Data Representation and Scalar Algorithms

Anders Hast

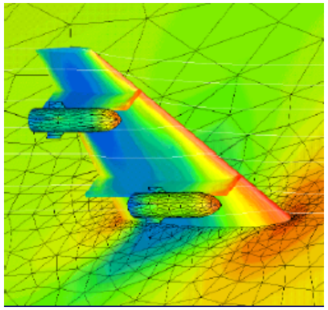
with some slides by

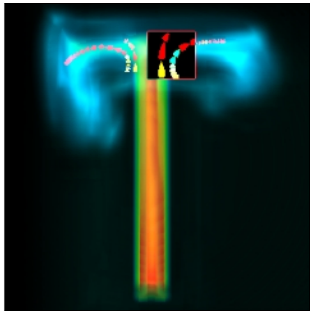
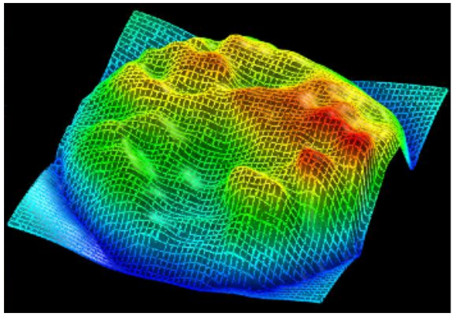
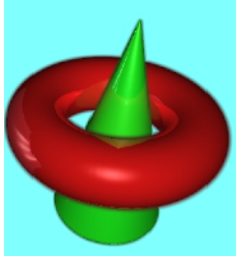
Alexandru C. Telea

(indicated)

Scientific Visualization Module 2

Data representation



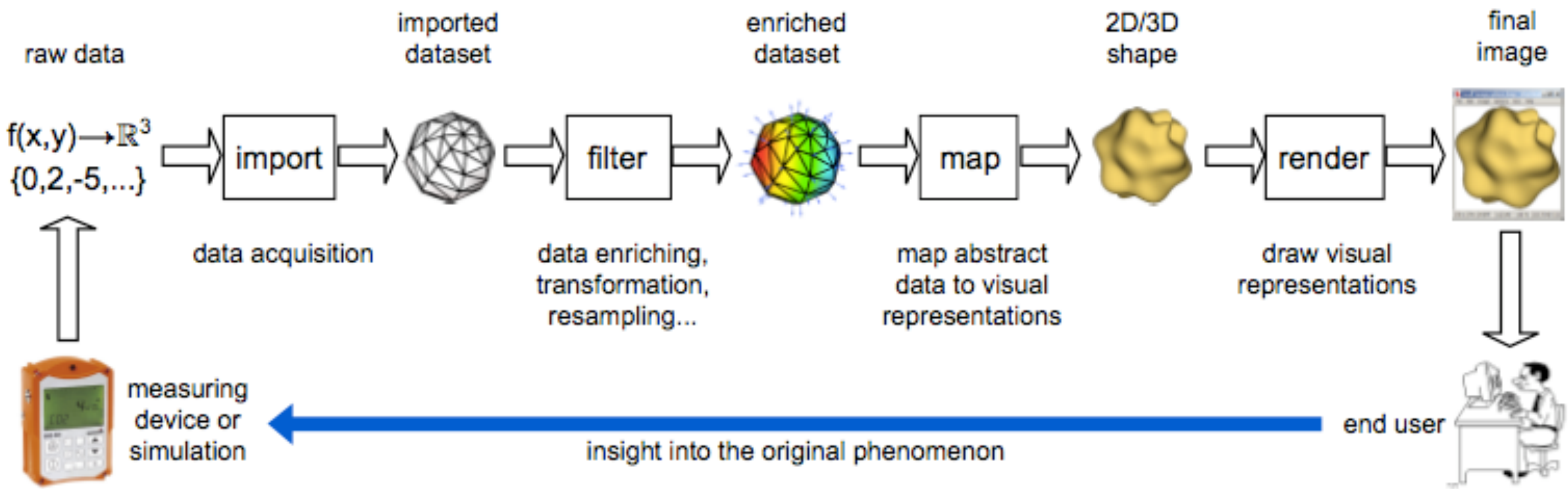




prof. dr. Alexandru (Alex) Telea

Department of Mathematics and Computer Science
University of Groningen, the Netherlands

The Visualization Pipeline - Recall

any kind of data		formatted data		filtered data		spatial data		2D image
Dataset	→ Process	→ Dataset	→ Process	→ Dataset	→ Process	→ Dataset	→ Process	→ Dataset



The Visualization Pipeline - Recall

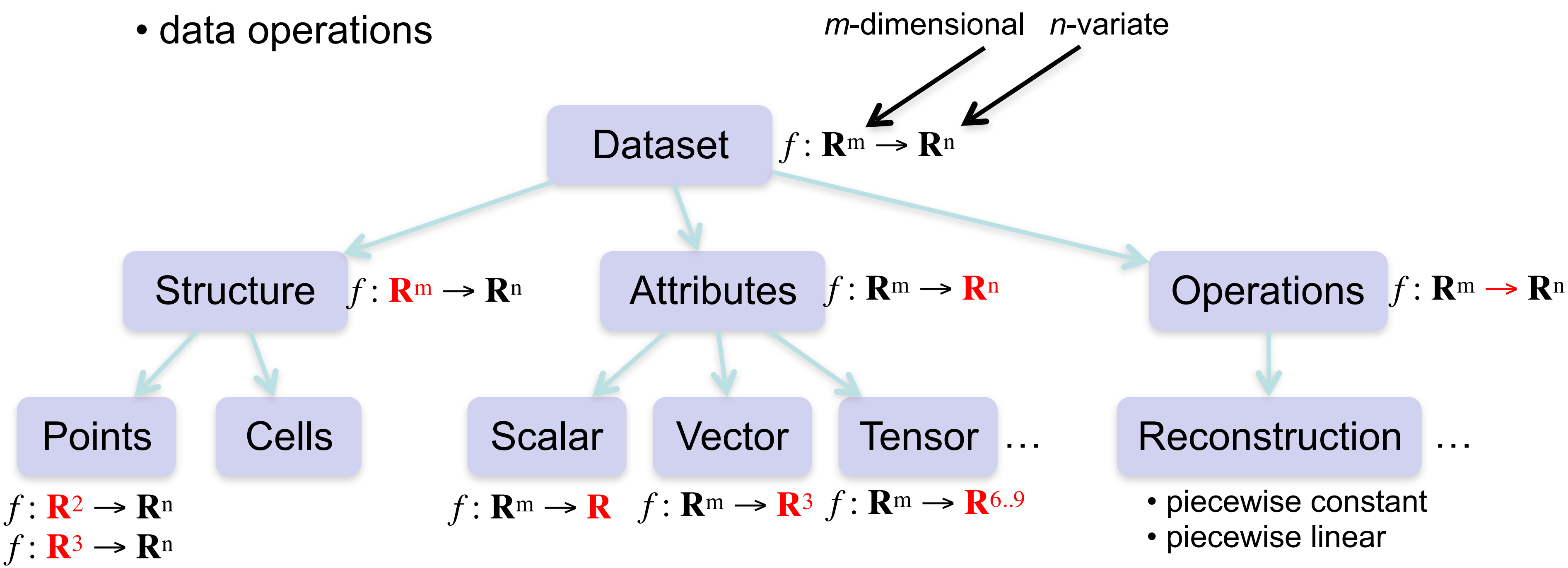
any kind of data		formatted data		filtered data		spatial data		2D image
Dataset →	Input →	Dataset →	Filtering →	Dataset →	Mapping →	Dataset →	Rendering →	Dataset

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

Scientific Visualization - The Dataset

Dataset

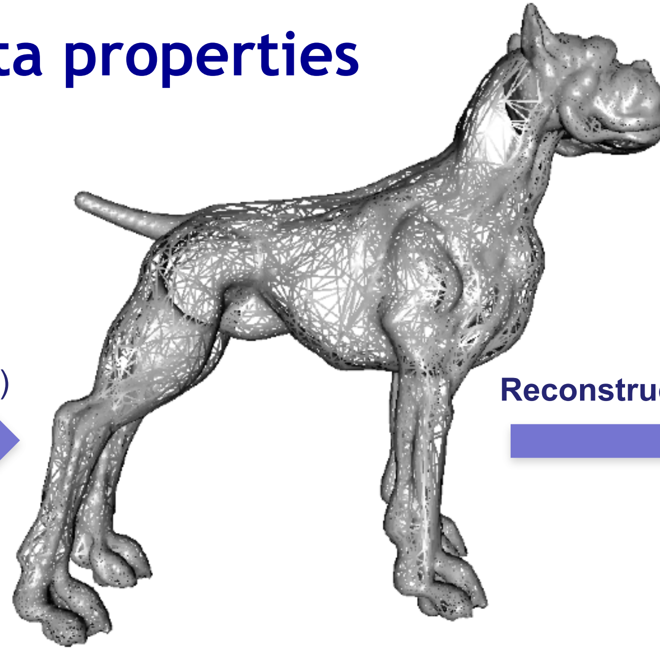
- key notion in visualization (SciVis, InfoVis, SoftVis)
- captures all relevant characteristics of a data collection
  - structure
  - data values
  - data operations

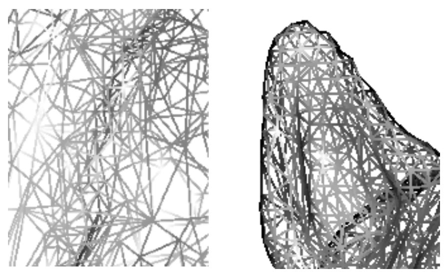


We'll detail all these next

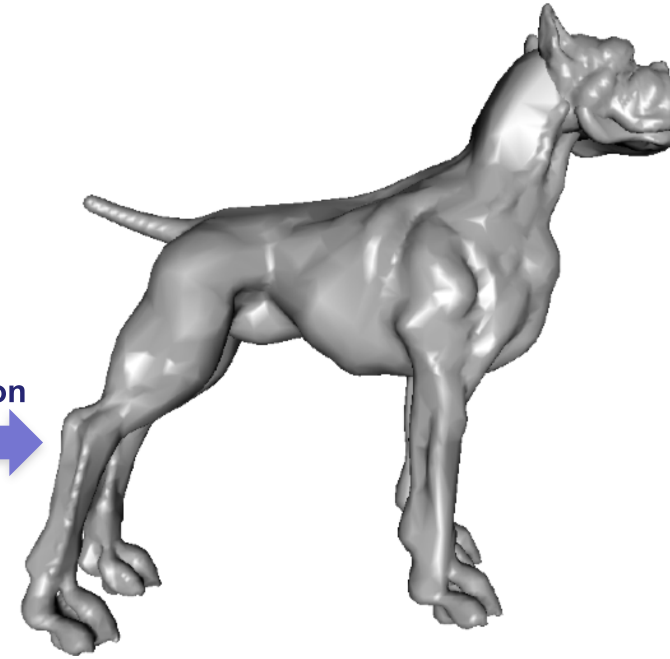
Visualization data properties

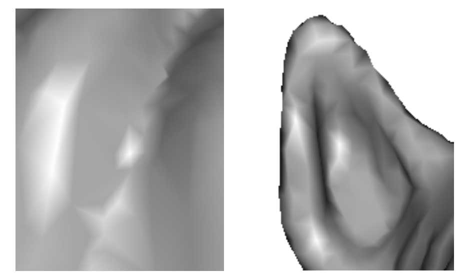
Sampling
(data importing)





Reconstruction





Domain — Co-domain

$$f : \mathrm{D} \rightarrow \mathrm{C}$$

Continuous data

Sampling

$$\{p_i, f_i\} \quad p_i \in \mathrm{D} \quad f_i \in \mathrm{C}$$

Measurements (samples) at *discrete* set of points

Reconstruction

$$\tilde{f} : \mathrm{D} \rightarrow \mathrm{C} \quad \tilde{f}(p_i) = f(p_i) = f_i$$

Continuous data, as close as possible to input

Continuous data

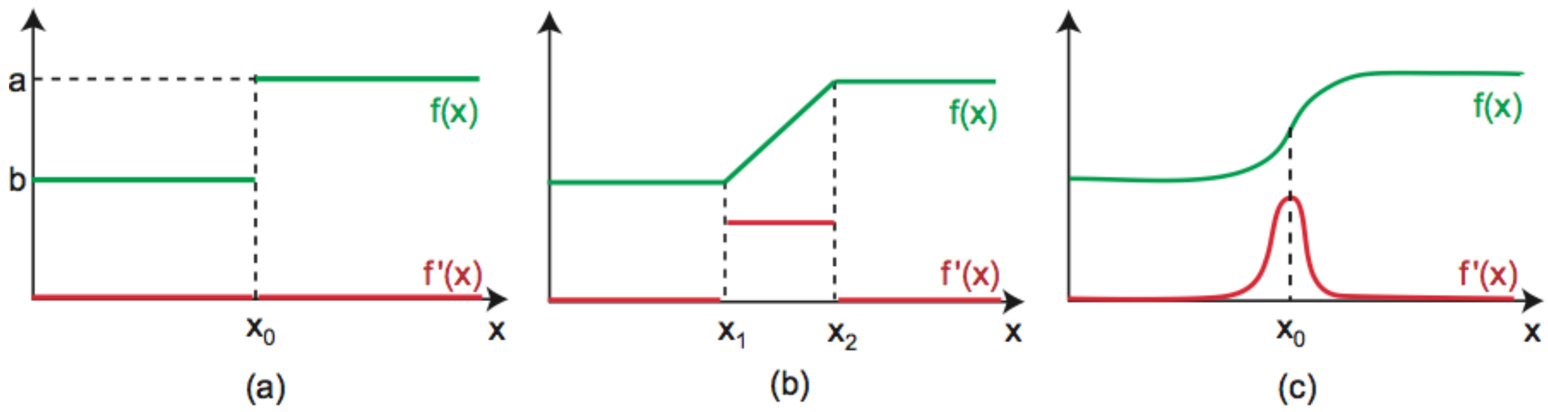


Figure 3.1. Function continuity. (a) Discontinuous function. (b) First-order $\mathcal{C}^0$ continuous function. (c) High-order $\mathcal{C}^k$ continuous function.

Cauchy definition of continuity

A function f is continuous iff

$$\forall \epsilon > 0, \exists \delta > 0 \text{ such that if } \|x - p\| < \delta, x \in \mathrm{C} \text{ then } \|f(x) - f(p)\| < \epsilon.$$

- $\mathcal{C}^{-1}$ discontinuous (graph of function has “holes”)
- $\mathcal{C}^{0}$ first-order continuous (graph of function has “kinks”)
- $\mathcal{C}^{k}$ first k derivatives of the function are continuous

Sampled data

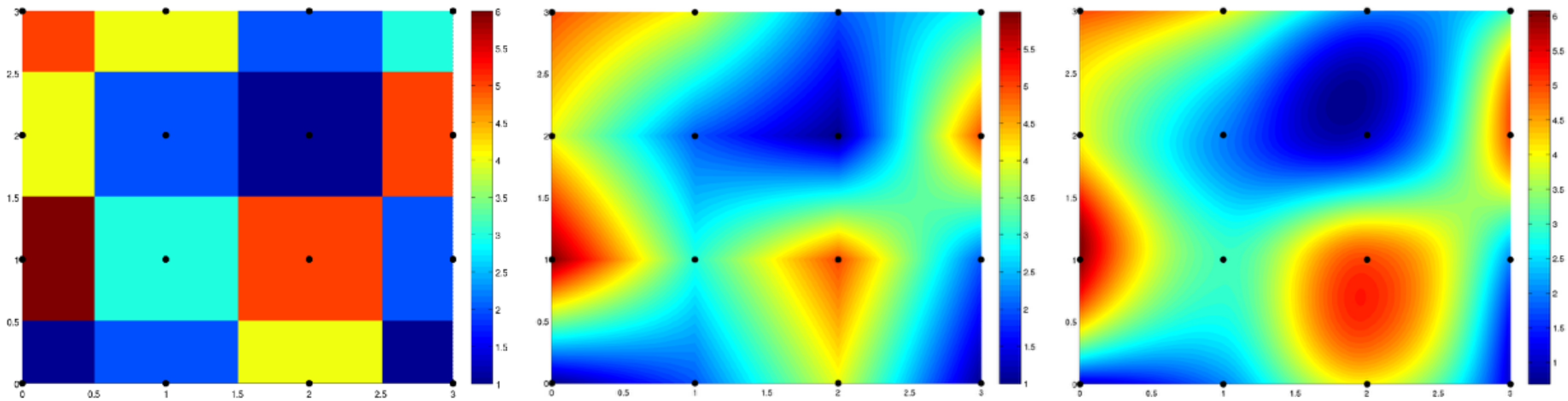
Functional properties

- **finite**
    - captures continuous signal at a finite set of points (measurements)
- **accurate**
    - can reconstruct a signal close to input accurately
    - reconstruction guarantees continuity properties

Non-functional properties

- **efficient**
    - reconstruction is fast
- **compact**
    - store Gbytes of sample points compactly
- **generic**
    - few data structures cover most dataset types
- **simple**
    - learn to create & use such data structures quickly

Why Interpolate?



Nearest neighbour | Bilinear | Bicubic

- Interpolation usually gives a better representation of the sampled data
- Interpolation is always a “guess” of what the “missing" data would be like.

https://en.wikipedia.org/wiki/Interpolation

Interpolation

Fundamental tool for signal reconstruction

Doesn't necessarily pass through sample values = approximation

1. **Reconstruction** formula

$$\tilde{f} = \sum_{i=1}^{N} f_i \phi_i \qquad \phi_i : \mathrm{D} \rightarrow \mathrm{C}$$ are basis (or interpolation) functions

2. **Interpolation**: reconstruction passes through (interpolates) the sampled values

$$\sum_{i=1}^{N} f_i \phi_i(p_j) = f_j, \forall j.$$ because $\tilde{f}(p_i) = f(p_i) = f_i$

3. **Orthogonality** of basis functions

$$\phi_i(p_j) = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$$ why? Just apply (2) to $f = \begin{cases} 1, p = p_j \\ 0, p \neq p_j \end{cases}$

4. **Normality** of basis functions

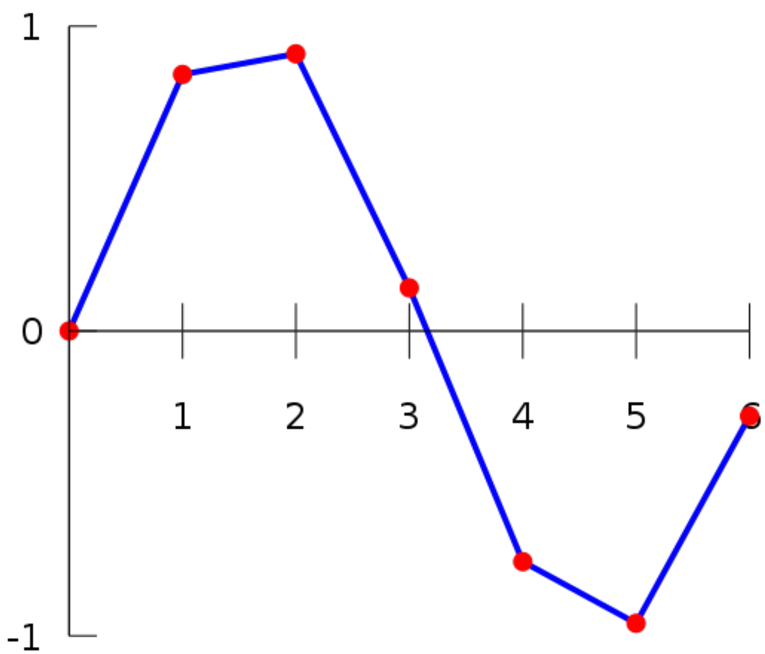
$$\sum_{i=1}^{N} \phi_i(x) = 1, \forall x \in \mathrm{D}$$ why? $\sum_{i=1}^{N} \phi_i(p_j) = 1, \forall p_j$ (sum (3) over $i = 1..N$)

and apply above to all $p_i \in D$

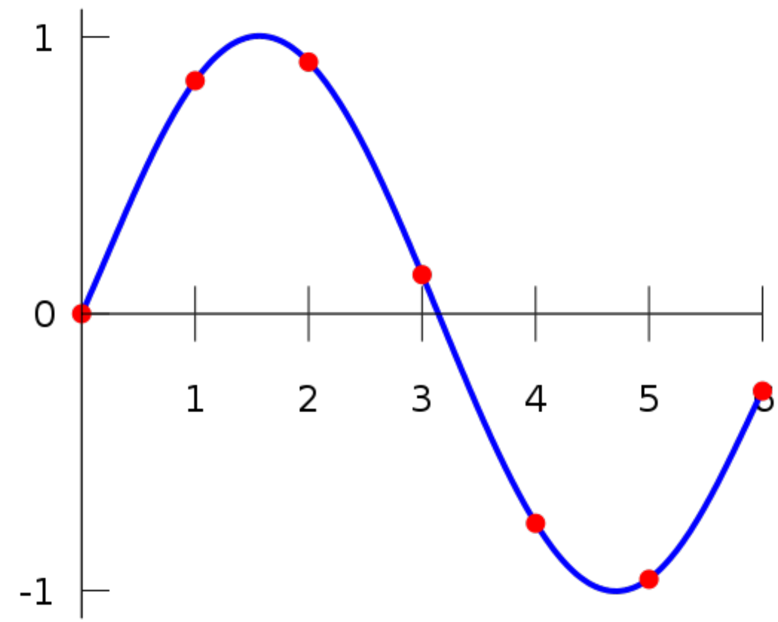
UPPSALA UNIVERSITET

Informationsteknologi

Interpolation, Examples



Linear



Polynomial

https://en.wikipedia.org/wiki/Interpolation

UPPSALA UNIVERSITET

Informationsteknologi

Linear Interpolation

- Linear Interpolation:

$$\mathrm{I} = I_0(1-u) + I_1 u$$

- *u* varies from 0 to 1.
- Expand

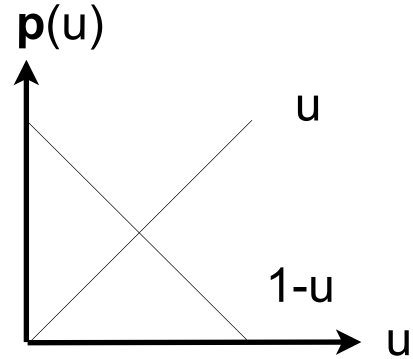
$$\mathrm{I} = I_0 - uI_0 + I_1 u$$

The line equation!!!

$$y = kx + m$$

$$\mathrm{I} = (I_1 - I_0)u + I_0$$

- Basis (blending) Functions



Practical 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

Note:
Cubic Polynomials are often used in Computer graphics

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 space

Consider a simple 1D function $f : \mathbf{R} \to \mathbf{R}$

1. Sample the 1D axis at some points p_i
2. Define cells $c_i=(p_i, p_{i+1})$
3. Consider the reference basis functions for a reference cell $(0,1)$

   $\phi_{0,1} : [0,1] \to [0,1],\ \phi_0(r)=1-r,\ \phi_1(r)=r$
4. Define a linear transformation T_i from the reference to actual cell c_i

$$(x, y, z) = T(r, s, t) = \sum_{i=1}^{n} p_i \Phi_i^1(r, s, t)$$

5. For c_i, define the actual basis functions $\Phi_{0,1}$ using $\phi_{0,1}$ and T_i^{-1} and rewrite the final interpolation

This is covered in the CG course!

$$\tilde{f}(x, y) = \sum_{i=1} f_i \Phi_i^1(T^{-1}(x, y))$$

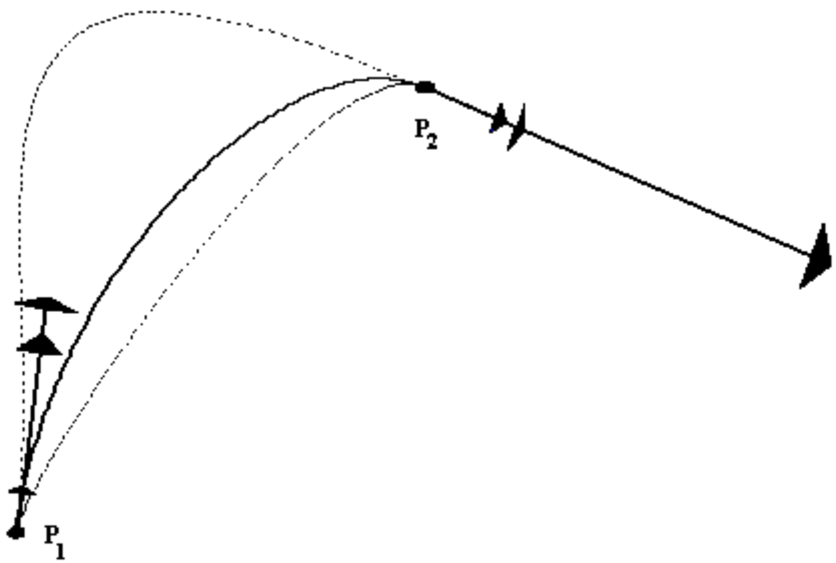
- Apply (5) to interpolate all points in c_i using only samples at vertices p_i, p_{i+1} of c_i
- Repeat from 4 for next cell c_{i+1}

Note: see Sec. 3.4 for expressions for all T^{-1}

UPPSALA
UNIVERSITET

Hermite Splines

- A cubic Hermite curve is defined by four constraints, the two endpoints **p1**, **p2** and the tangents at these points **t1** and **t2**.



UPPSALA UNIVERSITET

Informationsteknologi

How to Derive the Equations

- We can use the following equations:

$$\mathbf{P}(u) = \mathbf{a}u^3 + \mathbf{b}u^2 + \mathbf{c}u + \mathbf{d}$$

$$\mathbf{P}'(u) = 3\mathbf{a}u^2 + 2\mathbf{b}u + \mathbf{c}$$

- Let u=0 and u=1 and solve:

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{t}_1 \\ \mathbf{t}_2 \end{bmatrix}$$

Compute the **inverse** of the matrix to get the coefficients!

UPPSALA UNIVERSITET

Informationsteknologi

Basis and Geometry

- Basis matrix and Geometry matrix

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{t}_1 \\ \mathbf{t}_2 \end{bmatrix}$$

UPPSALA UNIVERSITET

Informationsteknologi

Solution

- Insert vector with different degrees of *u*

$$\mathbf{P}(u) = \begin{bmatrix} u^3, u^2, u, 1 \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} u^3, u^2, u, 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{t}_1 \\ \mathbf{t}_2 \end{bmatrix}$$

P(*u*)=uMG

UPPSALA UNIVERSITET

Informationsteknologi

Blending functions

- Blends the geometry together

$$\mathbf{P}(u) = \underbrace{\left[u^3, u^2, u, 1\right]\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\text{Blending functions}}\begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{t}_1 \\ \mathbf{t}_2 \end{bmatrix}$$

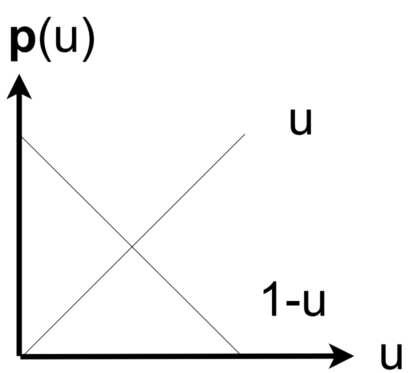
UPPSALA UNIVERSITET

Informationsteknologi

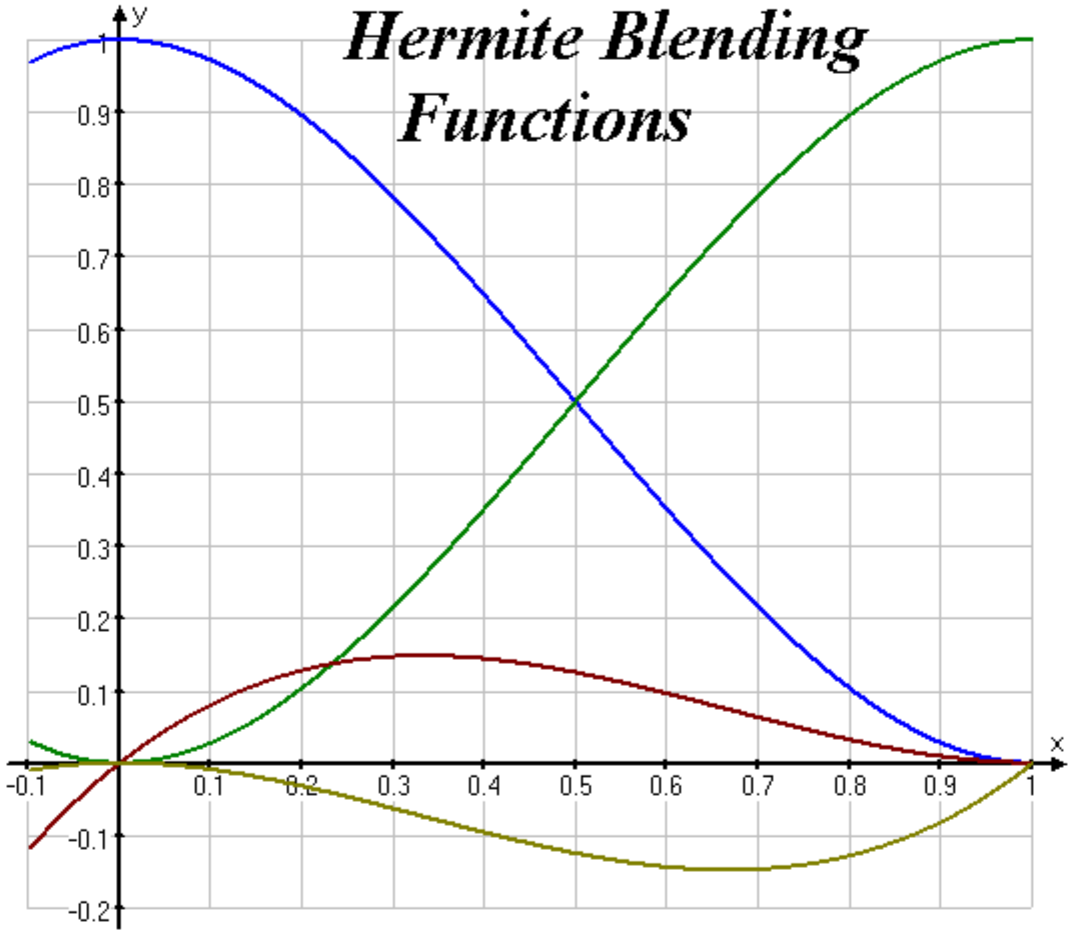
Blending Functions

- The polynomials functions that blends the geometry
  - i.e. blends the control points
- Do you remember linear Interpolation?

$$\mathbf{p}(u)=\mathbf{p}_0(1-u)+\mathbf{p}_1(u)$$



Blending Functions

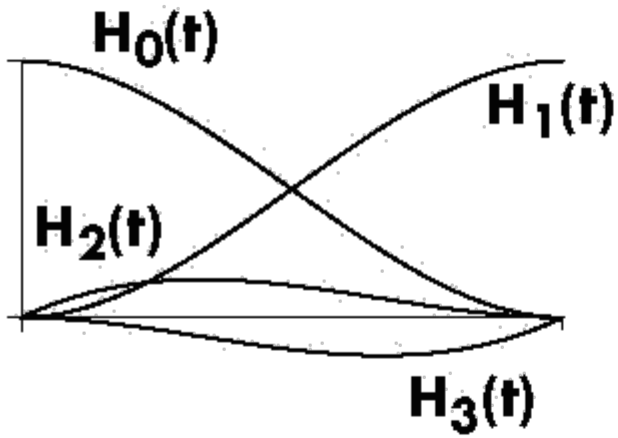


UPPSALA UNIVERSITET

Informationsteknologi

In Detail...

Hermite Basis Functions



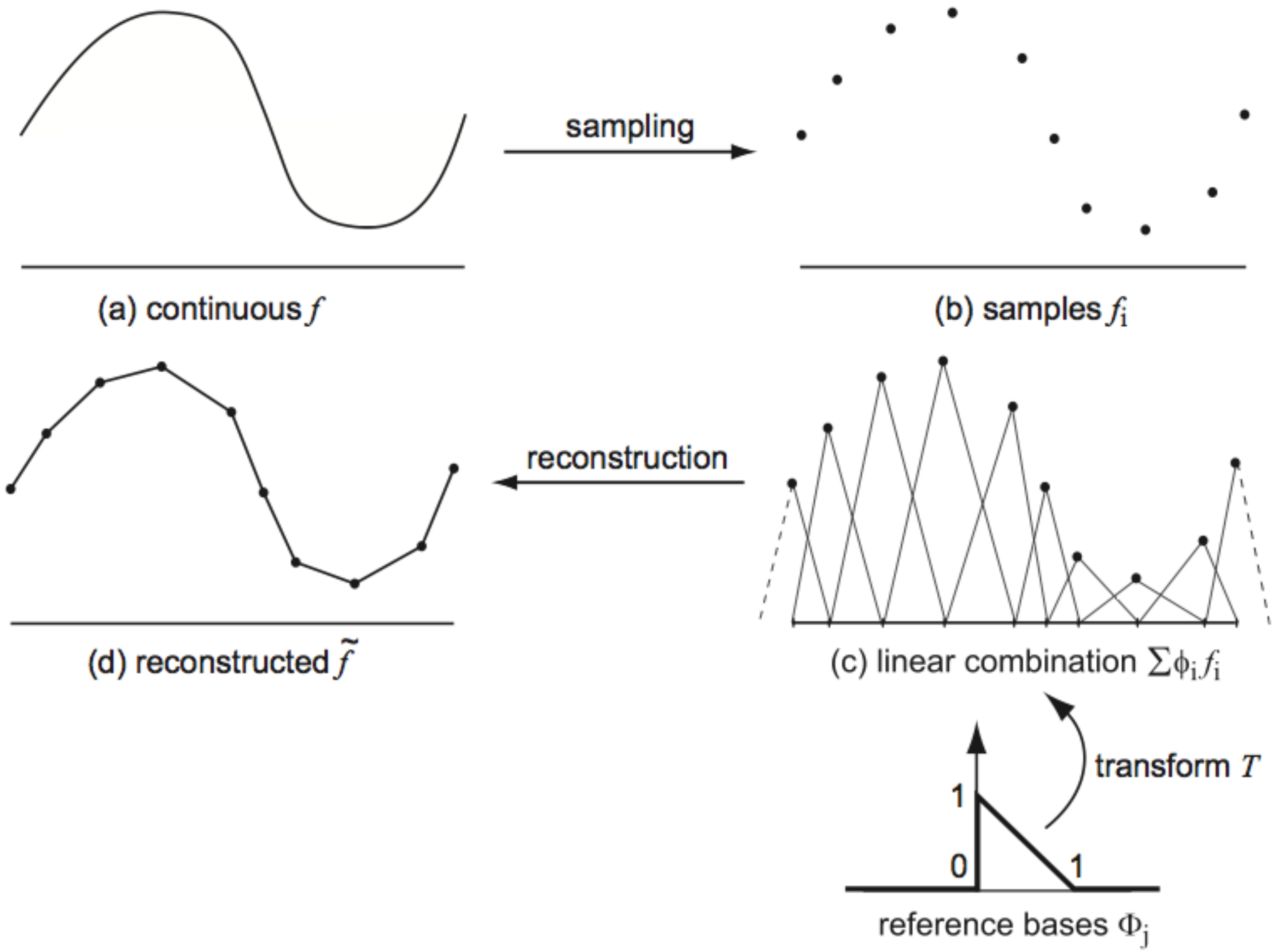
$$H_0(t) = 2t^3 - 3t^2 + 1$$
$$H_1(t) = -2t^3 + 3t^2$$
$$H_2(t) = t^3 - 2t^2 + t$$
$$H_3(t) = t^3 - t^2$$

$$\mathbf{M}_H = \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$

CS248 Fall 98 Lecture 13

t instead of *u* and transposed!

Cells: 1D example (cont'd)



Remarks

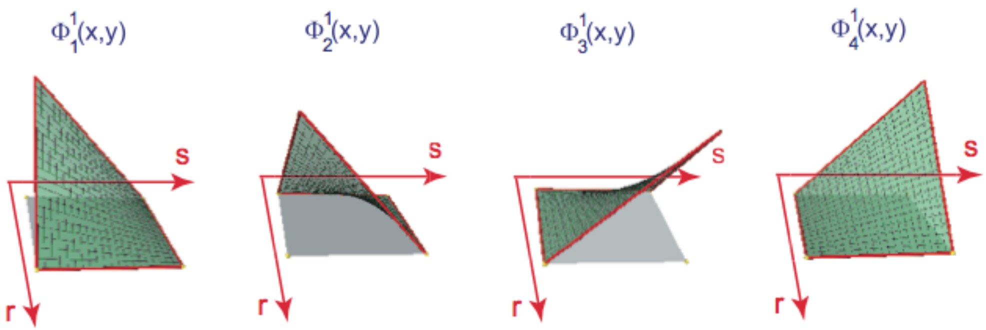
- interpolation & reconstruction goes cell-by-cell
- only need sample points at a cell vertices to interpolate over that cell
- reconstruction is piecewise $\mathcal{C}^1$ because ϕ_i are $\mathcal{C}^1$

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

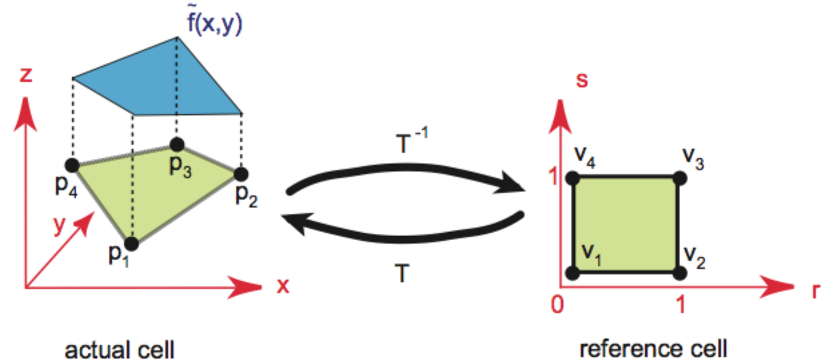
Bilinear basis functions



reference basis functions

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

Bilinear transforms



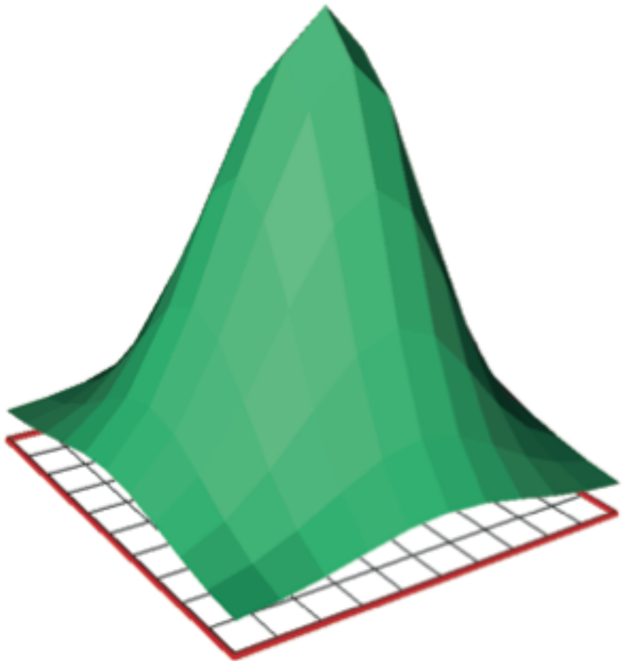
Remember that these functions "support" the points! I.e. when equal to 1 we have the point Vx

$$\sum_{i=1}^{N} f_i \phi_i(p_j) = f_j, \forall j.$$

See book, p 47-50

2D cells: Quads

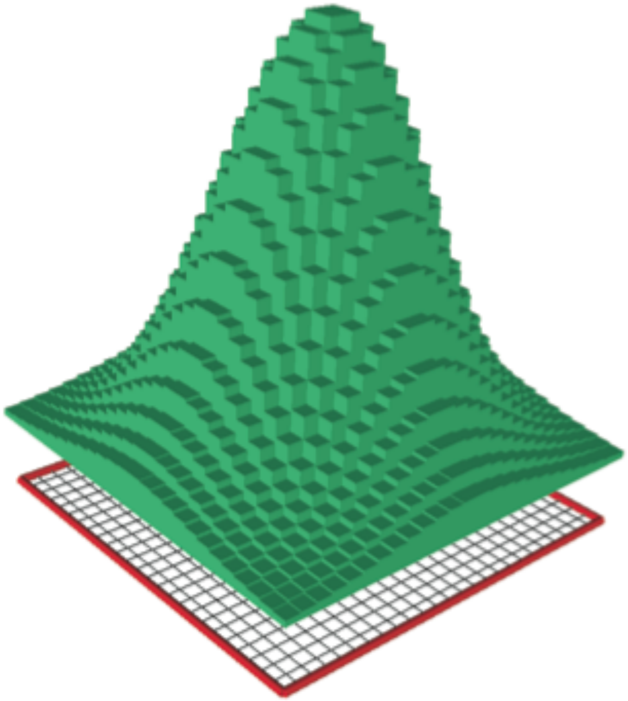
Bilinear interpolation



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

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

Constant interpolation

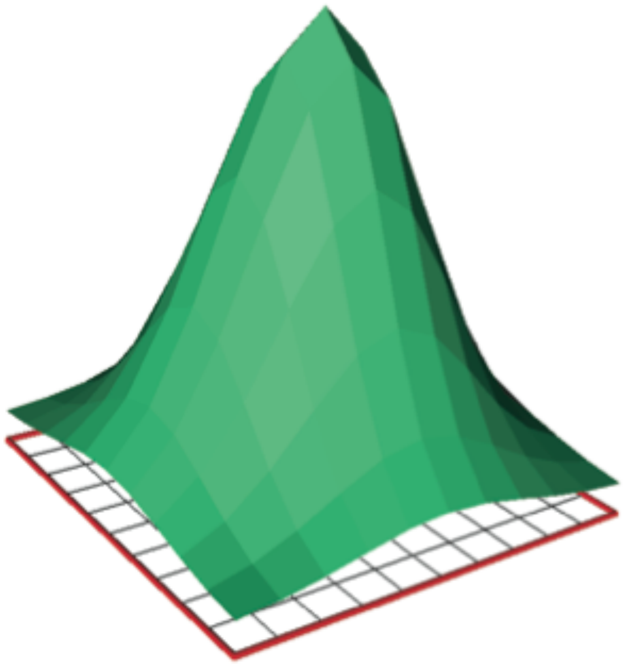


$$\phi_i^0(x) = \begin{cases} 1, & x \in c_i, \\ 0, & x \notin c_i. \end{cases}$$

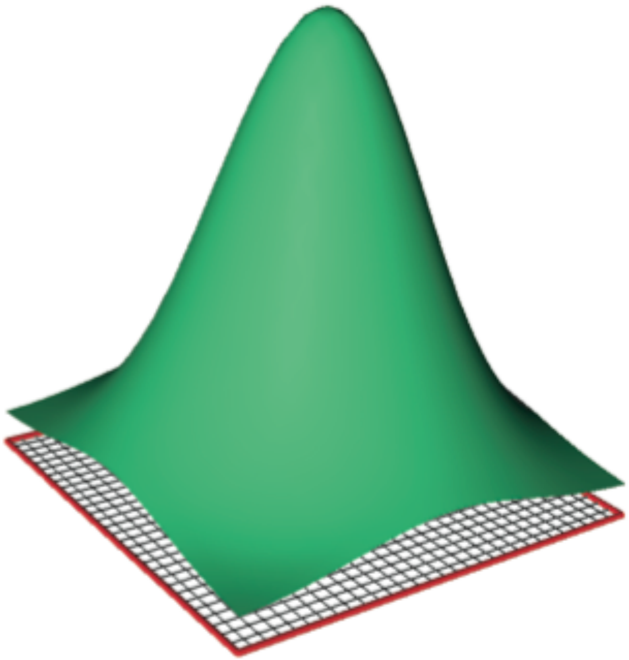
- 1 functions per whole cell
- result: not even $\mathcal{C}^0$
- good for cell-based samples

Intermezzo

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



Flat shading



Gouraud shading

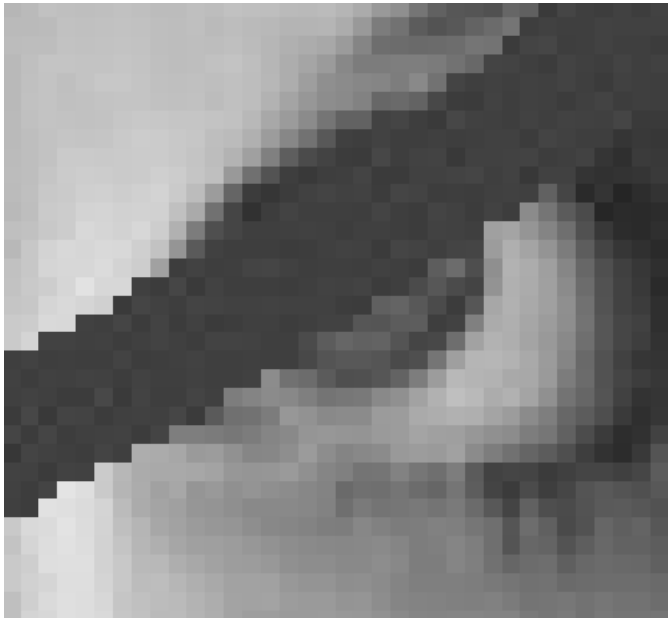
- surface: bilinear interpolation
- colors: constant interpolation

- surface: bilinear interpolation
- colors: bilinear interpolation

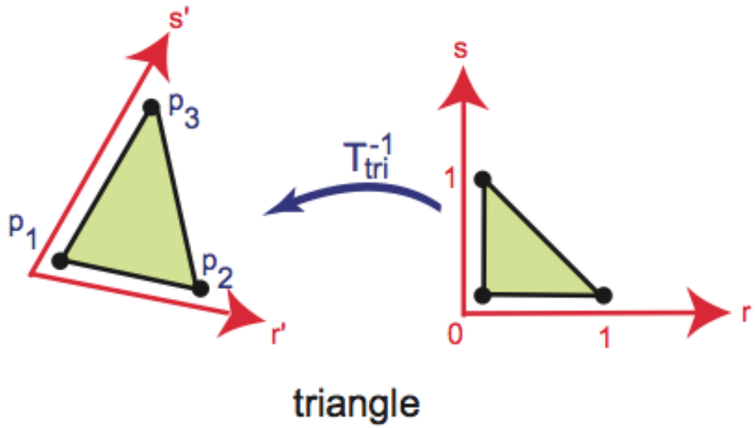
Note: do not confuse *Gouraud shading* (color interpolation) with the *Phong lighting model* (color computation from normals)

2D cells: Quads

Images (color or grayscale)
•use constant basis functions
•cells = pixels
•data (color) is defined at the center of pixels, not corners
•we'll see why this is important in Module 3



2D cells: Triangles



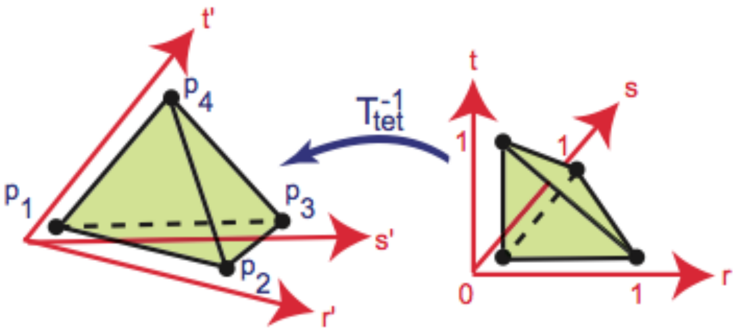
$$\Phi_1^1(r,s) = 1 - r - s,$$
$$\Phi_2^1(r,s) = r,$$
$$\Phi_3^1(r,s) = s.$$

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



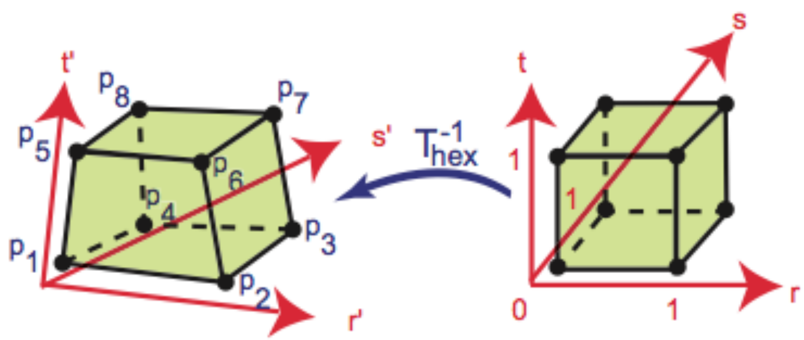
tetrahedron

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

Remarks

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

3D cells: Hexahedra



hexahedron

$$
\begin{aligned}
\Phi_1^1(r,s,t) &= (1-r)(1-s)(1-t),\\
\Phi_2^1(r,s,t) &= r(1-s)(1-t),\\
\Phi_3^1(r,s,t) &= rs(1-t),\\
\Phi_4^1(r,s,t) &= (1-r)s(1-t),\\
\Phi_5^1(r,s,t) &= (1-r)(1-s)t,\\
\Phi_6^1(r,s,t) &= r(1-s)t,\\
\Phi_7^1(r,s,t) &= rst,\\
\Phi_8^1(r,s,t) &= (1-r)st.
\end{aligned}
$$

Remarks

•counterparts of quads in 3D

•interpolate volumetric functions $f : \mathbf{R}^3 \to \mathbf{R}$

•**tri**linear interpolation

•particular case: cubic cells or voxels (studied later in Module 7)

Cell types for constant/linear basis functions

0D

•point

1D

•line

2D

•triangle, quad, rectangle

3D

•tetrahedron, parallelepiped, box, pyramid, prism, …

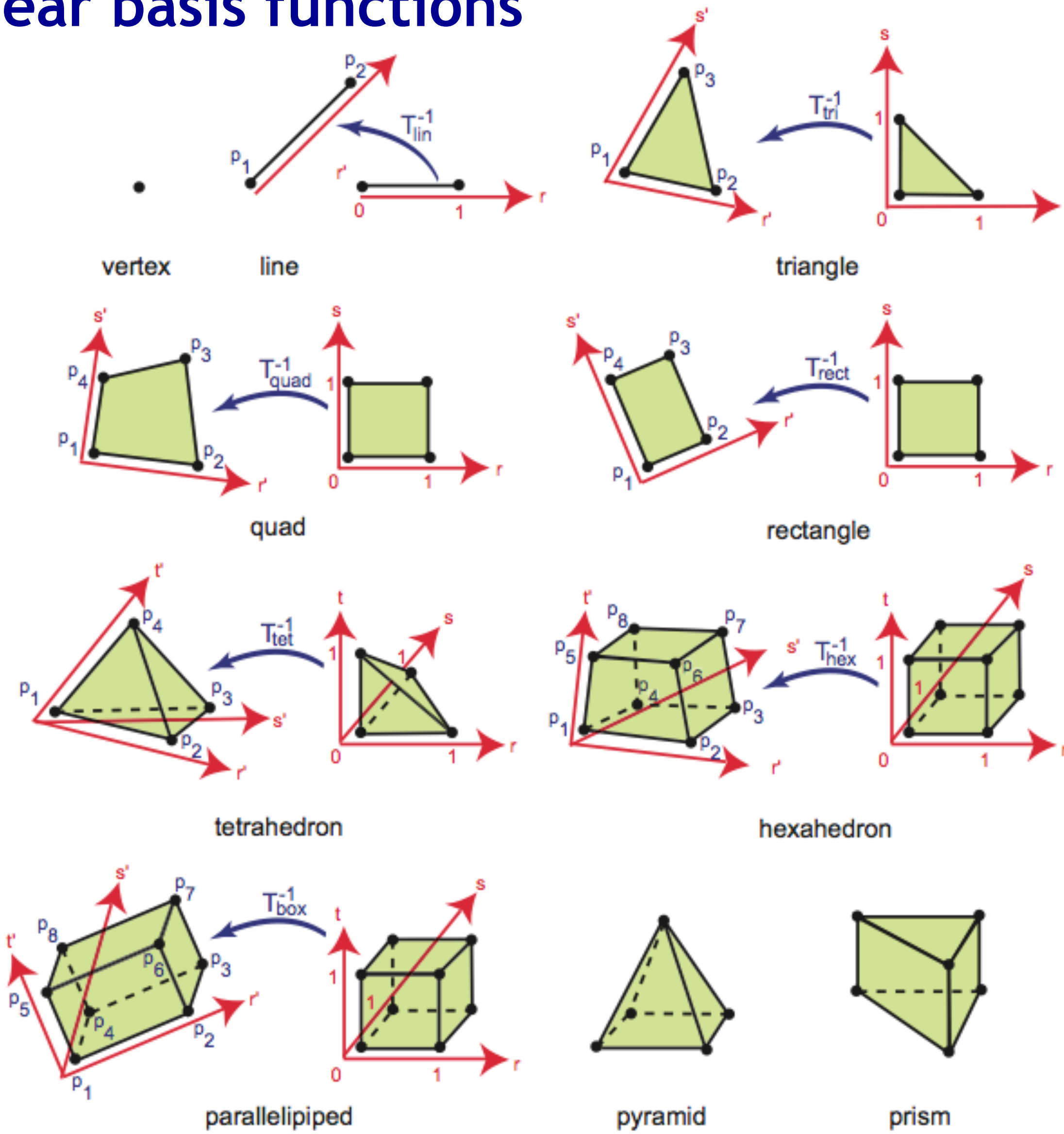


Figure 3.5. Cell types in world and reference coordinate systems.

Quadratic cells

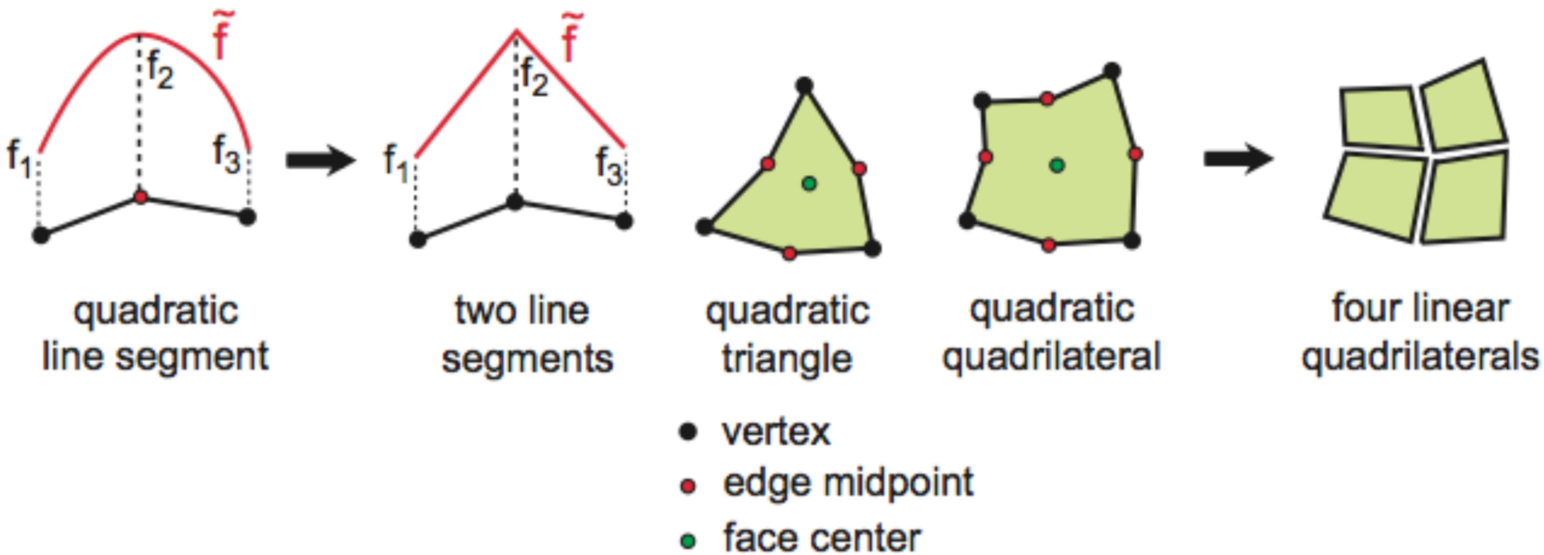


Figure 3.6. Converting quadratic cells to linear cells.

- allow defining quadratic basis functions
- higher precision for interpolation
- however, we need data samples at extra midpoints, not just vertices
- used in more complex numerical simulations (e.g. finite elements)
- split into linear cells for visualization purposes

UPPSALA UNIVERSITET

Informationsteknologi

Non Linear Interpolation

- Splines are often used to interpolate data
- They are polynomials
  - often *second* or *third* degree Polynomials
- Two Points: Linear Interpolation
- Three points: Quadratic Interpolation
- Four points: Cubic Interpolation and so forth
- Larger degree does not necessarily give better interpolation!

UPPSALA UNIVERSITET

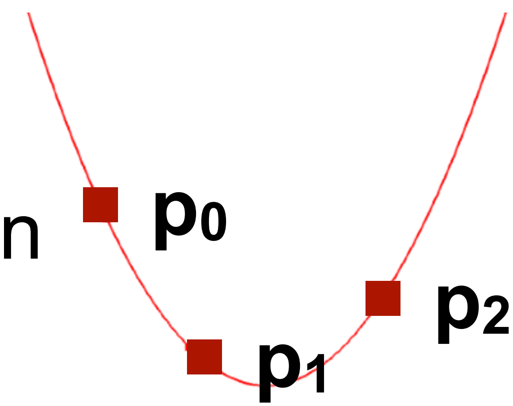
Informationsteknologi

Quadratic interpolation

- $\mathbf{p}(u) = \mathbf{a}u^2 + \mathbf{b}u + \mathbf{c}$
- Solve the system of equations to obtain the coefficients

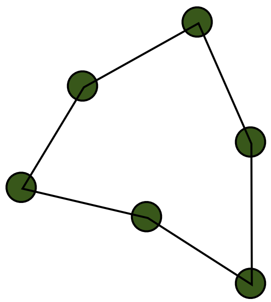
$$\begin{bmatrix} 0 & 0 & 1 \\ 1/4 & 1/2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \end{bmatrix} = \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \end{bmatrix}$$

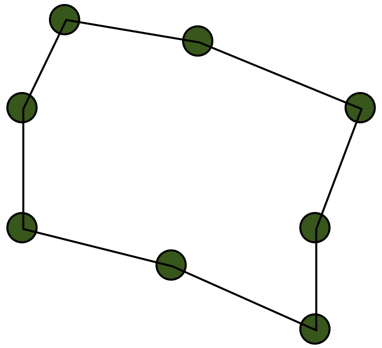
- This curve is defined between $\mathbf{p}_0$ and $\mathbf{p}_1$ for u=[0..1]



Quadratic Interpolation

- Can be used on triangles
  - Must have 6 points of data
- Quads
  - needs 8 points of data





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 (why? Think about interpolation)

$\bigcup_i c_i = D$ the cells cover all our domain (why? Think about our end goal)

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

Uniform grids

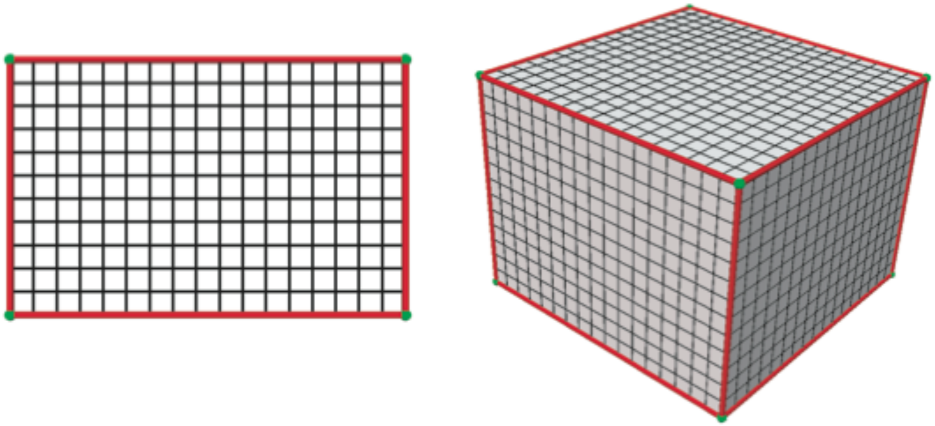
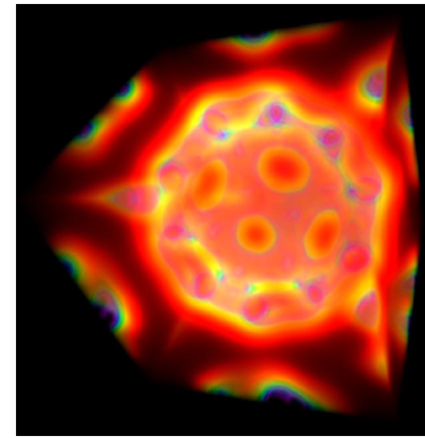


Figure 3.7. Uniform grids. 2D rectangular domain (left) and 3D box domain (right).

image



volume

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

- Efficient Storage!

Rectilinear grids

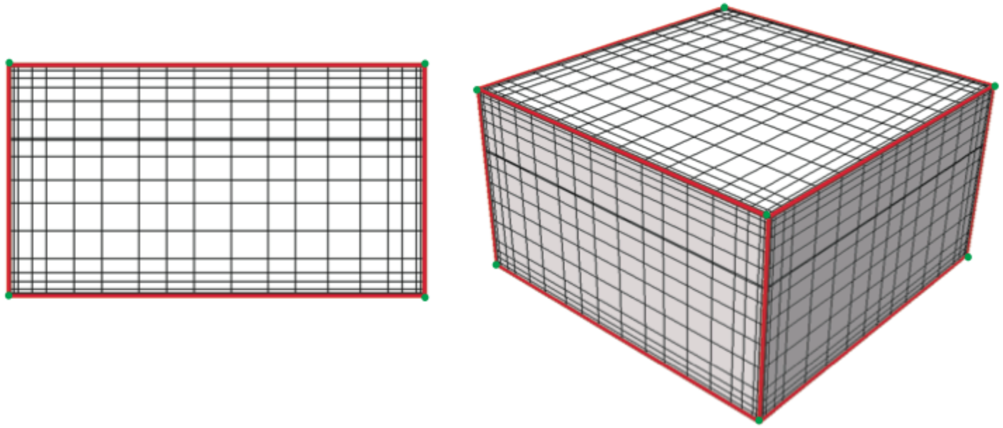


Figure 3.8. Rectilinear grids. 2D rectangular domain (left) and 3D box domain (right).

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

Structured grids

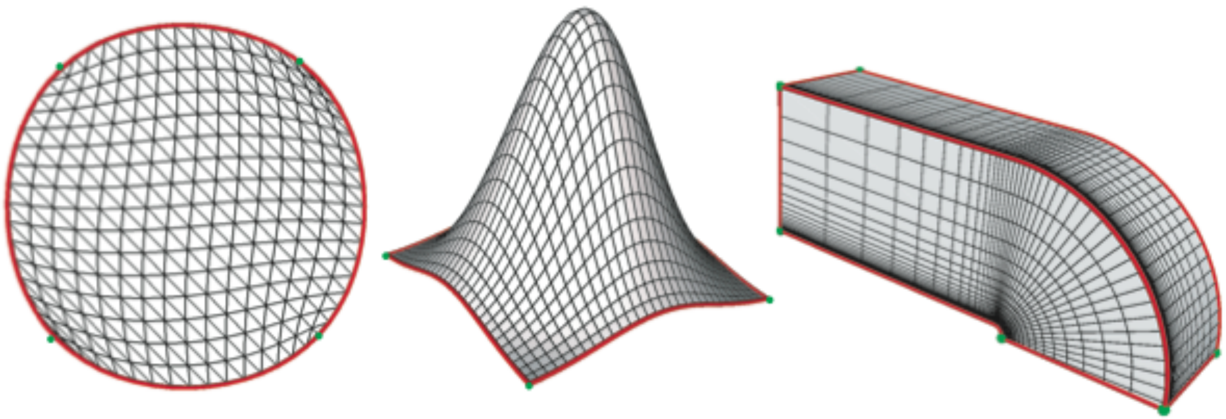
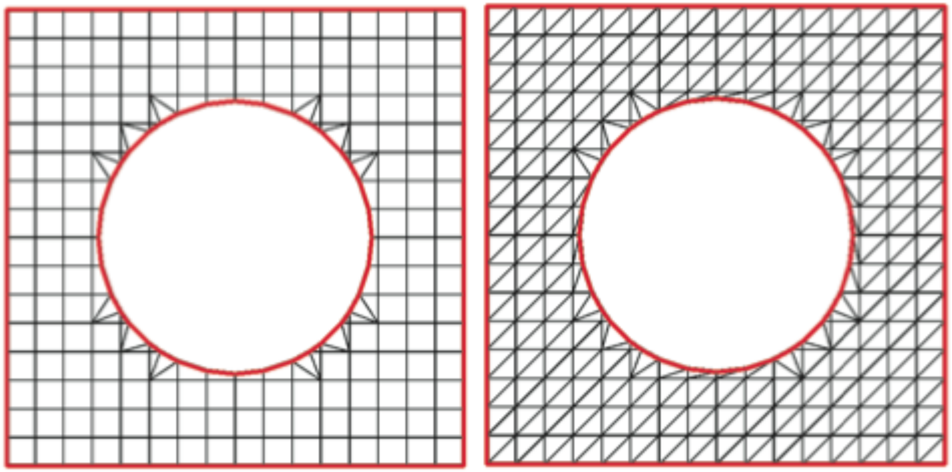


Figure 3.9. Structured grids. Circular domain (left), curved surface (middle), and 3D volume (right). Structured grid edges and corners are drawn in red and green, respectively.

- 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

Unstructured grids

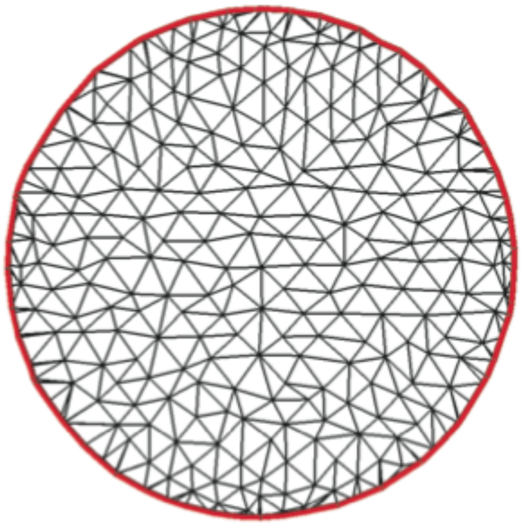
Consider the domain D: a square with a hole in the middle

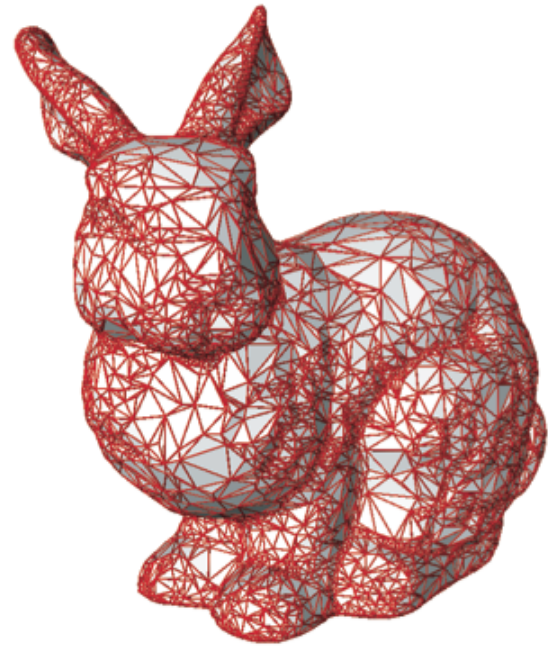


We cannot cover such a domain with a structured grid (why?)
•it's not of genus 0, so cannot be covered with a matrix-like distribution of cells

For this, we need unstructured grids (see next)

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 $V = \{v_i\}$

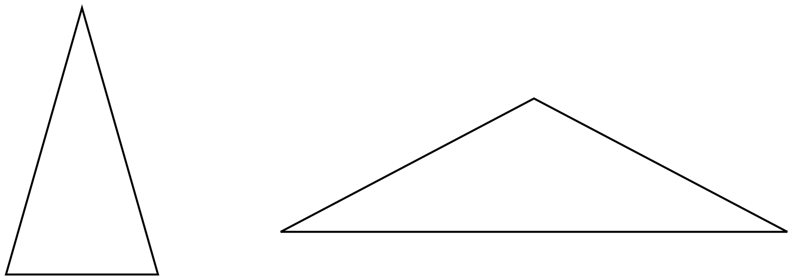
  cell set $C = \{c_i = (\mathit{indices\ of\ vertices\ in\ V})\}$
- most flexible, but most complex/expensive grid type

UPPSALA UNIVERSITET

Informationsteknologi

Topology vs. Geometry

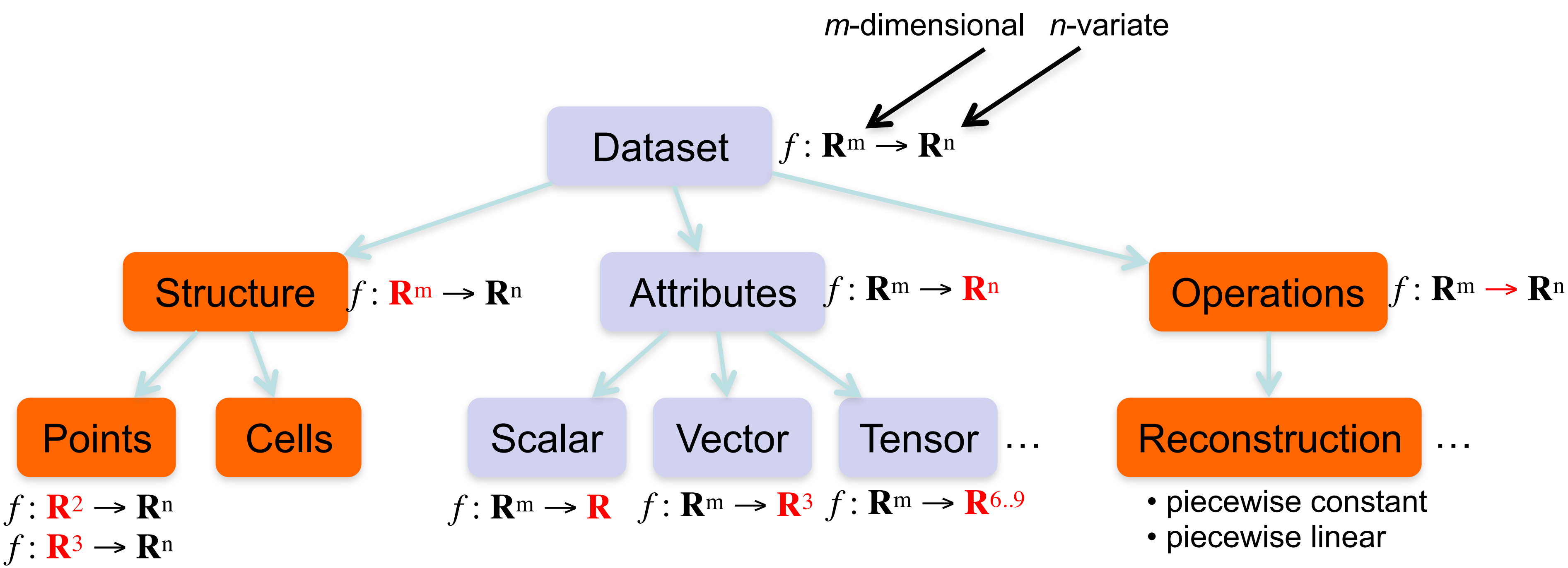
- Each type has its topology
  - Example:
    - A triangle has three vertices but a line has only two, etc
- Geometry
  - Can differ within the same type



Data Representation

- Cells
  - Linear
  - Non linear
- Topology vs. Geometry
- Attribute Data
  - Scalar
  - Vectors
  - Normals
  - Texture Coordinates
  - Tensors

Recapitulation: Dataset



- We discussed about **these** (grids, interpolation, reconstruction)
- We discuss next about **attributes**

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)

Summary

Data Representation (book Chapter 2)

- reconstruct continuous representations of sampled signals
  - efficiently
  - accurately

- interpolation, grids, and cells

- data attributes (scalars, vectors, tensors)

- advanced issues (resampling, grid-less interpolation)

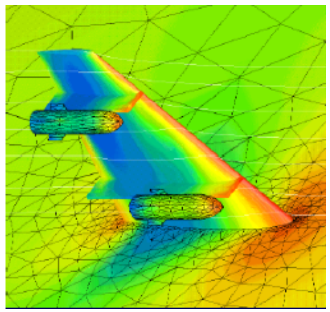
- read Ch. 2 *in detail* to understand all the math!

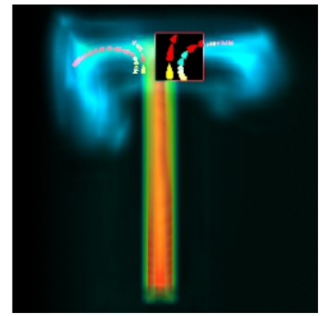
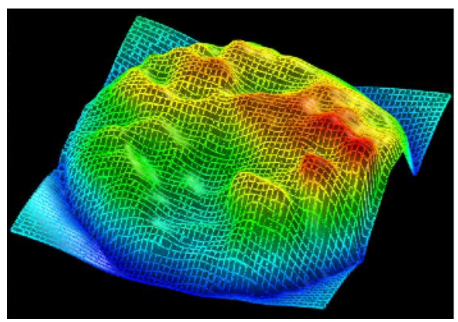
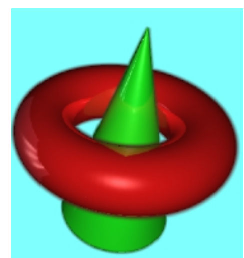
Next module

- visualization algorithms

Happy so far?

Scientific Visualization Module 3
Scalar Algorithms



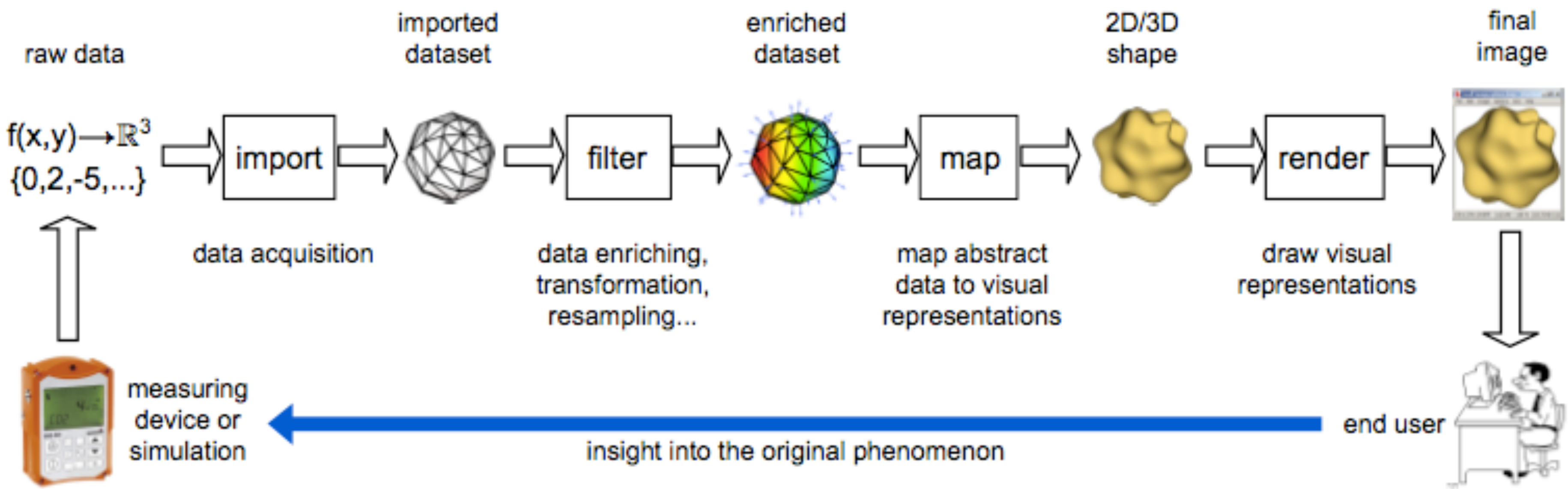




prof. dr. Alexandru (Alex) Telea

Department of Mathematics and Computer Science
University of Groningen, the Netherlands

The Visualization Pipeline - Recall

	data formatting		data filtering		data mapping		3D to 2D rendering	
Dataset →	Process →	Dataset →	Process →	Dataset →	Process →	Dataset →	Process →	Dataset



Algorithm classification

1. **Scalar algorithms**
   - operate on scalar data
   - color mapping, contouring, height plots

2. **Vector algorithms**
   - operate on vector data
   - hedgehogs, glyhps, derived quantities, stream surfaces, image-based methods

3. **Tensor algorithms**
   - operate on symmetric 3x3 tensors
   - tensor glyphs, hyperstreamlines, fiber tracing, principal component analysis

4. **Modeling algorithms**
   - change attributes and/or underlying **grid**
   - implicit functions, distance fields, cutting, selection, grid-less interpolation, grid processing

Color mapping

Basic idea

•Map each scalar value $f \in \mathbf{R}$ at a point to a color via a function $c : [0,1] \rightarrow [0,1]^3$

Color tables

•precompute (sample) c and save results into a table $\{c_i\}_{i=1..N}$

•index table by normalized scalar values

input data: scalar value f

↓

determine input range: scalar value range $[f_{\min}, f_{\max}]$

↓

normalize input to [0,N]: $i = N \dfrac{f - f_{\min}}{f_{\max} - f_{\min}}$

↓

Color mapping ← $c_i = c\left(\dfrac{i}{N}\right)$ (color table) ← $c : [0,1] \rightarrow [0,1]^3$ (desired color transfer function)

↓

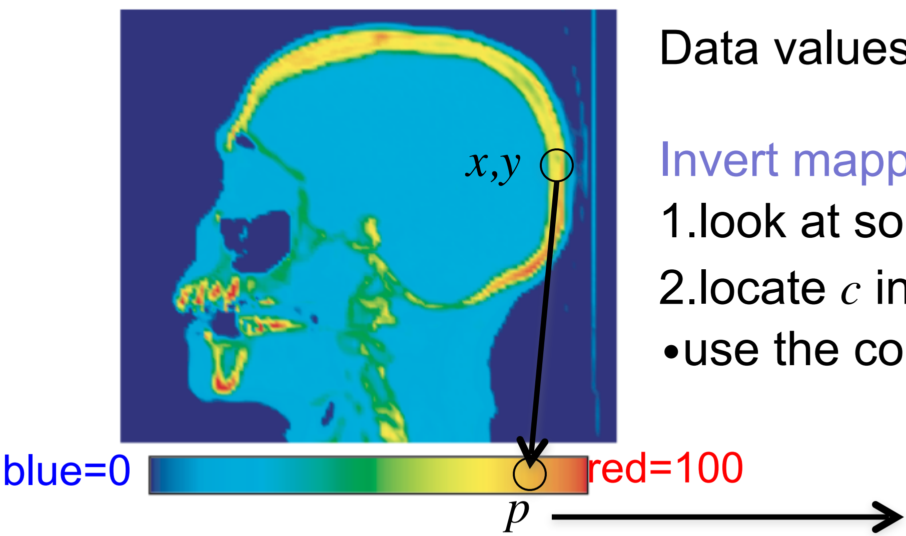
f is color-mapped to c_i

Colormap design

What makes a good colormap?

- map scalar values to colors *intuitively*…
- …so we can visually *invert* the mapping to tell scalar values from colors

Recall example in Module 1



Data values mapped to RGB colors via a colormap

Invert mapping:

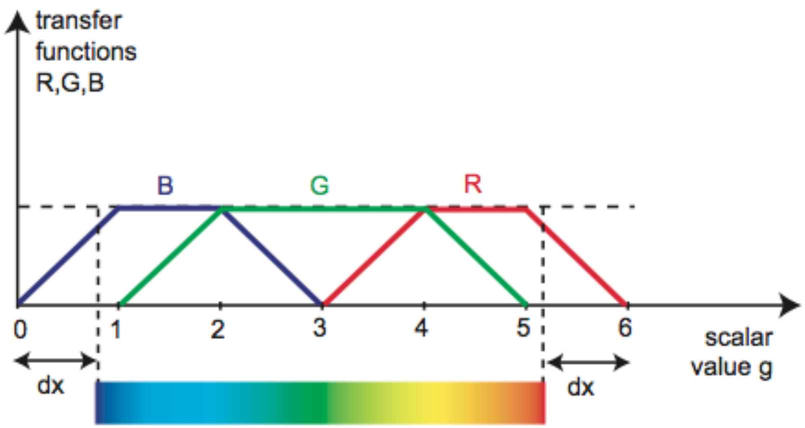
1. look at some point (x,y) in the image $\rightarrow$ color c
2. locate c in colormap at some position p

- use the colormap legend to derive data value s from p

answer: $s = 90$

Rainbow colormap

- probably the most (in)famous in data visualization
- intuitive 'heat map' meaning
    - cold colors = low values
    - warm colors = high values



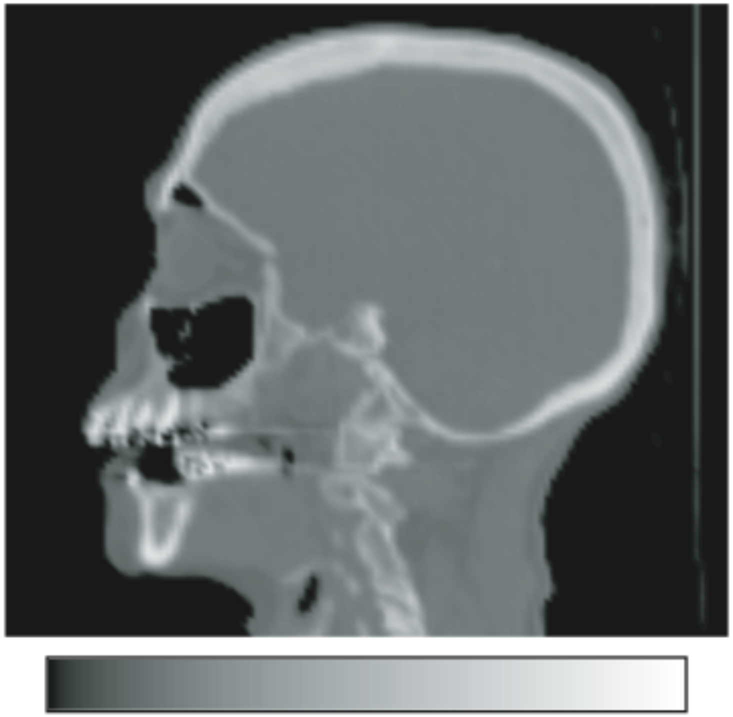

```
void c(float f, float& R, float& G, float& B)
{
    const float dx = 0.8;
    f = (f<0)? 0 : (f>1)? 1 : f;          //clamp f in [0,1]
    g = (6-2*dx)*f + dx;                   //scale f to [dx, 6 - dx]
    R = max(0,(3-fabs(g-4)-fabs(g-5))/2);
    G = max(0,(4-fabs(g-2)-fabs(g-4))/2);
    B = max(0,(3-fabs(g-1)-fabs(g-2))/2);
}
```

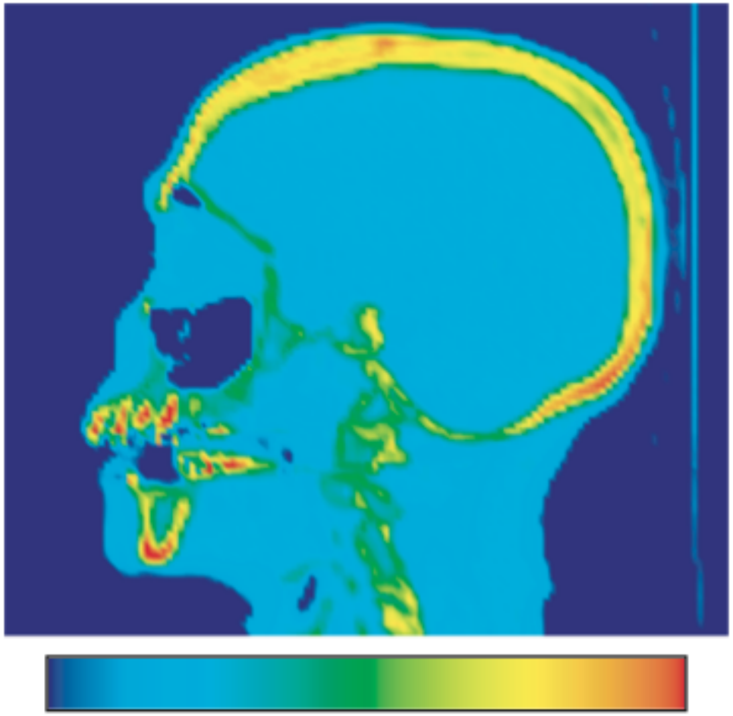
Simple to implement
(see Sec. 5.2)

Gray-value colormap

•brightness = value
•natural in some domains (X-ray, angiography)

2D slice in 3D CT dataset
Scalar value: tissue density



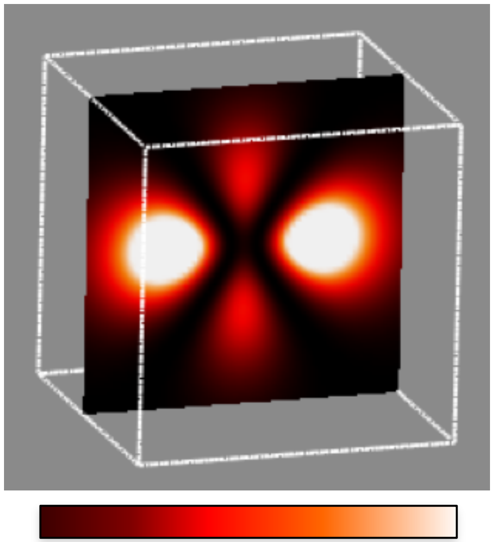


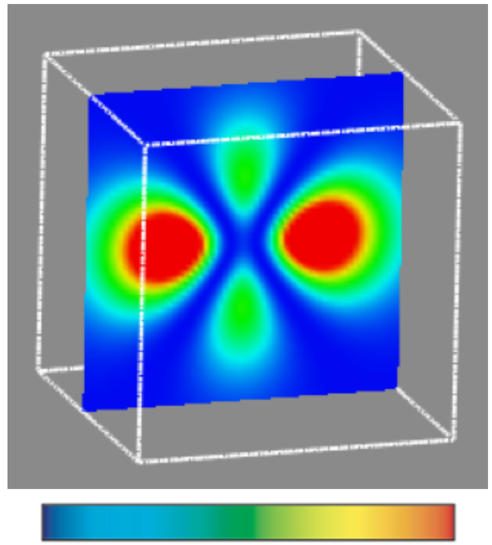
Gray-value colormap
•white = hard tissues (bone)
•gray = soft tissues (flesh)
•black = air

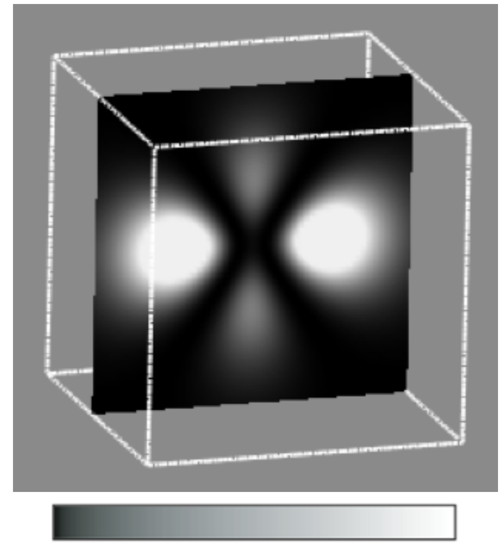
Rainbow colormap
•red = hard tissues (bone)
•blue = air
•other colors = soft tissues

Colormap comparison

2D slice in 3D hydrogen atom potential field







Heat colormap
•maxima highlighted well
•lower values better separable than with gray-value colormap

Rainbow colormap
•maxima not prominent
•lower values better
•separable

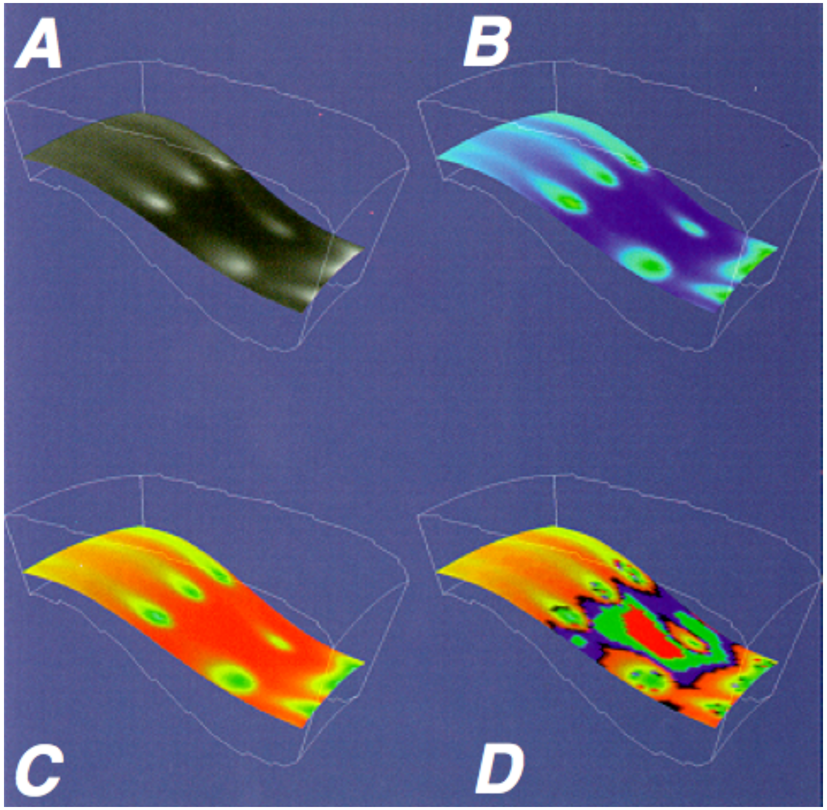
Gray-value colormap
•maxima are highlighted well
•lower values are unclear

Which is the better colormap? Depends on the application context!

Colormap comparison

2D slice in 3D pressure field in an engine

A. Gray-value colormap
•maxima highlighted well
•low-contrast



B. Purple-to-green colormap
•maxima highlighted well
•good high-low separation

C. Red-to-green colormap
•luminance not used
•color-blind problems..

D. 'Random'
•equal-value zones visible
•little use for the rest

Which is the better colormap? Depends on the application context!

Colormap design techniques

We cannot give universal design rules

•but some technical guidelines/tricks still exist

1. Fully use the perceptual spectrum

•colormap entries should differ in more, rather than less, HSV components

scalar value ~ V; H,S not used

scalar value ~ H; S,V not used

scalar value ~ H,V; S not used

2. Colormap should be easily invertible

•avoid colormap entries with

- similar HSV entries
- which are *perceived* as similar (see color blindness issues)
- which are hard to perceive (e.g. dark or strongly desaturated colors)

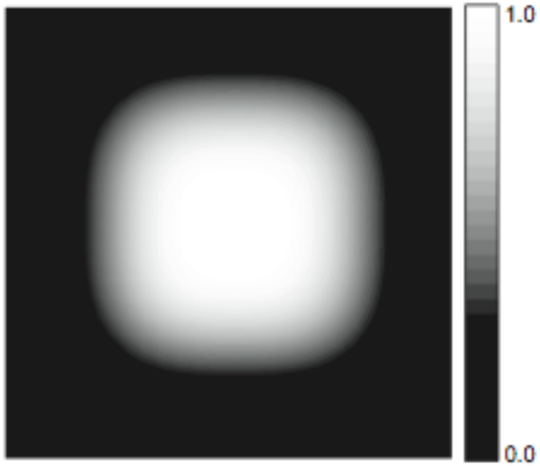
Good design guidelines: www.colorbrewer.org

Colormap design techniques

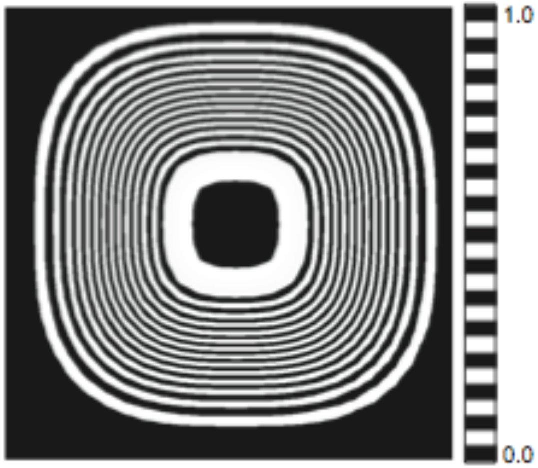
3. Design based on what you *need* to emphasize

- specific value ranges
- specific values
- value change rate (1st derivative of scalar data)
- …

2D function $f(x,y) = e^{-10(x^4+y^4)}$



Gray-scale colormap
- highlights plateaus
- value transitions hard to see



Zebra colormap
- highlights value variations (1st derivative)
- dense, thin bands: fast variation
- thick bands: slow variation

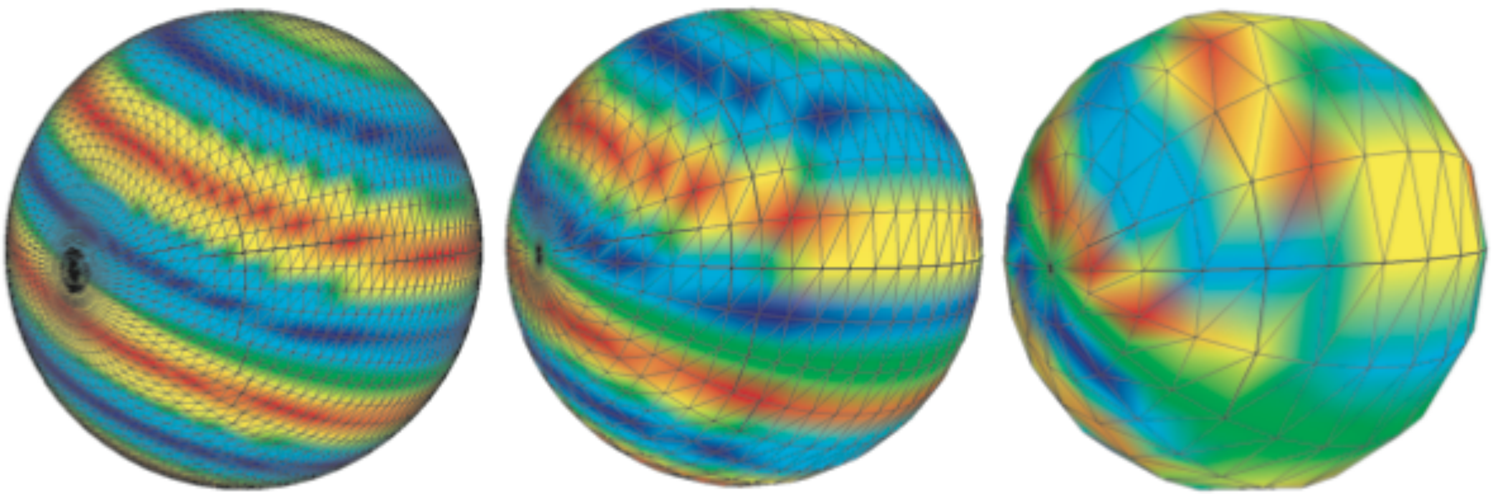
Colormap implementation details

Where to apply the colormap?

•per grid-cell vertex

Note:
Compare to
Gouraud Shading

2D periodic high-frequency function



64x64 points 32x32 points 16x16 points

As we decrease the sampling frequency, strong colormapping artifacts appear

Why is this so?

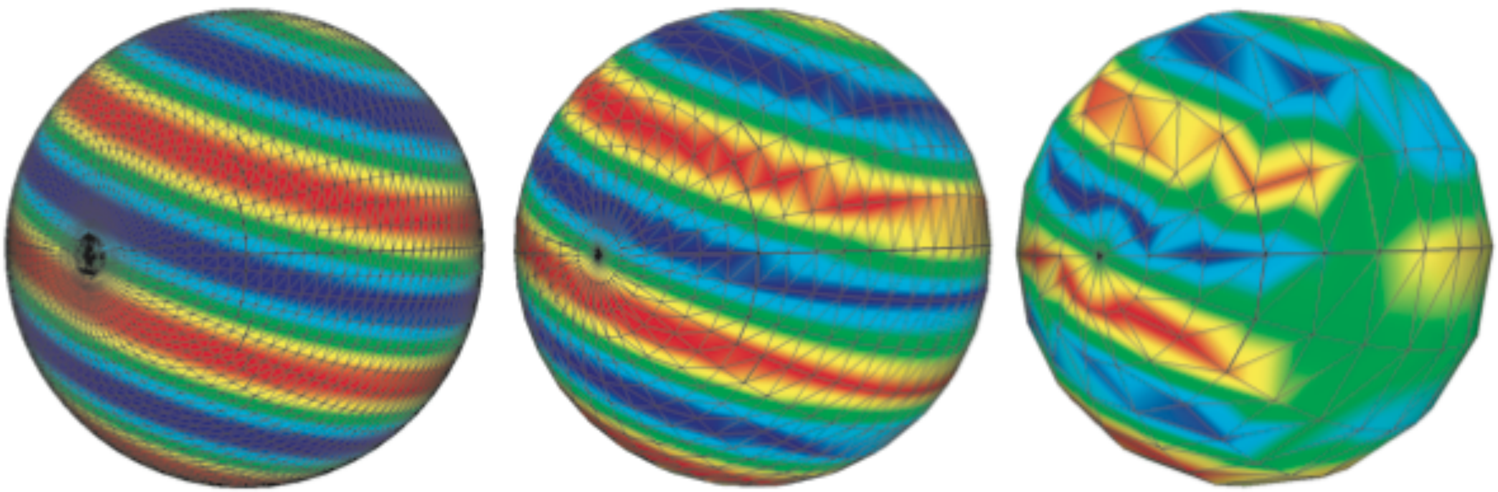
Colormap implementation details

Note:
Compare to
Phong Shading

Where to apply the colormap?

•per pixel drawn – better results than per-vertex colormapping
•done using 1D textures

2D periodic high-frequency function



64x64 points 32x32 points 16x16 points

Explanation

•per-vertex: $f \rightarrow c(f) \rightarrow$ interpolation($c(f)$) color interpolation can fall outside colormap!

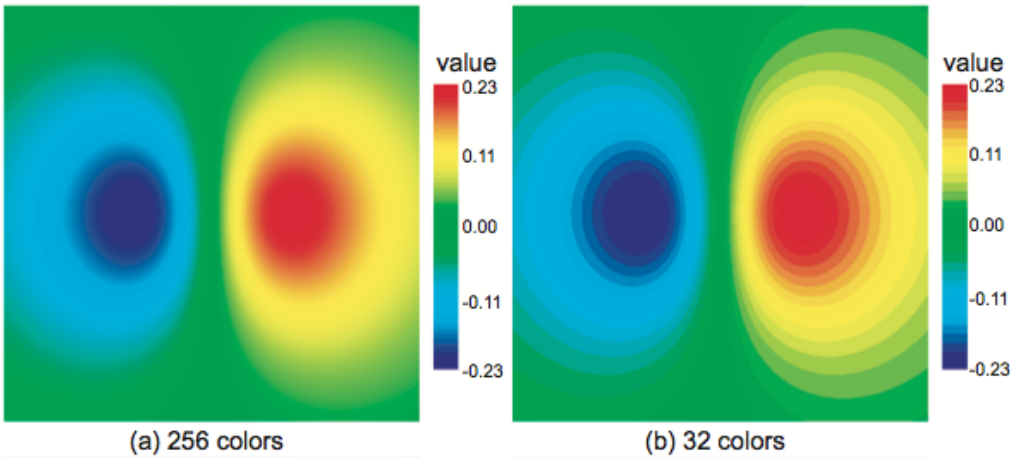
•per-pixel: $f \rightarrow$ interpolation(f)$\rightarrow c$(interpolation(f)) colors always stay in colormap

See Sec. 5.2 for details

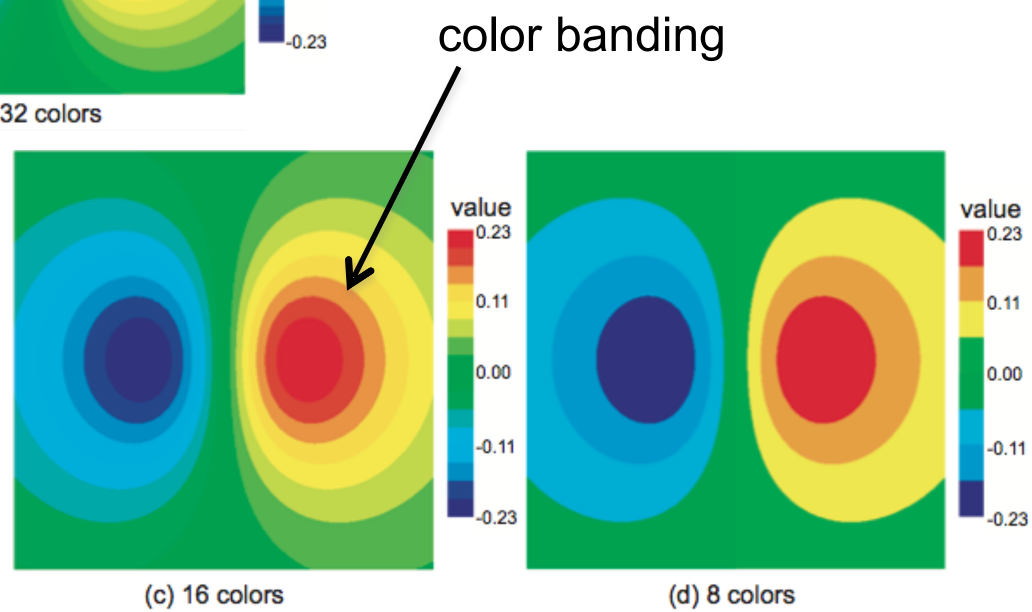
Color banding

How many distinct colors N to use in a color table?

•more colors: better sampled c thus smoother results
•fewer colors: color banding appears



(a) 256 colors (b) 32 colors

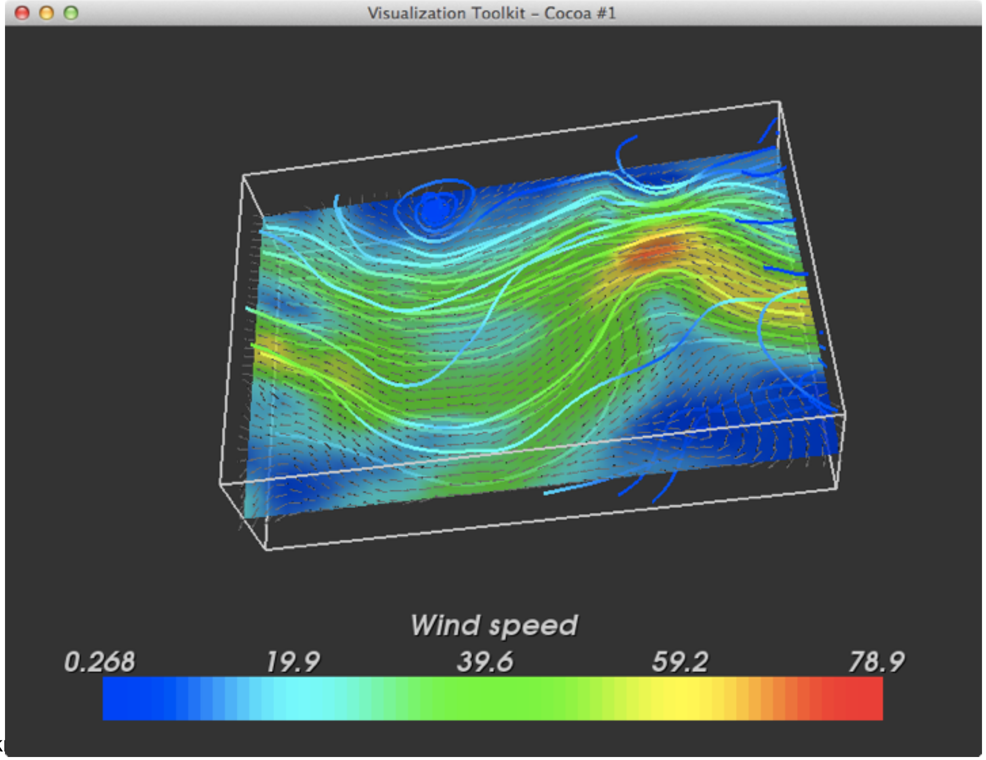


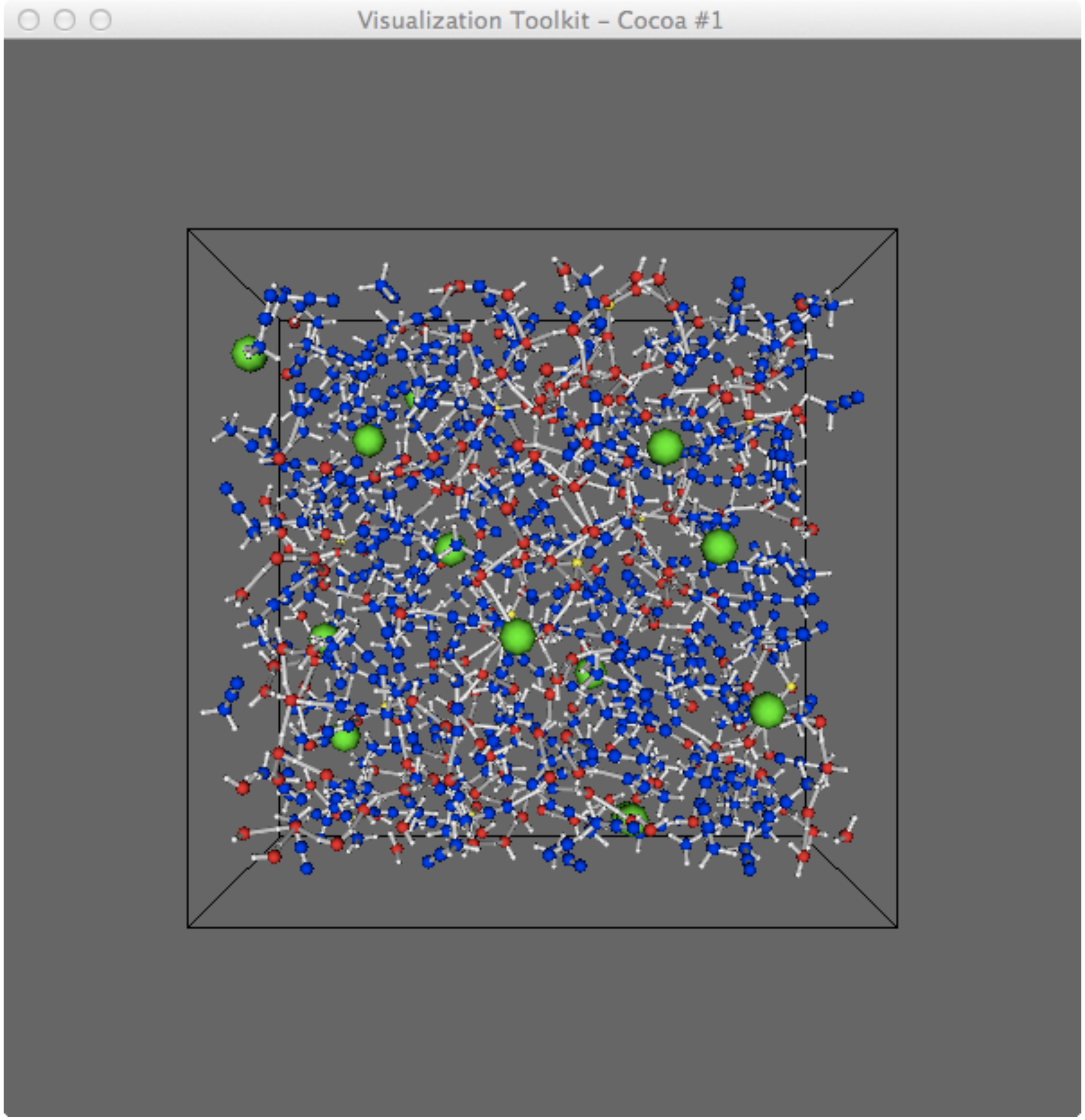
(c) 16 colors (d) 8 colors

Question
•can we see sharp color banding with per-vertex colormapping? Why (not)?

Colour Mapping

- Maps scalar data to colour
- Can be done by a colour look up table



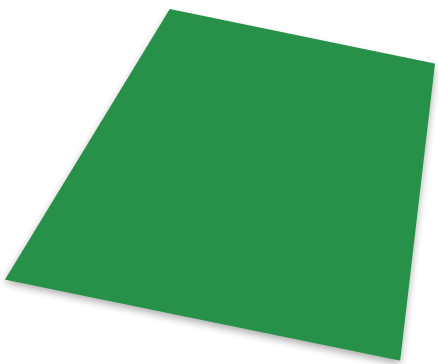

Visualization Toolkit – Cocoa #1

Height / displacement plots

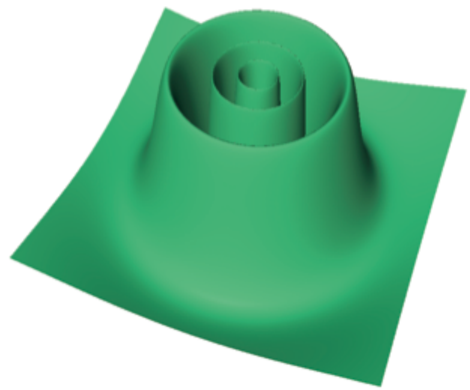
Displace a given surface $S \subseteq D$ in the direction of its normal

Displacement value encodes the scalar data f

$$S_{displ}(x) = x + n(x)\, f(x), \;\; \forall x \in S$$



input surface S

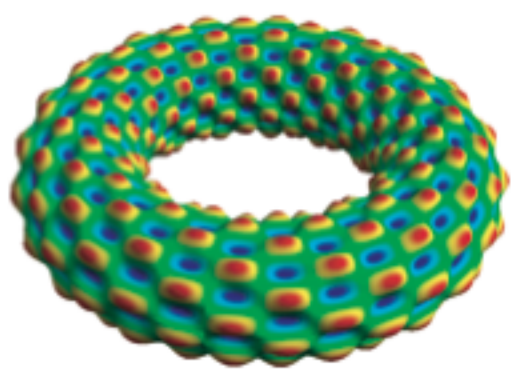


displaced surface S_{displ}

Height plot

- $S = xy$ plane
- displacement always along z

input surface S



displaced surface S_{displ}

Displacement plot

- S = any surface in $\mathbf{R}^3$
- useful to visualize 3D scalar fields

UPPSALA UNIVERSITET

Informationsteknologi

Conclusion

- Data interpolation
  - Fills in “missing data”
- Simple visualisation of scalar data
  - Colour Mapping
- Next time scalar continued and vector visualisations