of the vertex list
add the super triangle to the triangle list
for (each point in the vertex list)
    calculate the triangle circumcircle center and radius
insert new point
    if (new point lies in a circumcircle)
        add the three triangle edges to the edge buffer
        remove the triangle from the triangle list
delete multiple specified edges from the edge buffer, which
  leaves the edges of the enclosing polygon
add all triangles formed of the point and the enclosing
  polygon
remove all triangles from the triangulation that use the
  super triangle vertices and remove their vertices from the
  vertex list
```


3.1. Voronoi Diagrams and Delaunay Triangulation

- Demo



3.1. Voronoi Diagrams and Delaunay Triangulation

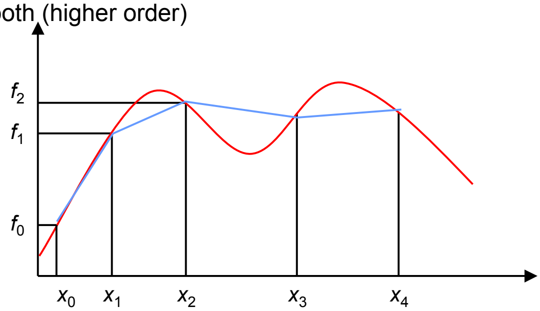
- Other techniques exist
  - Radial sweep
  - Intersecting halfspaces
  - Divide and conquer (merge-based or split-based)

- Running times (worst-case)

	Dim.	
Flipping	2	$O(n^2)$
Plane-sweep (as above)	2	$O(n^2)$
Randomized incremental (BW)	2	$O(n \log n)$
Improved plane-sweep	2	$O(n \log n)$
Divide and conquer	2	$O(n \log n)$
Randomized incremental (BW)	≥ 3	$O(n^{\lceil d/2 \rceil})$

3.2. Univariate Interpolation

- Univariate interpolation: interpolation for one variable
  - Nearest neighbor (0 order)
  - Linear (first order)
  - Smooth (higher order)



3.2. Univariate Interpolation

- Taylor interpolation
- Basis functions: monom basis (polynomials)

  $m_i = x^i$ with $i \in \mathbf{N}_0$
- $\boldsymbol{P}^m=\{1,x,x^2,\ldots,x^m\}$ is m+1-dimensional vector space of all polynomials with maximum degree m
- Coefficients c_i with $f = \Sigma_i c_i \cdot x^i$
- Representation of samples:

  $f(x_j)=f_j \quad \forall j=1..n$
- Interpolation problem

  $$\boldsymbol{V} \cdot \boldsymbol{c} = \boldsymbol{f}$$

  samples

  coefficients (to be solved)

  with the Vandermond matrix $\boldsymbol{V}_{ij} = x_i^{\,j-1}$

3.2. Univariate Interpolation

- Properties of Taylor interpolation
  - Unique solution
  - Numerical problems / inaccuracies
  - Complete system has to be solved again if a single value is changed
  - Not intuitive

3.2. Univariate Interpolation

- Generic interpolation problem:
  - Given are n sampled points $X = \{x_i\} \subseteq \Omega \subseteq \mathbf{R}^d$ with function values f_i
  - n-dimensional function space $\Phi_n^d(\Omega)$ with basis $\{\phi_{i=1..n}\}$
  - Coefficients c_i with $f = \Sigma_i c_i \cdot \phi_i$
  - Representation of samples:

    $f(x_j)=f_j \quad \forall j=1..n$
  - Solving the linear system of equations

    $$\boldsymbol{M} \cdot \boldsymbol{c} = \boldsymbol{f}$$

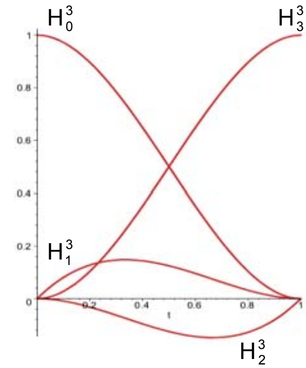
    with $\boldsymbol{M}_{ji} = \phi_i(x_j)$, $\boldsymbol{c}_i = c_i$, and $\boldsymbol{f}_j = f_j$
  - **Note**: number of points n determines dimension of vector space (= degree of polynomials)

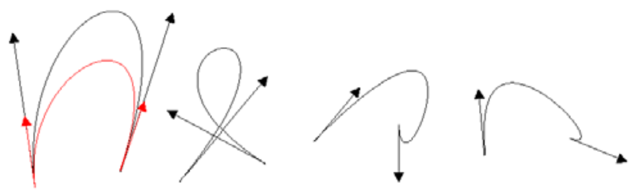
3.2. Univariate Interpolation

- Other basis functions result in other interpolations schemes:
  - Lagrange interpolation
  - Newton interpolation
  - Bernstein basis: Bezier curves (approximation)
  - Hermite basis

3.2. Univariate Interpolation

- Cubic Hermite polynomials **H**
- Coefficients describe:
  - End points
  - Tangent vectors at end points

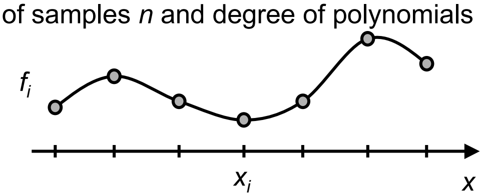




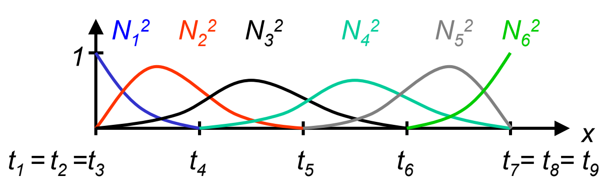
$$H_0^3(t) = (1-t)^2(1+2t)$$
$$H_1^3(t) = t(1-t)^2$$
$$H_2^3(t) = -t^2(1-t)$$
$$H_3^3(t) = (3-2t)t^2$$

3.2. Univariate Interpolation

- Problem: coupling of number of samples *n* and degree of polynomials



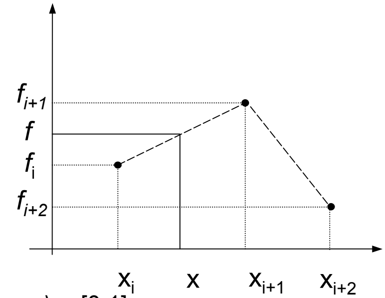
- Solution: Spline interpolation
  - Smooth piecewise polynomial function
  - Continuity / smoothness at segment boundaries!



B-Spline

3.2. Univariate Interpolation

- Piecewise linear interpolation
  - Simplest approach (except for nearest-neighbor sampling)
  - Fast to compute
  - Often used in visualization applications
  - C^0 continuity at segment boundaries



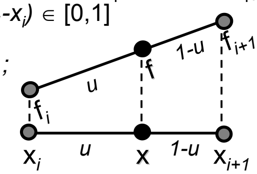
  - Data points: $(x_0,f_0),\ldots,(x_n,f_n)$
  - For any point x with

    $x_i \le x \le x_{i+1}$

    described by local coordinate $u=(x-x_i)/(x_{i+1}-x_i) \in [0,1]$

    that is $x= x_i +u(x_{i+1}-x_i) =(1-u)x_i +ux_{i+1}$;

    evaluate $f(x)=(1-u)f_i + uf_{i+1}$



3.3. Differentiation on Grids

- First approach
  - Replace differential by „finite differences“
  - Note that approximating the derivative by

$$f'(x) = \frac{df}{dx} \rightarrow \frac{\Delta f}{\Delta x}$$

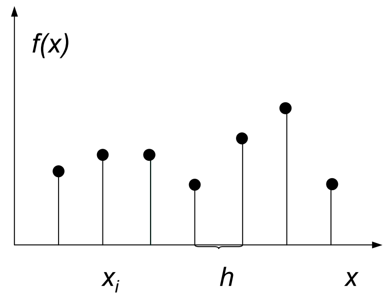
causes subtractive cancellation and large rounding errors for small h

$$f'(x) \approx \frac{f(x+h)-f(x)}{h}$$

- Second approach
  - Approximate/interpolate (locally) by differentiable function and differentiate this function

3.3. Differentiation on Grids

- Finite differences on uniform grids with grid size *h* (1D case)



3.3. Differentiation on Grids

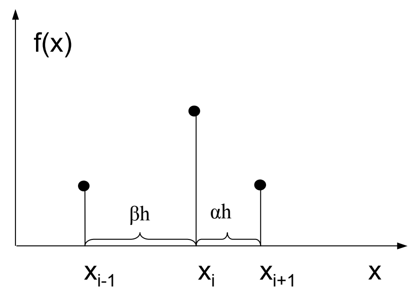
- Finite differences on uniform grids with grid size *h* (1D case)
  - Forward differences $$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h}$$
  - Backward differences $$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h}$$
  - Central differences $$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h}$$
- Error estimation:
  - Forward/backward differences are first order
  - Central differences are second order

3.3. Differentiation on Grids

- Finite differences on non-uniform grids
  - Forward and backward differences as for uniform grids with

$$x_{i+1} - x_i = \alpha h$$

$$x_i - x_{i-1} = \beta h$$



3.3. Differentiation on Grids

- Finite differences on non-uniform grids
  - Central differences by Taylor expansion around the point x_i

$$f(x_{i+1}) = f(x_i) + \alpha h f'(x_i) + \frac{(\alpha h)^2}{2} f''(x_i) + \ldots$$

$$f(x_{i-1}) = f(x_i) - \beta h f'(x_i) + \frac{(\beta h)^2}{2} f''(x_i) + \ldots$$

$$\Rightarrow \frac{1}{\alpha^2}(f(x_{i+1}) - f(x_i)) - \frac{1}{\beta^2}(f(x_{i-1}) - f(x_i)) = \frac{h}{\alpha} f'(x_i) + \frac{h}{\beta} f'(x_i) + O(h^3)$$

  - The final approximation of the derivative:

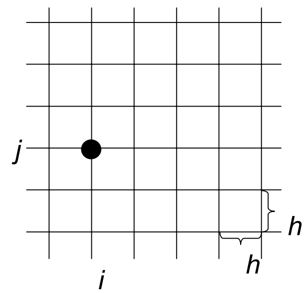
$$f'(x_i) = \frac{1}{h(\alpha+\beta)} \left(\frac{\beta}{\alpha} f(x_{i+1}) - \frac{\alpha}{\beta} f(x_{i-1}) + \frac{\alpha^2 - \beta^2}{\alpha\beta} f(x_i) \right)$$

3.3. Differentiation on Grids

- 2D or 3D uniform or rectangular grids
  - Partial derivatives

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

  - Same as in univariate case along each coordinate axis

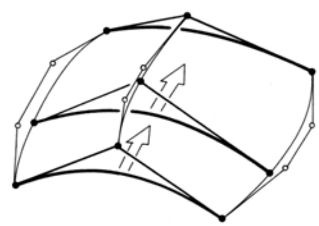


  - Example: gradient in a 3D uniform grid

$$\operatorname{grad} f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{pmatrix} = \begin{pmatrix} \frac{f_{i+1,j,k} - f_{i-1,j,k}}{2h} \\ \frac{f_{i,j+1,k} - f_{i,j-1,k}}{2h} \\ \frac{f_{i,j,k+1} - f_{i,j,k-1}}{2h} \end{pmatrix}$$

3.4. Interpolation on Grids

- Manifolds with more than 1D
- Tensor product
- Combination of several univariate interpolations
- Example for 2D surface:
  - $n \cdot m$ values f_{jl} with $j=1..n$ and $l=1..m$ given at points $X \times Y = (x_1,\ldots,x_n) \times (y_1,\ldots,y_m)$
  - n univariate basis functions $\xi_j(x)$ on X
  - m univariate basis functions $\psi_l(y)$ on Y
  - $n \cdot m$ basis functions on $X \times Y$:



$$\phi_{ij}(x,y) = \xi_i(x) \cdot \psi_j(y)$$

  - Tensor product: $$f(x,y) = \sum_{i=1,j=1}^{n,m} \phi_{ij}(x,y) c_{ij}$$

3.4. Interpolation on Grids

- Tensor product

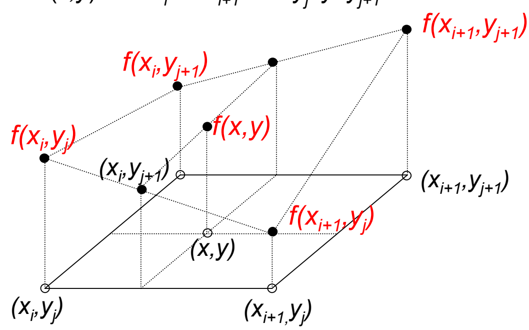
$$f(x,y) = \sum_{i=1,j=1}^{n,m} \phi_{ij}(x,y)\,c_{ij}$$

- Solve a linear system of equations for the unknown coefficients c_{ij}

- Extension to *k* dimensions in the same way

3.4. Interpolation on Grids

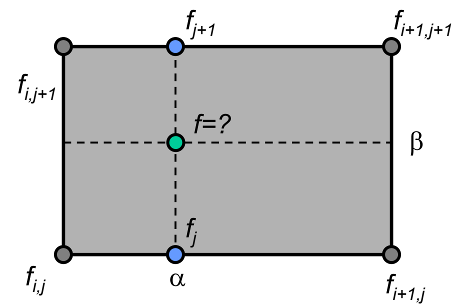
- Bilinear interpolation on a rectangle
  - Tensor product for two linear interpolations
  - 2D local interpolation in a cell
  - Known solution of the linear system of equations for the coefficients c_{ij}
  - Four data points $(x_i,y_j),\ldots,(x_{i+1},y_{j+1})$ with scalar values $f_{i,j}=f(x_i,y_j)$, …
  - Bilinear interpolation of points (x,y) with $x_i \le x < x_{i+1}$ and $y_j \le y < y_{j+1}$



3.4. Interpolation on Grids

- Bilinear interpolation on a rectangle

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



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

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

and local coordinates

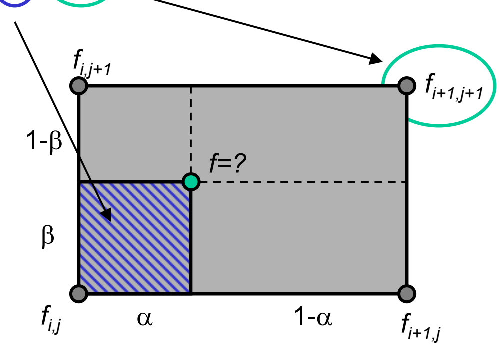
$$\alpha = \frac{x - x_i}{x_{i+1} - x_i}, \qquad \beta = \frac{y - y_i}{y_{i+1} - y_i}, \qquad \alpha, \beta \in [0,1]$$

3.4. Interpolation on Grids

- Bilinear interpolation on a rectangle

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

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



3.4. Interpolation on Grids

- Trilinear interpolation on a 3D uniform grid
  - Straightforward extension of bilinear interpolation
  - Three local coordinates α,β,γ
  - Known solution of the linear system of equations for the coefficients c_{ij}
  - Trilinear interpolation is not linear!
  - Efficient evaluation:
    $f(\alpha,\beta,\gamma)= a+\alpha(b+ \beta(e+h\gamma))+ \beta(c+f \gamma)+ \gamma(d+g\alpha)$
    with coefficients a, b, c, d, e, f, g from data at the corner vertices

- Extension to higher order of continuity
  - Piecewise cubic interpolation in 1D
  - Piecewise bicubic interpolation in 2D
  - Piecewise tricubic interpolation in 3D
  - Based on Hermite polynomials

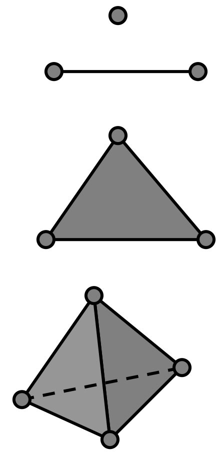
3.4. Interpolation on Grids

- Interpolation on un/structured grids (triangle meshes etc.) ?

- Affine combination of points **x** (in Euclidean space):
  - Linear combination $\Sigma_i\, \alpha_i \cdot \mathbf{x}_i$
  - $0 \le \alpha_i \le 1, \forall i$
  - $\Sigma_i\, \alpha_i = 1$
  - α_i are barycentric coordinates
- Affinely independent set of points:
  - No point can be expressed as affine combination of the other points
  - Maximum number of points is $d+1$ in $\boldsymbol{R}^d$

3.4. Interpolation on Grids

- Simplex in $\boldsymbol{R}^d$
  - $d+1$ affinely independent points
  - Span of these points
  - 0D: point
  - 1D: line
  - 2D: triangle
  - 3D: tetrahedron



3.4. Interpolation on Grids

- Barycentric interpolation on a simplex
  - $d+1$ points $\mathbf{x}_i$ with function values f_i
  - Point $\boldsymbol{x}$ within the simplex described as affine combination of $\mathbf{x}_i$
  - Possible approach:
    solve for coefficients α_i based on $\mathbf{x} = \Sigma_i\, \alpha_i \cdot \mathbf{x}_i$ and $\Sigma_i\, \alpha_i = 1$
  - Function value at $\mathbf{x}$: $f = \Sigma_i\, \alpha_i \cdot f_i$ is affine combination of f_i

- Barycentric coordinates from area/volume considerations:

$$\alpha_i = \frac{\mathbf{Vol}(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1},\mathbf{x},\mathbf{x}_{i+1},\ldots,\mathbf{x}_{d+1})}{\mathbf{Vol}(\mathbf{x}_1,\ldots,\mathbf{x}_{d+1})}$$

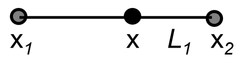
$$\mathbf{Vol}(\mathbf{x}_1,\ldots,\mathbf{x}_{d+1}) = \mathbf{det}\begin{pmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_{d+1} \\ 1 & \cdots & 1 \end{pmatrix}$$

generalized measure for area/volume

3.4. Interpolation on Grids

- Barycentric coordinates from area/volume considerations

d = 1



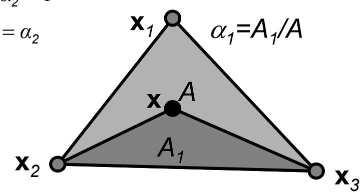
$\alpha_1 = L_1 / L$

Opposite local length

$x = \alpha_1 x_1 + \alpha_2 x_2 \qquad \alpha_1 + \alpha_2 = 1$

$x = (1-u)x_1 + u x_2 \qquad u = \alpha_2$

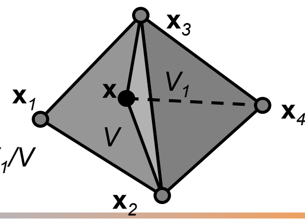
d = 2



$\alpha_1 = A_1 / A$

Opposite local area

d = 3



$\alpha_1 = V_1 / V$

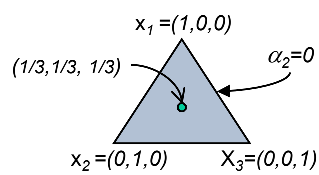
Opposite local volume

3.4. Interpolation on Grids

- Barycentric interpolation in a triangle
  - Geometrically, barycentric coordinates are given by the ratios of the area of the whole triangle and the subtriangles defined by ***x*** and any two points of $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$.

$$\mathbf{Vol}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = \mathbf{det}\begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix} =$$

$$= \pm 2\,\mathrm{Area}(\Delta(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3))$$

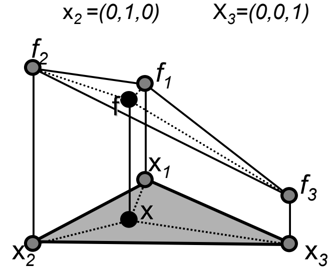


$$\alpha_1 = \frac{\mathbf{Vol}(\mathbf{x}, \mathbf{x}_2, \mathbf{x}_3)}{\mathbf{Vol}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)} \qquad \alpha_1 + \alpha_2 + \alpha_3 = 1$$

$$\mathbf{x} = \alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2 + \alpha_3 \mathbf{x}_3$$

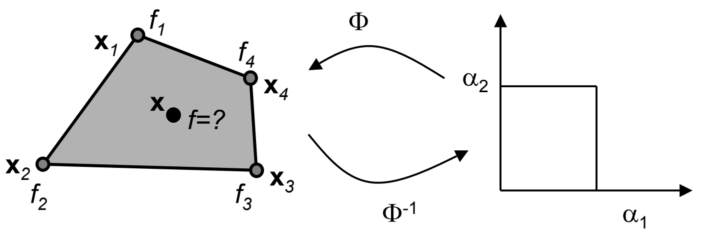
$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix}; \quad \alpha_3 = 1 - \alpha_1 - \alpha_2$$

$$f(\mathbf{x}) = \alpha_1 f_1 + \alpha_2 f_2 + \alpha_3 f_3$$



3.4. Interpolation on Grids

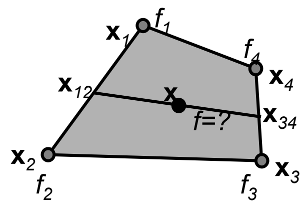
- Interpolation in a generic quadrilateral
  - Main application: curvilinear grids
  - Problem: find a parameterization for arbitrary quadrilaterals



3.4. Interpolation on Grids

- Mapping ϕ from rectangular domain to quadratic domains is known: Bilinear interpolation on a rectangle

$$\mathbf{x}_{12} = \alpha_1 \cdot \mathbf{x}_1 + (1- \alpha_1) \cdot \mathbf{x}_2 \qquad \alpha_1 \in [0,1]$$
$$\mathbf{x}_{34} = \alpha_1 \cdot \mathbf{x}_4 + (1- \alpha_1) \cdot \mathbf{x}_3$$
$$\mathbf{x} = \alpha_2 \cdot \mathbf{x}_{12} + (1- \alpha_2) \cdot \mathbf{x}_{34} \qquad \alpha_2 \in [0,1]$$



- Computing the inverse of ϕ is more complicated:
  - Analytically solve quadratic system for α_1, α_2
  - Or: numerical solution by Newton iteration

- Final value: $f = \alpha_2 \cdot (\alpha_1 \cdot f_1 + (1- \alpha_1) \cdot f_2) + (1- \alpha_2) \cdot (\alpha_1 \cdot f_4 + (1- \alpha_1) \cdot f_3)$

3.4. Interpolation on Grids

- Jacobi matrix $J(\Phi)$
  - $J(\Phi)_{ij} = {}^{\partial\Phi_i}/_{\partial\alpha_j}$
  - $J(\Phi)_{.j}$ describes direction and speed of position changes of Φ when α_j are varied

- Newton iteration

start with seed points as start configuration, e.g., $\alpha_i = {}^1/_2$

while ($\|x - \Phi(\alpha_1,\alpha_2,\alpha_3)\| > \varepsilon$)

  compute $J(\Phi(\alpha_1,\alpha_2,\alpha_3))$

  transform **x** in coordinate system $J(\Phi)$:

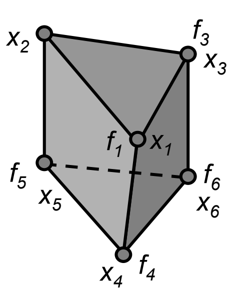
  $$x_\alpha = J(\Phi(\alpha_1,\alpha_2,\alpha_3))^{-1} \cdot (x - \Phi(\alpha_1,\alpha_2,\alpha_3))$$

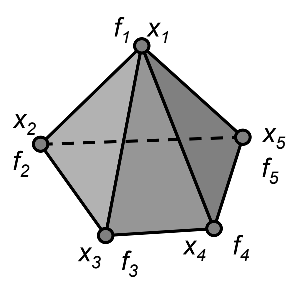
  update $\alpha_i = \alpha_i + x_{\alpha,i}$

maximum error ε

3.4. Interpolation on Grids

- Other primitive cell types possible





Prism:
- twice barycentric
- once linear

Pyramid:
- bilinear on base face
- then linear

3.4. Interpolation on Grids

- Inverse distance weighting
  - Shepard interpolation [D. Shepard, A two-dimensional interpolating function for irregularly spaced data. Proc. ACM. nat. Conf., 517--524, 1968]
  - Originally developed for scattered data
  - Interpolated values: $f(\mathbf{x}) = \Sigma_i \, \phi_i(\mathbf{x}) \, f_i$
  - Sample points are vertices of the cell
  - Basis functions $$\phi_i(\mathbf{x}) = \frac{\|\mathbf{x} - \mathbf{x}_i\|^{-p}}{\sum_j \|\mathbf{x} - \mathbf{x}_j\|^{-p}}$$
  - Define values at sample points $f(\mathbf{x}_i) := f_i = \lim_{x \to x_i} f(\mathbf{x})$

3.5. Interpolation without Grids

- Shepard interpolation
  - Different exponents for inner and outer neighborhood
    (default: 2 in the inner neighborhood and 4 in the outer neighborhood)
  - Neighborhood sizes determine how many points are included in inverse distance weighting
  - The neighborhood size can be specified in terms of
    - Radius or
    - Number of points or
    - Combination of the two
  - Neighborhood is not given explicitly (as opposed to inverse distance weighting on grids)

3.5. Interpolation without Grids

- Radial basis functions (RBF)
  - n function values f_i given at n points $\mathbf{x}_i$
  - Interpolant $$f(\mathbf{x}) = \sum_{i=1}^{n} \lambda_i \, \phi(\|\mathbf{x} - \mathbf{x}_i\|) + \sum_{m=0}^{k} c_m \, p_m(\mathbf{x})$$
  - Univariate radial basis $\phi(r)$
  - Examples:
    - Polynomials r^{ν}
    - Gaussians $\exp(r^{-2})$
  - Polynomial basis $\{p_m\}$ for $(k+1)$-dimensional vector space

3.5. Interpolation without Grids

- Radial basis functions (RBF)
  - Under-determined system:
    n equations for $n + (k+1)$ unknowns
  - Additional constraints (orthogonality / side conditions):

$$\sum_{i=1}^{n} \lambda_i \, p_m(\mathbf{x}_i) = 0 \quad \forall\, m = 0 \ldots k$$

  - Well-defined system of linear equations (vector / matrix notation):

Coefficients for radial function

Function values at sample points

$$\mathbf{A}_{i,j} = \phi(\|\mathbf{x}_i - \mathbf{x}_j\|) \qquad \begin{pmatrix} \mathbf{A} & \mathbf{P} \\ \mathbf{P}^T & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda} \\ \mathbf{c} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{0} \end{pmatrix}$$

Polynomial basis

Coefficients for polynomials

3.6. Filtering by Projection or Selection

- Very often: too much information to be visualized at once
- Strategy is to reduce the displayed information by filtering

- Popular approach:
  Reduce from *n***d***m***v** to *n'***d***m'***v** , with n' < n and / or m' < m [Wong]
- Techniques:
  - Projection
  - Selection
  - Slicing
- User input needed

3.6. Filtering by Projection or Selection

- Projection π
- Functional description for both the
  - Domain and
  - Data values
- Projection into subspaces
- Often a mapping to a subset of the original values is chosen

3.6. Filtering by Projection or Selection

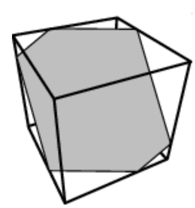
- Selection σ

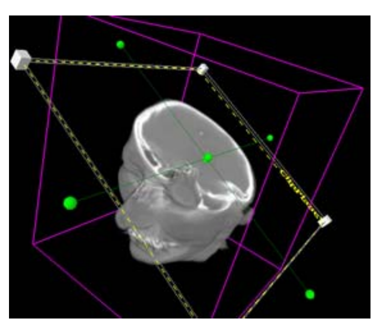
- Selection of data according to logical conditions (predicates)
- Example:
  - Height field **2d***1***v** with data *(x,y,h)*
  - $D_\sigma = \{ (x,y,h) \mid (x^2+y^2 < 5km) \wedge (h > 1km) \}$

3.6. Filtering by Projection or Selection

- Slicing

- Example: 2D cutting surface (slice) through a 3D volume





3.7. Fourier Transform

- Fourier analysis
- Function *h(t)* in coordinate representation (time domain)
- Analogous representation $H(\nu)$ with frequencies ν (frequency domain)
- Fourier transform:

$$H(\nu) = \int_{-\infty}^{\infty} h(t) e^{-2\pi i \nu t} dt \quad \Leftrightarrow \quad h(t) = \int_{-\infty}^{\infty} H(\nu) e^{2\pi i \nu t} d\nu$$

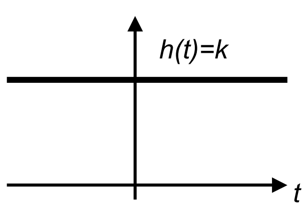
forward transform   inverse transform

- Convolution $$(g * h)(t) = \int_{-\infty}^{\infty} g(\tau) \cdot h(t - \tau) d\tau$$

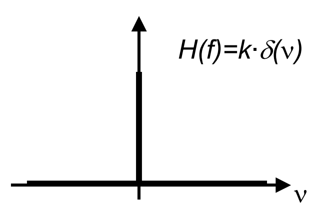
- Convolution theorem: $(g * h)(t) \Leftrightarrow G(\nu) \cdot H(\nu)$

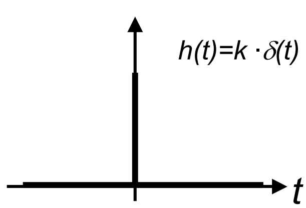
3.7. Fourier Transform

- Examples

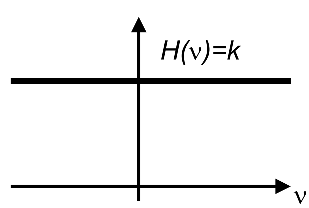


$\Leftrightarrow$



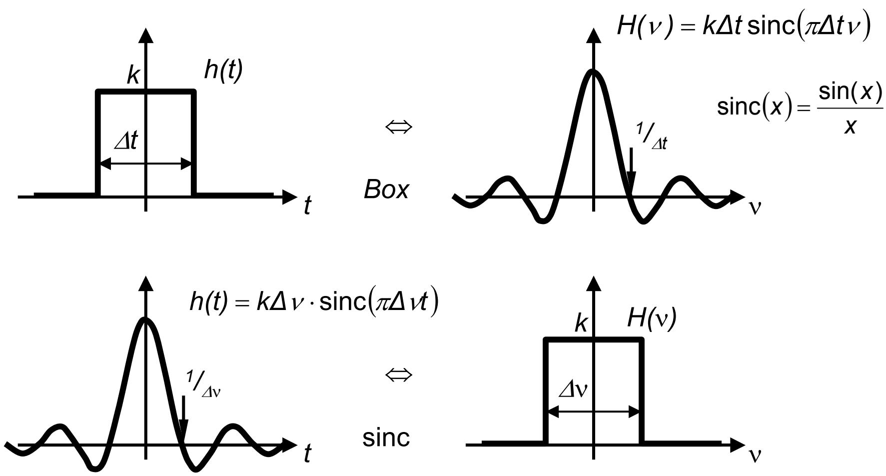


$\Leftrightarrow$



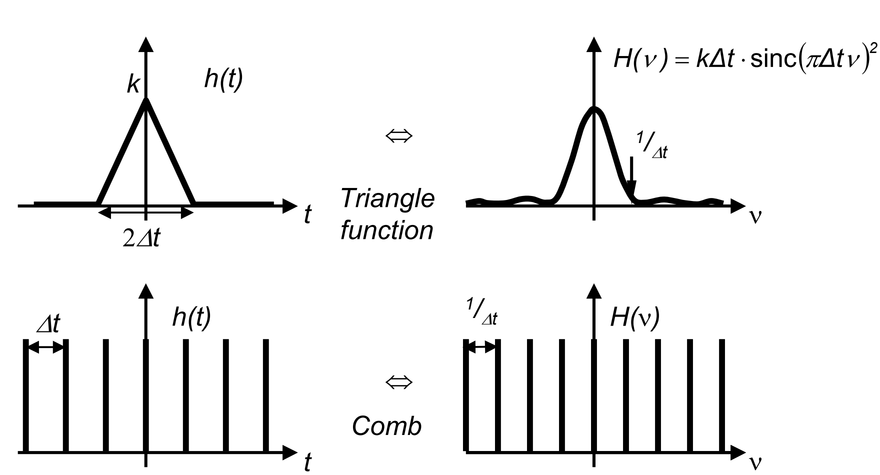
3.7. Fourier Transform

- Examples



3.7. Fourier Transform

- Examples



3.7. Fourier Transform

- In applications: mostly discrete Fourier transforms
- Based on a discrete signal
- Implementation in the form of the Fast Fourier Transform (FFT)

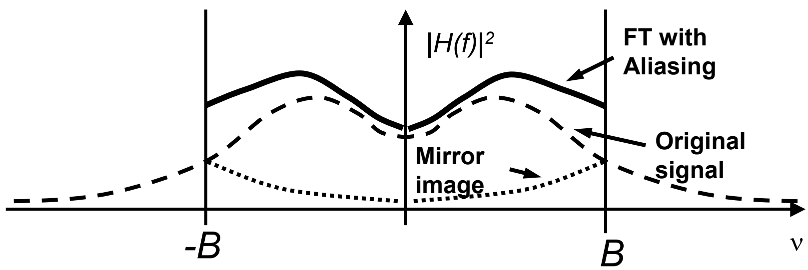
3.8. Sampled Signals

- $h(t)$ is assumed to be band limited with frequencies smaller than B
- Nyquist frequency $\nu_{Nyq} = 2B$
- Discretization with constant step size $\Delta t = 1/\nu_{Nyq} = 1/(2B)$
- Sampled signal: $h_j = h(j \cdot \Delta t)$
- Periodicity is assumed if only a finite interval $j = 0..n-1$ is considered
- Sampling theorem (Shannon 1949):
  If $H(f) = 0$ for all $|\nu| > B = \nu_{Nyq} / 2$, then $h(t)$ is uniquely given by the samples h_i:

$$h(t) = \Sigma_{j=0..n-1} h_j \cdot \mathrm{sinc}(\pi \nu_{Nyq}(t - j \cdot \Delta t))$$

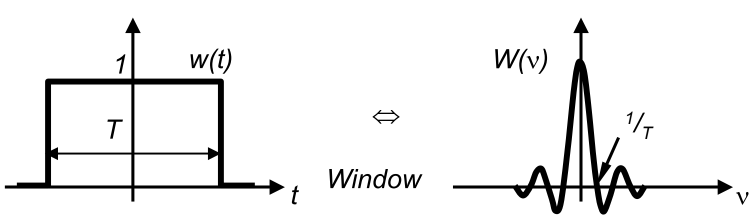
3.8. Sampled Signals

- Issue 1: Undersampling
  - If $h(t)$ has frequencies larger than $B = \nu_{Nyq} / 2$
  - $h(t)$ cannot be reconstructed from sampled values
  - Aliasing



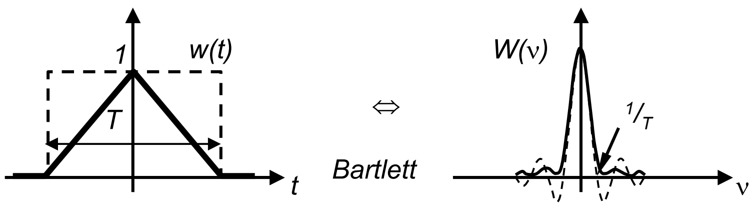
3.8. Sampled Signals

- Issue 2: Finite window size
  - Fourier transform is theoretically defined for signals of infinite duration or for periodic signal
  - Often $h(t)$ is measured on a finite interval $[-T/2, T/2]$ (without periodicity)
  - Yielding a multiplication with a window function: $h(t) \cdot \mathbf{1}_{[-T/2, T/2]}(t)$
  - Convolution with *sinc()* function in frequency space



3.8. Sampled Signals

- Issue 2: Finite window size
  - Problem: Differences between the starting and ending values of the segment produces a discontinuity which generates high-frequency spurious components
  - Solution: Data windowing
  - Bartlett window is often used

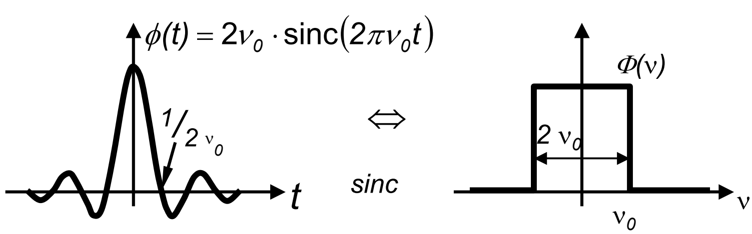


  - Other examples: Hamming, Hann windows

3.9. Reconstruction and Frequency Filtering

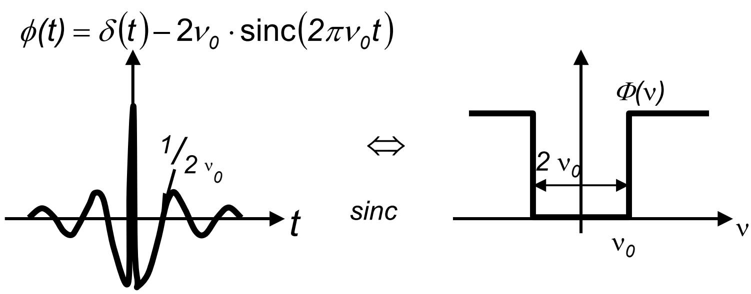
- Filter design based on Fourier analysis

- Low pass filter with limit frequency ν_0:
  - Convolution with sinc() function (in coordinate space) or
  - FFT, then multiplication with box filter $\Phi(\nu)$, then inverse FFT



3.9. Reconstruction and Frequency Filtering

- High pass filter with limit frequency ν_0:
  - Emphasizes features, e.g., edges



3.9. Reconstruction and Frequency Filtering

- Reconstruction issues
  - Measurements *m(t)* of the original signal *s(t)* are based on a point-spread function $p(t-t_i)$, not on the ideal delta function $\delta(t-t_i)$
  - Convolution in coordinates space, multiplication in frequency space

$$m(t) = \int_{-\infty}^{\infty} p(t-\tau)s(\tau)d\tau \quad \Leftrightarrow \quad M(\nu) = P(\nu)S(\nu)$$

  - Additional noise
  - What is the ideal, original signal *s(t)*?

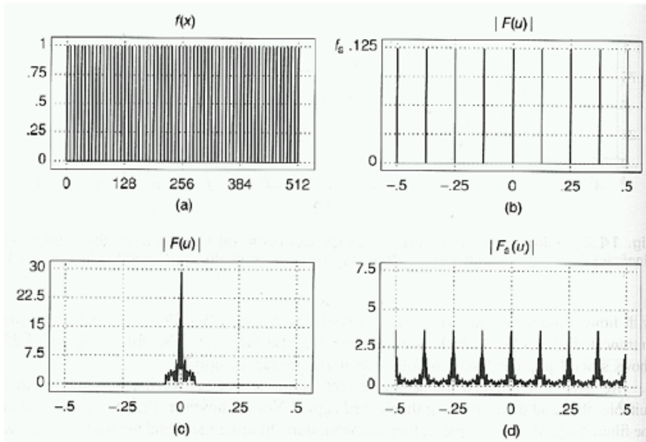
3.9. Reconstruction and Frequency Filtering

Sampling

Sampling in frequency domain

Signal in frequency space

Sampled signal in frequency domain



3.9. Reconstruction and Frequency Filtering

Function

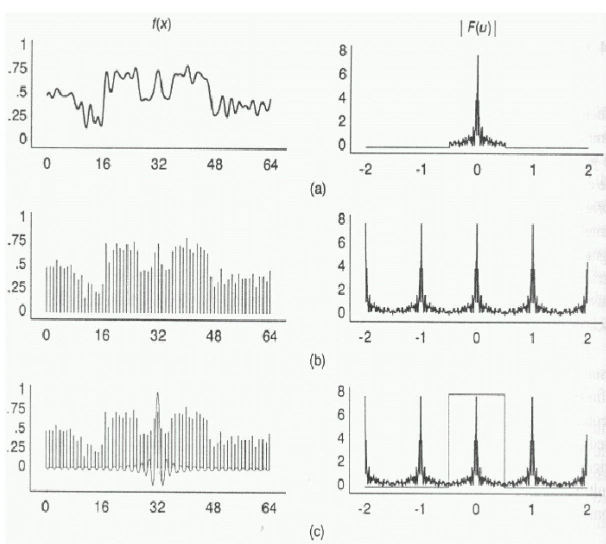
Function in frequency domain

Sampled signal

Signal in frequency domain

Before re-construction with *sinc()*

Box filter in frequency domain

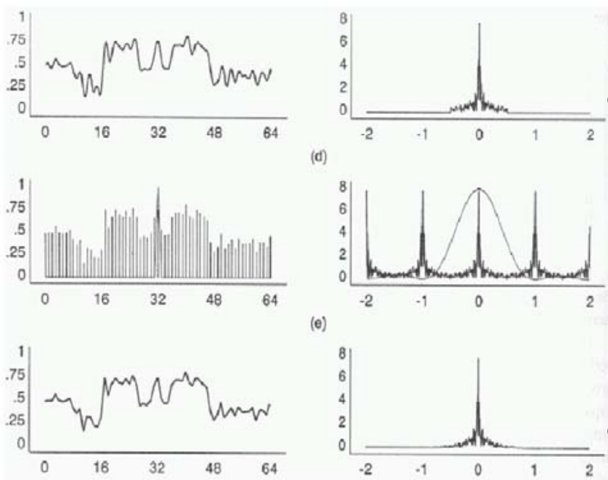


3.9. Reconstruction and Frequency Filtering

Reconstruction

Before re-construction with triangle

Reconstruction with triangle



Reconstruction in frequency domain

Triangle in frequency domain: *sinc()*^{2}

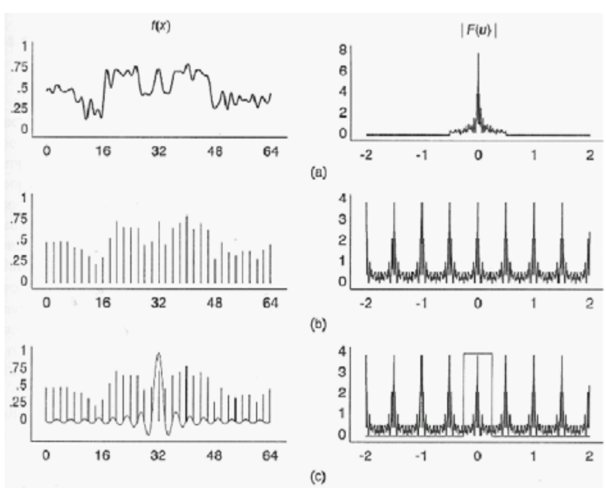
Reconstruction in frequency domain

3.9. Reconstruction and Frequency Filtering

Function

Under-sampled signal

Before re-construction with *sinc()*



Function in frequency domain

Signal in frequency domain

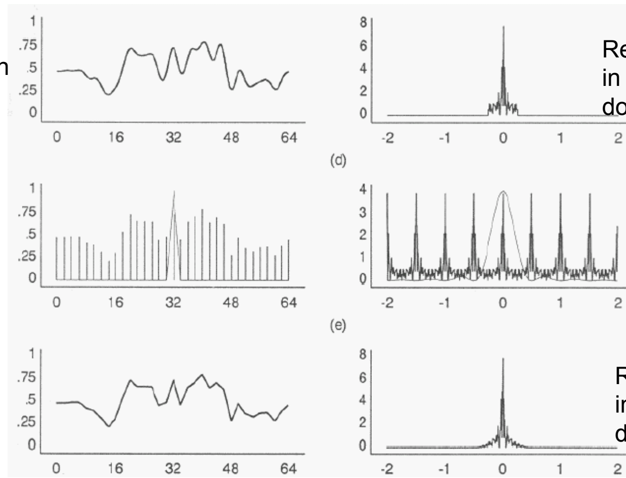
Box filter in frequency domain

3.9. Reconstruction and Frequency Filtering

Reconstruction

Before re-construction with triangle

Reconstruction with triangle



Reconstruction in frequency domain

Triangle in frequency domain: *sinc()*^{2}

Reconstruction in frequency domain

3.9. Reconstruction and Frequency Filtering

- Demo (Applet)