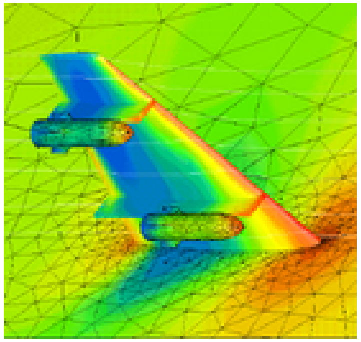
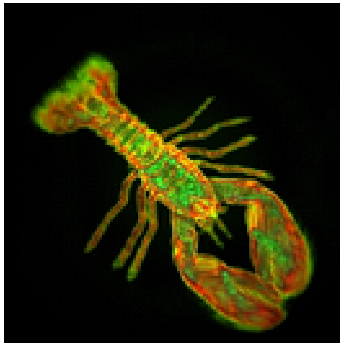
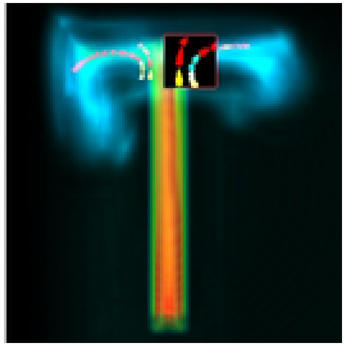
Scientific visualization concepts

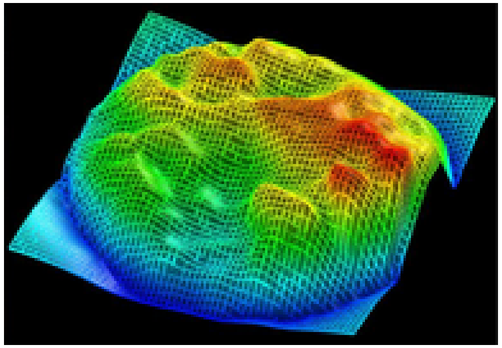
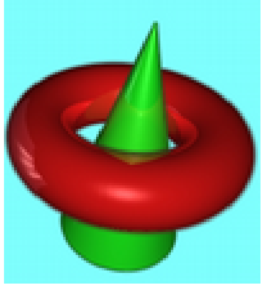
Luigi Calori

Slides material from:

Alex Telea, Groningen University: www.cs.rug.nl/svcg
Kitware: *www.**kitware**.com*
Sandia National Laboratories
Argonne National Laboratory
Julich supercomputing center
Tuomas Hätinen

1. Introduction to Data Visualization

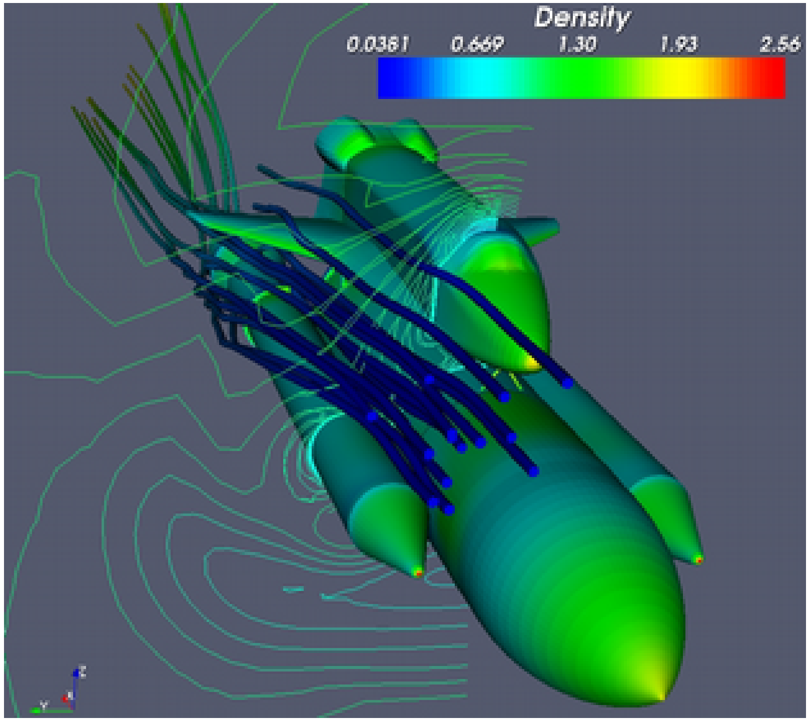
What is Data Visualization (for)?

Scientific **Visualization**: "The use of computers or techniques for comprehending data or to extract knowledge from the results of simulations, computations, or measurements"
[McCormick *et al.*, 1987]

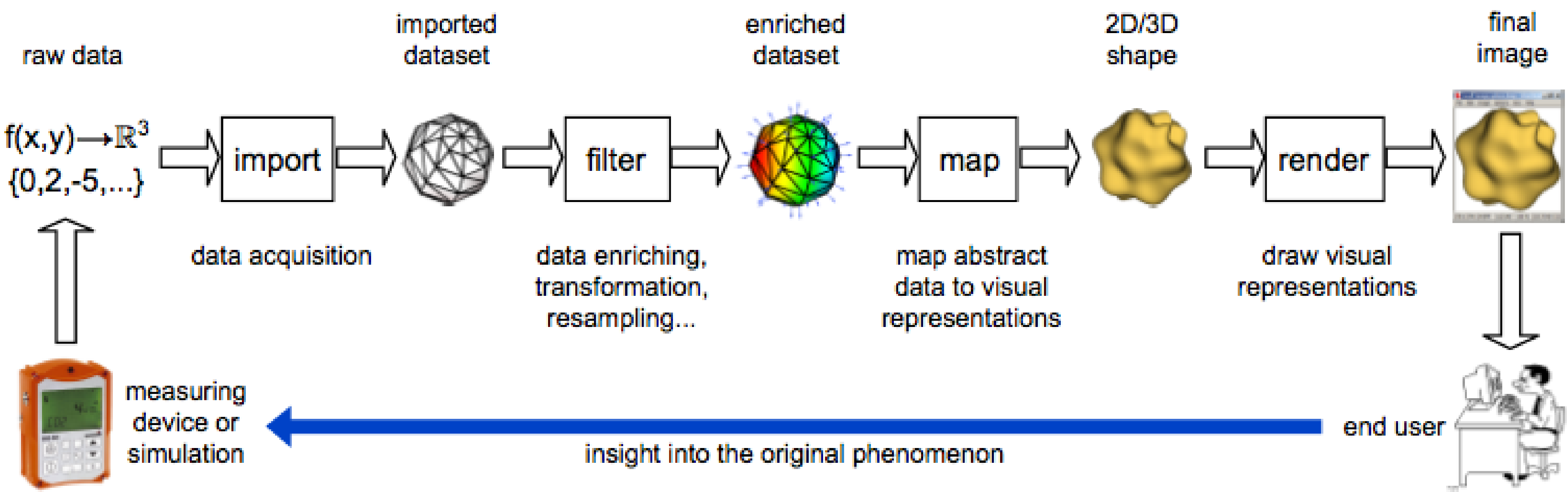
Information Visualization: "Visualization applied to abstract quantities and relations in order to get insight in the data"
[Chi, 2000]

Basics of Visualization


```
0265640 132304 133732 032051 037334 024721 015013 052226 001662
0265660 025537 064663 054606 043244 074076 124153 135216 126614
0265700 144210 056426 044700 042650 165230 137037 003655 006254
0265720 134453 124327 176005 027034 107614 170774 073702 067274
0265740 072451 007735 147620 061064 157435 113057 155356 114603
0265760 107204 102316 171451 046040 120223 001774 030477 046673
0266000 171317 116055 155117 134444 167210 041405 147127 050505
0266020 004137 046472 124015 134360 173550 053517 044635 021135
0266040 070176 047705 113754 175477 105532 076515 177366 056333
0266060 041023 074017 127113 003214 037026 037640 066171 123424
0266100 067701 037406 140000 165341 072410 100032 125455 056646
0266120 006716 071402 055672 132571 105645 170073 050376 072117
0266140 024451 007424 114200 077733 024434 012546 172404 102345
0266160 040223 050170 055164 164634 047154 126525 112514 032315
0266200 016041 176055 042766 025015 176314 017234 110060 014515
0266220 117156 030746 154234 125001 151144 163706 136237 164376
0266240 137055 062276 161755 115466 005322 132567 073216 002655
0266260 171466 126161 117155 065763 016177 014460 112765 055527
0266300 003767 175367 104754 036436 172172 150750 043643 145410
0266320 072074 000007 040627 070652 173011 002151 125132 140214
0266340 060115 014356 015164 067027 120206 070242 033065 131334
0266360 170601 170106 040437 127277 124446 136631 041462 116321
0266400 020243 005602 004146 121574 124651 006634 071331 102070
0266420 157504 160307 166330 074251 024520 114433 167273 030635
0266440 133614 106171 144160 010652 007365 026416 160716 100413
0266460 026630 007210 000630 121224 076033 140764 000737 003276
0266500 114060 042647 104475 110537 066716 104754 075447 112254
0266520 030374 144251 077734 015157 002513 173526 035531 150003
0266540 146207 015135 024446 130101 072457 040764 165513 156412
0266560 166410 067251 156160 106406 136770 030516 064740 022032
0266600 142166 123707 175121 071170 076357 037233 031136 015232
0266620 075074 016744 044055 102230 110063 033350 052765 172463
```

The Visualization Pipeline



- transform raw data into insightful answers
- sequence of **steps**
  - data acquisition (conversion, formatting, cleaning)
  - data enrichment (transformation, resampling, filtering)
  - data mapping (produce visible shapes from data)
  - rendering (draw and interact with the shapes)

When is visualization useful?

1. Too much data:
   - do not have time to analyze it all (or read the analysis results)
   - show an overview, discover which questions are relevant
   - refine search either visually or analytically

2. Qualitative / complex questions:
   - cannot capture question compactly/exactly in a query
   - question/goal is inherently qualitative: understand what is going on
   - show an overview, answer the question by seeing relevant patterns

3. Communication / presentation / decision making:
   - transfer results to different (non technical) stakeholders
   - learn about a new domain or problem
   - Teach / train people who do not already have deep understanding

SCAI
SuperComputing Applications and Innovation

When is visualization NOT useful?

1. Queries:

- if a question can be answered by a compact, precise query, why visualize?
- “what is the largest value of a set”
- When human perceptual system is not effective
- When there are cheaper substitutes for human perceptual system (google car)

2. Automatic decision-making:

- if a decision can be automated, why use a human in the loop?
- “how to optimize a numerical simulation”

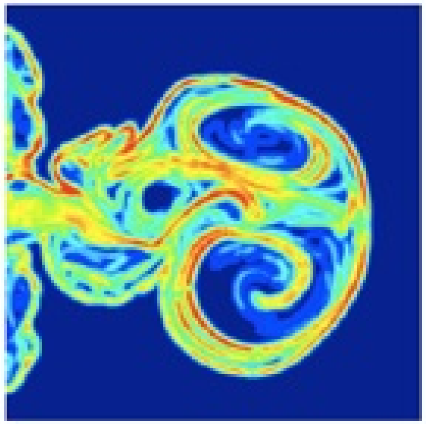
Key thing to remember:

- visualization is *mainly* a cost vs benefits (or value vs waste) proposal
    - cost: effort to create and interpret the images
    - benefits: problem solved by interpreting the images

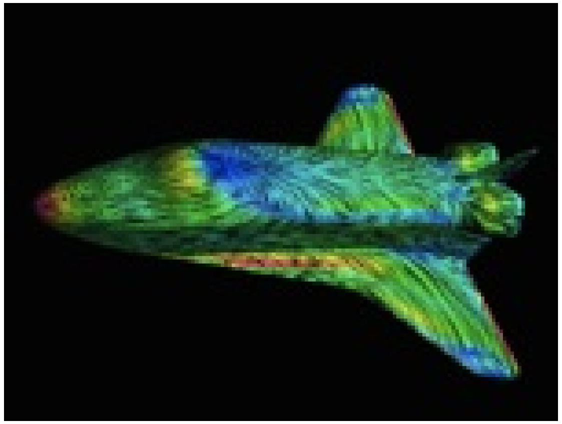
✗ B. Lorensen, On the Death of Visualization, Proc. NIH/NSF Fall Workshop on Visualization Research Challenges, 2004
✗ S. Charters, N. Thomas, M. Munro, The end of the line for Software Visualisation? Proc. IEEE VISSOFT, 2003
✗ S. Reiss, The paradox of software visualization, Proc. IEEE VISSOFT, 2005
✓ J. J. van Wijk, The Value of Visualization, Proc. IEEE Visualization, 2005

PRACE CINECA

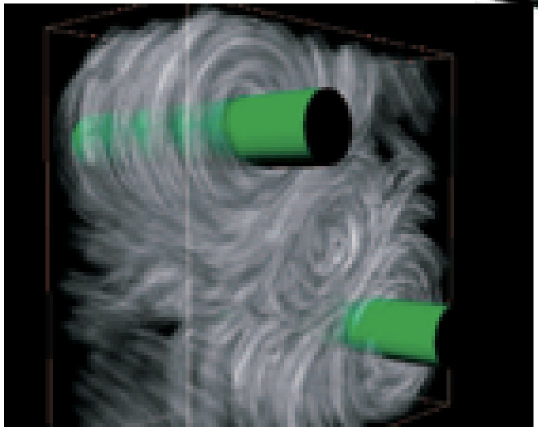
Visualization examples: Fluid flow



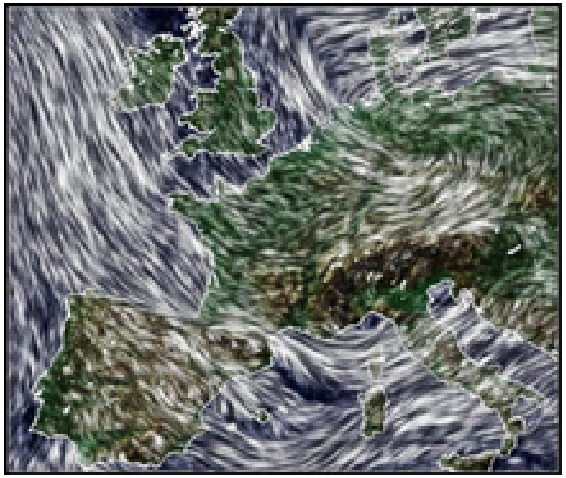
mixing of substances
(macro chemistry)



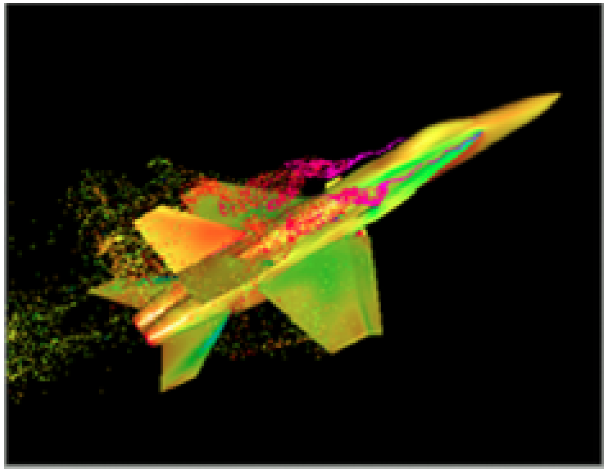
flow on surface
(aircraft design)



flow in volume
(engine design)

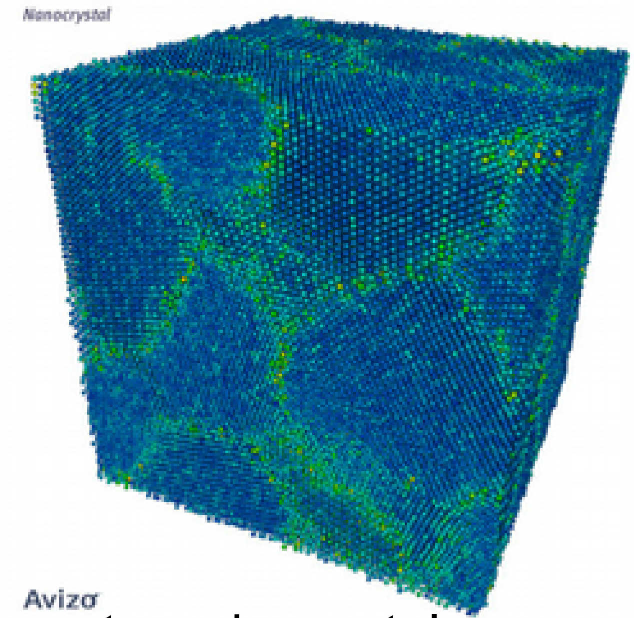


wind flow atop geo map
(weather forecast)

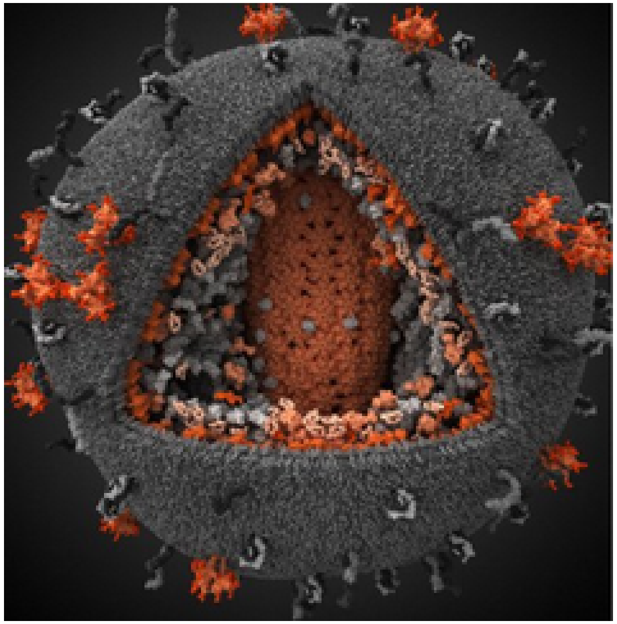


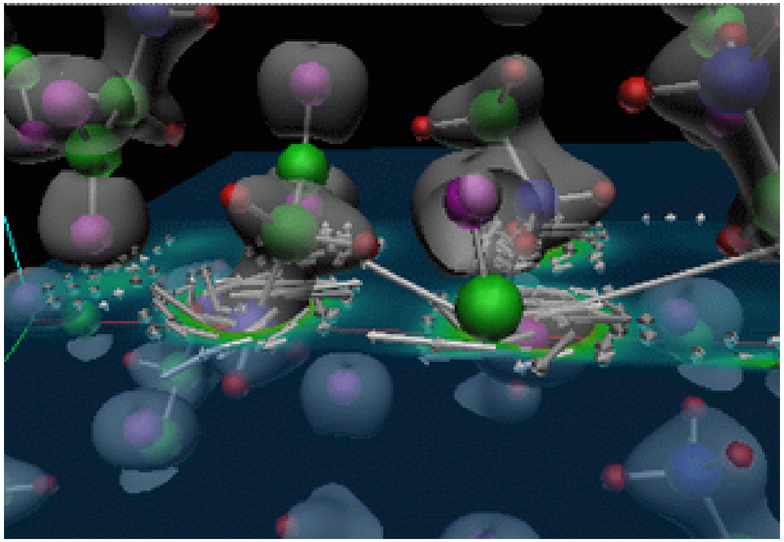
particle flow close to surface
(aircraft design 2)

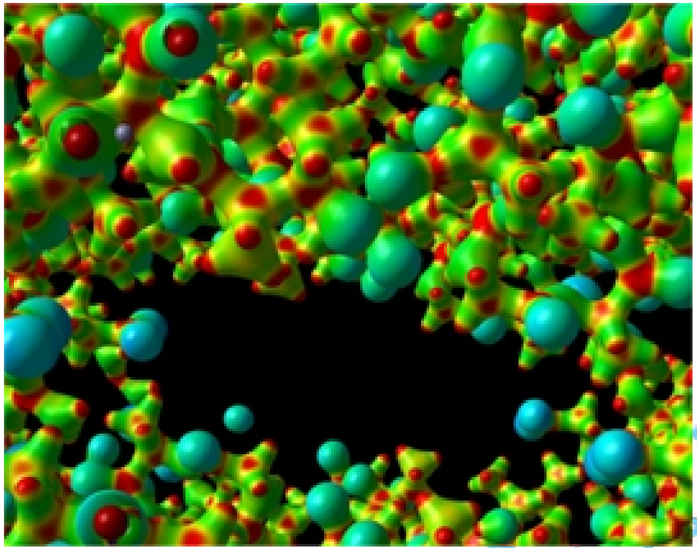
Viz examples: Material/biosciences



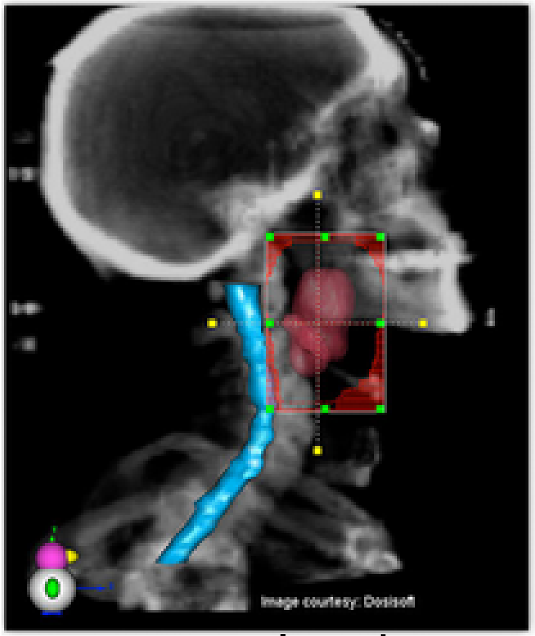
atoms in crystal
(crystallography)


3D HIV model

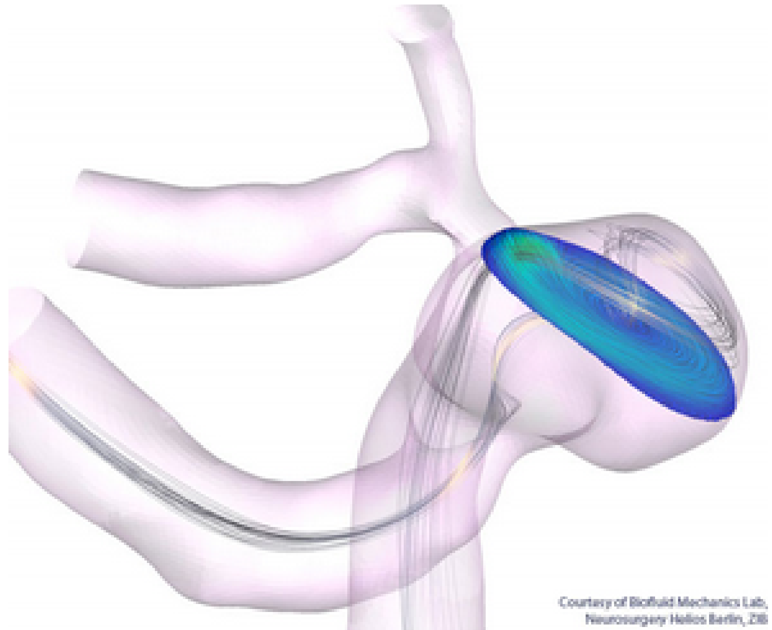

glycine crystal simulation


potential field in crystal structure

Viz examples: Medical sciences

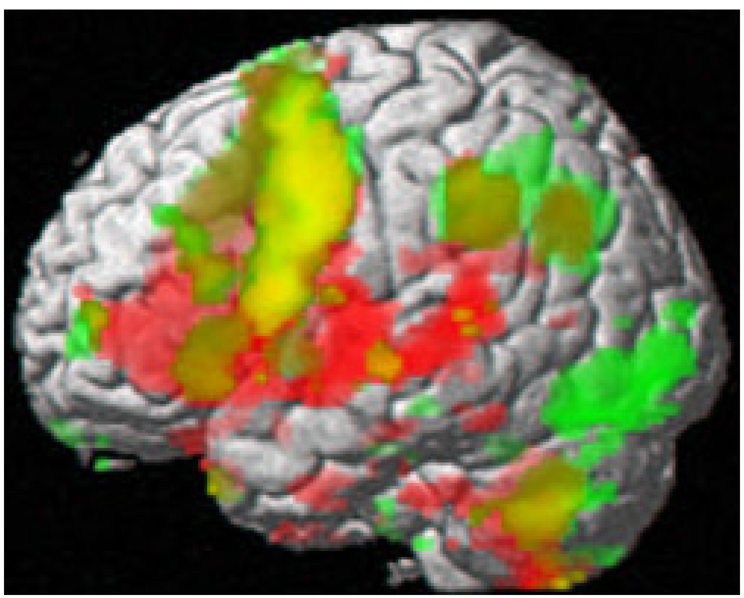


surgery planning



blood flow in aneurysm

bone tissue density

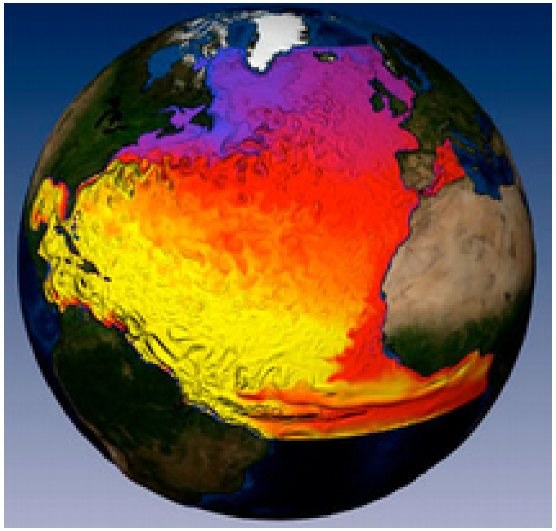


brain activity (fMRI)

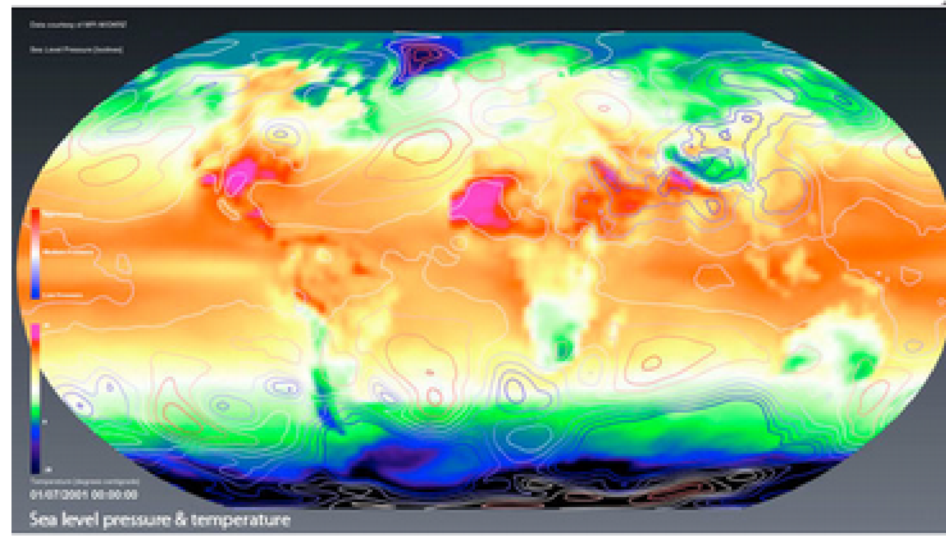
MRI scan - tissues

bone + skin surface

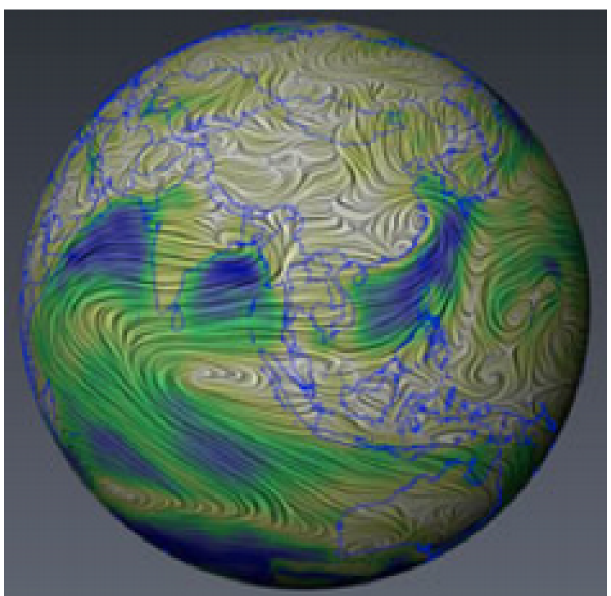
Viz examples: Geosciences



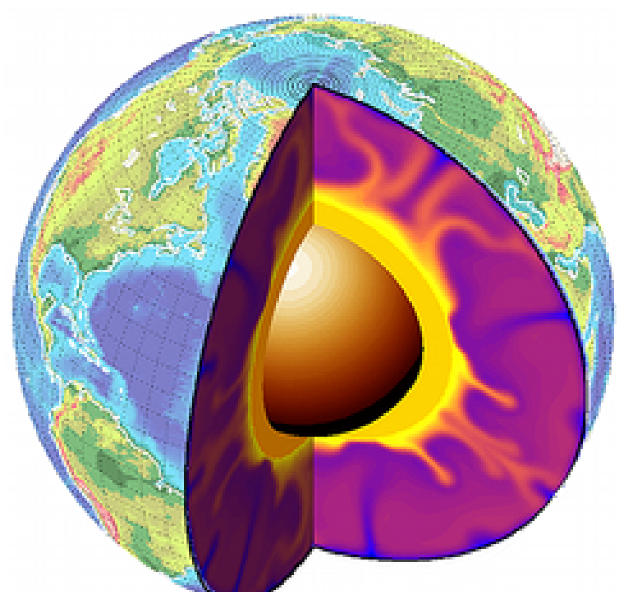
ocean velocity
and surface temperature



sea level pressure and temperature



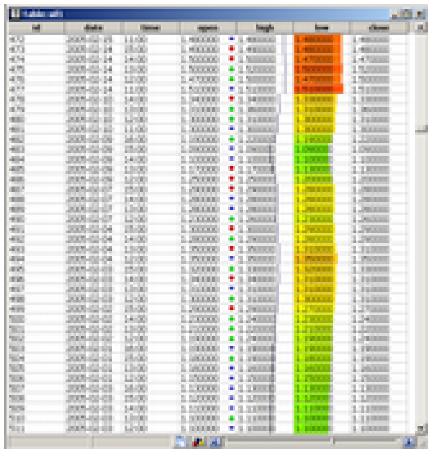
wind flow paths over
Earth's surface



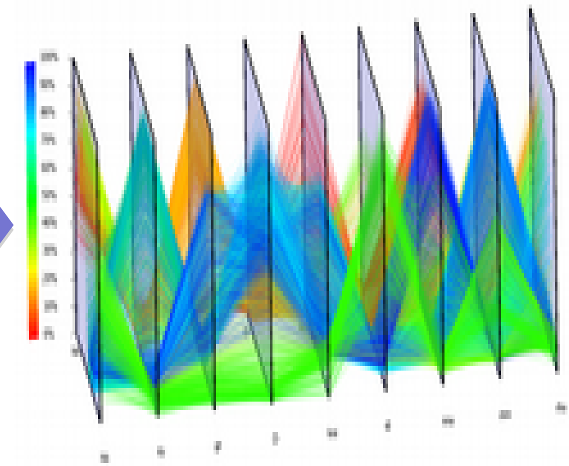
Earth surface and inner temperature

Viz examples: Abstract data

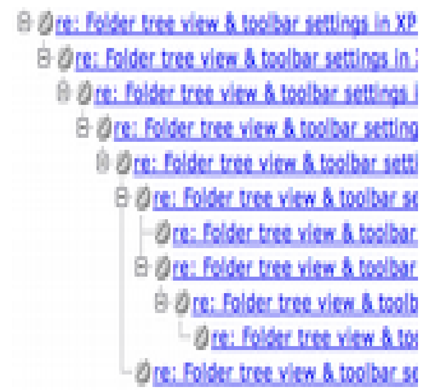
- mapping is not ‘neutral’ or natural, but reflects the problem/question to be solved



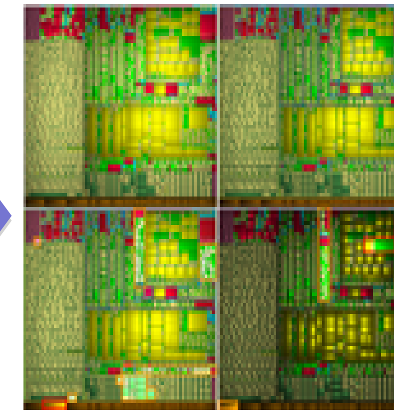
data table: classical view



data table: parallel coordinates view

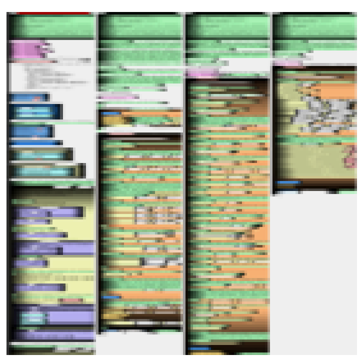


tree: explorer view

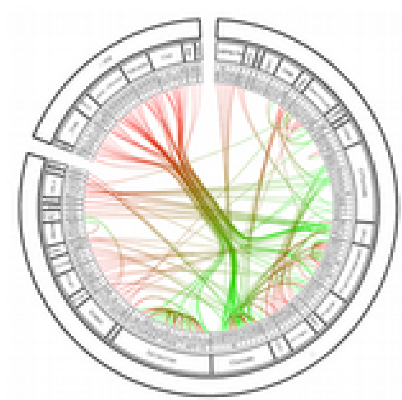


tree: cushion treemap view

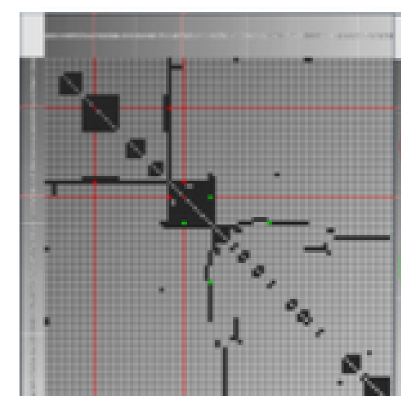
source code: classical view



source code: dense pixel view



graph: bundled view

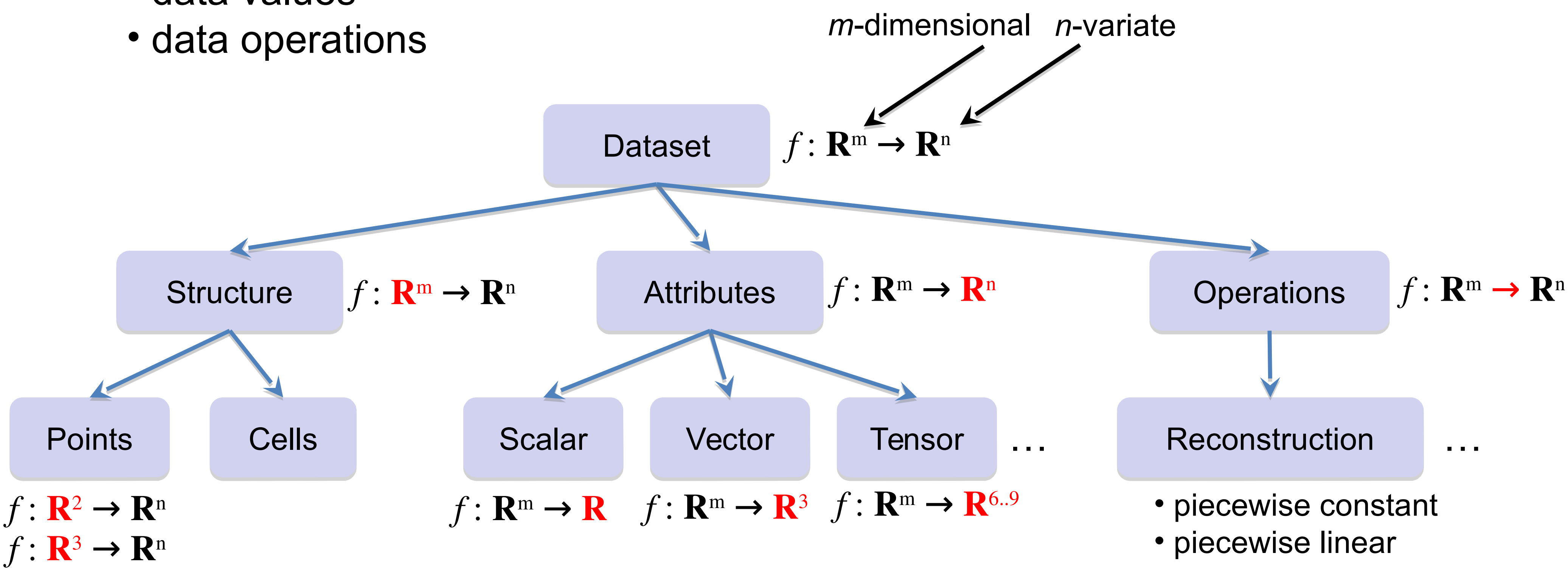


graph: adjacency matrix

Scientific Visualization – The Dataset

Dataset

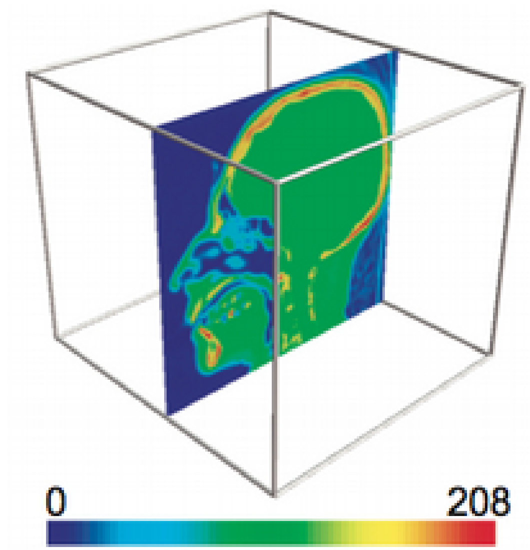
- key notion in visualization (SciVis, InfoVis)
- a dataset captures all relevant characteristics of a data collection
  - structure
  - data values
  - data operations



Our input: Dataset examples

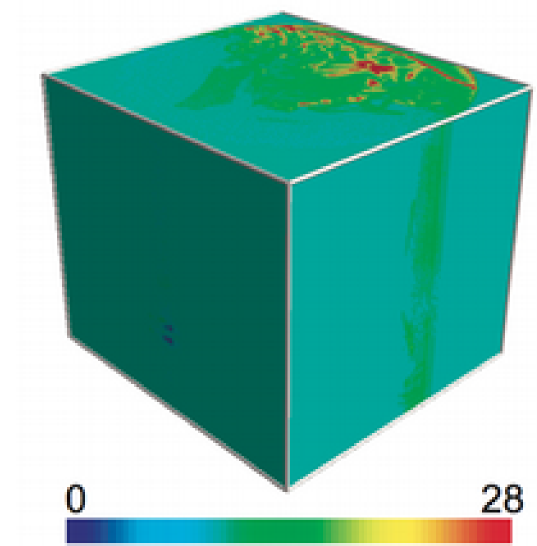
$f : \mathbf{R}^2 \rightarrow \mathbf{R}$

a planar slice



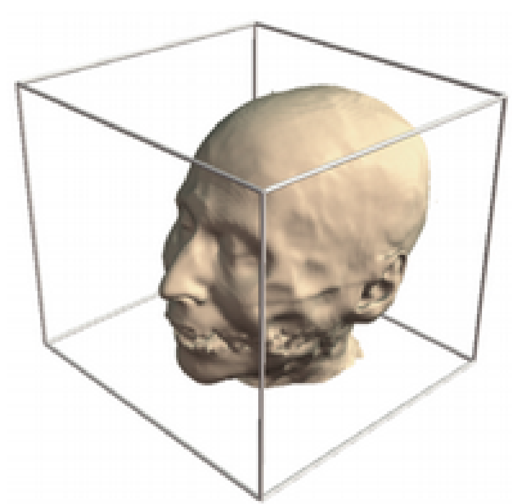
$f : \mathbf{R}^3 \rightarrow \mathbf{R}$

a volume



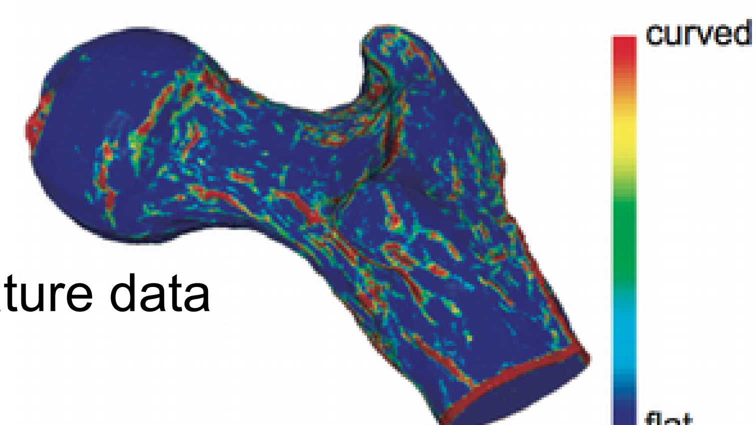
$f : \mathbf{R}^2 \rightarrow \mathbf{R}^0$

a surface



$f : \mathbf{R}^2 \rightarrow \mathbf{R}$

a surface
with curvature data



Our output: The image

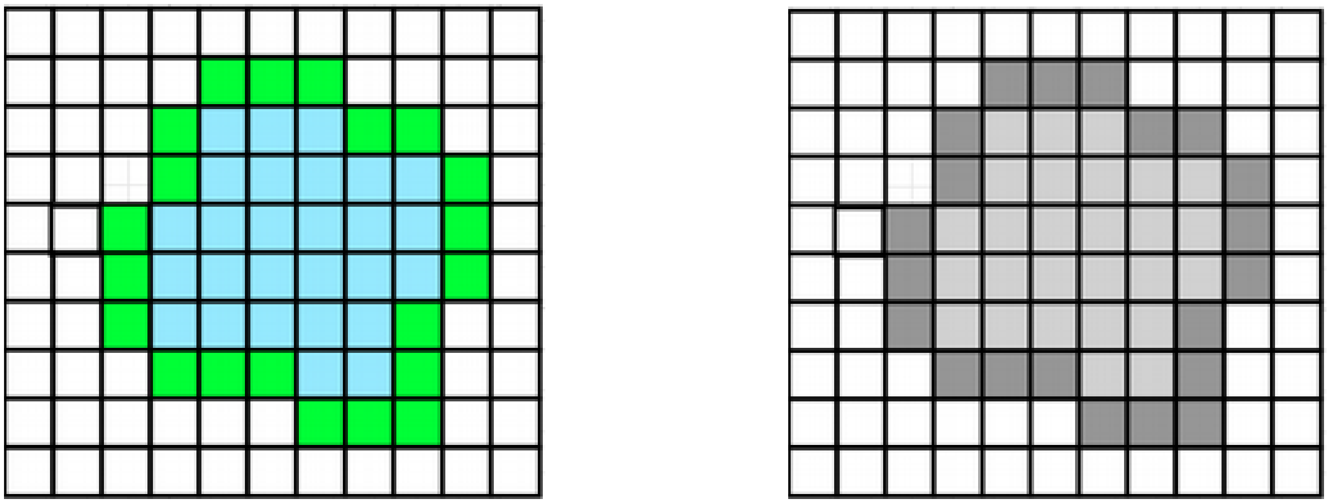
- domain: 2D space (the pixel positions)
- co-domain: the pixel (RGB) colors or grayscale values

color image

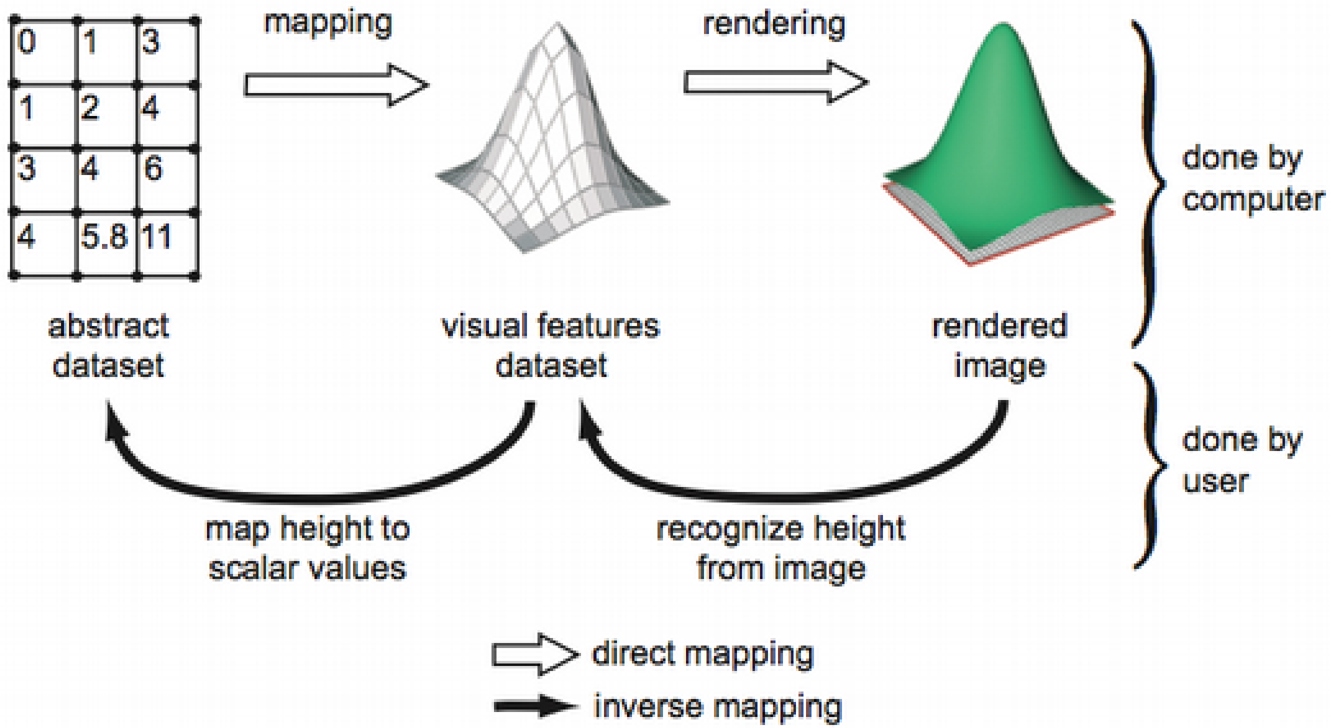
$f : \mathbf{R}^2 \rightarrow \mathbf{R}^3$

grayscale image

$f : \mathbf{R}^2 \rightarrow \mathbf{R}^+$



Functional view on visualization



- input: dataset in high-dimensional space $d \subseteq \mathbf{D}^{m \times n}$
- output: color image $i \subseteq \mathbf{R}^{2 \times 3} = \mathbf{R}^5$
- visualization: function $v : \mathbf{D}^m \rightarrow \mathbf{R}^5$ (from data to images)
- analysis: inverse function $v^{-1} : \mathbf{D}^m \rightarrow \mathbf{R}^5$ (from images to data)

Visualization Challenges

Dimensionality

- input dataset typically of much higher dimensionality than 2D images ($m+n$ • •2+3)
- where to put all those dimensions?

Data size

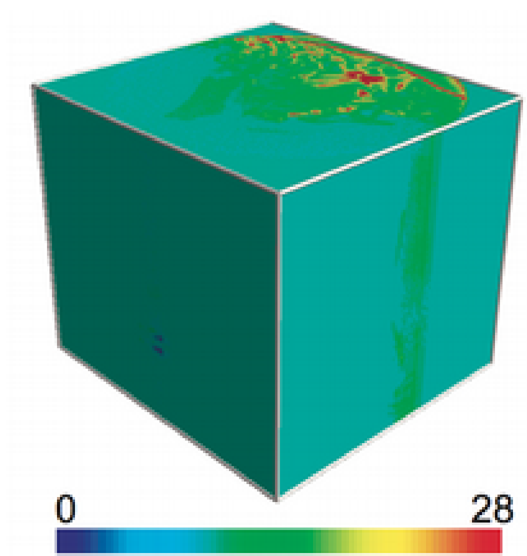
- input number of data points much higher than screen resolution
- where to draw all those data points?

Analysis

- visualization function not (fully) invertible
- how to go from shapes/colors back to data?

$f : \mathbf{R}^3 \to \mathbf{R}$

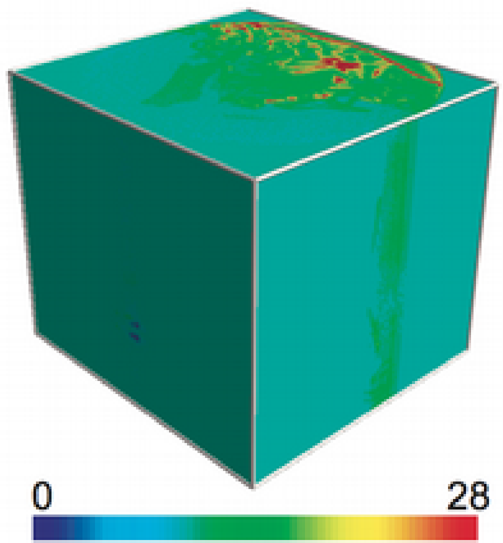
a volume



$v : \mathbf{D}^4 \to \mathbf{R}^5$

?? color image

Simple Solutions



filtering

extract slice

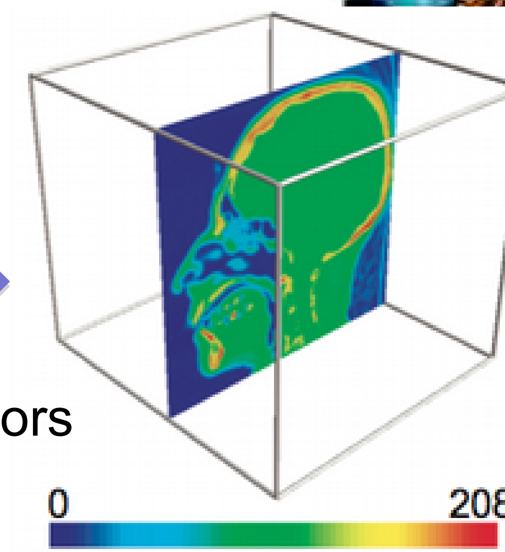
planar slice

$f : \mathbf{R}^2 \rightarrow \mathbf{R}$

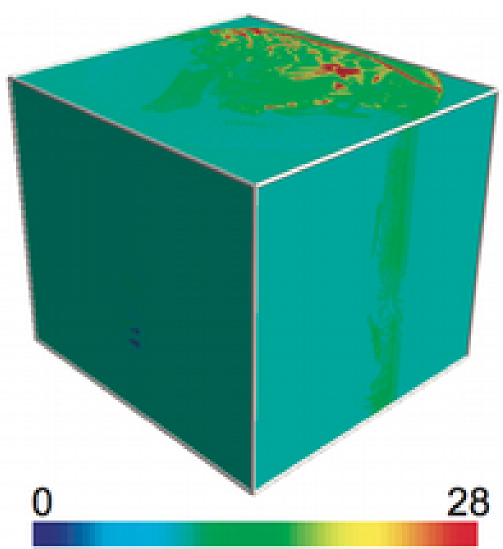
mapping

draw slice

map f to colors



data volume $f : \mathbf{R}^3 \rightarrow \mathbf{R}$



filtering

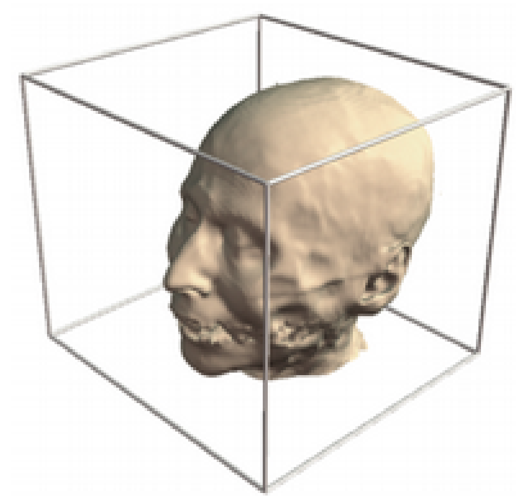
extract surface

surface

$f : \mathbf{R}^2 \rightarrow \mathbf{R}^0$

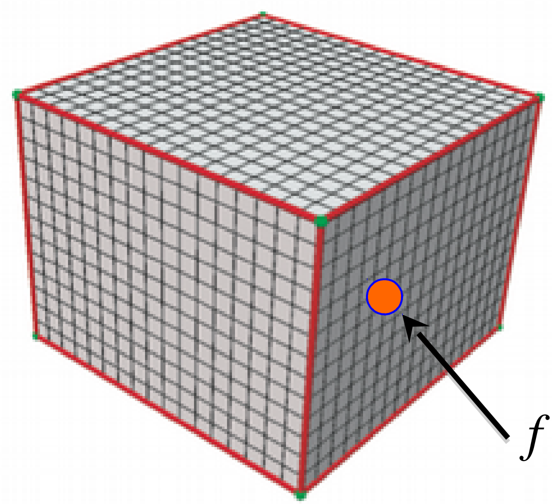
mapping

draw surface



More complex cases

Multi-variate data $f : \mathbf{R}^3 \rightarrow \mathbf{R}^{n>>1}$



where to draw all those n data values?

$f\,(x,y,z) = (f_1,\ldots,f_n)$

Multi-dimensional data $f : \mathbf{R}^{m>>3} \rightarrow \mathbf{R}$

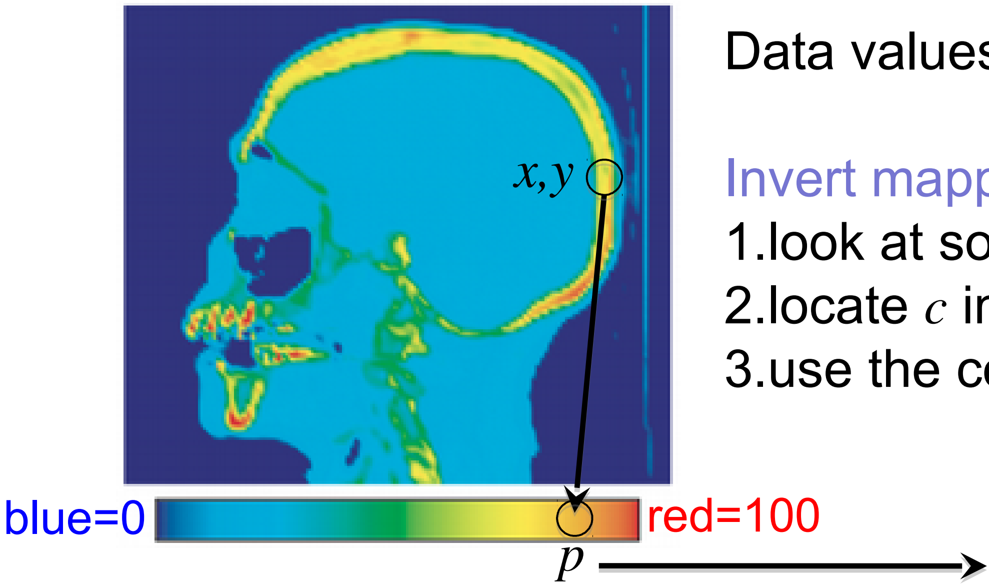
- **where** to draw all those m dimensions in a 2D image?

Non-spatial data $f : \mathbf{D} \rightarrow \mathbf{C}$ **D, C** are not subsets of $\mathbf{R}^k$

- graphs, trees, databases, software source code, …
- **how** to map D, C to image attributes (positions, color)?

Visualization challenges (cont'd)

How to make the visualization function v invertible?



Data values mapped to RGB colors via a colormap

Invert mapping:

1. look at some point (x,y) in the image→ color c
2. locate c in colormap at some position p
3. use the colormap legend to derive data value s from p

answer: $s = 90$

Problems

- what if we cannot distinguish colors well? (step 1)
- what if we cannot compare colors well? (step 2)
- what if the colormap is bad? (step 3, *e.g.* more values s_1, s_2 map to same color c)
- what if there's no color legend?
- what if there's no colormap?
- …

The Visualization Pipeline - Recall

any kind of data		formatted data		filtered data		spatial data		2D image
Dataset ➡	Input ➡	Dataset ➡	Filtering ➡	Dataset ➡	Mapping ➡	Dataset ➡	Rendering ➡	Dataset

1. **Input data**
   - your primary “raw” source of information
   - can be anything (measurements, simulations, databases, …)
2. **Formatted data**
   - converted to points, cells, attributes (discussed next in this module)
   - Ready to use for visualization algorithms
3. **Filtered data**
   - eliminates the unneeded **data**, adds the needed **information**
   - read and written by visualization algorithms
4. **Spatial (mapped) data**
   - has spatial embedding → can be **drawn**
5. **2D Image**
   - final image you look at to get your answers

interpolation: Cells

Recall the interpolation formula

$$\tilde{f} = \sum_{i=1}^{N} f_i \phi_i$$

This becomes very inefficient if

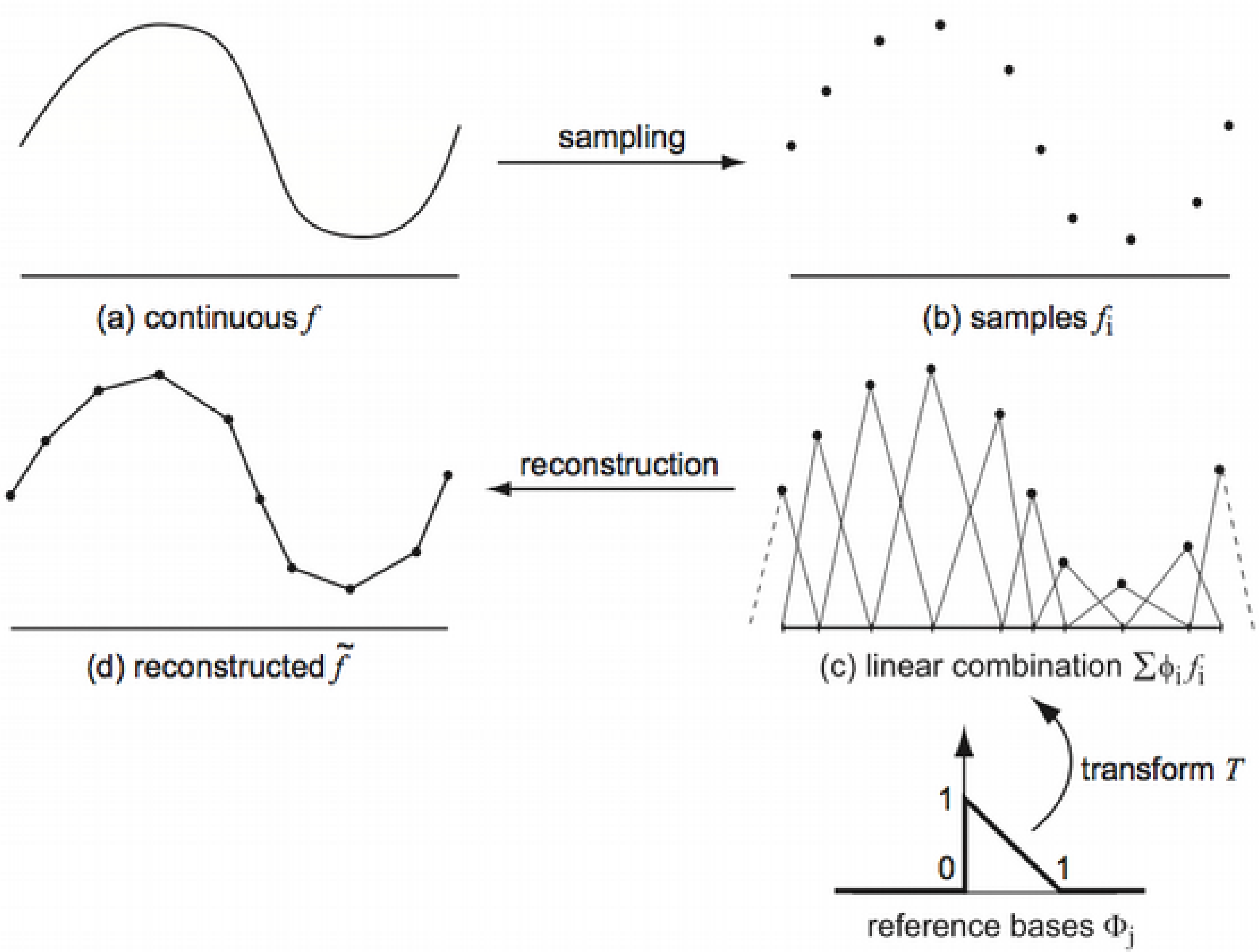
- N is very large and we have to evaluate φ_i at all these N points
- φ_i have complicated expressions

Practical basis functions

- are non-zero over small spatial 'pieces' of D only (limited support)
- have the same simple formula at all sample points p_i

We will discretize our spatial domain D into **cells**

Cells: 1D example



Remarks

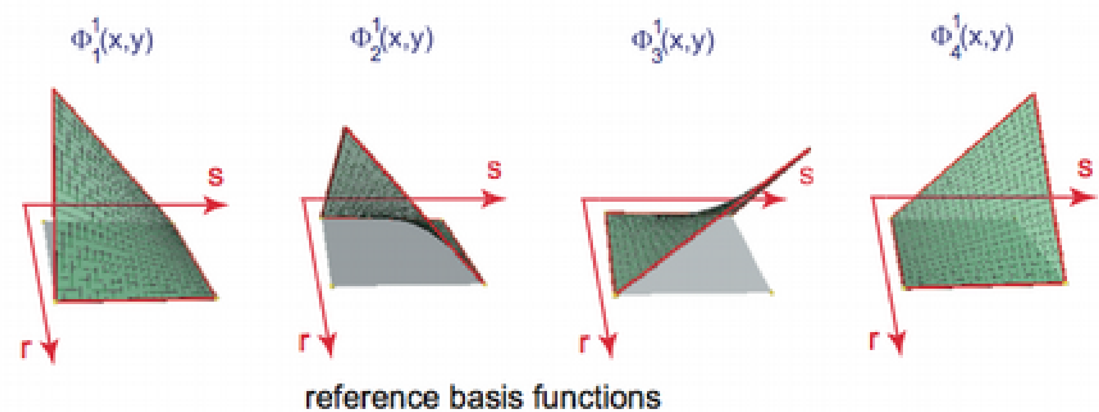
•interpolation & reconstruction goes cell-by-cell
•only need sample points at a cell vertices to interpolate over that cell
•reconstruction is C^1 because φ_i are C^1 and interpolation formula is are C^∞

2D cells: Quads

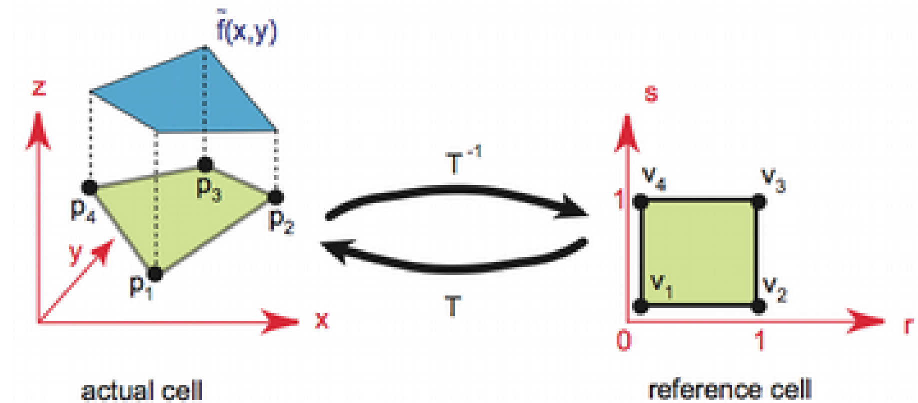
Same as in 1D case, but

- we have to decide on different cells; say we take quads
- quads → 4 vertices, 4 basis functions
- particular case: square cells = pixels

Bilinear basis functions



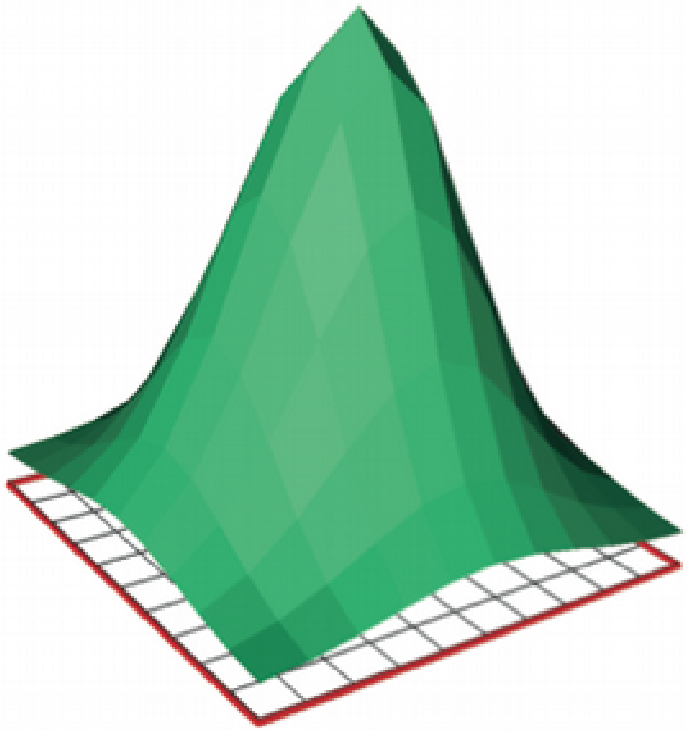
Bilinear transforms



$$
\begin{aligned}
\Phi_1^1(r,s) &= (1-r)(1-s),\\
\Phi_2^1(r,s) &= r(1-s),\\
\Phi_3^1(r,s) &= rs,\\
\Phi_4^1(r,s) &= (1-r)s;
\end{aligned}
$$

2D cells: Quads

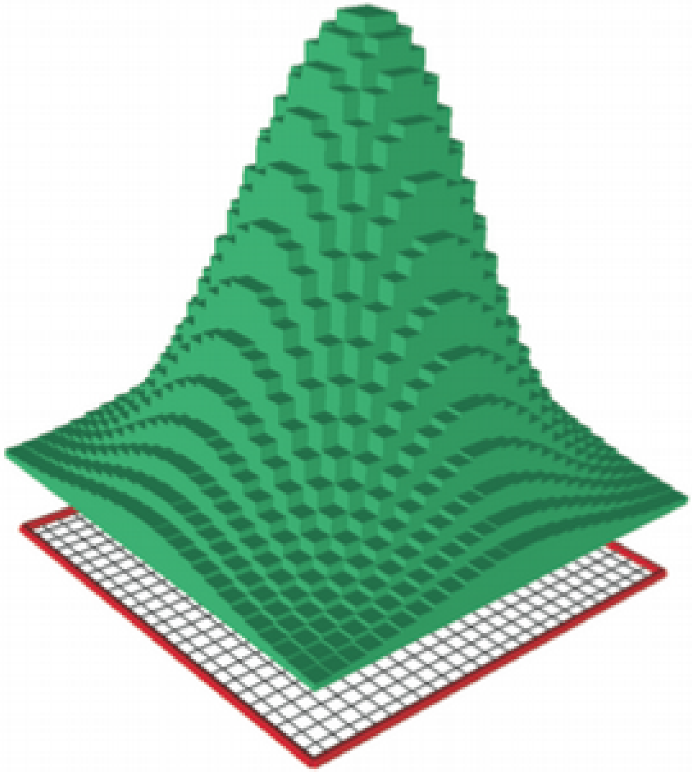
Bilinear interpolation



$$\Phi_1^1(r,s) = (1-r)(1-s),$$
$$\Phi_2^1(r,s) = r(1-s),$$
$$\Phi_3^1(r,s) = rs,$$
$$\Phi_4^1(r,s) = (1-r)s;$$

- 4 functions, one per vertex
- result: C^0 but never C^1 (why?)
- good for vertex-based samples

Constant interpolation

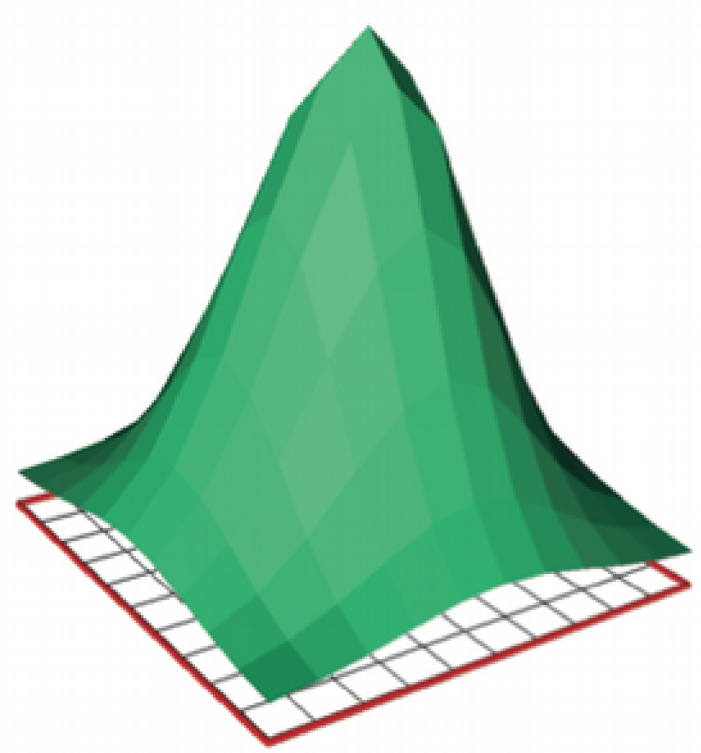


$$\phi_i^0(x) = \begin{cases} 1, & x \in c_i, \\ 0, & x \notin c_i. \end{cases}$$

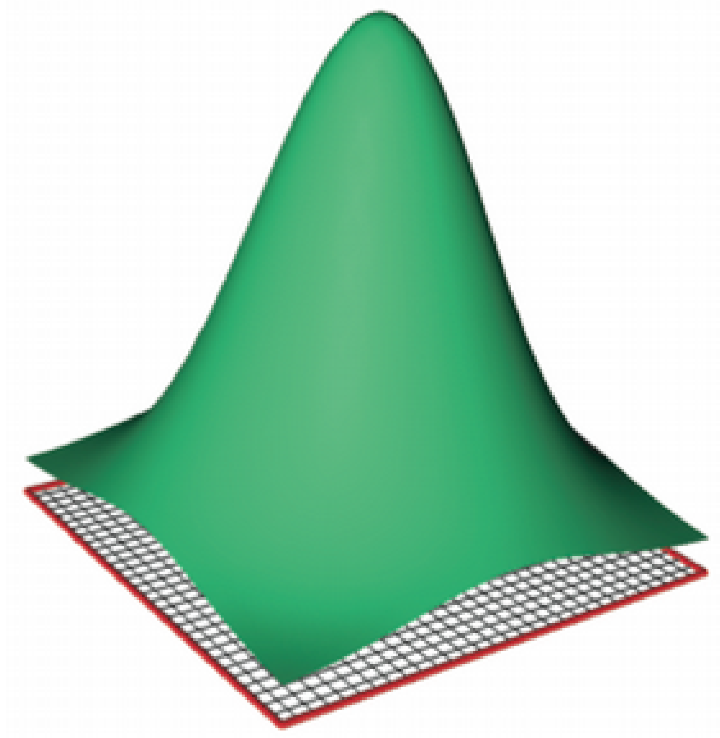
- 1 functions per whole cell
- result: not even C^0
- good for cell-based samples

Visual effects of interpolation options

What is the difference between flat and Gouraud (smooth) shading?



Flat shading

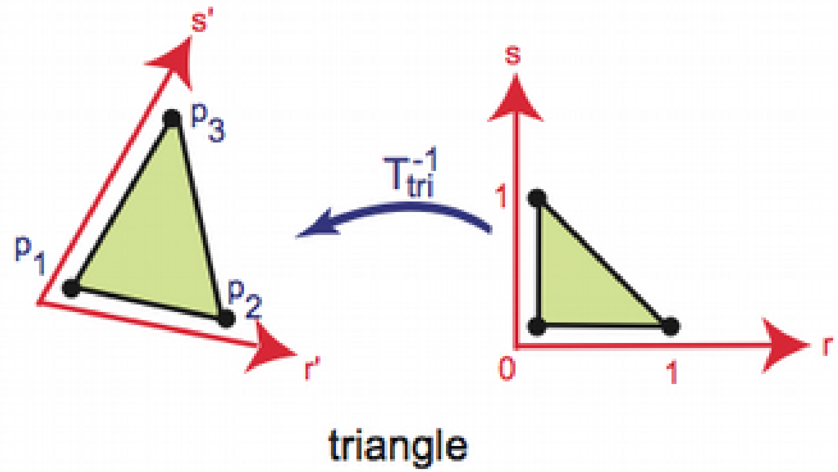


Gouraud shading

- surface: bilinear interpolation
- colors: constant interpolation

- surface: bilinear interpolation
- colors: bilinear interpolation

2D cells: Triangles



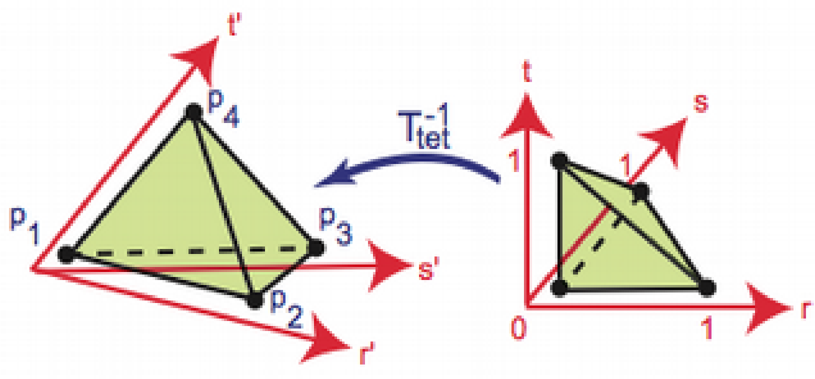
$$\Phi_1^1(r,s) = 1 - r - s,$$
$$\Phi_2^1(r,s) = r,$$
$$\Phi_3^1(r,s) = s.$$

Remarks

•triangles and quads offers largely same pro's and con's
•quad basis functions are not planes (they are **bi**linear)
•in graphics/visualization, triangles used more often than quads

- easier to cover complex shapes with triangles than quads
- same computational complexity

3D cells: Tetrahedra

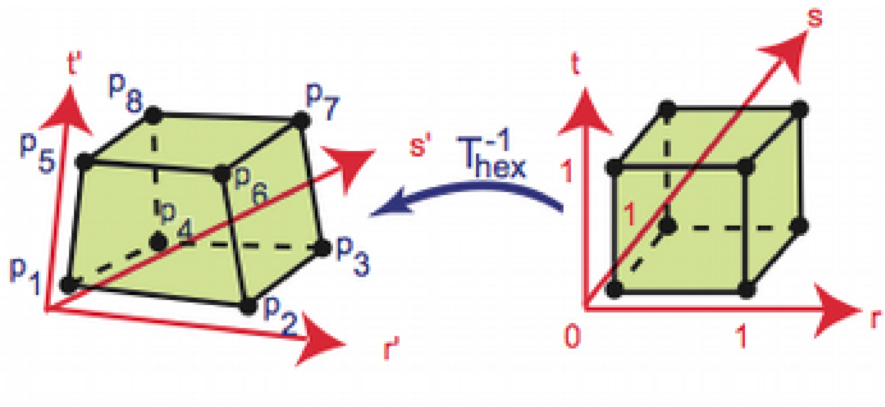


$$\Phi_1^1(r,s,t) = 1 - r - s - t,$$
$$\Phi_2^1(r,s,t) = r,$$
$$\Phi_3^1(r,s,t) = s,$$
$$\Phi_4^1(r,s,t) = t.$$

Remarks

- counterparts of triangles in 3D
- interpolate volumetric functions $f : \mathbf{R}^3 \to \mathbf{R}$
- three parametric coordinates r, s, t
- **tri**linear interpolation

3D cells: Hexahedra



hexahedron

$$\Phi^1_1(r,s,t) = (1-r)(1-s)(1-t),$$
$$\Phi^1_2(r,s,t) = r(1-s)(1-t),$$
$$\Phi^1_3(r,s,t) = rs(1-t),$$
$$\Phi^1_4(r,s,t) = (1-r)s(1-t),$$
$$\Phi^1_5(r,s,t) = (1-r)(1-s)t,$$
$$\Phi^1_6(r,s,t) = r(1-s)t,$$
$$\Phi^1_7(r,s,t) = rst,$$
$$\Phi^1_8(r,s,t) = (1-r)st.$$

Remarks

- counterparts of quads in 3D
- interpolate volumetric functions $f : \mathbf{R}^3 \to \mathbf{R}$
- trilinear interpolation
- particular case: cubic cells or voxels

Cell types for constant/linear basis functions

0D
•point
1D
•line
2D
•triangle, quad, rectangle
3D
•tetrahedron, parallelepiped, box, pyramid, prism, …

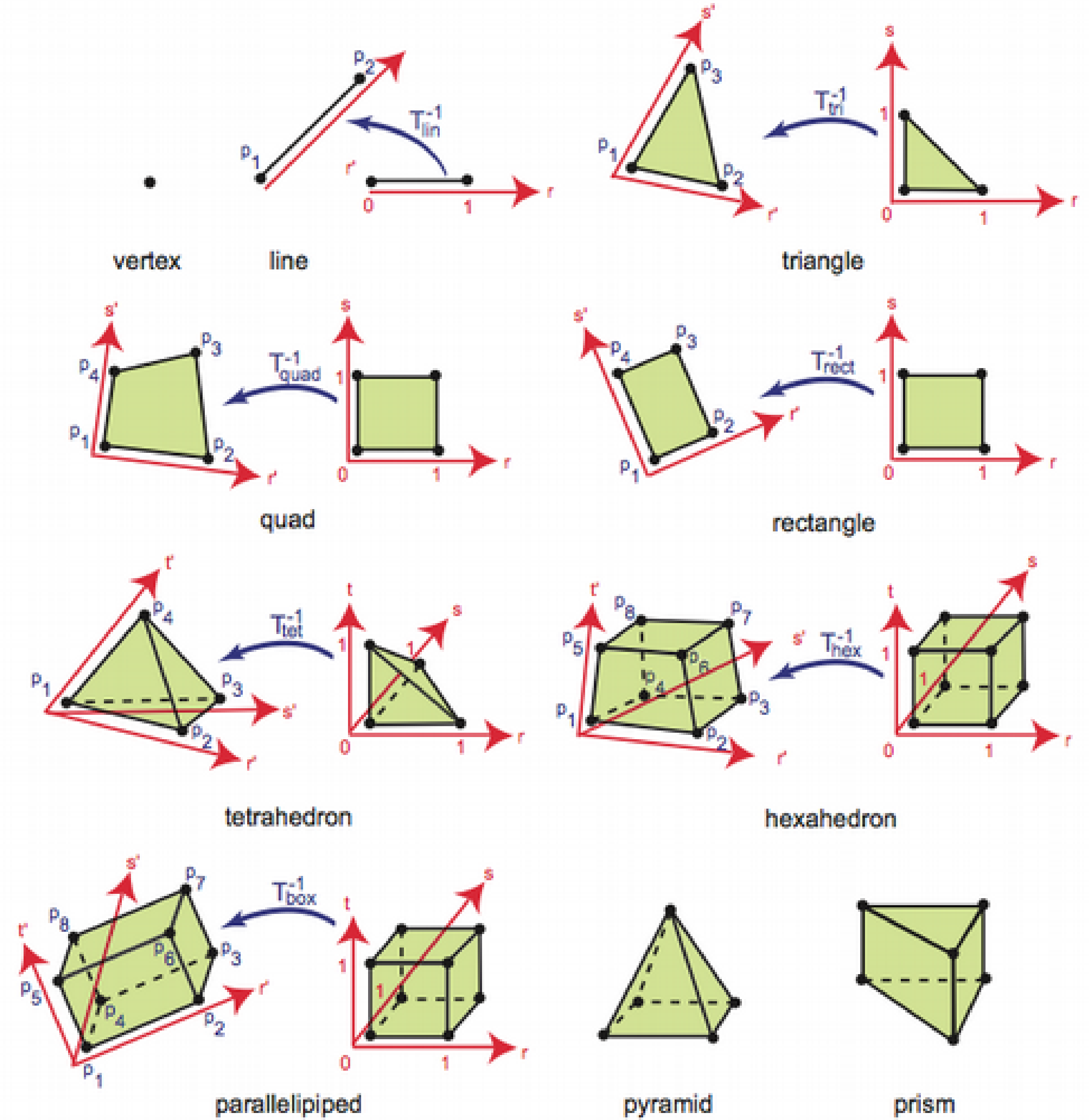


Figure 3.5. Cell types in world and reference coordinate systems.

From cells to grids

Cells

•provide interpolation over a small, simple-shapedspatial region

Grids

•partition our complex data domain D into cells

•allow applying per-cell interpolation (as described so far)

Given a domain D…

A grid G = {ci} is a set of cells such that

$c_i \cap c_j = \emptyset, \forall i \neq j$ no two cells overlap

$\bigcup_i c_i = D$ the cells cover all our domain

The dimension of the domain D constrains which cell types we can use: see next

Uniform grids

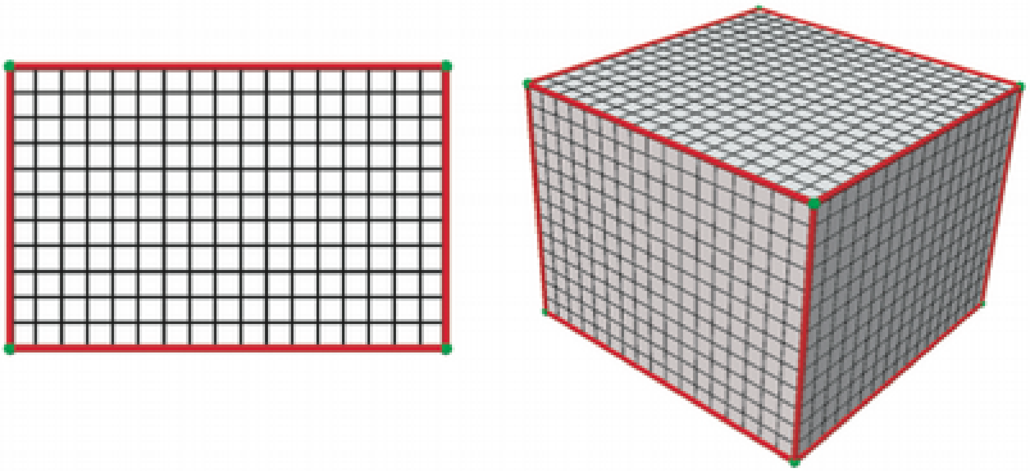
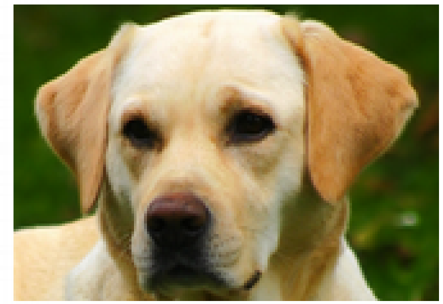
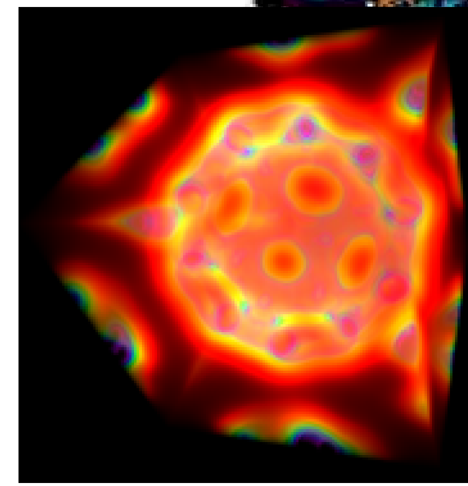


Figure 3.7. Uniform grids. 2D rectangular domain (left) and 3D box domain (right).



image



volume

- all cells have identical size and type (typically, square or cubic)
- cannot model non-axis-aligned domains

Storage requirements

- m integers for the #cells along each of the m dimensions of D (e.g. m=2 or 3)

Rectilinear grids

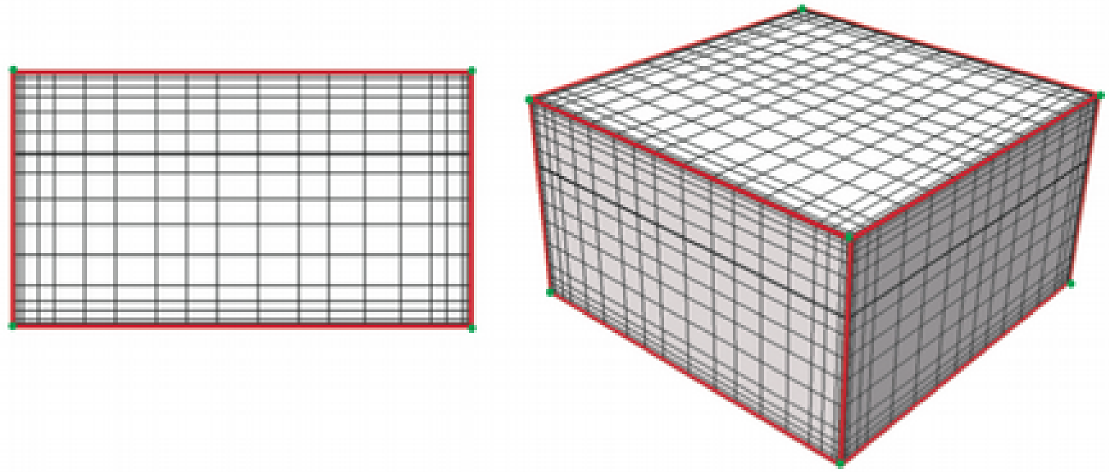


Figure 3.8. Rectilinear grids. 2D rectangular domain (left) and 3D box domain (right).

- all cells have same type
- cells can have different dimensions but share them along axes
- cannot model non-axis-aligned domains

Storage requirements

$\sum_{i=1}^{m} d_i$ floats (coordinates of vertices along each of the m axes of D)

Structured grids

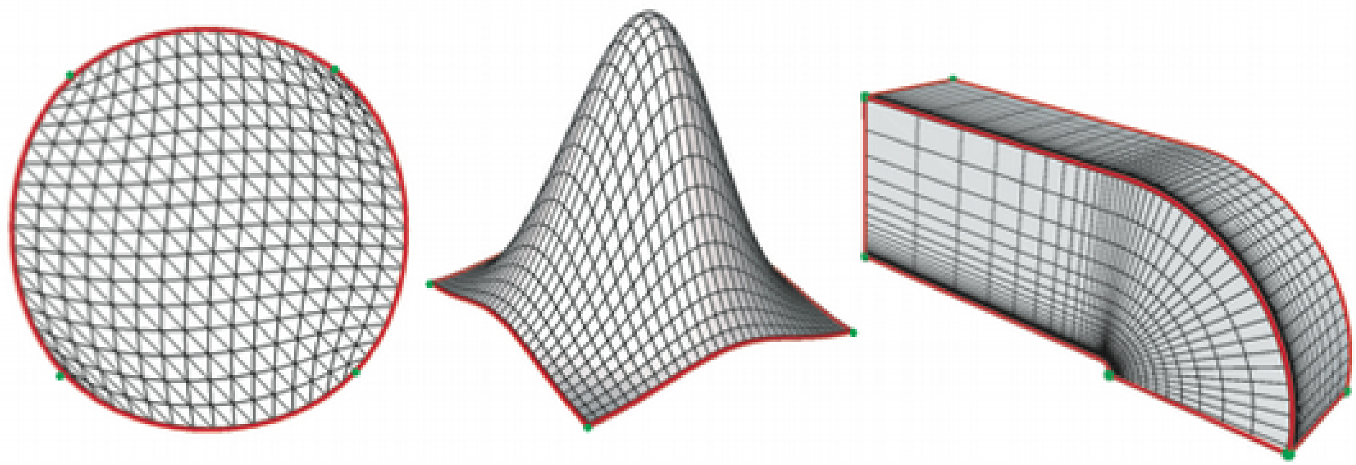


Figure 3.9. Structured grids. Circular domain (left), curved surface (middle), and 3D volume (right). Structured grid edges and corners are drawn in red and green, respectively.

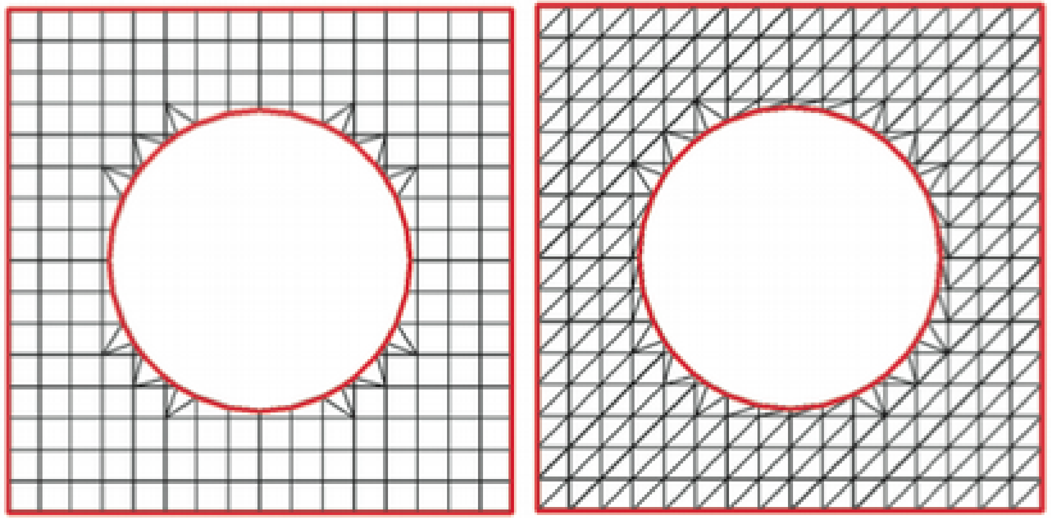
- all cells have same type
- cell vertex coordinates are freely (explicitly) specifiable…
- …as long as cells assemble in a matrix-like structure
- can approximate more complex shapes than rectilinear/uniform grids

Storage requirements

$\prod_{i=1}^{m} d_i$ floats (coordinates of all vertices)

Unstructured grids

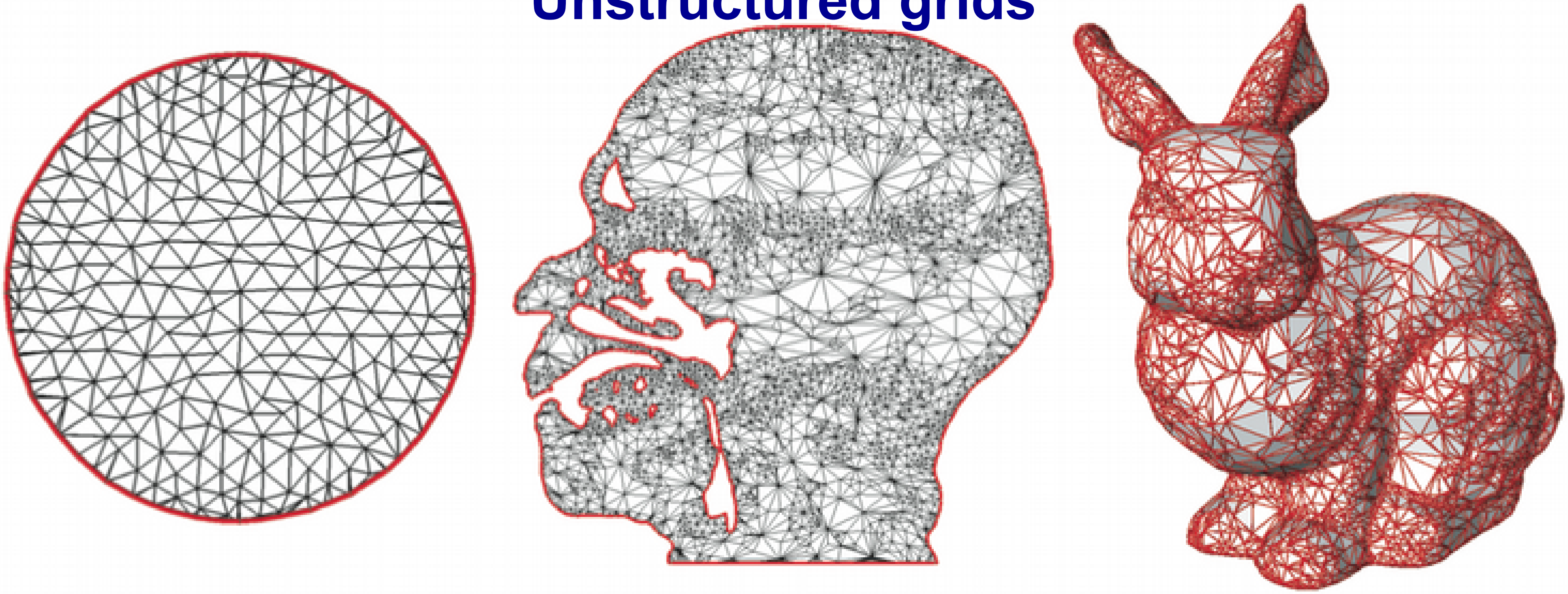
Consider the domain D: a square with a hole in the middle



We cannot cover such a domain with a structured grid (why?)

•it's not of genus 0, so cannot be covered with a matrix-like distribution of cells

Unstructured grids



- different cell types can be mixed (though it's not usual)
- both vertex coordinates and cell themselves are freely (explicitly) specifiable
- implementation

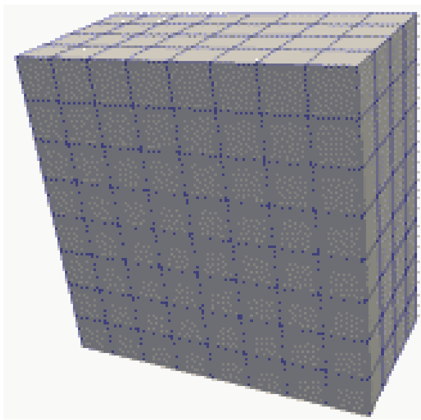
  vertex set $V = \{v_i\}$

  cell set $C = \{c_i = (indices\ of\ vertices\ in\ V)\}$
- most flexible, but most complex/expensive grid type

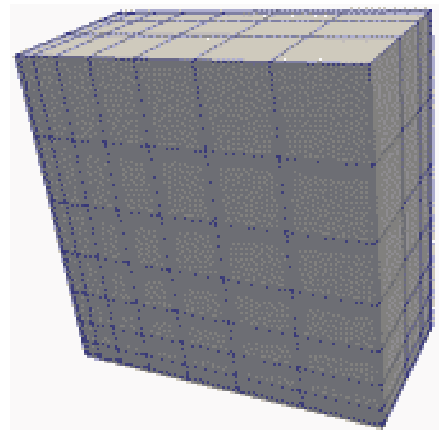
Storage requirements

$m\|V\| + s\|C\|$ for a m-dimensional grid with cells having s vertices each

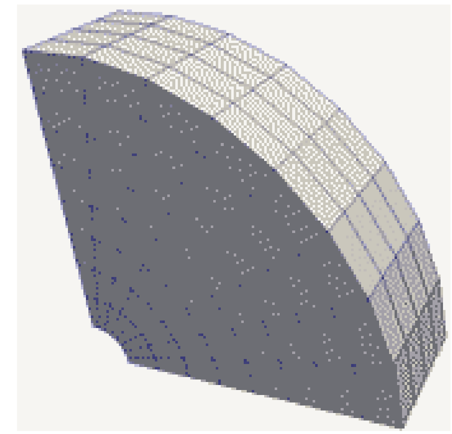
VTK and Paraview Data Types



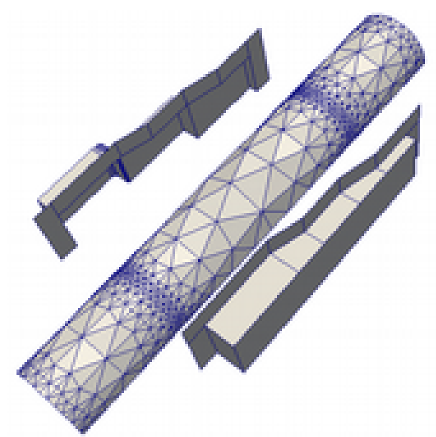
Uniform Rectilinear
(vtkImageData)



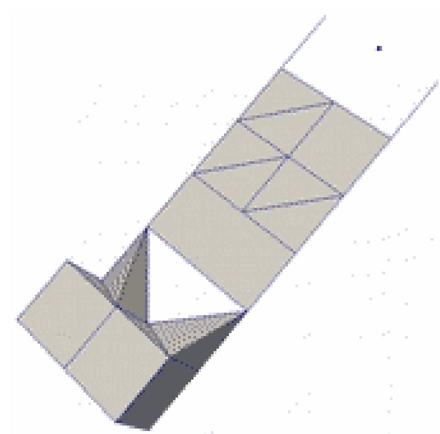
Non-Uniform Rectilinear
(vtkRectilinearData)



Curvilinear
(vtkStructuredData)



Polygonal
(vtkPolyData)



Unstructured Grid
(vtkUnstructuredGrid)

Multi-block

Hierarchical Adaptive Mesh Refinement (AMR)

Hierarchical Uniform AMR

Octree

Data attributes

$f : \mathbf{R}^m \rightarrow \mathbf{R}^n$

- n=0 no attributes (we model a shape only e.g. a surface)
- n=1 scalars (e.g. temperature, pressure, curvature, density)
- n=2 2D vectors
- n=3 3D vectors (e.g. velocity, gradients, normals, colors)
- n=6 symmetric tensors (e.g. diffusion, stress/strain – Modules 5..6)
- n=9 assymetric general tensors (not very common)

Remarks

- an attribute is usually specified for **all** sample points in a dataset (why?)
- different measurements will generate different attributes
- each attribute is interpolated separately
- different visualization methods for each n (see Module 3 next)

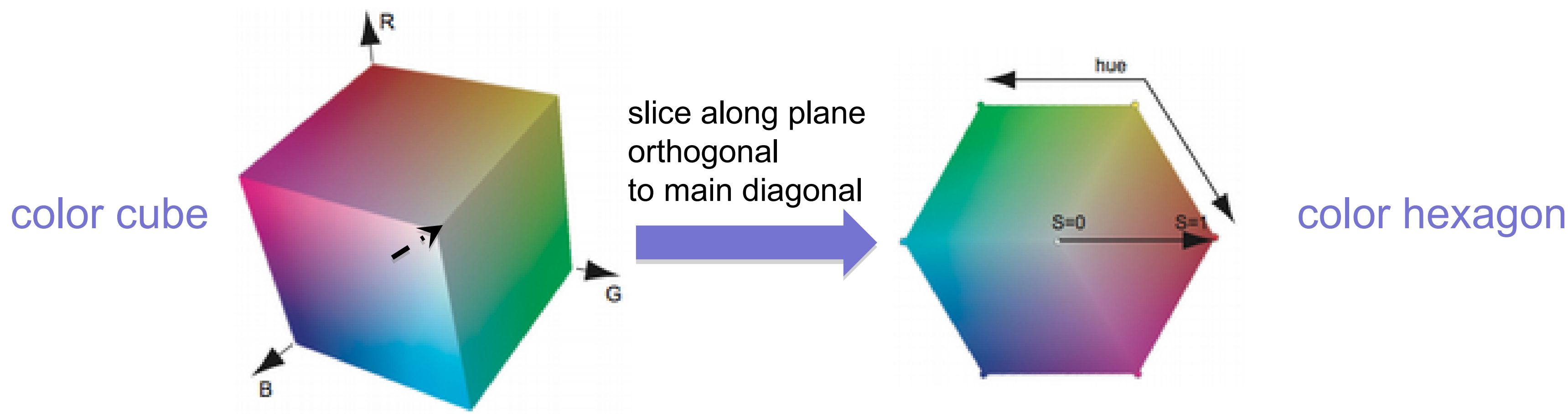
Data attributes: Color

- complex topic (measurement, perception, representation)
- we'll mainly focus on representation and a bit on perception

RGB color system

$$c = (c_R, c_G, c_B) \in [0,1]^3$$

- three floating-point components in $[0,1]$
- additive system (add, or mix, components to obtain result)



- perfect for synthesis (e.g. in the graphics card)
- unintuitive for humans, who think easier in **hues**

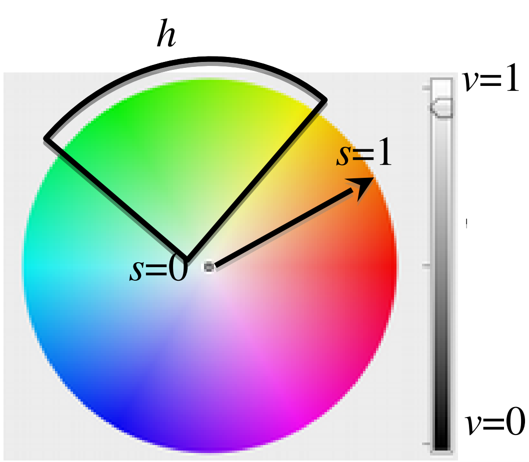
HSV color system

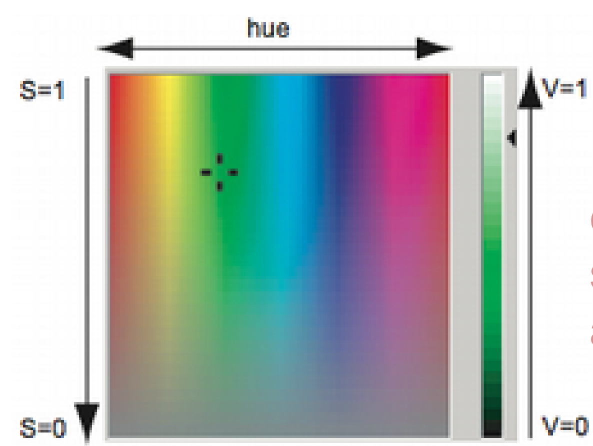
•three floating-point components in $[0,1]$

$$c = (h, s, v) \in [0,1]^3$$

•hue: tint of the color (red, green, blue, yellow, cyan, magenta, yellow, …)
•saturation: strong color (s=1), grayish color ($0<s<1$) or gray (s=0)
•value: luminance; white (v=1), dark ($0<v<1$), or black (v=0)

color wheel





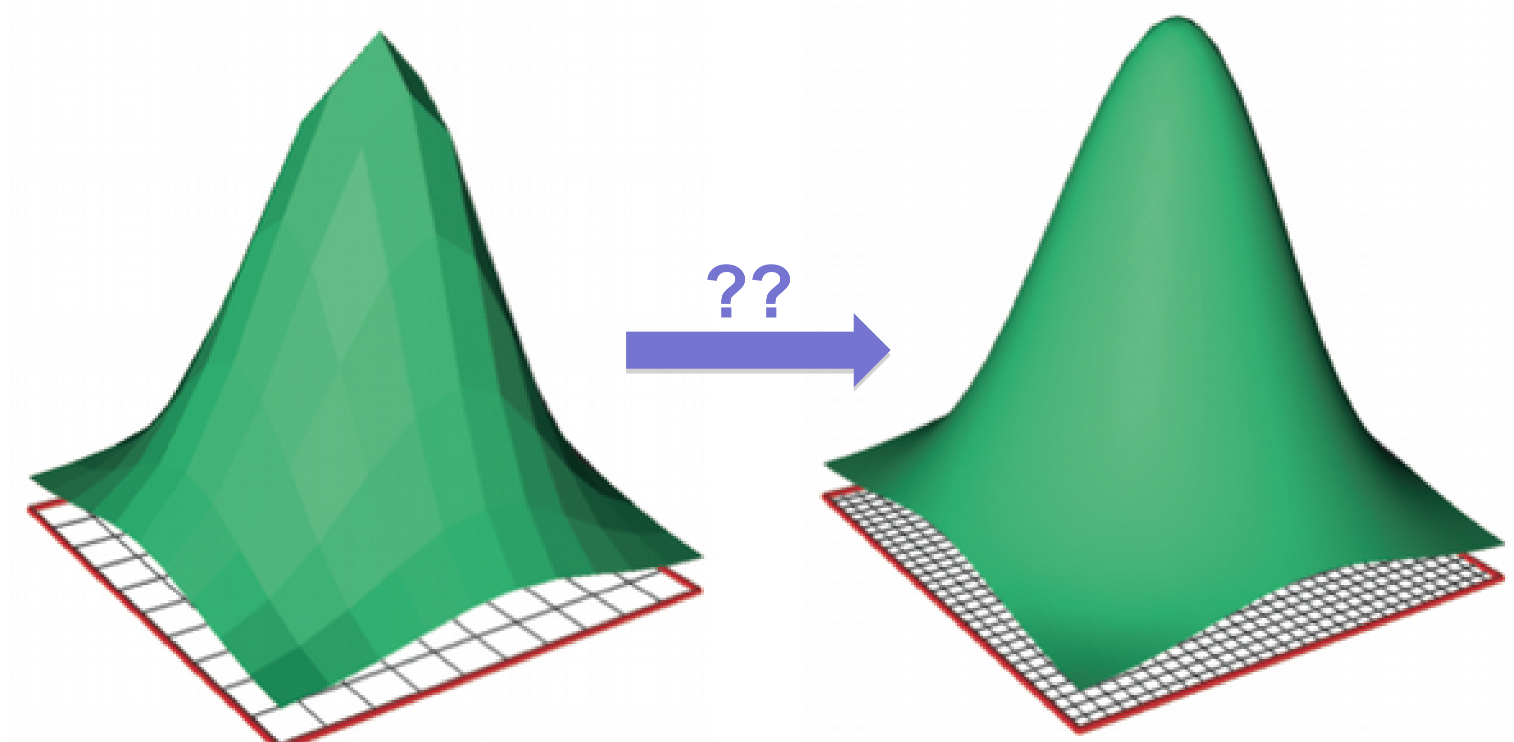
color wheel stretched onto a square

•HSV widgets: typically specify h and s in a 2D canvas and v separately (slider)
•show a 'surface slice' in the RGB cube

Data representation issues

Data resampling

- consider building a Gouraud-shaded surface plot



Flat shading

Gouraud shading

- how to compute vertex attributes (normals) when we have cell attributes?
  - normals computed per **cell**
  - normals required per **vertex**

Data resampling: cell data to vertex data

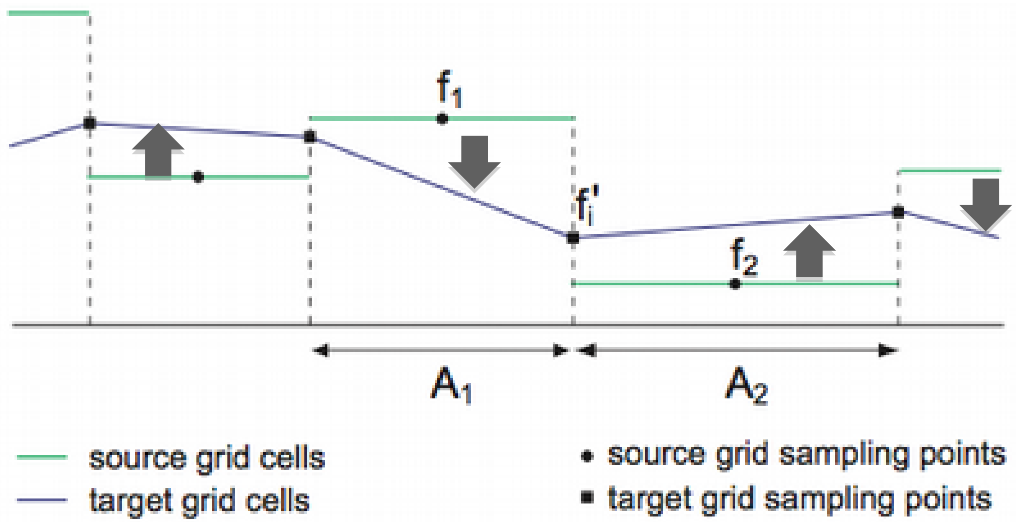


Figure 3.16. Converting cell to vertex attributes. The vertex value f_i' equals $\frac{A_1 f_1 + A_2 f_2}{A_1 + A_2}$, the area-weighted average of the cell values using vertex i.

Resampling a signal $\tilde{f}$ over some target domain D' should yield a 'similar' signal $\tilde{f}'$

$$\int_{c_i'} \tilde{f}' ds \approx \int_{c_i'} \tilde{f} ds, \quad \forall \text{ cells } c_i' \in \mathcal{D}' \quad \Longrightarrow \quad f_i' = \frac{\sum_{c_j \in \text{ cells}(p_i)} A(c_j) f_j}{\sum_{c_j \in \text{ cells}(p_i)} A(c_j)}$$

see Sec. 3.9.1

•this is the classical area-weighted normal averaging used in Gouraud shading

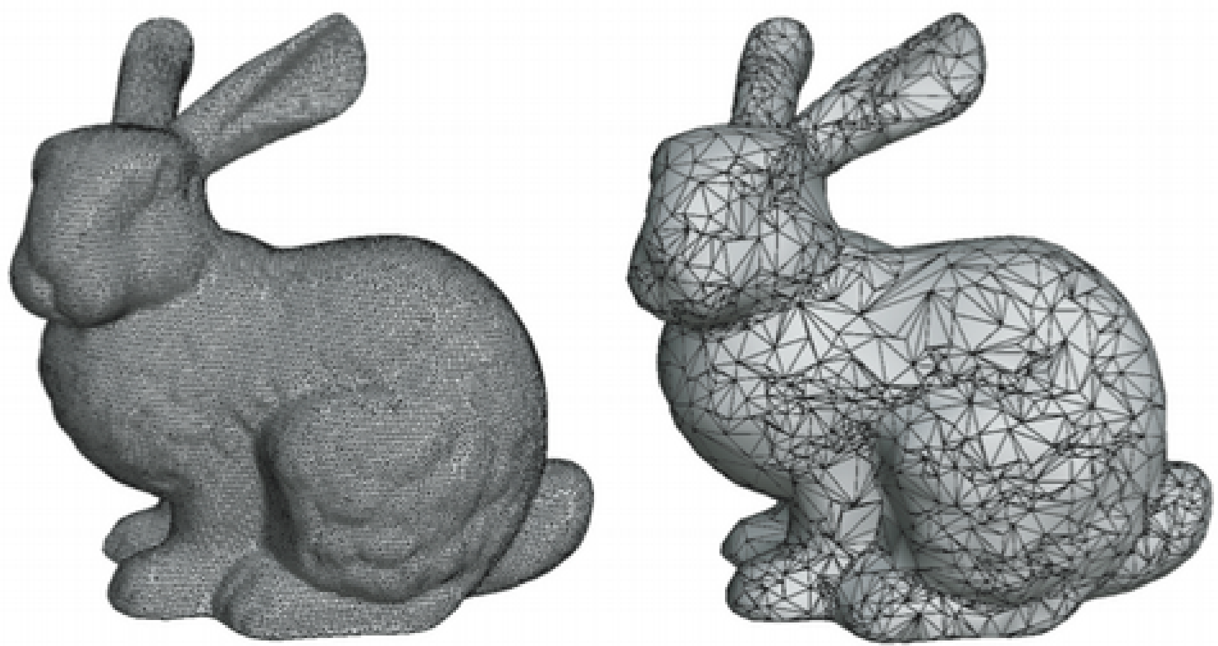
Resampling vertex data to cell data (same reasoning as above)

$$f_i = \frac{\sum_{p_j \in \text{ points}(c_i)} f_j'}{C}$$

•this is the classical averaging of vertex values to compute cell values

Data super/subsampling

- we have data on some grid
- we want data on a 'similar' grid having more or less cells
- the interpolation functions stay the **same** (unlike in resampling)



unstructured grid
36000 vertices

subsampling →
← supersampling

unstructured grid
3510 vertices