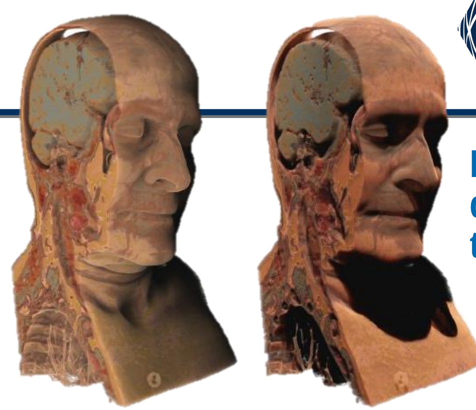
Volume Visualization and Rendering

Intro & Data Preparation

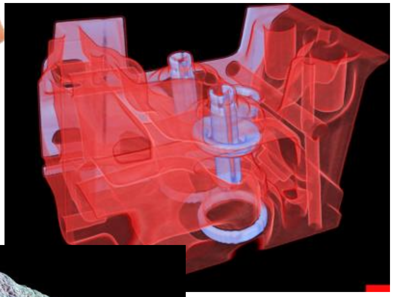
Data Sources

- driver application is medical imaging: CT, MRI, ultra sound, etc.
- material science: engine block, 3D print preview, etc.
- biology: 3D microscopy, electron microscopy, NanoCT, etc.
- simulation: particles, finite elementes, feature film, etc.



Medical Dataset with two different illumination techniques (pdf)

image: Oliver Kreylos



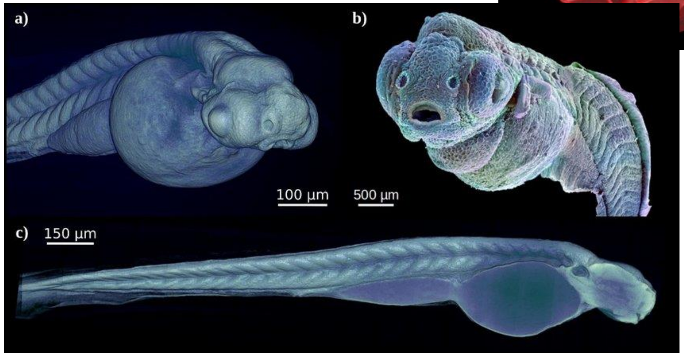
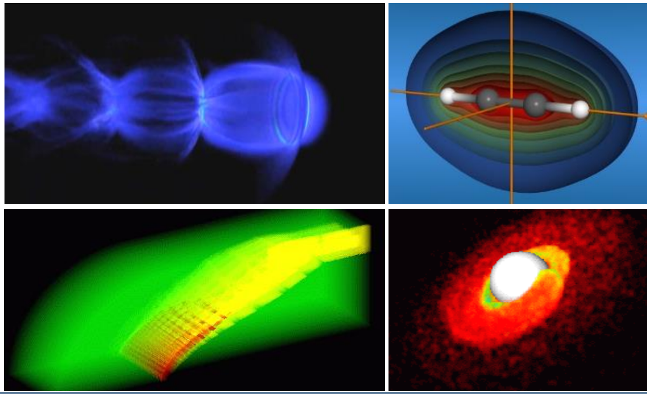


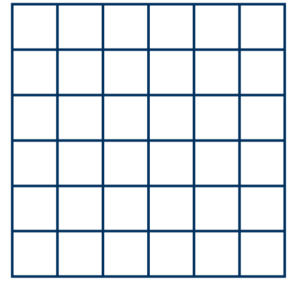
image: Mark Müller

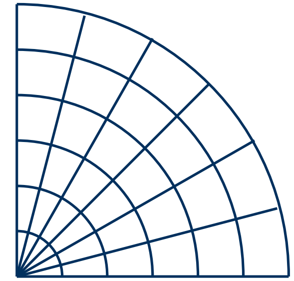
image: rebelway

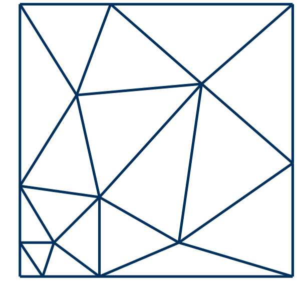


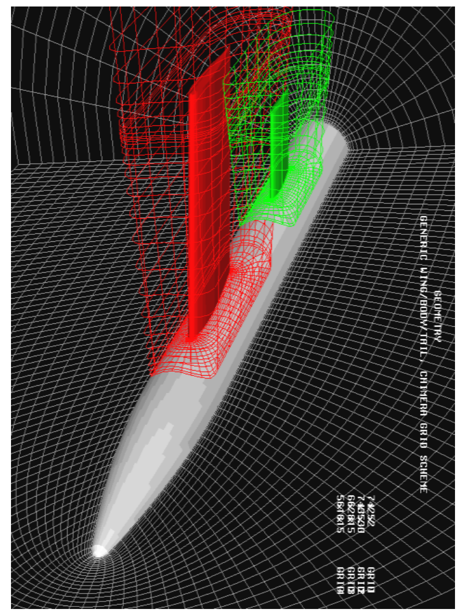
Data Specification

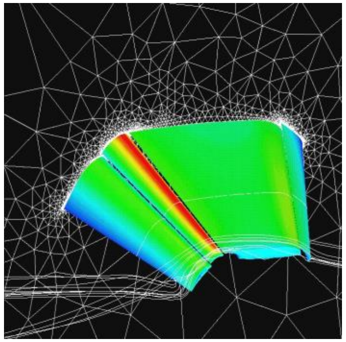
- Observation space: $\boldsymbol{R}^3$
- Grid types:
  - Mostly regular grids (voxel grids)
  - unstructured grids (tetrahedral mesh)
  - curvi-linear grids
  - scattered data without grid
  - sliced data
- Feature space: $S \in [a, b]$ e.g. $[0, 255]$
  - Often we only consider a single scalar feature at a time


regular grid


structured grid

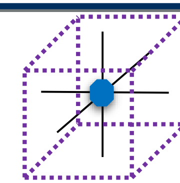

tetrahedral mesh


curvi-linear grid from simulation


tetrahedral mesh

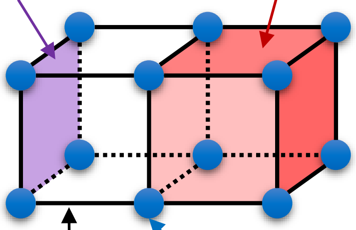

slices from microscope

Data Specification – Voxels



Voxel Grid

- **voxel** (**vol**ume **el**ement) corresponds to observation point with feature value (vertices of voxel grid)
- **edge** connects two voxels
- **cell** cube/tet spanned by 8/4 voxels
- **face** separates two cells

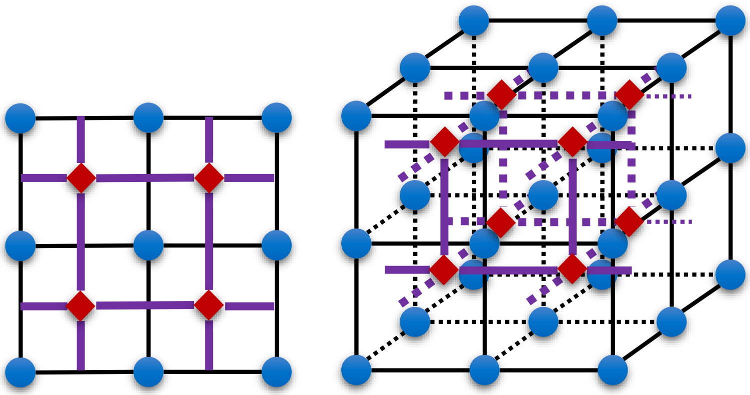


Dual grid

- **dual cell** one per vertex corresponding to Voronoi cell
- **dual vertex** one per cell
- **dual edge** one per face: connects dual vertices

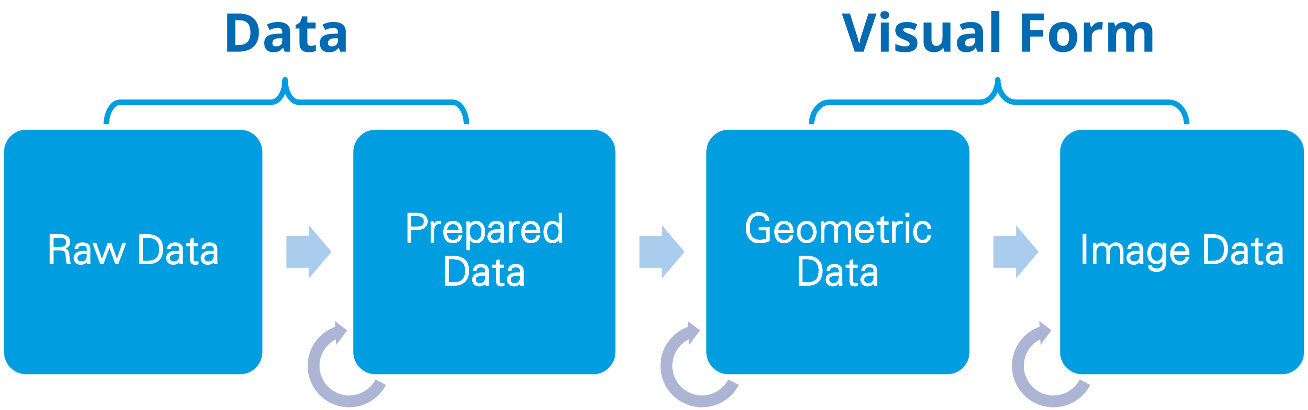
interpolation schemes

- nearest neighbor: voxel values are constant over dual cells
- trilinear: voxel defines value at corner of $2^3 =$ 8 incident cells



Volume Visualization Pipeline

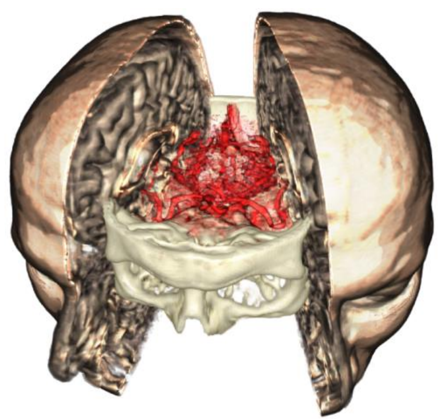
Computer Graphics and Visualization



Data Preparation
Noise Filtering
Reconstruction
Interpolation
Derivative Estimation
Histogram Extraction

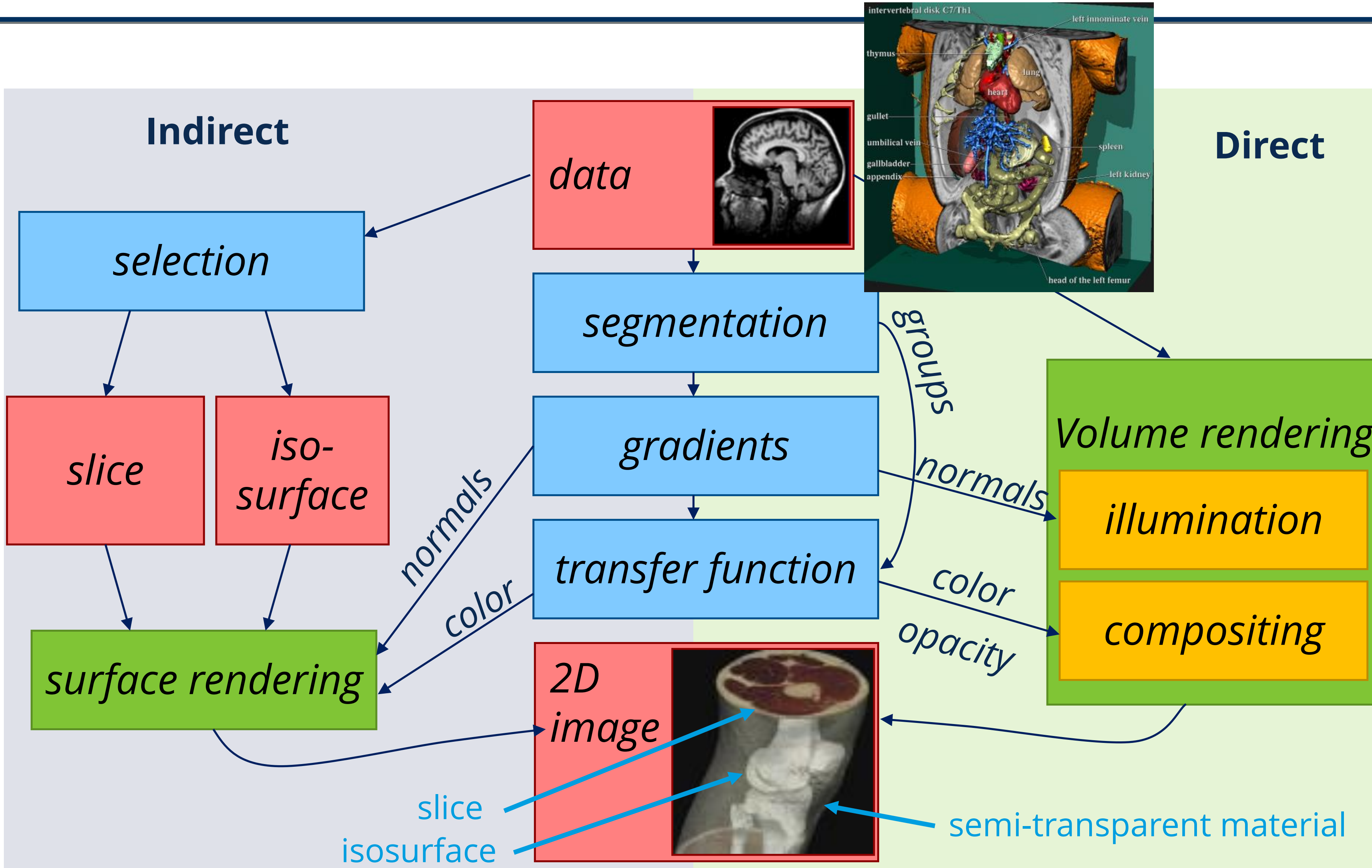
Visual Mappings
Transfer Function Design
Selection
Cutting

3D Rendering
Visibility Sorting
Rendering Algorithms
Lighting



Example for Cutting

Volume Visualization – Overview



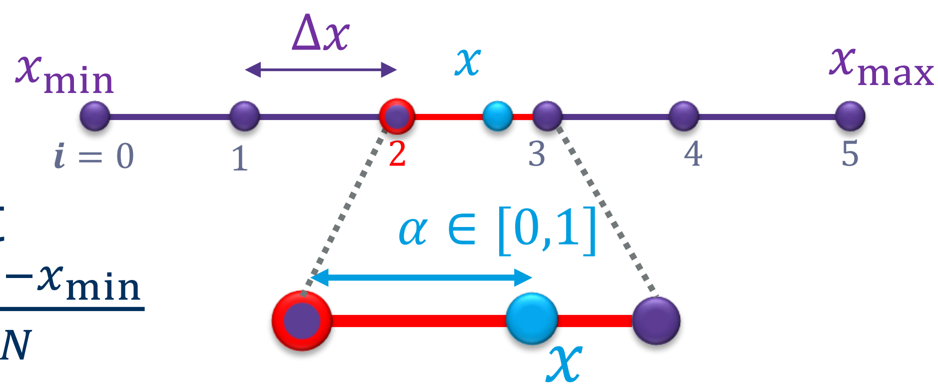
Content

- Data Preparation
  - Reconstruction
  - Tetrahedral meshes
  - Filtering
- Indirect Volume Visualization
  - Slicing
  - Contouring
- Direct Volume Visualization
  - Compositing
  - Volume Rendering Integral
  - Transfer Functions & Pre-Integration
  - Rendering Algorithms
  - Continuous Histograms & Scatter Plots
  - Multi-Dimensional Transfer Functions

Data Preparation

RECONSTRUCTION

Grids – Linear Interpolation



Input

- extent: $[x_{\min}, x_{\max}]$
- $N + 1$ scalars S_i sampled at $x_i = x_{\min} + i \cdot \Delta x,\ \ \Delta x = \frac{x_{\max} - x_{\min}}{N}$

Point Location

- for given x we need to determine index i and local coordinates α first

$$i = \text{floor}\left(\frac{x - x_{\min}}{\Delta x}\right)$$

$$\alpha = \frac{x - (x_{\min} + i \cdot \Delta x)}{\Delta x}$$

Interpolation

- Finally the scalar is interpolated from adjacent samples with function $s_i(x)$

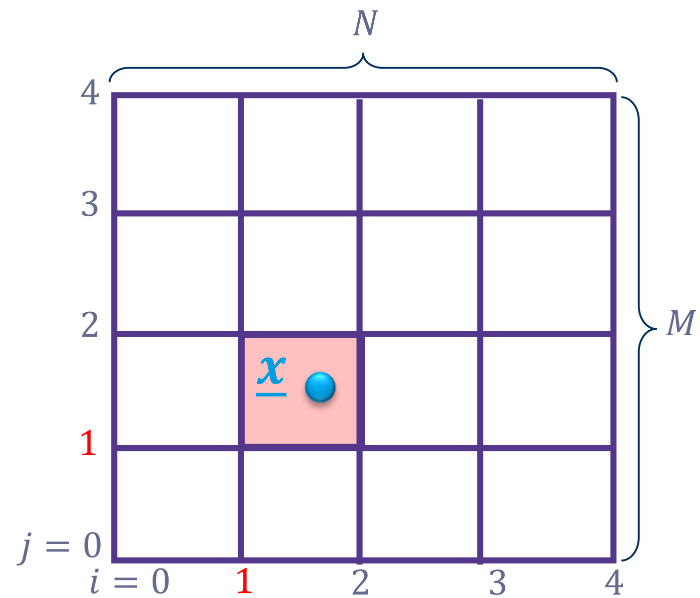
$\forall x \in [x_i, x_i + 1]$:

$$s_i(x) \coloneqq \text{mix}(S_i, S_{i+1}, \alpha) = (1 - \alpha) S_i + \alpha S_{i+1}$$

Grids – Multi-Linear Interpolation

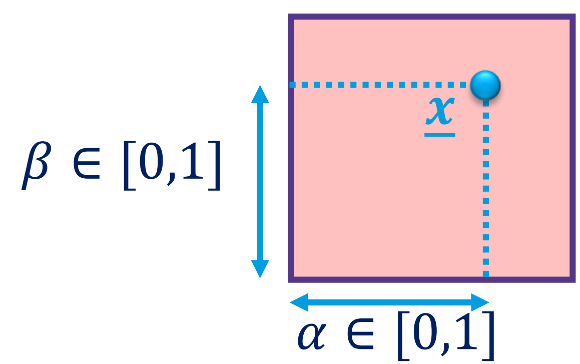
Input

- extent: $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$
- $(N+1) \times (M+1)$ samples S_{ij} at $\begin{pmatrix} x_i \\ y_j \end{pmatrix} = \begin{pmatrix} x_{\min} + i \cdot \Delta x \\ y_{\min} + j \cdot \Delta y \end{pmatrix}$,



Point Location

- for given (x, y) determine indices i, j and local coordinates α, β as in linear case



Interpolation

- Bilinear interpolation function is linear interpolation along y of linear interpolants along x (tensor product construction)

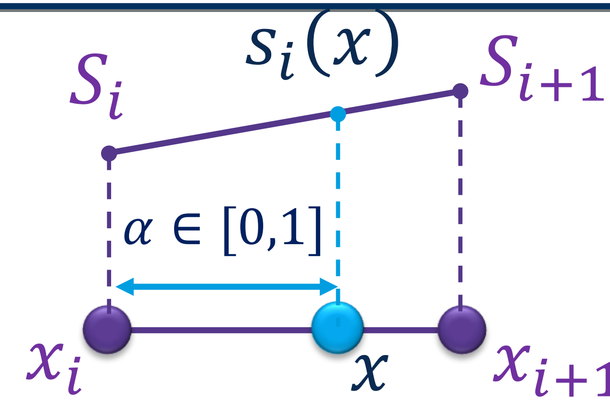
$$\forall (x, y) \in [x_i, x_i + 1] \times [y_j, y_{j+1}]:$$
$$\sigma_{ij}(\alpha) := \mathrm{mix}\left(S_{ij}, S_{(i+1)j}, \alpha\right)$$
$$s_{ij}(\underline{\boldsymbol{x}}) := \mathrm{mix}\left(\sigma_{ij}(\alpha), \sigma_{i(j+1)}(\alpha), \beta\right)$$

B-Spline Interpretation

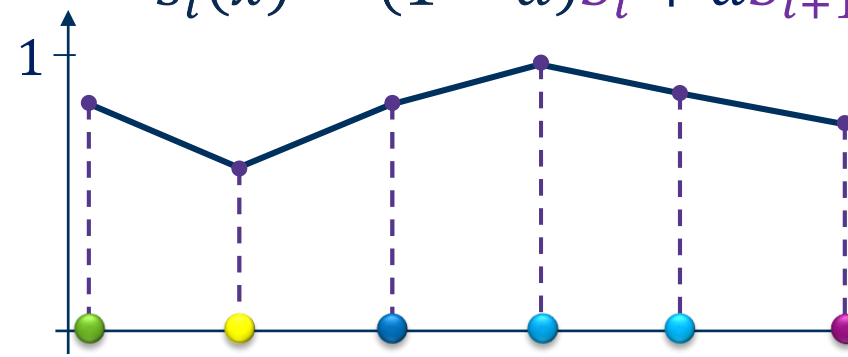
- Linear interpolation gives continuous piecewise linear function with a jump in the derivative at the samples
- It is basically a degree 1 B-spline:

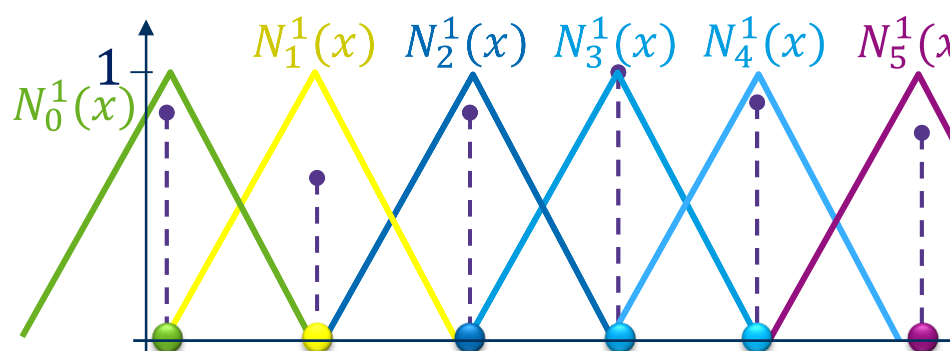
$$s(x) = \sum_{i=0}^{N} S_i N_i^1(x)$$

- With the natural basis function $N_i^1(x)$ that have a triangular shape



$$s_i(x) = (1-\alpha)S_i + \alpha S_{i+1}$$





Why cubic interpolation?

Computer Graphics and Visualization

$$\rho(x,y,z) = \frac{(1 - \sin(\pi z/2) + \alpha(1 + \rho_r(\sqrt{x^2+y^2}))}{2(1+\alpha)},$$

where

$$\rho_r(r) = \cos(2\pi f_M \cos(\frac{\pi r}{2})).$$

Marschner, S. R., & Lobb, R. J. (1994, October). An evaluation of reconstruction filters for volume rendering. In *Proceedings Visualization'94* (pp. 100-107). IEEE.

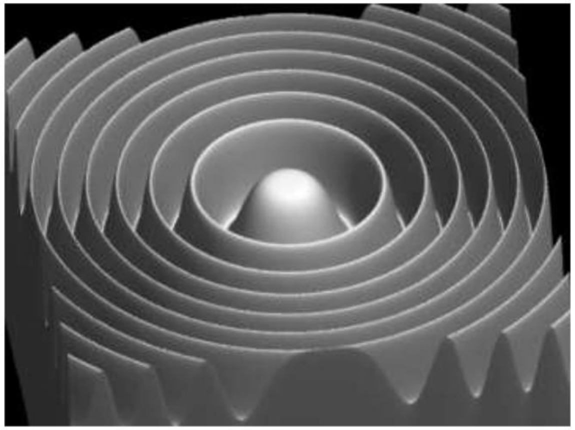
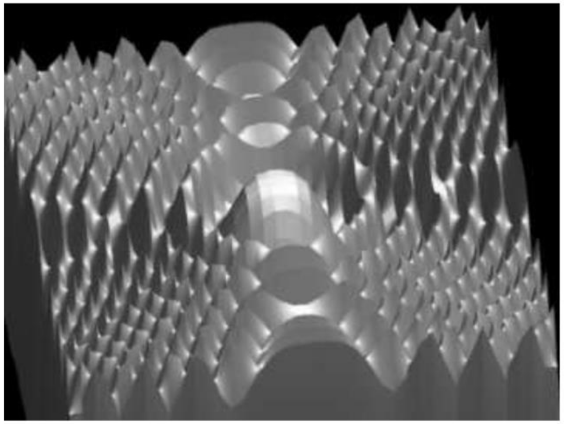
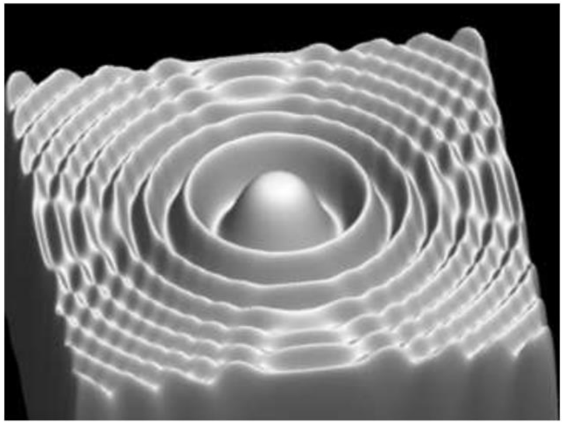


Figure 5: The unsampled test signal.

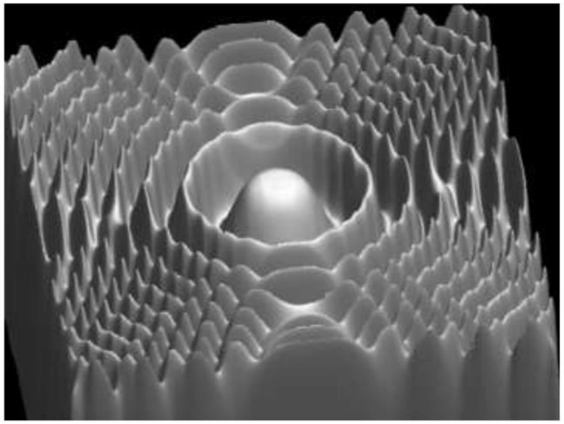
isosurface $\rho(x,y,z) = 0.5$



(d) Trilinear



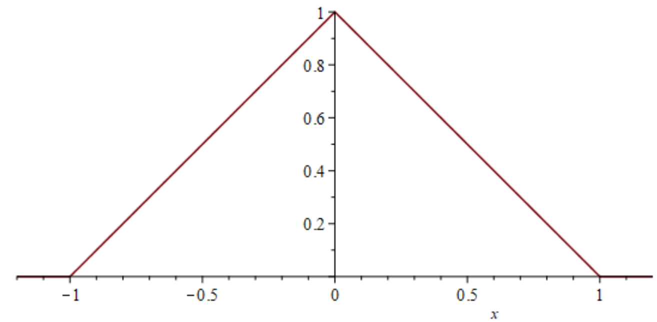
(a) B-spline



(b) Catmull-Rom

Convolution Interpretation

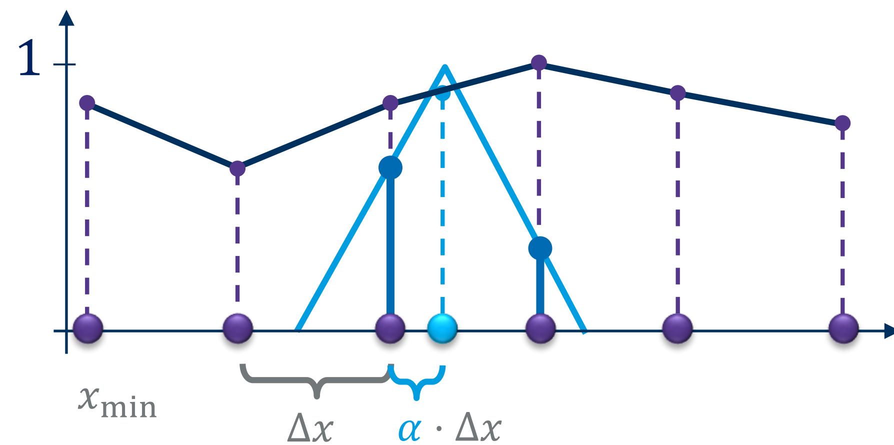
- Given the filter kernel
$$h(x) = N_0^1(x)$$



$$h(x) = \begin{cases} 1 - |x| & |x| < 1 \\ 0 & \text{otherwise} \end{cases}$$

- The linear interpolant at x can be computed with a discrete convolution directly:
$$s(x) = \sum_{i=0}^{N} S_i \cdot h\left(\frac{x_{\min} + i\Delta x - x}{\Delta x}\right)$$

- As $h(x)$ has support $[-1,1]$, the convolution simplifies with $x = (\alpha + i) \cdot \Delta x + x_{\min}$ to:
$$s(x) = h(-\alpha)S_i + h(1-\alpha)S_{i+1}$$
$$= \omega_0(\alpha)S_i + \omega_1(\alpha)S_{i+1}$$
$$= (1-\alpha)S_i + \alpha \cdot S_{i+1}$$

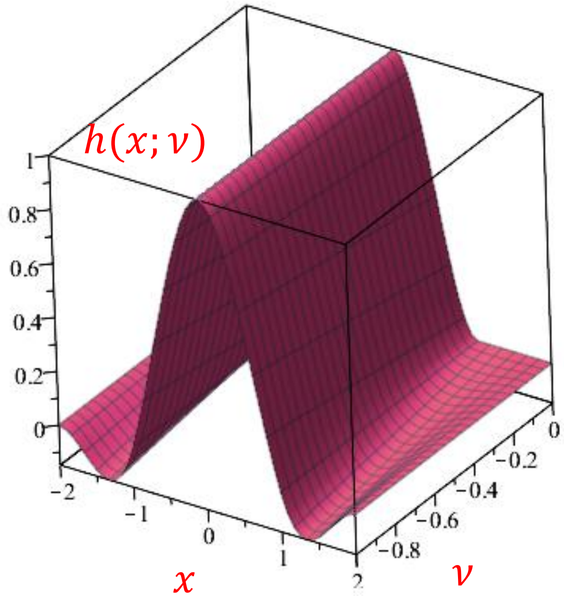


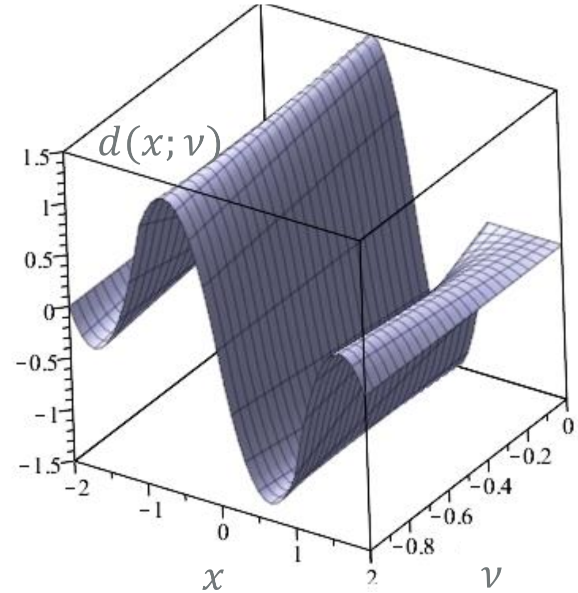
Cubic Interpolation

- Keys developed 1981 a one parameter family $h(x;\nu)$ of interpolating cubic kernels, where derivative can also be computed with unsymmetric derivative filter $d(x;\nu)$:

$$h := x \mapsto \begin{cases} (\nu+2)\,|x|^3 - (\nu+3)\,|x|^2 + 1 & |x| < 1 \\ \nu\,|x|^3 - 5\,\nu\,|x|^2 + 8\,\nu\,|x| - 4\,\nu & 1 \le |x| < 2 \\ 0 & \textit{otherwise} \end{cases}$$

$$d := x \mapsto \begin{cases} -3\,\nu\,x^2 - 10\,\nu\,x - 8\,\nu & -2 < x \le -1 \\ -(3\,\nu+6)\,x^2 - (2\,\nu+6)\,x & -1 < x \le 0 \\ (3\,\nu+6)\,x^2 - (2\,\nu+6)\,x & 0 < x < 1 \\ 3\,\nu\,x^2 - 10\,\nu\,x + 8\,\nu & 1 \le x < 2 \\ 0 & \textit{otherwise} \end{cases}$$



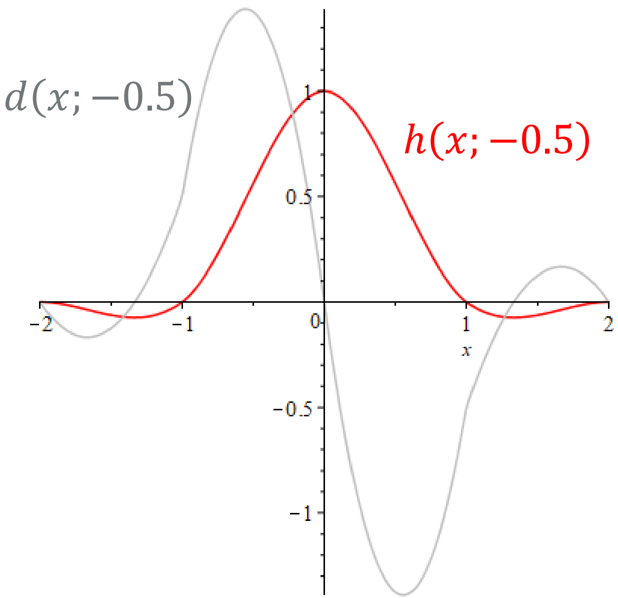


Cubic Interpolation

- Keys found that parameter $\nu = -0.5$ yields the best approximation performance

$$h(x;-0.5) = \begin{cases} 0 & x \le -2 \\ 0.5\,(x+2.)^2\,(x+1.) & x \le -1 \\ 1 - 1.5\,x^3 - 2.5\,x^2 & x < 0 \\ 1 + 1.5\,x^3 - 2.5\,x^2 & x < 1 \\ -0.5\,(x-1.)\,(x-2.)^2 & x < 2 \\ 0 & 2 \le x \end{cases}$$

$$d(x;-0.5) = \begin{cases} 0 & x \le -2 \\ 1.5\,x^2 + 5.0\,x + 4.0 & x \le -1 \\ -4.5\,x^2 - 5.0\,x & x \le 0 \\ 4.5\,x^2 - 5.0\,x & x < 1 \\ -1.5\,x^2 + 5.0\,x - 4.0 & x < 2 \\ 0 & 2 \le x \end{cases}$$

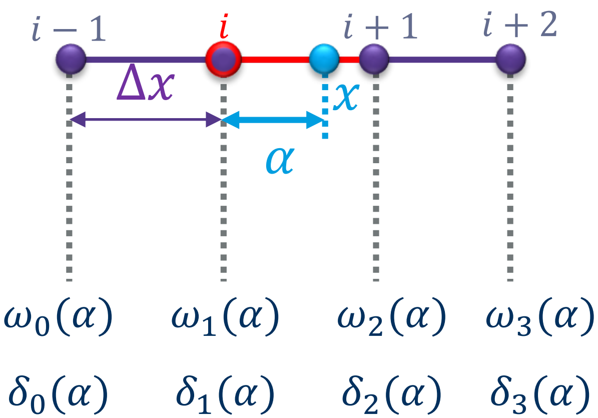


Cubic Interpolation

- For a 1D cubic interpolation one also first determines i and α.
- From α one computes the weights $\omega_i(\alpha)$ and or $\delta_i(\alpha)$ for the function value and or its derivative:

$$\omega_0(\alpha) = \nu\alpha^3 - 2\nu\alpha^2 + \nu\alpha$$
$$\omega_1(\alpha) = (\nu+2)\alpha^3 - (\nu+3)\alpha^2 + 1$$
$$\omega_2(\alpha) = -(\nu+2)\alpha^3 + (2\nu+3)\alpha^2 - \nu\alpha$$
$$\omega_3(\alpha) = -\nu\alpha^3 + \nu\alpha^2$$
$$\delta_0(\alpha) = 3\nu\alpha^2 - 4\nu\alpha + \nu$$
$$\delta_1(\alpha) = 3(\nu+2)\alpha^2 - 2(\nu+3)\alpha$$
$$\delta_2(\alpha) = -3(\nu+2)\alpha^2 + 2(2\nu+3)\alpha - \nu$$
$$\delta_3(\alpha) = -3\nu\alpha^2 + 2\nu\alpha$$

- And then function value and or deriv.



$$s(x) = \omega_{0(\alpha)}S_{i-1} + \omega_{1(\alpha)}S_i + \omega_{2(\alpha)}S_{i+1} + \omega_{3(\alpha)}S_{i+2}$$

$$s'(x) = \delta_{0(\alpha)}S_{i-1} + \delta_{1(\alpha)}S_i + \delta_{2(\alpha)}S_{i+1} + \delta_{3(\alpha)}S_{i+2}$$

BC-Splines

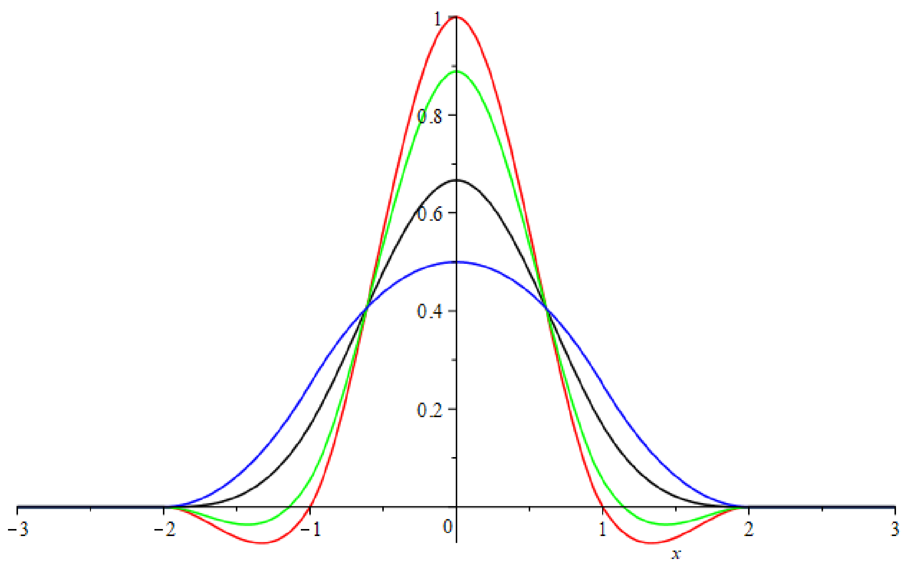
Computer Graphics and Visualization

- Mitchell et al. proposed 1988 a two parameter kernel family $h(x;B,C)$ without the interpolation constraint:

$$h(x;B,C) = \frac{1}{6}\begin{cases}(12-9B-6C)|x|^3 + (-18+12B+6C)|x|^2+(6-2B) & \text{if } |x|<1 \\ (-B-6C)|x|^3+(6B+30C)|x|^2+(-12B-48C)|x|+(8B+24C) & \text{if } 1\le|x|<2 \\ 0 & \textit{otherwise}\end{cases}$$

it includes several known cases for appropriate (B,C):

- $(1,0)$ … standard B-Spline
- $(0,0.5)$ … Catmull Rom Spline (Hermite Spline with derivatives from finite differences)
- $(1.5,-0.25)$ … notch-spline with good antialiasing
- $(1/3,1/3)$ … best looking results for 2D image reconstruction



plot of known spline cases colored is in text

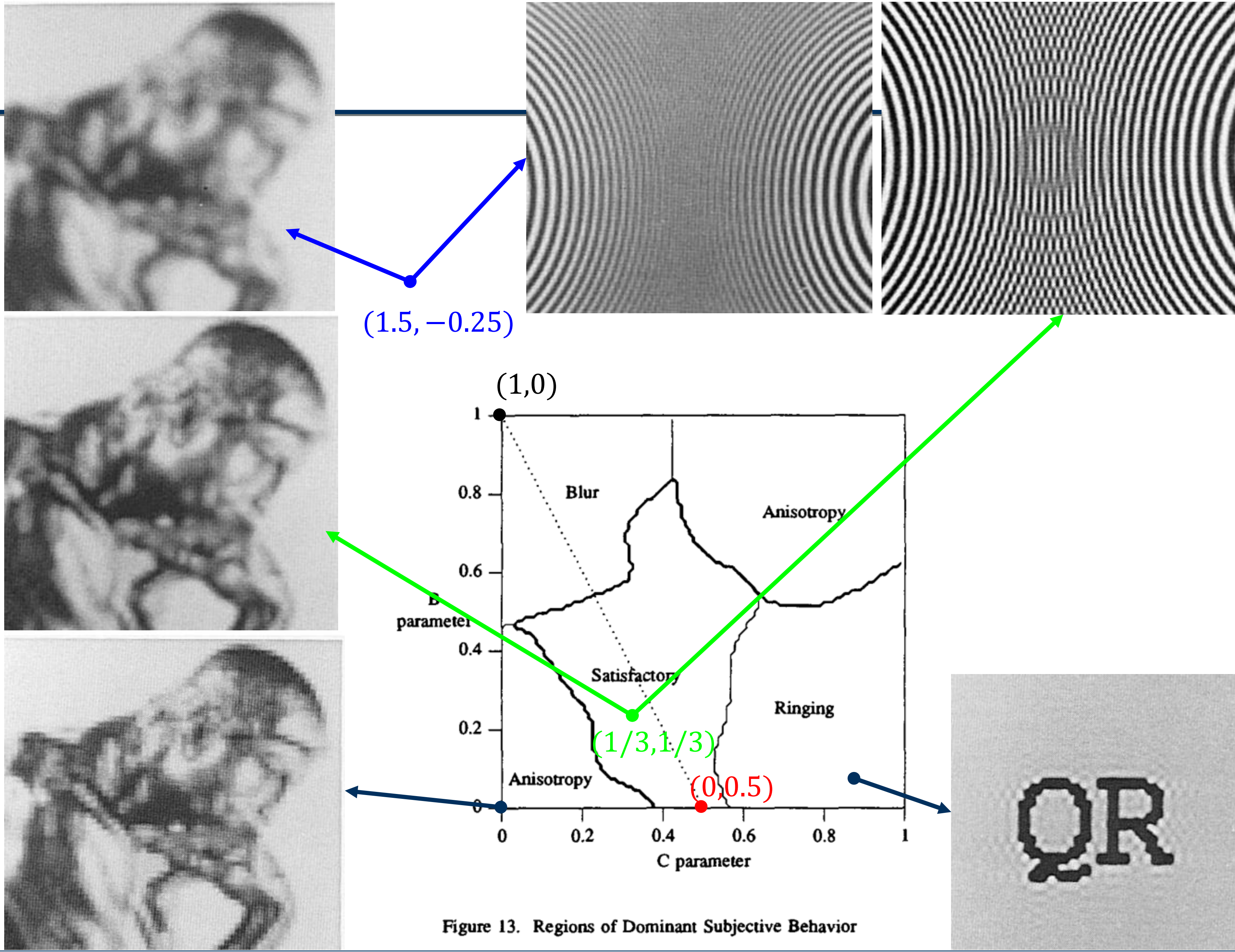
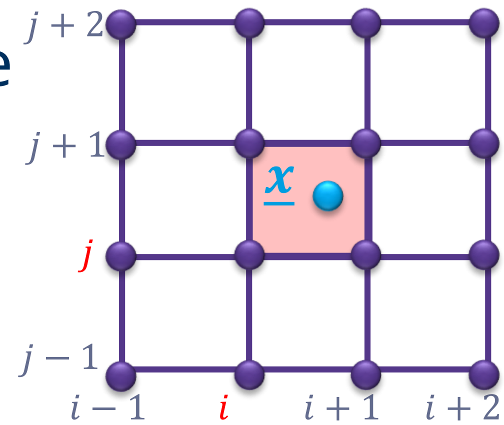


Figure 13. Regions of Dominant Subjective Behavior

Multi-Cubic Interpolation

- For the 2D and 3D case one uses again the tensor product construction on the matrix

$$\mathbf{S}^{ij} = \begin{bmatrix} S_{(i-1)(j-1)} & S_{i(j-1)} & S_{(i+1)(j-1)} & S_{(i+2)(j-1)} \\ S_{(i-1)j} & S_{ij} & S_{(i+1)j} & S_{(i+2)j} \\ S_{(i-1)(j+1)} & S_{i(j+1)} & S_{(i+1)(j+1)} & S_{(i+2)(j+1)} \\ S_{(i-1)(j+2)} & S_{i(j+2)} & S_{(i+1)(j+2)} & S_{(i+2)(j+2)} \end{bmatrix}$$



- let $\vec{\boldsymbol{\omega}}(\alpha) = (\omega_0(\alpha) \quad \omega_1(\alpha) \quad \omega_2(\alpha) \quad \omega_3(\alpha))^T$ be the weight vector of kernel $h(x;\nu)$ and $\vec{\boldsymbol{\omega}}(\beta)$ the one for $h(y;\nu)$, then the 2D tensor product is computed from

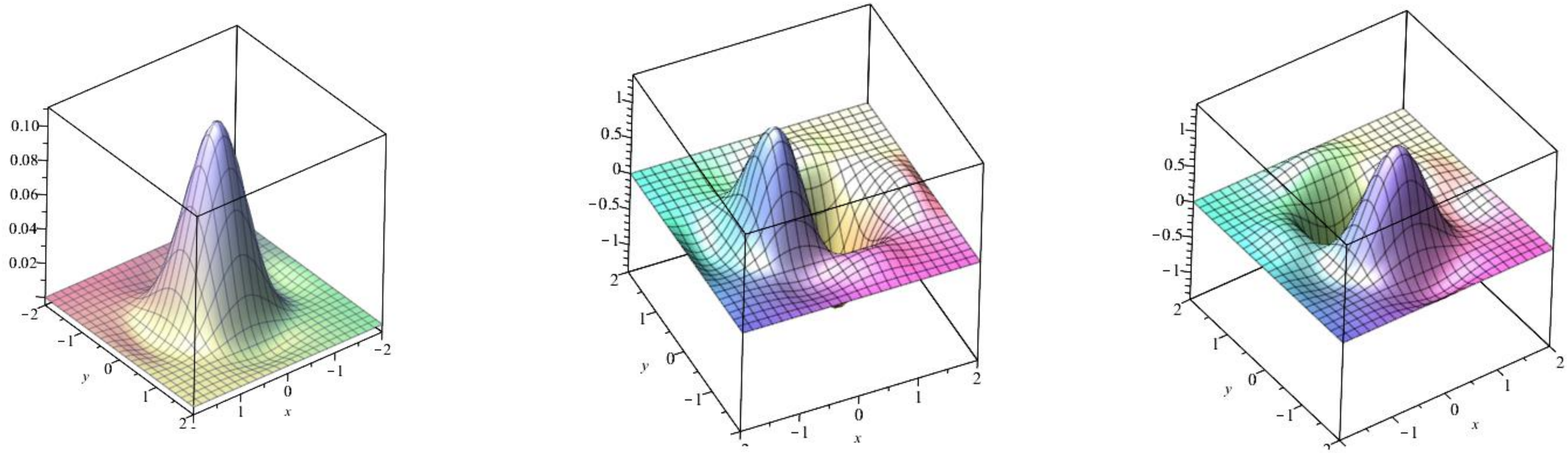
$$s_{ij}(\alpha,\beta) = \vec{\boldsymbol{\omega}}^T(\beta)\mathbf{S}^{ij}\vec{\boldsymbol{\omega}}(\alpha)$$

- Similarly the derivatives for x and y compute to

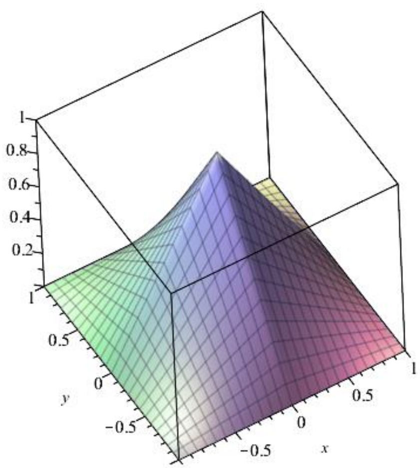
$$\partial_x s_{ij}(\alpha,\beta) = \vec{\boldsymbol{\omega}}^T(\beta)\mathbf{S}^{ij}\vec{\boldsymbol{\delta}}(\alpha)$$

$$\partial_y s_{ij}(\alpha,\beta) = \vec{\boldsymbol{\delta}}^T(\beta)\mathbf{S}^{ij}\vec{\boldsymbol{\omega}}(\alpha)$$

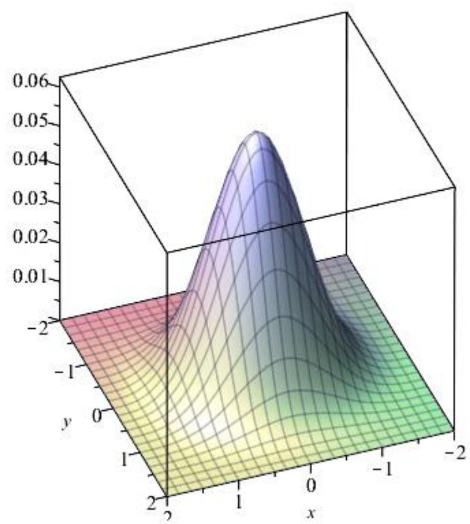
Multi-Cubic Interpolation



Tensor product kernels for $\nu = 0.5$, left to right: $h(x;\nu) \otimes h(y;\nu)$, $d(x;\nu) \otimes h(y;\nu)$, $h(x;\nu) \otimes d(y;\nu)$



Bilinear filter



B-Spline Tensor Product Filter

Fast GPU-Evaluation of Cubic Interp.

- GPUs are highly optimized for bilinear and trilinear interpolated texture access
- Ruijters et. al extended 2008 the work of Hartwinger et al. from 2005, in which cubic interpolation in n-dimensional space can be evaluated with 2^n multi-linear texture lookups instead of 4^n unfiltered lookups
- The basic idea in 1D:
  - goal: $s(x) = \omega_0(\alpha)S_{i-1} + \omega_1(\alpha)S_i + \omega_2(\alpha)S_{i+1} + \omega_3(\alpha)S_{i+2}$
  - Observation: $a \cdot S_i + b \cdot S_{i+1} = (a+b) \cdot \mathrm{mix}\left(S_i, S_{i+1}, \frac{b}{a+b}\right)$

$$mix\left(S_i, S_{i+1}, \frac{b}{a+b}\right) = \left(1 - \frac{b}{a+b}\right)S_i + \frac{b}{a+b}S_{i+1} = \frac{a}{a+b}S_i + \frac{b}{a+b}S_{i+1}$$

  - With this: $s(x) = w_0 \cdot \mathrm{mix}(S_{i-1}, S_i, a_0) + w_1 \cdot \mathrm{mix}(S_{i+1}, S_{i+2}, a_1)$

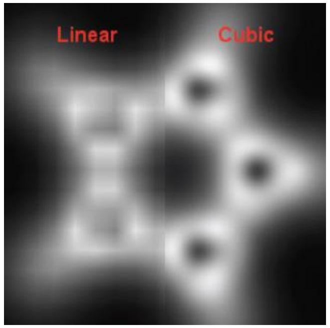
$$w_0 = \omega_0 + \omega_1;\ w_1 = \omega_2 + \omega_3;\ a_0 = \frac{\omega_1}{w_0};\ a_1 = \frac{\omega_3}{w_1};$$

Fast GPU-Evaluation of Cubic Interp.

GLSL code on right can be generalized:

- to 3D cubic interpolation by working with vec3 and 4 additional mix operations for z direction
- for other cubic versions by computing the ω_i with formula of other kernels

```
vec4 interpolate_bicubic(in sampler2D tex, vec2 pnt)
{
      // point location extracts index and fractional part
      vec2 coord_grid = pnt - vec2(0.5);
      vec2 index = floor(coord_grid);
      vec2 fraction = coord_grid - index;
      vec2 one_frac = 1.0 - fraction;
      vec2 one_frac2 = one_frac * one_frac;
      vec2 fraction2 = fraction * fraction;
      // compute b-spline weights
      vec2 omega0 = 1.0/6.0 * one_frac2 * one_frac;
      vec2 omega1 = 2.0/3.0 - 0.5 * fraction2 * (2.0-fraction);
      vec2 omega2 = 2.0/3.0 - 0.5 * one_frac2 * (2.0-one_frac);
      vec2 omega3 = 1.0/6.0 * fraction2 * fraction;
      // prepare fast interpolation
      vec2 w0 = omega0 + omega1;
      vec2 w1 = omega2 + omega3;
      vec2 a0 = (omega1 / w0) - 0.5 + index;
      vec2 a1 = (omega3 / w1) + 1.5 + index;
      // fetch the four bilinear interpolations
      float tex00 = texture(tex, vec2(a0.x, a0.y));
      float tex10 = texture(tex, vec2(a1.x, a0.y));
      float tex01 = texture(tex, vec2(a0.x, a1.y));
      float tex11 = texture(tex, vec2(a1.x, a1.y));
      // weigh along the y-direction
      tex00 = mix(tex01, tex00, w0.y);
      tex10 = mix(tex11, tex10, w0.y);
      // weigh along the x-direction
      return mix(tex10, tex00, w0.x);  }
```



Volume Preparation

TETRAHEDRAL MESHES

Tetrahedral Meshes

Minimalistic Definition

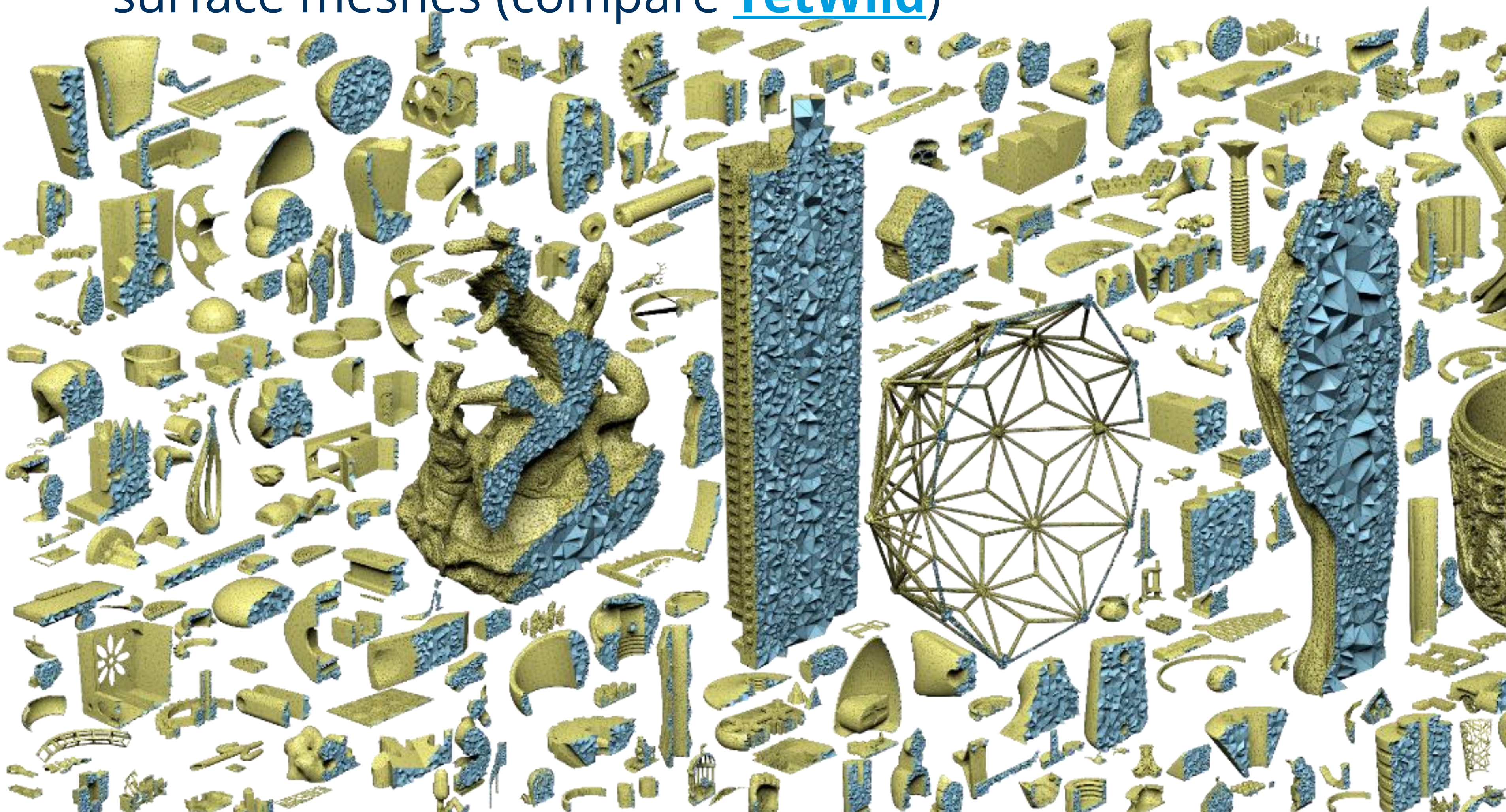
- a tetrahedral mesh $M = (V, T)$ is given by a set V of n_v vertices v_i and a set T of n_t tetrahedra or tets t_j
- each vertex v_i has a position $\underline{\boldsymbol{x}}_i \in \boldsymbol{R}^3$ and further attributes like scalar density S_i
- each tet $t_j = (i_{j,0}, i_{j,1}, i_{j,2}, i_{j,3})$ is an ordered quadrupel of vertex indices

Tet Mesh Generation

- Tetrahedral meshes can be generated from a set of points through a Delaunay Tetrahedralization that minimizes largest circum sphere. (compare qhull)

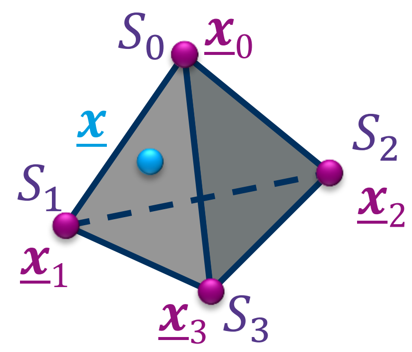
Tetrahedral Meshes

- Tet meshes are often generate for simulation from surface meshes (compare **TetWild**)



Tet Meshes – Barycentric Interpolation

- On individual tet with corner locations $\underline{\boldsymbol{x}}_0, \underline{\boldsymbol{x}}_1, \underline{\boldsymbol{x}}_2, \underline{\boldsymbol{x}}_3$ linear interpolation of an attribute f sampled at the corners S_0, S_1, S_2, S_3 can be defined with barycentric coordinates $\sigma_0, \sigma_1, \sigma_2, \sigma_3$ suming to 1

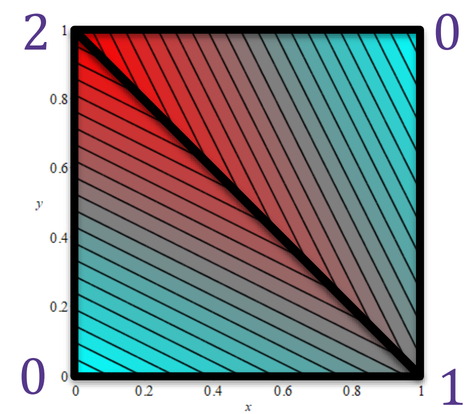


tetrahedron

- Location $\underline{\boldsymbol{x}}$ and its attribute value $S(\underline{\boldsymbol{x}})$ are mixed from corner locations and attributes with barycentric coordinates

$$\begin{pmatrix} \underline{\boldsymbol{x}} \\ 1 \end{pmatrix} = \sum_{i=0}^{3} \sigma_i \begin{pmatrix} \underline{\boldsymbol{x}}_i \\ 1 \end{pmatrix}, S(\underline{\boldsymbol{x}}) = \sum_{i=0}^{3} \sigma_i S_i$$

- barycentric interpolation is continuous on tet mesh but not differentiable over face adjacencies



TetMesh – Point Location 1

- **Input:** Target point
- **Output:** Tetrahedron that contains target point in case point falls inside of tetmesh

Algorithm for Point Localization

- Start with random tetrahedron
- repeat
  - Check for each tet face whether target point is on the outside
  - In case all checks fail, target tet is found
  - Otherwise move to tet adjacent to edge where point was outside first or in case of boundary triangle terminate and output boundary triangle

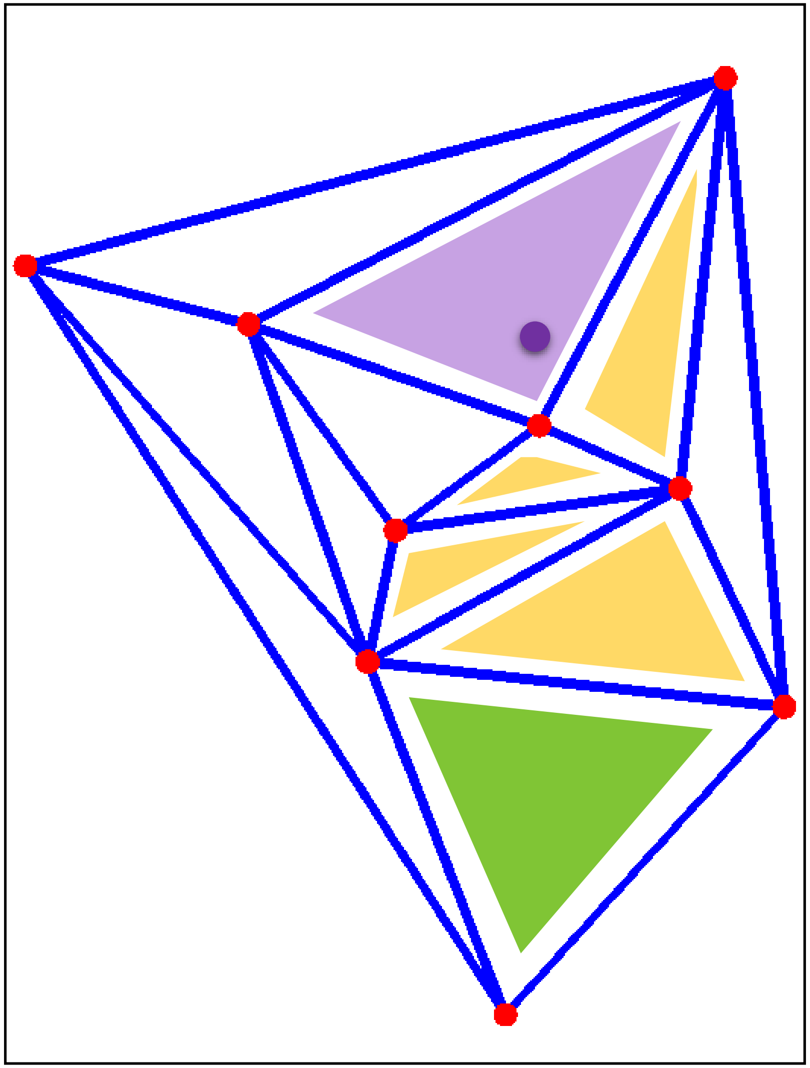
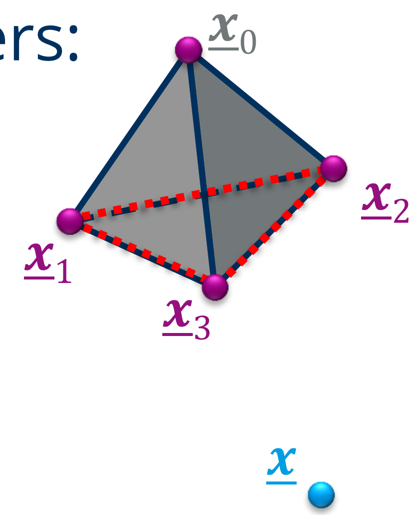


Illustration on triangle mesh instead of tetmesh

TetMesh – Point Location 2

- **Tet Volume** can be computed from corners:

$$V = \frac{1}{6} \cdot \det\begin{pmatrix} \underline{x}_0 & \underline{x}_1 & \underline{x}_2 & \underline{x}_3 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$



Tet Face Localization

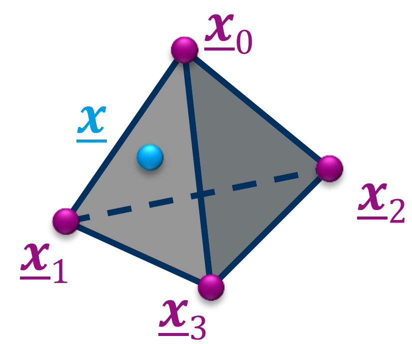
- only one geometric check necessary:
  - Target point is outside of tet face if it is on different side than fourth point of tet
  - This can be checked from sign switch of determinant of the point matrices extended by a homogeneous component:

$$\operatorname{sgn}\left[\det\begin{pmatrix} \underline{x}_0 & \underline{x}_1 & \underline{x}_2 & \underline{x}_3 \\ 1 & 1 & 1 & 1 \end{pmatrix}\right] = -\operatorname{sgn}\left[\det\begin{pmatrix} \underline{x} & \underline{x}_1 & \underline{x}_2 & \underline{x}_3 \\ 1 & 1 & 1 & 1 \end{pmatrix}\right]$$

$$\Rightarrow \underline{x} \text{ is outside}$$

TetMesh – Point Location 3

- To compute barycentric coordinates of $\underline{\boldsymbol{x}}$ with respect to the $\underline{\boldsymbol{x}}_i$ one introduces matrix-vector notation:



$$\begin{pmatrix} \underline{\boldsymbol{x}} \\ 1 \end{pmatrix} = \begin{pmatrix} \underline{\boldsymbol{x}}_0 & \underline{\boldsymbol{x}}_1 & \underline{\boldsymbol{x}}_2 & \underline{\boldsymbol{x}}_3 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \sigma_0 \\ \sigma_1 \\ \sigma_2 \\ \sigma_3 \end{pmatrix}$$

$$\Rightarrow \quad \widetilde{\boldsymbol{x}} = \widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{\sigma}}$$

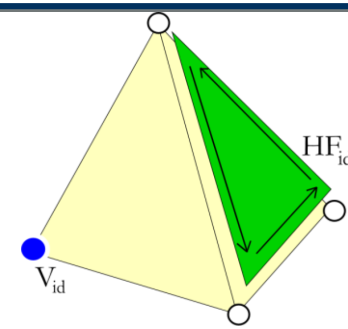
- This can be easily solved for barycentric coordinate vector:

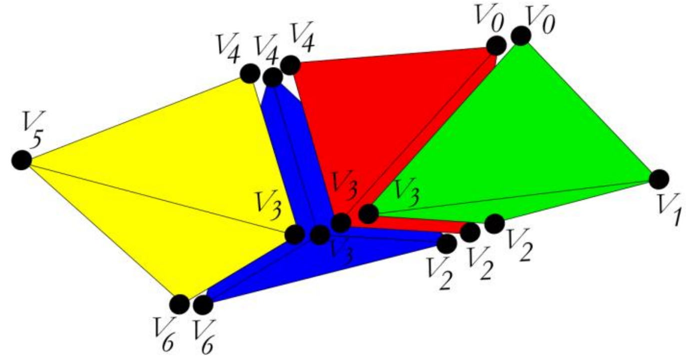
$$\widetilde{\boldsymbol{\sigma}} = \widetilde{\boldsymbol{X}}^{-1}\widetilde{\boldsymbol{x}}$$

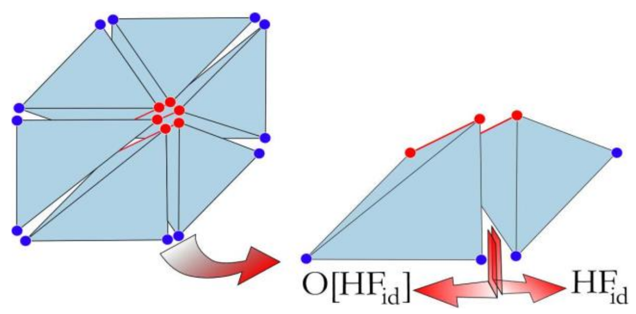
- Further acceleration of the point localization approach by use of a hierarchy

Tet Meshes – Half-Face Data Structure

- Vertex position and other attributes are stored in arrays
- The face of a tet is called half-face $\mathrm{HF_{id}}$ and is identified with opposite vertex $\mathrm{V_{id}}$
- The basic connectivity of a tet mesh is stored with 4 indices per tet – for each half-face the index of the opposite vertex
- To support access to neighbor tet for each half-face the incident half-face in the adjacent tet is stored and called opposite half-face.







Lage, M., Lewiner, T., Lopes, H., & Velho, L. (2005, October). CHF: a scalable topological data structure for tetrahedral meshes. In *XVIII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'05)* (pp. 349-356). IEEE.

Tet Meshes - Opposite Half-Face Matching

similar to inverse matching to build half-edge data structure (compare CG1) we can link opposite half-faces by sorting:

1. first sort vertex indices of each half-face internally
2. next sort half-faces externally according to their internally sorted vertex triple
3. finally, go through sorted list of half-faces and link half-faces with identical internally sorted vertex triple

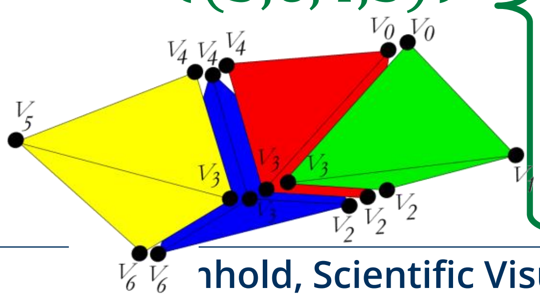
- Runtime:
  - internal sort $O(n_t)$
  - external sort $O(n_t \log n_t)$ or $O(n_v + n_t)$ with bucket sort
  - linking $O(n_t)$
  - In summary this can be implemented in $O(n_t)$

Tet Meshes - Opposite Half-Face Matching

Computer Graphics and Visualization

$T =$

4 tets	16 half-faces	internal	external	link
(0,1,2,3),	(0: 1,2,3),	(0: 1,2,3),	(3: 0,1,2),	
	(1: 0,3,2),	(1: 0,2,3),	(2: 0,1,3),	
	(2: 0,1,3),	(2: 0,1,3),	(1: 0,2,3),	$O[1] := 7$
	(3: 1,0,2),	(3: 0,1,2),	(7: 0,2,3),	$O[7] := 1$
(0,3,2,4),	(4: 3,2,4),	(4: 2,3,4),	(5: 0,2,4),	
	(5: 0,4,2),	(5: 0,2,4),	(6: 0,3,4),	
	(6: 0,3,4),	(6: 0,3,4),	(0: 1,2,3),	
	(7: 3,0,2),	(7: 0,2,3),	(4: 2,3,4),	$O[4] := B$
(3,2,4,6),	(8: 2,4,6),	(8: 2,4,6),	(B: 2,3,4),	$O[B] := 4$
	(9: 3,6,4),	(9: 3,4,6),	(A: 2,3,6),	
	(A: 3,2,6),	(A: 2,3,6),	(8: 2,4,6),	
	(B: 2,3,4),	(B: 2,3,4),	(D: 3,4,5),	
(3,6,4,5)	(C: 6,4,5),	(C: 4,5,6),	(9: 3,4,6),	$O[9] := F$
	(D: 3,5,4),	(D: 3,4,5),	(F: 3,4,6)	$O[F] := 9$
	(E: 3,6,5),	(E: 3,5,6),	(E: 3,5,6),	
	(F: 6,3,4)	(F: 3,4,6)	(C: 4,5,6),	

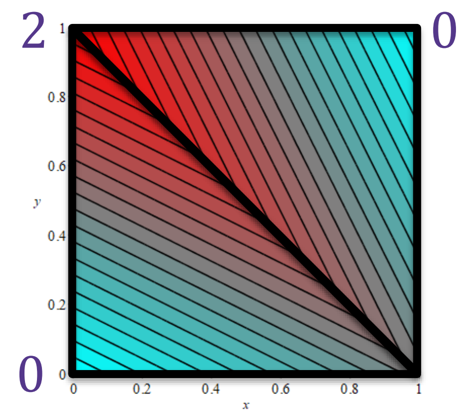


TetMesh – Gradient Computation 1

- We extend matrix-vector notation to attribute values

$$S(\underline{x}) = \sum_{i=0}^{3} \sigma_i S_i = \begin{pmatrix} \sigma_0 & \sigma_1 & \sigma_2 & \sigma_3 \end{pmatrix} \begin{pmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{pmatrix}$$

$$\Rightarrow S(\underline{x}) = \langle \tilde{\boldsymbol{\sigma}}, \tilde{\boldsymbol{S}} \rangle = \langle \tilde{\boldsymbol{S}}, \tilde{\boldsymbol{\sigma}} \rangle = \tilde{\boldsymbol{S}}^T \tilde{\boldsymbol{\sigma}}$$



- and plug in $\tilde{\boldsymbol{\sigma}} = \tilde{\boldsymbol{X}}^{-1} \tilde{\boldsymbol{x}}$ resulting in

$$S(\underline{x}) = \langle \tilde{\boldsymbol{S}}, \tilde{\boldsymbol{\sigma}} \rangle = \tilde{\boldsymbol{S}}^T \tilde{\boldsymbol{X}}^{-1} \tilde{\boldsymbol{x}}$$

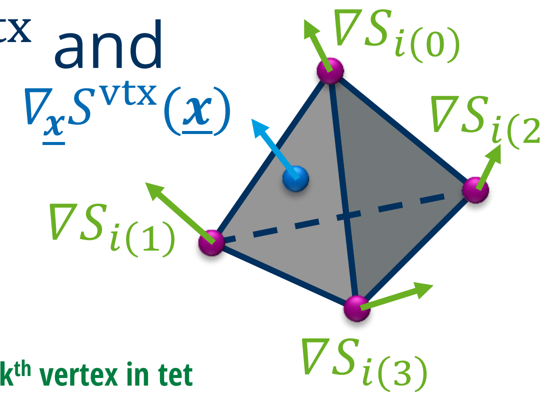
- Applying the gradient operator yields

$$\nabla_{\underline{x}} S^{\text{tet}}(\underline{x}) = \tilde{\boldsymbol{S}}^T \tilde{\boldsymbol{X}}^{-1} \left(\nabla_{\underline{x}} \tilde{\boldsymbol{x}} \right) = \left(\tilde{\boldsymbol{S}}^T \tilde{\boldsymbol{X}}^{-1} \right) \Big|_{xyz} = \text{const}$$

- which is constant over a tetrahedron

TetMesh – Gradient Computation 2

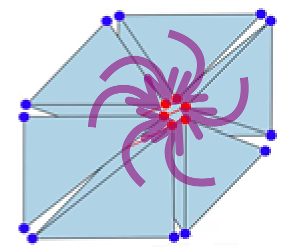
- To provide continuous gradients over the tetmesh one can estimate per vertex v_i gradients ∇S_i^{vtx} and baryzentrically interpolate them



$$\nabla_{\underline{x}} S^{\text{vtx}}(\underline{x}) = \sum_{k=0}^{3} \sigma_k \nabla S_{i(k)}^{\text{vtx}}$$

vertex index i of k^th vertex in tet

- Given a vertex i with incident tets $j \in N_i$ of volume V_j and constant gradients ∇S_j^{tet} the vertex gradient ∇S_i^{vtx} can be estimated to



$$\nabla S_i^{\text{vtx}} = \frac{1}{\sum_{j \in N_i} V_j} \sum_{j \in N_i} V_j \nabla S_j^{\text{tet}}$$

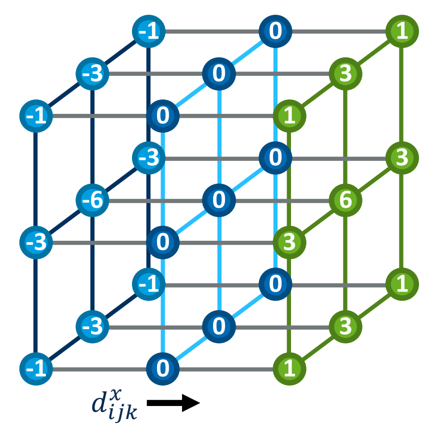
- With tet volume $V_j = \frac{1}{6} \cdot \det \begin{pmatrix} \underline{x}_{i_{j,0}} & \underline{x}_{i_{j,1}} & \underline{x}_{i_{j,2}} & \underline{x}_{i_{j,3}} \\ 1 & 1 & 1 & 1 \end{pmatrix}$

Regular Grid Gradient

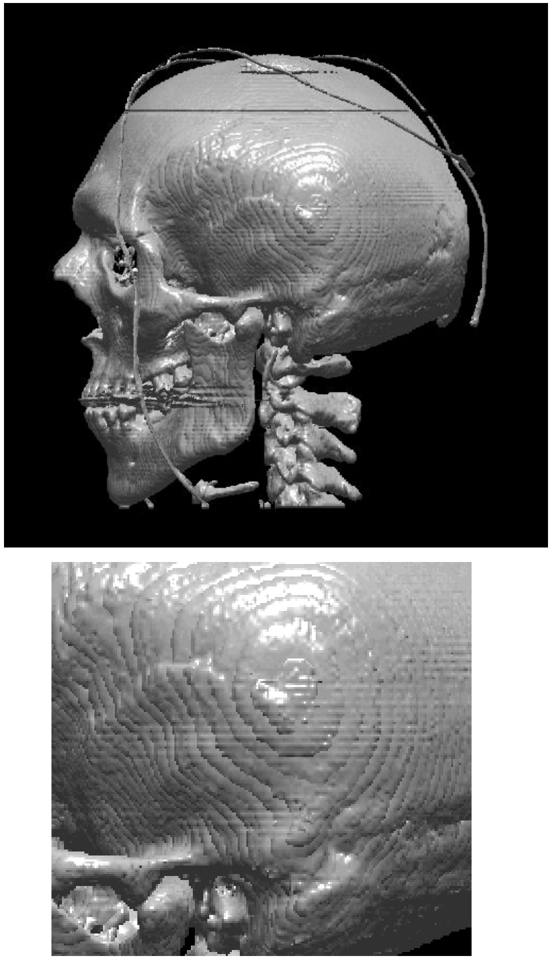
- Similarly one can precompute the gradient on a regular grid and interpolate it during rendering
- Finite differences are typically not sufficient and yield staircase artefacts in the illumination
- The discussed cubic interpolation filters can be used for gradient estimation, centered on grid vertex and evaluated on a 3x3x3 neighborhood. For proper scaling check on test function with known gradient
- As an alternative, one can use Sobel Operator normalized with $\frac{1}{44}$:

$$\frac{\partial S}{\partial x}(x_0, y_0, z_0) = \frac{1}{44} \sum_{i,j,k=-1}^{1} d_{ijk}^{x} \cdot S(x_i, y_j, z_k)$$

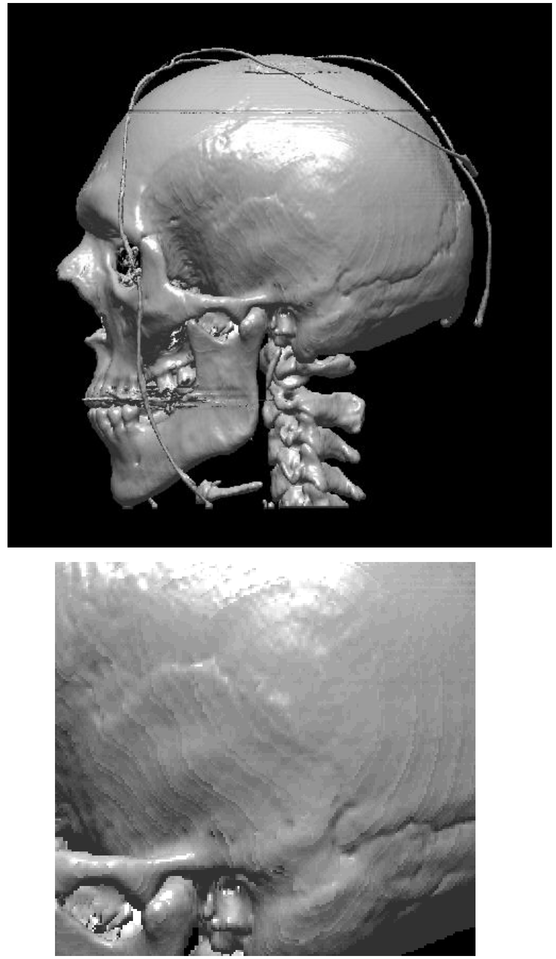
with rotated masks d_{ijk}^{y} and d_{ijk}^{z} for the other partial derivatives



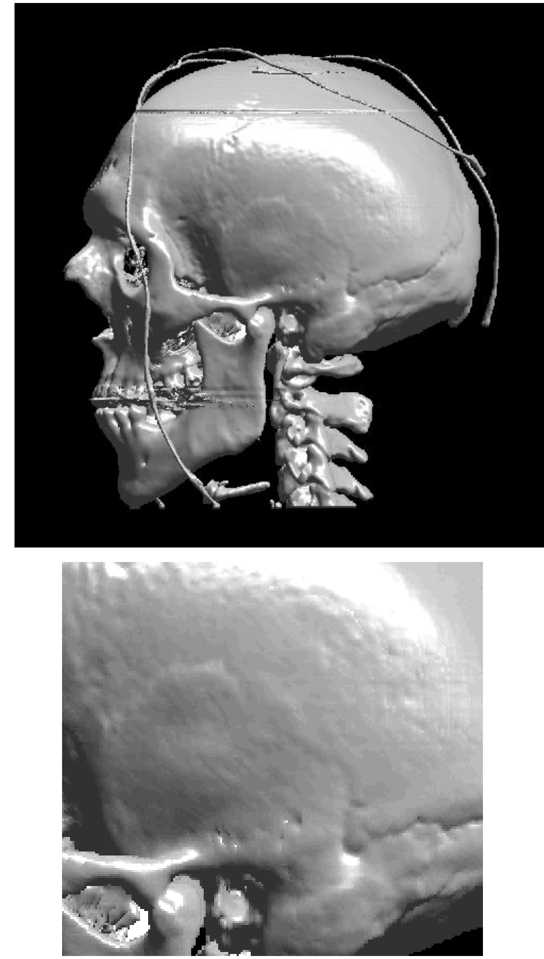
Regular Grid Gradient – Comparison



Forward differences



Central differences



Sobel Operator

Volume Preparation

FILTERING

Filtering

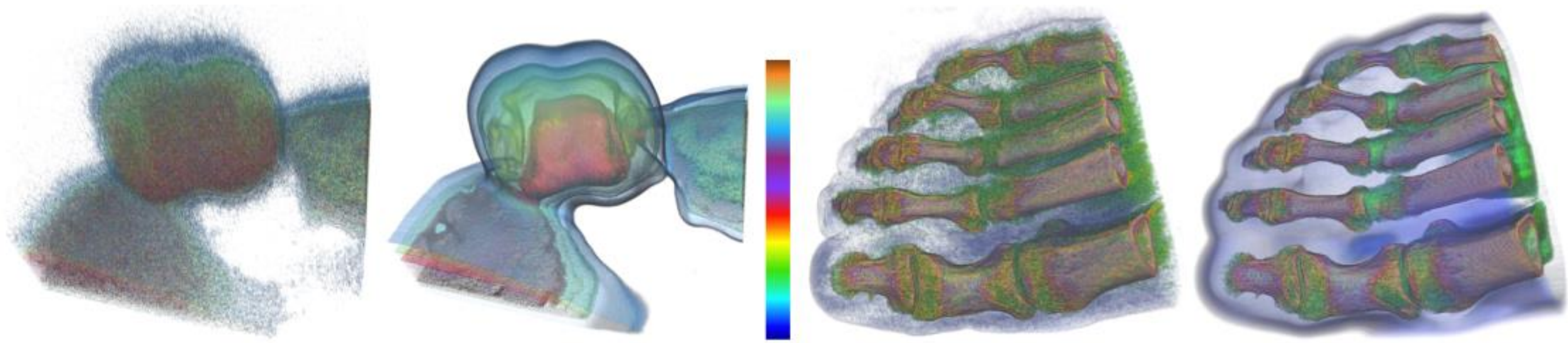


Figure 8: *Examples of volume denoising with our FGT-based fast bilateral filter: two iterations with* $\sigma^{g_2} = s_d$ *and* $(\varepsilon, r) = (10^2, 2)$. *Left-most: original noisy cell-cytokinesis volumetric dataset of size* $256 \times 256 \times 60$ *voxels obtained using a confocal laser microscope. Middle-left: it takes only 9.3 s for our FGT-based bilateral denoising with* $\sigma^{g_1} = (1.6, 1.6, 5)$. *Middle-right: noisy CT-foot volume with* 256^3 *voxels. Right-most: it takes 450 s for our FGT-based bilateral denoising with* $\sigma^{g_1} = (8, 8, 8)$.

- Yoshizawa, S., Belyaev, A., & Yokota, H. (2010, March). Fast gauss bilateral filtering. In *Computer Graphics Forum* (Vol. 29, No. 1, pp. 60-74). Oxford, UK: Blackwell Publishing Ltd.

Filtering

- Imaging noise can be removed by convolving volume with filter kernel, e.g. Gaussian $c \cdot e^{-d^2/\sigma^2}$ depending on distance d to sample
- with separable filters $h^{\otimes}(x,y,z) = h(x)h(y)h(z)$ the complexity of convolving a N^3 volume with a filter with M^3 support can be reduced from $N^3 \cdot M^3$ to $3N^3M$ by applying the linear filters in each dimension one after the other
- Bilateral filter multiplies secondary kernel that depends on distance r (range) in scalar value $c \cdot e^{-d^2/\sigma_d^2} \cdot e^{-r^2/\sigma_r^2}$ and supports preservation of edges which are important in volume rendering (see intro at https://people.csail.mit.edu/sparis/bf_course/slides/03_definition_bf.pdf)
- To exploit separation property also for bilateral filtering one can use fast implementation with permutohedral (tet) lattice: https://graphics.stanford.edu/papers/permutohedral/

Volume

REFERENCES

References

Computer Graphics and Visualization

- Keys, R. (1981). Cubic convolution interpolation for digital image processing. *IEEE transactions on acoustics, speech, and signal processing*, *29*(6), 1153-1160.
- Mitchell, D. P., & Netravali, A. N. (1988, August). Reconstruction filters in computer-graphics. In *ACM Siggraph Computer Graphics* (Vol. 22, No. 4, pp. 221-228). ACM.
- Moller, T., Machiraju, R., Mueller, K., & Yagel, R. (1996, October). Classification and local error estimation of interpolation and derivative filters for volume rendering. In *proceedings of 1996 Symposium on Volume Visualization* (pp. 71-78). IEEE.
- Ruijters, D., ter Haar Romeny, B. M., & Suetens, P. (2008). Efficient GPU-based texture interpolation using uniform B-splines. *Journal of Graphics Tools*, *13*(4), 61-69.
- Adams, A., Baek, J., & Davis, M. A. (2010, May). Fast high-dimensional filtering using the permutohedral lattice. In *Computer Graphics Forum* (Vol. 29, No. 2, pp. 753-762). Oxford, UK: Blackwell Publishing Ltd.
- Yoshizawa, S., Belyaev, A., & Yokota, H. (2010, March). Fast gauss bilateral filtering. In *Computer Graphics Forum* (Vol. 29, No. 1, pp. 60-74). Oxford, UK: Blackwell Publishing Ltd.