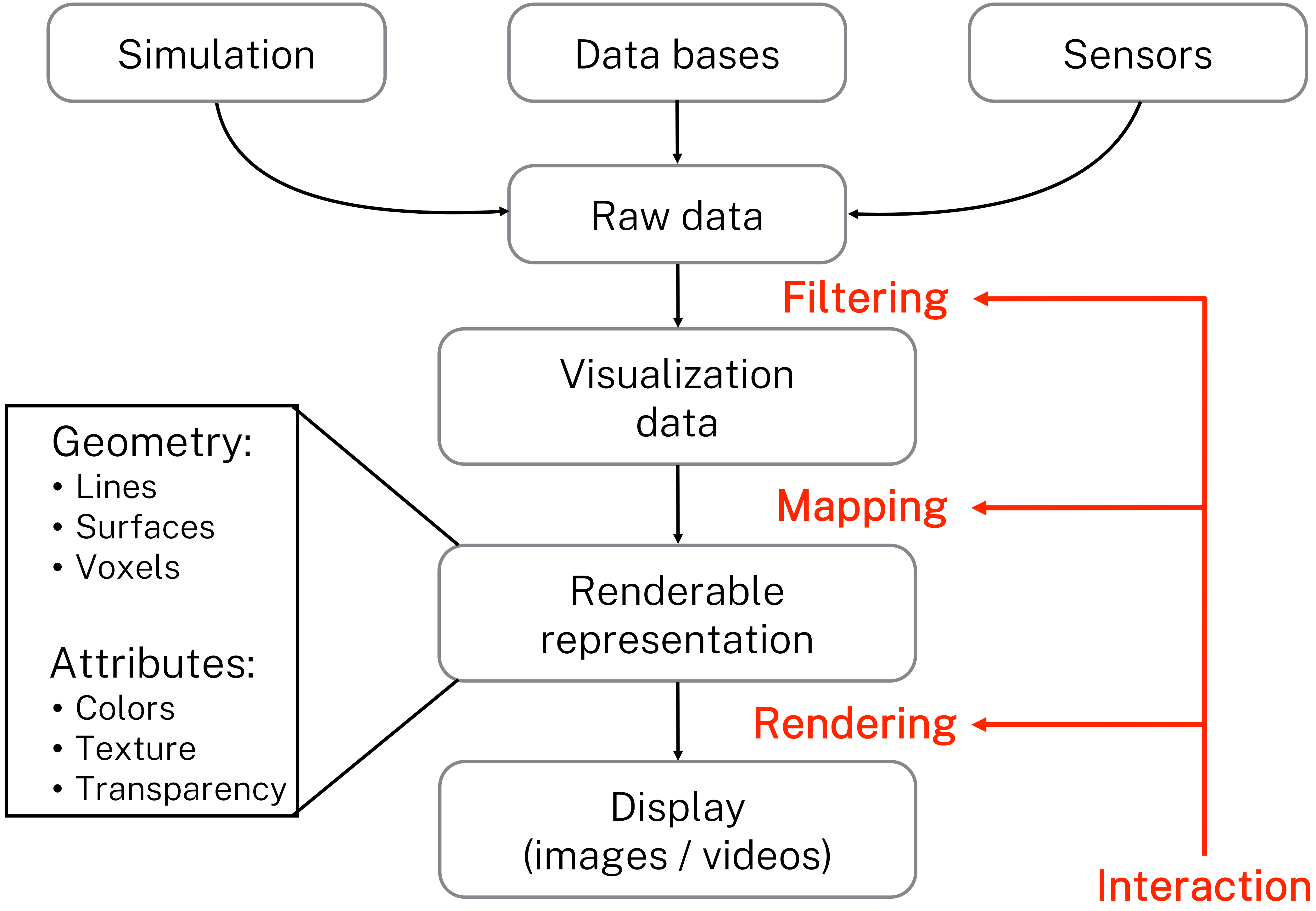
3. Fundamental Concepts

3.1 Visualization Pipeline

Visualization Pipeline

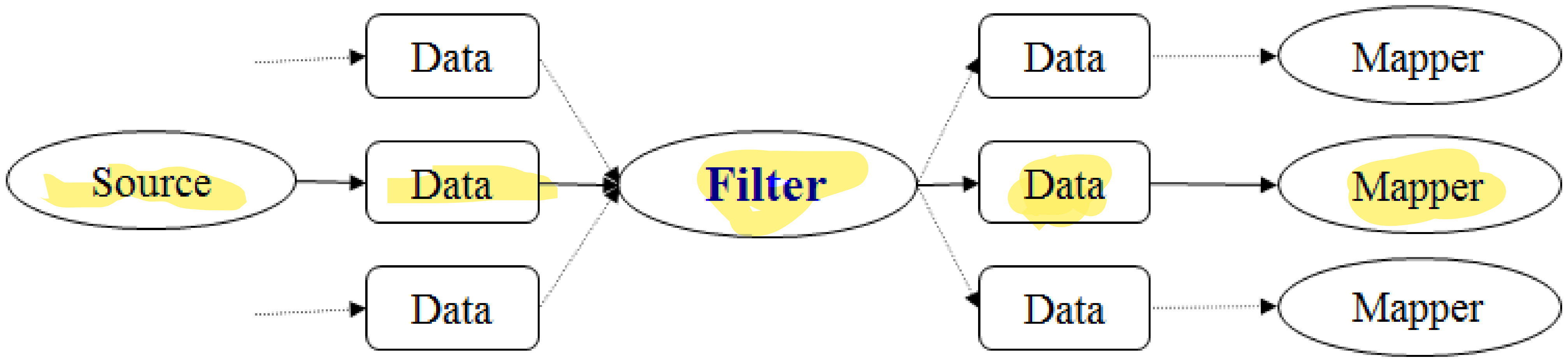


Data sources

- **Real-world**
  - Measurements and observation
- **Theoretical world**
  - Mathematical and technical models
- **Artificial world**
  - Designed data
- Traditional presentation techniques
  - Insufficient for increasing amount of data
  - Data from any source with almost arbitrary size
  - New developments required for efficient visualization of large-scale data sets and new data types

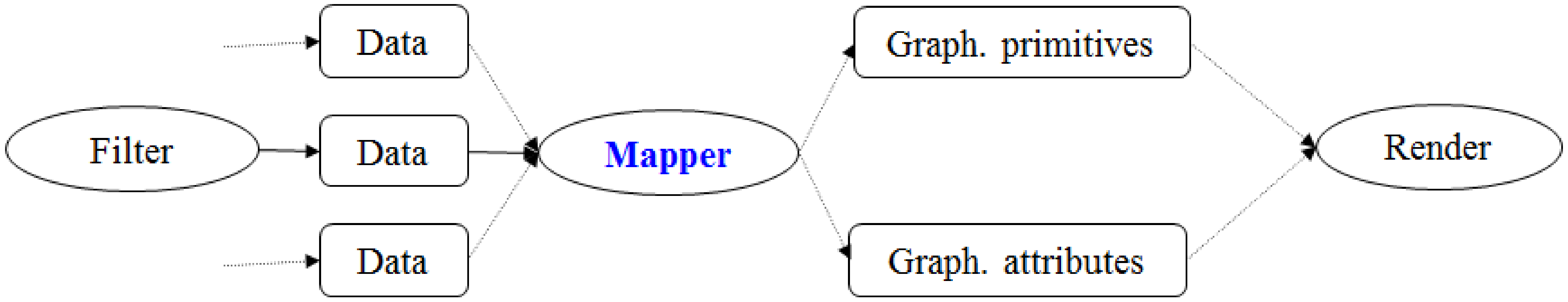
Visualization Pipeline

- **Filter**: transform data into data
  - Data format conversion
  - Clipping, cropping, de-noising
  - Slicing, resampling
  - Interpolation, approximation
  - Classification, segmentation



Visualization Pipeline

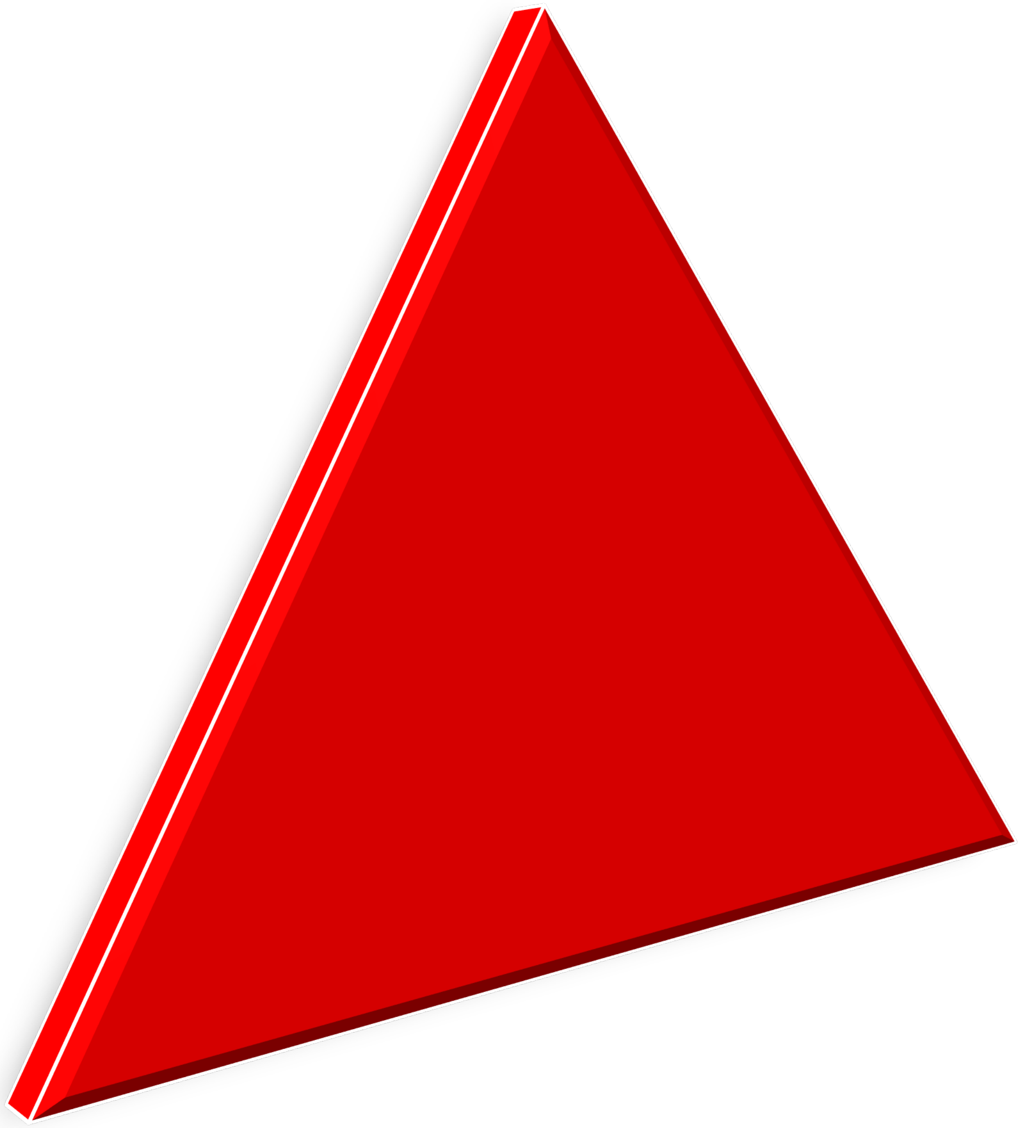
- **Mapper**: transform data into graphical primitives
  - Scalar field to iso-surface
  - Vector field to glyphs
  - …



ohm

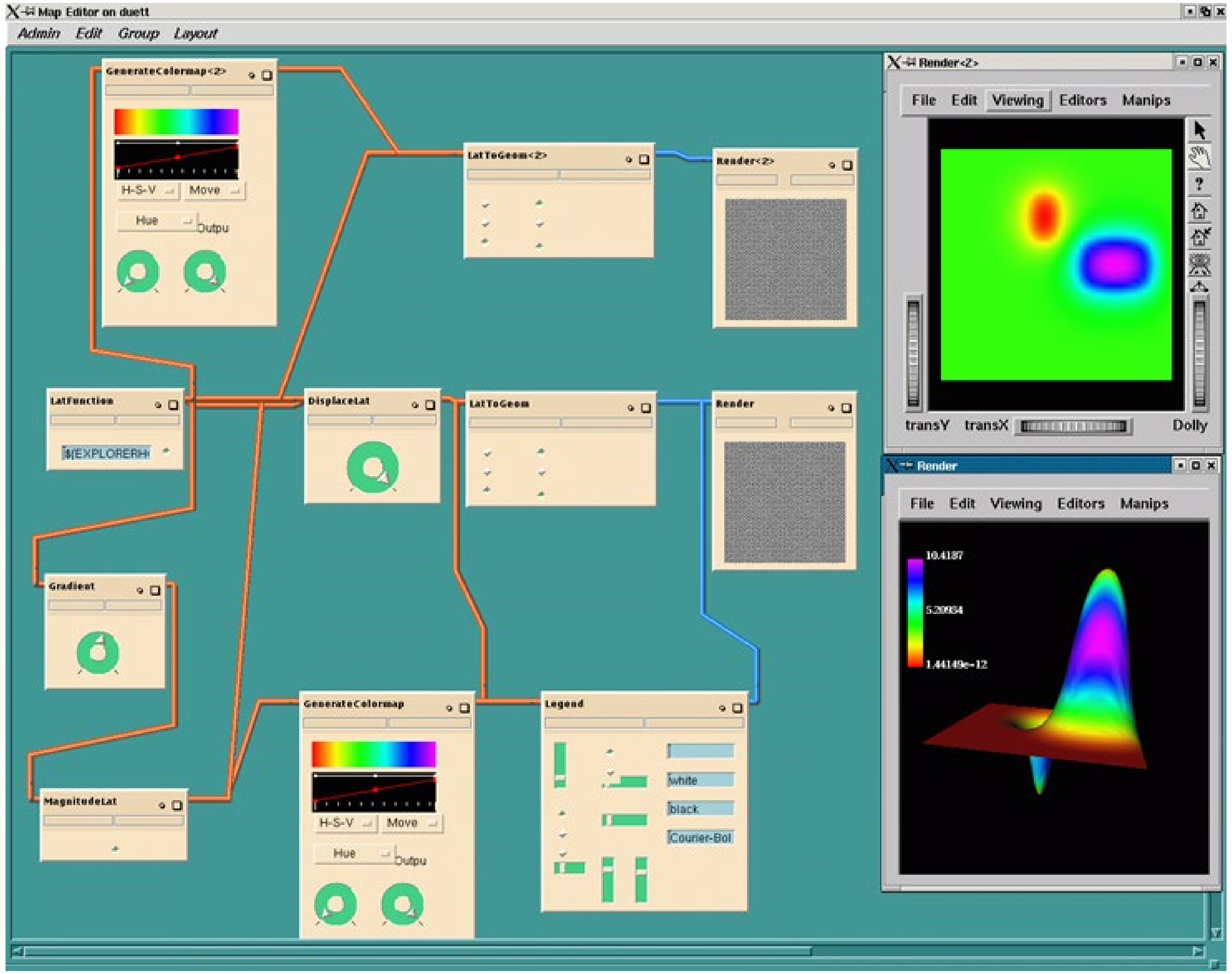
Visualization Pipeline

- Rendering
  - Geometry / images / volumes
    - Images / Video
  - “Realism” (e.g. shadows, ...)
  - Lighting, shading
  - Texturing



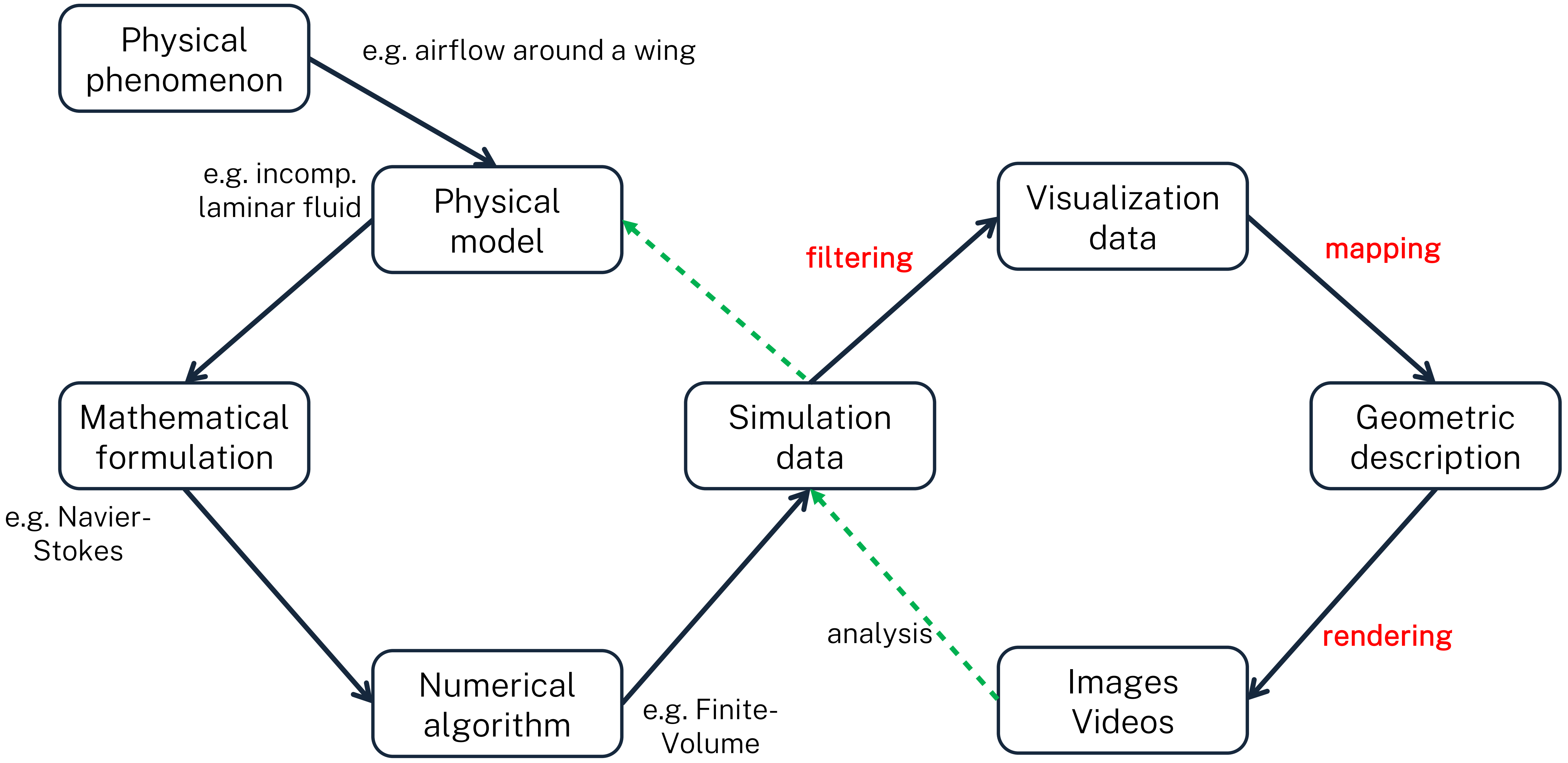
ohm

Visualization Pipeline



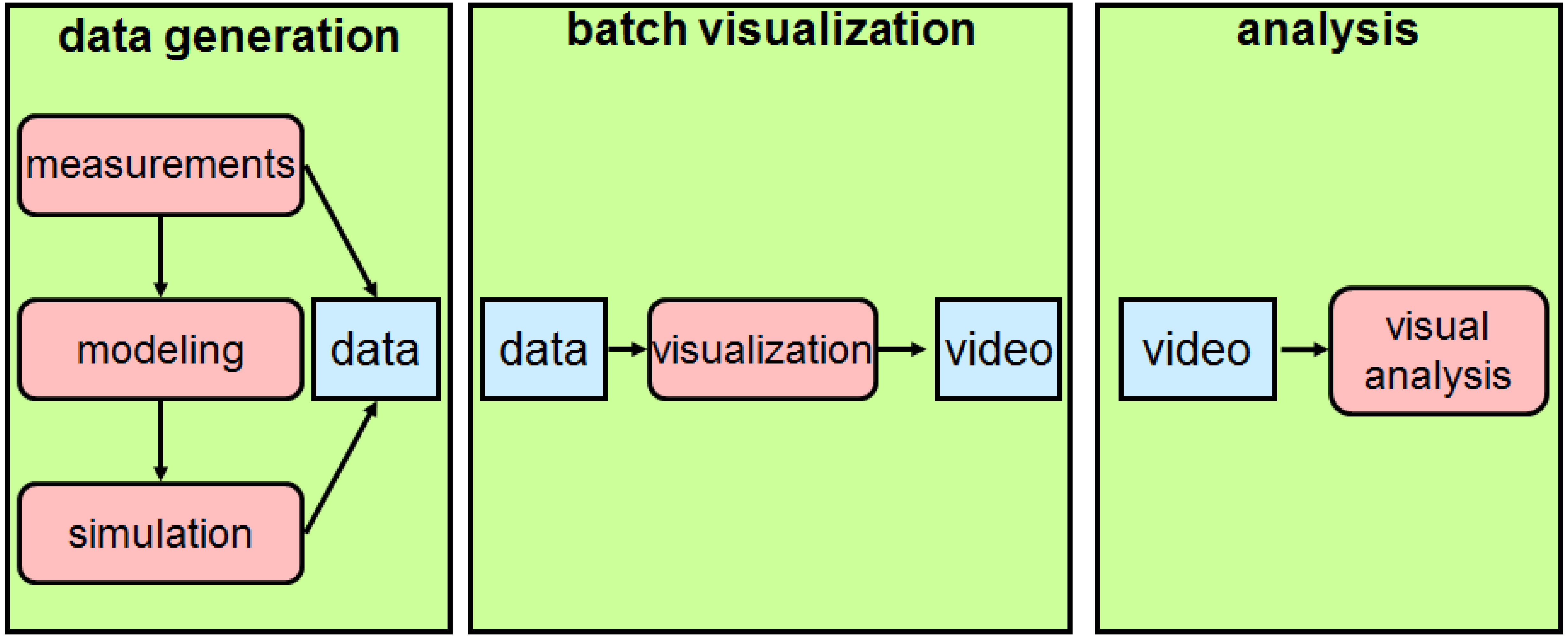
Visualization Pipeline

Visualization - Simulation cycle (example)



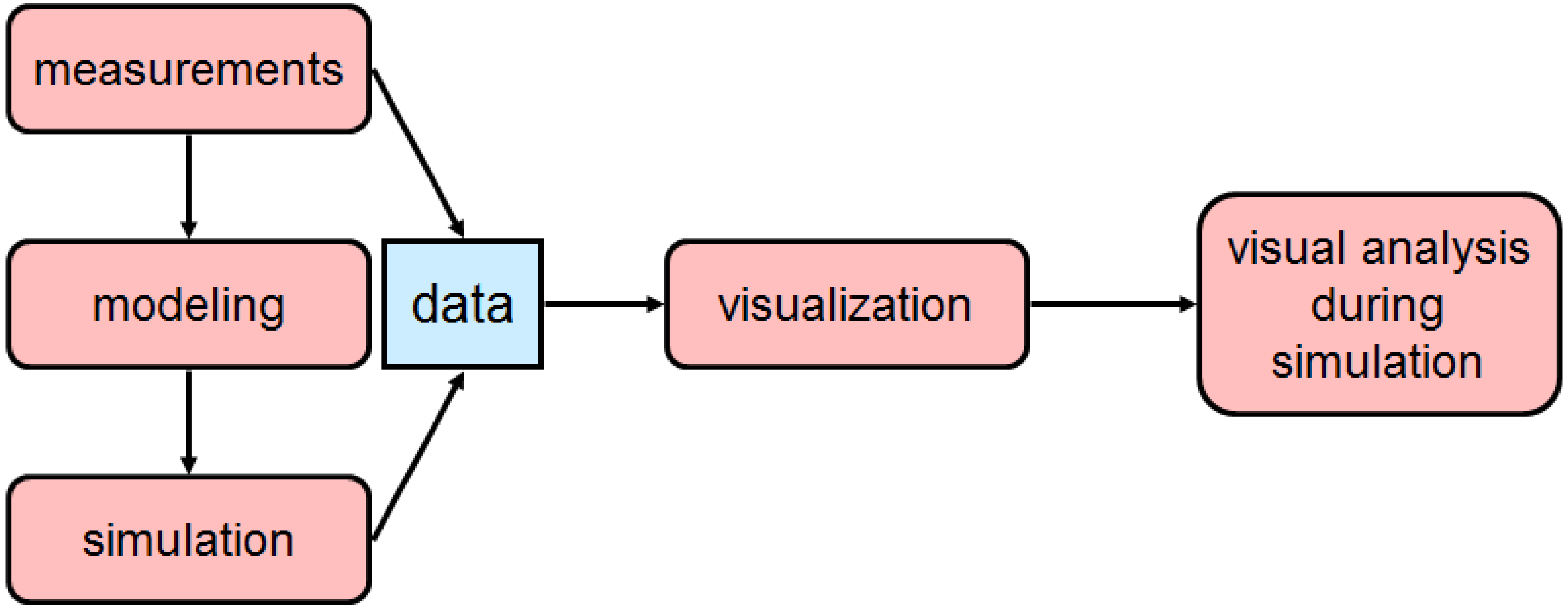
Visualization Pipeline

Example scenarios: Video/movie mode — *offline*, no interaction



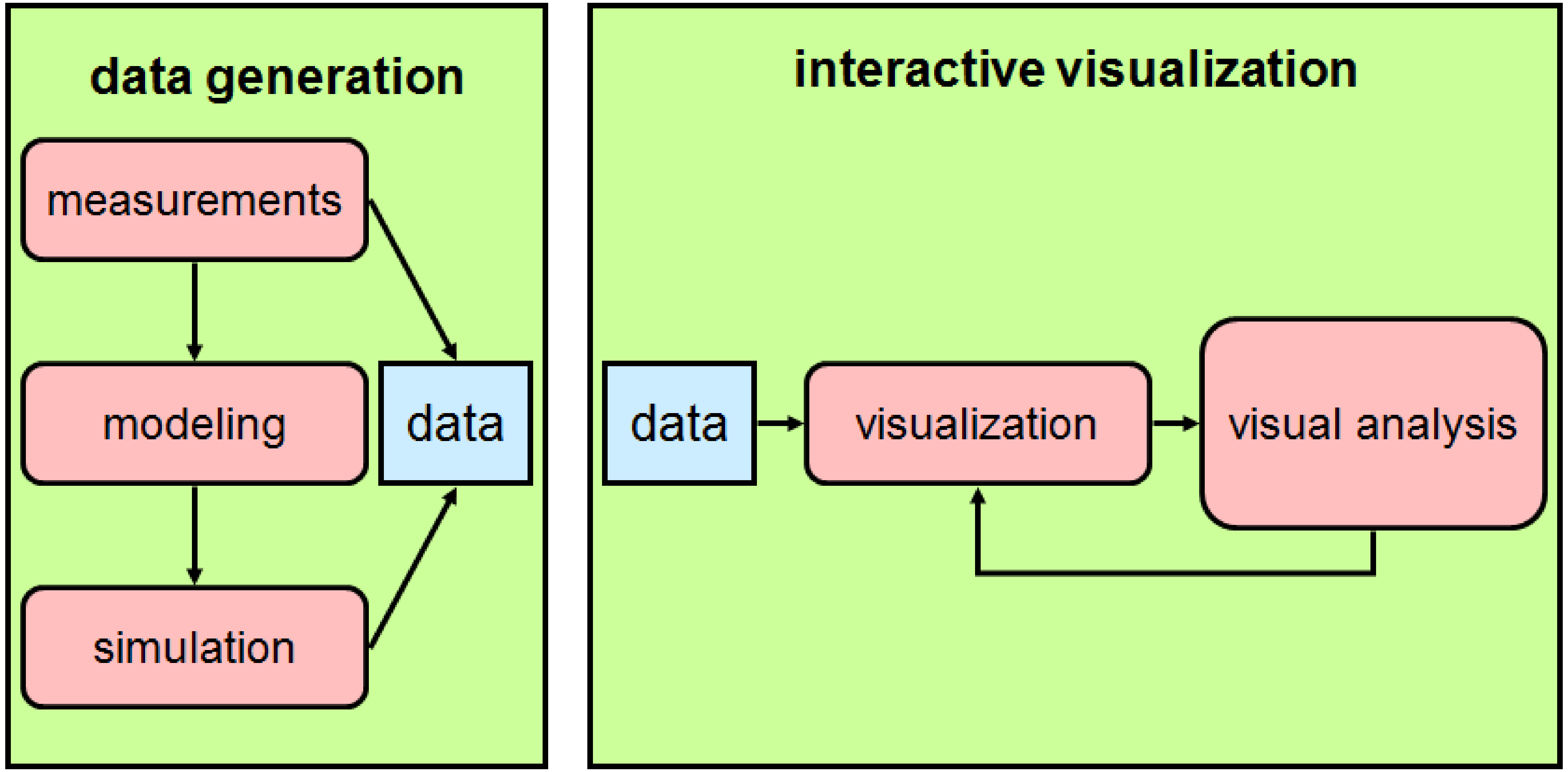
Visualization Pipeline

Example scenarios: Tracking — *online*, no interaction



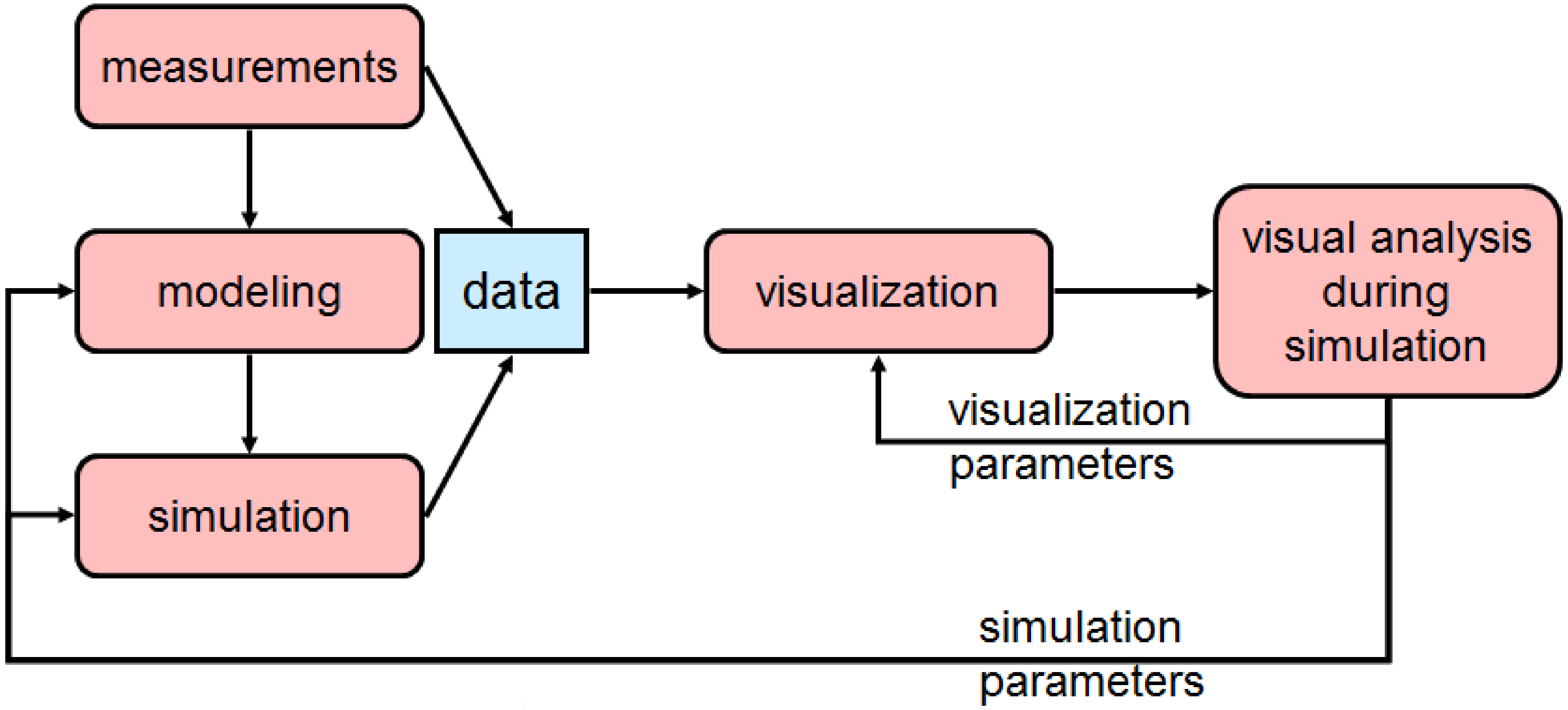
Visualization Pipeline

Example scenarios: Interactive post processing — *offline*



Visualization Pipeline

Example scenarios: Interactive steering / control



3.2 Sources of Error

Sources of Error

- Data acquisition
  - Sampling
    - Sufficient (spatial) sampling of the data to get what we need?
  - Quantization
    - Conversion of "real" data to representation with enough precision to discriminate the relevant features?
- Filtering
  - Are we retaining / removing the "important / non-relevant" structures?
  - Frequency / spatial domain filtering: noise, clipping and cropping
- Selecting the "right" variable
  - Does this variable reflect the interesting features?
  - Does this variable allow for a "critical point" analysis?

Sources of Error

- Functional model for resampling
  - Introduced information by interpolation and approximation?
- Mapping
  - Appropriate choice of graphical primitives?
  - Think of some real-world analogue (metaphor)
- Rendering
  - Need for interactive rendering
    - Often determines the chosen abstraction level
  - Consider limitations of the underlying display technology
    - Data, color, quantization
  - Carefully add "realism"
    - The most realistic image is not necessarily the most informative one

3.3 Data Domain

Data Domain

- Discrete representations
  - Target objects: **continuous**
  - Given data: **discrete** in space and/or time
  - Discrete structures
    - Consist of **samples** → generate **grids/meshes** consisting of **cells**

Data Domain

- Primitives in multi-dimensions

Dimension	Cell	Mesh
1D	Line (edge)	Polyline (Polygon)
2D	Triangle, quadrilateral (e.g. rectangle)	2D mesh
3D	Tetrahedron, hexahedron (e.g. cube), prism, pyramid,...	3D mesh

Data Domain

Classification of visualization techniques according to

- **Dimension** of the domain
  - 0D (means unstructured points)
  - 1D, 2D, 3D, nD

- **Type** of the data
  - Scalar, vector, tensor, multi modal

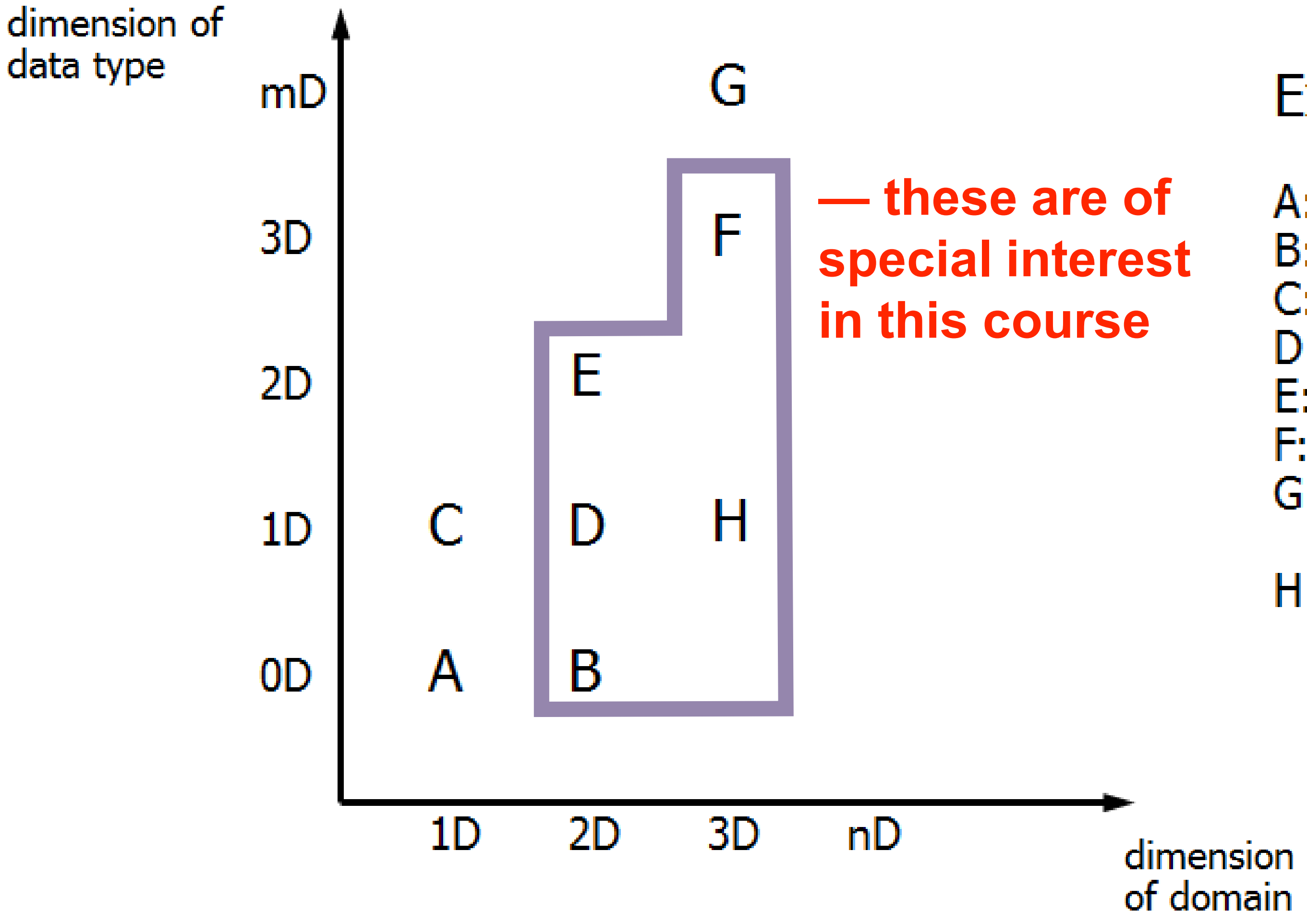
- **Grid** type
  - Uniform cartesian, structured, curvilinear
  - Unstructured, point sets (scattered data)

ohm

Data Domain

Classification of visualization based on

- Dimension of the problem domain (independent parameters)
- Dimension of the data type (dependent parameters)



Examples:

A: gas station along a road
B: map of cholera in London
C: temperature along a rod
D: height field of a continent
E: 2D air flow
F: 3D air flow in the atmosphere
G: stress tensor in a mechanical part
H: ozone concentration in the atmosphere

ohm

Data Domain

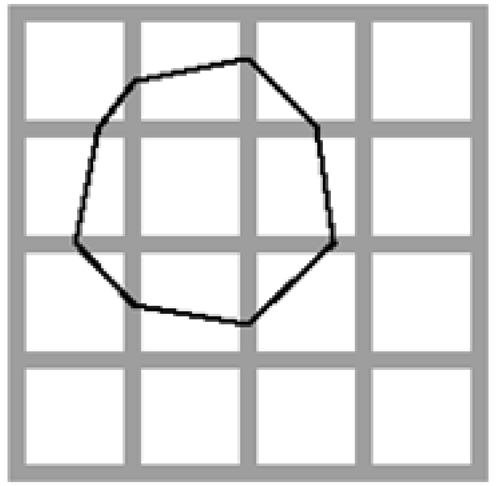
Classification of visualization based on mapping

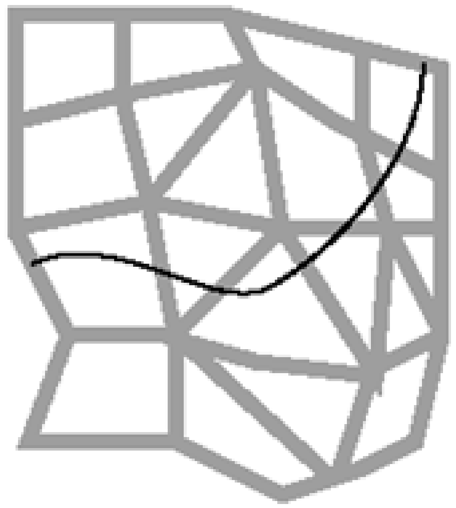
Domain type	scalar	vector	tensor, multivariate
3D	Volume rendering, isosurfaces	Stream lines, glyphs, topology	Glyphs, icons
2D	Height fields, color coding	Arrow plots, LIC, path lines	Attributes, symbols
1D	Function plots	N.A.	N.A.

Data type

Data Domain

Different data structures → different algorithms





Scalar fields cartesian	Vector fields unstructured	Trees, graphs Tables
↓	↓	↓
Medical / Volume Visualization	Flow Visualization	Information Visualization

ohm

Data Domain

Composition

- (Geometric) shape of the domain
  - Determined by the positions of sample points
- Domain characterized by
  - Dimension
  - Influence
  - Structure wie hängen die Punkte zusammen

- **Influence** of data points
  - *Values at sample points influence the data distribution in a certain region around these samples*
  - To reconstruct the data at *arbitrary* points within the domain, the distribution of all samples must be calculated (interpolation)

Data Domain

Influence types

- **Point** influence
  - Only influence on the point itself
- **Local** influence
  - Only influence within a certain region around the point
    - Voronoi-diagram
    - Cell-wise interpolation
- **Global** influence
  - Each sample might influence any other point within the domain
    - Material properties for whole object
    - Scattered data interpolation

3.4 Coordinate Systems

ohm

Coordinate Systems

2D coordinate systems

- Cartesian coordinates

$$P = (x, y)$$

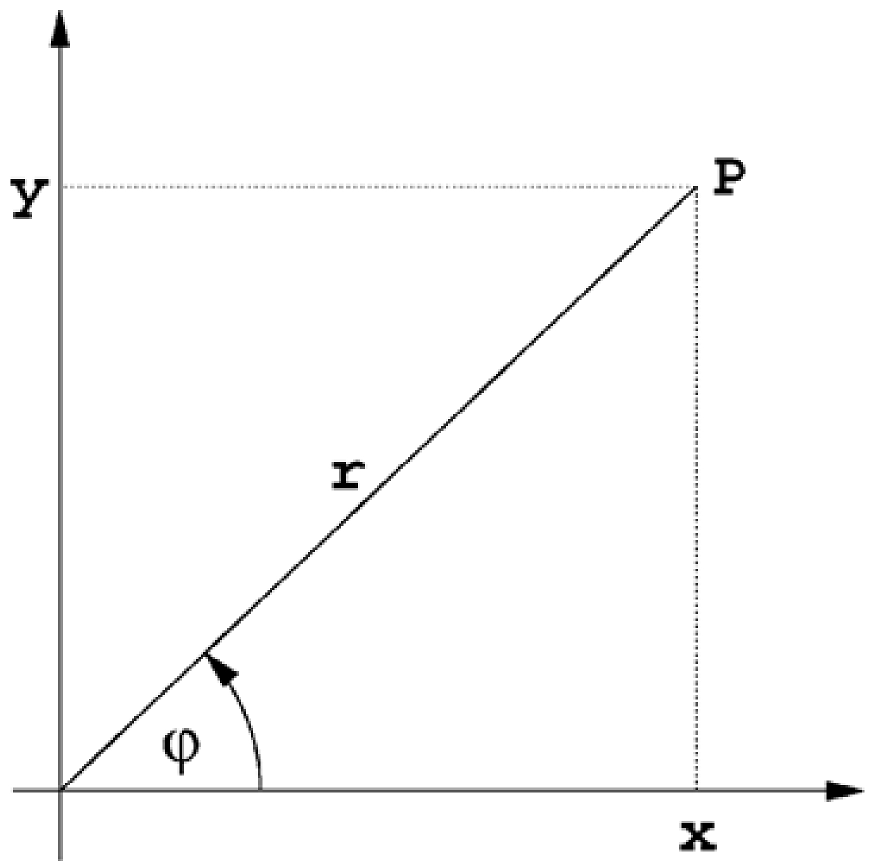
$$P = (r, \varphi)$$

- Polar coordinates

$$x = r\cos\varphi \qquad y = r\sin\varphi$$

$$r = \sqrt{x^2 + y^2}$$

$$\varphi = \arctan\frac{y}{x}$$

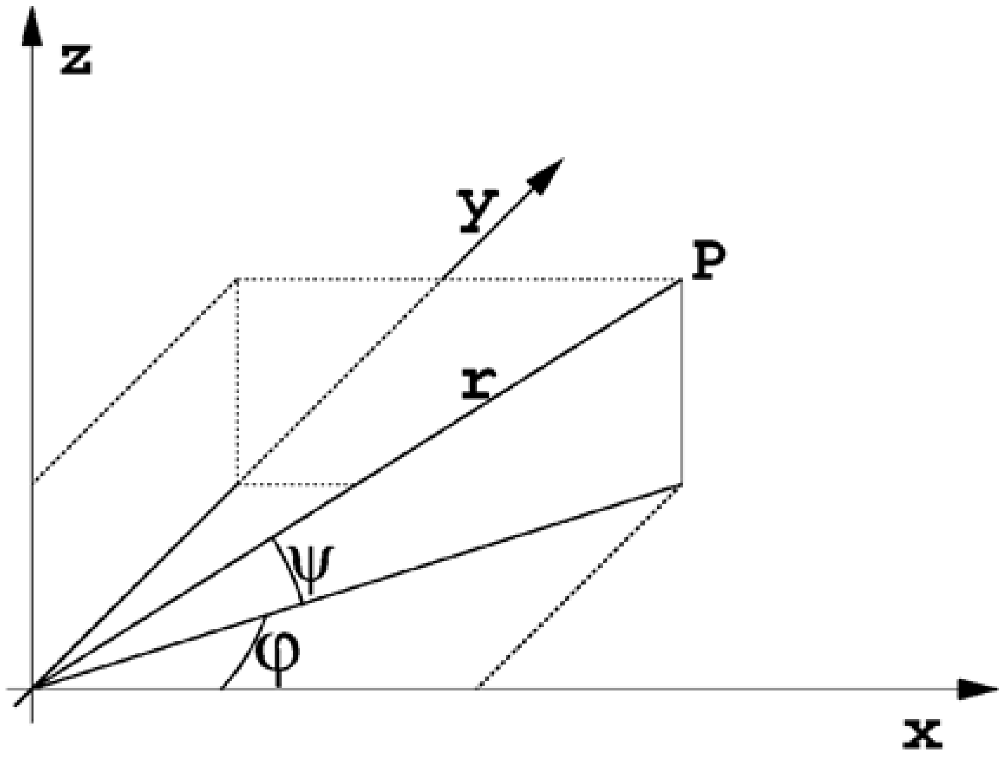


Coordinate Systems

3D coordinate systems

- Cartesian coordinates $P = (x, y, z)$
- Cylindrical coordinates $P = (r, \varphi, z)$
  - This is like polar coordinates in 2D
- Spherical coordinates $P = (r, \varphi, \psi)$
  - where

$$x = r \cos\varphi \cos\psi$$
$$y = r \sin\varphi \cos\psi$$
$$z = r \sin\psi$$



3D cartesian and spherical coordinates

Coordinate Systems

P-space ↔ C-space

- Transformation, where
  - P-space: physical space
  - C-space: corresponding uniform computer representation

3.5 Data Structures

Data Structures

- Requirements
  - Convenience of access
  - Space efficiency
  - Lossless vs. lossy
  - Portability
    - Binary - less portable, more space/time efficient
    - Text - human readable, portable, less space/time efficient
- Definitions
  - Scattered data
    - Arbitrarily distributed points with no connectivity in between
  - Otherwise
    - Data is composed of cells bounded by grid lines

Data Structures

Topology vs. Geometry

- Geometry
  - Specifies the position of the data
- Topology
  - Specifies the structure (connectivity) of the data
  - Main concern: qualitative questions about geometric structure
    - Are there holes?
    - Is everything connected?
    - Can it be split into individual parts?

ohm

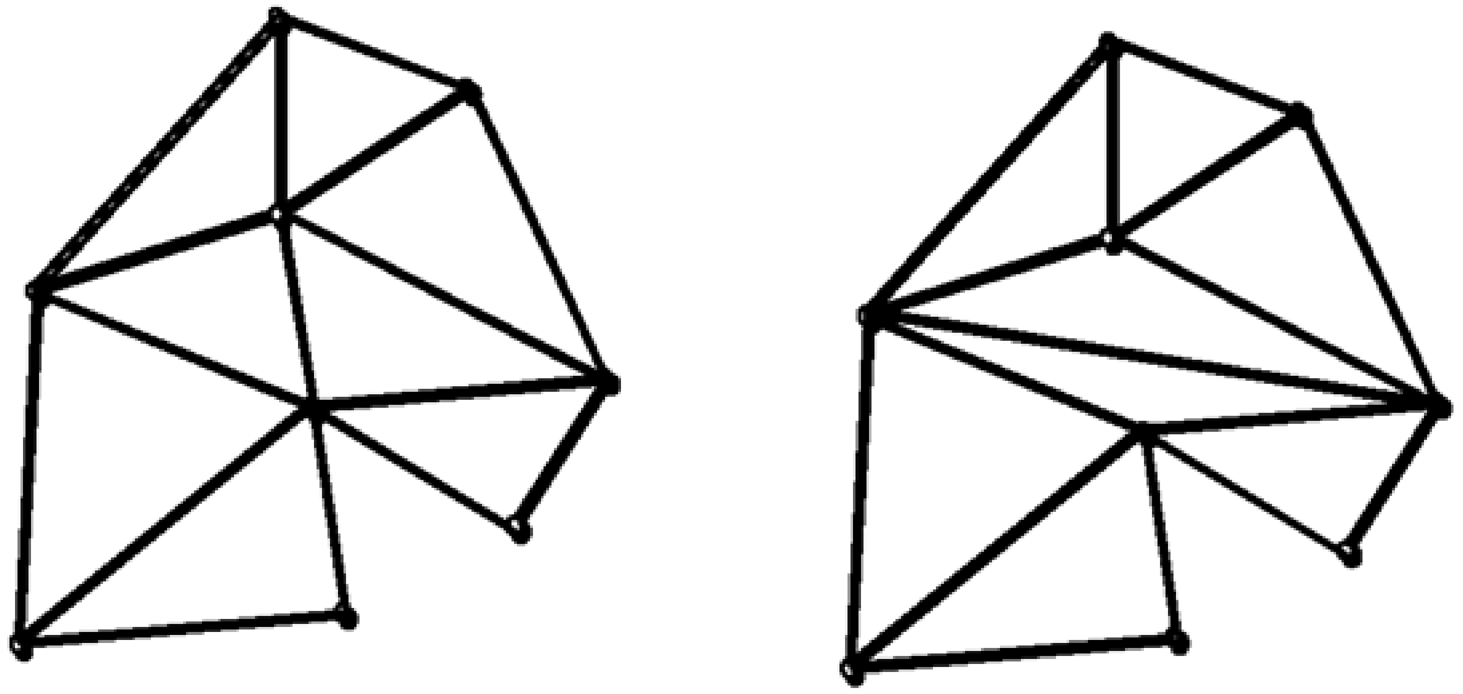
Data Structures

- Example: topological map of underground
  - Does not tell how far one station is from the other, but rather how the lines are connected

Data Structures

Topology

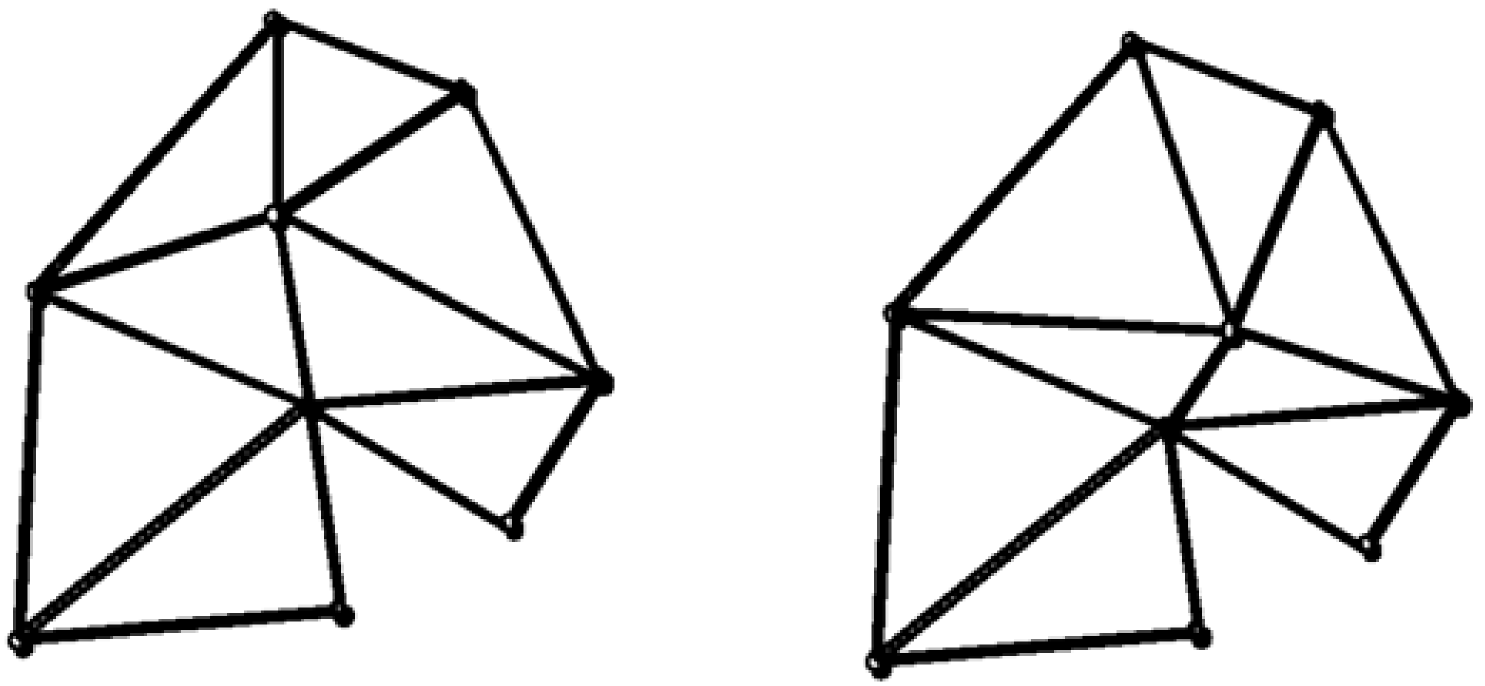
- Properties of geometric shapes that remain unchanged even under distortion



Same geometry (vertex positions), different topology (connectivity)

Data Structures

- Shapes that can be transformed into each other without tearing or introducing new connections are *topologically equivalent*



Topologically equivalent

Data Structures

Discrete representation of data: meshes / grids

- In general
  - List of vertices (explicit or implicit)
  - Global vertex index (explicit or given by order)
  - List of cells (explicit or implicit)
  - Global cell index (explicit or given by order)

Data Structures

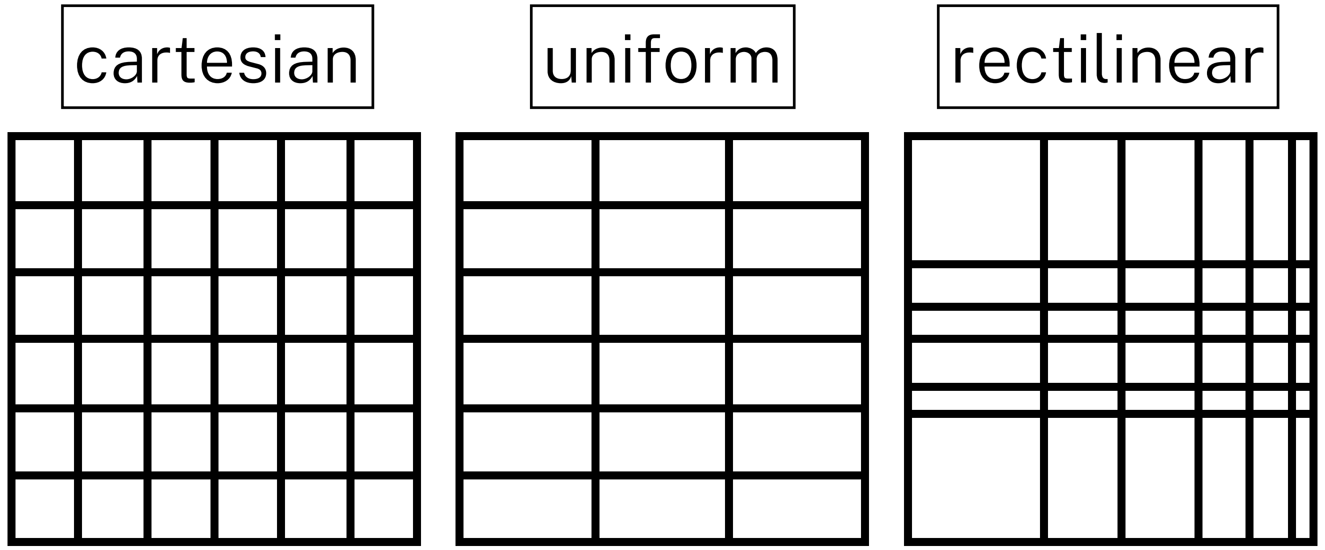
Discrete representation of data: meshes / grids

- Optional
  - List of edges, list of faces (3D)
  - Type flags
    - Edge: interior/boundary/complex/feature edge
    - Vertex: interior/boundary/complex/feature vertex (e.g. "corner")
  - Adjacencies/incidence (neighborhood relation)
    - cell → vertices, cell → faces, cell → edges, face → edges, ...

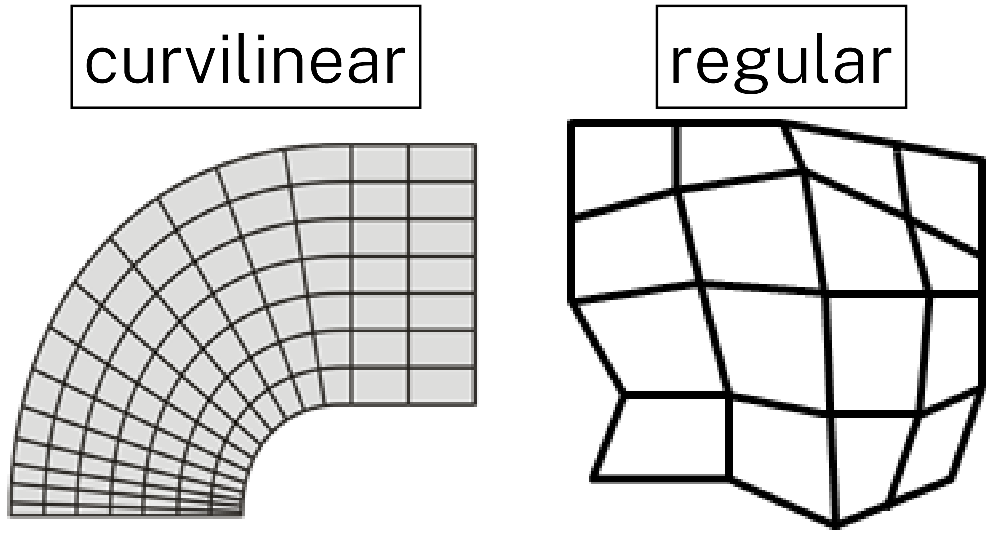
Data Structures

Structured meshes / grids

- Regular topology, regular / irregular geometry



- Cells
  - Evenly shaped
  - Squares
- Configuration
  - Uniform
  - Regular

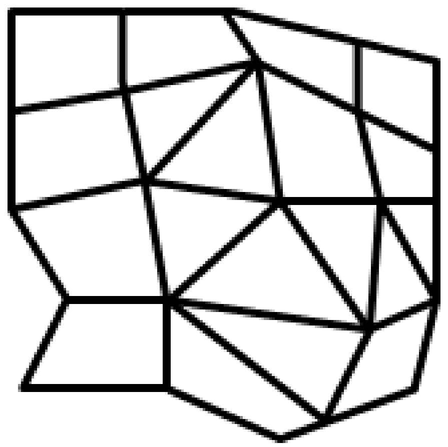


- Cells
  - Different quadrilaterals
- Configuration
  - Non uniform
  - Topologically structured

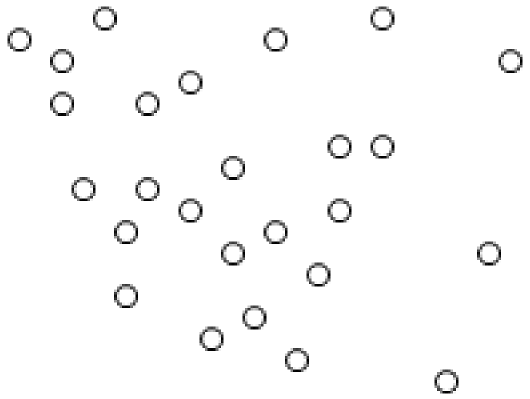
Data Structures

Unstructured meshes / grids

irregular



scattered data

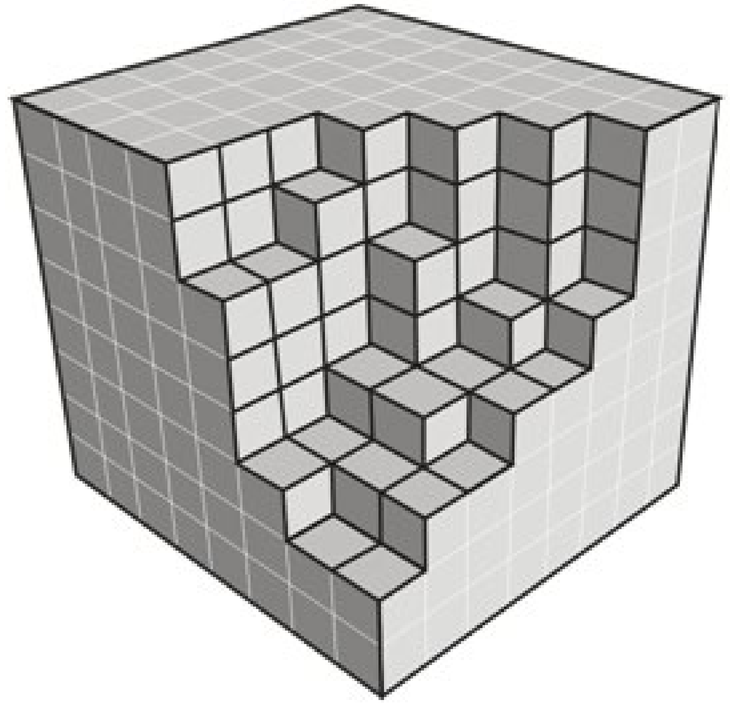


- Cells
  - Triangles (rarely rectangles or polygons)
- Configuration
  - Unstructured
  - Irregular topology and geometry

Data Structures

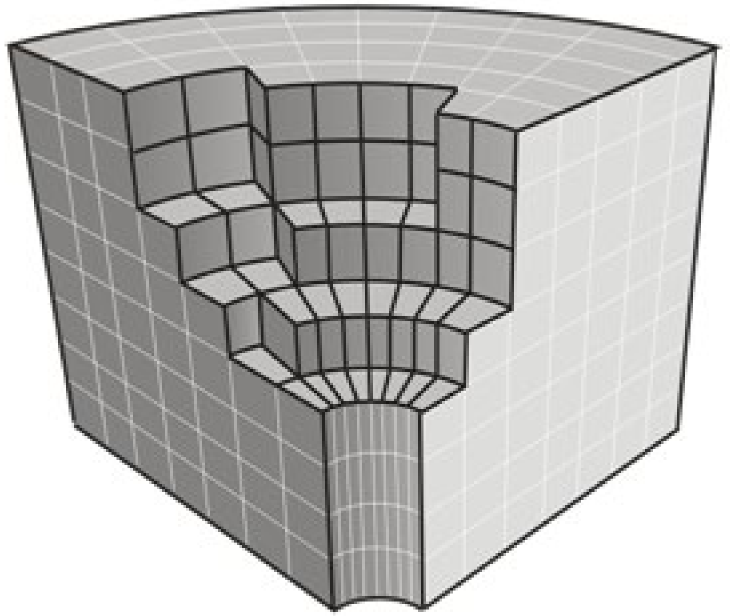
Meshes in 3D

uniform rectilinear



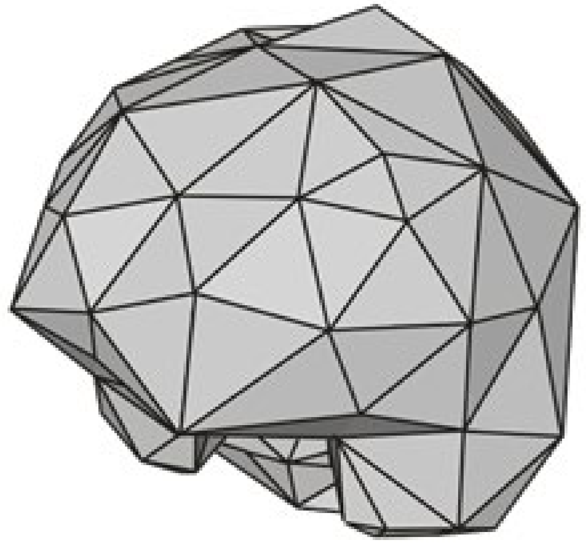
- Cells
  - Uniform hexahedra
- Configuration
  - Uniform
  - Regular

curvilinear



- Cells
  - Different shaped hexahedra
- Configuration
  - Non-uniform
  - Structured

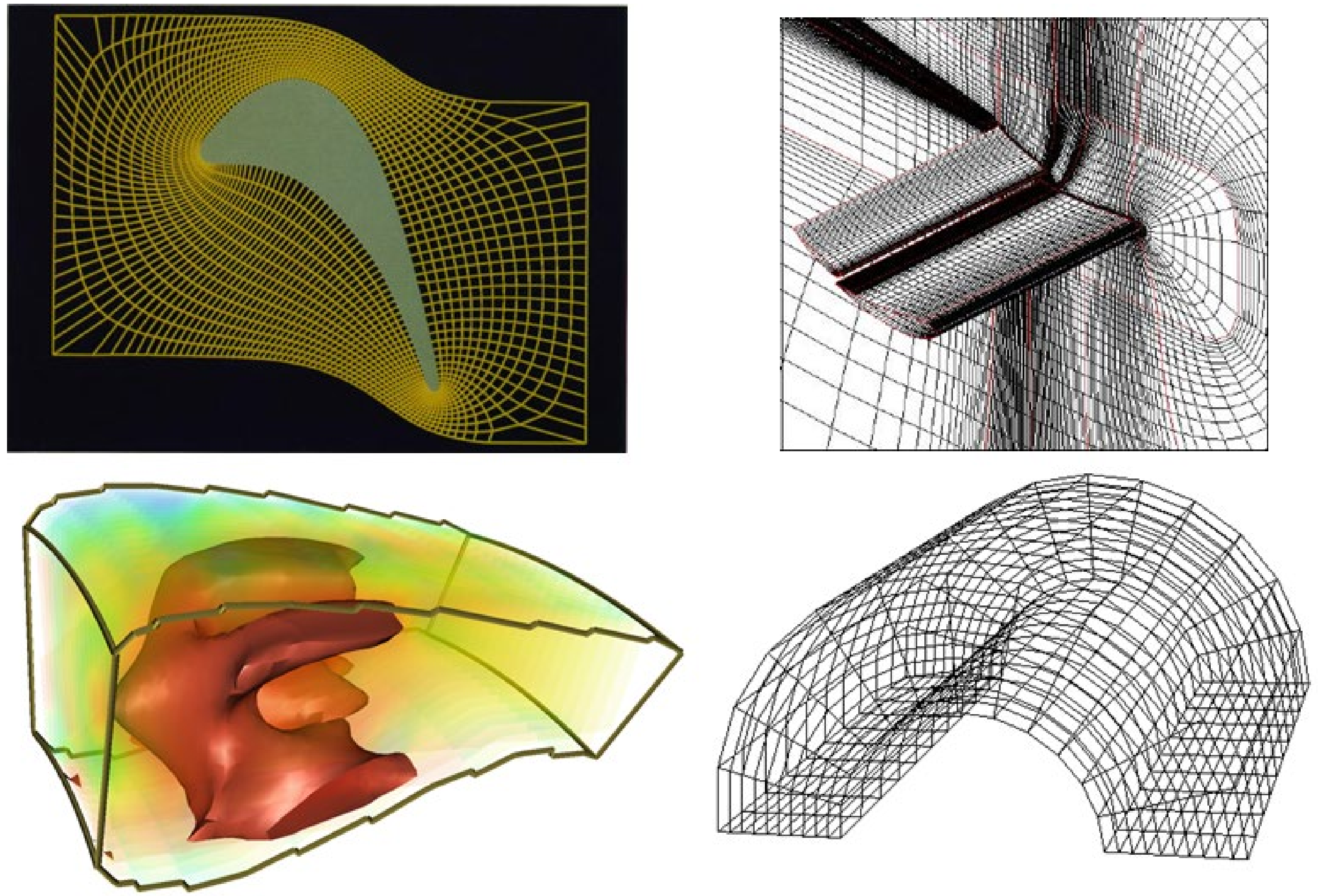
unstructured



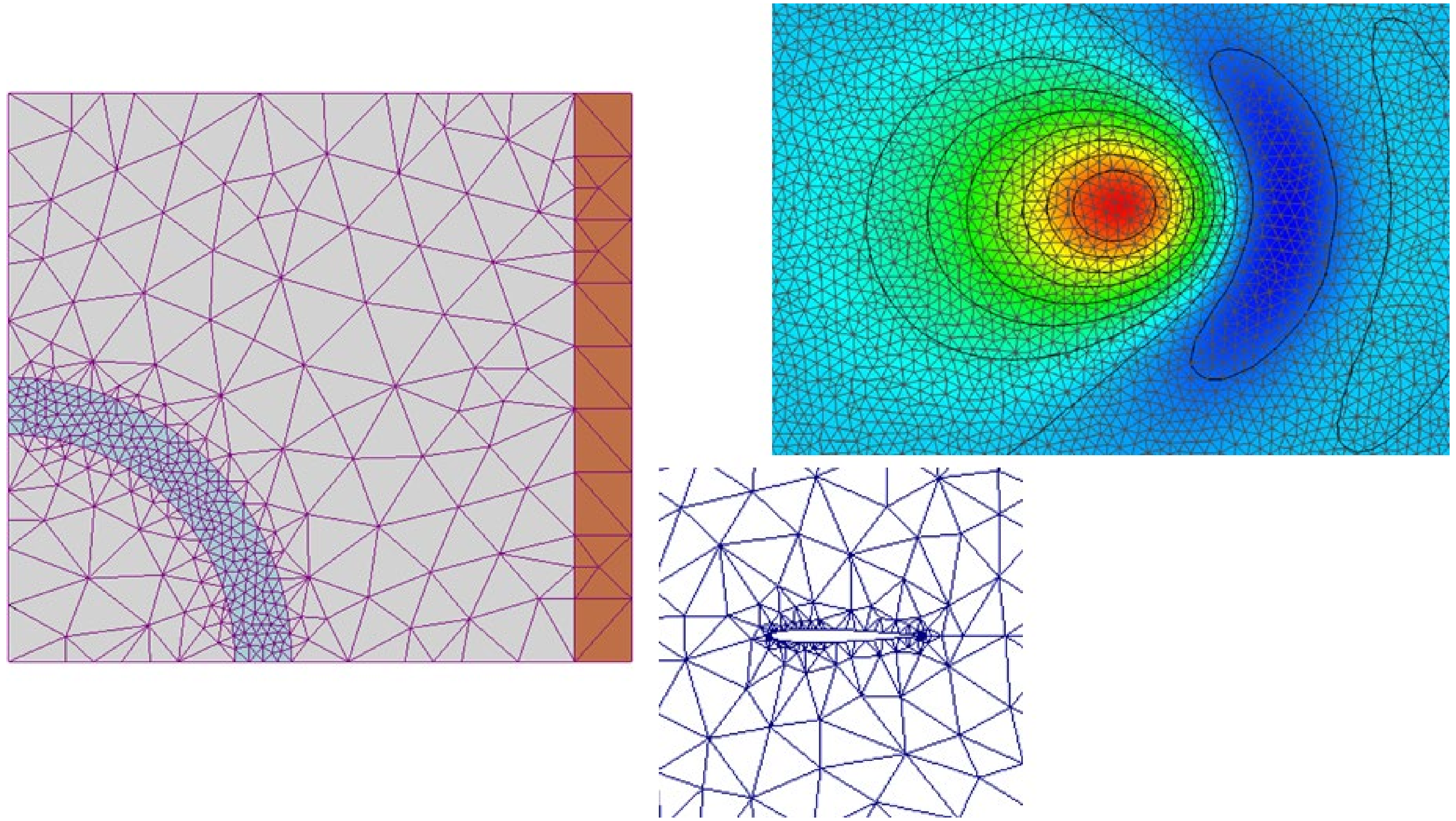
- Cells
  - Tetrahedra, pyramids, hexahedra, prisms
- Configuration
  - Unstructured

Data Structures

Examples of grids



structured



unstructured

3.5.1 Structured Grids

ohm

Structured Grids

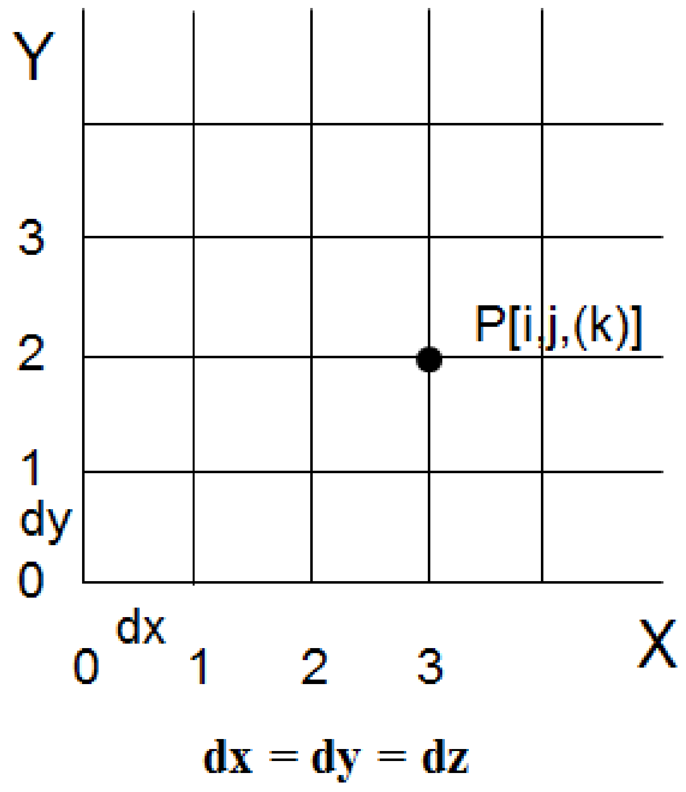
Characteristics of structured grids

- Simple computation
- Typical structure
  - Often parallelograms (hexahedra)
  - Cells being equal or non-linearly distorted
- May require more elements or badly shaped elements to cover the underlying domain
- Topology
  - Implicitly given by an n-vector of dimensions
- Geometry
  - Explicitly given by an array of points
- Every interior point has the same number of neighbors

Structured Grids

Cartesian or equidistant grids

- Structured grid
- Sequential numbering of cells and points
  - w.r.t increasing X, then Y, then Z
  - or vice versa
- Number of points
  - $N_X \cdot N_Y \cdot N_Z$
- Number of cells
  - $(N_X - 1) \cdot (N_Y - 1) \cdot (N_Z - 1)$



Structured Grids

Cartesian or equidistant grids

- Vertex positions are given implicitly from `[i,j.k]`
  - `P[i,j,k].x = origin + i*dx`
  - `P[i,j,k].y = origin + j*dy`
  - `P[i,j,k].z = origin + k*dz`
- Global vertex index: `I[i,j,k] = k*NY*NX + j*NX +i`
  - `k = I / (NY*NX)`
  - `j = (I % (NY*NX)) / NX`
  - `i = (I % (NY*NX) % NX) = I % NX`
- Global index allows for linear storage scheme
  - Wrong access patterns may destroy cache coherence

ohm

Data Structures

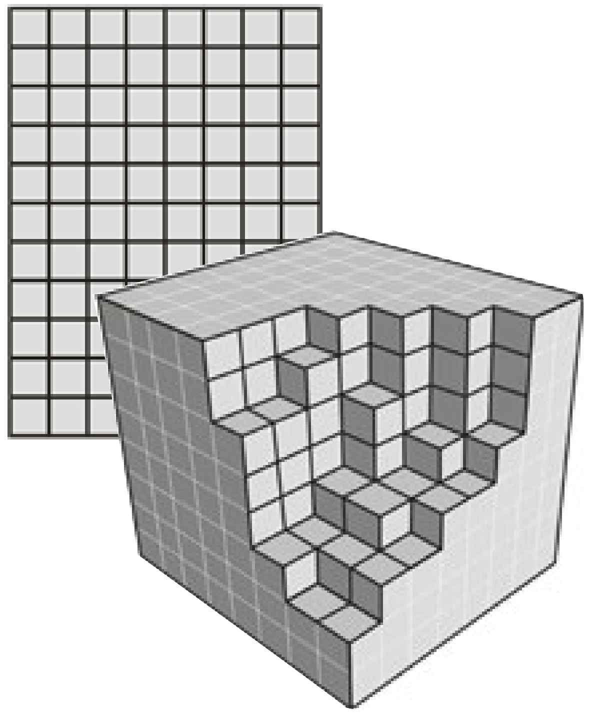
Uniform grids

- Like cartesian grids
- Equal cells, but with different resolution in at least one dimension `(dx <> dy (<> dz))`
  - Constant spacing in each dimension → same indexing scheme as for cart. grids
- Applications: data generated by 3D imaging devices w. different sampling rates for each dimension, e.g.:
  - Medical volume data consisting of slice images
    - Slice images with square pixels `(dx == dy)`
    - Larger (or smaller) slice distance `(dz > (dx == dy) || dz < (dx == dy))`

Structured Grids

Typical grid type in medical imaging: 2D/3D uniform grid

- Position of cells / vertices is given implicitly
  - Dimensions of Grid N_X, N_Y ,N_Z
    - Total number of cells: $(N_X - 1) \times (N_Y - 1) \times (N_Z - 1)$
  - Cell size $\Delta x, \Delta y, \Delta z$ (Note: data is in the cells!)
    - Distance of sampling points in x-, y- and z-direction



  - Spatial resolution
    - Pixel: $\Delta x \times \Delta y$, Voxel: $\Delta x \times \Delta y \times \Delta z$
    - Uniform grids (usually anisotropic): : $\Delta x = \Delta y \neq \Delta z$
    - Dimensions in continous space: $X = \Delta x \times N_X$, $Y = \Delta y \times N_Y$, $Z = \Delta z \times N_Z$,

ohm

Structured Grids

Representation of uniform grids

- Data stored as 1D-array with index i
  - $i = N_X \times N_Y \times z + N_X \times y + x$
  - $Origin$: X_0, Y_0, Z_0 (usually $X_0 = 0, Y_0 = 0, Z_0 = 0$)
- Implementation (C++): 1D- or multi-dimensional array

```
DataType *data = new DataType[NX * NY * NZ];

value = data[k * NX * NY + j * NX + i];
```

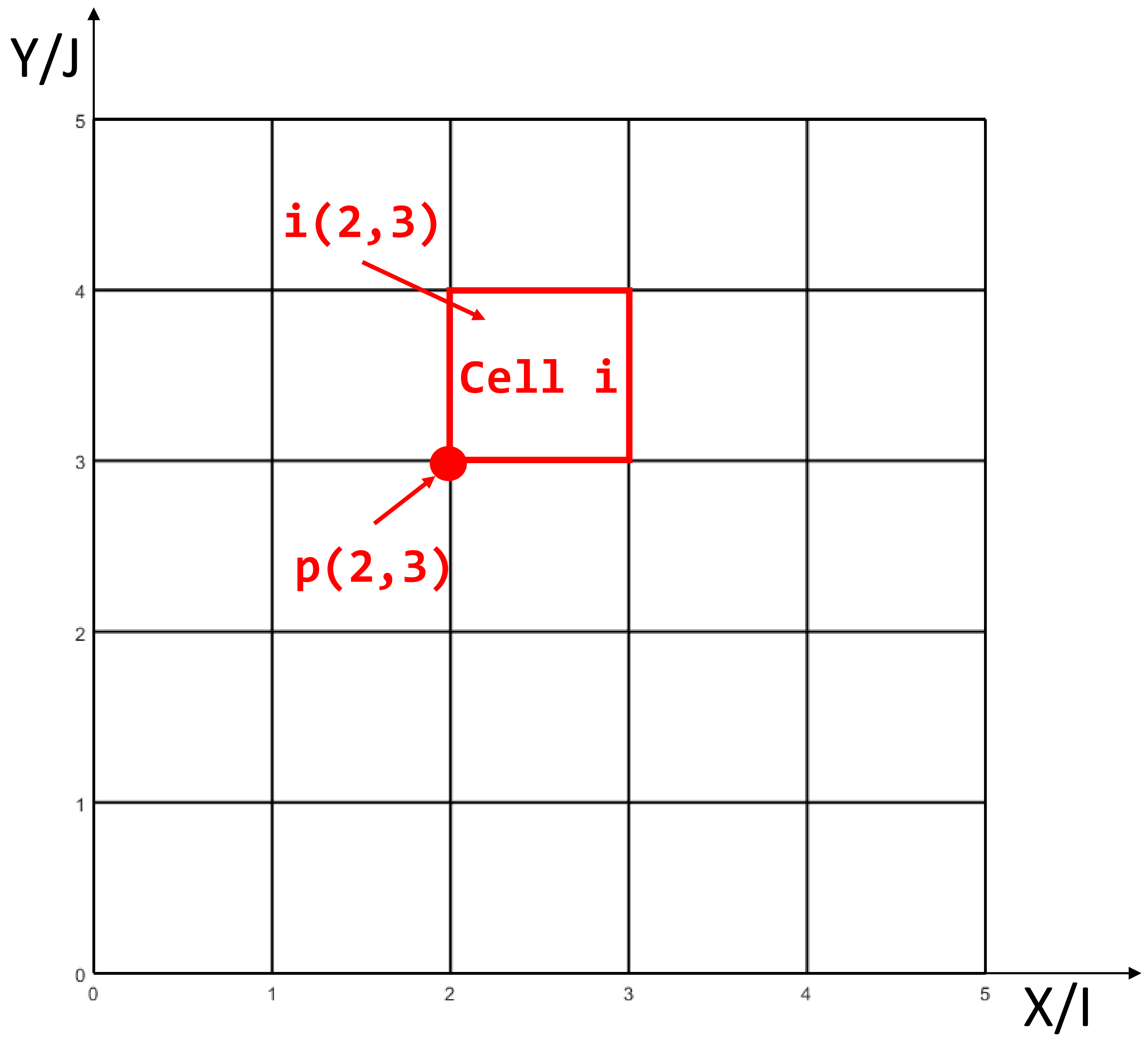
```
DataType ***data;
data = new DataType**[NX];
for (int i = 0; i < NX; ++i) {
  data[i] = new DataType*[NY];
  for (int j = 0; j < NY; ++j) {
    data[i][j] = new DataType[k];
  }
}
value = data[i][j][k];
```

- Note: often, i,j,k is used instead of x,y,z to distinguish between index and voxel/world coordinates

Structured Grids

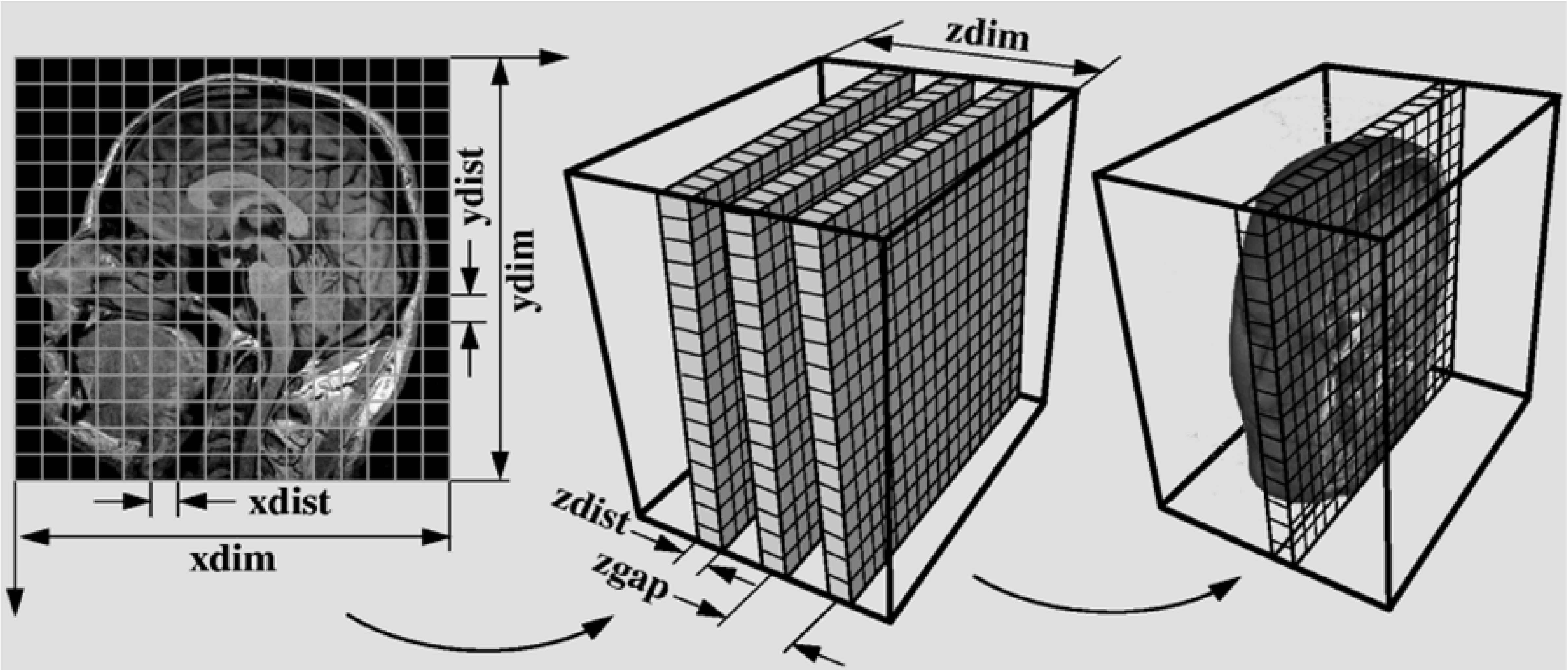
Example: 2D/3D uniform grid

- Example in 2D (voxel → pixel)
  - 6x6 grid → 36 points, 25 cells
- Coordinate and data index
  - $\mathrm{p}(2{,}3) = 6 * 3 + 2 = 20$
  - $i(2{,}3) = 5 * 3 + 2 = 17$
    - Cell contains the data value (e.g. image sample)
- Note
  - Discrete index coordinates
  - i, j and k are integer values
  - Independent of spacing!



Data Structures

Uniform grids in medical imaging

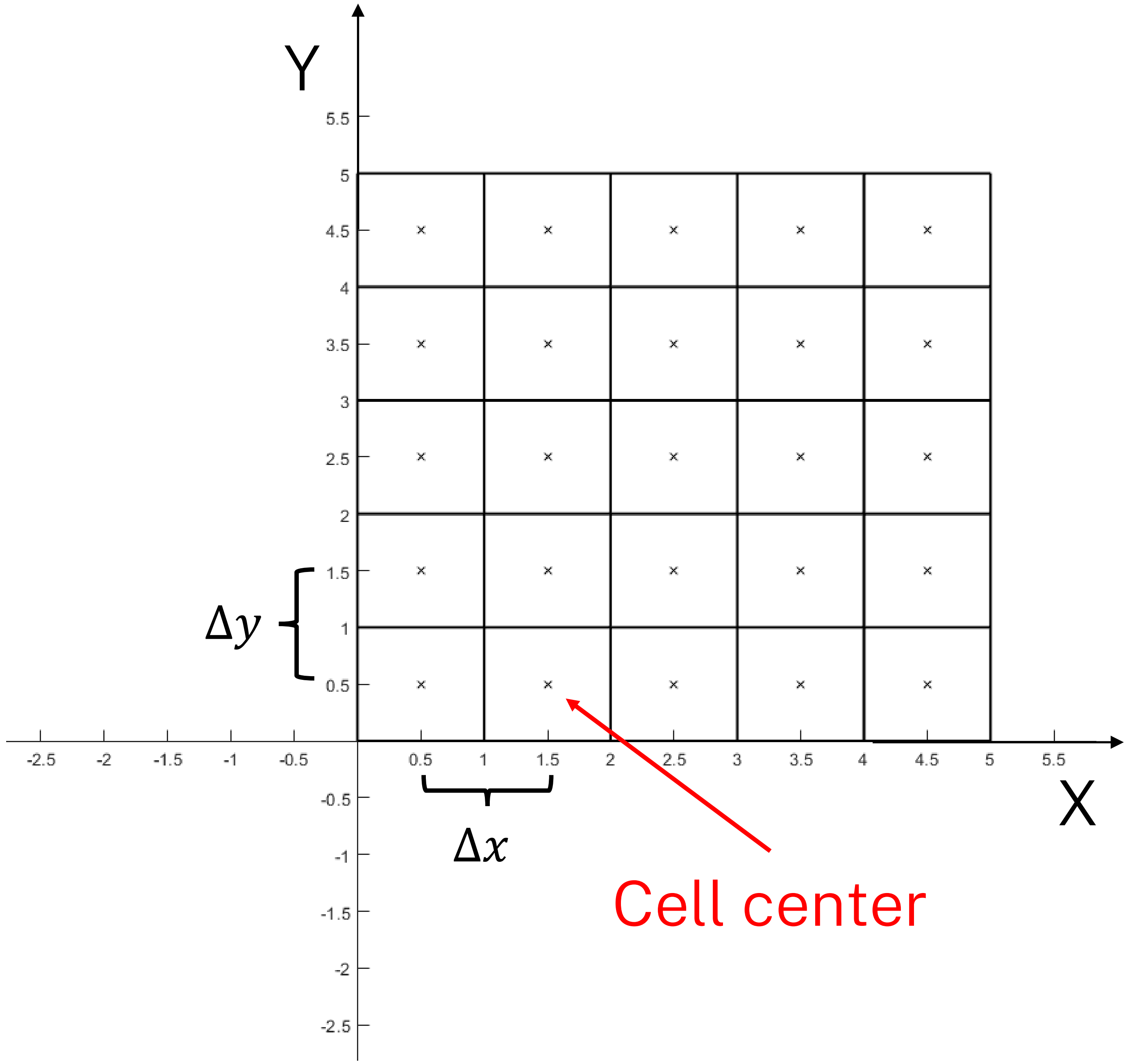


Slice image | Stack of slice images | Volume dataset

Data Structures

Impact of cell spacing

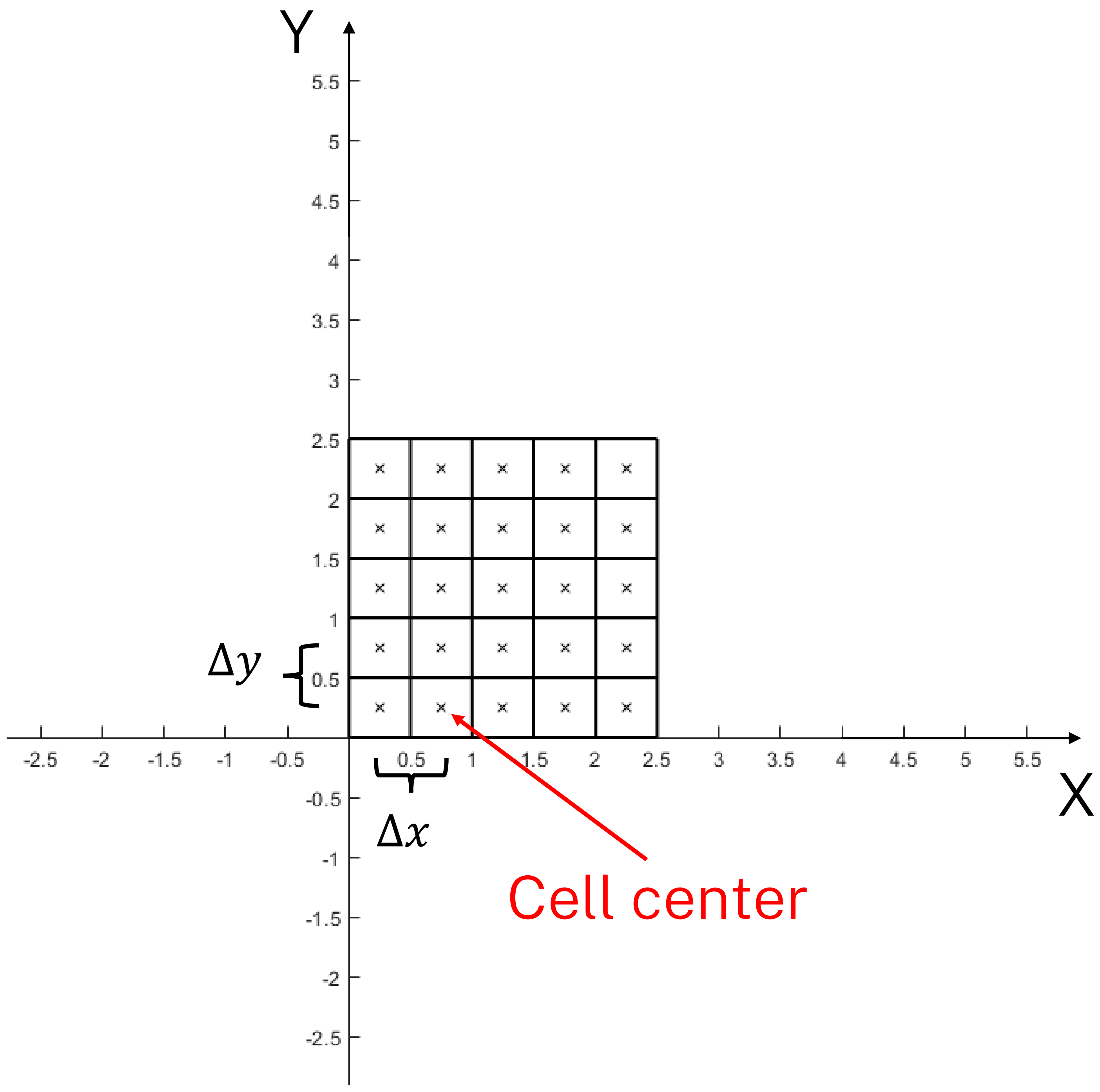
- Data continous on $[0..X,\ 0..Y, 0..Z]$
- Relative to data set
- Dependent on spacing
- Often anisotropic, sometimes non-orthogonal
- Data center (usually) at 0.5 (w.r.t. cell)
- Directly related to data in memory



Data Structures

Impact of cell spacing

- Same grid, same data
- Assume $\Delta x = \Delta y = 0.5$
- Data center still at 0.5 w.r.t. cell
  - Now 0.25 in world coordinates
- Affects almost all calculations, algroithms and visualization aspects
  - E.g. interpolation, differentiation…
- Neglecting cell spacing is a common implementation error



Data Structures

Rectilinear grids

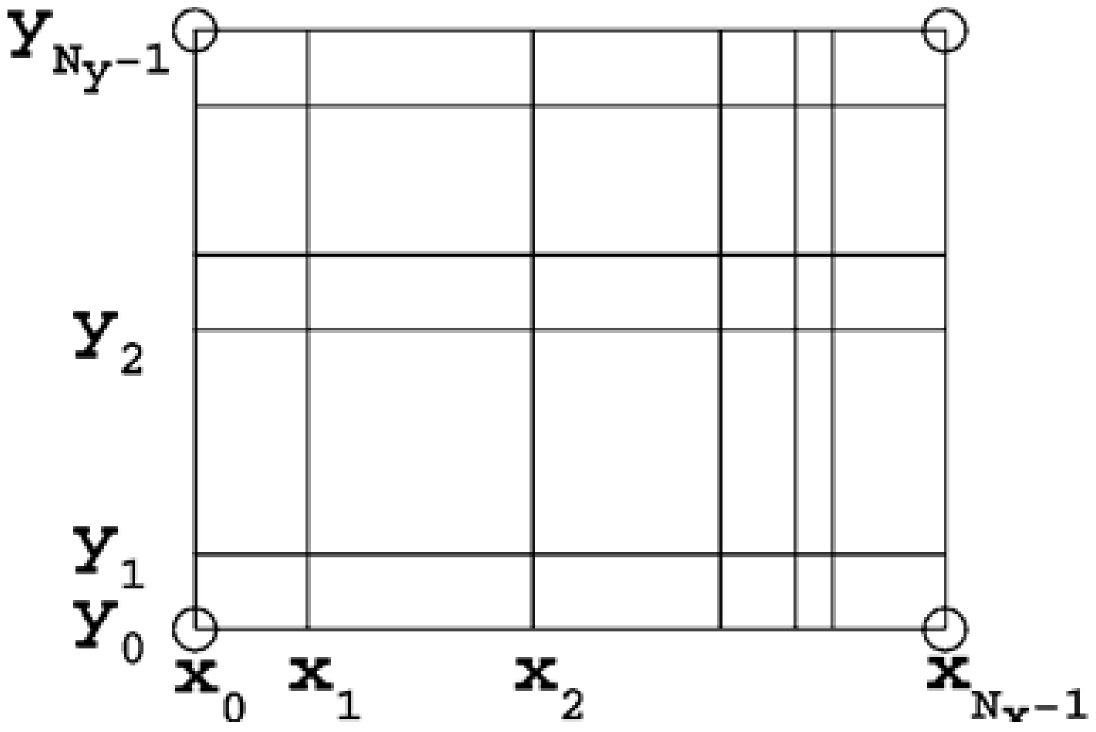
- Topology still regular and implicit
- But: irregular spacing between grid points (Geometry)
  - Non-linear spacing of positions along either axis
- Spacing must be stored explicitly
  - `x_coord[NX]`
  - `y_coord[NY]`
  - `z_coord[NZ]`

  Distance values

  and

  - `data[r]` with `r = k*NX*NY + j*NX + i`

  Data values



Data Structures

Generally structured or curvilinear grids

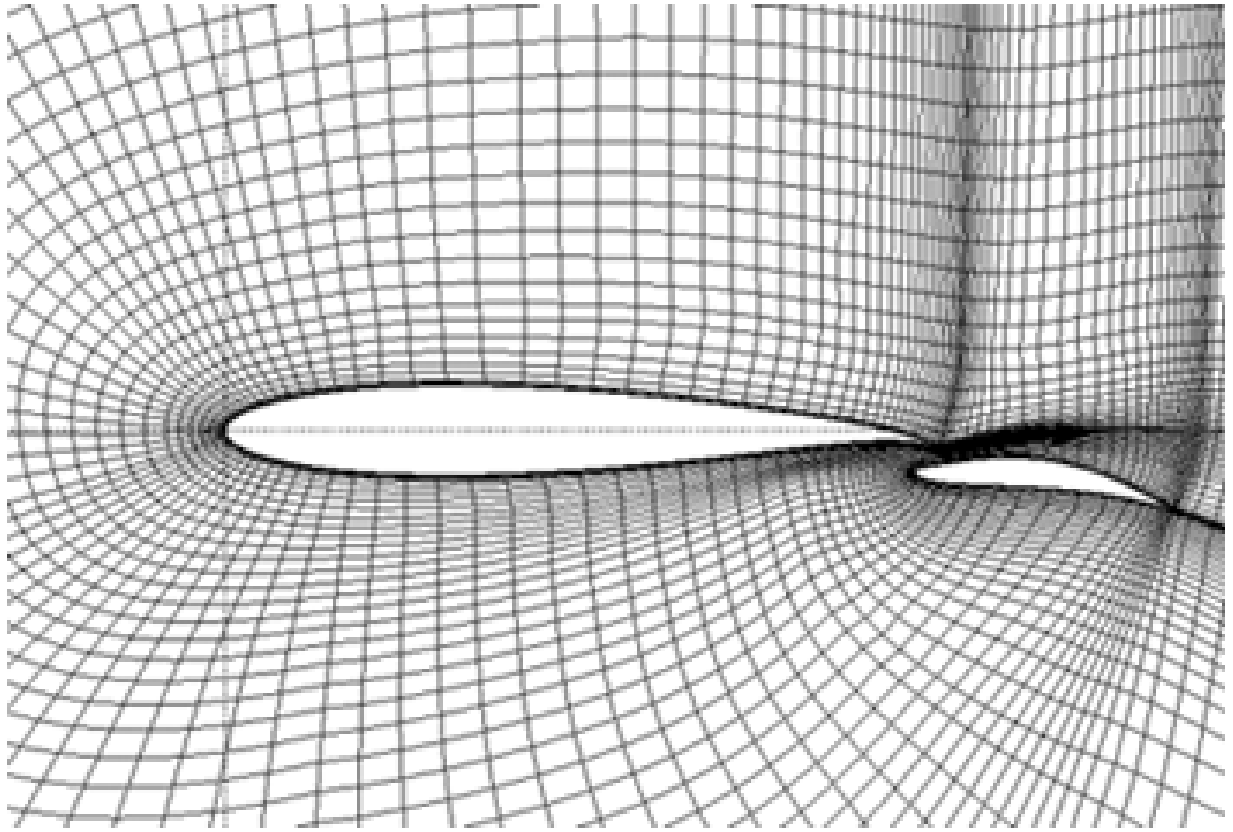
- Topology
  - Still regular, but irregular spacing between grid points
  - Positions are non-linearly transformed
- Geometry is explicitly stored
  - `x_coord[NX, NY, NZ]`
  - `y_coord[NX, NY, NZ]`
  - `z_coord[NX, NY, NZ]`

  Coordinates

  and

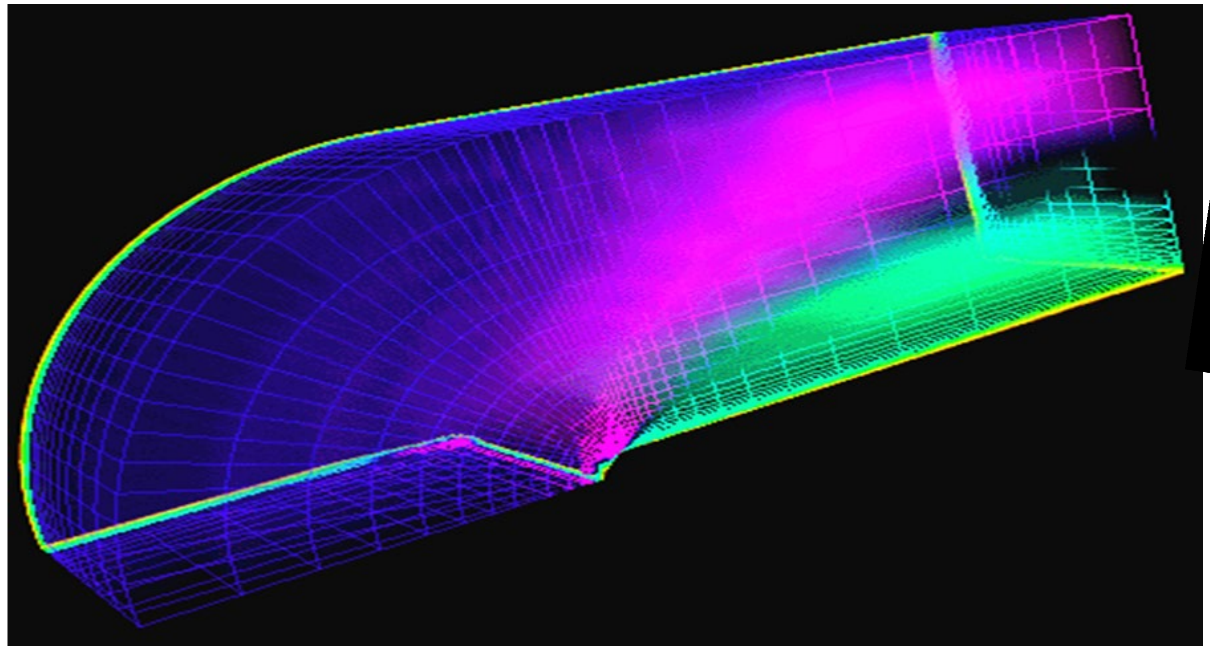
  - `data[r]`

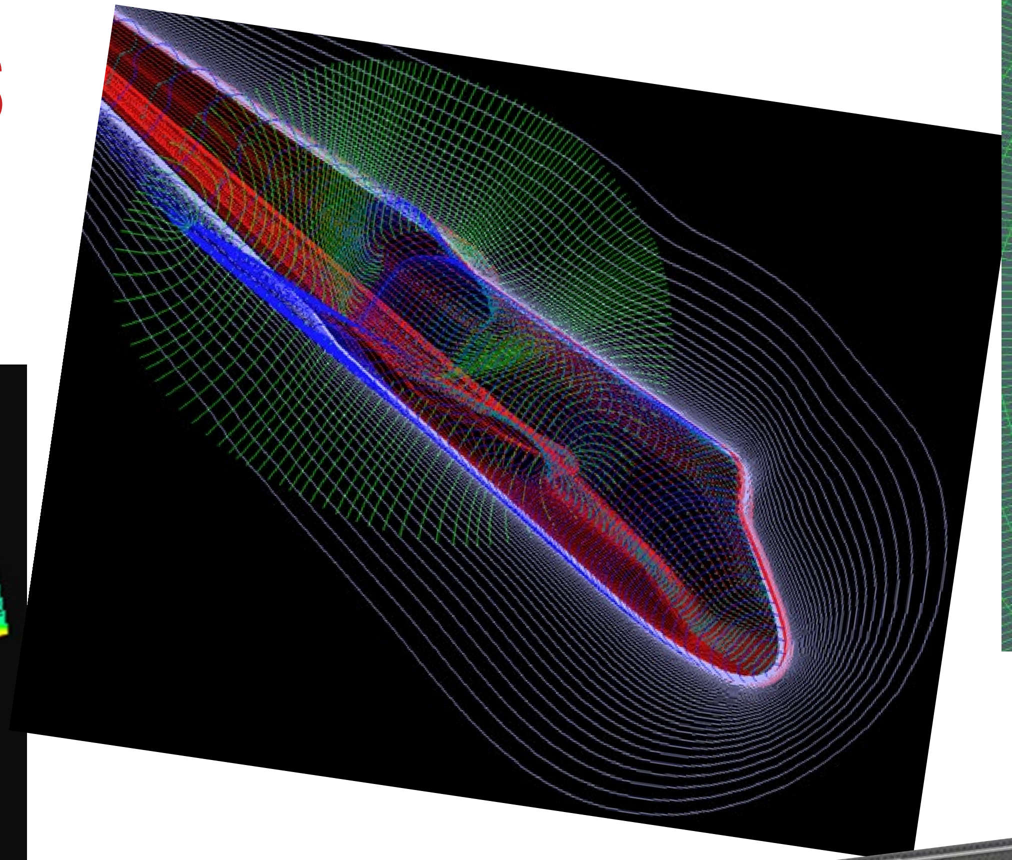
  Data values
- Geometric structure might result in **concave** grids

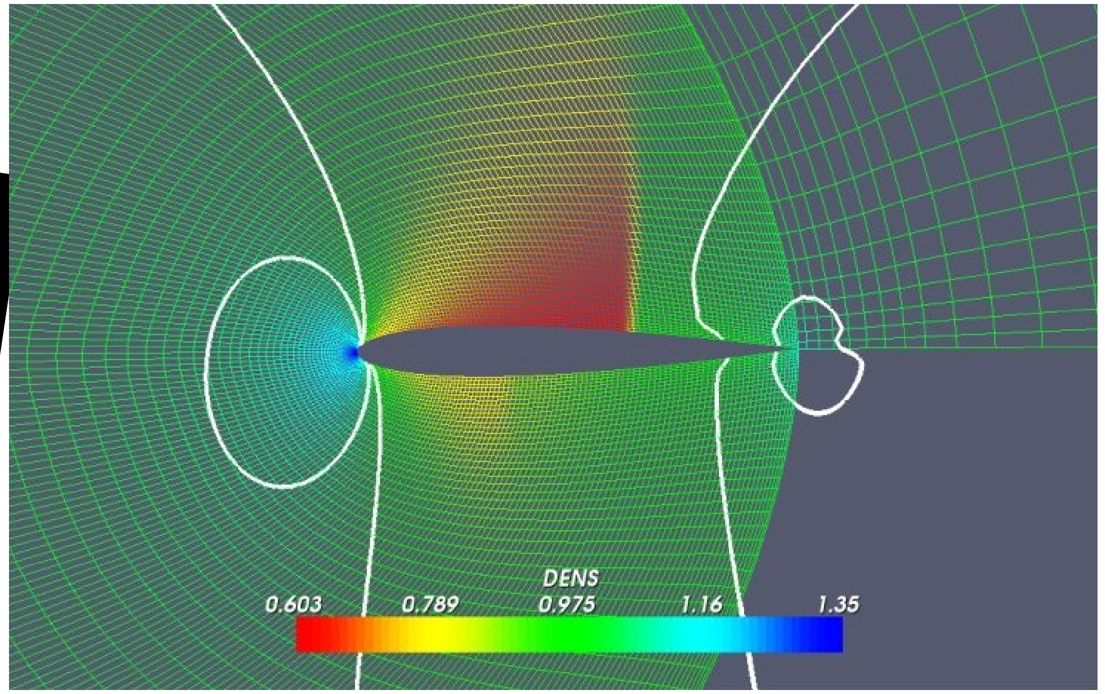


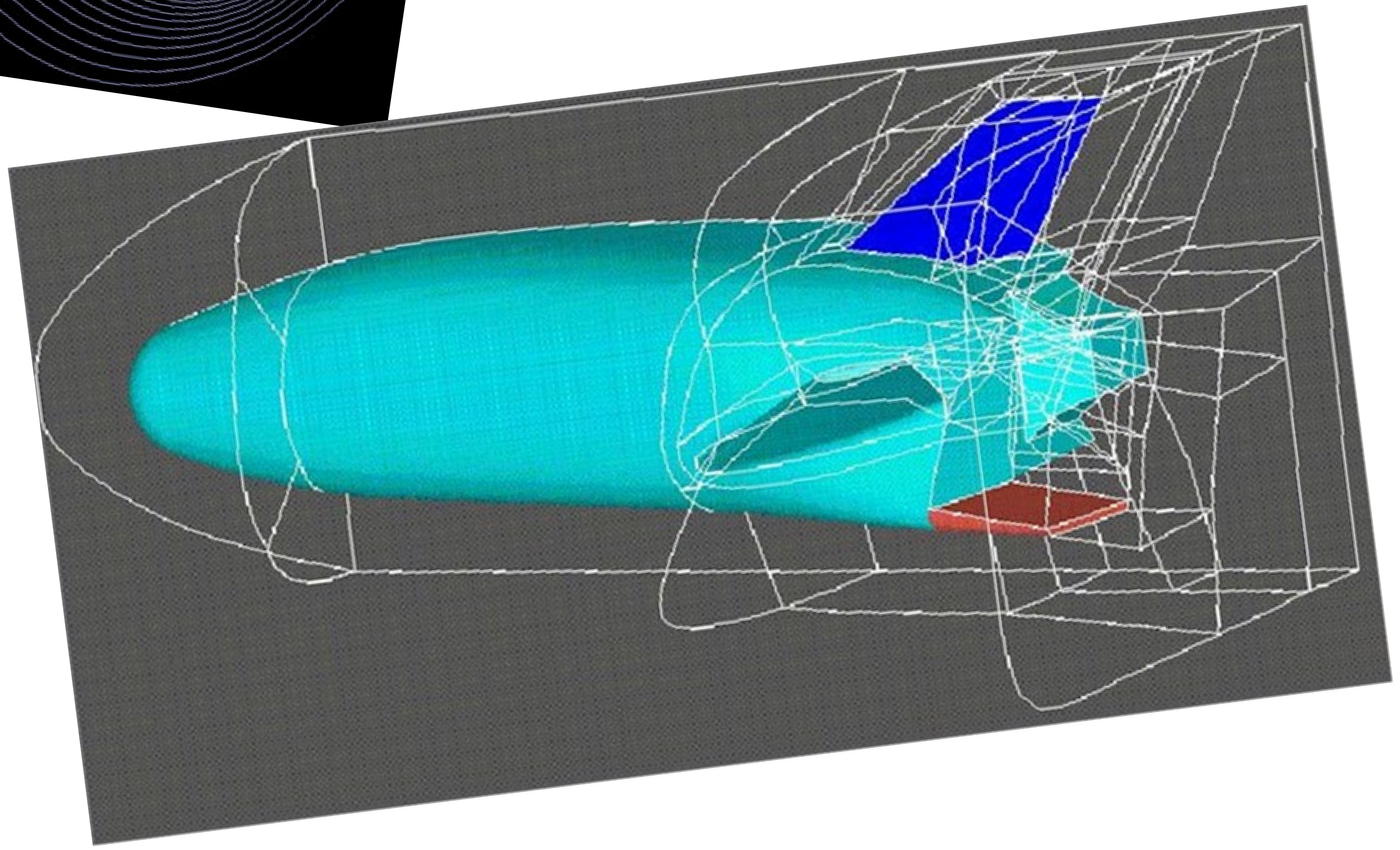
Data Structures

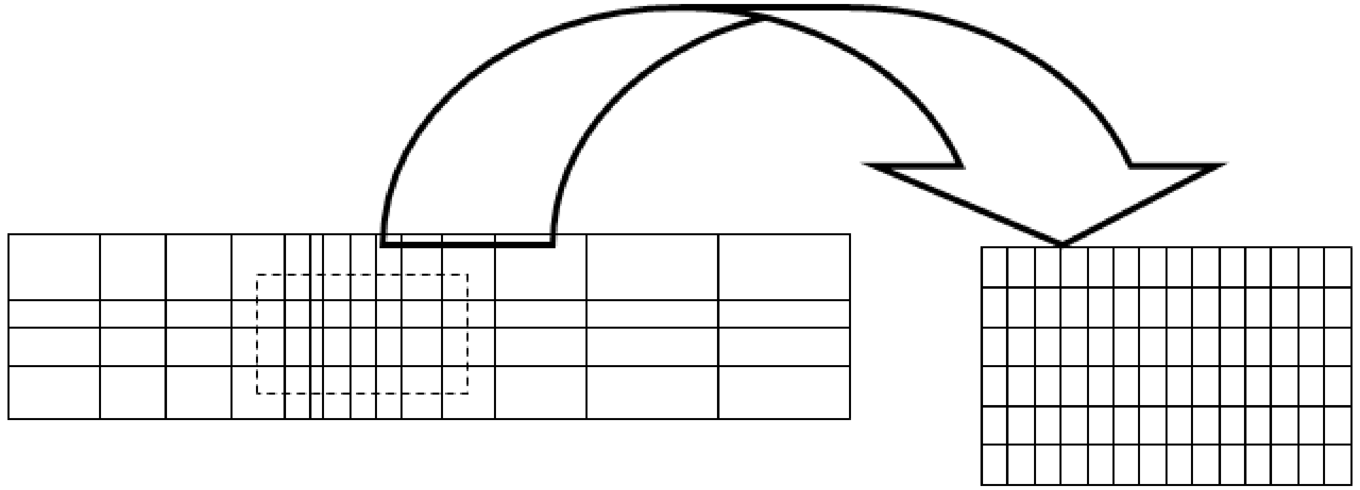
Examples











Data Structures

Summary

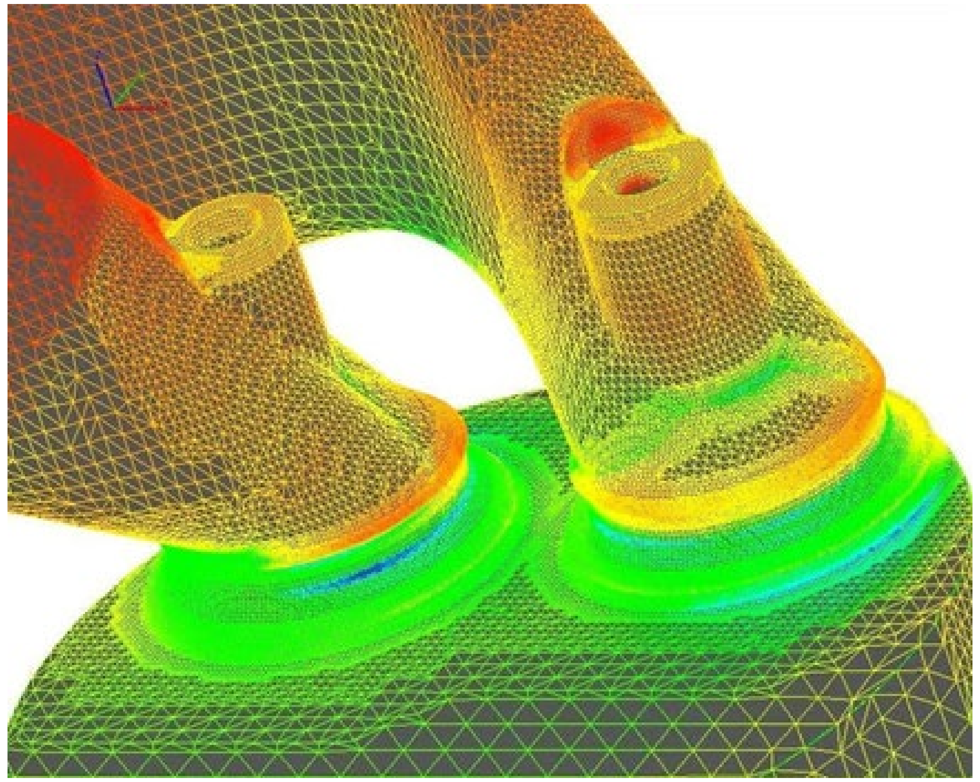
- Structured grids stored in 3D array
- Accessed by indexing
- Topology information implicitly given
  - Direct access to adjacent elements
- Cartesian, uniform, rectilinear grids
  - Necessarily convex
- Visibility ordering of elements implicitly given
  - With respect to viewing direction
- Rigid layout: inflexible for local features
- Curvilinear grids
  - More flexible, but complex sorting of grid elements

3.5.2 Unstructured Grids

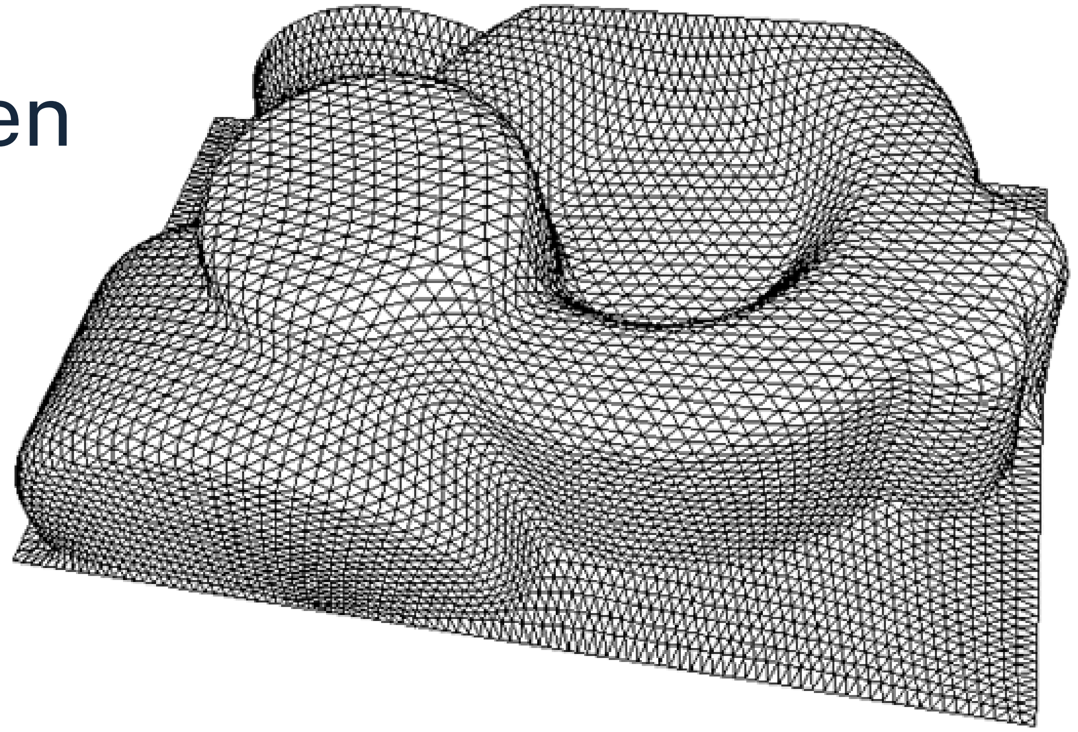
ohm

Unstructured Grids

Characteristics of unstructured grids



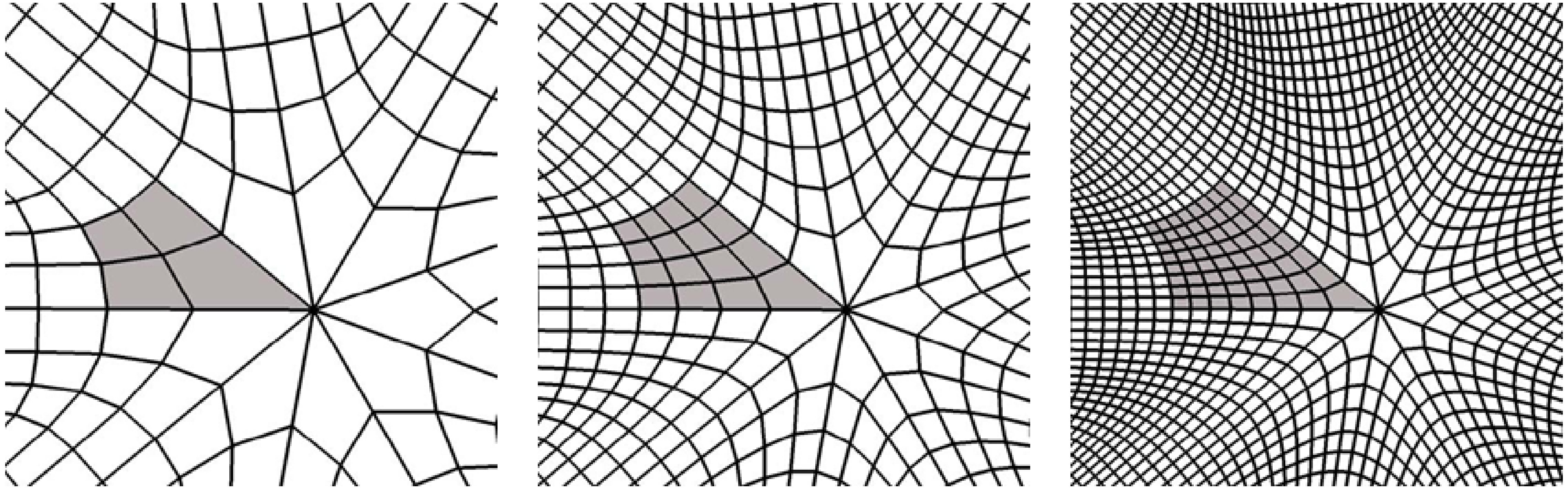
- Significantly more difficult and complex
  - But much more flexible as there are no constraints
- No implicit topological (connectivity) information given
  - Grid points + connectivity must be explicitly stored
- Dedicated data structures needed
  - Efficient traversal and data retrieval
- Often composed of triangles or tetrahedra
  - Requires triangulation or tetrahedrization
- Less elements are needed to cover the domain



ohm

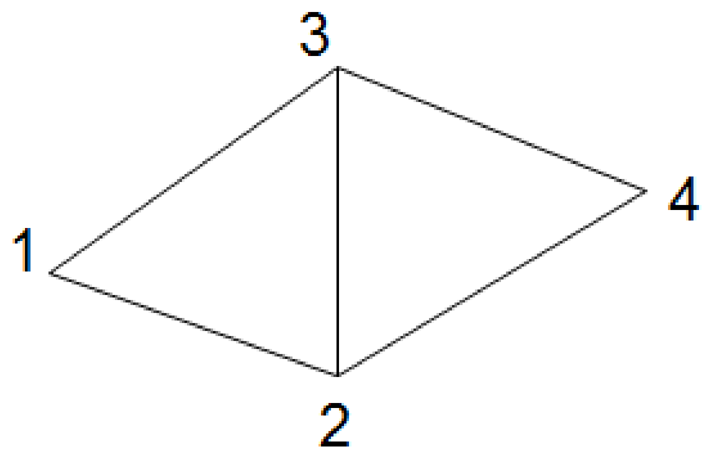
Unstructured Grids

- Dimension
- Number of vertices
- Number of cells
- Vertex list: $(x_0, y_0, z_0), (x_1, y_1, z_1), (x_2, y_2, z_2), \ldots$
- Cell list: $(\text{index}_{V1}, \text{index}_{V2}, \text{index}_{V3}, \text{index}_{V4}), \ldots$



Unstructured Grids

Representation in direct form



$$\left.\begin{array}{l} x1,\ y1 \\ x2,\ y2 \\ x3,\ y3 \end{array}\right\} \text{face 1}$$

$$\left.\begin{array}{l} x2,\ y2 \\ x3,\ y3 \\ x4,\ y4 \end{array}\right\} \text{face 2}$$

2D

```
Struct face {
    float verts [3][2];
    DataType value;
}
```

3D

```
Struct face {
    float verts [3][3];
    DataType value;
}
```

- Additionally, store the data values
- Problems: storage space, redundancy of edges

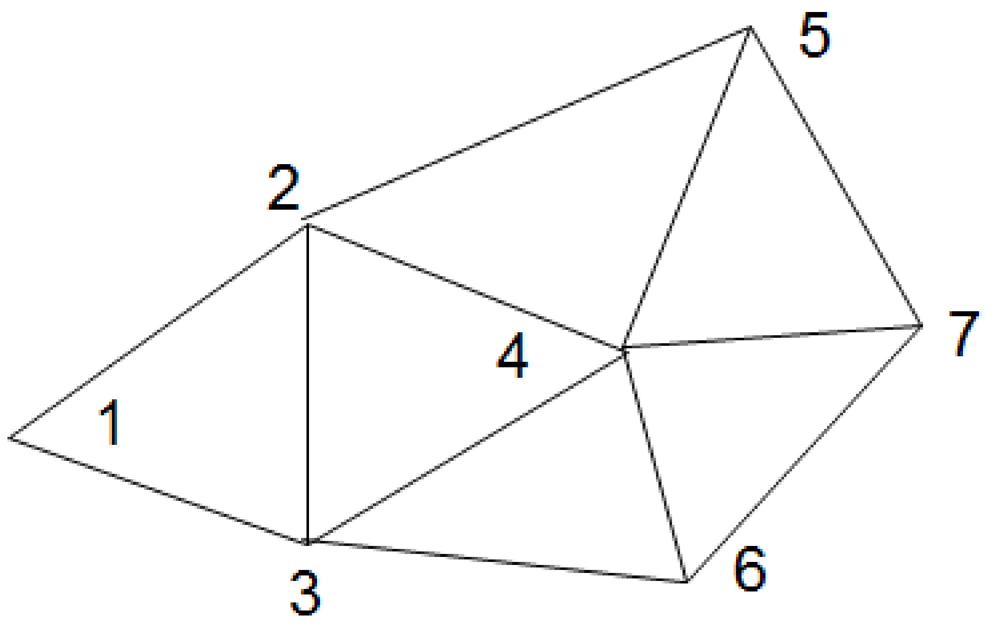
ohm

Unstructured Grids

Representation in indirect form ("indexed face set")

Vertex list	Face list
x1, y1	1,3,2
x2, y2	3,4,2
x3, y3	3,6,4
....	2,4,5
....	....

Uniform ordering
Counter clockwise



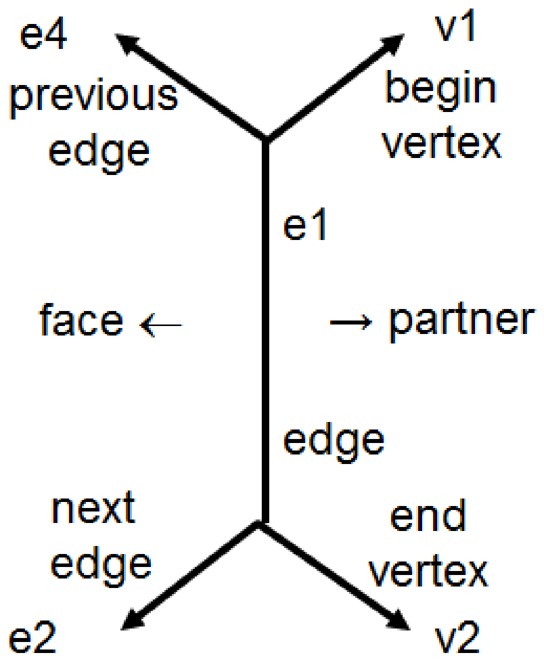
- More efficient than direct approach in terms of memory requirements
- But, still have to do global search to find local information
  - e.g., which faces share an edge?

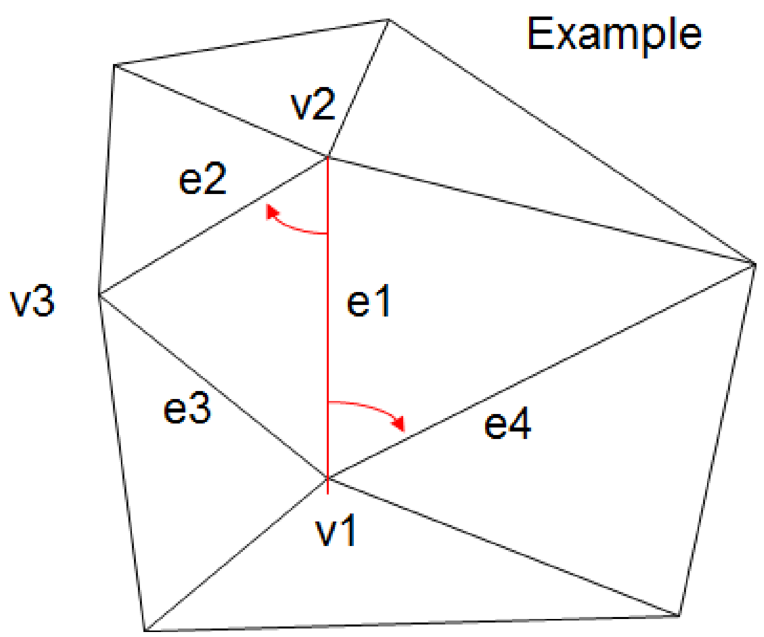
ohm

Unstructured Grids

Edge based approach ("winged edge")

- Name from underlying structure





- For every vertex (vertex table)
  - Store a pointer to an arbitrary edge that is adjacent
- For every face (face table)
  - Store a pointer to an edge on its boundary

Unstructured Grids

Edge based approach ("winged edge")

- Answers the following queries
  - Faces sharing an edge
  - Faces sharing a vertex
  - Walk around: faces, vertices (like a fan)
- Implicit assumption
  - Every edge has at most 2 faces which meet at the edge
- Advantages
  - Memory efficient and fast traversal

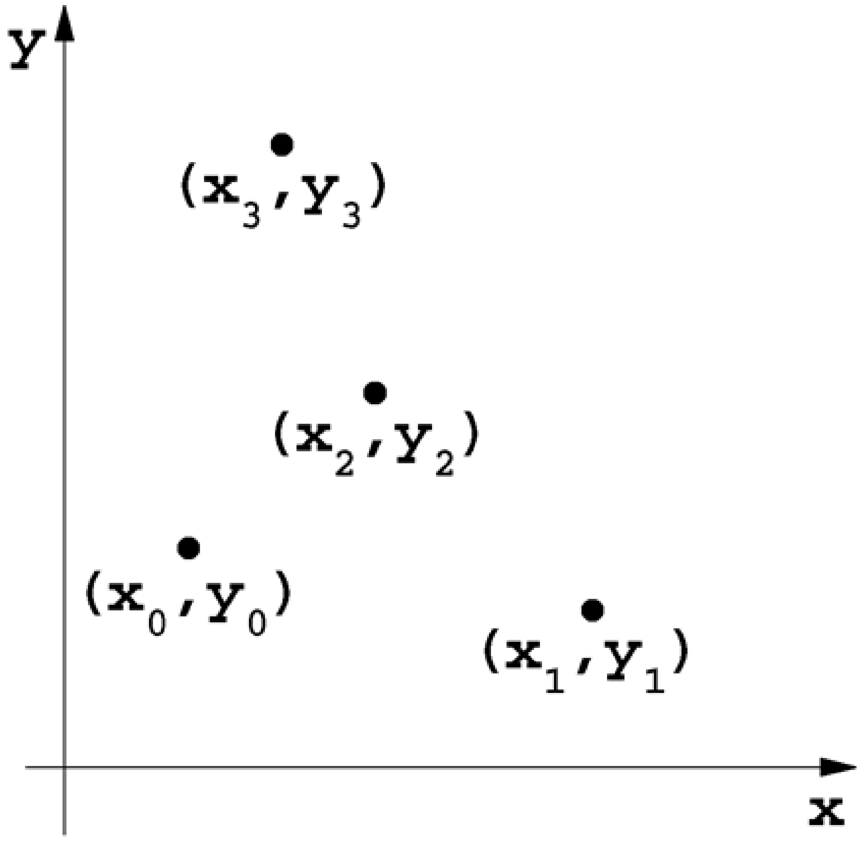
3.5.3 Scattered Data and Triangulation

ohm

Scattered Data

Scattered points

- Problem: create grid from scattered points
- Given
  - Number of vertices
  - Vertex list:
    - $x_0, y_0, z_0;$
    - $x_1, y_1, z_1;$
    - ...



Scattered points

Triangulation

Problem setting

- Given information
  - Data points: $(x_0, y_0), (x_1, y_1), \ldots, (x_{N-1}, y_{N-1})$
  - Data values: $f_0, f_1, \ldots, f_{N-1}$

- "Neighborhood information" is required (topology)
  - Task: Find triangular mesh with given scattered data points as vertices

- As measure for "good" triangulation of the data points, consider the "roundness" of triangles
  - radius incircle / radius out-circle
  - Maximal (or minimal) angle

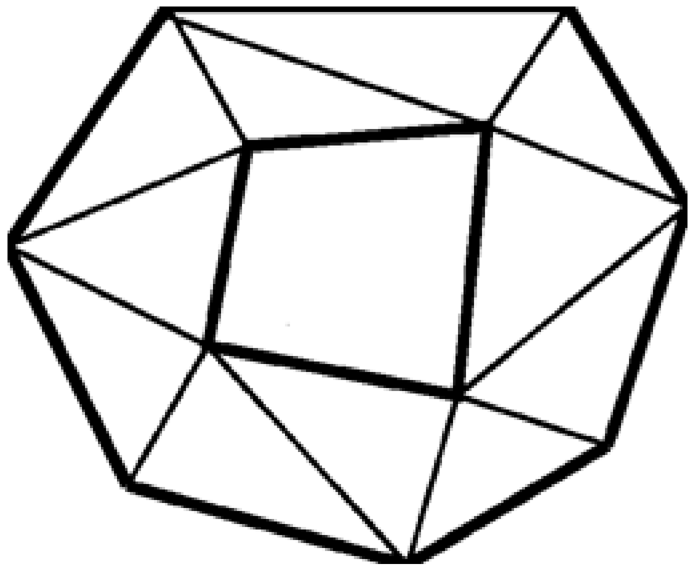
Triangulation

- A triangulation of data points $S = \{s_1, ..., s_n\}$ in R^2 consists of:
  - Vertices → 0D cells = S
  - Edges → 1D cells connecting 2 vertices
  - Faces → 2D cells connecting 3 vertices
  - Tetrahedra → 3D cells connecting 4 vertices

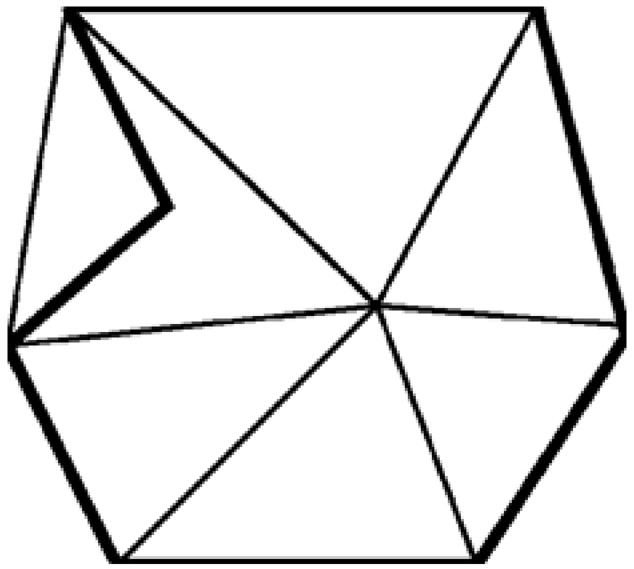
- Conditions to satisfy
  - $\cup$ faces... = conv(S),
    - i.e. the union of all faces (including the boundaries, which are the edges and vertices) is the convex hull of all vertices.
- Intersection of two triangles is
  - either empty
  - or a common vertex / edge / face

Triangulation

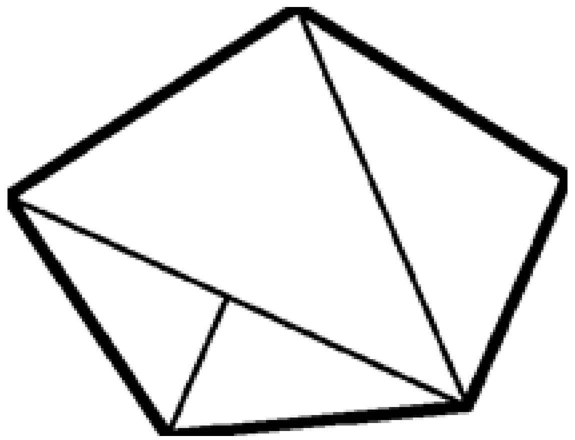
Non-valid triangulations



Hole



Faces overlap



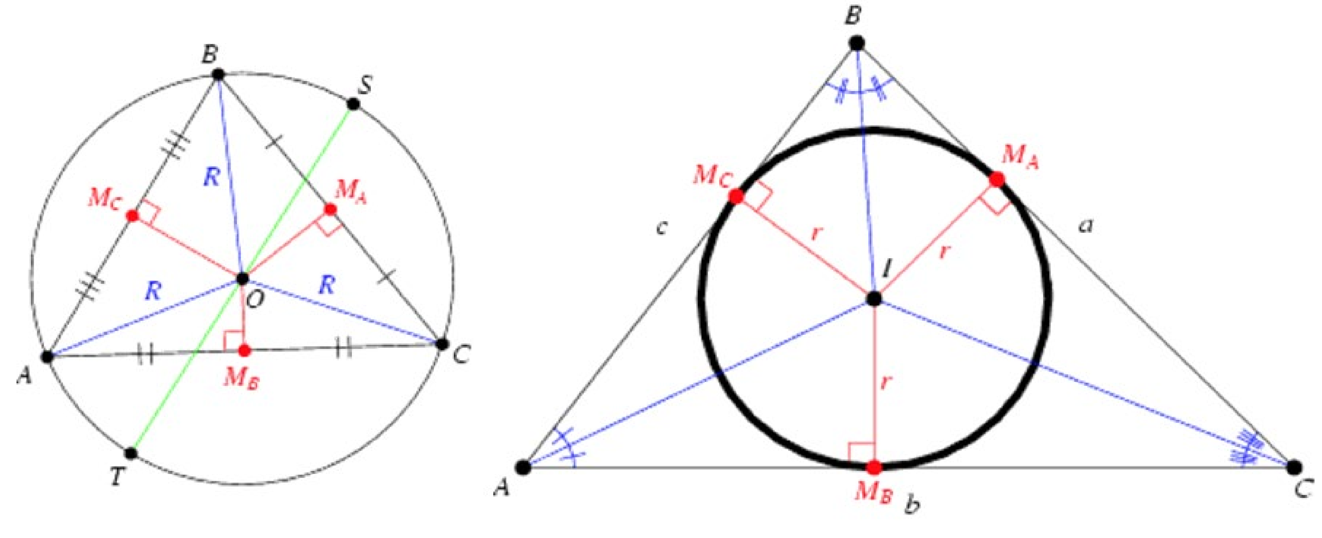
T-Vertex

Keine Lücken, keine Überlappungen, keine Verwendung von gleichen Kanten

ohm

Triangulation

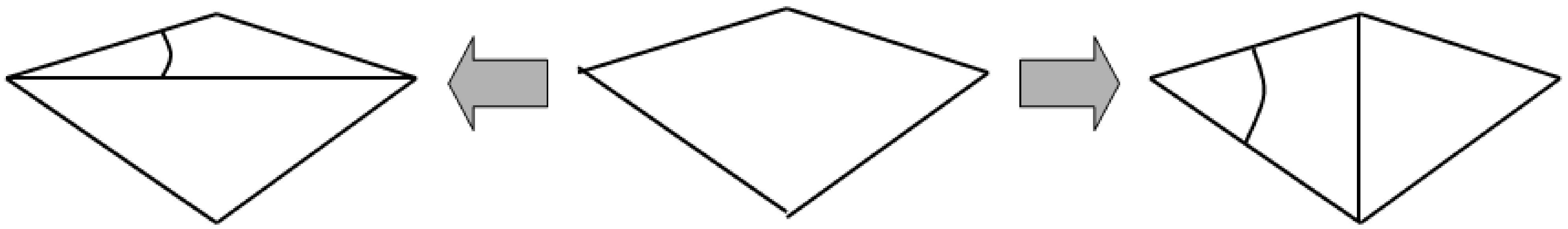
How to get "good" triangulations?



Circumcircle

Incircle

- For one set of points, there are different triangulations
  - Good ones and less optimal ones
- Good triangulations avoid long, thin triangles



Bad triangles

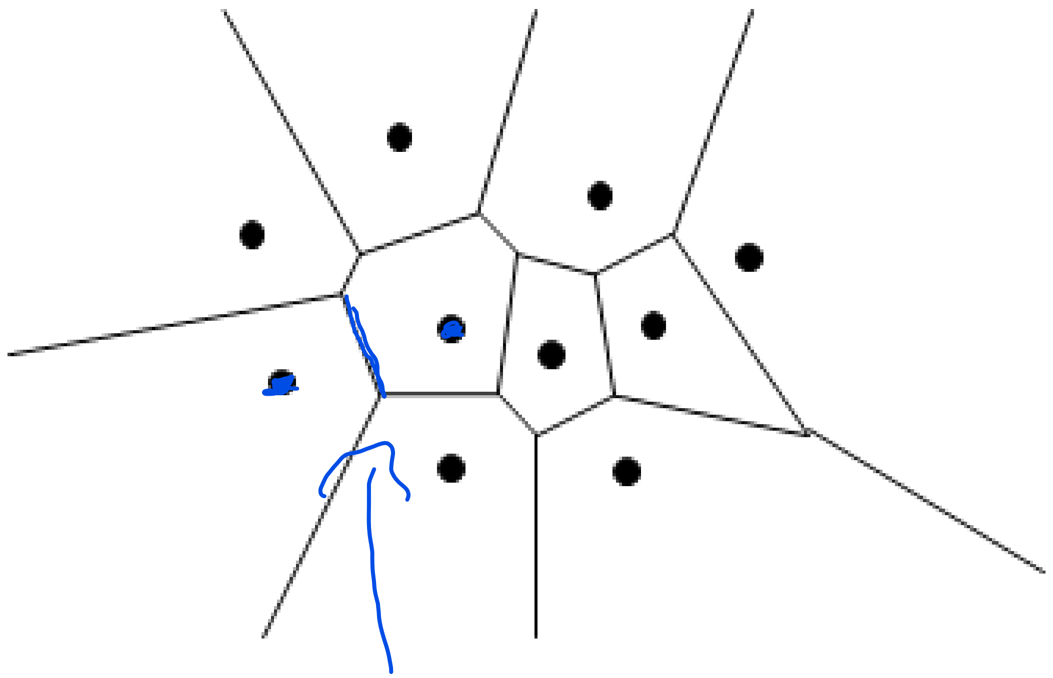
Good triangles

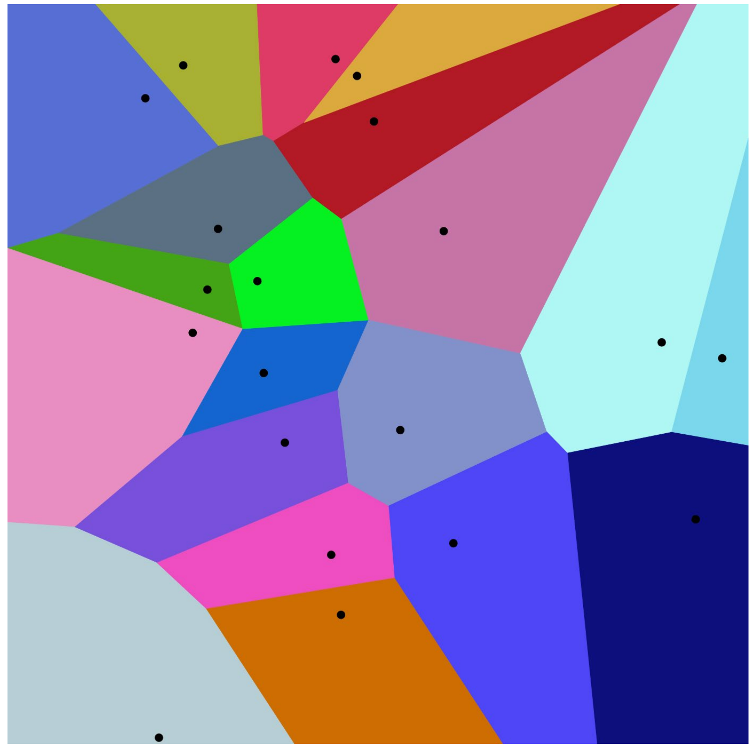
- Measure of quality: aspect ratio of triangles $\rho = \frac{r_{\text{incircle}}}{R_{\text{circumcircle}}} \rightarrow \max$

Triangulation

Voronoi diagram

- Around each sample point construct a region that is closer to that sample than to every other sample





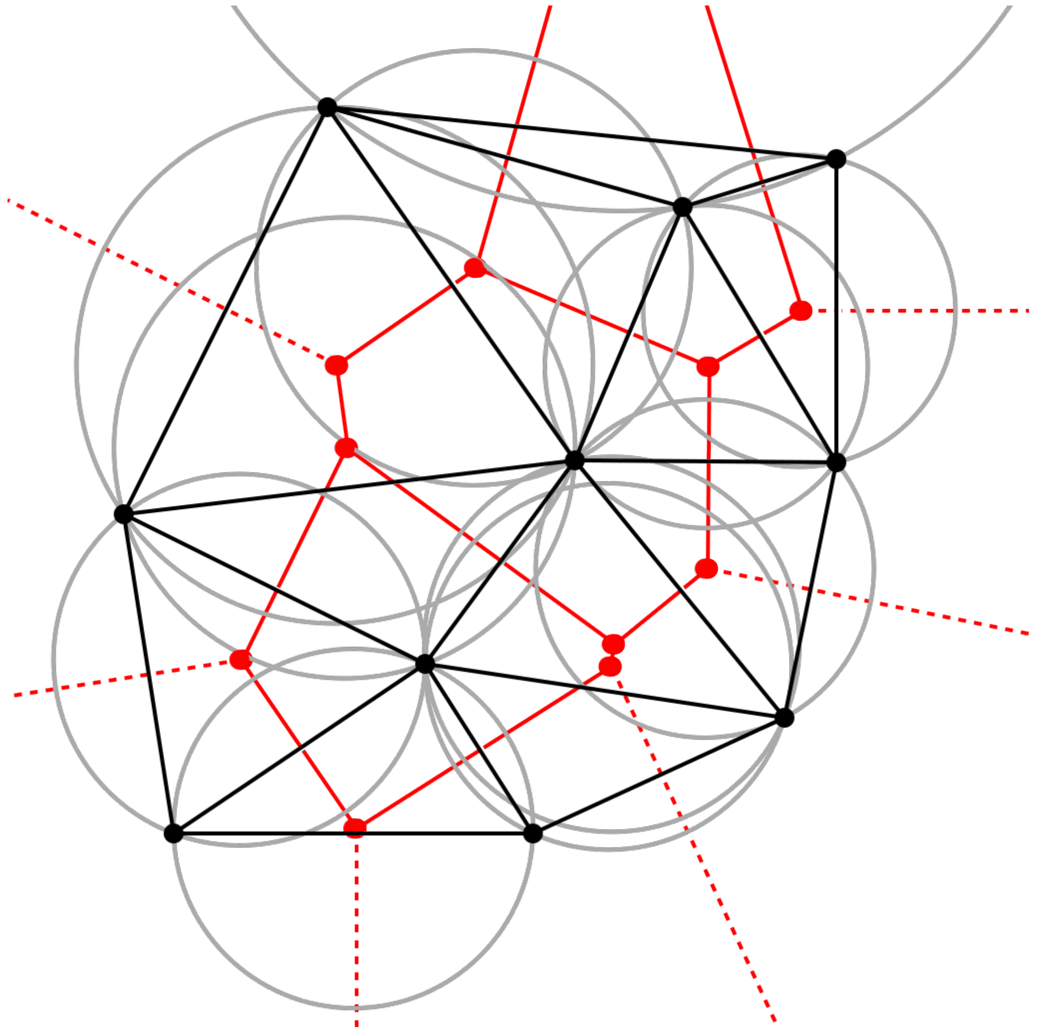
Alle Punkte auf einer Kante sind zu Sample Punkten gleich weit entfernt

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

Triangulation

Delaunay triangulation

- The vertices of the Voronoi diagram as circumcenters of triangles

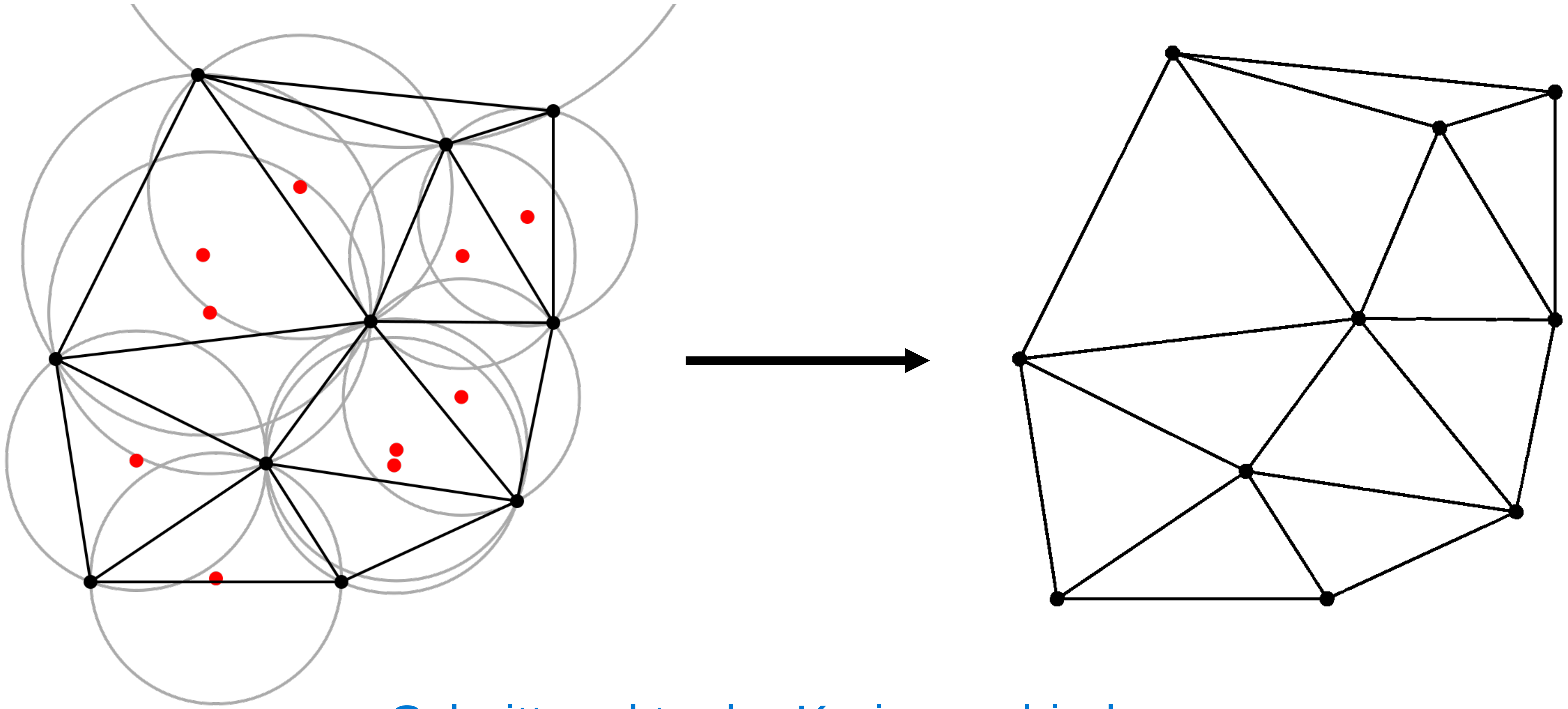


Source: https://en.wikipedia.org/wiki/Delaunay_triangulation

Triangulation

Delaunay triangulation

- The vertices of the Voronoi diagram as circumcenters of triangles

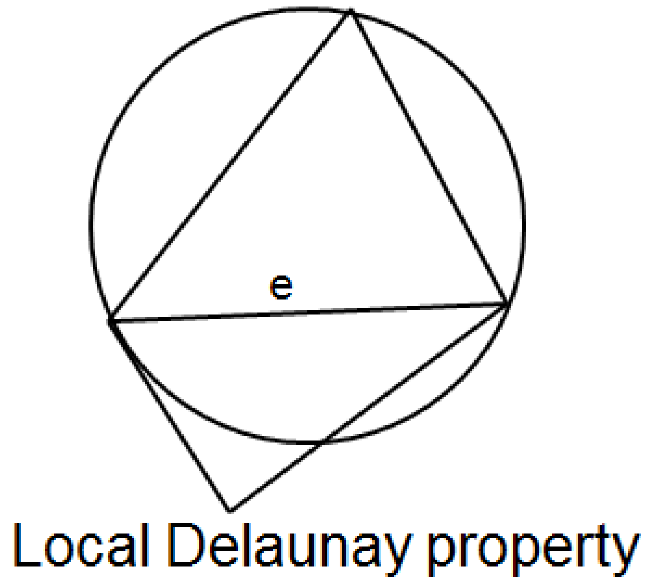


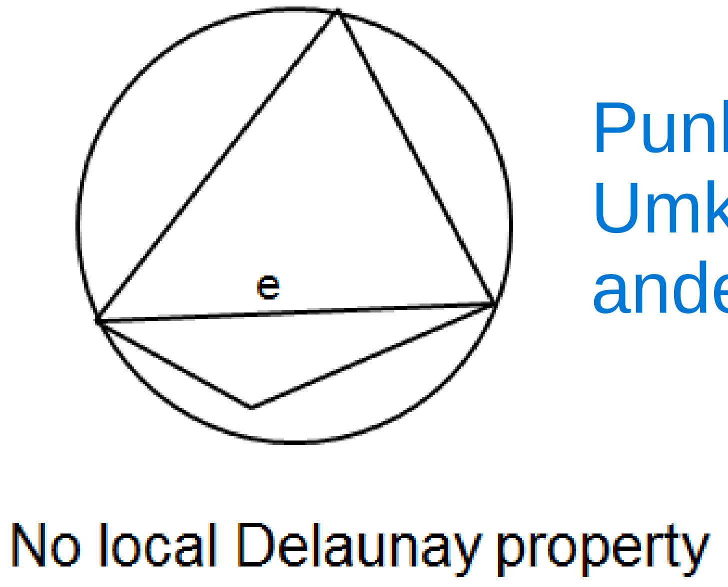
Source: https://en.wikipedia.org/wiki/Delaunay_triangulation

Schnittpunkte der Kreise verbinden

Triangulation

- Delaunay triangulation is the geometric dual of the Voronoi diagram
  - Maximizes the smallest angle and the aspect ratio of triangles
  - Triangles fulfill the "local Delaunay property"
    - For each edge, the perimeter of the adjacent triangle does not contain the 'other' vertex





Punkt liegt im
Umkreis des
anderen Dreiecks

ohm

Triangulation

Two-step algorithm: initial triangulation and edge flip

- Initial triangulation

```
- Sort points from left to right

- Construct initial triangle using first three vertices

- For (each point pi in vertex list) {
     Use last inserted point pi as starting point
     Take convex polygons of triangulation
     Find edge of minimal distance
     Triangulate edge + pi
  }
```

- Typically results in suboptimal triangulation

ohm

Triangulation

Two-step algorithm: initial triangulation and edge flip

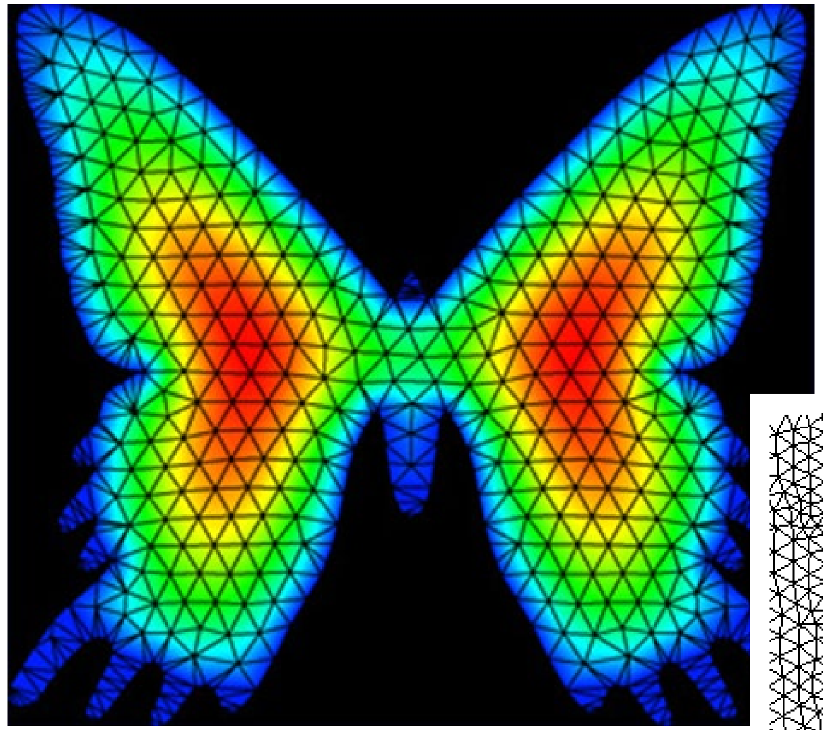
- Edge flip

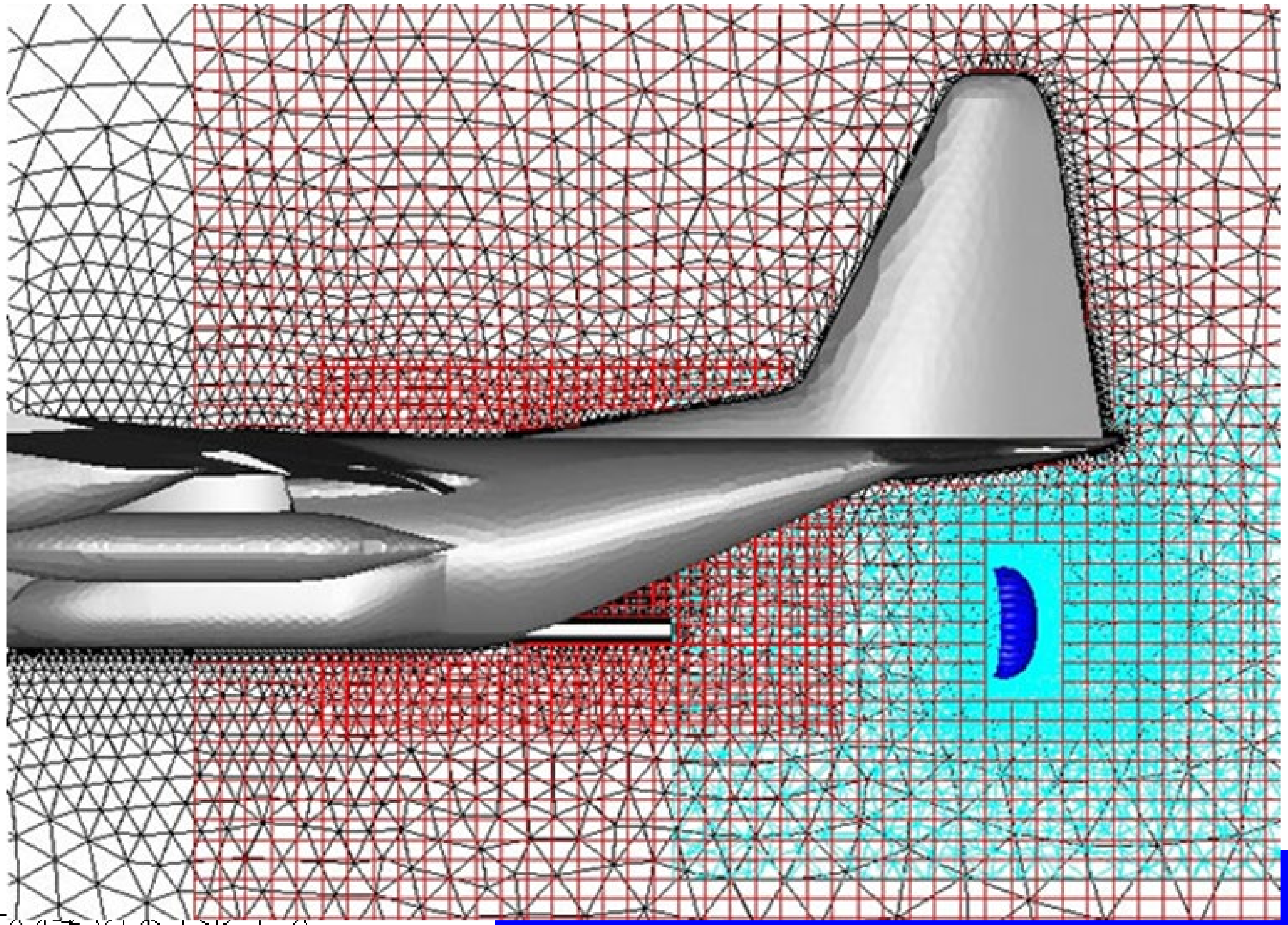
```
- Find initial (valid) triangulation
- Find all edges where local Delaunay property does not hold
- Mark these edges + push them onto stack

- While (stack not empty) {
    Pop edge from stack
    if (edge does no satisfy Delaunay property) {
      Flip edge
      Flip all neighboring edges that do
        not satisfy the local Delaunay property any more
    }
  }
```

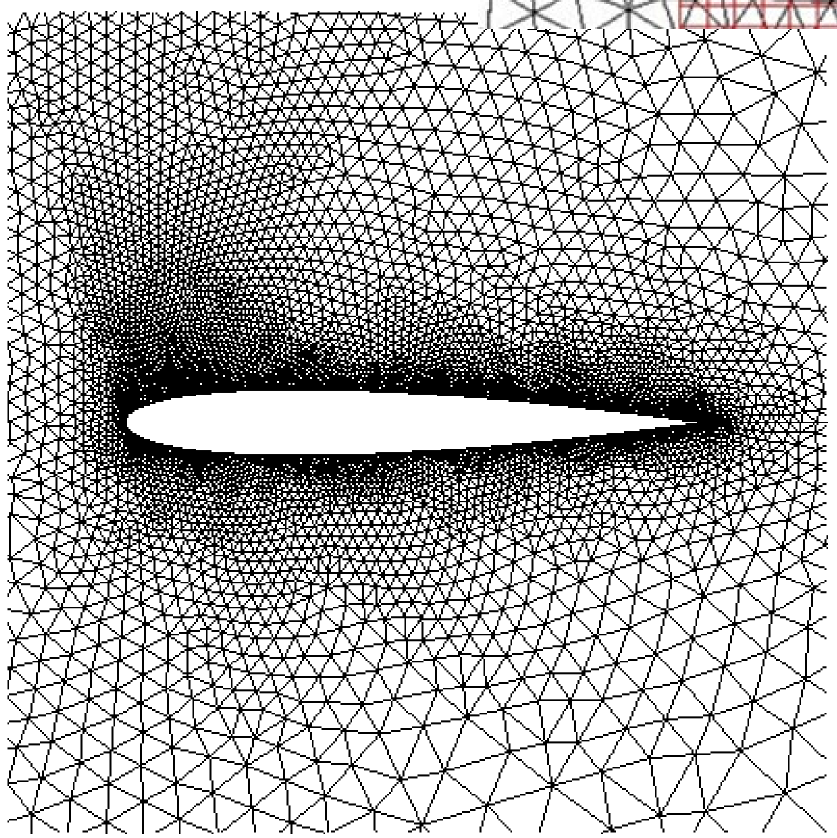
- Note: this algorithm terminates

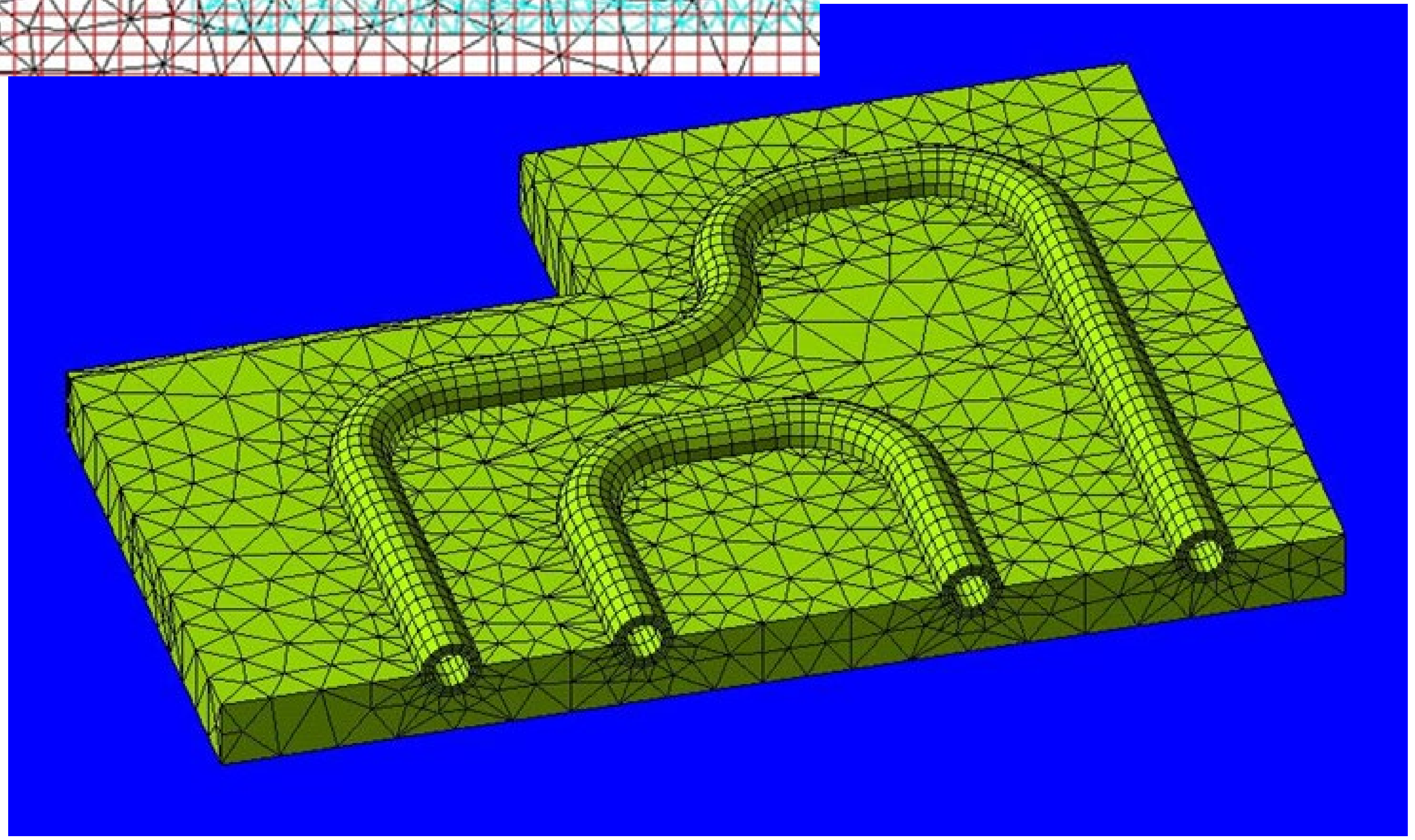
Triangulation

Examples









3.5.4 Differentiation on Grids

ohm

Differentiation on Grids

Problem

- Typically, discrete measured data are given
- We actually want to visualize continuous data
- Mathematical description
  - Scalar data: 1D-Scalar: $f : \mathbb{R} \mapsto \mathbb{R}$
    2D-Scalar: $g : \mathbb{R}^2 \mapsto \mathbb{R}$
    3D-Scalar: $h : \mathbb{R}^3 \mapsto \mathbb{R}$
  - Vector data: 2D/3D Vector: $F : \mathbb{R}^2 \mapsto \mathbb{R}^2$ or $G : \mathbb{R}^3 \mapsto \mathbb{R}^3$
  - Tensor: $I : \mathbb{R}^3 \mapsto Mat(3,3) = \mathbb{R}^{3x3}$
- Points of special interest
  - Locations where quantities *change rapidly* (**high variation**)
  - Changes are measured by **derivatives** (*differentials*)

Differentiation on Grids

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

Ableitung

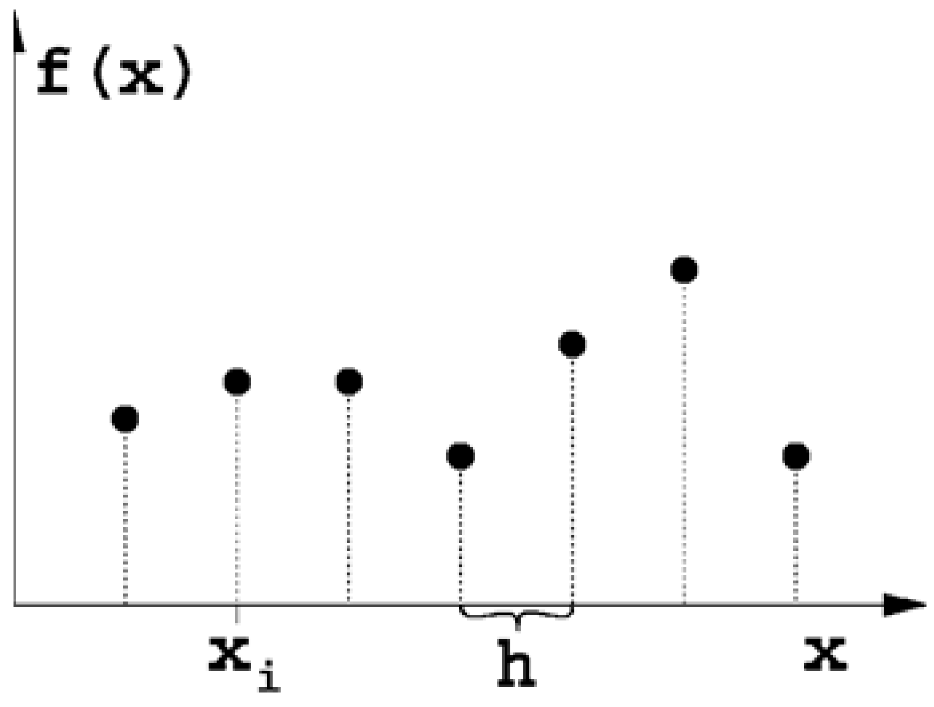
- Second approach
  - Replace differential by "finite differences"

$$f'(x) = \frac{\mathrm{d}f}{\mathrm{d}x} \longrightarrow \frac{\Delta f}{\Delta x}$$

Differentiation on Grids

1D uniform grids with grid size $h = \Delta x$

- Forward difference $$f'(x) = \frac{f(x_{i+1}) - f(x_i)}{h}$$
- Backward difference $$f'(x) = \frac{f(x_i) - f(x_{i-1})}{h}$$
- Central difference $$f'(x) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h}$$



- The error is $O(h)$ for forward/backward difference and $O(h^2)$ for central difference

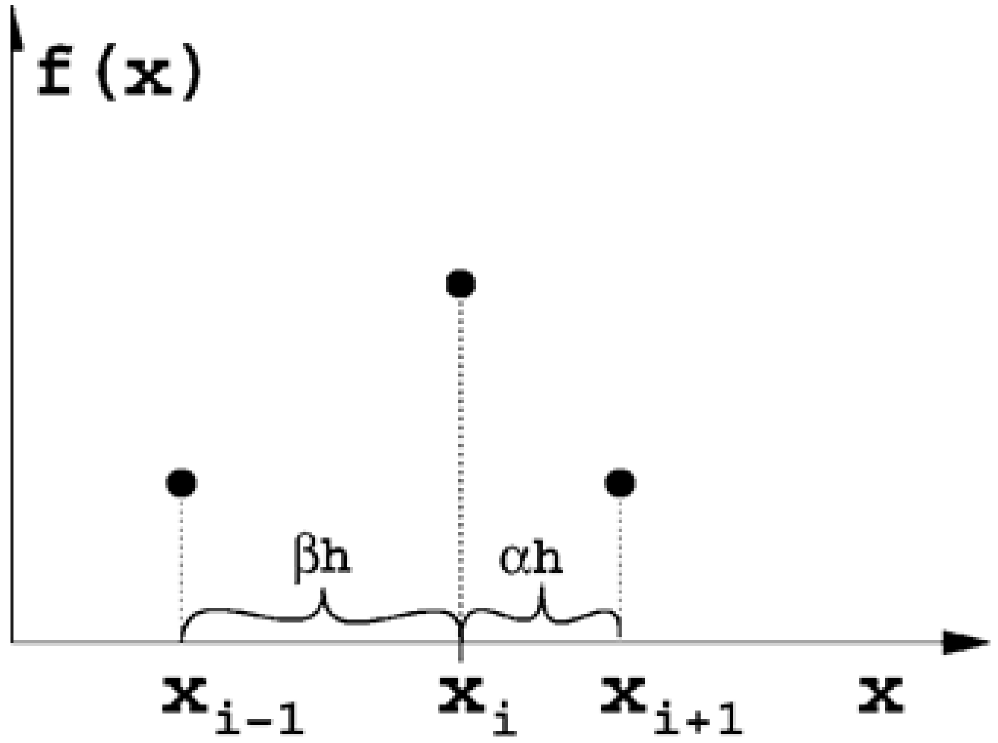
Differentiation on Grids

1D non-uniform grid

- Forward or backward difference is the same as for uniform grids
  - Note: $\boldsymbol{h} = \Delta\boldsymbol{x}$ is different in each direction, as spacing is non-uniform

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

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



Differentiation on Grids

1D non-uniform grid

- Central difference needs to consider the different spacing
  - From Taylor

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

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

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

  - Division by α^2 and β^2 eliminates the parts $\frac{h^2}{2} f''(x_i)$

ohm

Differentiation on Grids

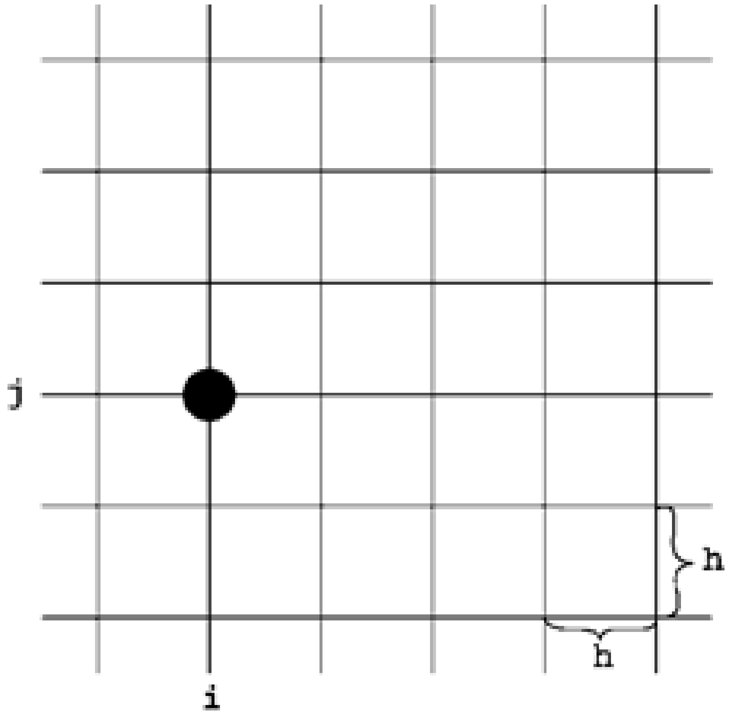
1D non-uniform grid

- Then, the final approximation of the derivative is

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

Differentiation on Grids

2D / 3D uniform or rectangular grids



- Partial derivatives $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

  Partielle Ableitung

  - Same as in univariate case along each coordinate axis

- Example
  - Gradient on a 3D uniform grid with size h $\mathrm{grad} f = \nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{pmatrix} = \begin{pmatrix} \frac{f_{i+1,j,k} - f_{i-1,j,k}}{2h} \\ \frac{f_{i,j+1,k} - f_{i,j-1,k}}{2h} \\ \frac{f_{i,j,k+1} - f_{i,j,k-1}}{2h} \end{pmatrix}$

Differentiation on Grids

Unstructured grids

- Dilemma
  - There is no generalization of the difference quotient!

- Solution (1)
  - Interpolation / approximation with function to differentiate
    - Easiest case: linear interpolation in each cell leads to one gradient per cell

- Solution (2)
  - Resampling into structured grid
    - Interpolation introduces additional error