UNIVERSITÄT SIEGEN

Scientific Visualization

Foundations and Applications

Siegener Graduate School

Development of Integral Heterosensor Architectures for n-Dimensional (Bio)chemical Analysis

Prof. Dr. Andreas Kolb

IVG

Institute for
Vision and Graphics (IVG)

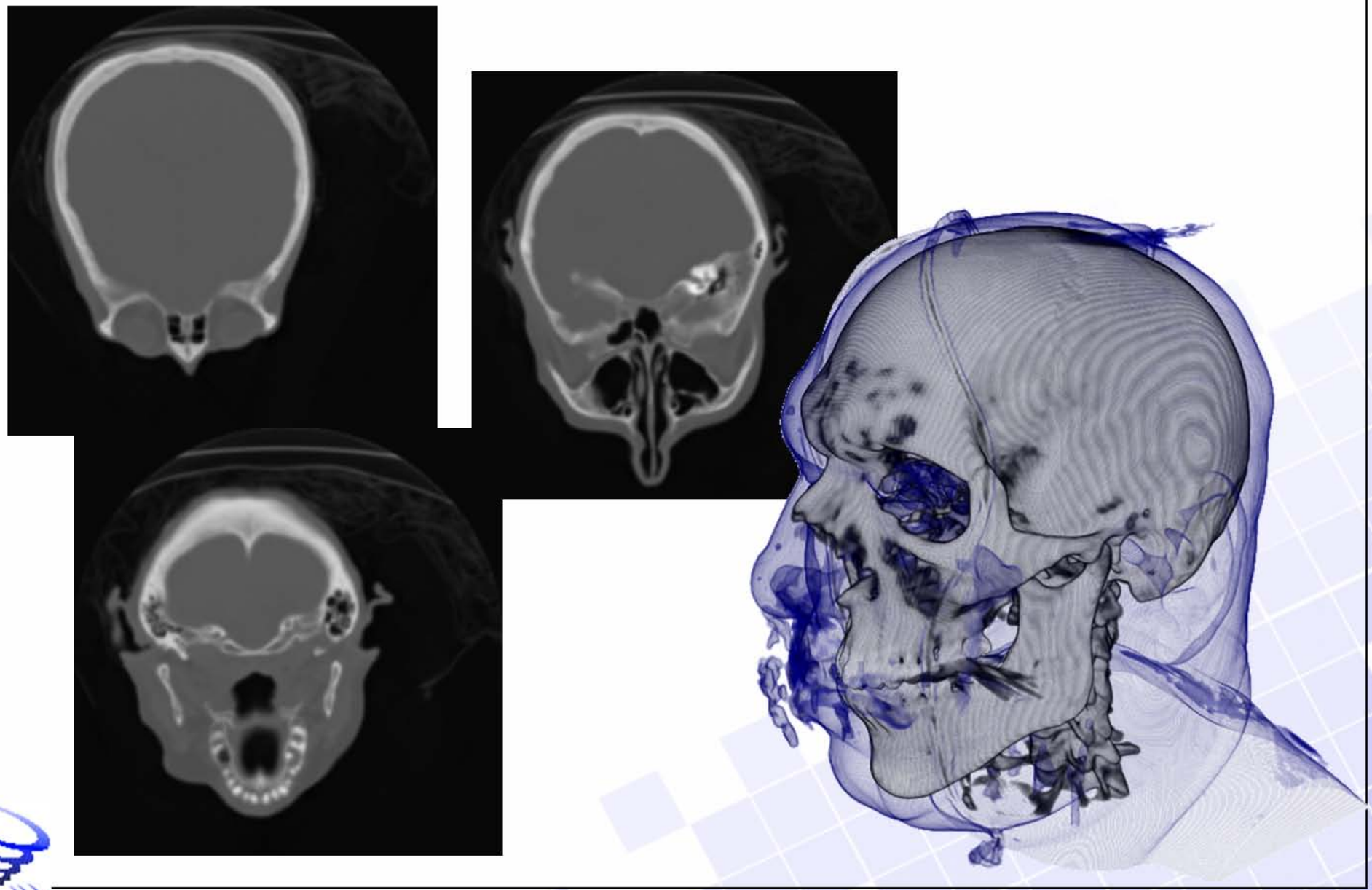
CG

Computer Graphics and
Multimedia Systems Group

What is Visualization about?

- Visualizing =
  making something visible; find visual expression
- Why should data/information be visualized?
  - numbers are useless without a context

Example



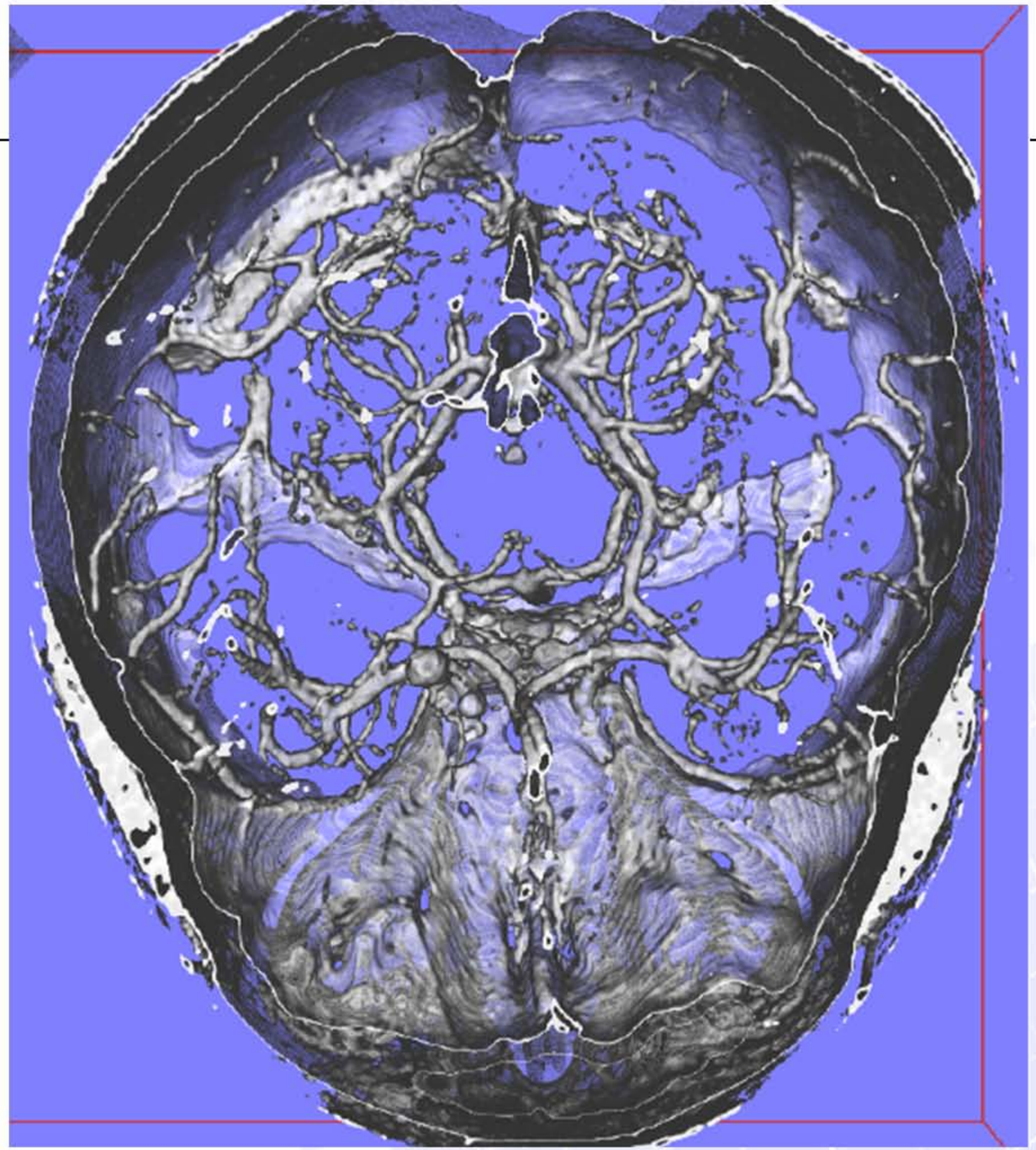
What is Visualization about?

- Visualizing =
  making something visible; find visual expression
- Why should data/information be visualized?
  - numbers are useless without a context
  - data is useless without interpretation
  - information is useless without understanding

„One picture is worth ten thousand words“

Fred R. Barnard

Beispiel



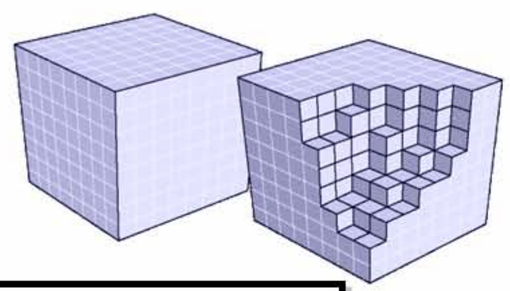
Overview

- Introduction
- Grids, Sampling and Interpolation
- 2D Scalar-Fields
- 2D Vector-Fields
- Volume Rendering (3D Scalar-Fields)
- 3D Vector-Fields

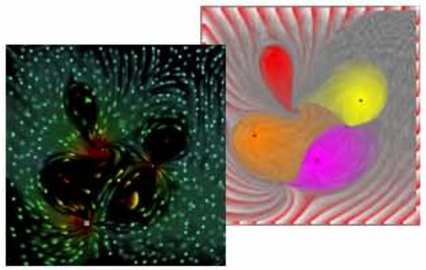
Challanges and Benefits

- **Challenges**
  - Hugh amount of data from sensors or simulation
  - High data complexity
  - Need for interactivity and real-time
- **Benefits**
  - Faster and better understanding of scientific processes
  - Thus: more security, efficiency and quality in engineering and production processes
  - Thus: shorter production cycles and products with more functionality and higher quality

Visualization Pipeline



Raw Data

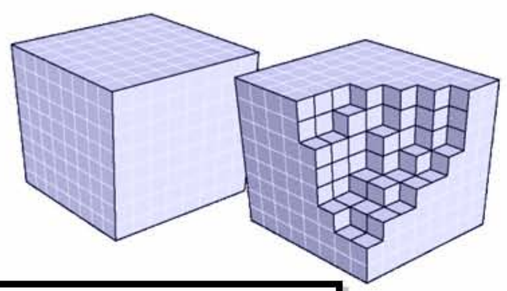


Image

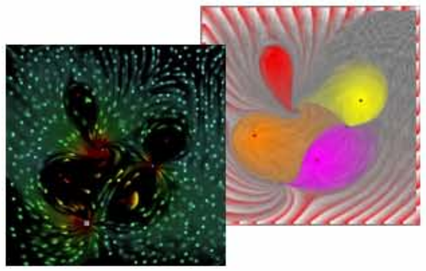
Filtering

- Selection of Data
- Pre-Processing,
  e.g. outlier removal

Visualization Pipeline



Raw Data



Image

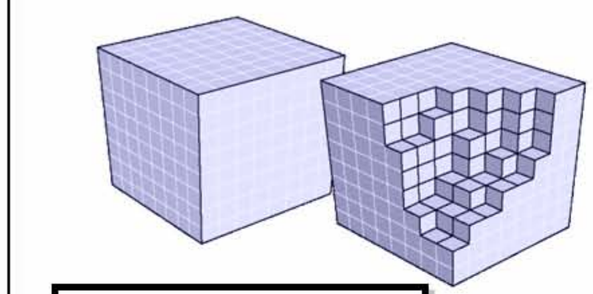
Filtering

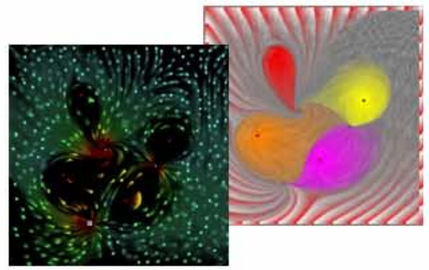
Mapping

Map data to

- **Geometry** and
- **visual attributes**, e.g. color, transparancy

Visualization Pipeline





Raw Data

Image

Filtering

Mapping

Rendering

Image Synthesis

- Generate image from geometry and visual attributes

Raw Data

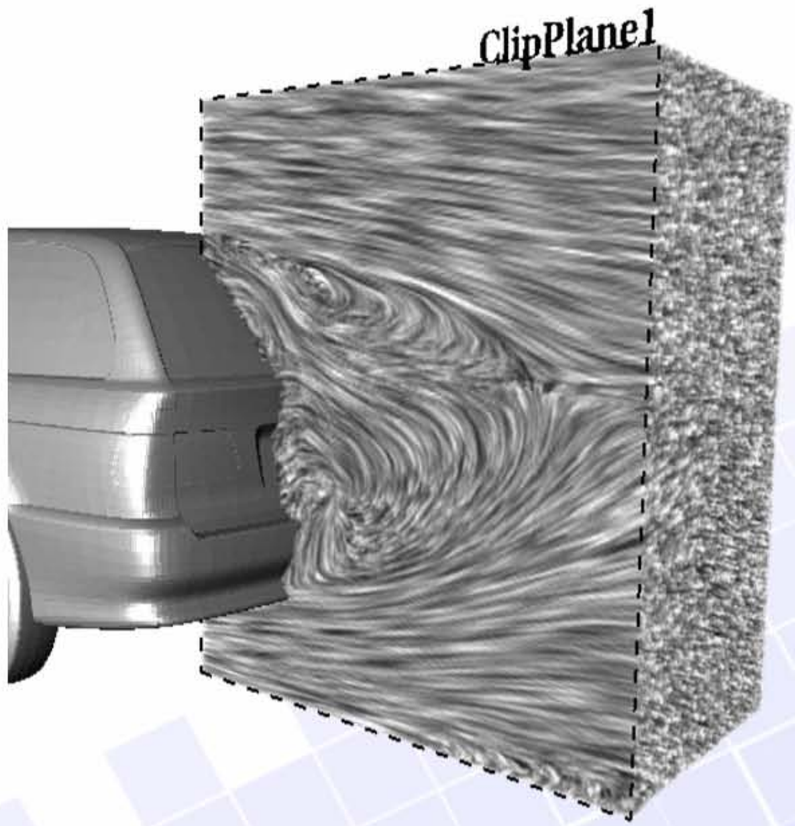
Categorization of raw data using:

- **Spacial Dimension**
  (with respect to the structure of the sample/data points)
  - **1D:** e.g. electro-magnetic signal
  - **2D:** e.g. temperature on a surface
  - **3D:** e.g. flow velocity in space
- **Dimensionality of the Data Values**
  (related to the information at the data points)
  - **Scalar:** e.g. pressure, density, temperature
  - **Vector:** e.g. flow
  - **Tensor:** e.g. stress-tensor
  - **Multivariate Data:** combination of several scalar, vector and tensor data

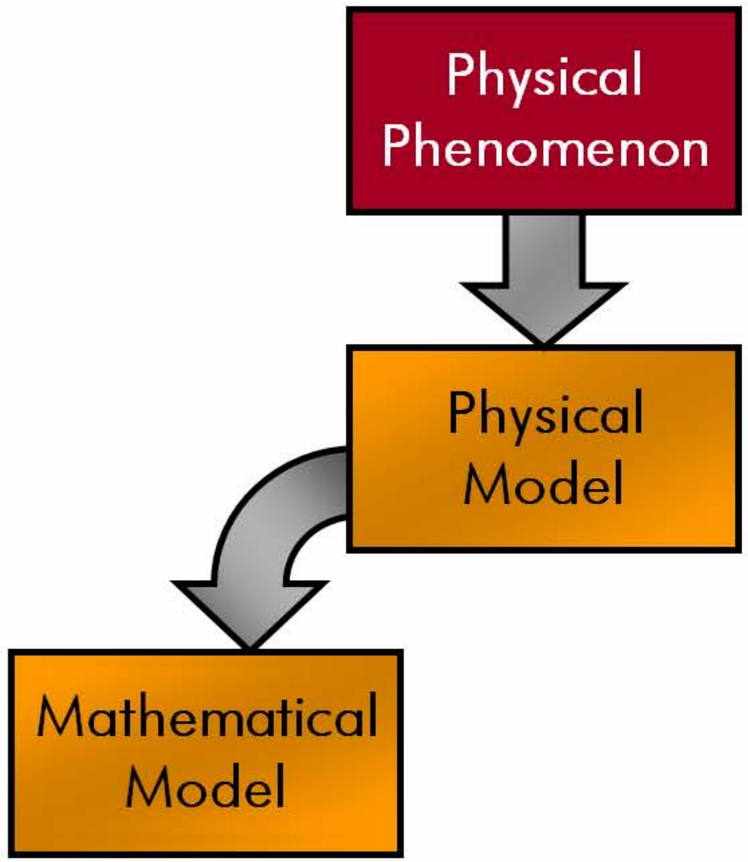
Simulation as Data Source

Physical Phenomenon

e.g. flow at a car body



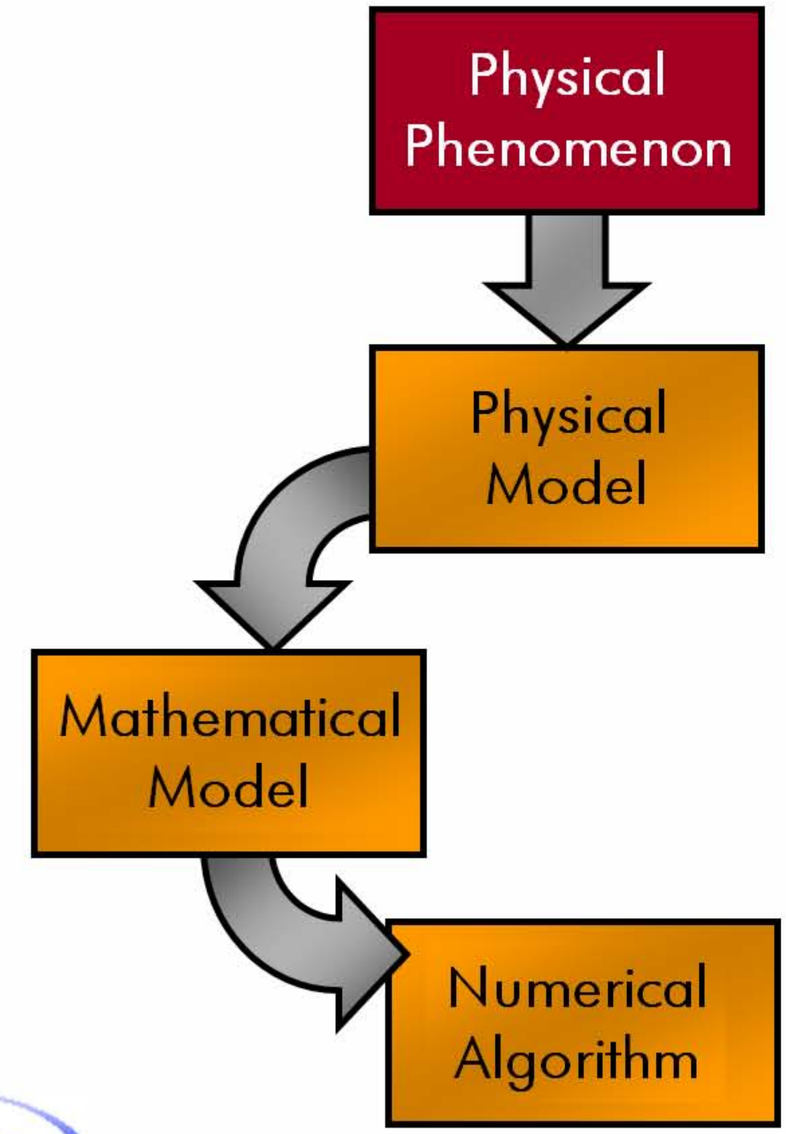
Simulation as Data Source



e.g. flow at a car body

e.g. Navier-Stokes equations

Simulation as Data Source

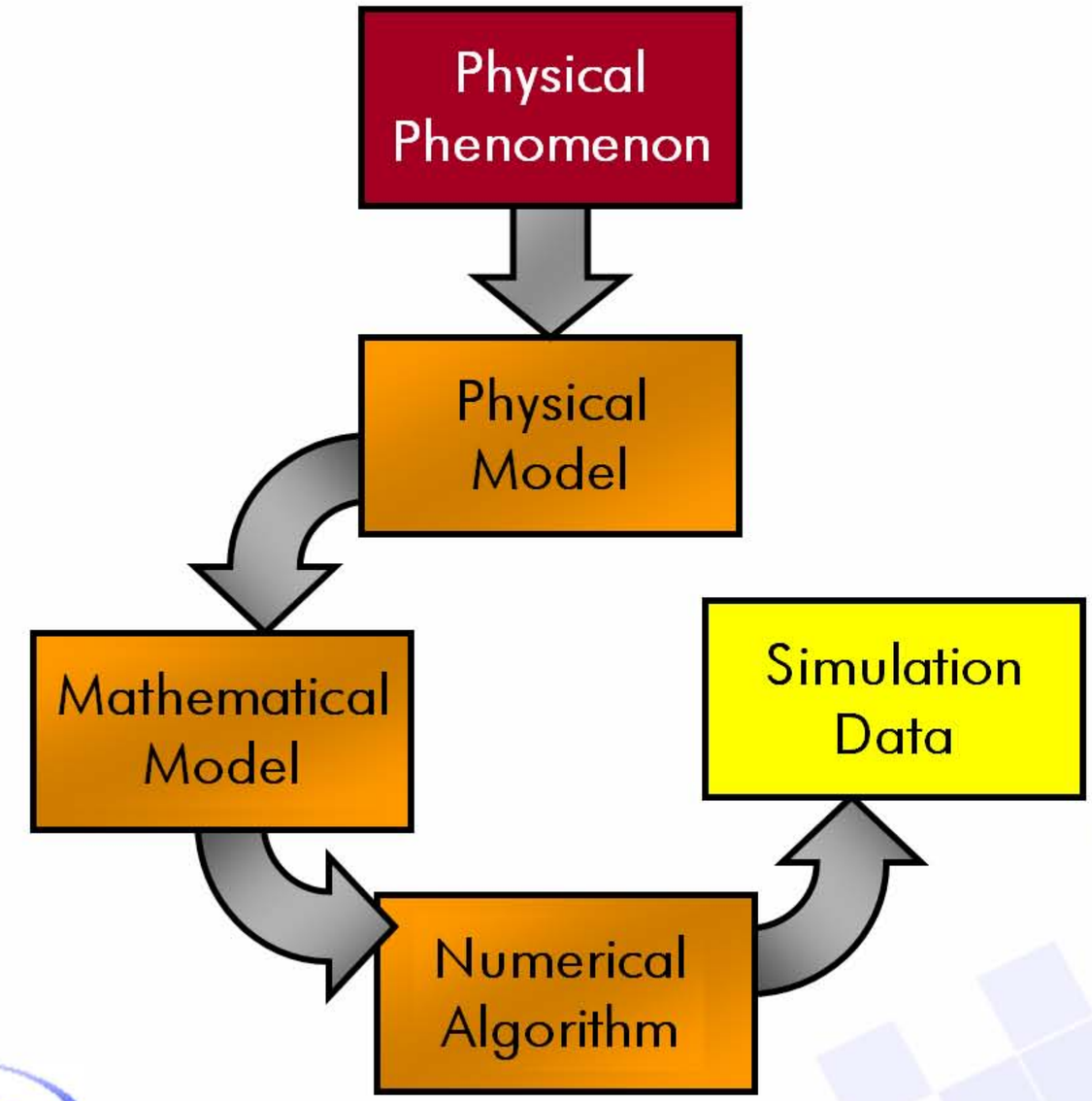


e.g. flow at a car body

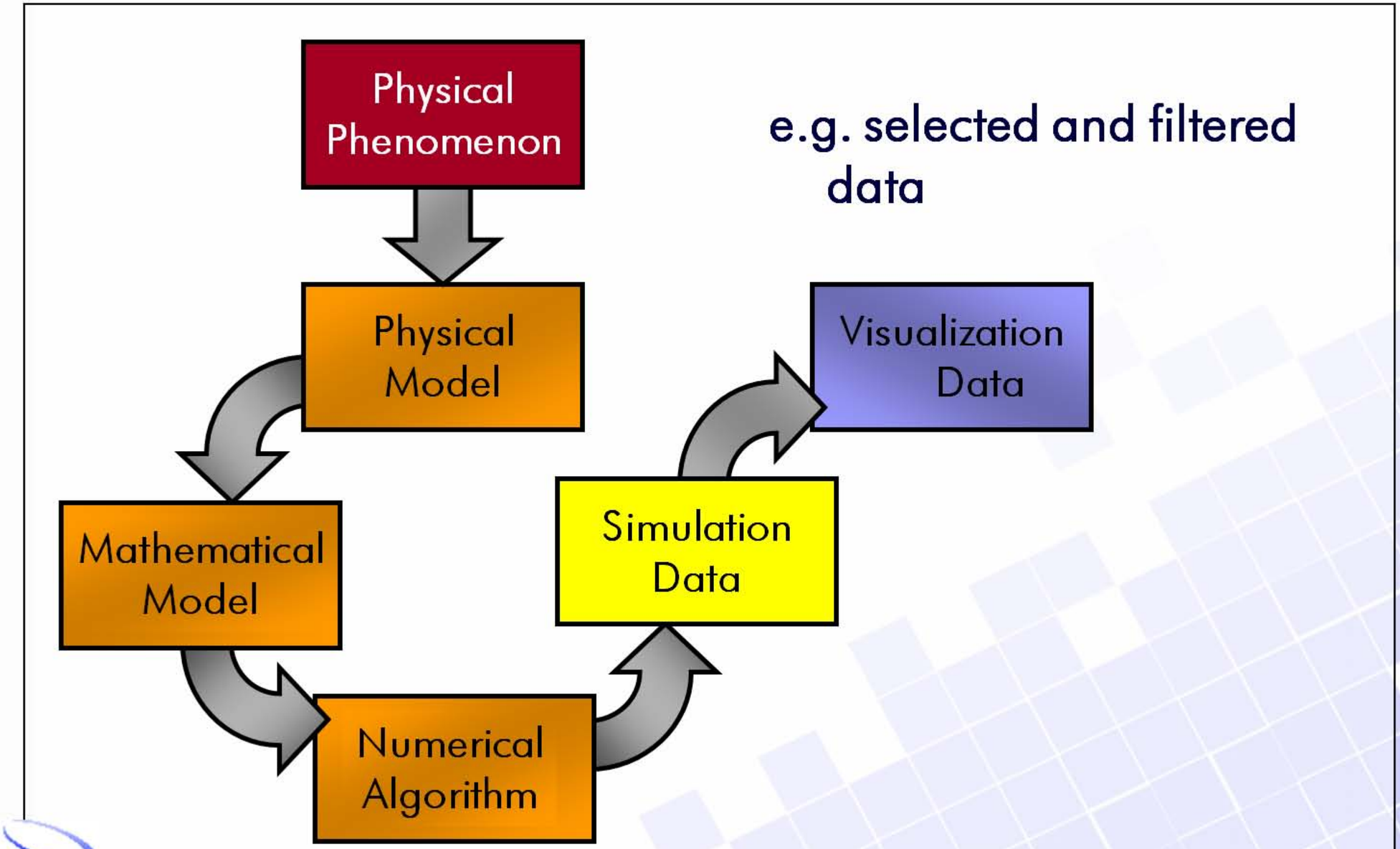
e.g. Navier-Stokes equations

e.g. finite element simulation

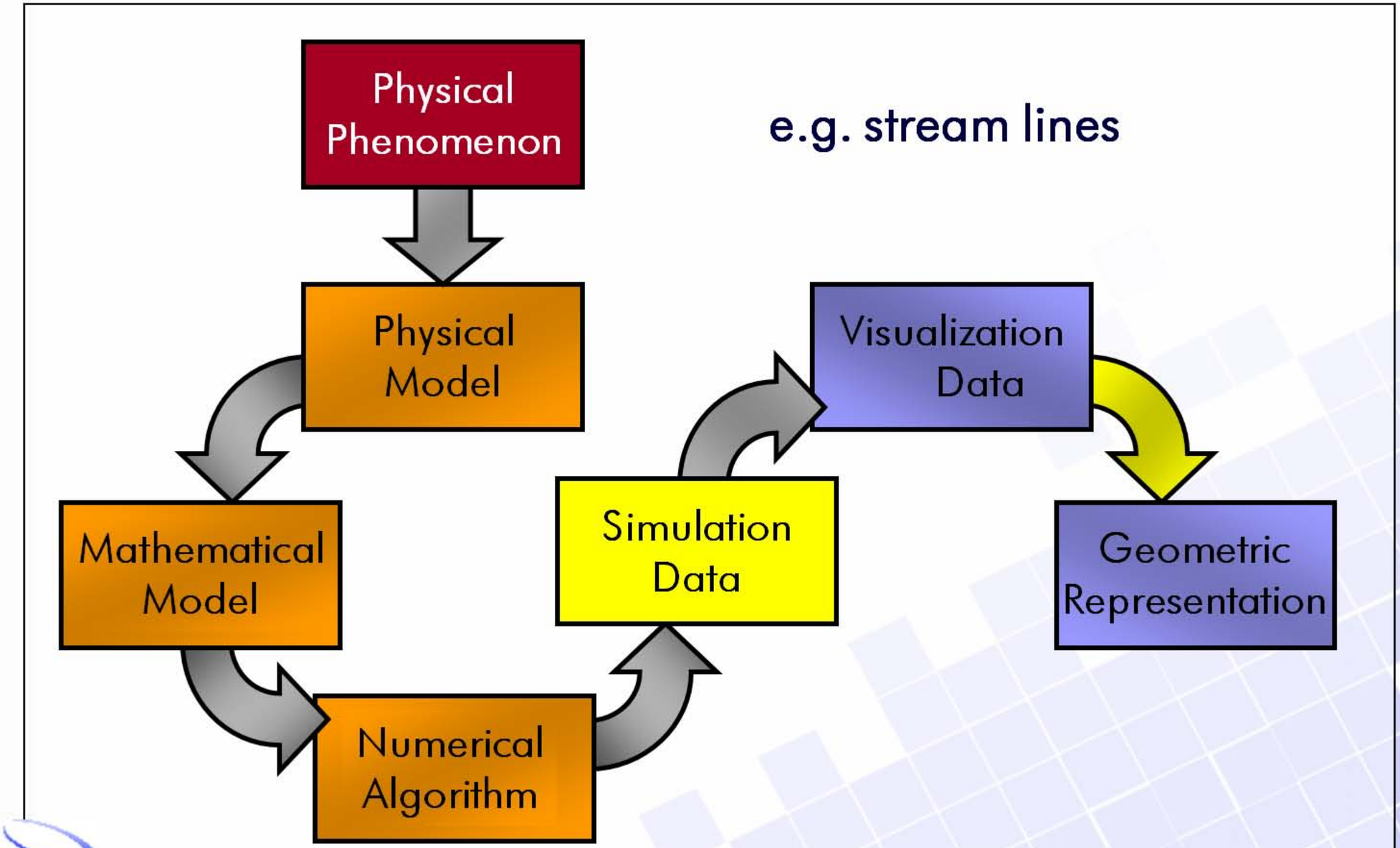
Simulation as Data Source



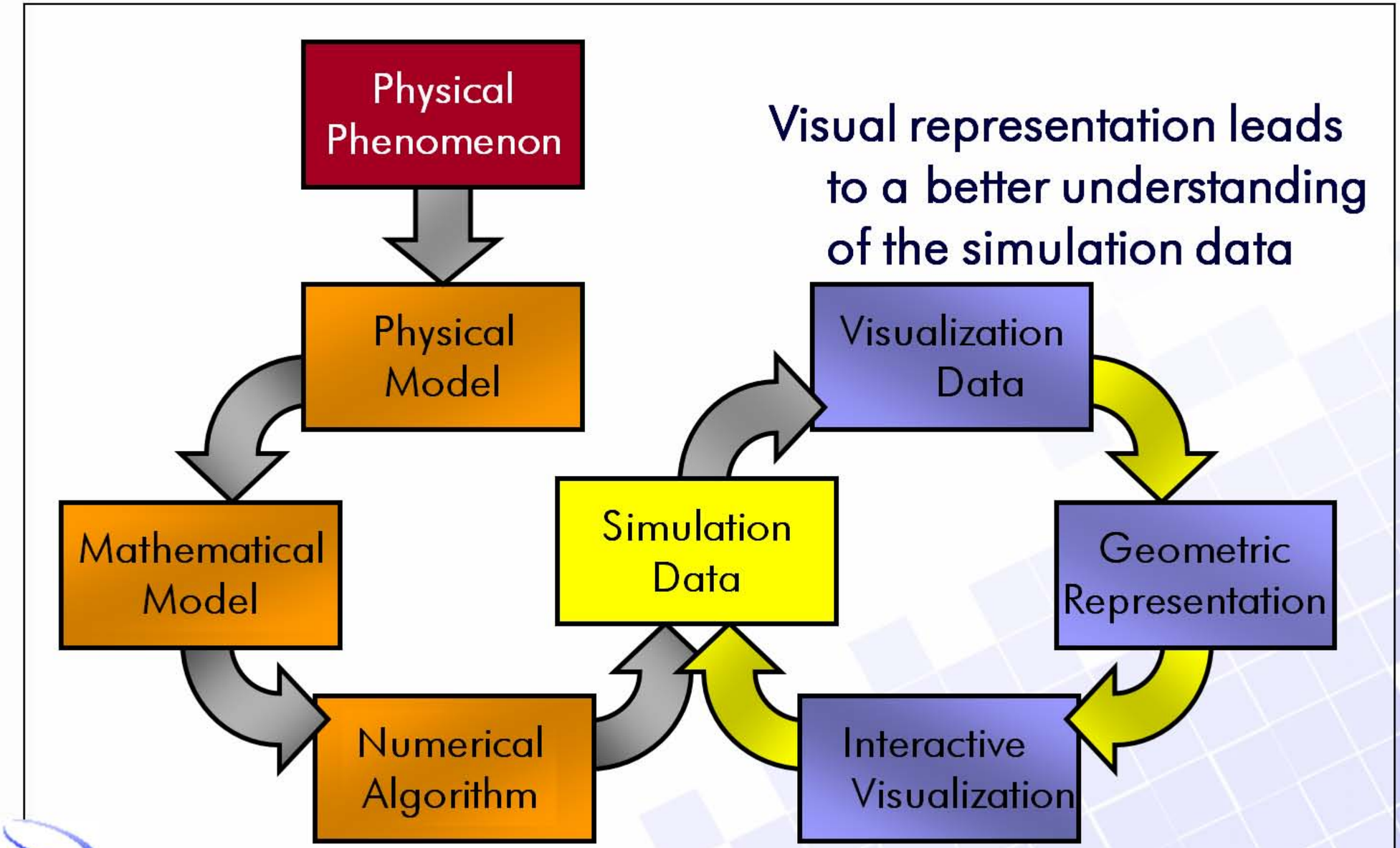
Simulation as Data Source



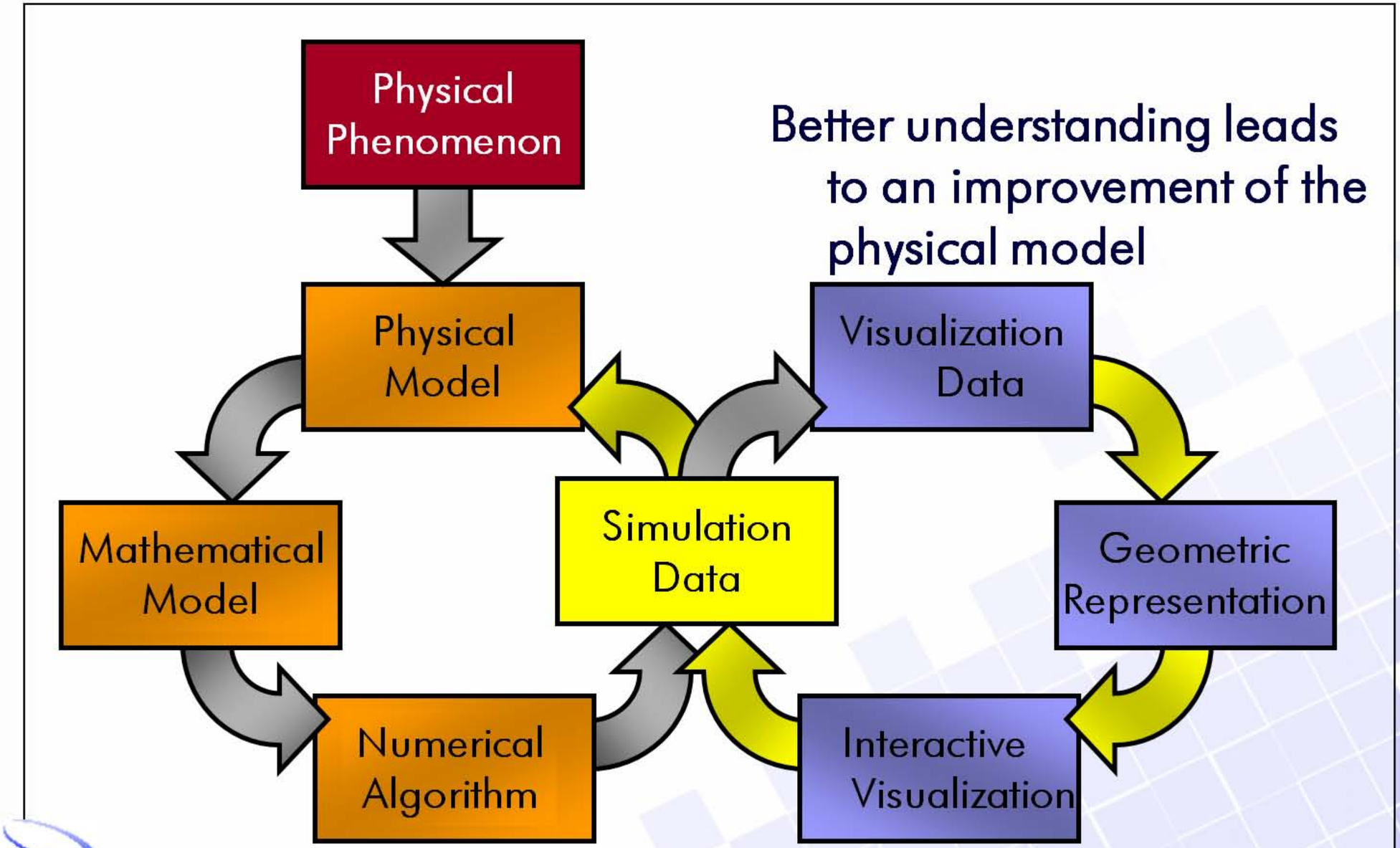
Simulation as Data Source



Simulation as Data Source



Simulation as Data Source

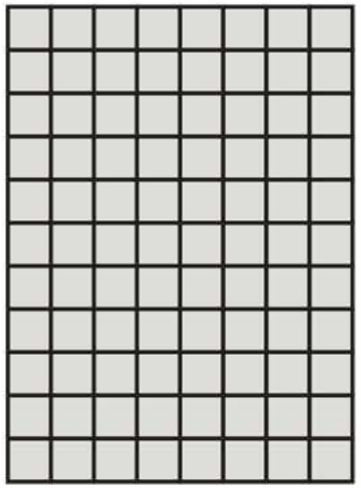


Grids

- What are grids good for?
  - Natural phenomena are in general continuous, i.e. data values are “present” everywhere
  - Computer can handle only a finite amount of data
- Therefore: continuous data needs to be sampled (digitized, rasterized) in order to be processed in a computer

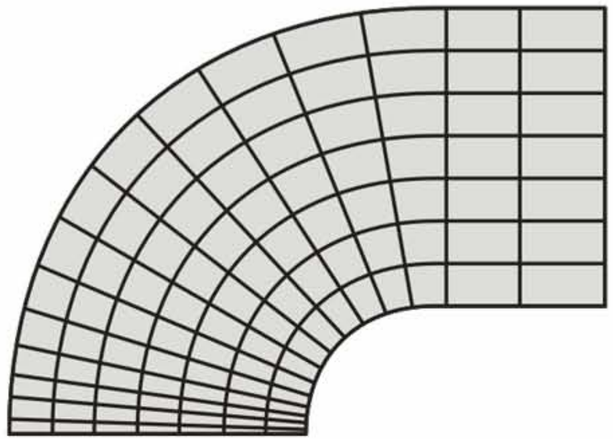
2D Grid Types

uniform rectilinear



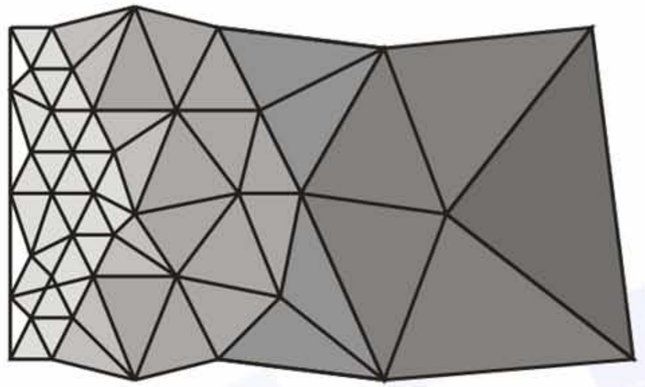
- *Cells:* equal sized rectangles or squres
- *Arrangement:* uniform regular

curvilinear



- *Cells:* quadrangles with varying shape
- *Arrangement :* non-uniform, but regular topology

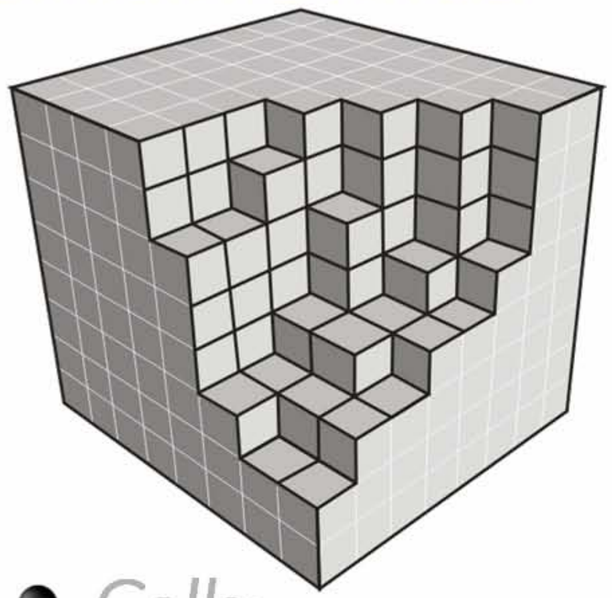
unstructured



- *Cells:* mainly triangles, *(sometimes quads or polys)*
- *Arrangement :* unstructured

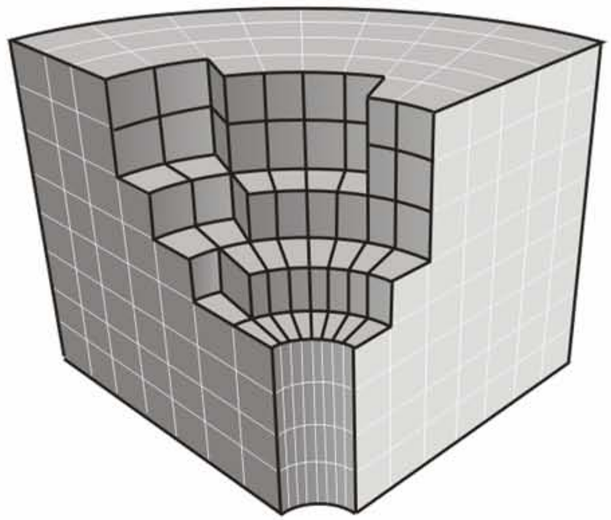
3D Grid Types

uniform rectilinear



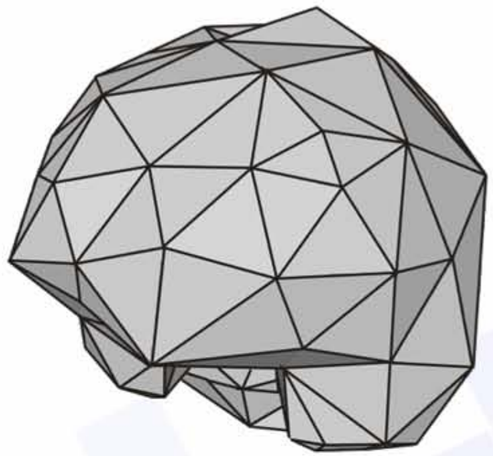
- *Cells:* equal-sized cubes,
- *Arrangement:* uniform regular

curvilinear



- *Cells:* non-equal hexahedras,
- *Arrangement:* non-uniform, but regular topolgy

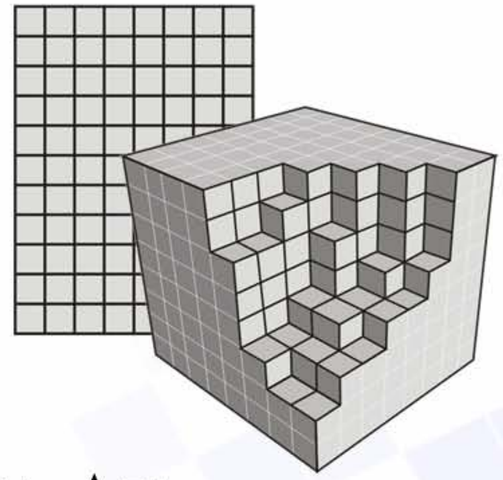
unstrukturiert



- *Cells:* tetrahedra, hexaheder, pyramids, prismas
- *Arrangement:* unstructured

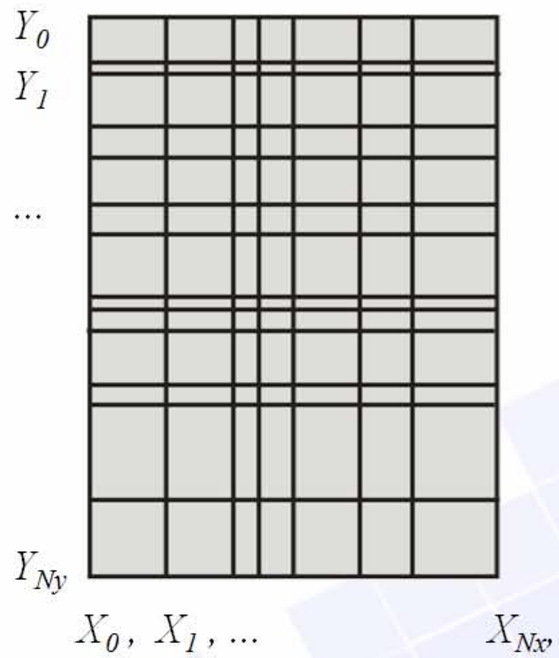
Representation of Uniform Grids

- Implicit storage of cells and vertices:
  - Store number of samples N_x, N_y, N_z in x-, y- and z-direction
  - Store cell size, i.e. grid spacing Δx, Δy, Δz: Distance between samples in x-, y- and z-direction
  - Store data an linear Array with length $N_x \cdot N_y \cdot N_z$
  - Array-index of cells (x,y,z) can be computed by $n = N_x \cdot N_y \cdot z + N_x \cdot y + x$



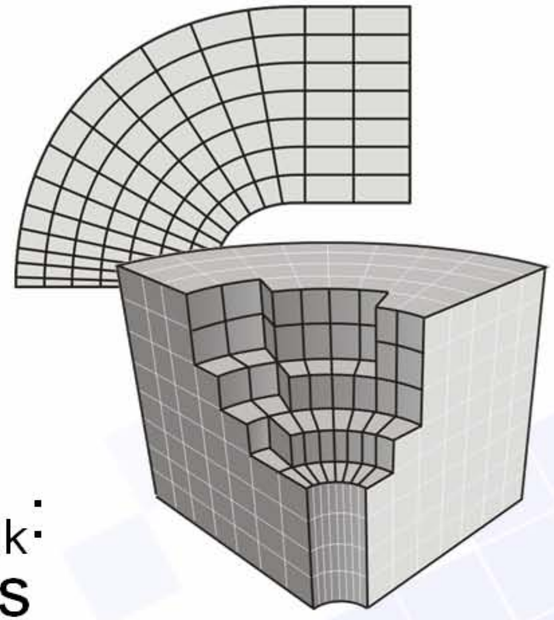
Representation of Rectilinear Grids

- Implicit storage of cells, explicit storage of vertices:
  - Store number of samples N_x, N_y, N_z in x-, y- and z-direction
  - Store coordinates along axis:
    - X_0 , X_1 , ... X_{Nx},
    - Y_0 , Y_1 , ... Y_{Ny},
    - Z_0 , Z_1 , ... Z_{Nz},
- Store data an linear Array with length $N_x \cdot N_y \cdot N_z$
- Array-index of cells (x,y,z) can be computed by $n = N_x \cdot N_y \cdot z + N_x \cdot y + x$



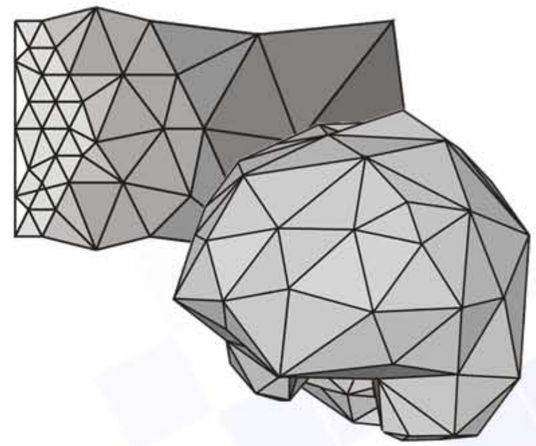
Representation of Curvilinear Grids

- Implicit storage of cells, explicit storage of vertices:
  - Store number of samples N_x, N_y, N_z in x-, y- and z-direction
  - Store list of all vertex coordinates P_{ijk}: x-, y- and z-coordinates of grid points
- Store data an linear Array with length $N_x \cdot N_y \cdot N_z$
- Array-index of cells (x,y,z) can be computed by $n = N_x \cdot N_y \cdot z + N_x \cdot y + x$



Represent. of Unstructured Grids

- Explicit storage of cells and vertices:
  - Store number of grid points N
  - Store number of cells Z
  - Store list of grid points P_{ijk}: x-, y- and z-coordinates
- Store cells by referencing the indices to the grid points: e.g. tetrahedron: 4 indices



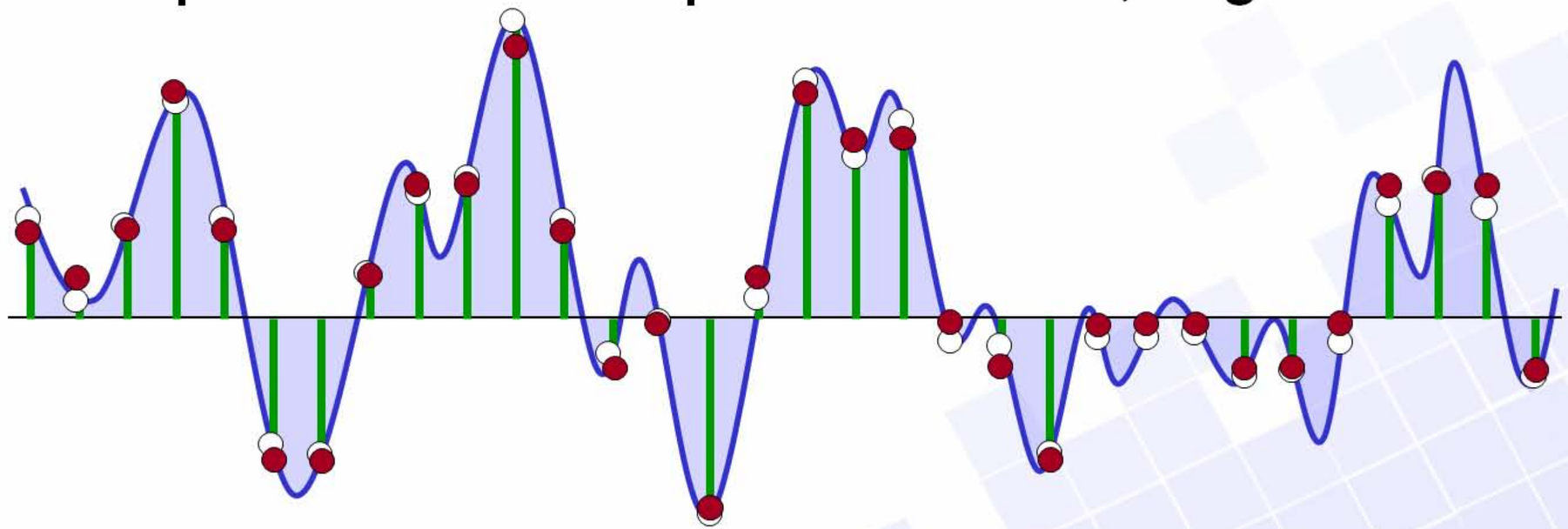
Comparison

- **Uniform rectilinear:**
  Pro: simple and efficient storage and data access, cell-search is trivial
  Con: constant resolution, no local adoption possible
- **Curvilinear:**
  Pro: more flexible than rectilinear grids
  Con: Explicit storage of point locations, cell-search is required (time consuming)
- **Unstructured:**
  Pro: Maximal flexibility and local adoption
  Con: Additional storage requirements (cells, i.e. topology), cell search (time consuming)

Signal Discretization and Quantization

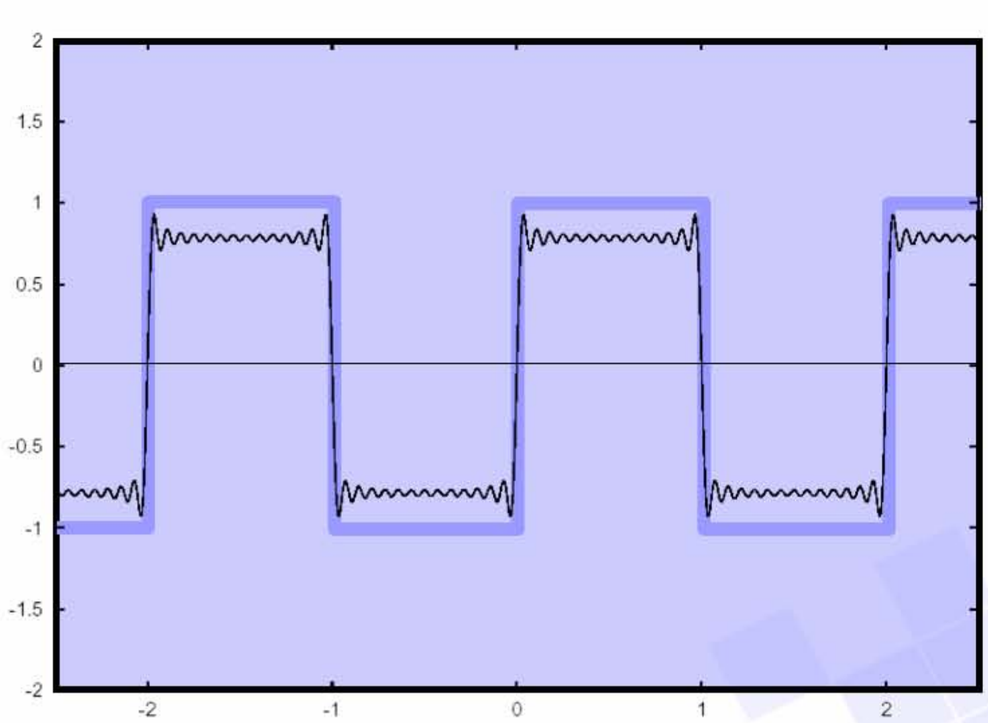
Continuous signals are sampled, yielding a ***discrete*** signal parameters

Quantization assigns the ***values*** for the parameter computer internal representations, e.g. 32 Bit



Frequency Spectrum

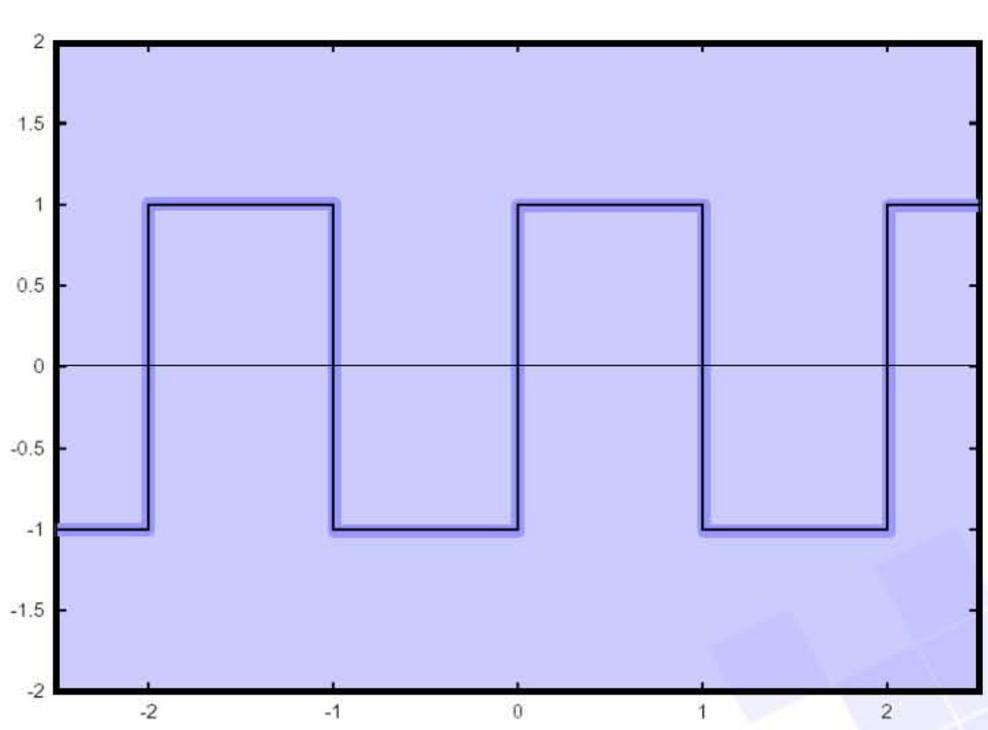
- Each continuous signal $f(x)$ can be represented as superposition of ***harmonic (sinusoidal) functions with varying frequency***
- Example:



$$\sin(x) + \frac{1}{3}\sin(3x) + \frac{1}{5}\sin(5x) + \frac{1}{7}\sin(7x) + \frac{1}{9}\sin(9x) + \frac{1}{11}\sin(11x) + \frac{1}{13}\sin(13x)$$

Frequency Spectrum

- Each continuous signal $f(x)$ can be represented as superposition of ***harmonic (sinusoidal) functions with varying frequency***
- Example:

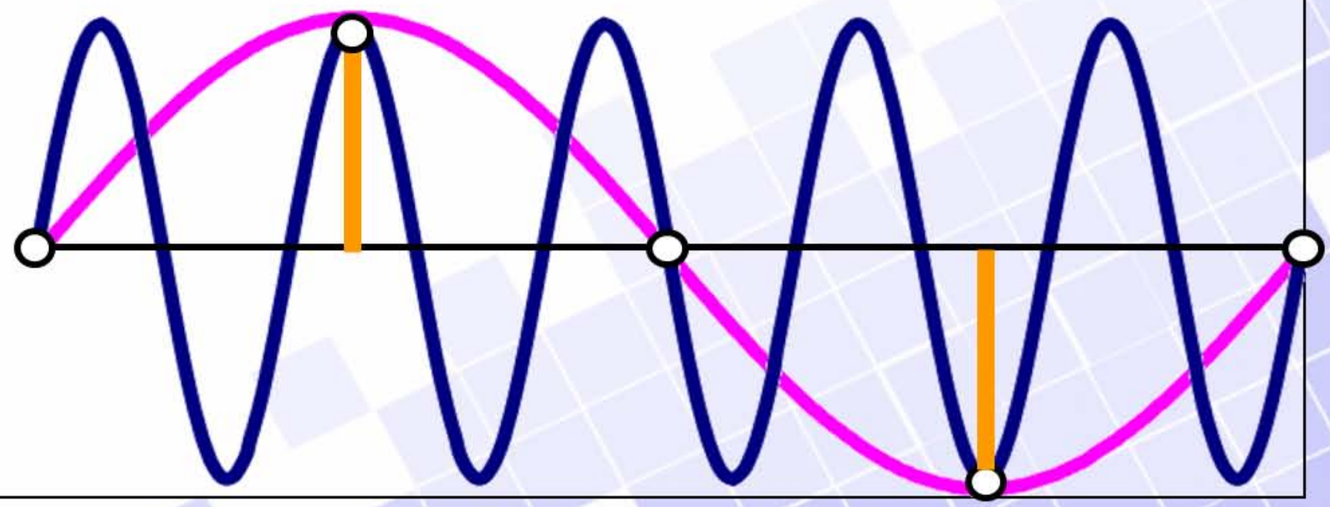


$$\sum_{n=1,3,5\ldots}^{\infty} \frac{1}{n} \sin(n \cdot x)$$

Signals with ***„sharp edges"*** contain infinitely high frequencies

Sampling Theorem

- Question: How do discrete and continuous signal relate to each other?
- If the continuous signal $f(x)$ has a bounded spectrum $F(t)$, e.g. $F(t) = 0$ for $|t| > t_{max}$ then the continuous signal can be reconstructed from the discrete signal, if sampled at step-size $\tau < 1/2t_{max}$
- Otherwise ***aliasing*** occurs



Derivatives and Gradient

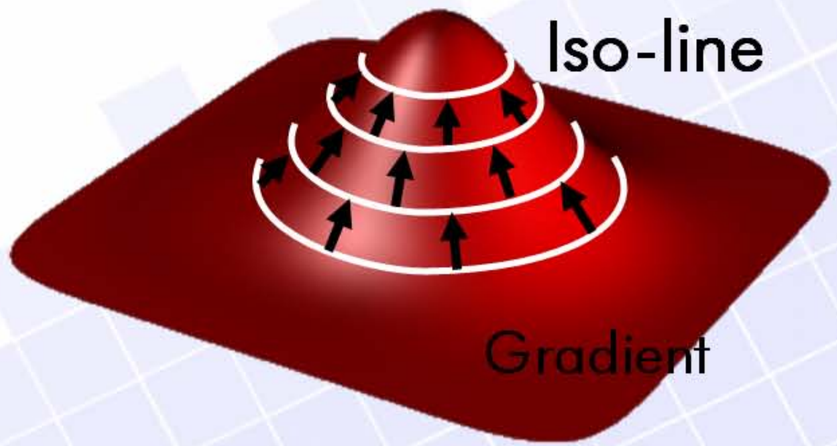
- Gradient:
  Vector consisting of
  partial derivatives

$$\mathrm{grad} f(x,y) = \begin{pmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{pmatrix}$$

- Properties of the gradient-vector:
  - points in direction of max. ascent
  - is perpendicular to iso-lines
  - is zero at (local) extremal points

Example:

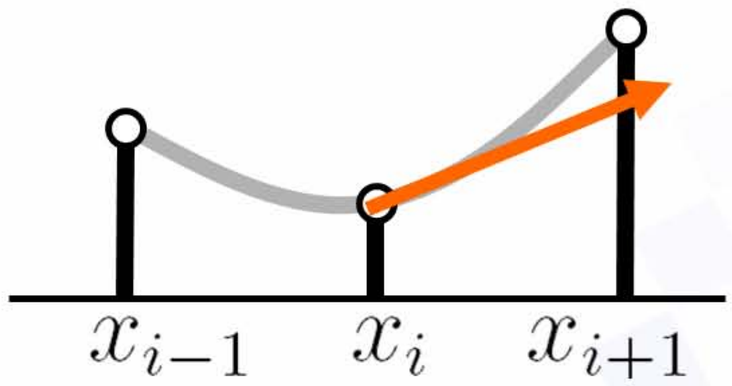
2D Hightfield



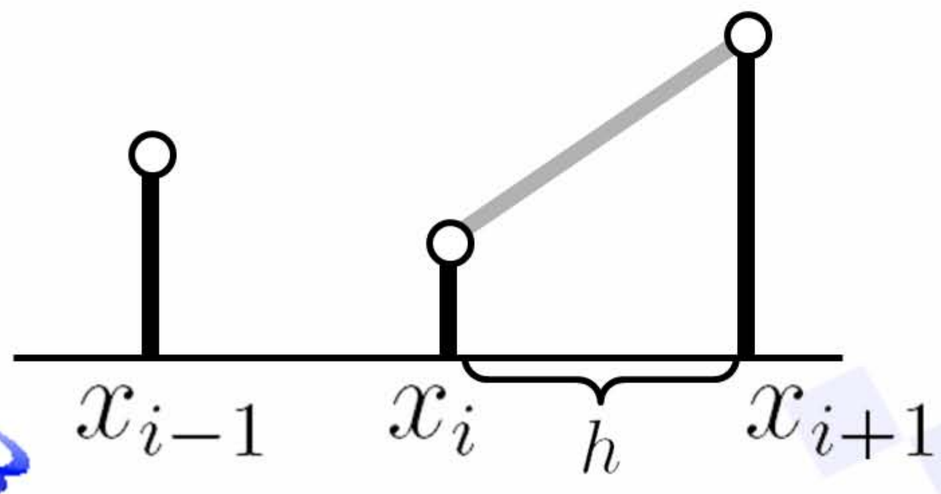
Differentiation on Grids

How to compute derivatives on a grid?

- **Approach 1:** Determine a (local) interpolating function and differentiate it



- **Approach 2:** Use finite differences



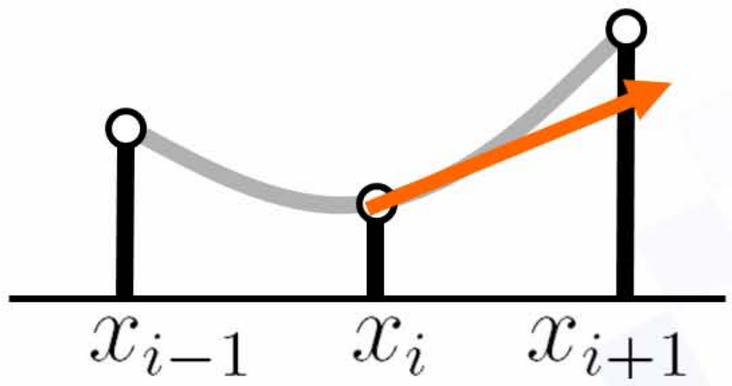
Forward:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{h}$$

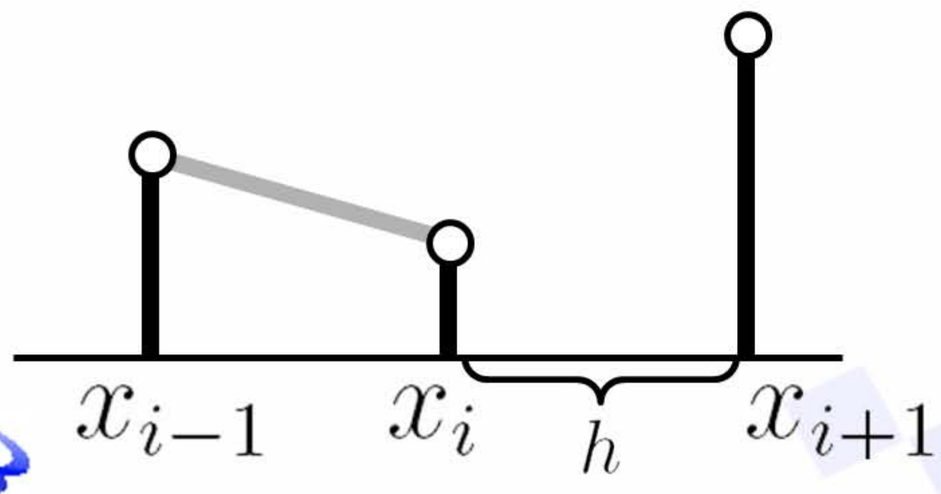
Differentiation on Grids

How to compute derivatives on a grid?

- **Approach 1:** Determine a (local) interpolating function and differentiate it



- **Approach 2:** Use finite differences



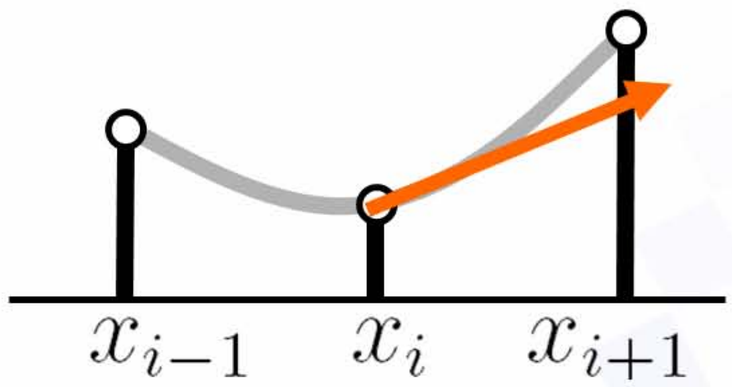
Backward:

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{h}$$

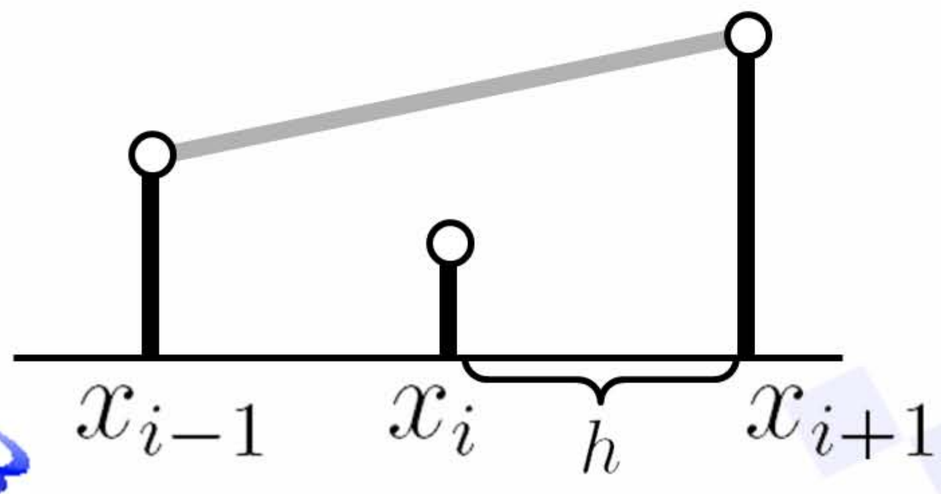
Differentiation on Grids

How to compute derivatives on a grid?

- **Approach 1:** Determine a (local) interpolating function and differentiate it



- **Approach 2:** Use finite differences

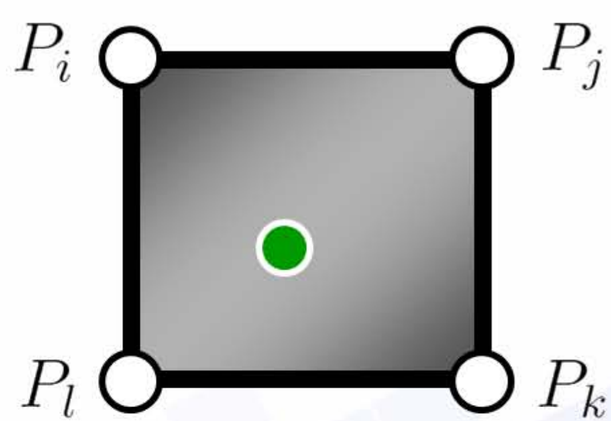


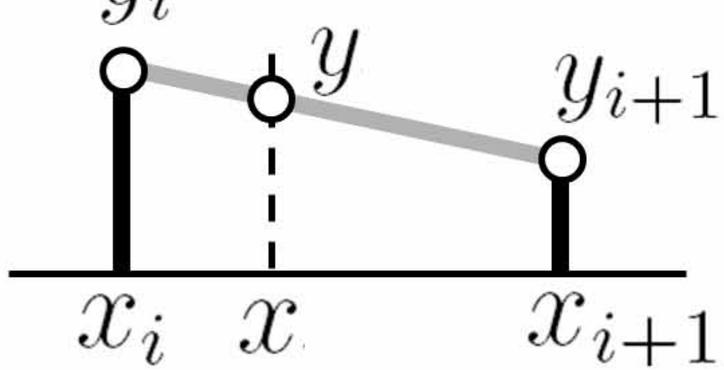
Central:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_{i-1})}{2h}$$

Local Interpolation

- Given: Data values at the grid points
- Needed: Data value at an interior point
- **Approach:** Linear Interpolation





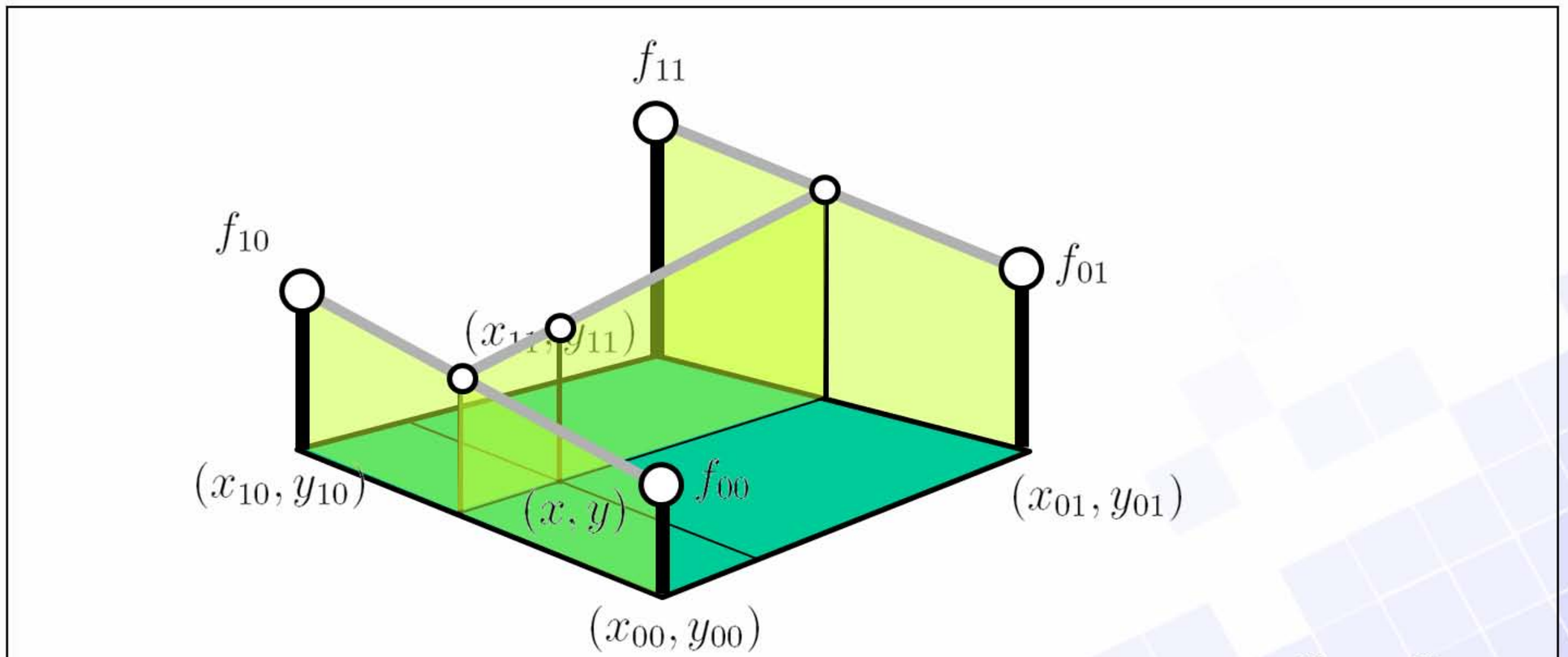
Linear ***combination,***

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

with

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

Bilinear Interpolation *(2D)*

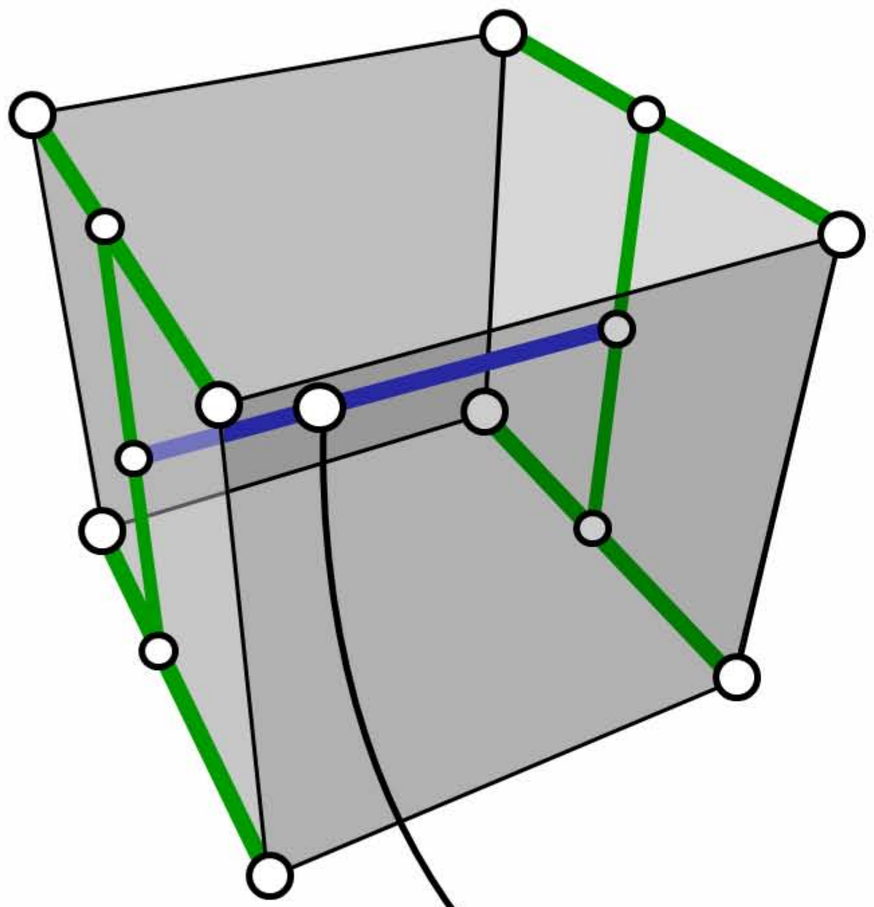


$$f_0 = \alpha f_{10} + (1-\alpha) f_{00}$$
$$f_1 = \alpha f_{11} + (1-\alpha) f_{01}$$
$$f(x,y) = \beta f_1 + (1-\beta) f_0$$

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

$$\beta = \frac{y - y_{00}}{y_{01} - y_{00}}$$

Trilinear Interpolation *(3D)*



4-fold linear interpolation

$$
\begin{aligned}
f_{00} &= \alpha\, f_{001} + (1-\alpha) f_{000} \\
f_{01} &= \alpha\, f_{011} + (1-\alpha) f_{010} \\
f_{10} &= \alpha\, f_{101} + (1-\alpha) f_{100} \\
f_{11} &= \alpha\, f_{111} + (1-\alpha) f_{110}
\end{aligned}
$$

2-fold bilinear interpolation

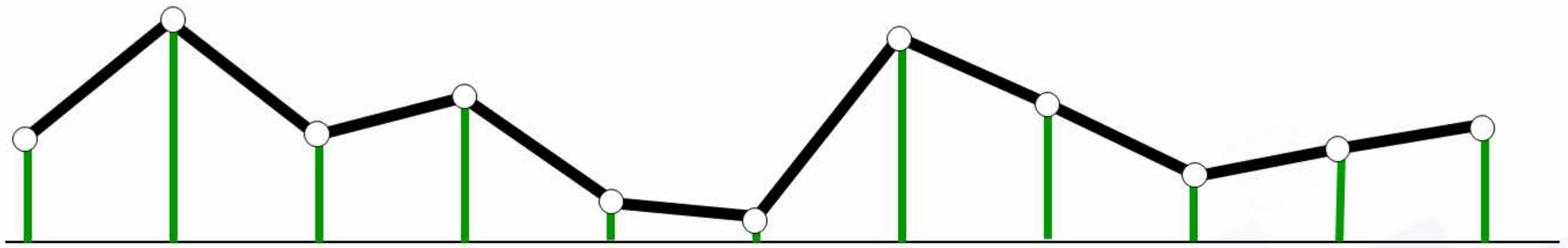
$$
\begin{aligned}
f_{0} &= \beta\, f_{01} + (1-\beta) f_{00} \\
f_{1} &= \beta\, f_{11} + (1-\beta) f_{10}
\end{aligned}
$$

1 trilinear interpolation

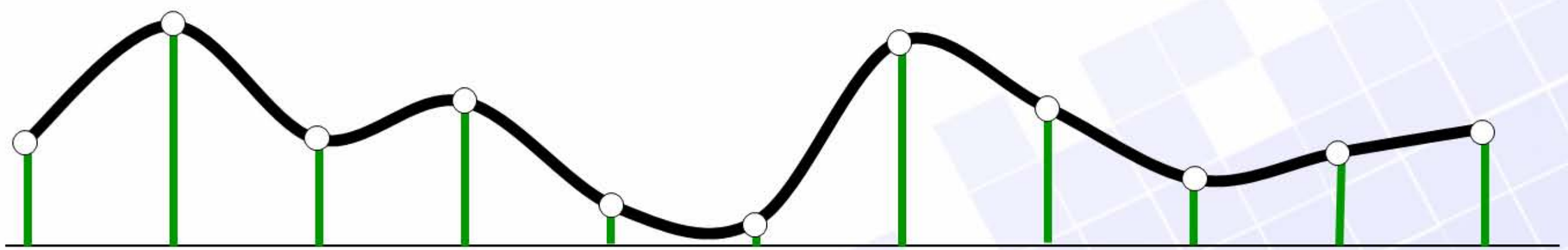
$$
f = \gamma\, f_{1} + (1-\gamma) f_{0}
$$

Higher Order Interpolation

- Lineare interpolation yields *C_0-continuous interpolant*



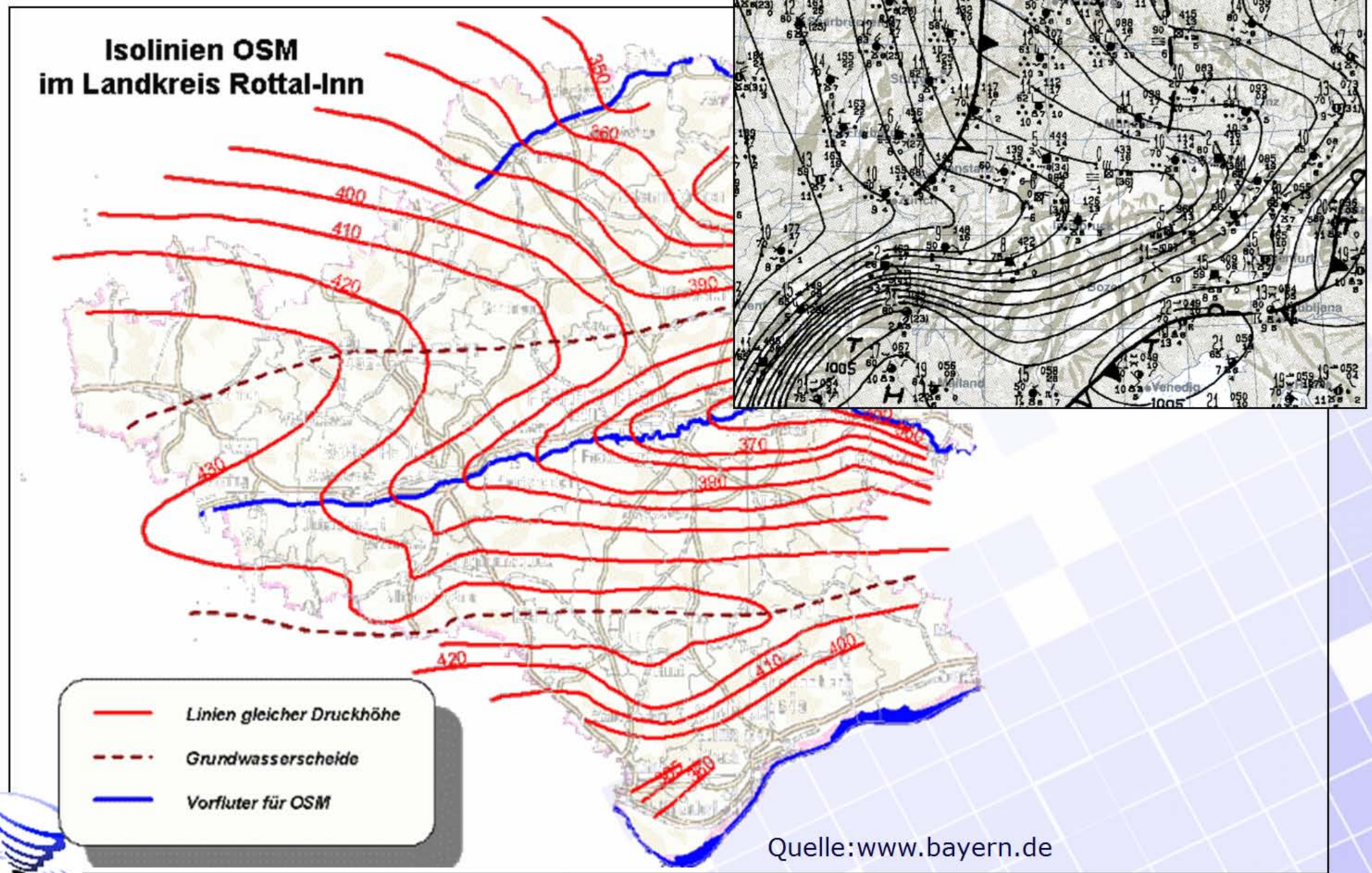
- Higher order, e.g. *C_1, results from higher order functions, e.g. cubic polynomials*



Further Fundamental Topics

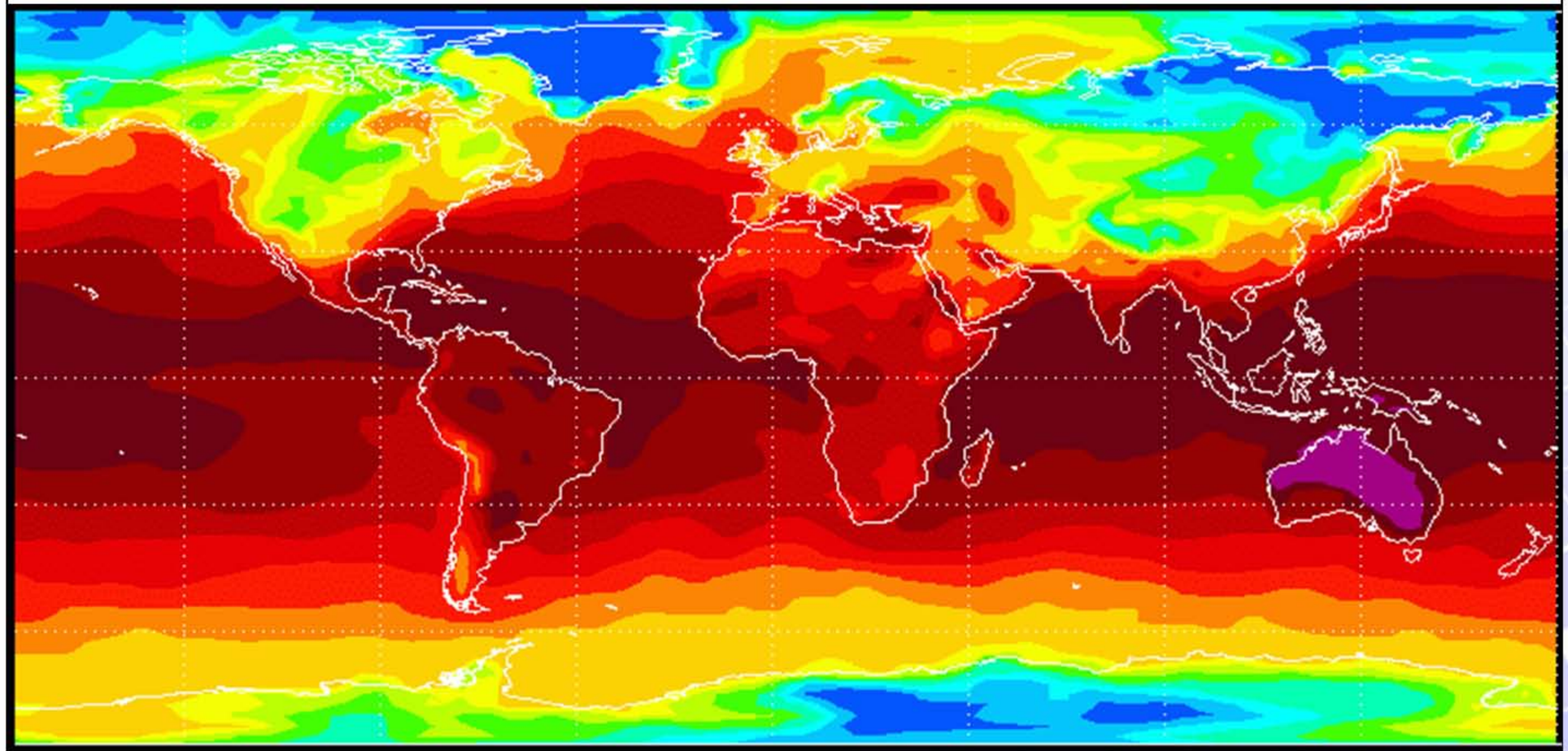
- Sampling and interpolation on non-uniform grids
  - Barycentric coordinates
- Filtering
  - Linear filtering, e.g. high-pass, low-pass
  - Non-linear filtering, e.g. morphological operators or anisotropic filters
- Computation of grids
  - Triangulations/Tetrahedrization
  - Hexagonization

2D Scalar-Fields



Example

- **Meteorology:** Temperature Distribution



Quelle: www.climateprediction.net

Overview 2D Scalar Fields

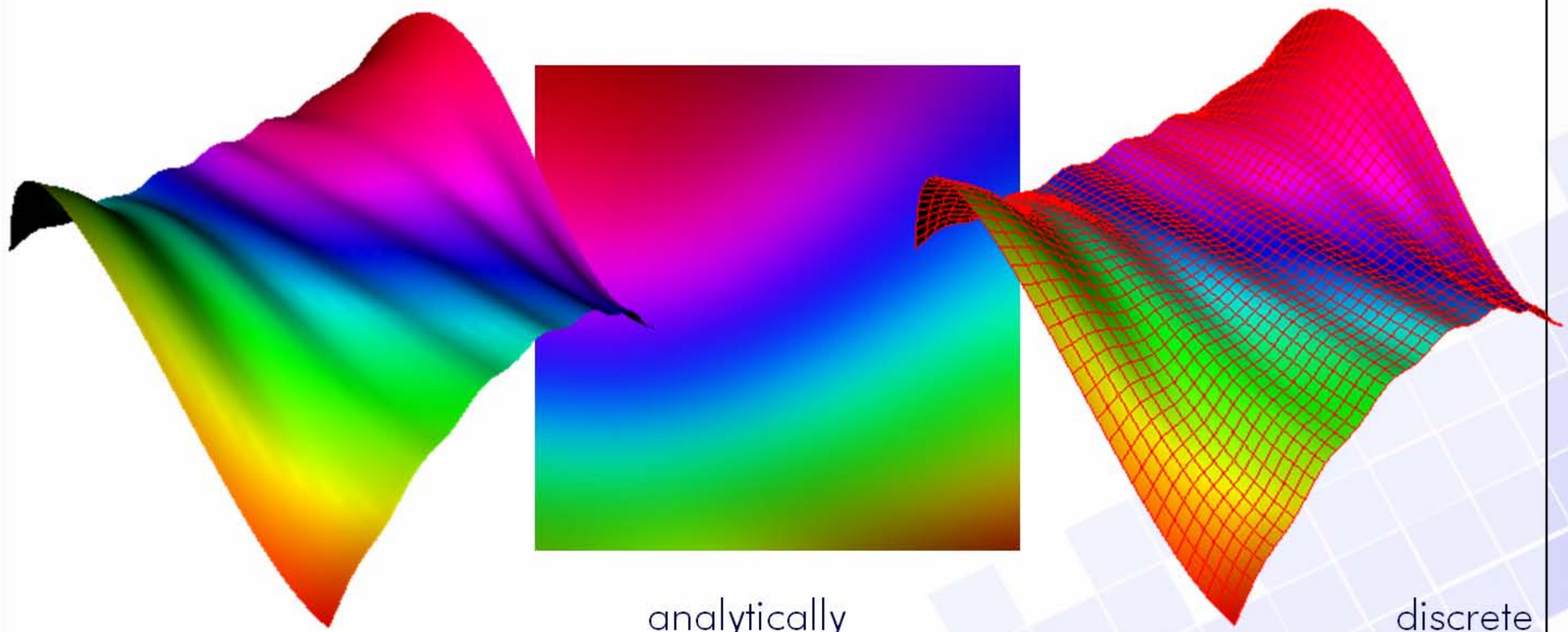
- Scalar-Fields:
  - are rarely given analytically: $s = f(x, y)$
  - usually, we have discrete data:

$$(x_i, y_i) \mapsto f_i \qquad i \in [0, \ldots N]$$

- Visualization Approaches:
  - Color Plots
  - Surface Plots
  - Iso-lines

2D Scalar-Fields

… need not to be planar!



analytically

$$\begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix} \leftarrow \quad (u,v) \quad \rightarrow f(u,v)$$

discrete

$$\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \leftarrow \quad (u_i,v_i) \quad \rightarrow f_i$$

Color-Coding

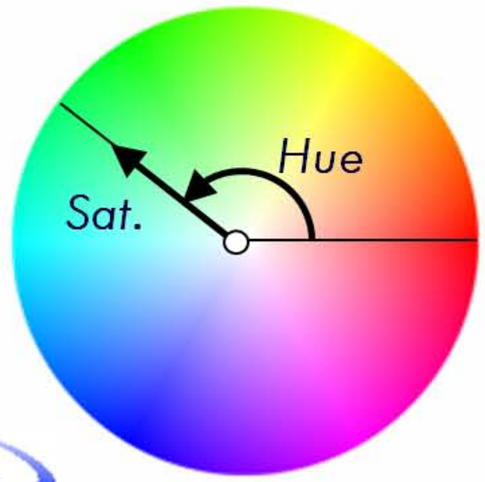
- Specify a color table, i.e.

  A mapping: Scalar value s ⟶ color value

  Example: Given a 2D pressure distribution

  $$\rho_{\min} \leq \rho \leq \rho_{\max}$$

  Linear mapping to HSV-color model's ***Hue-comp.***



Hue = angle
Saturation = distance from center point

Color-Coding

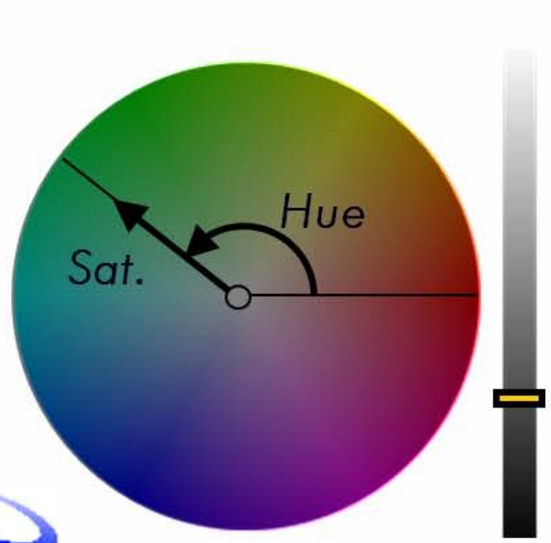
- Specify a color table, i.e.

  A mapping: Scalar value s ⟶ color value

  Example: Given a 2D pressure distribution

  $$\rho_{\min} \leq \rho \leq \rho_{\max}$$

  Linear mapping to HSV-color model's ***Hue-comp.***



Hue = angle
Saturation = distance from center point
Value = brightness

$$\alpha = \frac{\rho - \rho_{\min}}{\rho_{\max} - \rho_{\min}} \quad \in \quad [0, 1]$$

Color-Coding

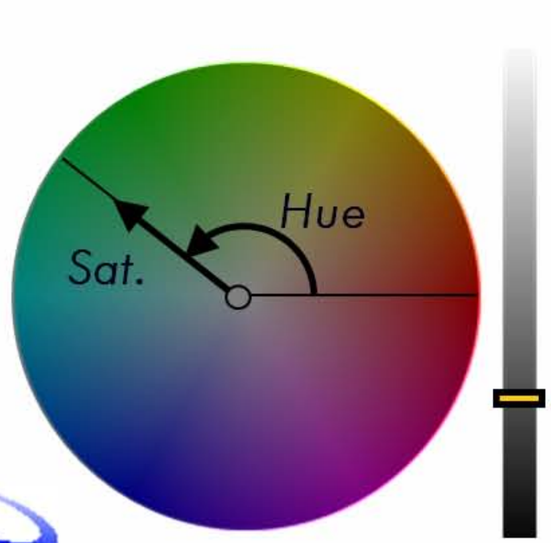
- Specify a color table, i.e.

  A mapping: Scalar value s ⟶ color value

  Example: Given a 2D pressure distribution

  $$\rho_{\min} \leq \rho \leq \rho_{\max}$$

  Linear mapping to HSV-color model's ***Hue-comp.***

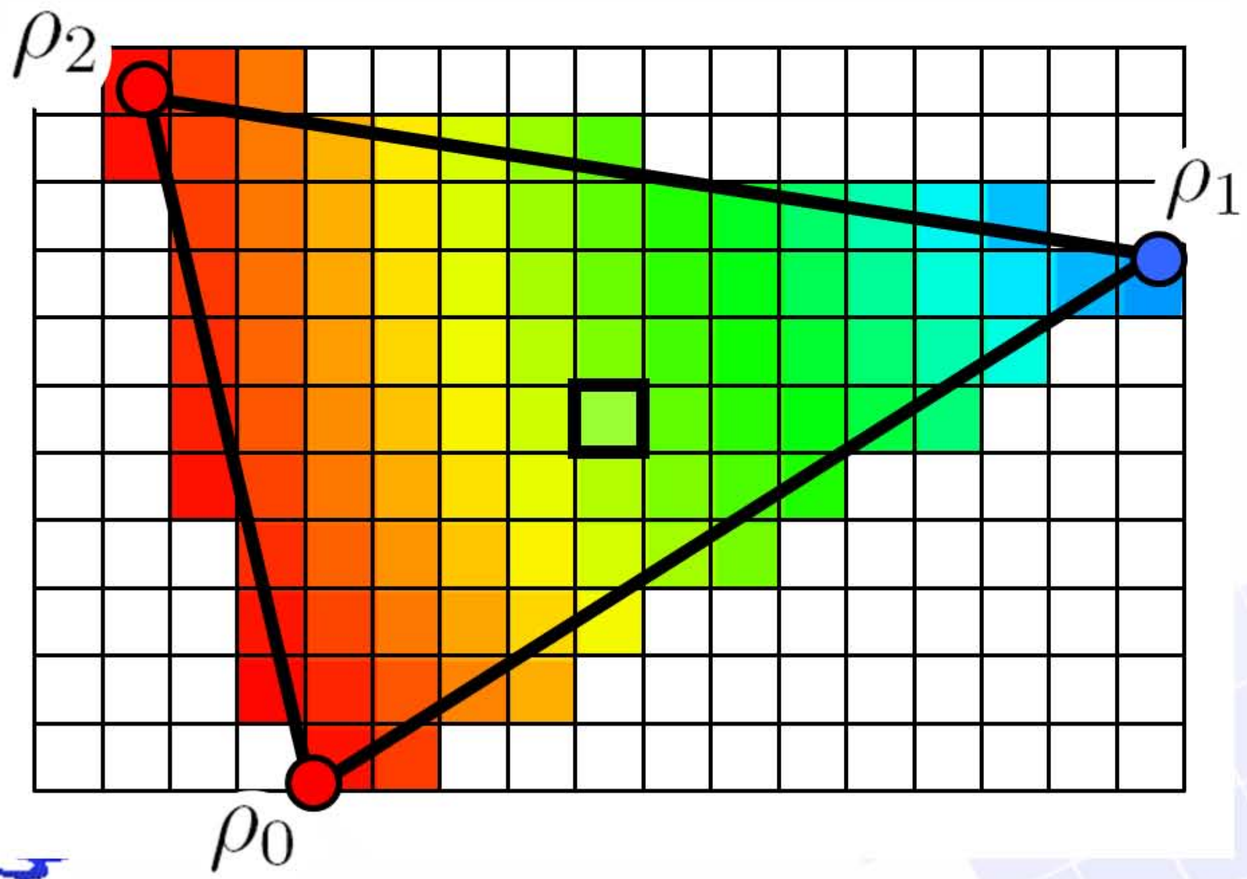


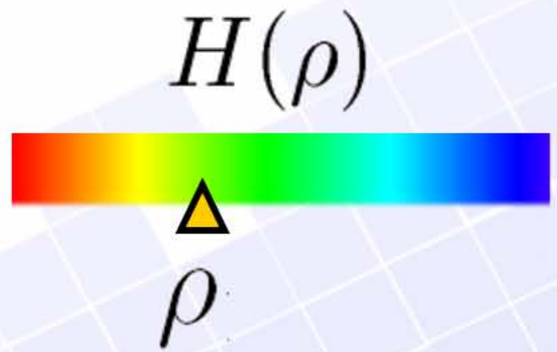
Hue = angle
Saturation = distance from center point
Value = brightness

$$H(\rho) = \alpha H_1 + (1 - \alpha) H_0$$

Rasterisation and Interpolation

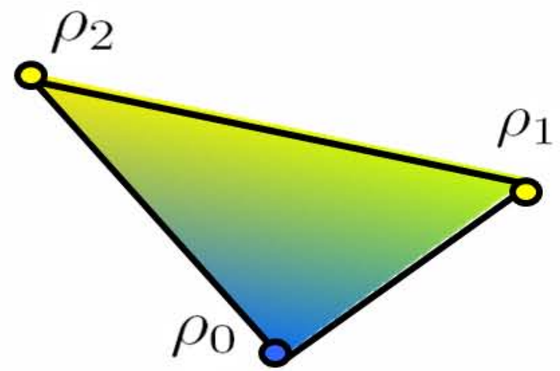
- How to compute color value at interior points?
  - Interpolate scalar value for each pixel
  - Map per-pixel scalar value to color

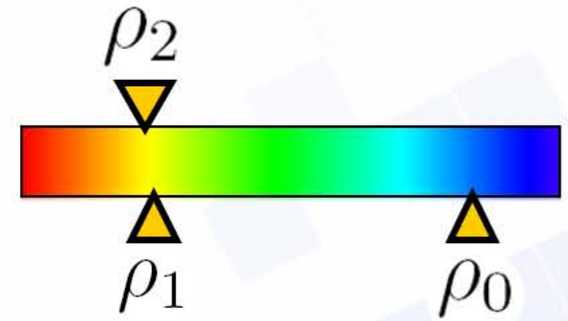




Simple Implementation

- Standard approach using Graphics Hardware:
  - Define per-vertex color
  - Rasterization interpolates colors within the triangle

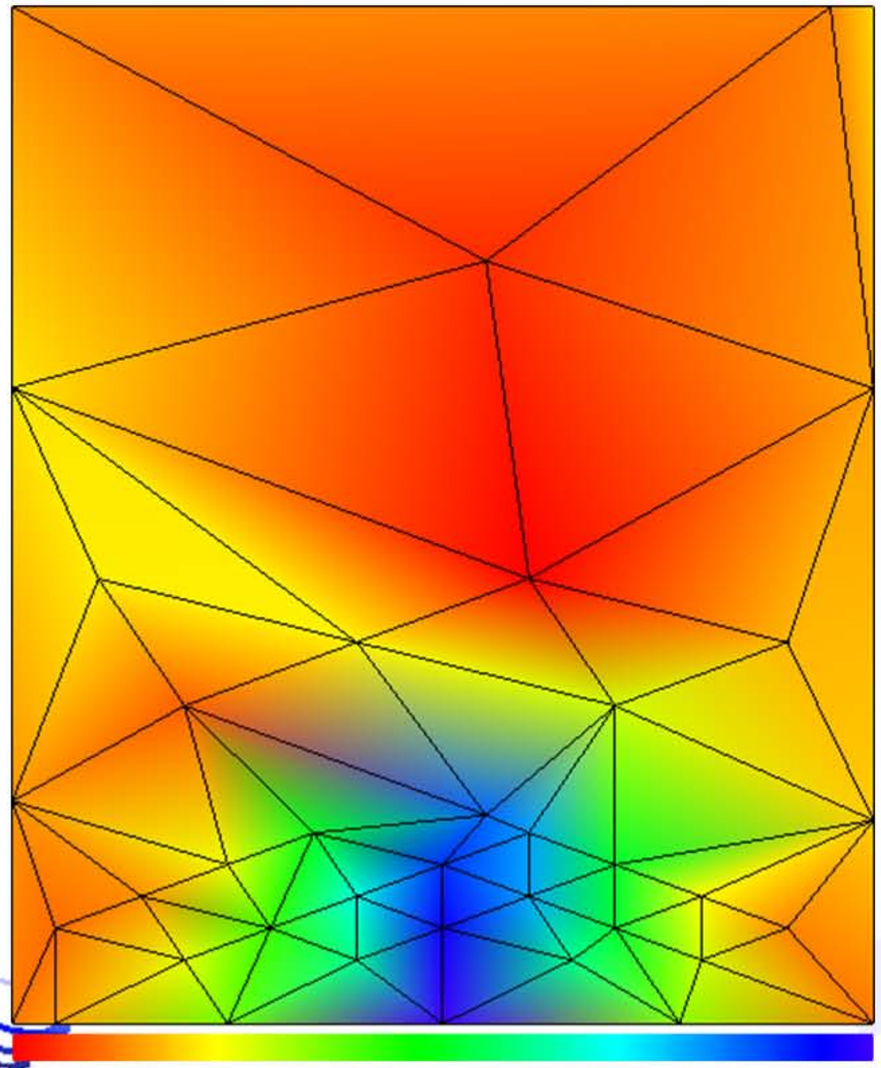




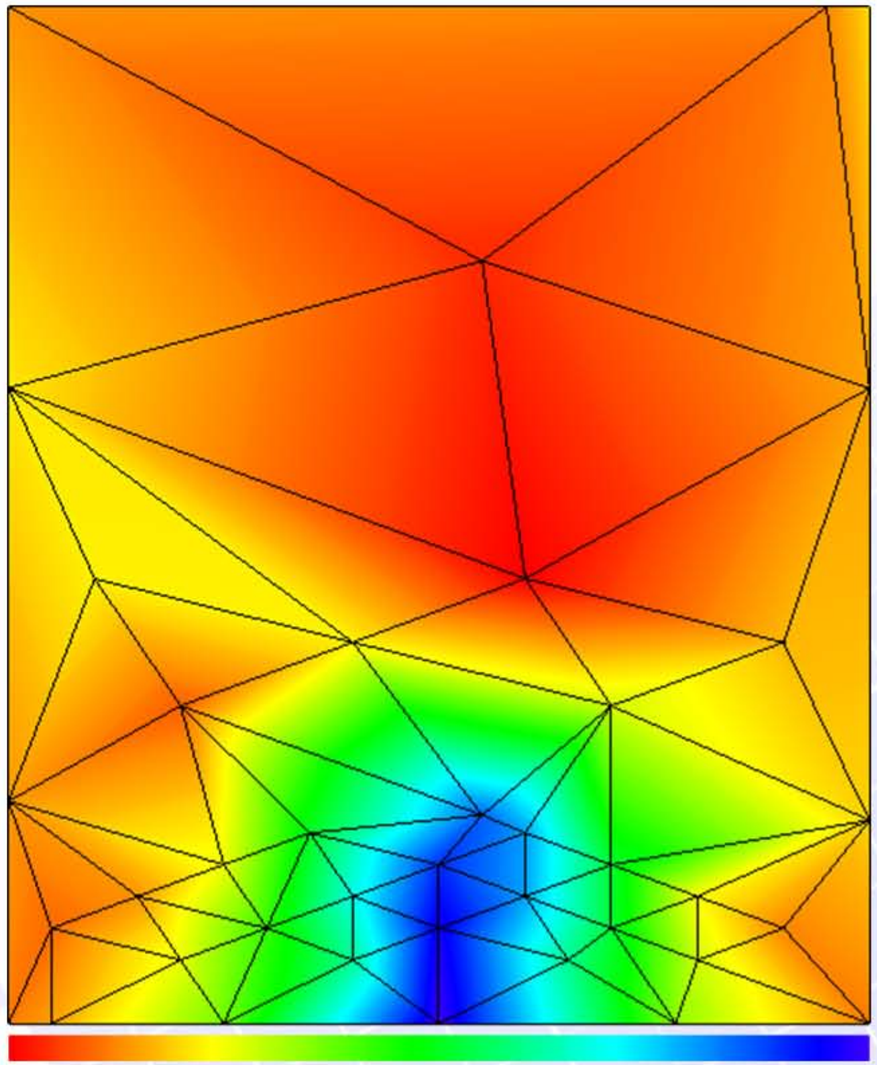
- But: color-interpolation != value interpolation
- Better: Use texture to store color table
  - Per-vertex value → texture coordinates
  - Texture interpolation + lookup yields correct result

Comparison

Color interpolation



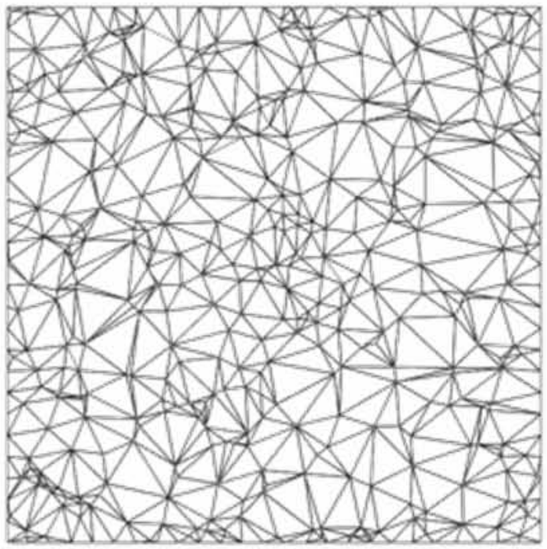
Texture interpolation



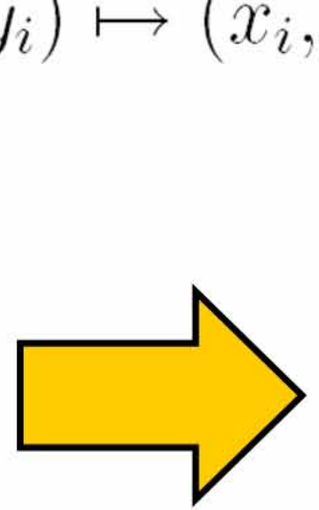
Surface Plots

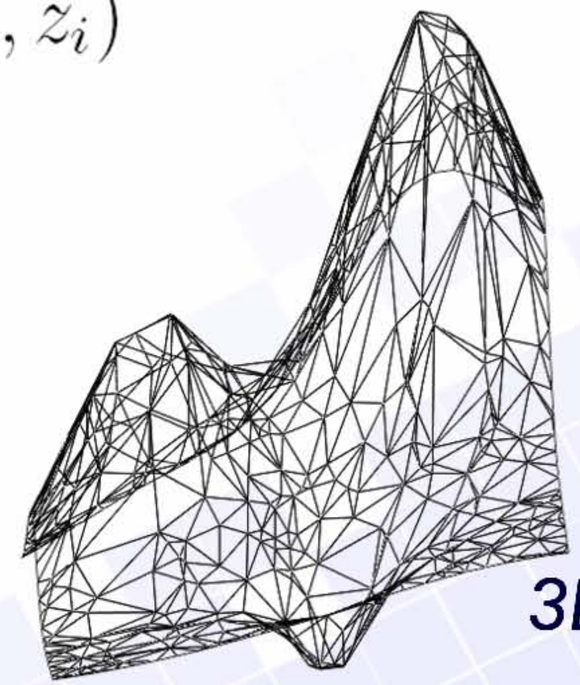
- Interprete scalar field as hightfield & draw 3D surface

$$(x_i, y_i) \mapsto (x_i, y_i, z_i)$$



2D





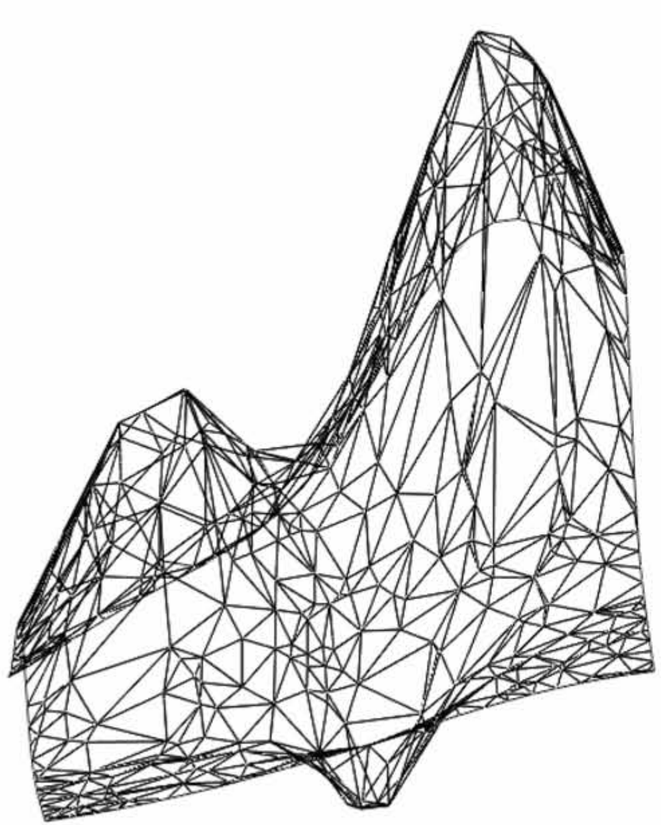
3D

$$\alpha = \frac{\rho - \rho_{\min}}{\rho_{\max} - \rho_{\min}}$$

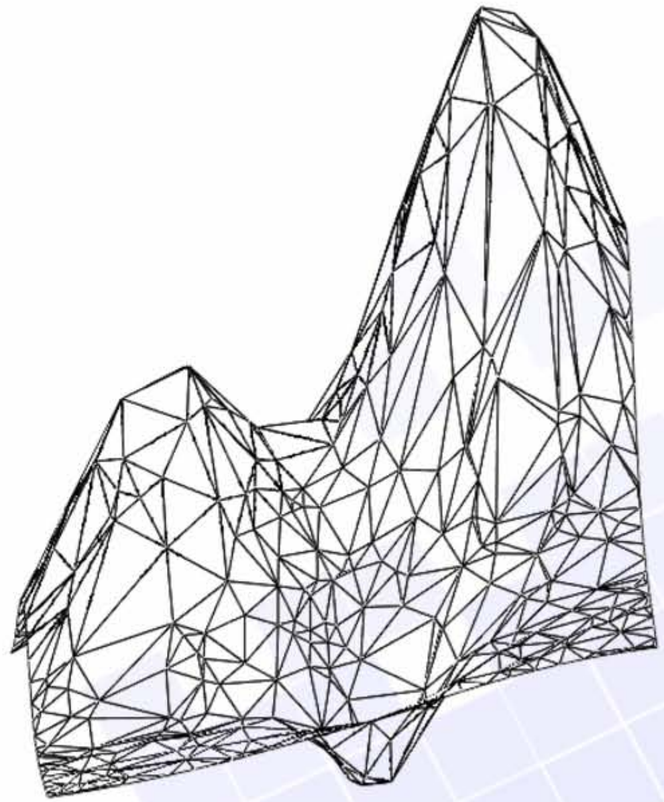
$$z_i = \alpha\, z_{\max} + (1 - \alpha) z_{\min}$$

Surface Plots

- Line-based plots



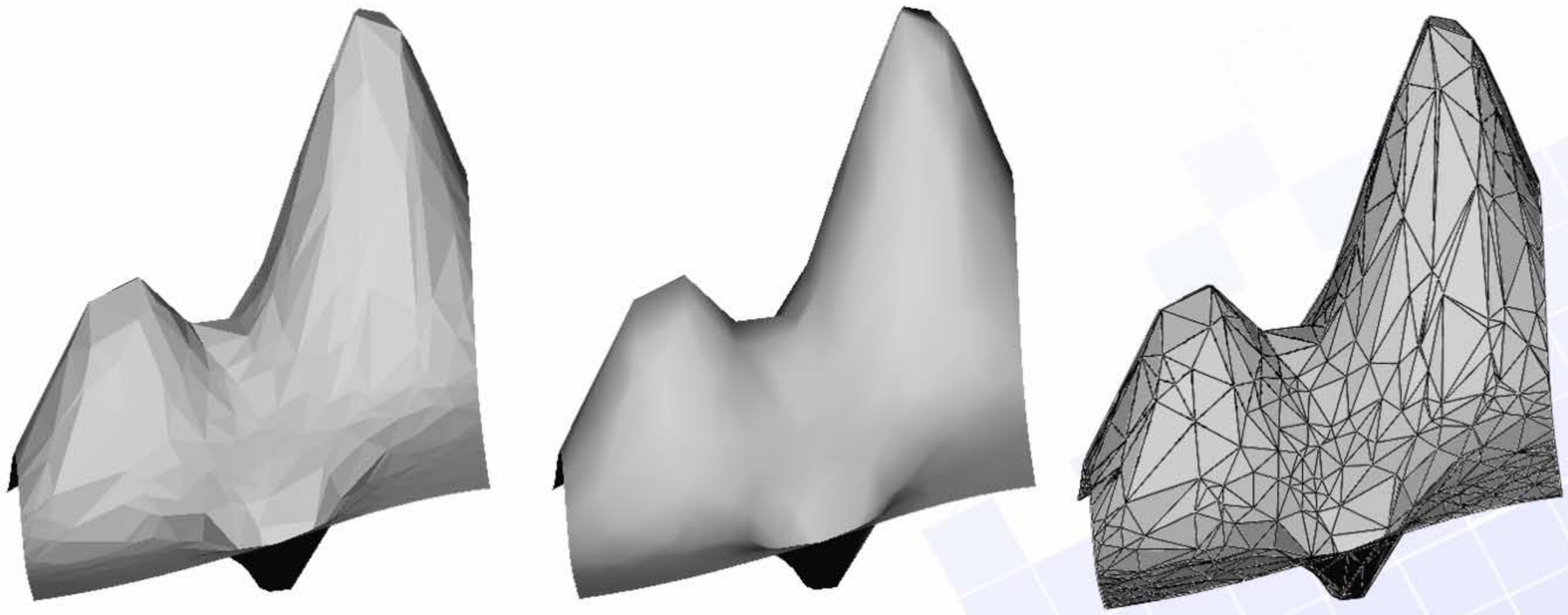
„wireframe“



„hidden line removal“

Surface Plots

- Surface plots using *Shading*

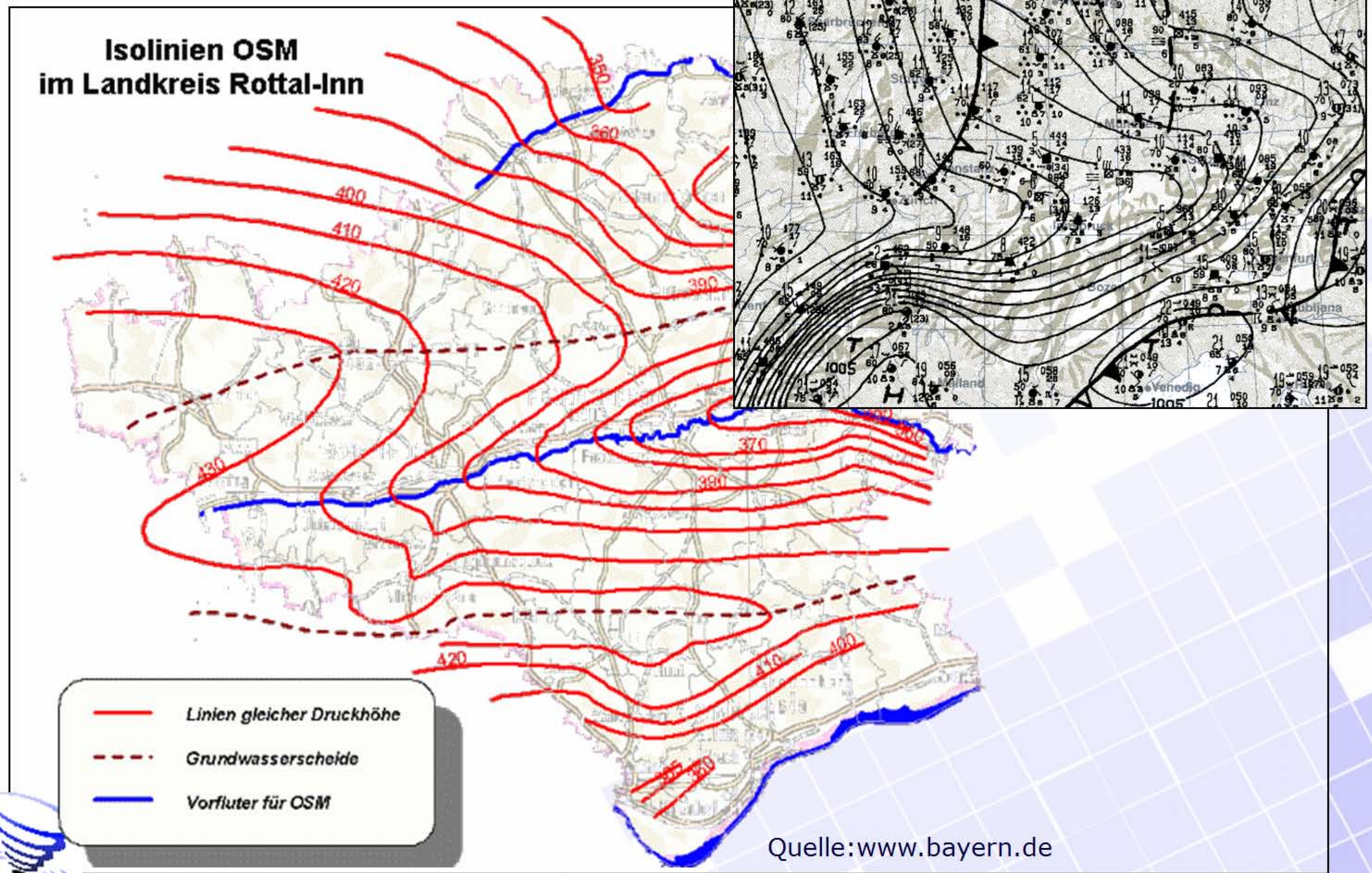


flat shading

smooth shading
Gouraud-Shading

combination with wireframe

2D Scalar-Fields



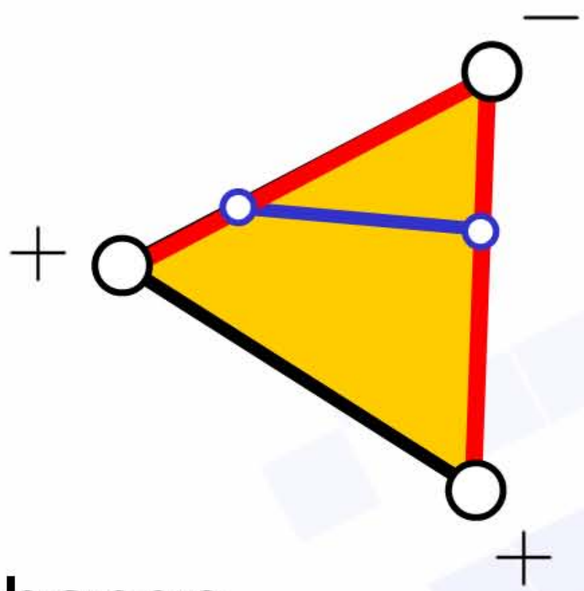
Quelle:www.bayern.de

Definition

- Given a scalar field $f : \mathbb{R}^2 \mapsto \mathbb{R}$ and a scalar value $c \in \mathbb{R}$ an iso-line is implicitly defined by $\{(x, y) \mid f(x, y) = c\}$
- If f is differentiable and $\mathrm{grad}(f) \neq 0$, than iso-lines are curves.
- It $\mathrm{grad}(f) = 0$, than iso-lines can degenerate to points or regions.

Iso-lines on Triangular Meshes

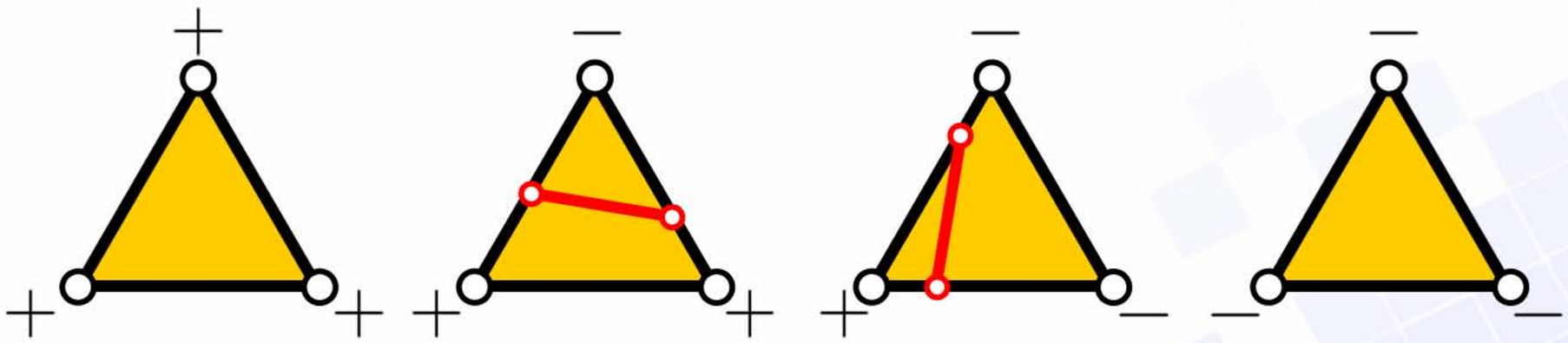
- Cell-order approach:
  - Handle each cell separately
  - Mark all vertices with
    - $+$ wenn $f_i > c$
    - $-$ wenn $f_i \leq c$



- Check each edge for sign change
  - If sign change: find point of intersection by interpolation along edge
  - Connect points of intersection using straight lines

Iso-lines on Triangular Meshes

- Either we have none or two edges with sign change, i.e. none or two intersection points



No point of intersection

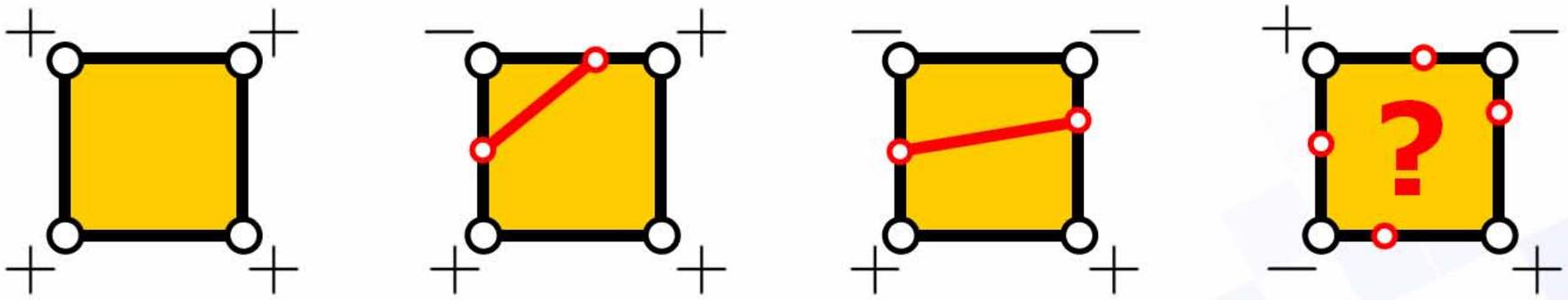
2 intersections
1 line segment

No point of intersection

(symmetric cases have been omitted)

Iso-lines on Quadilateral Meshes

- We have none, two or four edges with sign changes



No point of intersection

2 intersections
1 line-segment

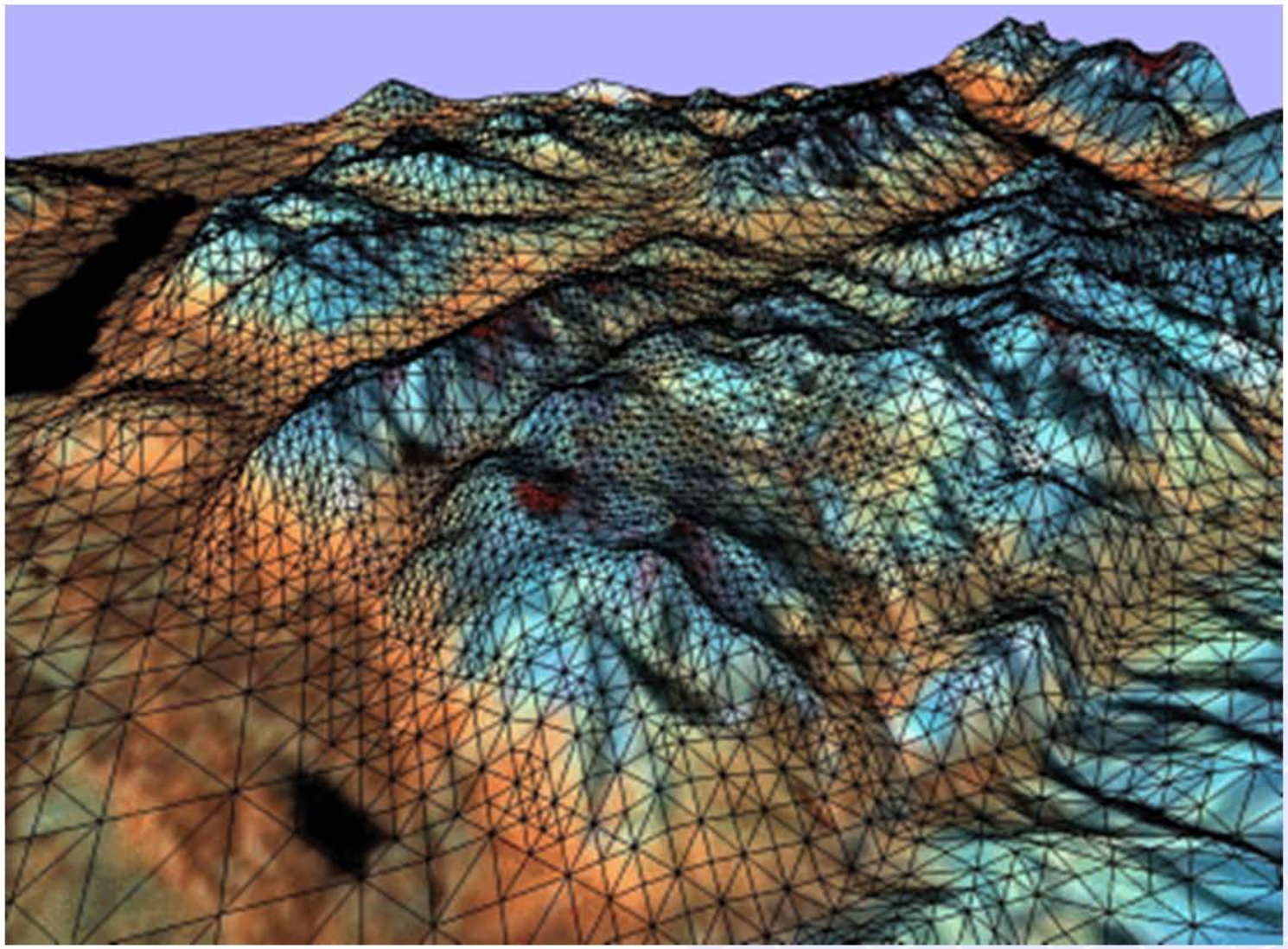
4 intersection
2 line-segment
ambigious

(symmetric cases have been omitted)

- Care has to be taking to make the right decision in case of four intersection points

Combined Scalar-Field Visualization

- Color-Codierung + Surface Plot



2D Vector-Fields

- Given a 2D vector field

$$\vec{v}(x,y) = \begin{pmatrix} v_x(x,y) \\ v_y(x,y) \end{pmatrix}$$

Compute partial derivatives of the Vector field

Change of v_x in x-direction

$\frac{\partial v_x}{\partial x}$ $\frac{\partial v_x}{\partial y}$

$\frac{\partial v_y}{\partial x}$ $\frac{\partial v_y}{\partial y}$

Change of v_x in y-direction

Change of v_y in x-direction

Change of v_y in y-direction

2D Vector-Fields

- Given a 2D vector field

$$\vec{v}(x,y) = \begin{pmatrix} v_x(x,y) \\ v_y(x,y) \end{pmatrix}$$

- The *total differential* is given by

$$J_{\vec{v}} = \begin{pmatrix} \frac{\partial v_x}{\partial x} & \frac{\partial v_x}{\partial y} \\ \frac{\partial v_y}{\partial x} & \frac{\partial v_y}{\partial y} \end{pmatrix}$$

The ***Jacobian (fundamental matrix)*** describes the local change of the vector field

Local Characteristics

Divergence is the sum of the Jacobians diagonal values

$$\begin{pmatrix} \frac{\partial v_x}{\partial x} & \frac{\partial v_x}{\partial y} \\ \frac{\partial v_y}{\partial x} & \frac{\partial v_y}{\partial y} \end{pmatrix}$$

$$\operatorname{div} \vec{v}(\vec{x}) = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}$$

- Divergence describes the convergence (**sink**) an the divergence (**source**) of the flow

Source at $\vec{x}_0$

$\operatorname{div} \vec{v}(\vec{x}_0) > 0$

Sink at $\vec{x}_0$

$\operatorname{div} \vec{v}(\vec{x}_0) < 0$

Neither sink nor source if

$\operatorname{div} \vec{v}(\vec{x}_0) = 0$

Local Characteristics

Rotation/curl is computed as the difference of the off-diagonal elements of the Jacobian

$$\begin{pmatrix} \frac{\partial v_x}{\partial x} & \frac{\partial v_x}{\partial y} \\ \frac{\partial v_y}{\partial x} & \frac{\partial v_y}{\partial y} \end{pmatrix}$$

$$\operatorname{rot} \vec{v}(\vec{x}) = \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y}$$

- Rotation tells us something about the ***local rotation***, e.g. the **vorticity** of the flow

No rotation at $\vec{x}_0$

$$\operatorname{rot} \vec{v}(\vec{x_0}) = 0$$

Local rotation at $\vec{x}_0$

$$\operatorname{rot} \vec{v}(\vec{x_0}) \neq 0$$

Local Characteristics

- ***Critical Points***:

  Points with no flow, i.e.

$$\vec{v}(\vec{x_0}) = \vec{0}$$

  - Taylor expansion yields

$$\vec{v}(\vec{x}_0 + \vec{h}) \approx \vec{v}(\vec{x_0}) + J_{\vec{v}}(\vec{x_0})\,\vec{h}$$

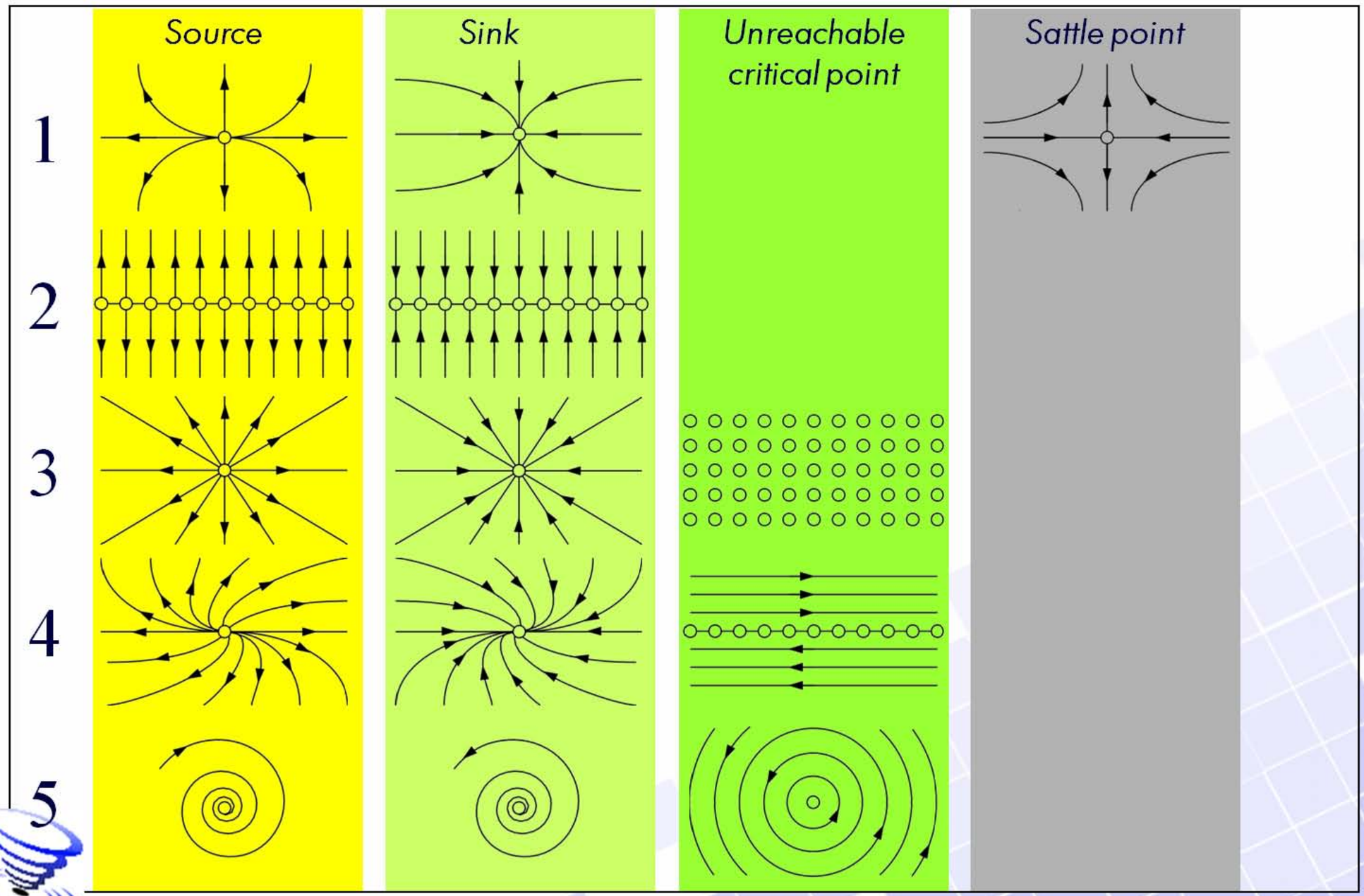
  and thus we have

$$\vec{v}(\vec{x}_0 + \vec{h}) \approx J_{\vec{v}}(\vec{x_0})\,\vec{h}$$

  →The local behavior of the flow around a critical point is characterized by the Jacobian $J_{\vec{v}}(\vec{x_0})$

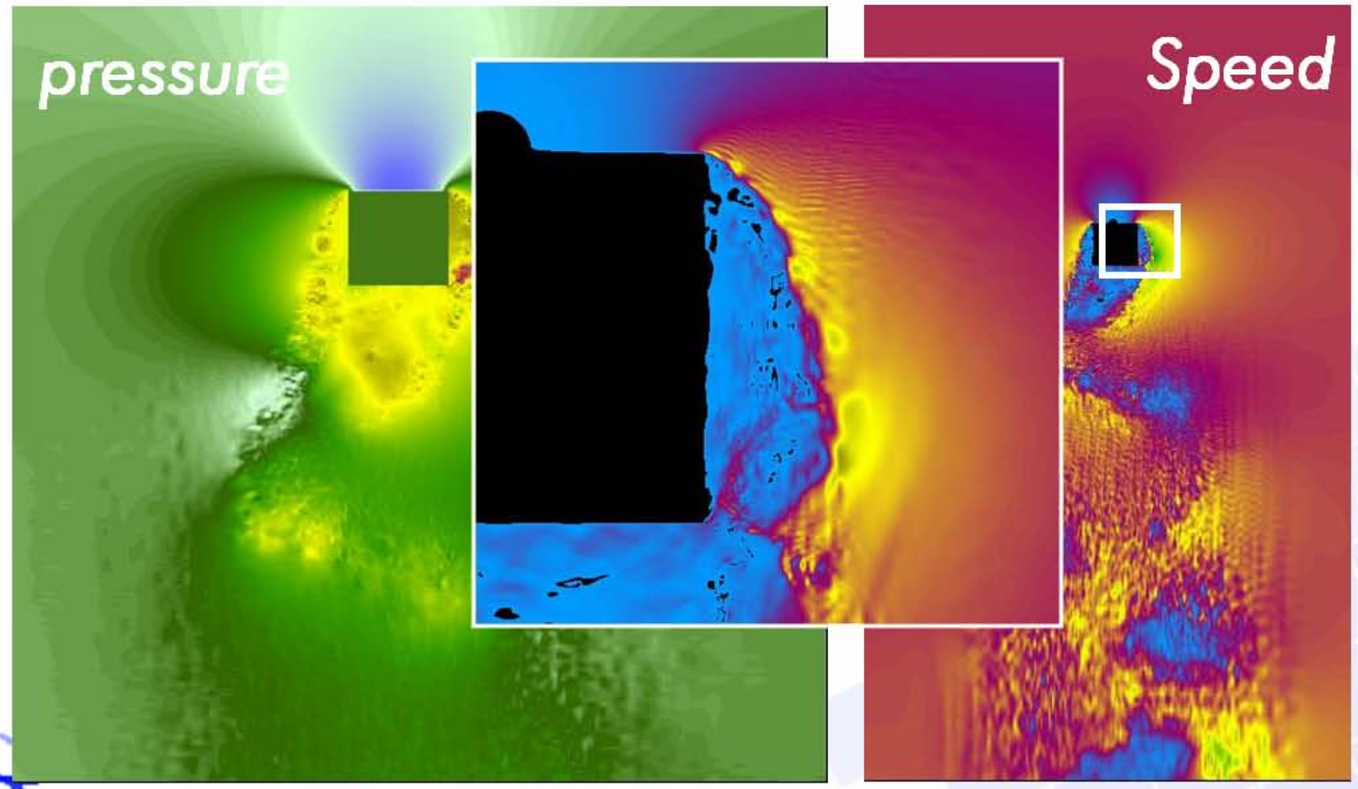
- The complete characterization of critical points can be determined using the Eigen-vectors and –values of the Jacobian

2D Vector-Field Topologies



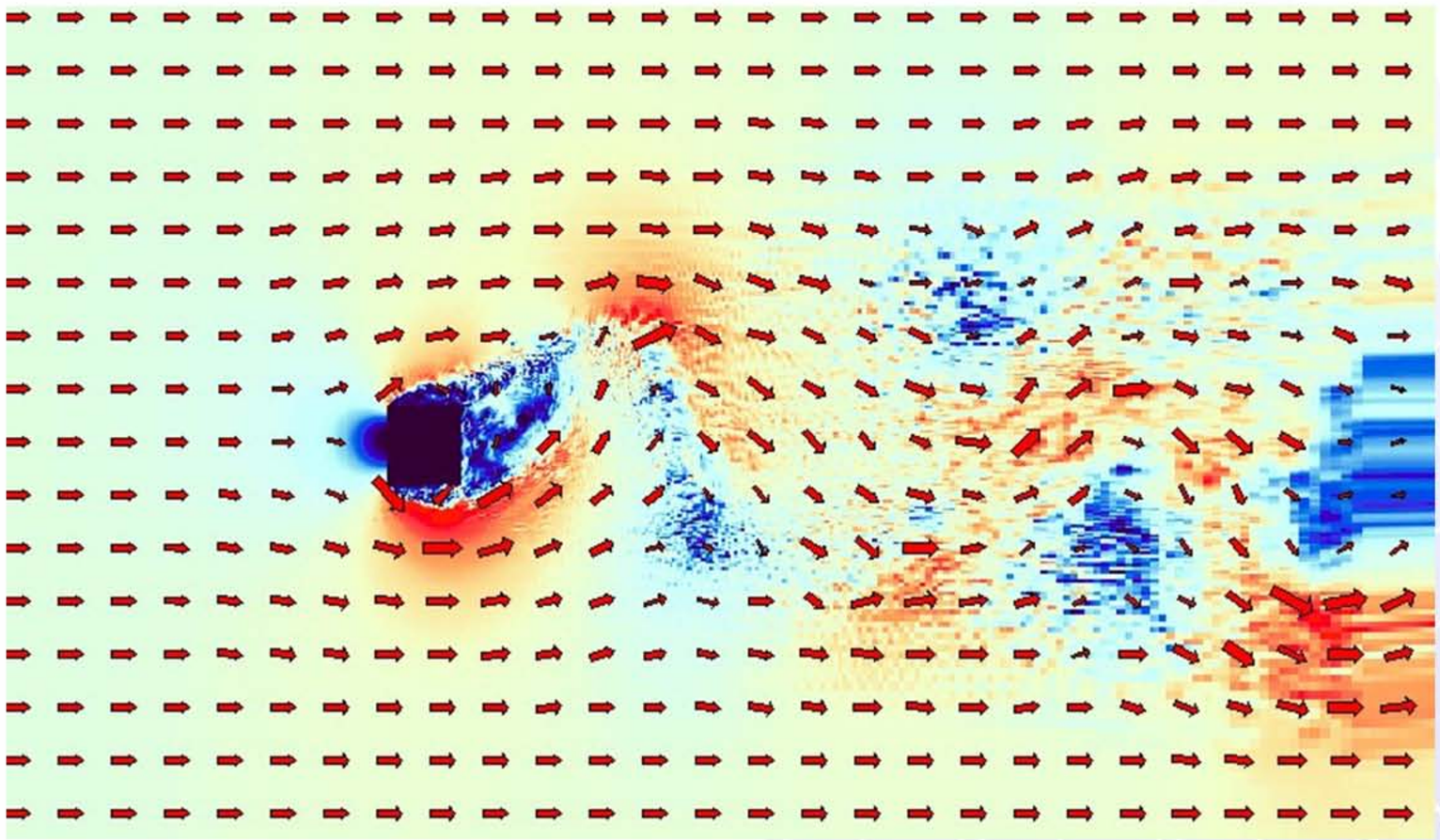
Visualization of 2D Flows

- Color coding:
  - select a „relevant“ scalar parameter (vorticity, speed, pressure, etc.)
  - Apply 2D scalar visualization



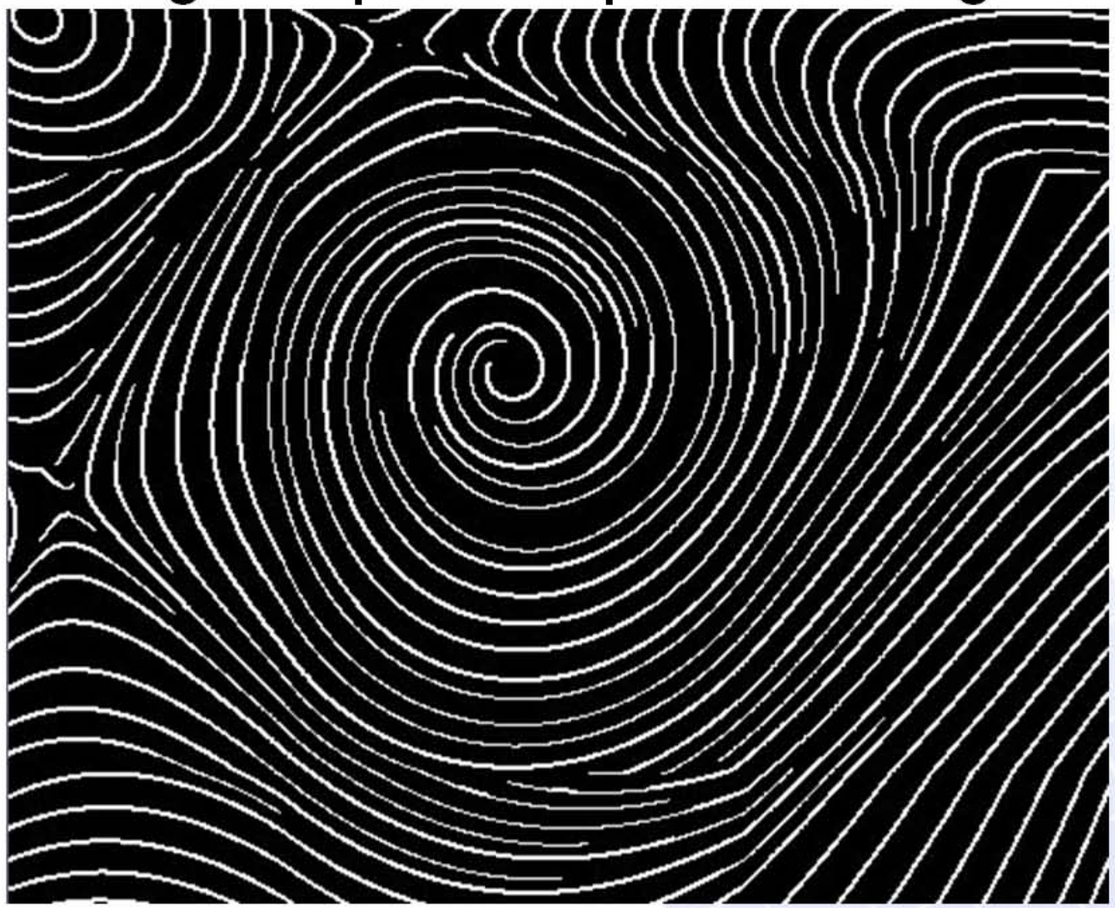
Visualization of 2D Flows

- Arrows:
  - Placement of vector arrows on regular grid
  - Direction and length from flow field



Visualization of 2D Flows

- Line-based techniques using particles
  - Excite particles at starting points
  - Trace/integrate particle paths through flow

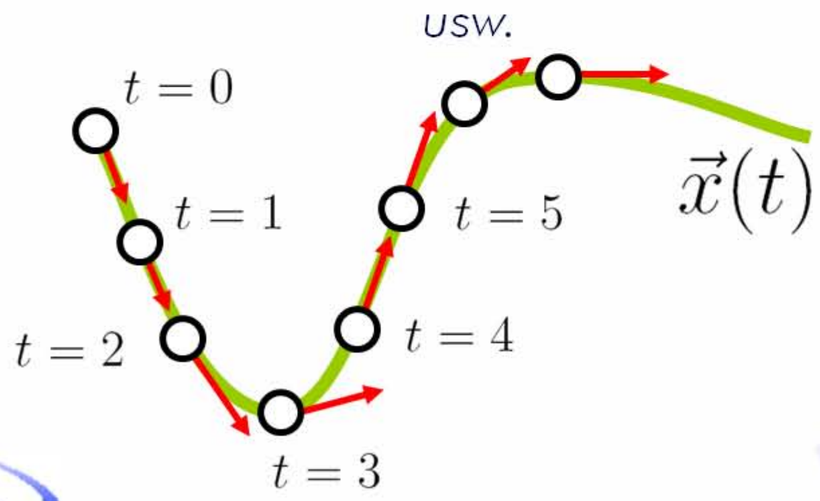


Streamlines

- Given a static vector field
- We search for the trace of a ***particle***, i.e. it's motion through the field

$$\vec{x}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$$

- The lines follows the flow field, i.e. it is always tangential to it



Euler Integration

- Given a static vector field $\vec{v}(\vec{x})$ and a starting position $\vec{x}_0 = \vec{x}(t_0)$
- Goal: a streamline $\vec{x}(t)$

- *Taylor expansion:*

$$\vec{x}(t_0 + \tau) = \vec{x}(t_0) + \tau \frac{\partial \vec{x}}{\partial t}(t_0) + \dots$$

$$\vec{x}(t_0 + \tau) = \vec{x}_0 + \tau \vec{v}(x_0) + \dots$$

Euler Integration

- Given a static vector field $\vec{v}(\vec{x})$ and a starting position $\vec{x}_0 = \vec{x}(t_0)$
- Goal: a streamline $\vec{x}(t)$

- *Taylor expansion:*

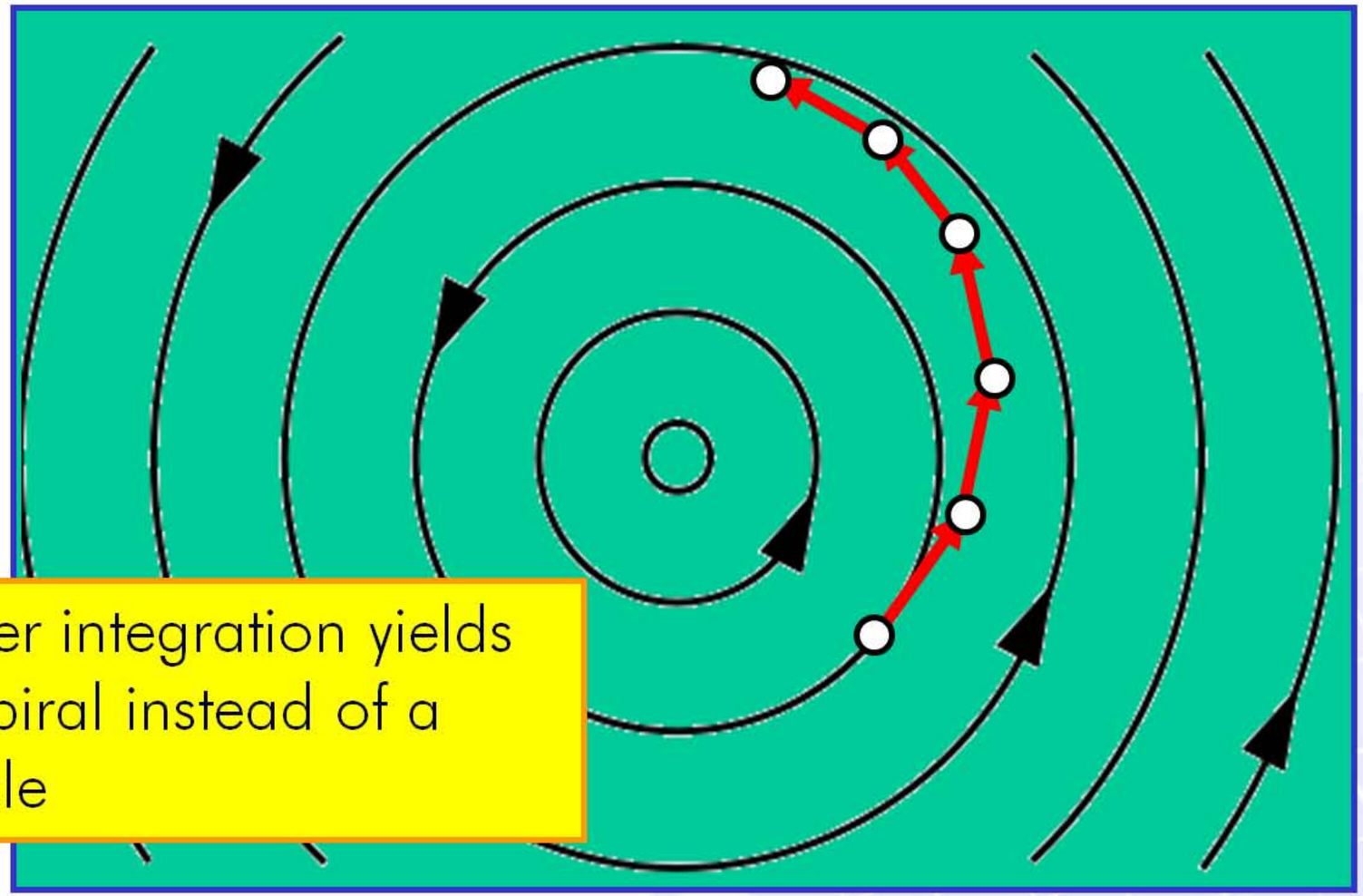
$$\vec{x}(t_0 + \tau) = \vec{x}(t_0) + \tau \frac{\partial \vec{x}}{\partial t}(t_0) + \dots$$

$$\vec{x}(t_0 + \tau) = \vec{x}_0 + \tau \vec{v}(x_0) + \dots$$

$$\vec{x}_{i+1} \approx \vec{x}_i + \tau \vec{v}(x_i)$$

Example Euler Integration

Circular vector field $\vec{v}(x,y) = \begin{pmatrix} -y \\ x \end{pmatrix}$



Euler integration yields a spiral instead of a circle

Euler Integration

- The error can be controlled by the step-size
  - Small step-size yields less error
  - But: requires more computational effort
- Alternatives are higher order integration schemes required, e.g.
  - Heun
  - Kunge-Kutta 2nd or 4th order

Time-variant Data

- ***Until now:*** Particle tracing in static (time is not varying) vector field:

$$\frac{\partial \vec{x}(t)}{\partial t} = \vec{v}(\vec{x}(t))$$

- ***Now:*** Particle tracing dynamic (time is varying) vector field:

$$\frac{\partial \vec{x}(t)}{\partial t} = \vec{v}(\vec{x}(t), t)$$

What are the consequnces?

Time needs to be taken into account as additional Parameter, e.g. Euler step $t_{i+1} = t_i + \tau$

$$\vec{x}(t_{i+1}) = \vec{x}(t_i) + \tau\, \vec{v}\Big(\vec{x}(t_i), t_i + \tau\Big)$$

Particle Tracing for Time-Variant Data

- **Pathlines:** Particle follows the time-varying flow field

$$\frac{\partial \vec{x}(t)}{\partial t} = \vec{v}(\vec{x}(t), t) \qquad \vec{x}_0 = \vec{x}(t_0)$$

  Associated experiment: „Inject at time t_0 a single drop of dye into the flow and make a photo with extremely long exposure time “

- **Streamlines:** Compute particle trace for a fixed point in time (results in a static flow field)

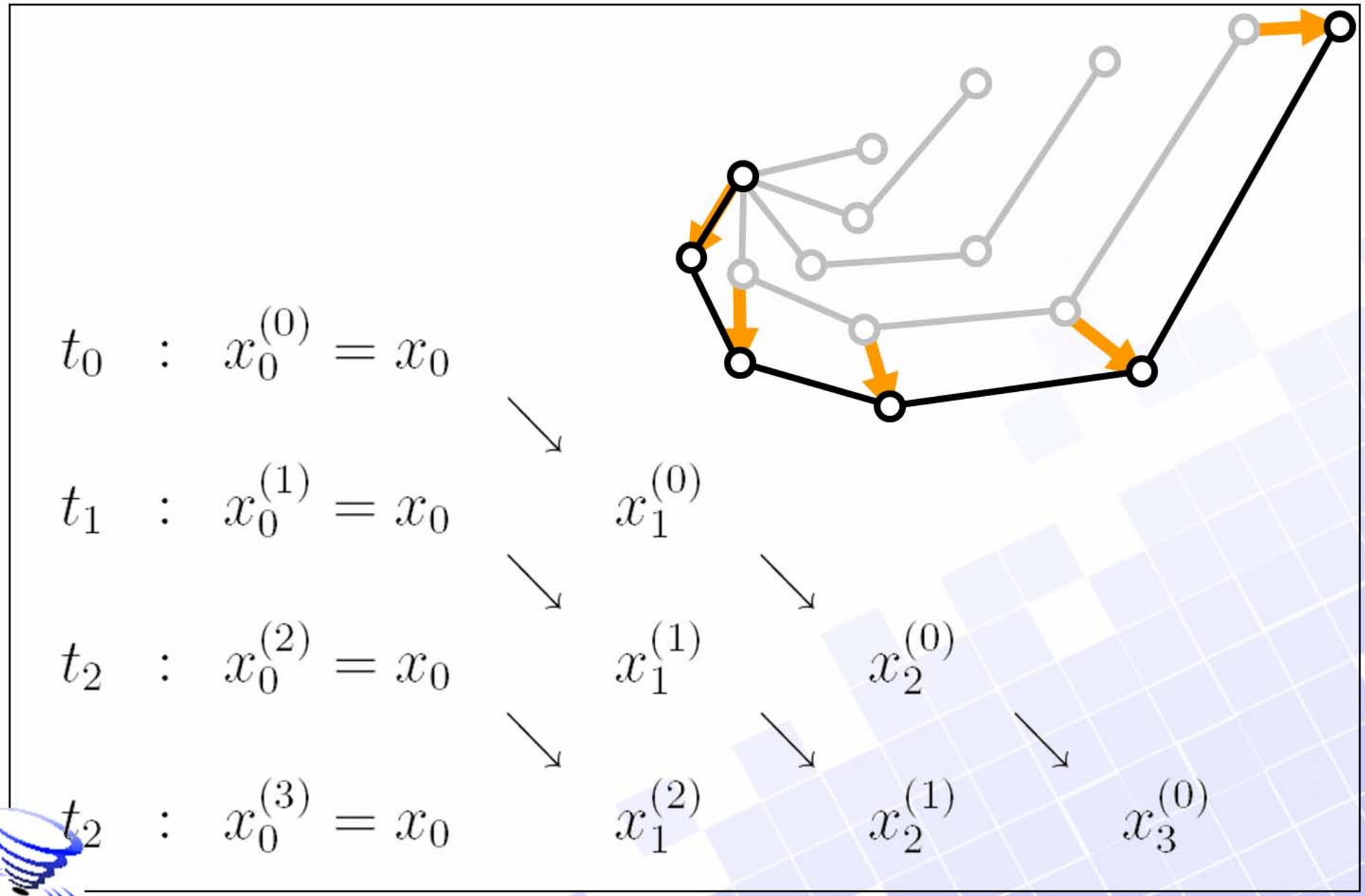
$$\frac{\partial \vec{x}(t)}{\partial t} = \vec{v}(\vec{x}(t), T) \qquad T = const.$$

- **Streakline:** Trace several particles, that have been started at different points in time

$$\frac{\partial \vec{x}(t)}{\partial t} = \vec{v}(\vec{x}(t), t) \quad \vec{x}_0 \text{ constant}, \quad t_0 \text{ variabel}$$

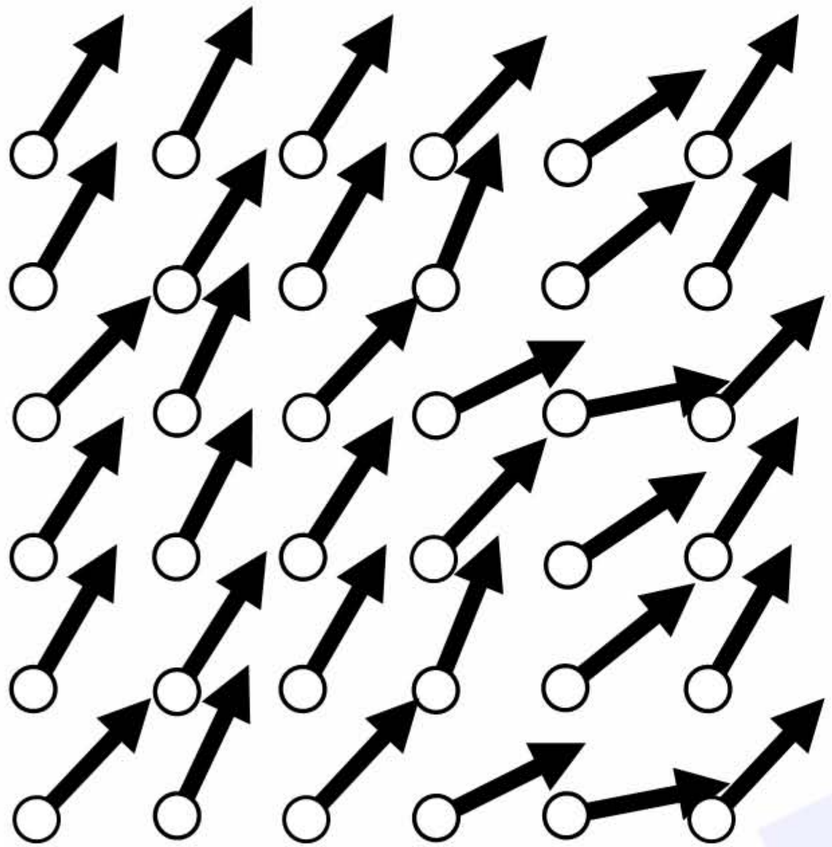
  Associated experiment: „Continuously inject drops of dye at starting position into the flow“

Streaklines



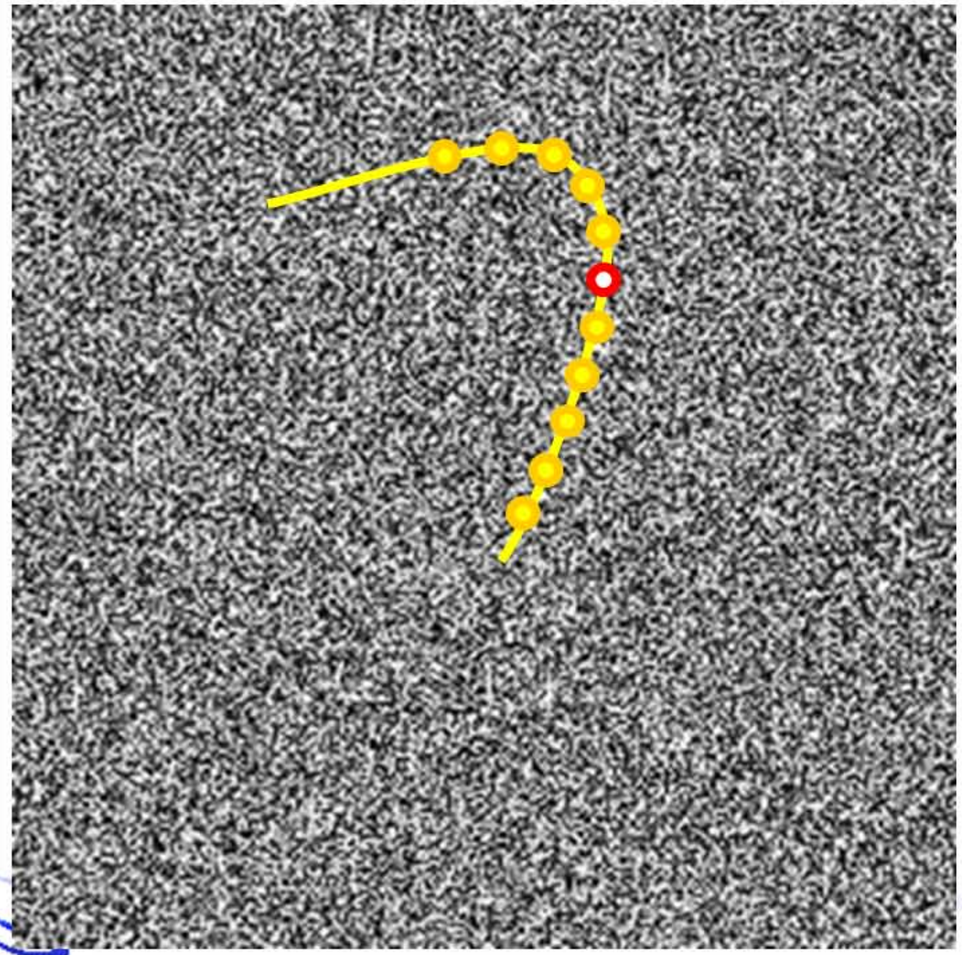
Line Integral Convolution (LIC)

- Goal: Visualize flow globally, i.e. show flow at all points in the plane at the same time



Line Integral Convolution (LIC)

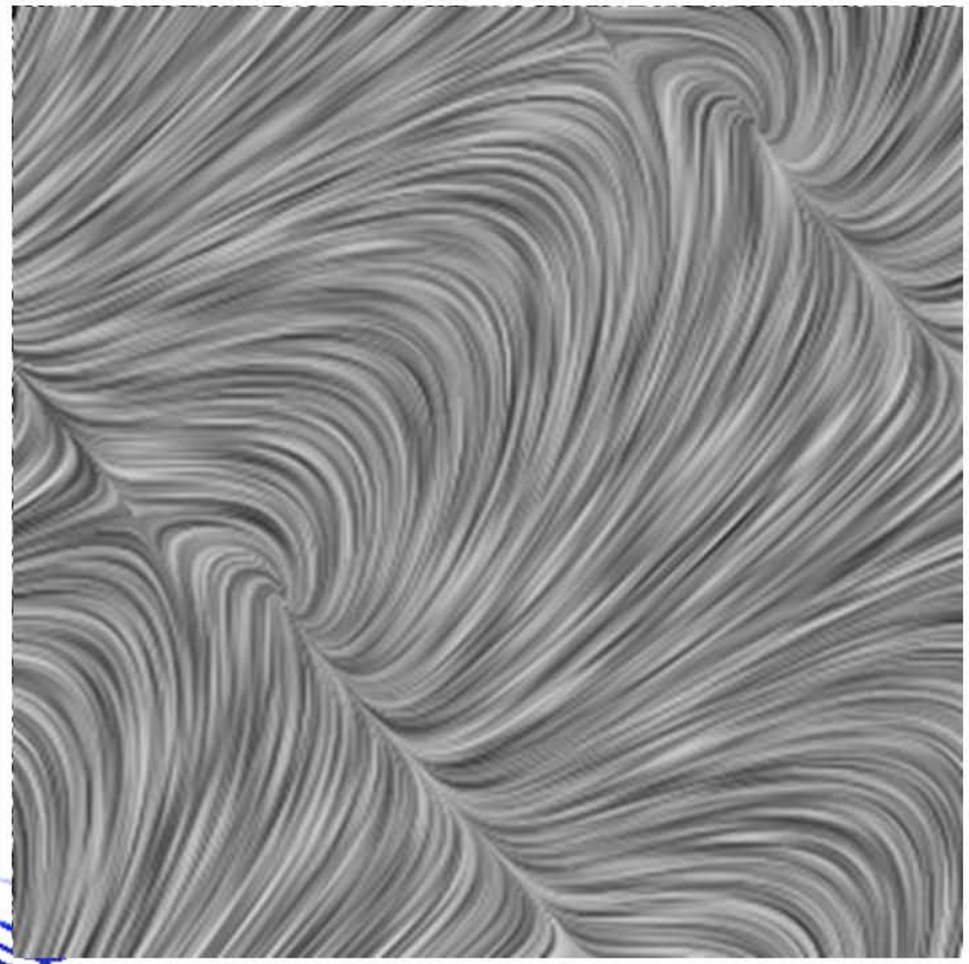
- Goal: Visualize flow globally, i.e. show flow at all points in the plane at the same time



- Foreach pixel: Compute a (short) streamline (back- and foreward)
- Compute the weighted sum of the noise intensities along line (filtering, convolution)

Line Integral Convolution (LIC)

- Goal: Visualize flow globally, i.e. show flow at all points in the plane at the same time



- Foreach pixel: Compute a (short) streamline (back- and foreward)
- Compute the weighted sum of the noise intensities along line (filtering, convolution)

Visualization of 3D Data

- General considerations
  - Significantly more data to be handled, still we want to have realtime applications
  - Almost all 2D approaches can be applied to 3D, beside
    - Scalar data: Surface plots (color plots require specific handling → volume visualization)
    - Vector fields: space-filling line integral convolution makes little sense
- Challenges:
  - More difficult user orientation in 3D
  - Hidden objects (= hidden information!)
    → proper selection of relevant information

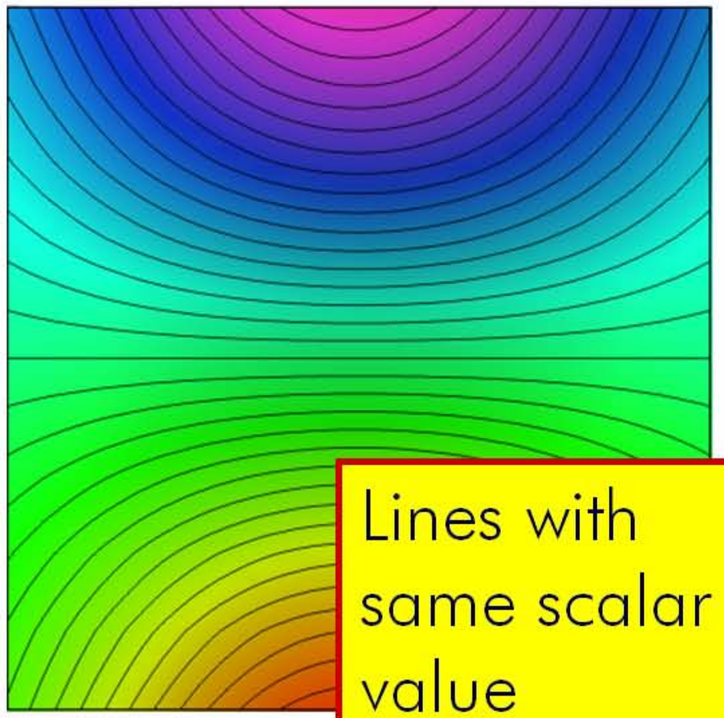
3D Scalar Data: Volume Visualization

- Volume data = 3D scalar data
- Origin: Mainly imaging technology like
  - X-Ray
  - Computer tomography (CT)
  - Magnet resonance tomography (MR)
- Main approaches
  - **Indirect techniques:** Compute a “specific” surface from the volume data and visualize it
  - Direct techniques: Interpret scalar field as semi-transparent to make hidden regions (partially) visible

Isosurfaces (indirect technique)

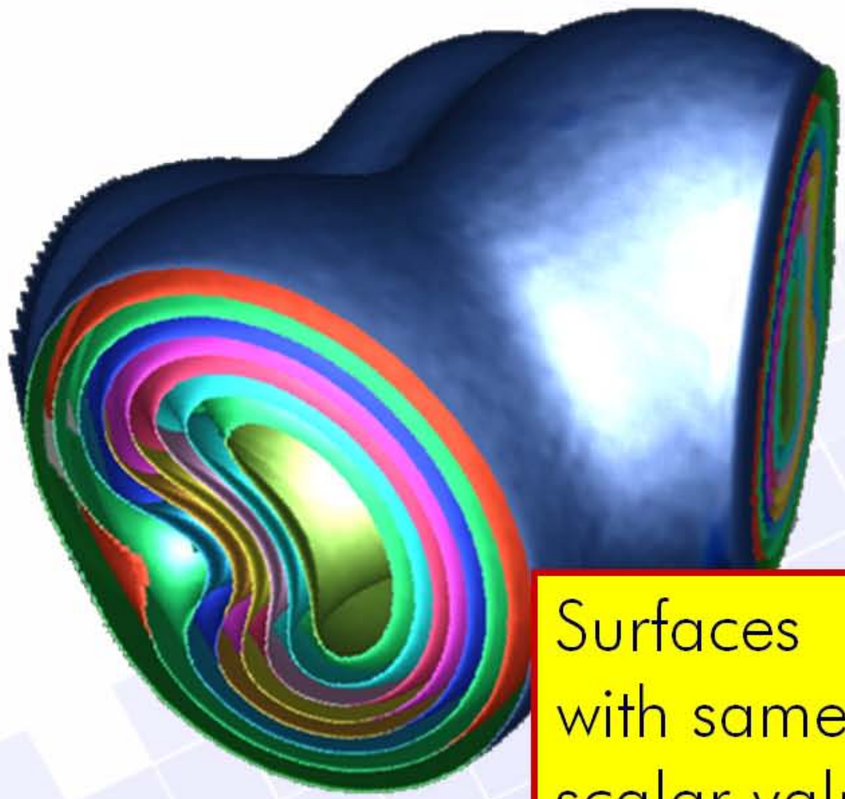
- Analogous to iso-lines:
  select points with the same iso value

2D: Iso-line



Lines with same scalar value

3D: Iso-surface



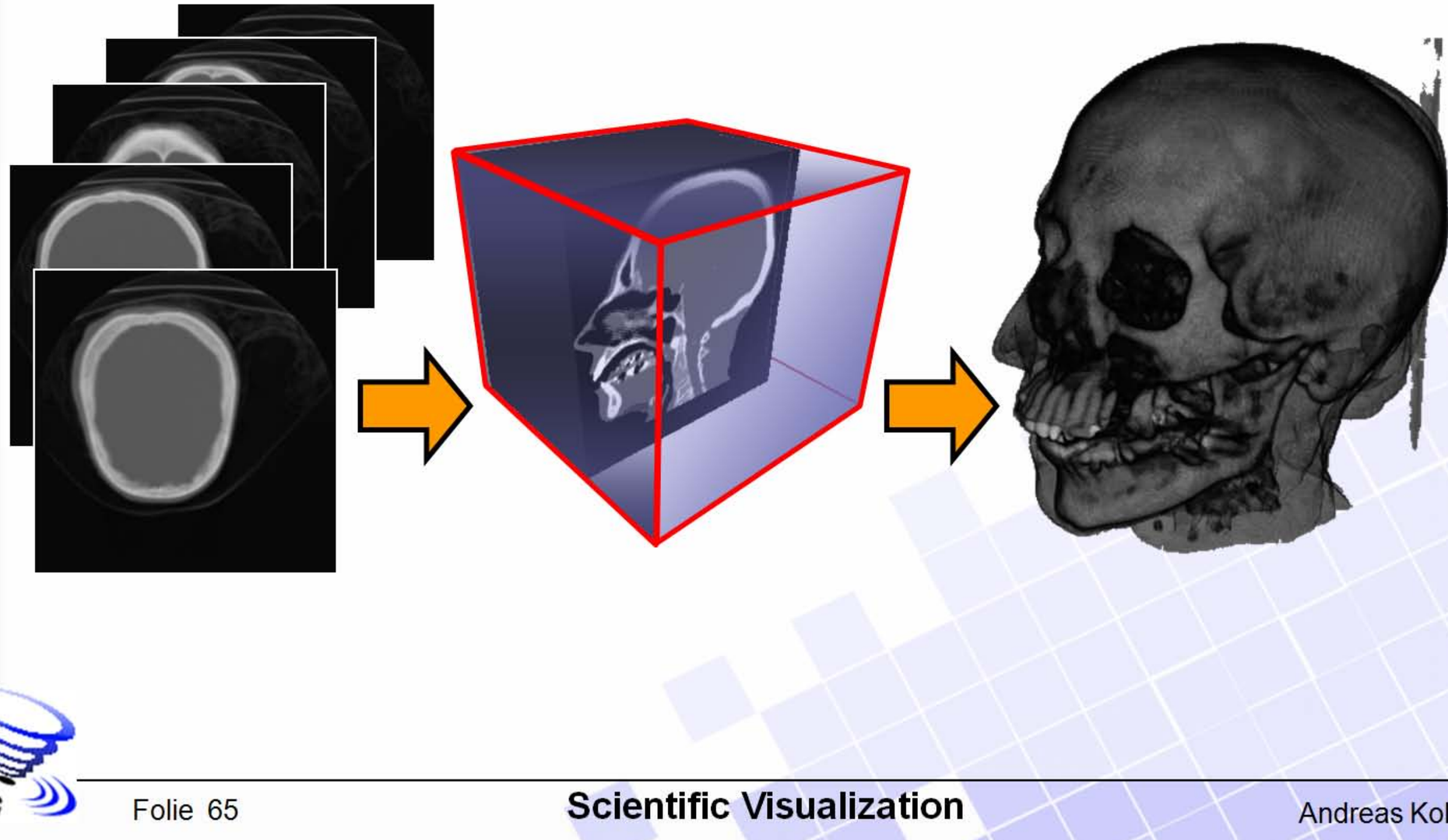
Surfaces with same scalar value

Direct Volume Visualization

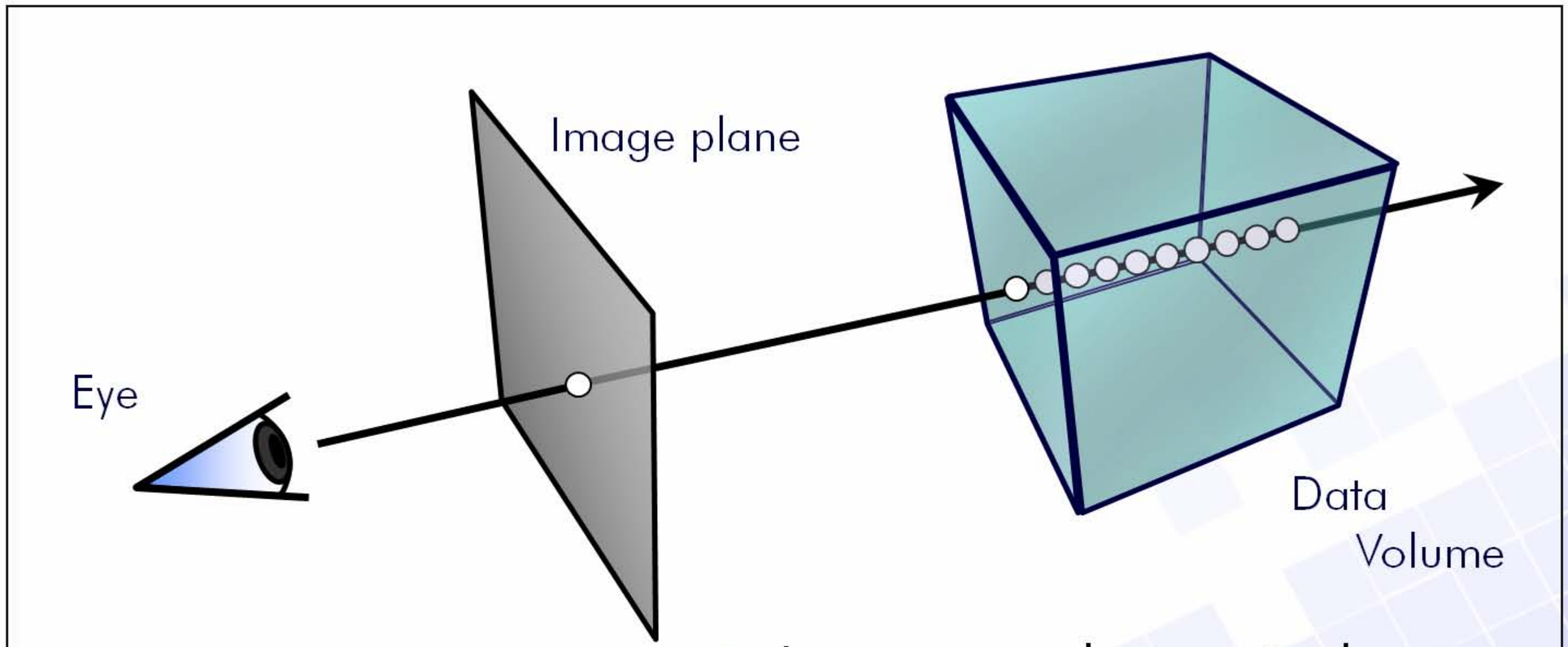
Data Set

3D Alignment

Classification



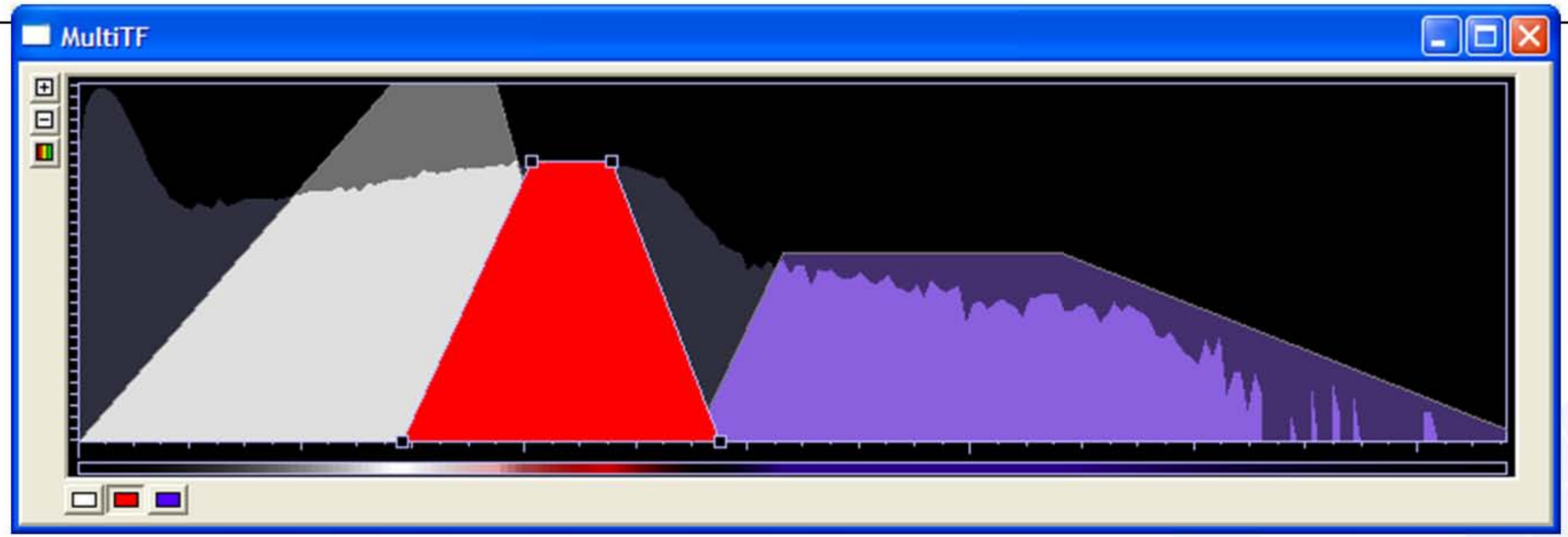
Ray Casting (Concept)

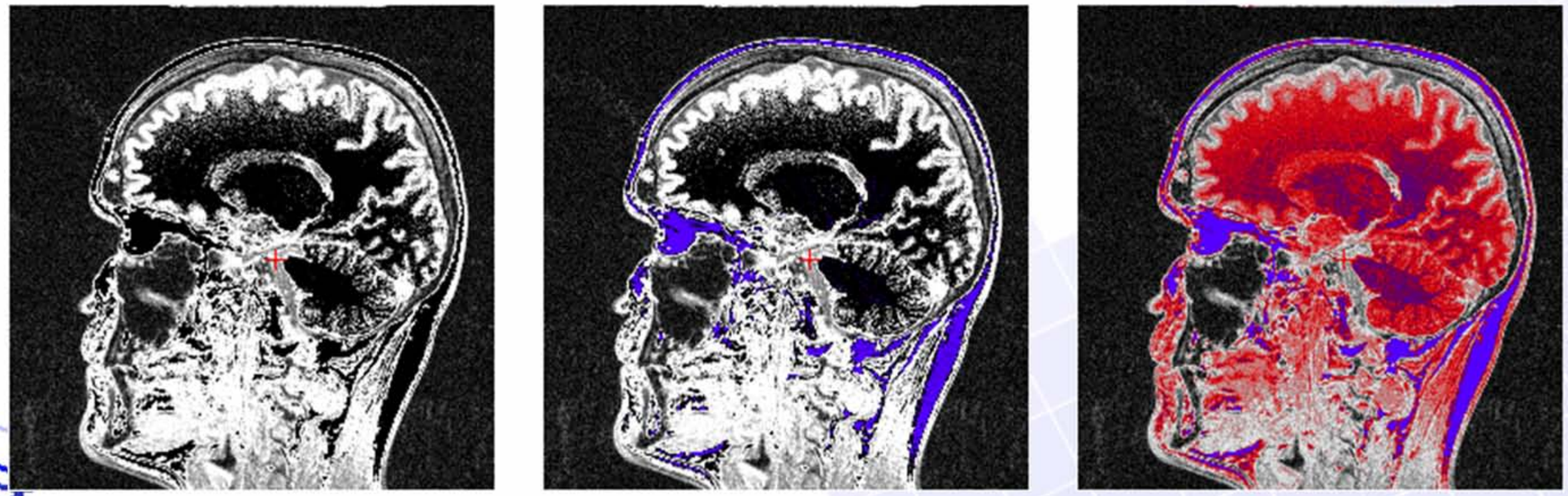


- Integrate along pixel ray
- Sample scalar values along ray & assign transparency

⇨ Transfer function controls transparency assignment

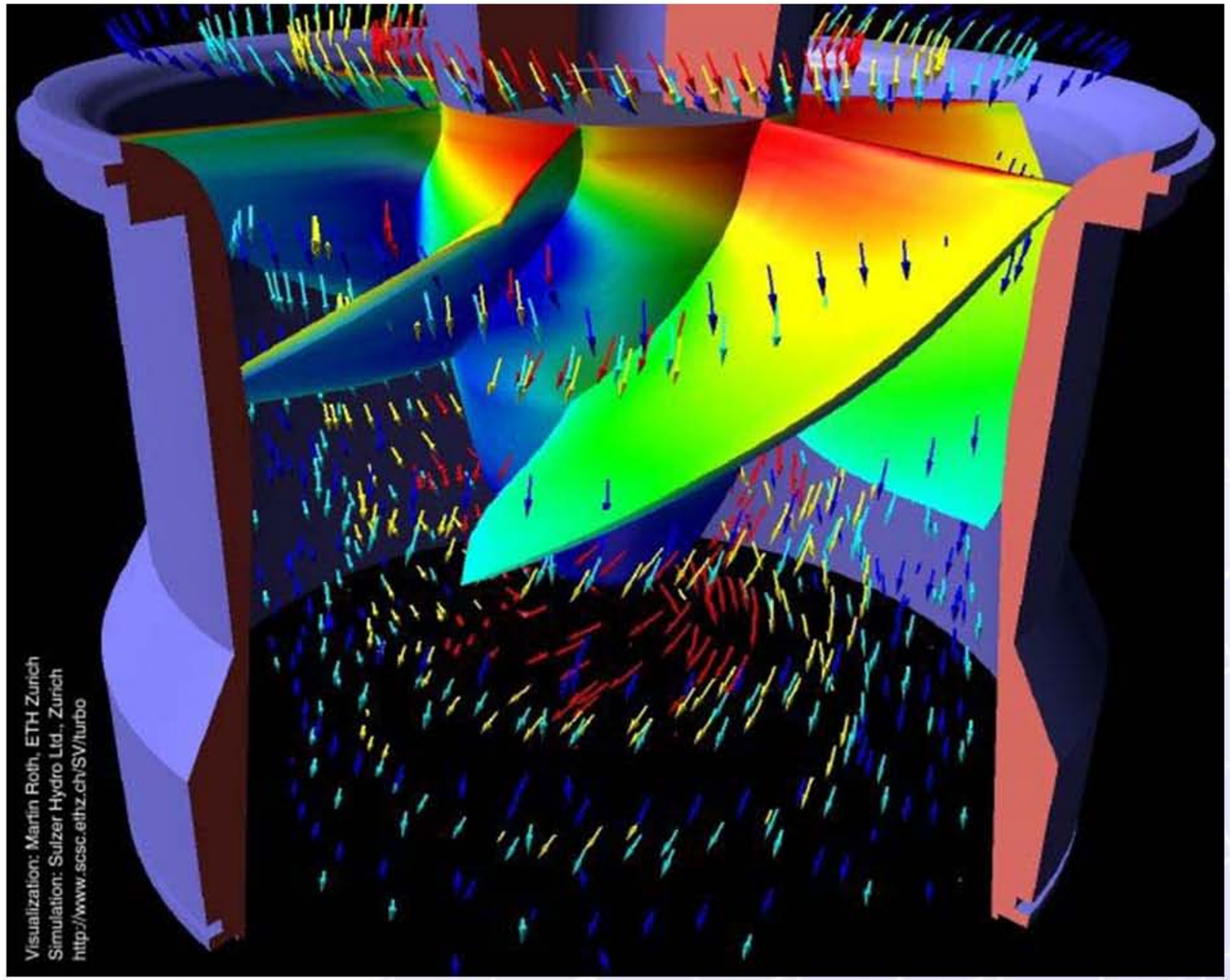
Transfer Functions





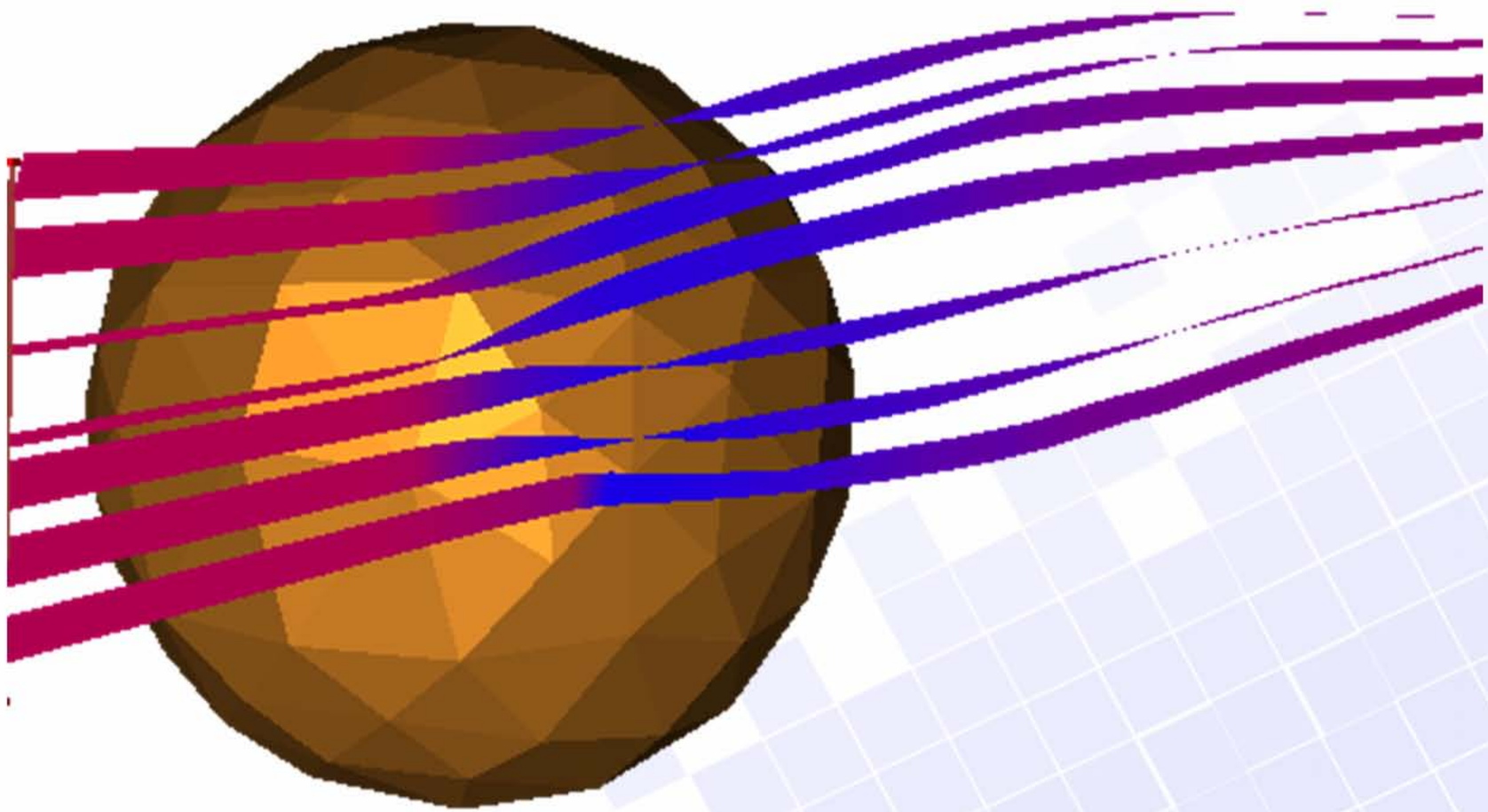
3D Flow Visualization

- 3D critical points: More different cases
- Example:
  Arrow plots



3D Flow Visualization

- 2D flow primitives, e.g. stream ribbons
  - Visualize rotation along path



Further Issues

- Efficient computation for interactive applications
  → Programmable Graphics Hardware (GPU)
- Adequate user guidance
  - Avoid artifacts; they may lead to misinterpretation of data
  - Allow user control on domain-level, not on the technical scale