King Abdullah University of Science and Technology KAUST

CS 247 – Scientific Visualization
Lecture 4: The Visualization Pipeline; Data Representation, Pt. 2

Markus Hadwiger, KAUST

Reading Assignment #2 (until Feb 8)

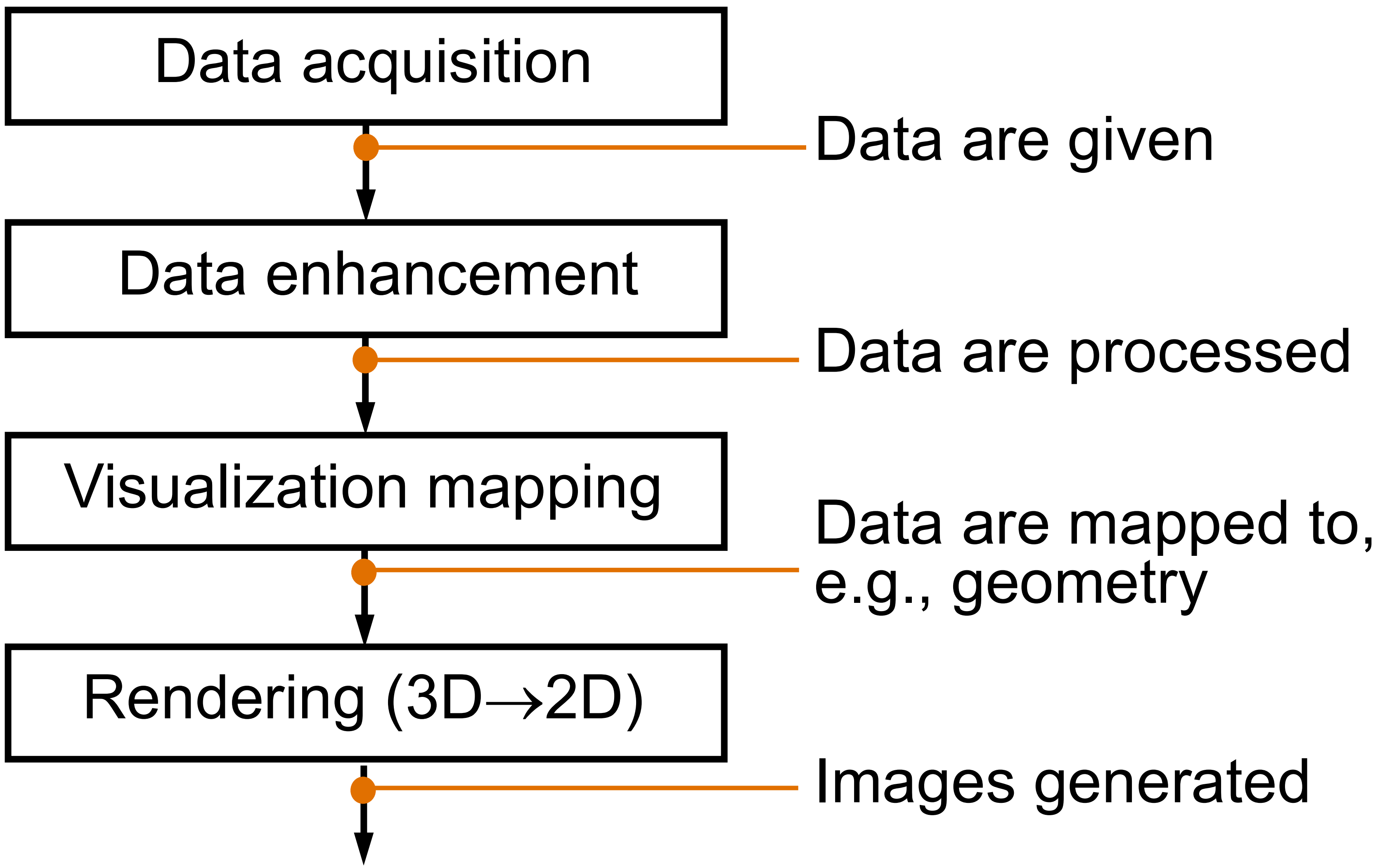
Read (required):

- Data Visualization book, finish Chapter 2
- Data Visualization book, Chapter 3 until 3.5 (inclusive)
- Data Visualization book, Chapter 4 until 4.1 (inclusive)

- Continue familiarizing yourself with OpenGL if you do not know it !

The Visualization Pipeline

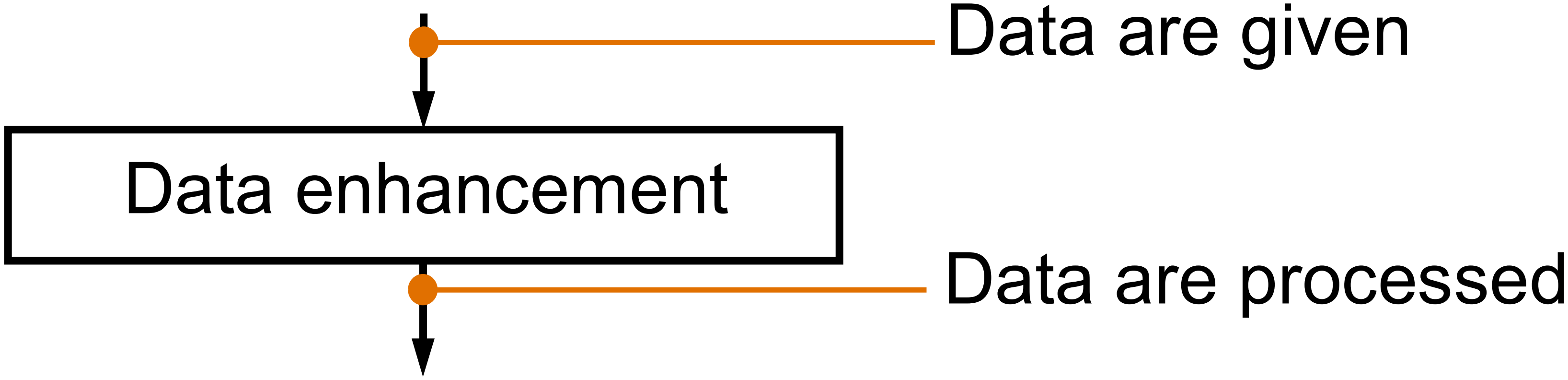
The Visualization Pipeline – Overview



The Visualization Pipeline – Stage 1

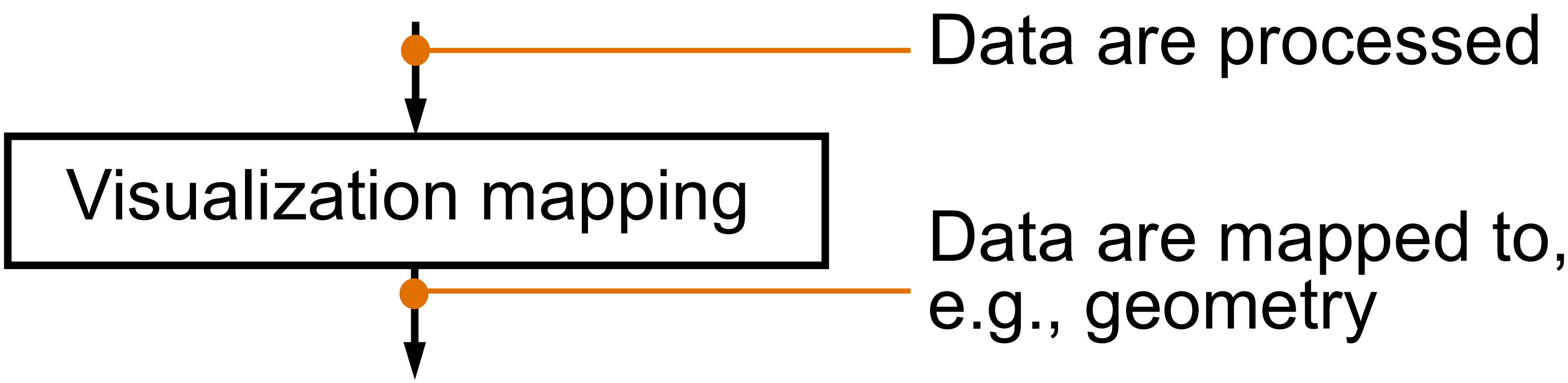
- Measurements, e.g., CT/MRI
- Simulation, e.g., flow simulation
- Modeling, e.g., game theory

The Visualization Pipeline – Stage 2



- Filtering, e.g, smoothing (de-noising, …)
- Resampling, e.g., on a different-resolution grid
- Data derivation, e.g., gradients, curvature
- Data interpolation, e.g., linear, cubic, …

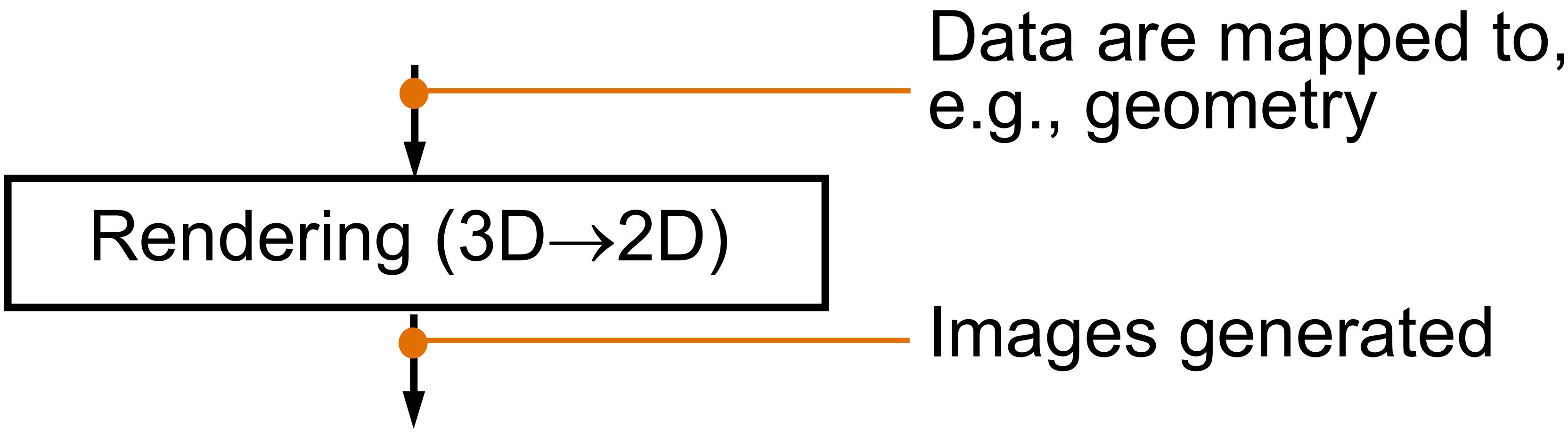
The Visualization Pipeline – Stage 3



Make data “renderable”

- Iso-surface calculation
- Glyphs, icons determination
- Graph-layout calculation
- Voxel attributes: color, transparency, …

The Visualization Pipeline – Stage 4



Rendering = image generation with computer graphics

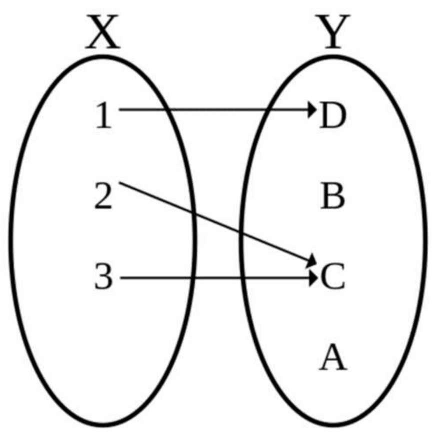
- Visibility calculation
- Illumination
- Compositing (combine transparent objects, …)
- Animation

Data == Functions

Mathematical Functions

Associates every element of a set (e.g., X) with *exactly one* element of another set (e.g., Y)

Maps from domain (X) to codomain (Y)



$$f : \mathbb{R}^n \to \mathbb{R}^m$$
$$x \mapsto f(x)$$

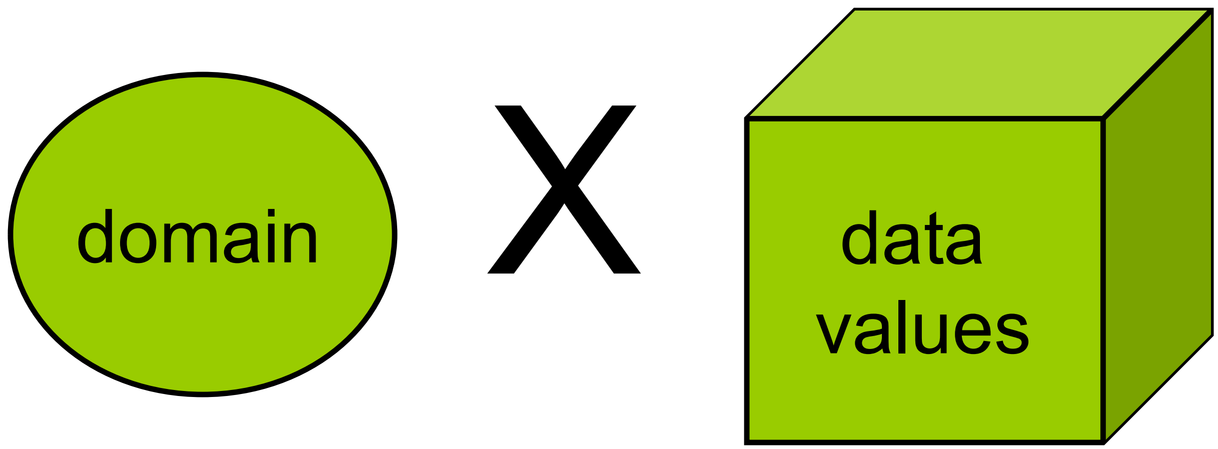
Also important: *range/image*; *preimage*; continuity, differentiability, dimensionality, ...

Graph of a function (mathematical definition):

$$G(f) := \{(x, f(x)) | x \in \mathbb{R}^n\} \subset \mathbb{R}^n \times \mathbb{R}^m \simeq \mathbb{R}^{n+m}$$

Data Representation

$\boldsymbol{R}^n$ $\boldsymbol{R}^m$



independent variables *dependent variables*

scientific data $\subseteq \boldsymbol{R}^{n+m}$

Example: Scalar Fields

2D scalar field

$$f : \mathbb{R}^2 \to \mathbb{R}$$
$$x \mapsto f(x)$$

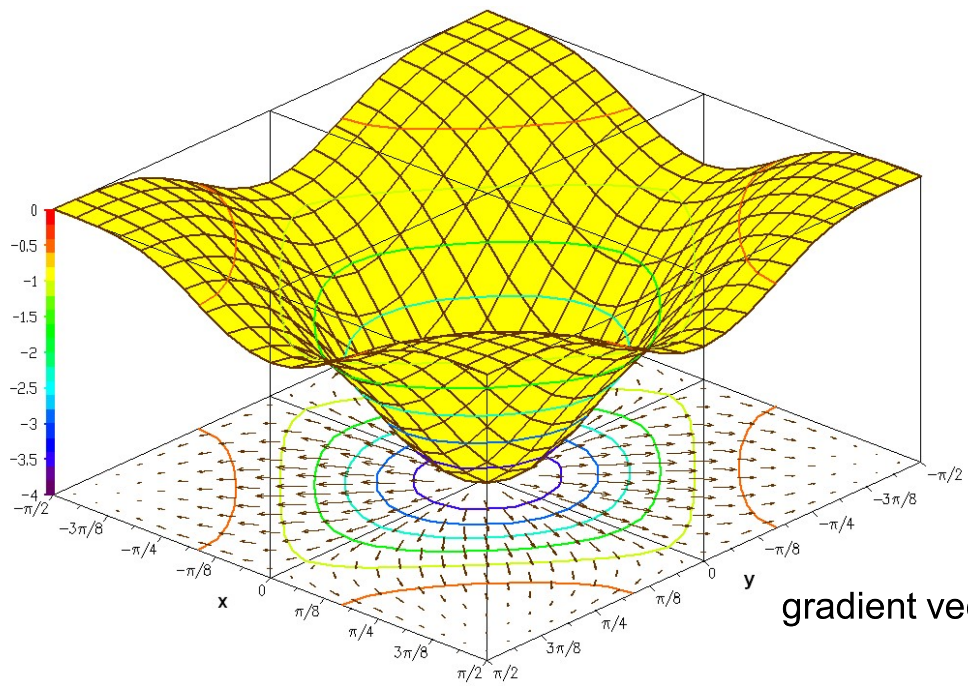
Graph: $G(f) := \{(x, f(x)) | x \in \mathbb{R}^2\} \subset \mathbb{R}^2 \times \mathbb{R} \simeq \mathbb{R}^3$

pre-image

$$S(c) := f^{-1}(c)$$

iso-contour

$$(\nabla f \neq 0)$$



gradient vector field ∇f

Example: Scalar Fields

3D scalar field

$$f: \mathbb{R}^3 \to \mathbb{R}$$
$$x \mapsto f(x)$$

Graph: $G(f) := \{(x, f(x)) | x \in \mathbb{R}^3\} \subset \mathbb{R}^3 \times \mathbb{R} \simeq \mathbb{R}^4$

pre-image

$$S(c) := f^{-1}(c)$$

iso-surface

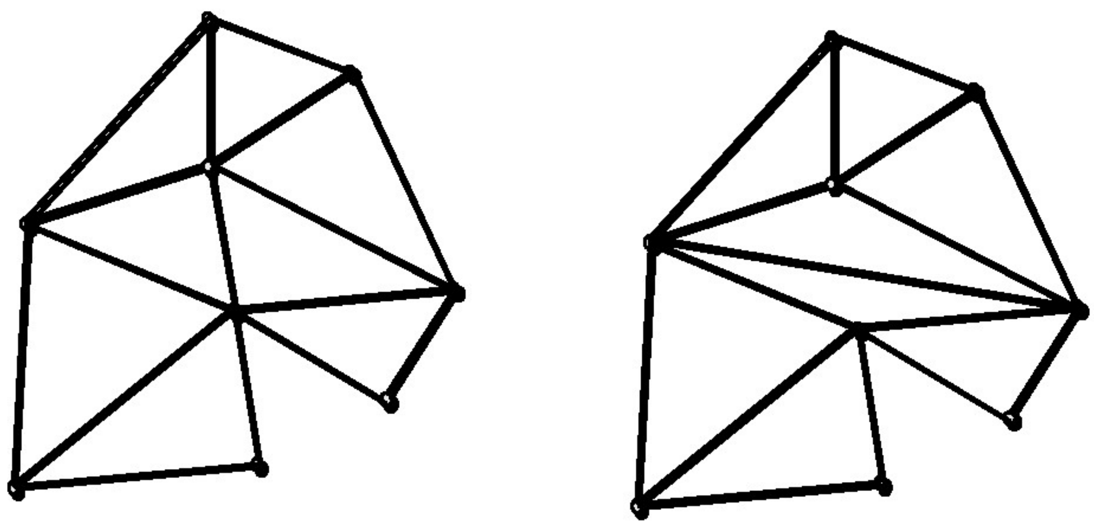
$$(\nabla f \neq 0)$$

?

Sampled Functions and Data Structures

Data Structures

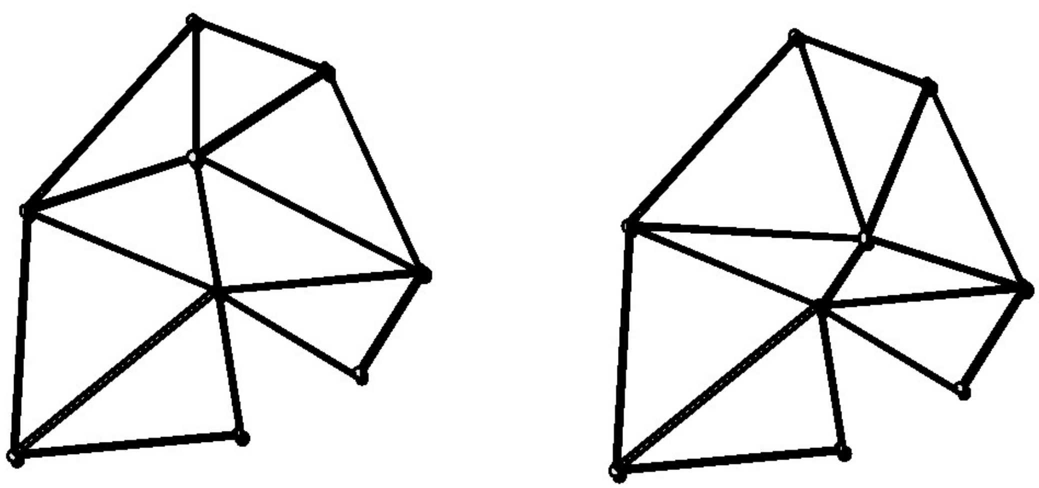
- Topology
  - Properties of geometric shapes that remain unchanged even when under distortion



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

Data Structures

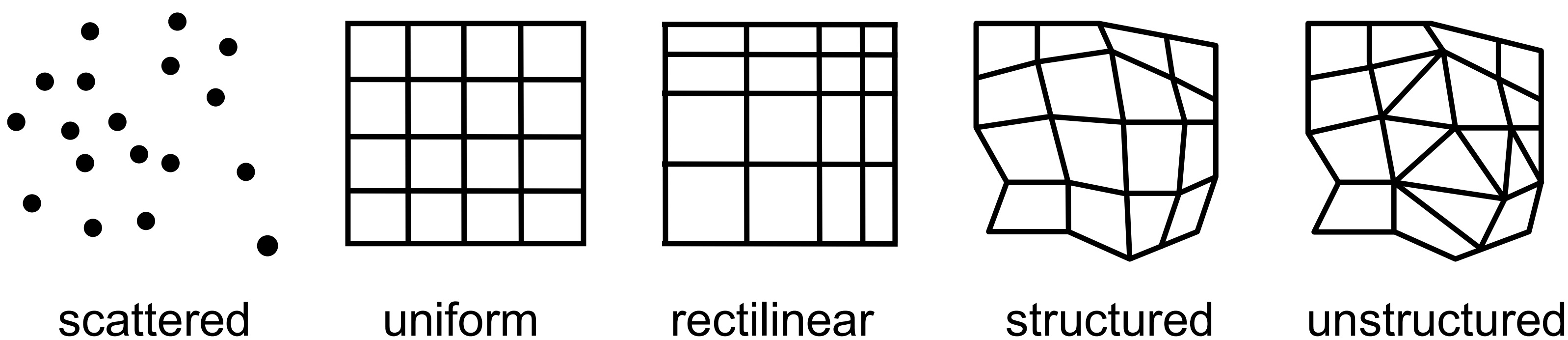
- Topologically equivalent
  - Things that can be transformed into each other by stretching and squeezing, without tearing or sticking together bits which were previously separated



topologically equivalent

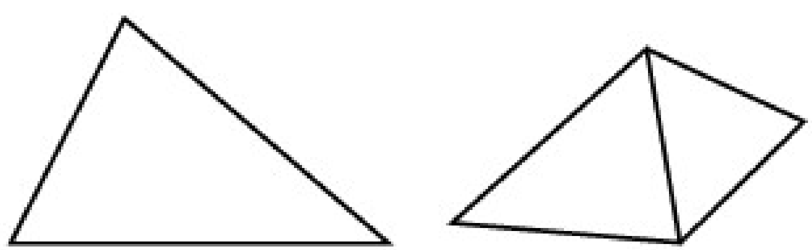
Data Structures

- Grid types
  - Grids differ substantially in the cells (basic building blocks) they are constructed from and in the way the topological information is given

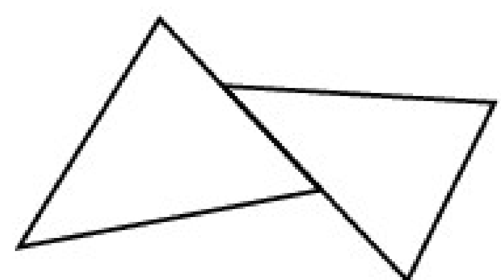


Data Structures

- An *n*-simplex
  - The convex hull of $n + 1$ affinely independent points
  - Lives in R^m , with $n \leq m$
  - 0: points, 1: lines, 2: triangles, 3: tetrahedra
- Partitions via simplices are called triangulations
- Simplical complex *C* is a collection of simplices with:
  - Every face of an element of *C* is also in *C*
  - The intersection of two elements of *C* is empty or it is a face of both elements
- Simplical complex is a space with a triangulation



Simplical complexes

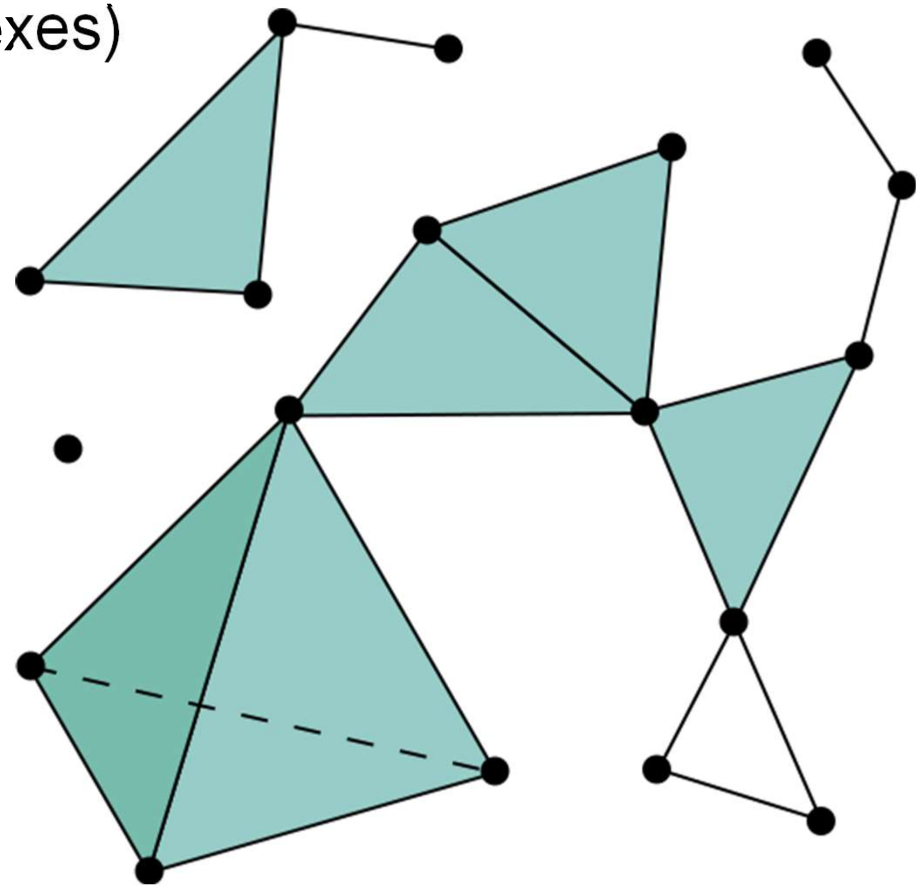


Not a simplical complex

Data Structures

- Simplicial complexes can be of mixed dimensions up to ≤ n
(except if "pure" complexes)

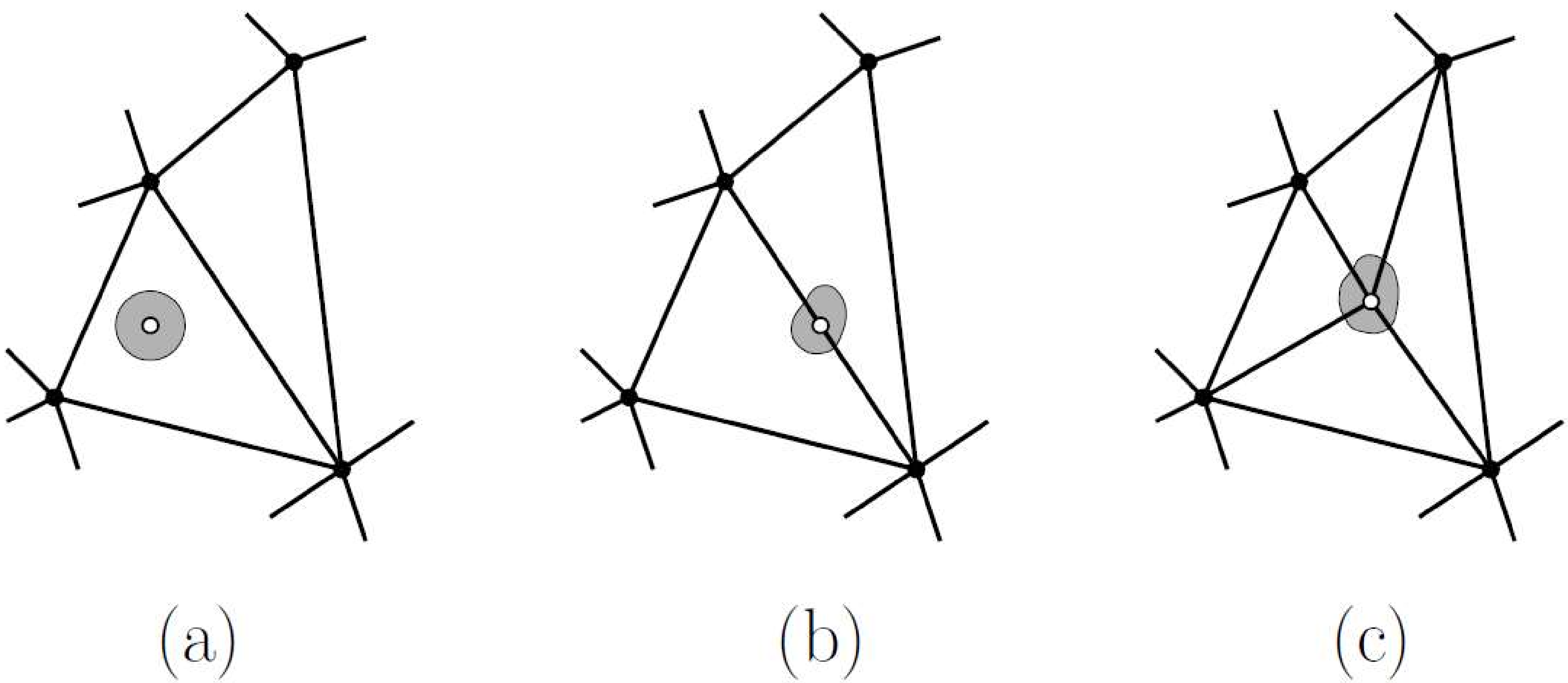
- Example:
Simplicial
3-complex



[Wikipedia.org]

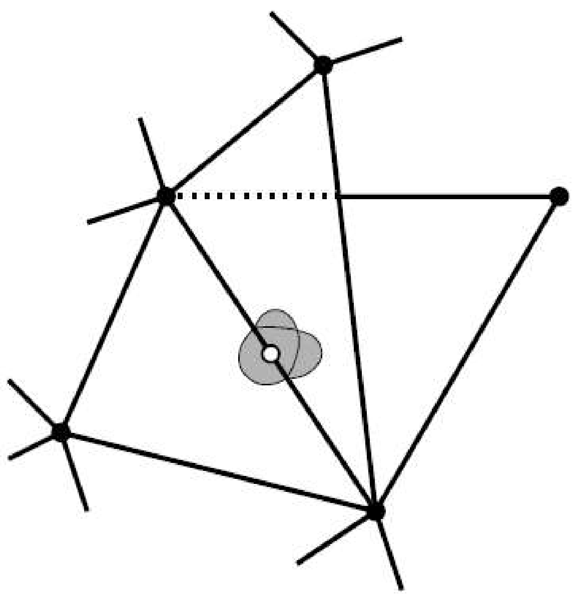
Data Structures

- 2-manifold meshes: neighborhood is 2-dimensional topological disc (or half disc for manifolds with boundary)

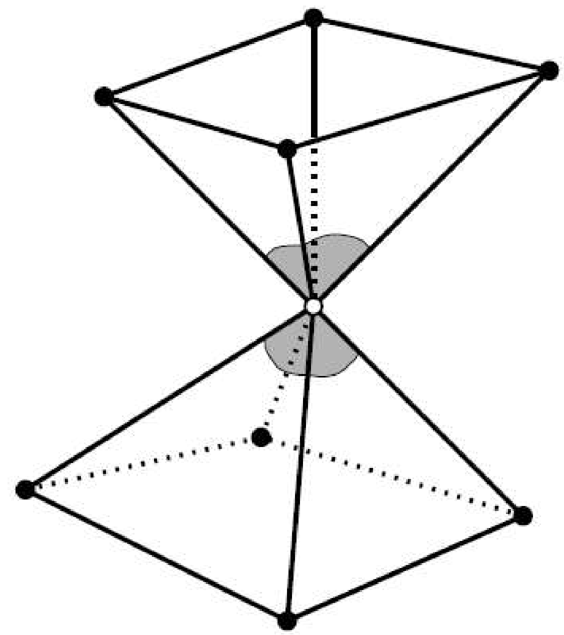


Data Structures

- Non-manifold meshes



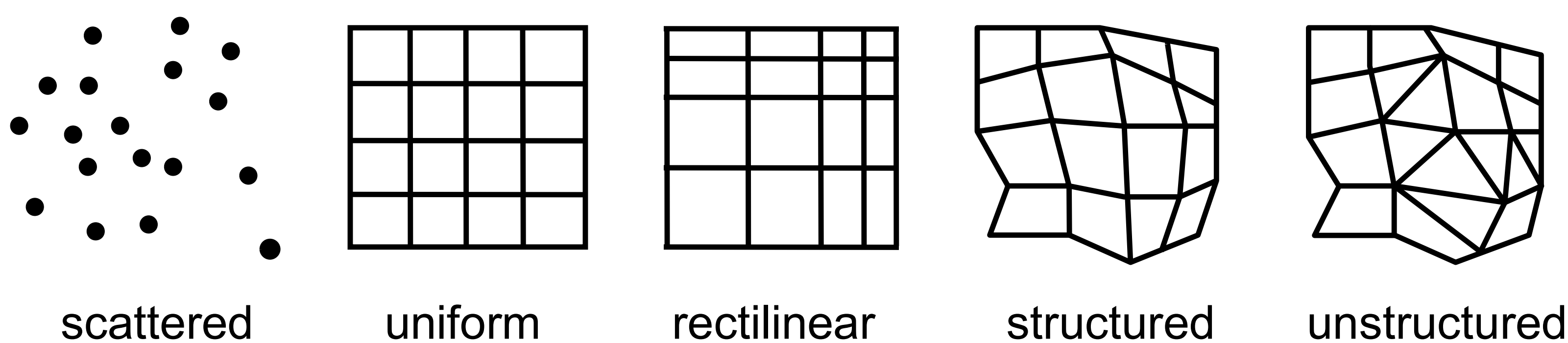
(d)



(e)

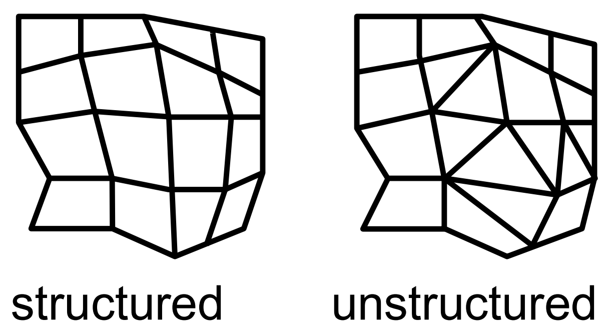
Data Structures

- Grid types
  - Grids differ substantially in the cells (basic building blocks) they are constructed from and in the way the topological information is given

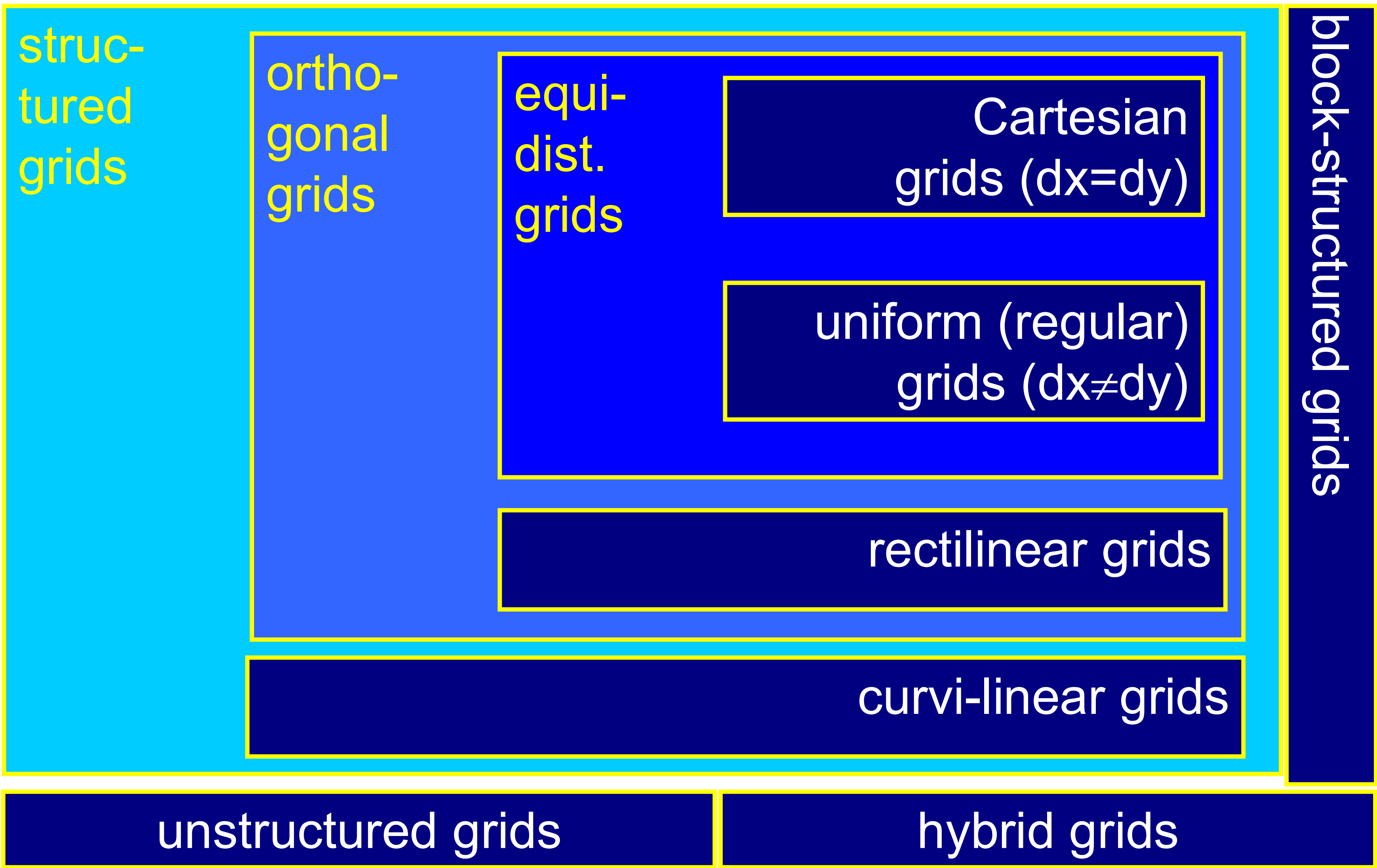


Data Structures

- Structured and unstructured grids can be distinguished by the way the elements or cells meet
- Structured grids
  - Have a regular topology and regular / irregular geometry
- Unstructured grids
  - Have irregular topology and geometry



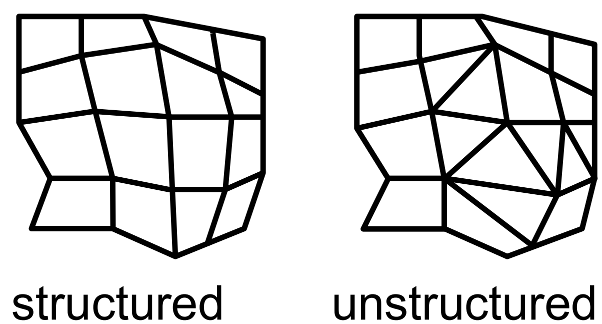
Grid Types - Overview



Structured Grids

Data Structures

- Characteristics of structured grids
  - Easier to compute with
  - Often composed of sets of connected parallelograms (hexahedra), with cells being equal or distorted with respect to (non-linear) transformations
  - May require more elements or badly shaped elements in order to precisely cover the underlying domain
  - Topology is represented implicitly by an *n*-vector of dimensions
  - Geometry is represented explicitly by an array of points
  - Every interior point has the same number of neighbors



Data Structures

- Characteristics of structured grids
  - Structured grids can be stored in a 2D / 3D array
  - Arbitrary samples can be directly accessed by indexing a particular entry in the array
  - Topological information is implicitly coded
    - Direct access to adjacent elements
  - Cartesian, uniform, and rectilinear grids are necessarily convex
  - Their visibility ordering of elements with respect to any viewing direction is given implicitly
  - Their rigid layout prohibits the geometric structure to adapt to local features
  - Curvilinear grids reveal a much more flexible alternative to model arbitrarily shaped objects
  - However, this flexibility in the design of the geometric shape makes the sorting of grid elements a more complex procedure

Data Structures

- Typical implementation of structured grids

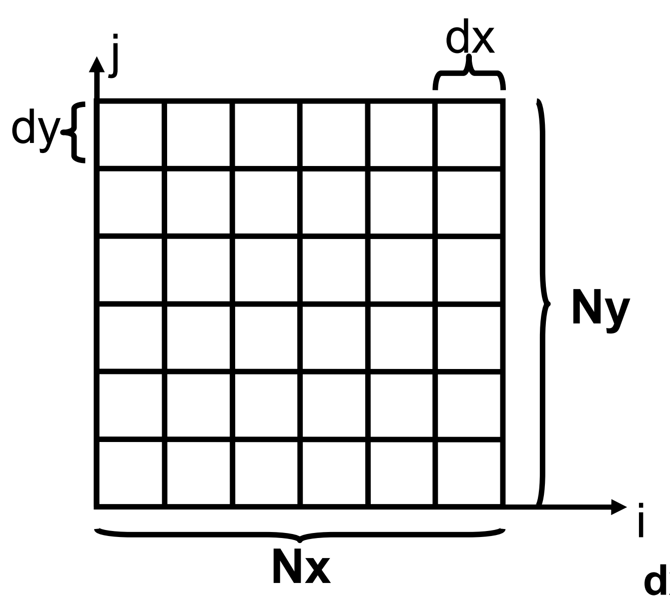
```
DataType *data = new DataType [Nx * Ny * Nz ];
val = data[ i + j * Nx + k * ( Nx * Ny ) ];
```

… code for geometry …

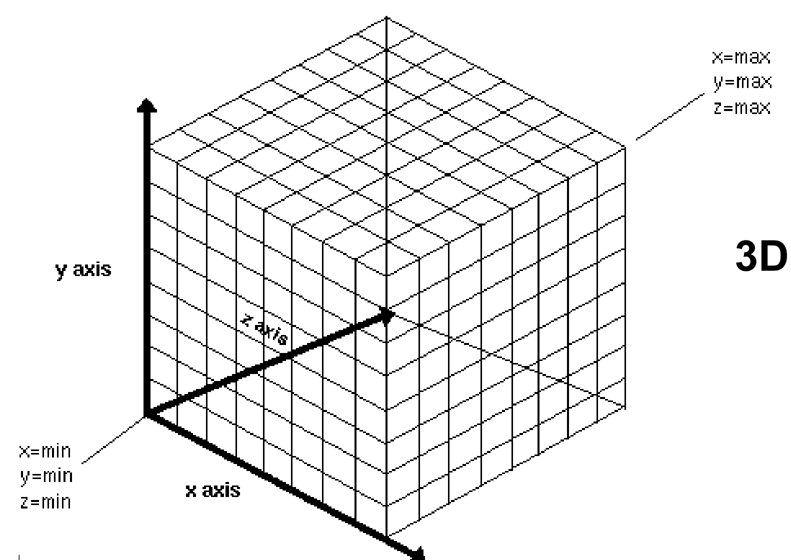
Data Structures

- Cartesian or equidistant grids
  - Structured grid
  - Cells and points are numbered sequentially with respect to increasing X, then Y, then Z, or vice versa
  - Number of points = Nx•Ny•Nz
  - Number of cells = (Nx-1)•(Ny-1)•(Nz-1)

2D



dx = dy = dz



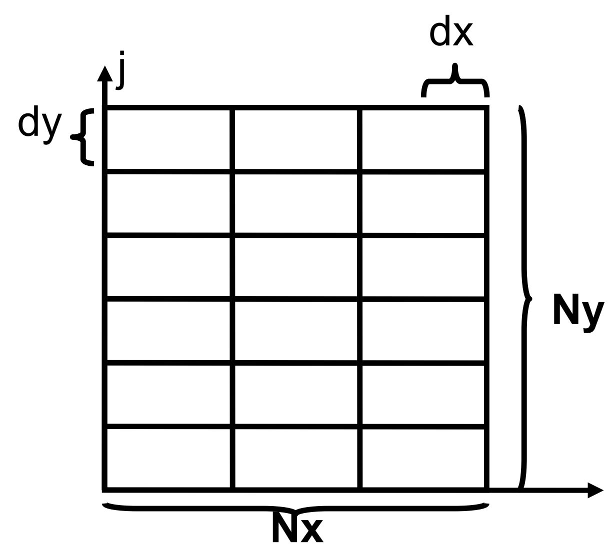
3D

Data Structures

- Cartesian grids
  - Vertex positions are given implicitly from [i,j,k]:
    - P[i,j,k].x = origin_x + i • dx
    - P[i,j,k].y = origin_y + j • dy
    - P[i,j,k].z = origin_z + 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
  - Global index allows for linear storage scheme
    - Wrong access pattern might destroy cache coherence

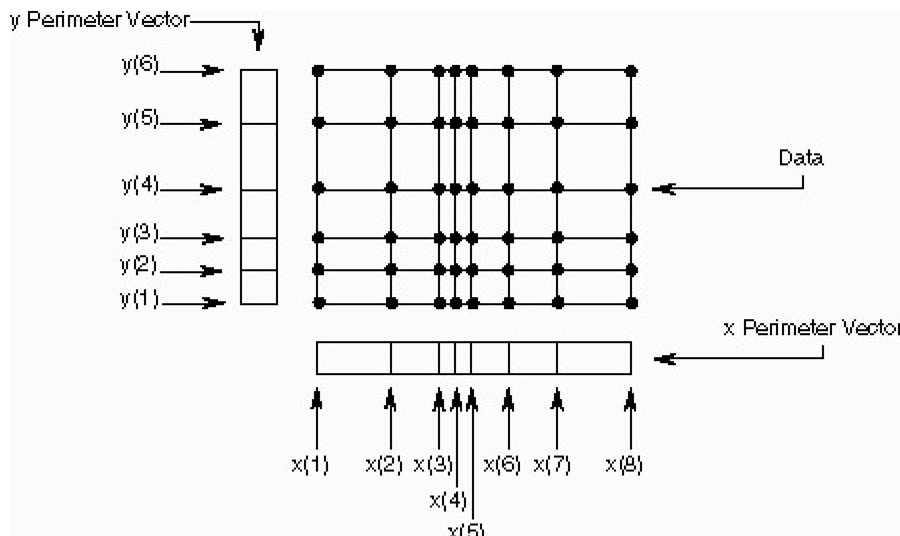
Data Structures

- Uniform grids
  - Similar to Cartesian grids
  - Consist of equal cells but with different resolution in at least one dimension ($dx \neq dy$ ($\neq dz$))
  - Spacing between grid points is constant in each dimension → same indexing scheme as for Cartesian grids
  - Most likely to occur in applications where the data is generated by a 3D imaging device providing different sampling rates in each dimension
  - Typical example: medical volume data consisting of slice images
    - Slice images with square pixels ($dx = dy$)
    - Larger slice distance ($dz > dx = dy$)



Data Structures

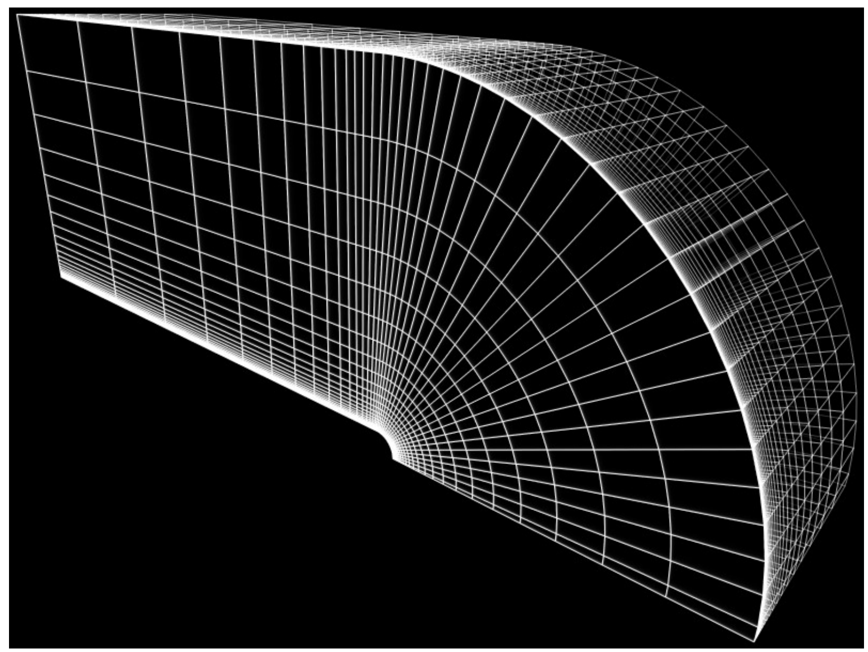
- Rectilinear grids
  - Topology is still regular but irregular spacing between grid points
    - Non-linear scaling of positions along either axis
    - Spacing, x_coord[L], y_coord[M], z_coord[N], must be stored explicitly
  - Topology is still implicit



(2D perimeter lattice:
rectilinear grid in IRIS Explorer)

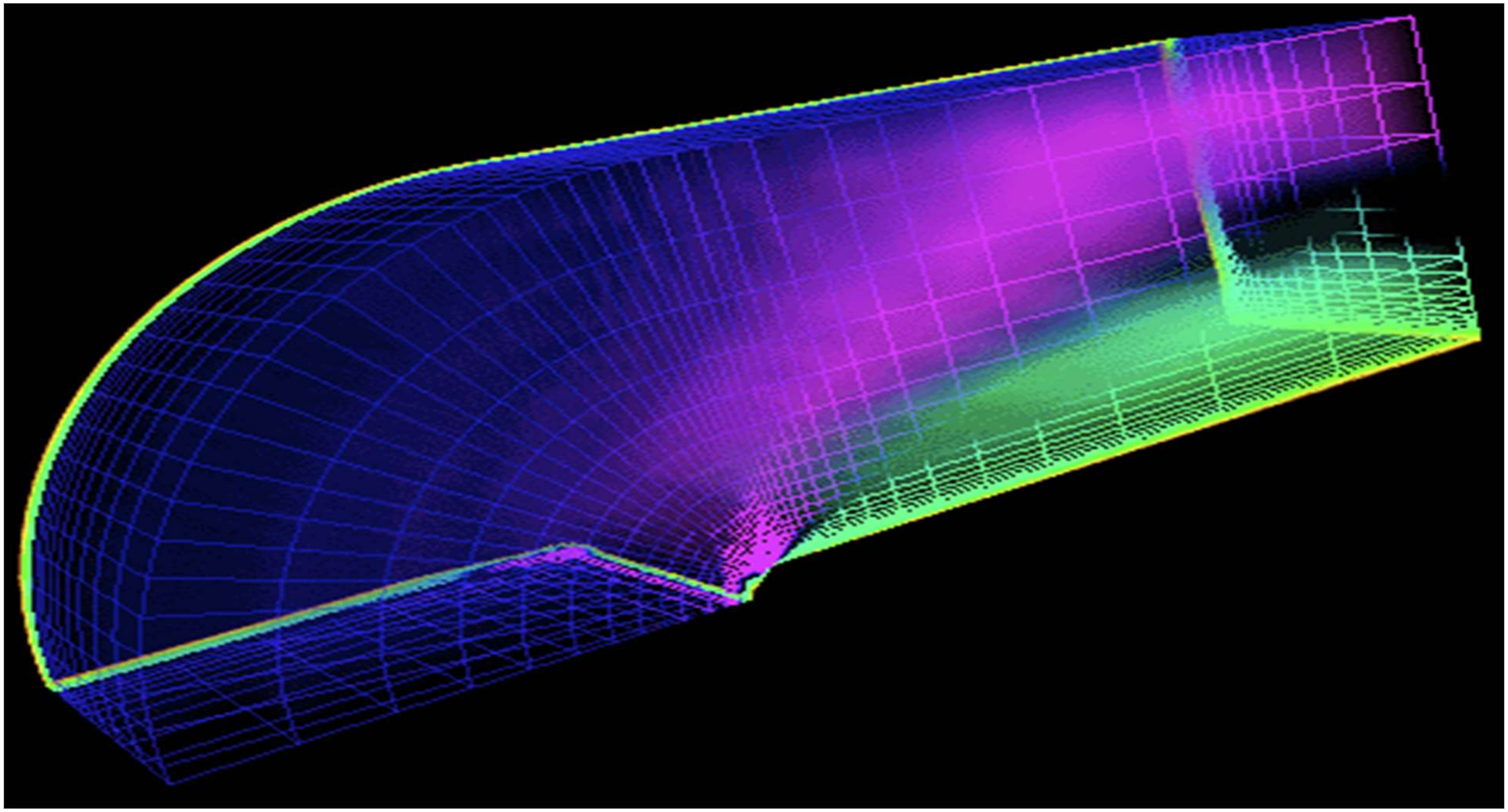
Data Structures

- Curvilinear grids
  - Topology is still regular but irregular spacing between grid points
    - Positions are non-linearly transformed
  - Topology is still implicit, but vertex positions are explicitly stored
    - x_coord[L,M,N]
    - y_coord[L,M,N]
    - z_coord[L,M,N]
  - Geometric structure might result in concave grids



Data Structures

- Curvilinear grids



Thank you.

Thanks for material

- Helwig Hauser
- Eduard Gröller
- Daniel Weiskopf
- Torsten Möller
- Ronny Peikert
- Philipp Muigg
- Christof Rezk-Salama