King Abdullah University of Science and Technology KAUST

CS 247 – Scientific Visualization
Lecture 7: Data Representation, Pt. 4

Markus Hadwiger, KAUST

Reading Assignment #4 (until Feb 26)

Read (required):

- Real-Time Volume Graphics book, Chapter 5 until 5.4 inclusive (*Terminology, Types of Light Sources, Gradient-Based Illumination, Local Illumination Models*)
- Paper:
  Marching Cubes: A high resolution 3D surface construction algorithm,
  Bill Lorensen and Harvey Cline, ACM SIGGRAPH 1987
  [> 18,600 citations and counting...]

  https://dl.acm.org/doi/10.1145/37402.37422

Read (optional):

- Paper:
  Flying Edges, William Schroeder et al., IEEE LDAV 2015

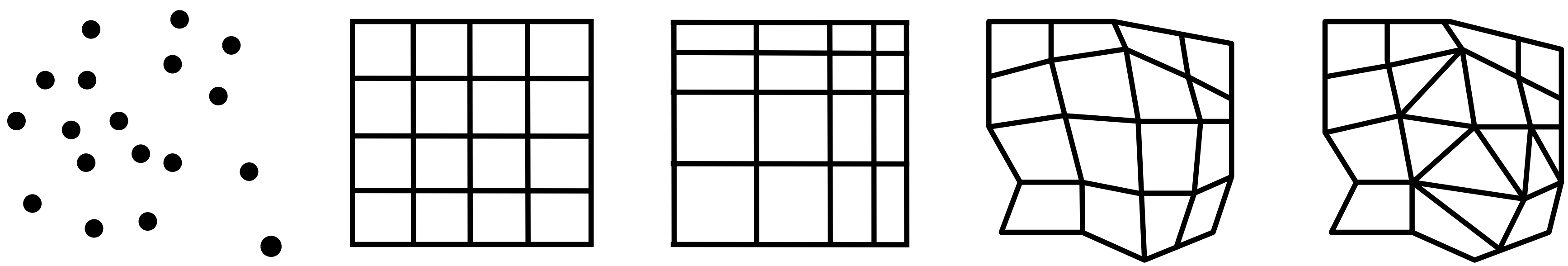
  https://ieeexplore.ieee.org/document/7348069

Programming Assignments Schedule (tentative)

Assignment 0:	Lab sign-up: join discord, setup github account + get repo	until	Feb 5
	Basic OpenGL example		
Assignment 1:	Volume slice viewer	until	Feb 18
Assignment 2:	Iso-contours (marching squares)	until	Mar 2
Assignment 3:	Iso-surface rendering (marching cubes)	until	**Mar 23**
Assignment 4:	Volume ray-casting, part 1	until	**Apr 13**
	Volume ray-casting, part 2	until	**Apr 20**
Assignment 5:	Flow vis, part 1 (hedgehog plots, streamlines, pathlines)	until	**May 4**
Assignment 6:	Flow vis, part 2 (LIC with color coding)	until	**May 14**

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



scattered uniform rectilinear structured unstructured

Grid Types - Overview

struc-
tured
grids

ortho-
gonal
grids

equi-
dist.
grids

Cartesian
grids (dx=dy)

uniform (regular)
grids (dx≠dy)

rectilinear grids

curvi-linear grids

block-structured grids

unstructured grids

hybrid grids

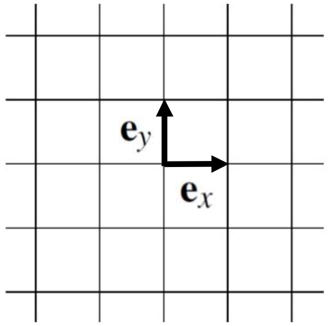
Interlude: Naming / Definition Caveats

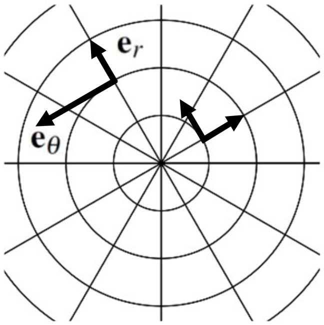
Beware of different naming conventions / different definitions

Example:

- On the previous slide, we used the term “orthogonal grid” in a simple, “global” way for the entire grid, i.e., different types of rectilinear grids, …
- In differential geometry, an orthogonal coordinate system is defined pointwise, i.e., a curvilinear grid with orthogonal basis vectors at each point is orthogonal

In differential geometry, both of these are orthogonal (in our context, the right one is not):

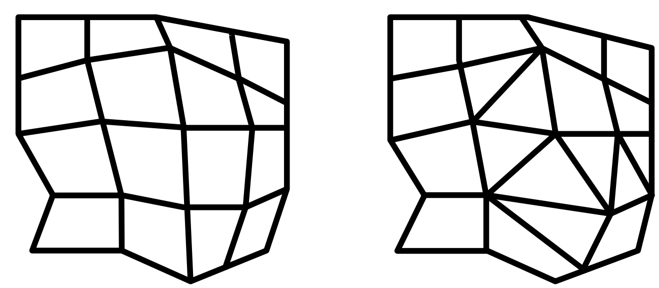




Structured Grids

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



structured unstructured

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

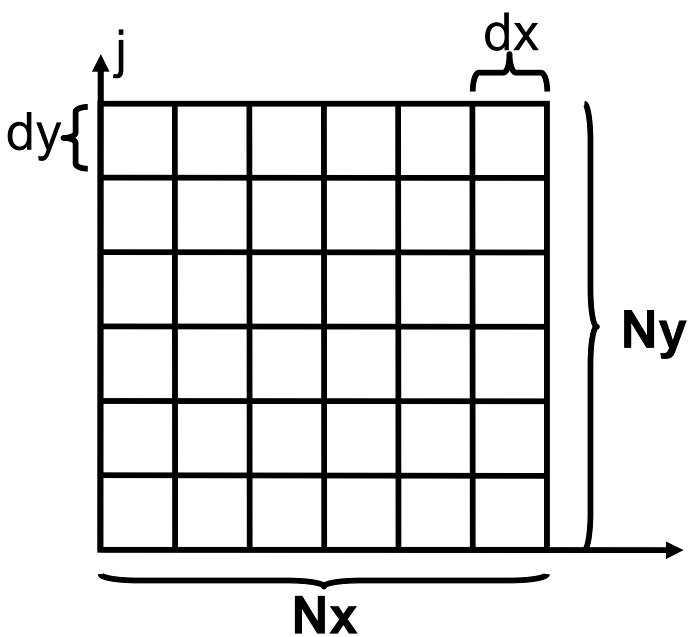
```
DataType *data = new DataType [Nx * Ny * Nz ];
val = data[ i + j * Nx + k * ( Nx * Ny ) ];
```

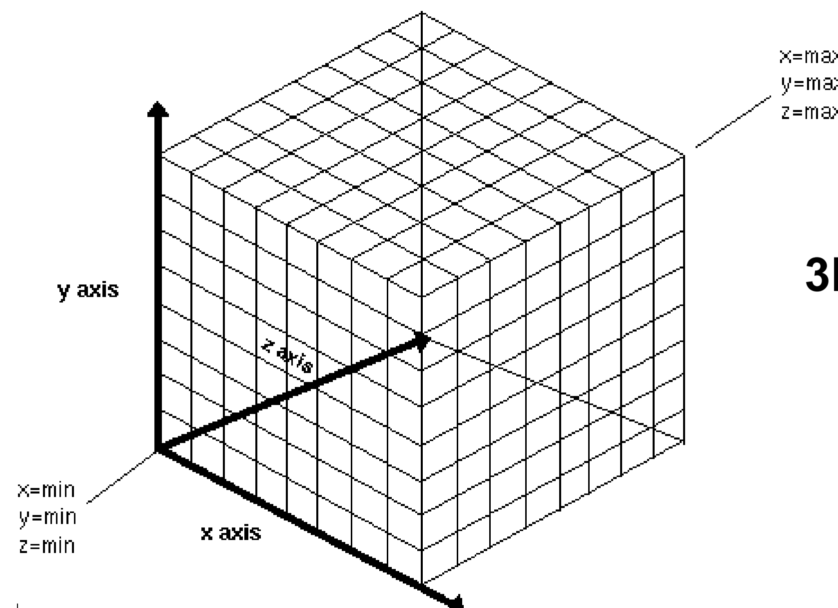
… code for geometry …

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





3D

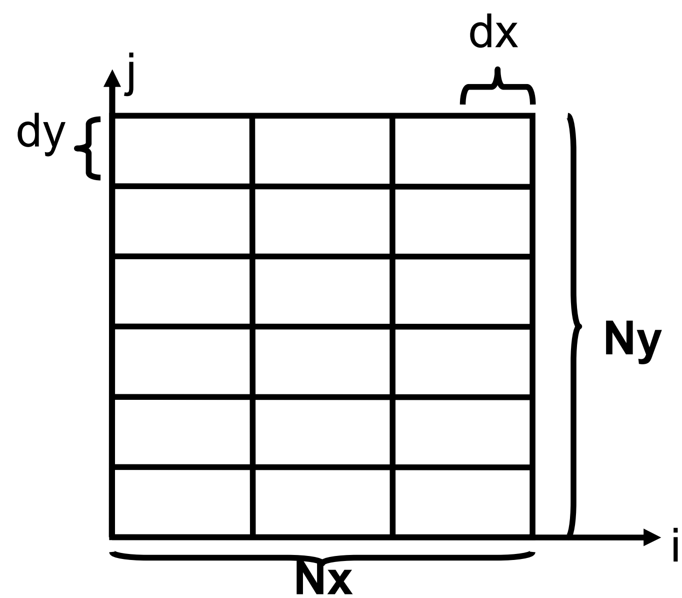
dx = dy = dz

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

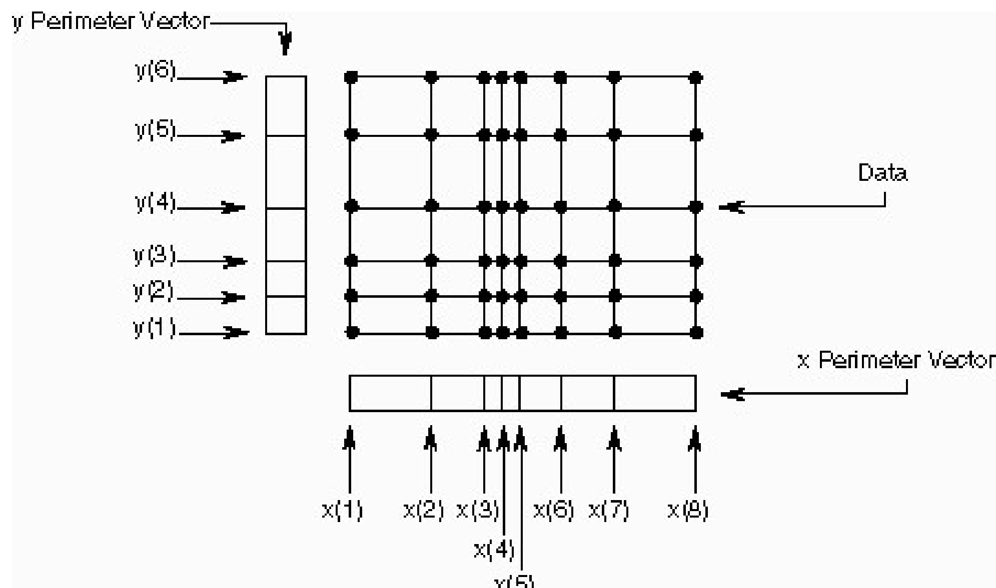
Data Structures

- 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

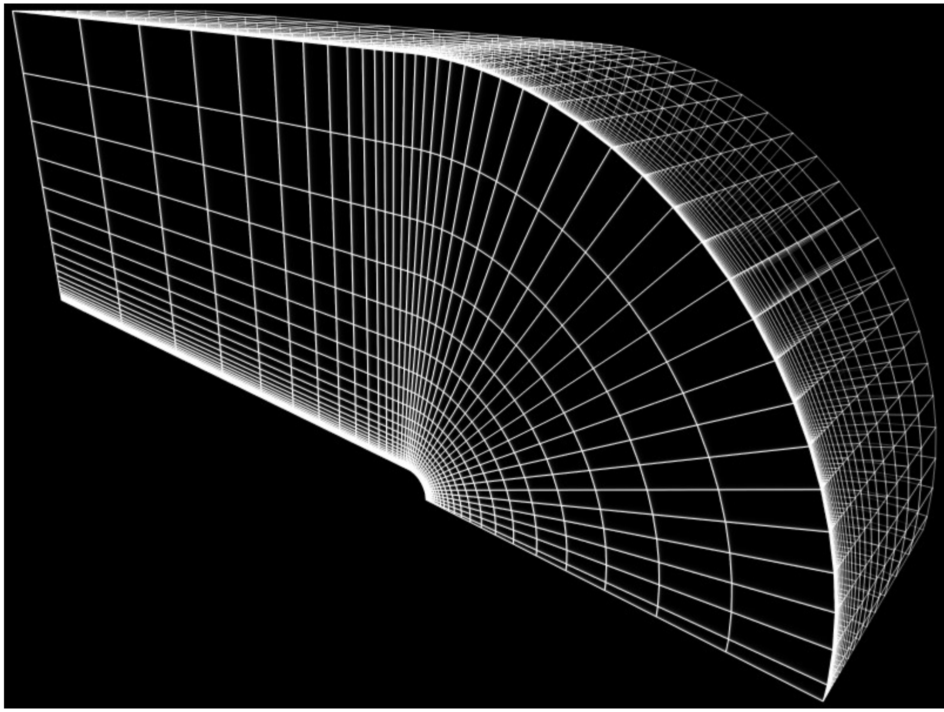
- Rectilinear grids
  - Topology is still regular but irregular spacing between grid points
    - Non-linear scaling of positions along either axis
    - Spacing, x_coord[L], y_coord[M], z_coord[N], must be stored explicitly
  - Topology is still implicit



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

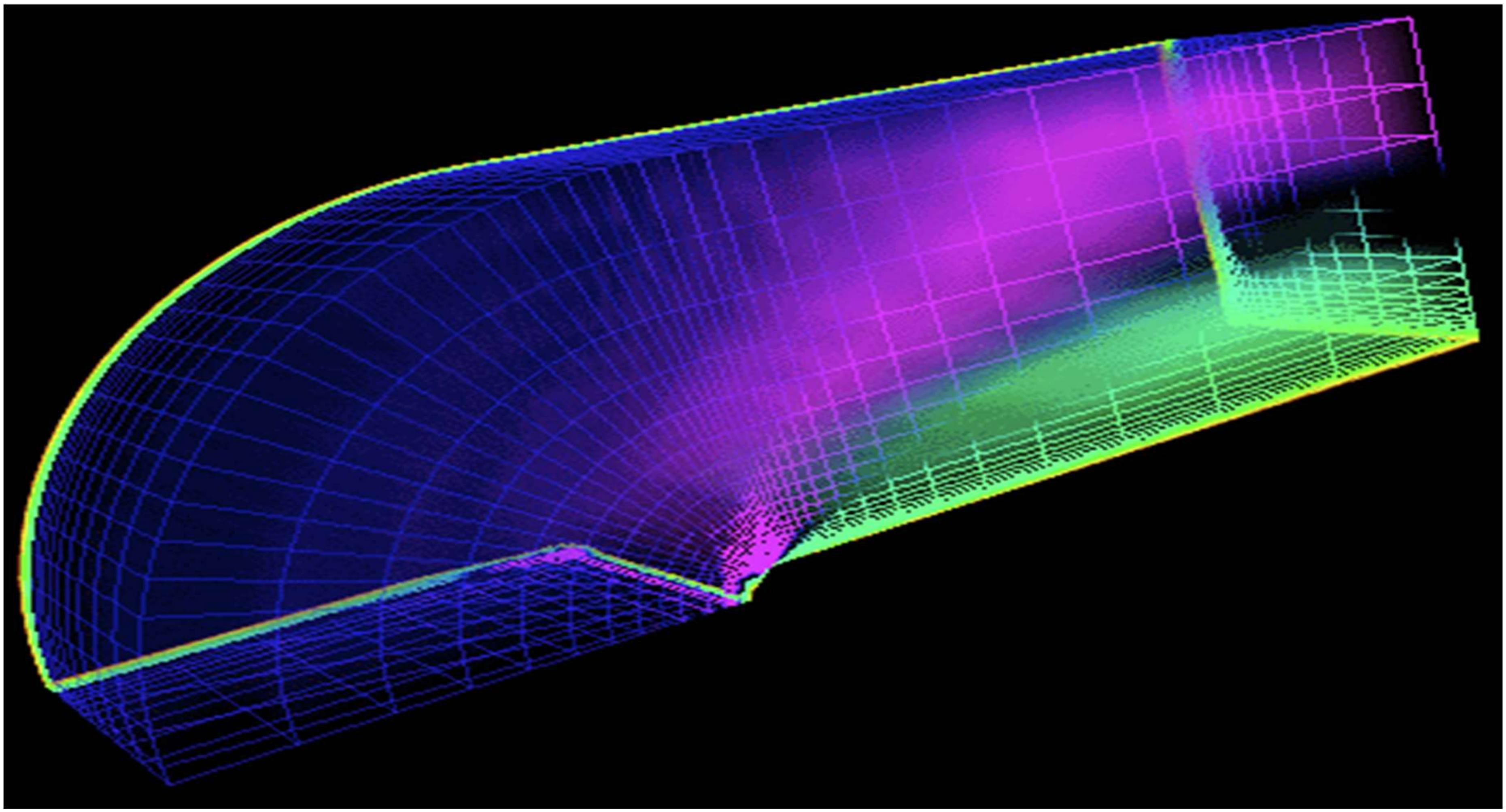
Data Structures

- 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

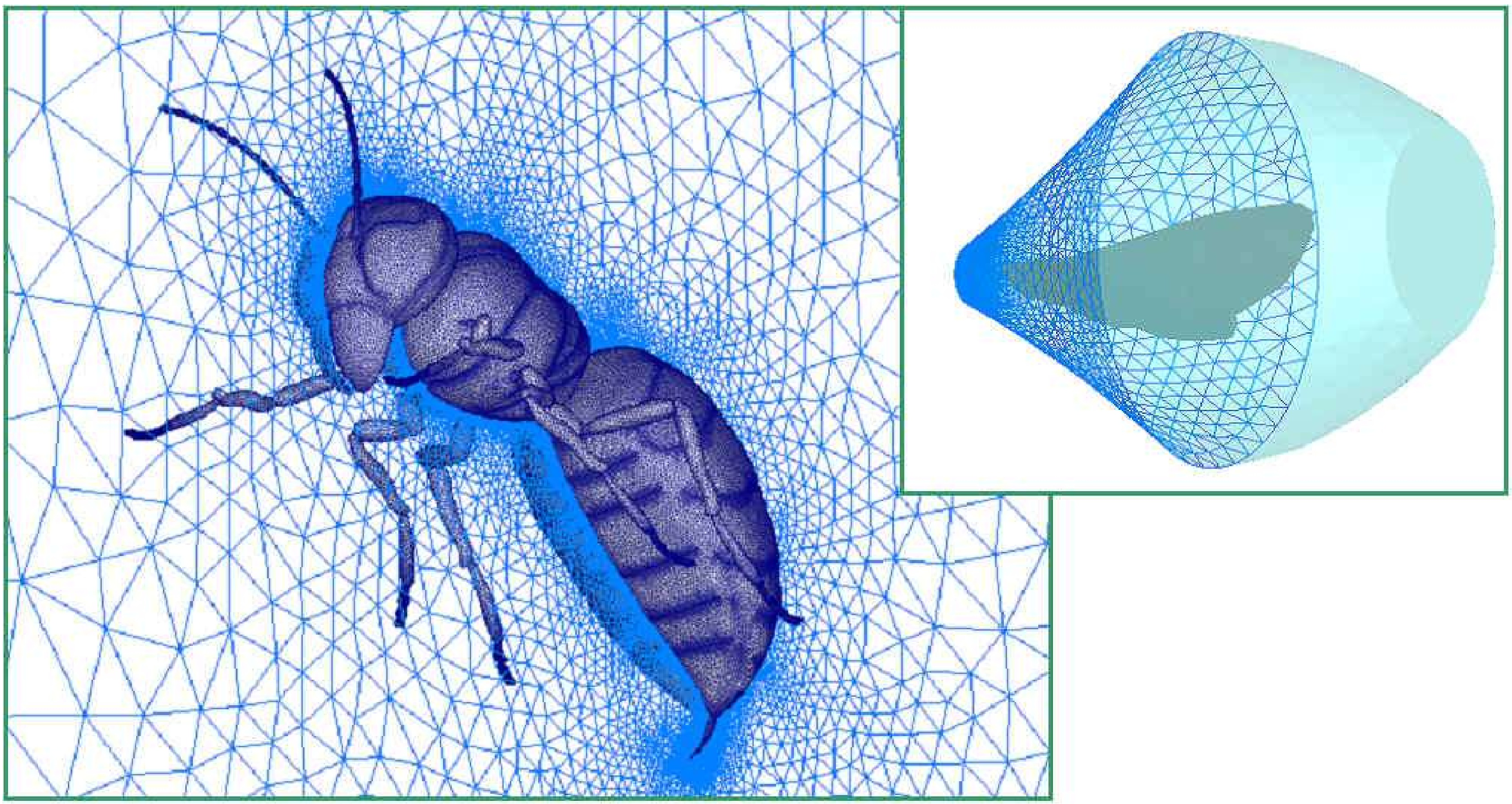
- Curvilinear grids



Unstructured Grids

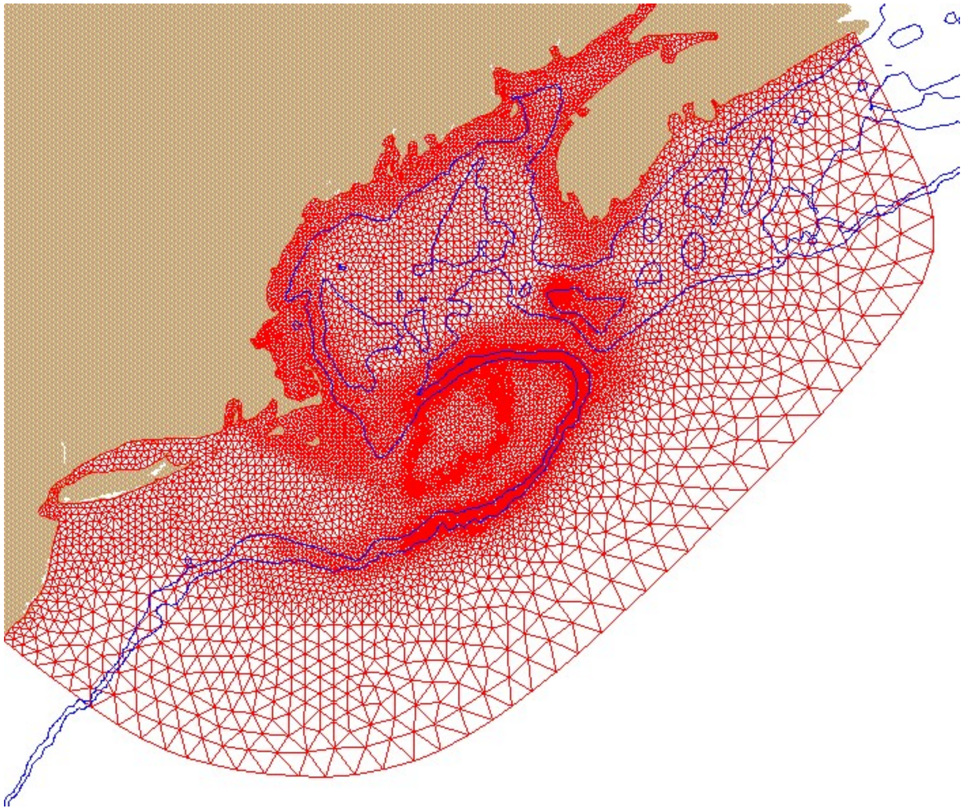
Data Structures

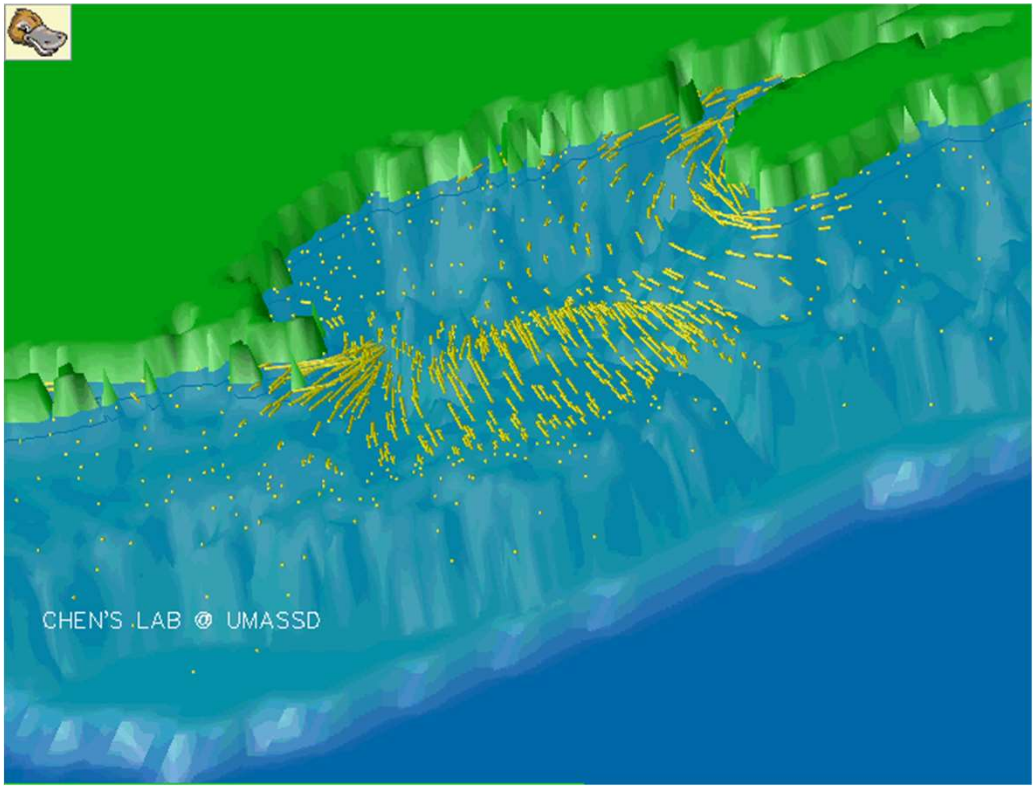
- Unstructured grids
  - Can be adapted to local features



Data Structures

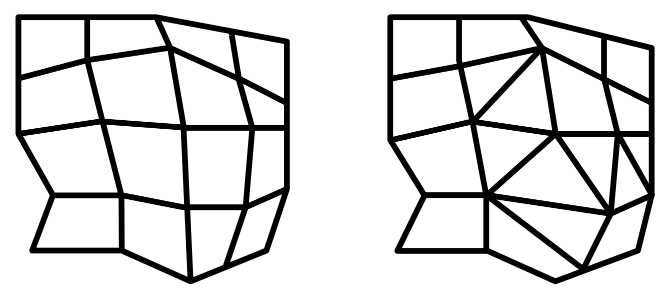
- Unstructured grids
  - Can be adapted to local features





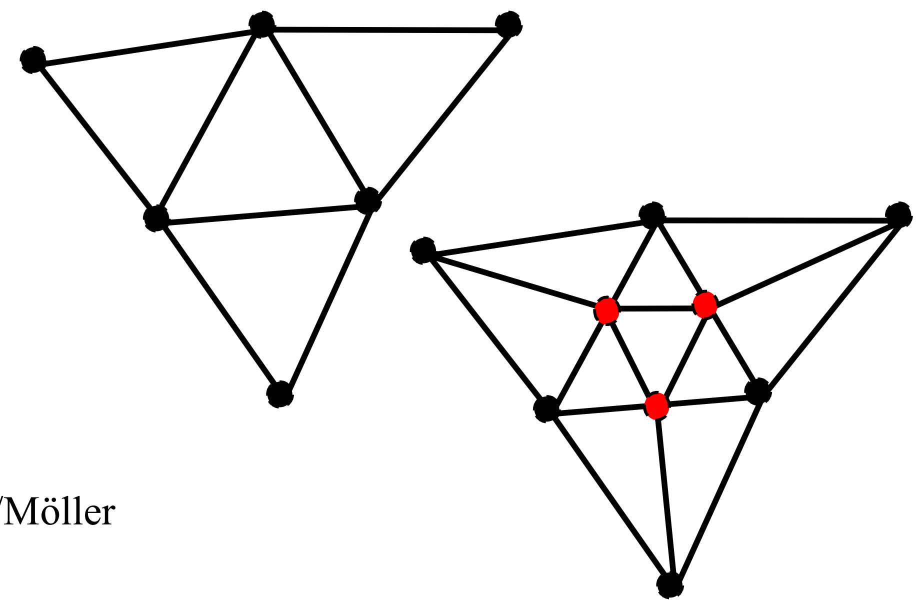
Data Structures

- If no implicit topological (connectivity) information is given, the grids are called unstructured grids
  - Unstructured grids are often computed using quadtrees (recursive domain partitioning for data clustering), or by triangulation of point sets
  - The task is often to create a grid from scattered points
- Characteristics of unstructured grids
  - Grid point geometry and connectivity must be stored
  - Dedicated data structures needed to allow for efficient traversal and thus data retrieval
  - Often composed of triangles or tetrahedra
  - Typically, fewer elements are needed to cover the domain



Data Structures

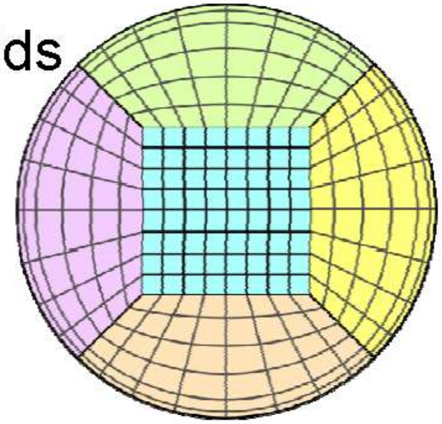
- Unstructured grids
  - Composed of arbitrarily positioned and connected elements
  - Can be composed of one unique element type or they can be hybrid (tetrahedra, hexas, prisms)
  - Triangle meshes in 2D and tetrahedral grids in 3D are most common
  - Can adapt to local features (small vs. large cells)
  - Can be refined adaptively
  - Simple linear interpolation in simplices



Data discretizations

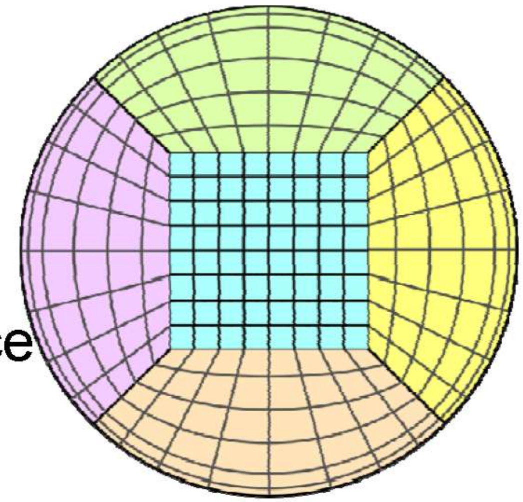
Types of data sources have typical types of discretizations:

- Measurement data:
  - typically scattered (no grid)
- Numerical simulation data:
  - structured, block-structured, unstructured grids
  - adaptively refined meshes
  - multi-zone grids with relative motion
  - etc.



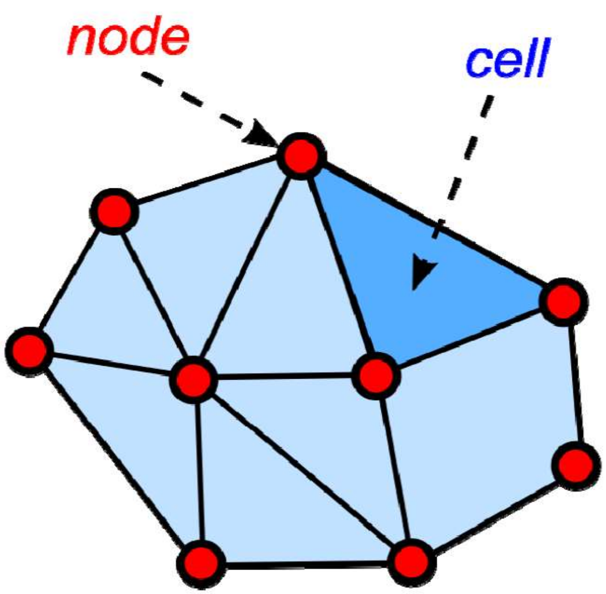
- Imaging methods:
  - uniform grids
- Mathematical functions:
  - uniform/adaptive sampling on demand

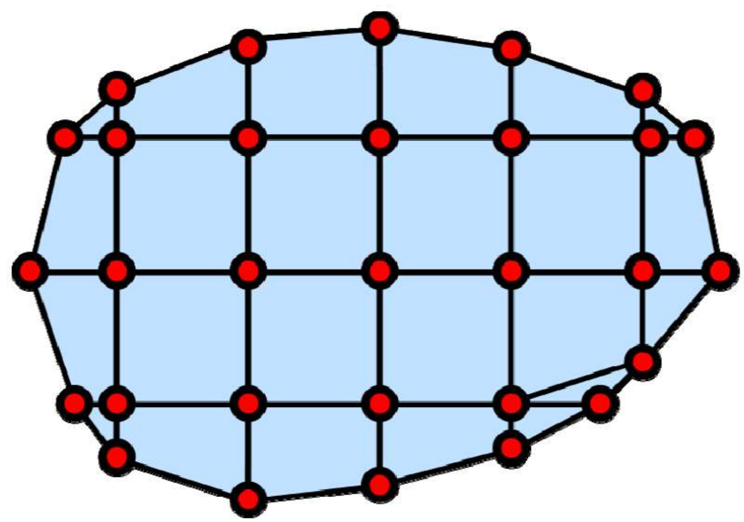
Unstructured grids



2D unstructured grids:

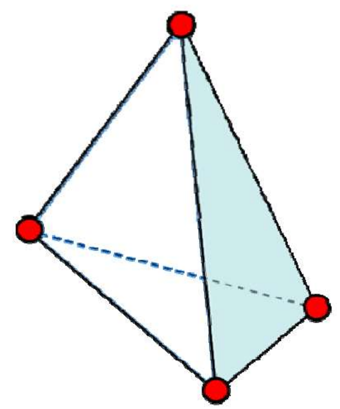
- cells are triangles and/or quadrangles
- domain can be a surface embedded in 3-space (distinguish n-dimensional from n-space)

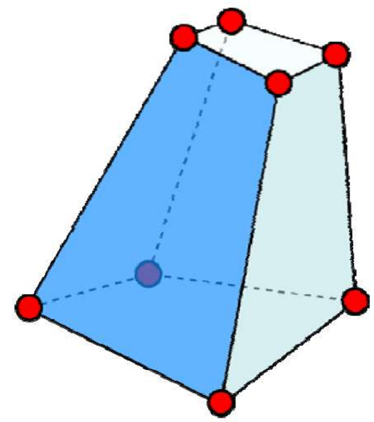




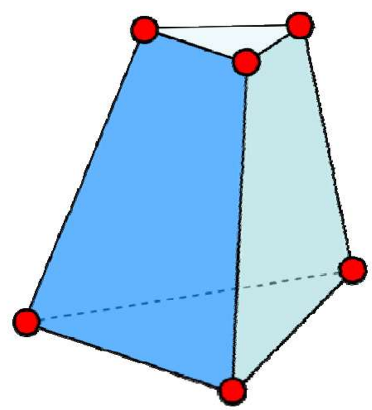
3D unstructured grids:

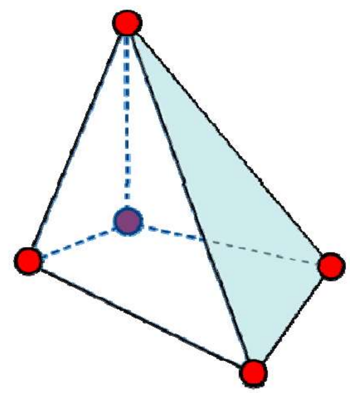
- cells are tetrahedra or hexahedra





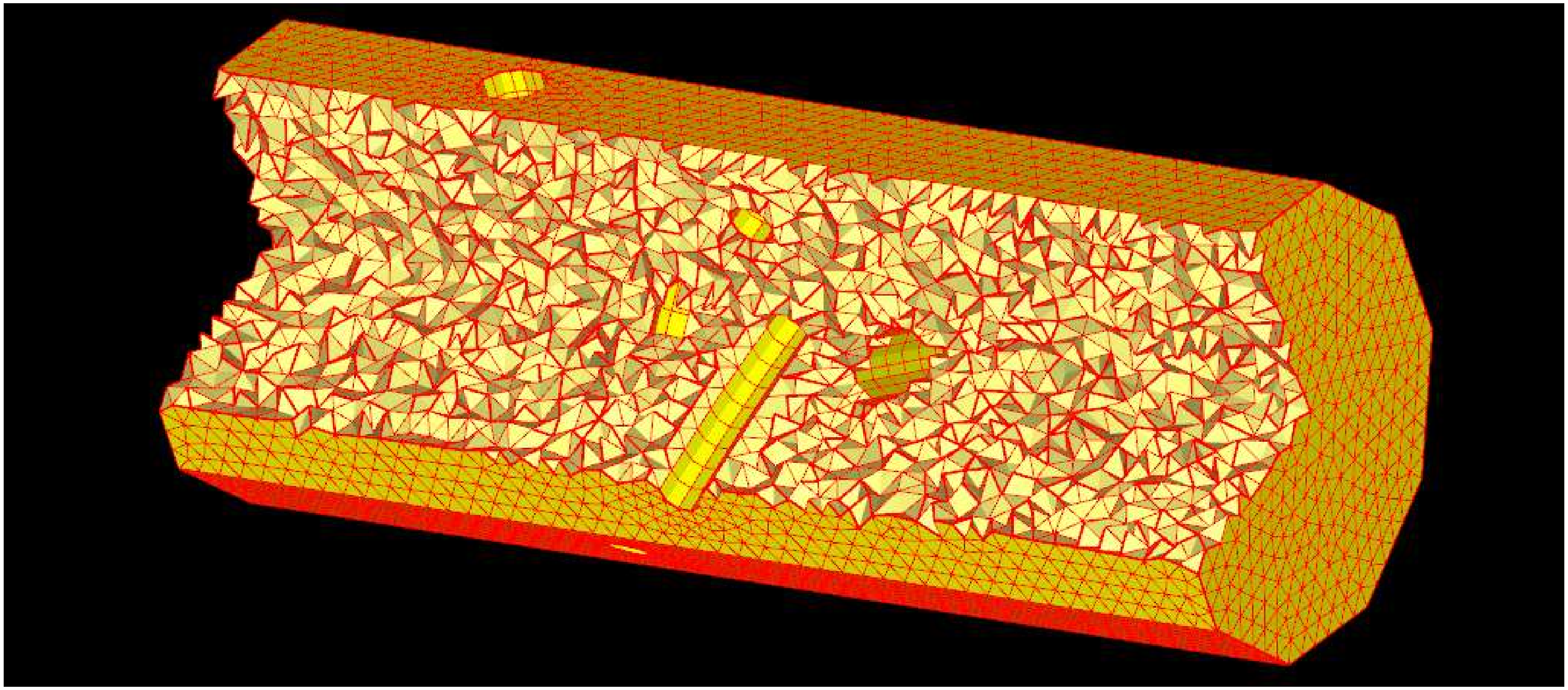
- mixed grids ("zoo meshes") require additional types: wedge (3-sided prism), and pyramid (4-sided)





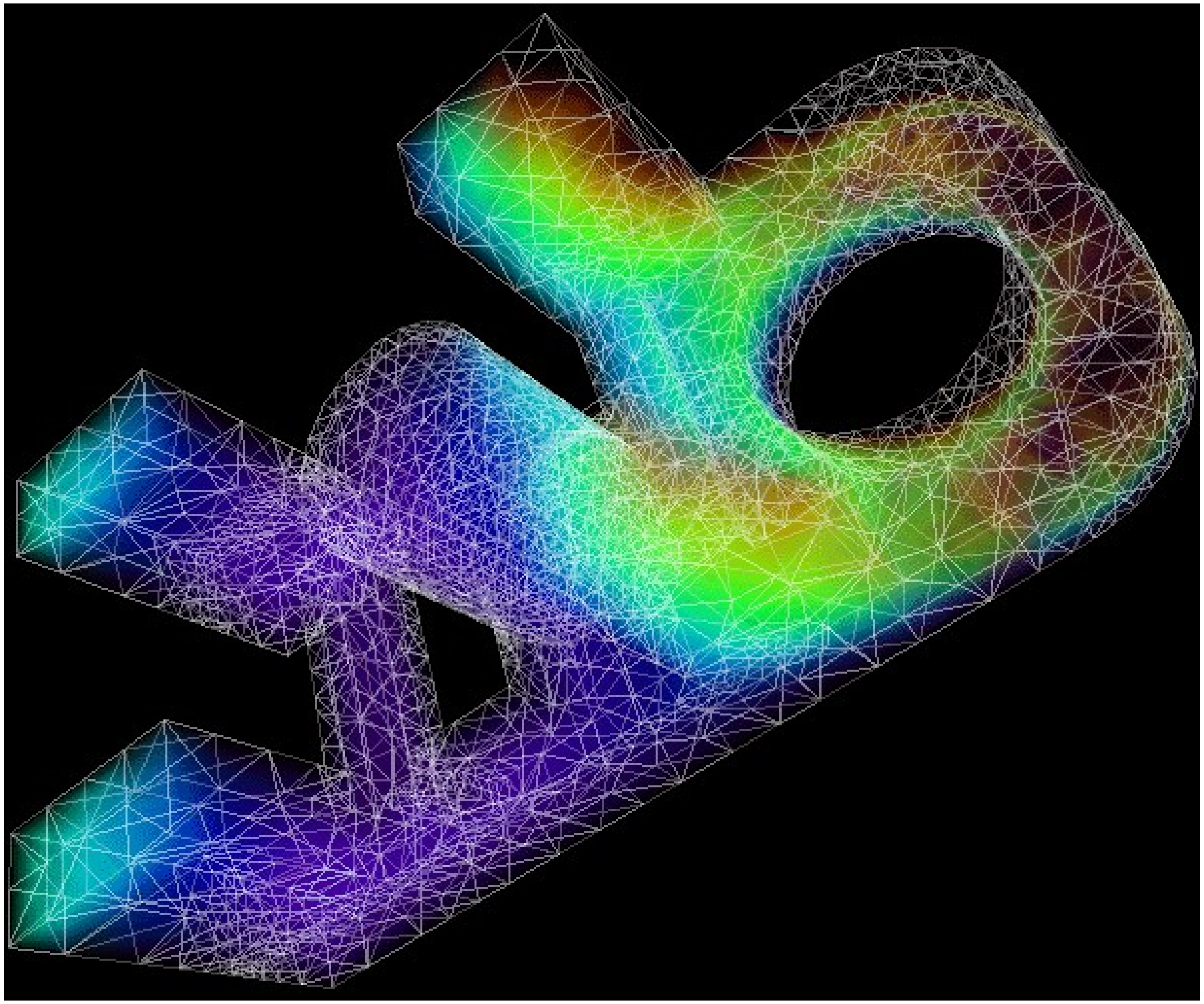
Common Unstructured Grid Types (1)

- Simplest: purely tetrahedral



Grid Structures

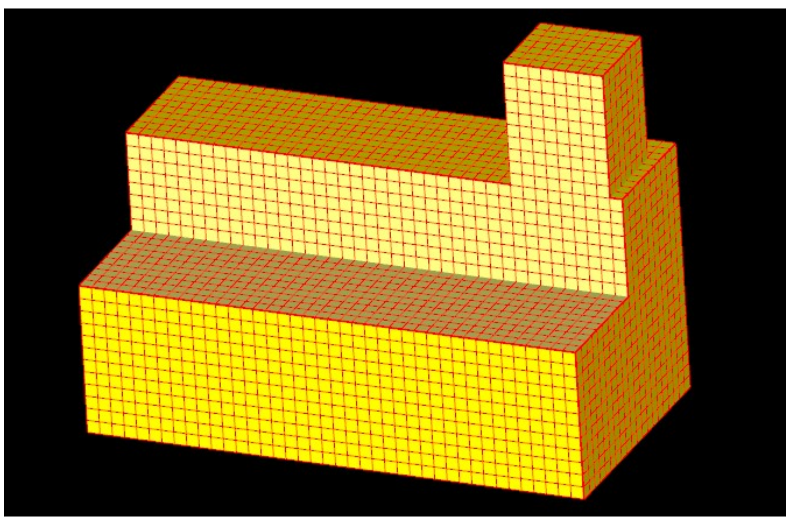
Tet grid example

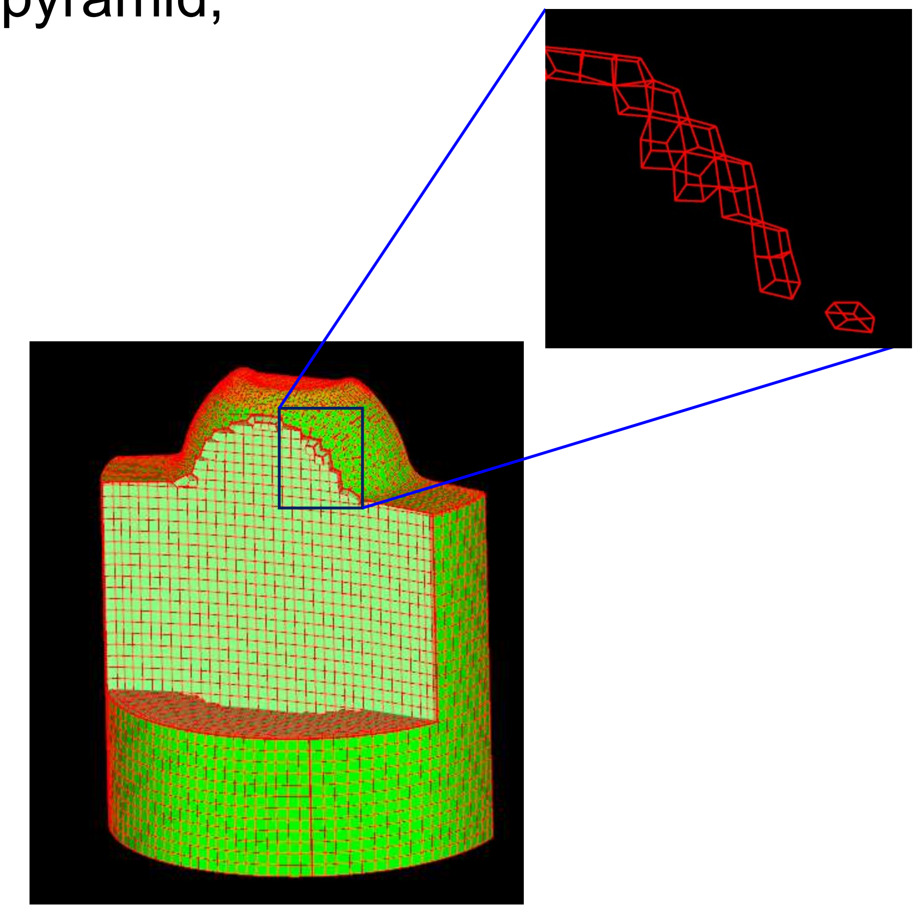


Common Unstructured Grid Types (2)

Pre-defined cell types
(tetrahedron, triangular prism, quad pyramid, hexahedron, octahedron)

- Only triangle / quad faces
- Planar / non-planar faces

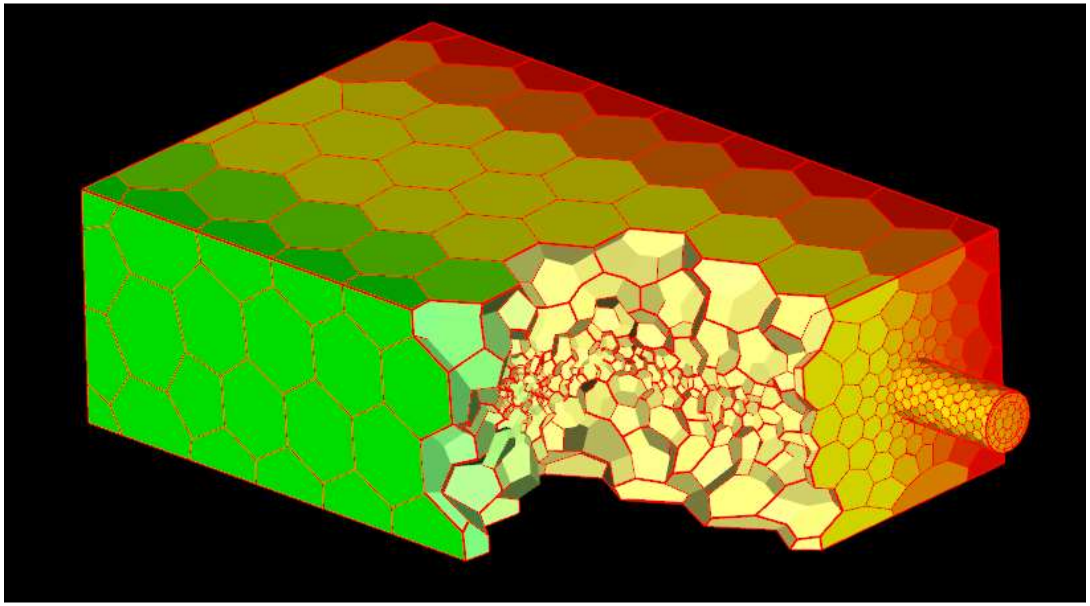


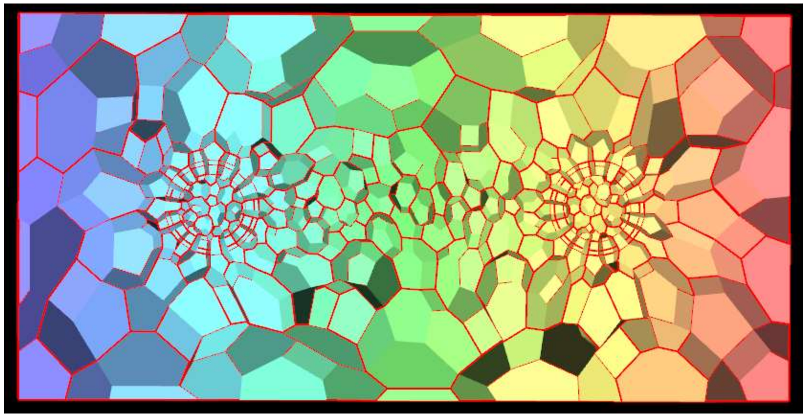


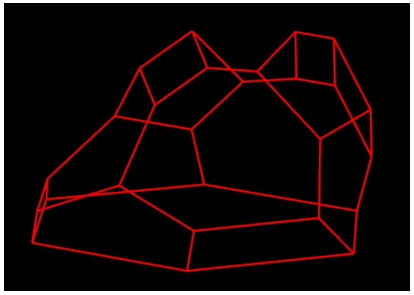
Common Unstructured Grid Types (3)

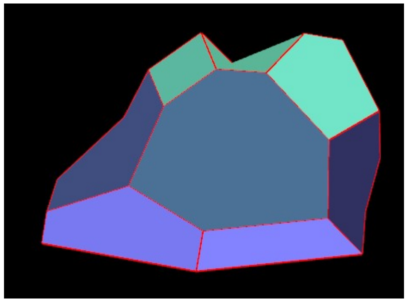
(Nearly) arbitrary polyhedra

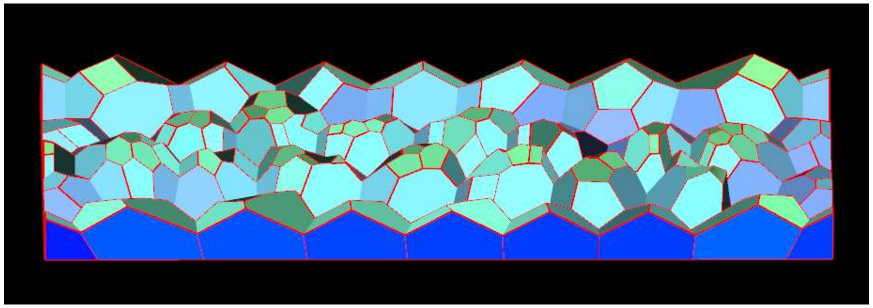
- Possibly non-planar faces







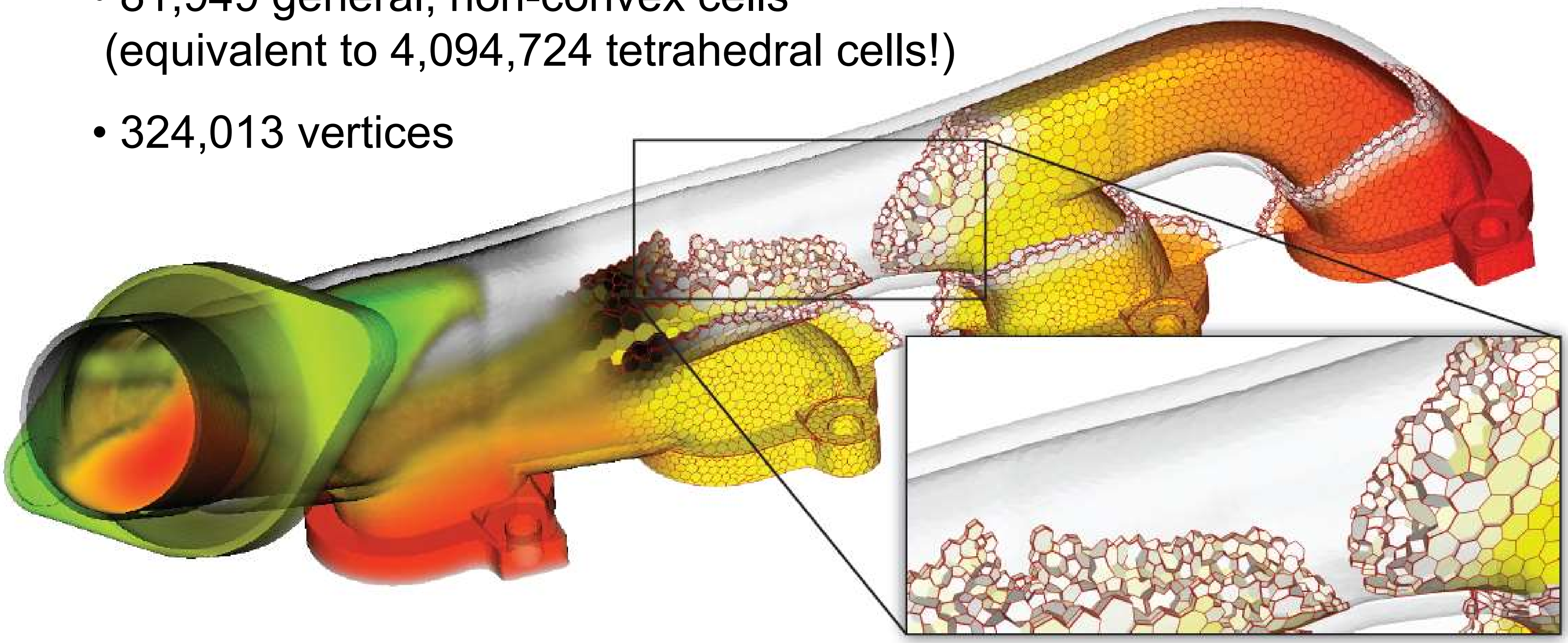




Example: General Polyhedral Cells

Exhaust manifold

- 81,949 general, non-convex cells
  (equivalent to 4,094,724 tetrahedral cells!)
- 324,013 vertices

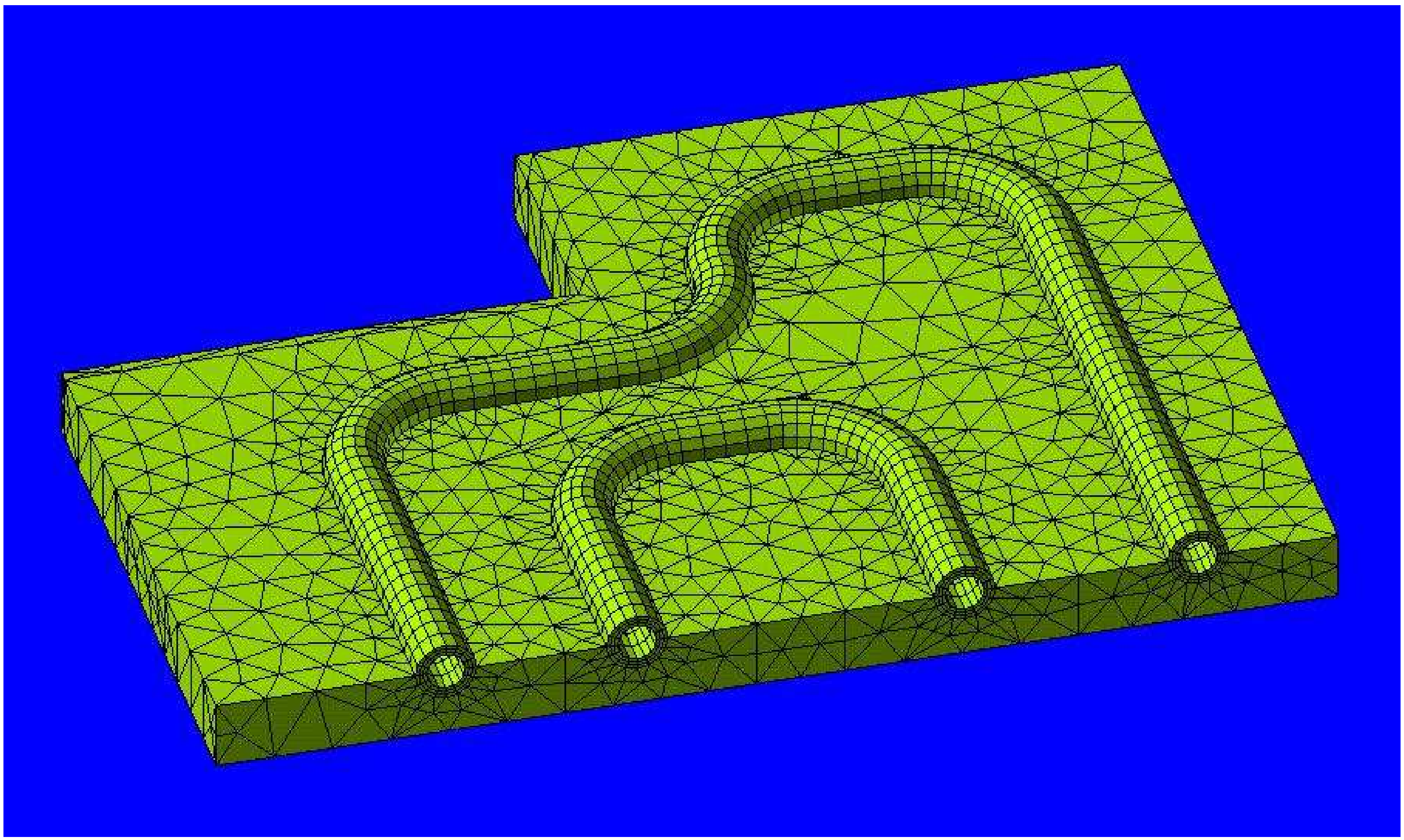


- Color coding: temperature distribution

Hybrid Grids

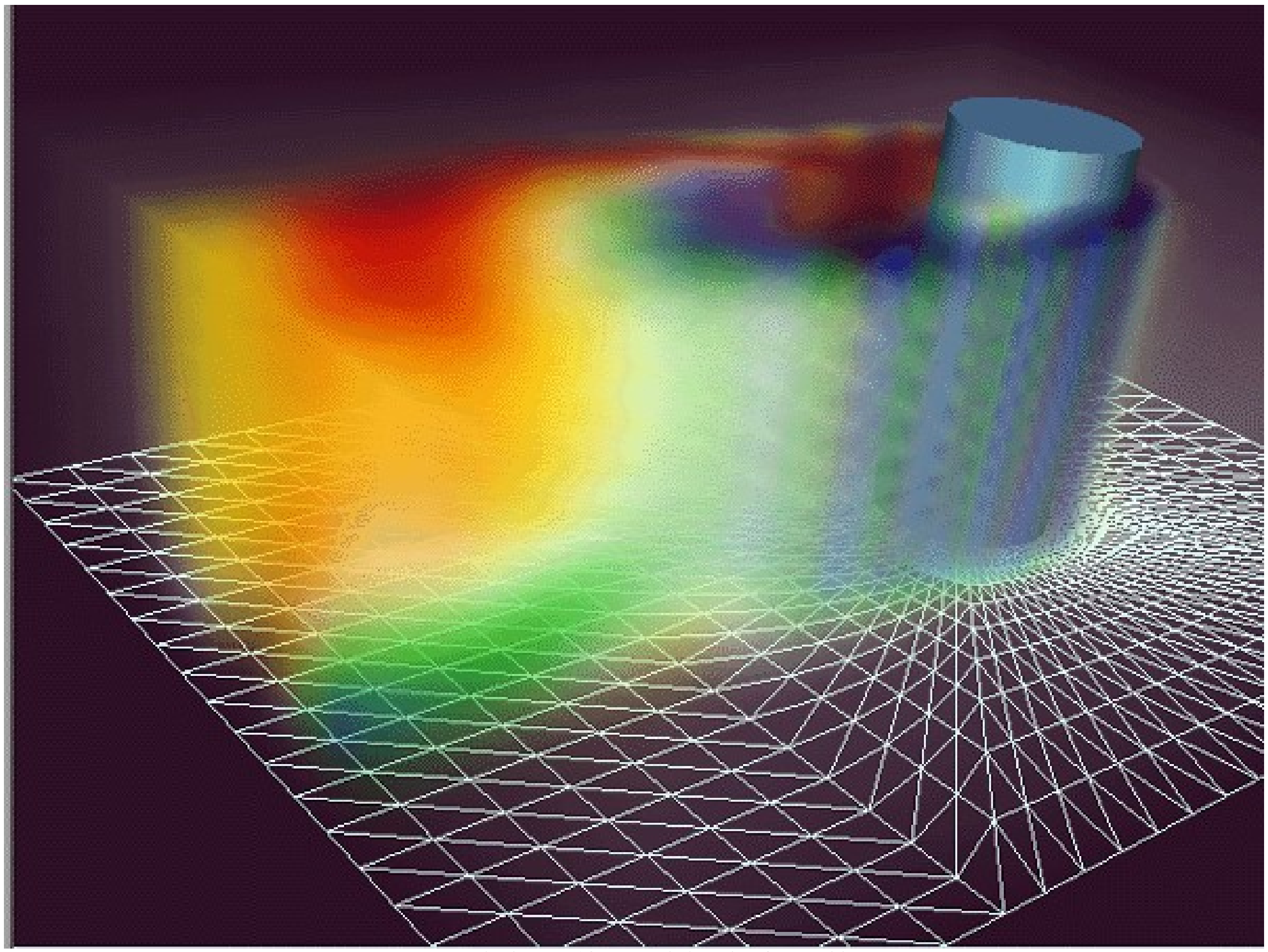
Data Structures

- Hybrid grids
  - Combination of different grid types

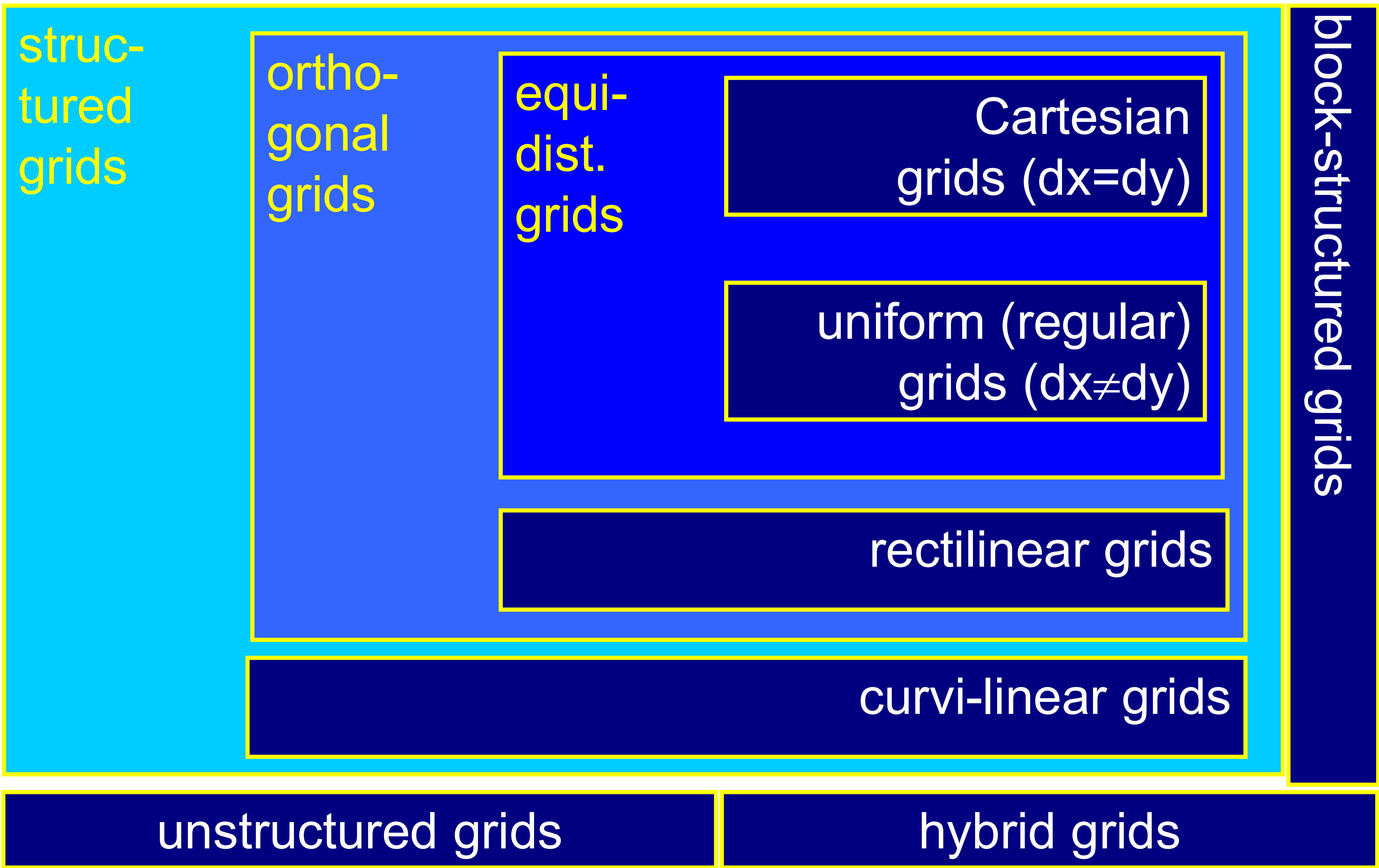


Data Structures

Hybrid grid example

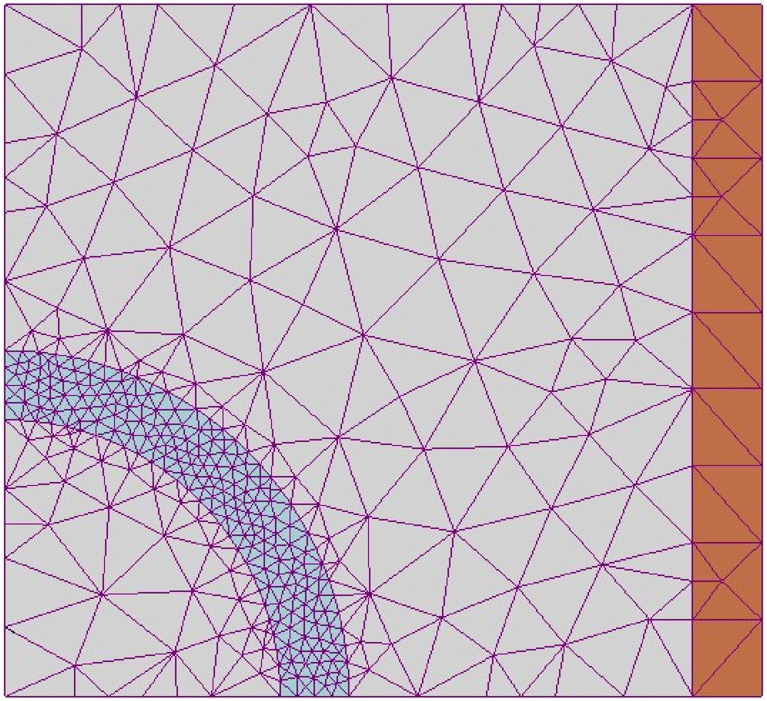


Grid Types - Overview

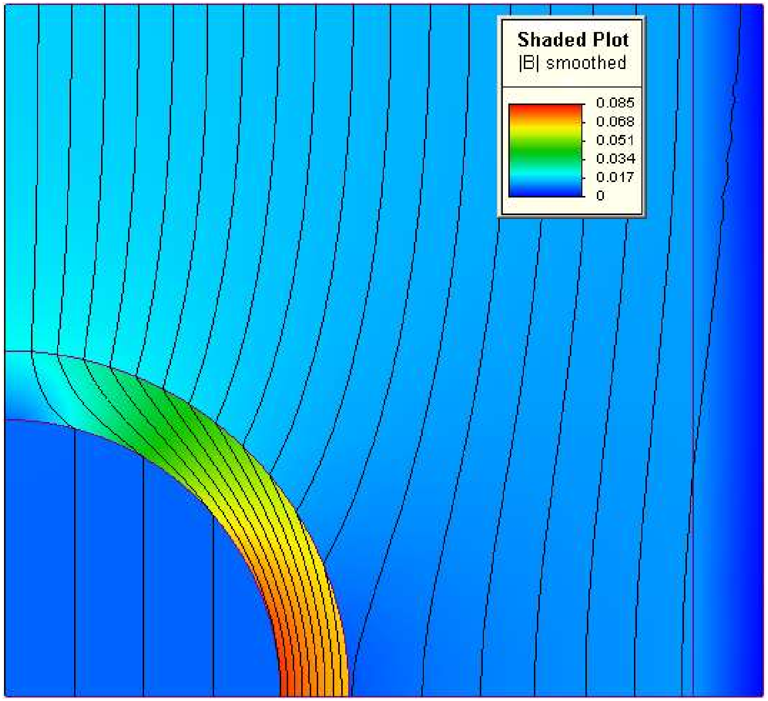


Grids vs. Data on Grids

grid



scalar field on grid



wikipedia

Unstructured Grid (Mesh) Data Structures

Unstructured 2D Grid: Direct Storage

Store list of vertices; vertices shared by triangles are replicated

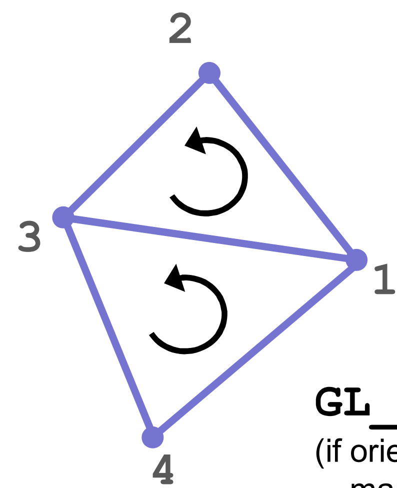
Render, e.g., with OpenGL immediate mode, …

coords for vertex 1 →

```
x1,y1,z1  ┐
x2,y2,z2  │ face 1
x3,y3,z3  ┘
x1,y1,z1  ┐
x3,y3,z3  │ face 2
x4,y4,z4  ┘

...
```

```
struct face
  float verts[3][3]
  DataType val;
```



GL_CCW
(if orientable manifold)

Redundant, large storage size, cannot modify shared vertices easily

Store data values per face, or separately

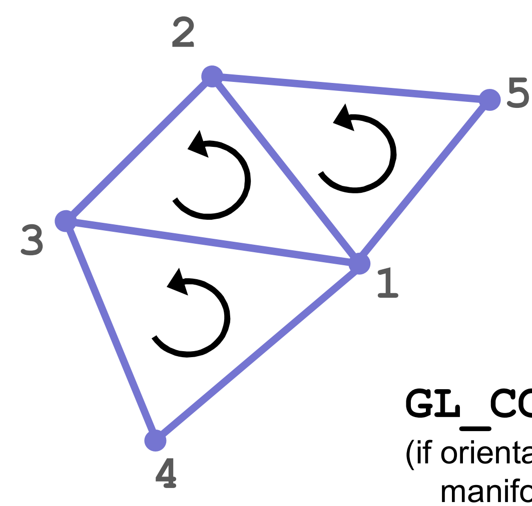
Unstructured 2D Grid: Indirect Storage

Indexed face set: store list of vertices; store triangles as indexes

Render using separate vertex and index arrays / buffers

coords for vertex 1 →

vertex list	face list
`x1,y1,(z1)`	`1,2,3`
`x2,y2,(z2)`	`1,3,4`
`x3,y3,(z3)`	`2,1,5`
`x4,y4,(z4)`	`...`
`...`	



`GL_CCW`
(if orientable manifold)

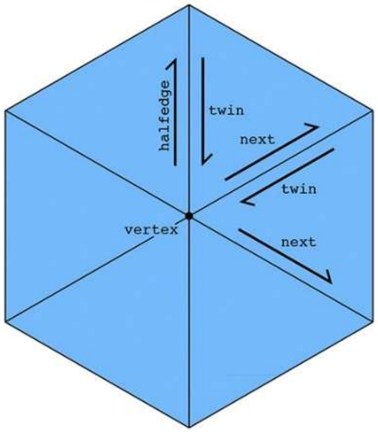
Less redundancy, more efficient in terms of memory

Easy to change vertex positions; still have to do (global) search for shared edges (local information)

Unstructured 2D Grids: Connectivity/Incidence

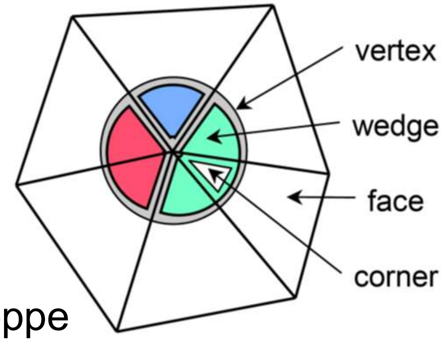
Half-edge (doubly-connected edge list) data structure

- Pointer to half-edge (twin) in neighboring face
  (mesh needs to be orientable 2-manifold)
- Pointer to next half-edge in same face
- Half-edge associated with one vertex, edge, face



Modifications: attributes, mesh simplification, …

- Vertices, corners, wedges, faces
- Express attribute continuity vs. discontinuity



Hugues Hoppe

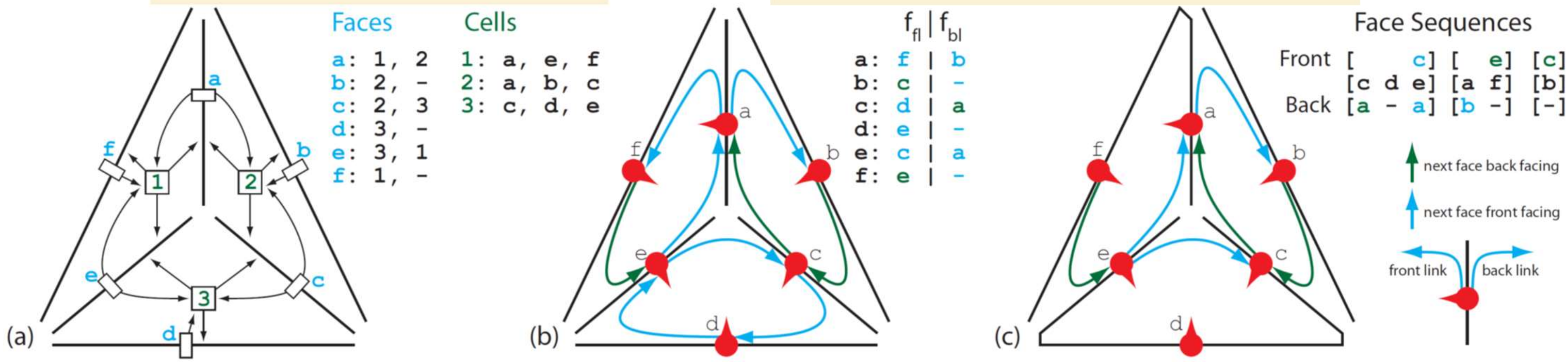
Visualization often needs volumetric version of these ideas
(tet meshes, polyhedral meshes, …)

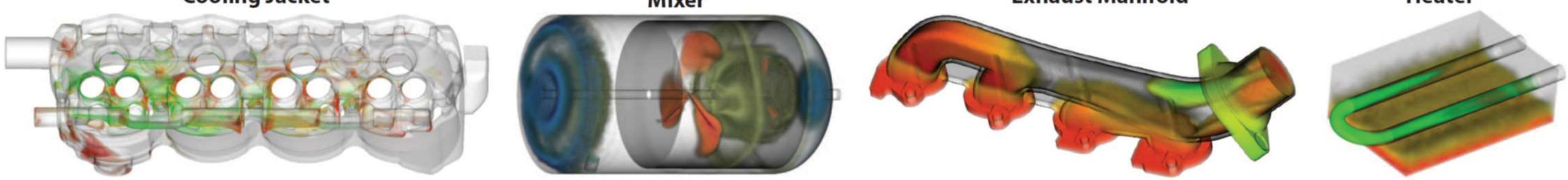
3D Grids: Two-Sided Face Sequence Lists

General polyhedral grids (arbitrary polyhedral cells); example: TSFSL (Muigg et al., 2011)

standard face/cell incidence

two-sided face (sequence) lists





	Cooling Jacket	Mixer	Exhaust Manifold	Heater
Cells/Vertices/Faces:	1,538K / 1,631K / 4,707K	1,362K / 7,432K / 8,869K	82K / 324K / 441K	17K / 68K / 91K
Tetrahedra:	17,044K (~8.5 byte/tet)	89,417K (~7.5 byte/tet)	4,095K (~7.0 byte/tet)	851K (~7.0 byte/tet)
Celltypes:	tets/pyramids/wedges/hexas	general (non-convex) polyhedra	general (non-convex) polyhedra	general (non-conv.) polyh.
Bricks/Cell Overhead:	4/1.7%	10/8.6%	1/0%	1/0%
TSFSL Creation Time:	4.0s	9.0s	1.7s	1.0s

Thank you.

Thanks for material

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