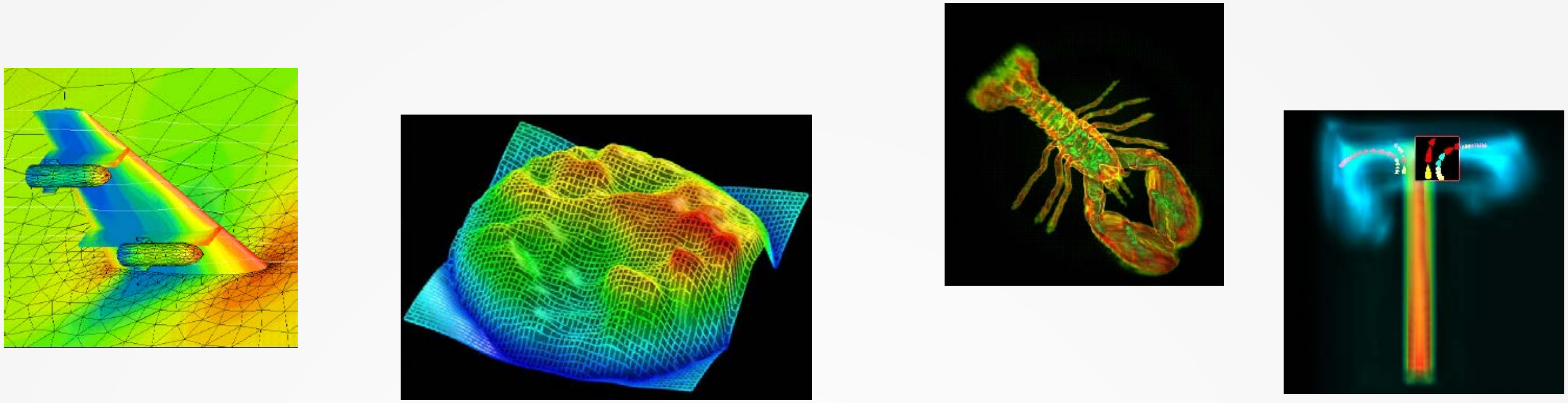


Scalar Algorithms

Cmpt 767 Visualization

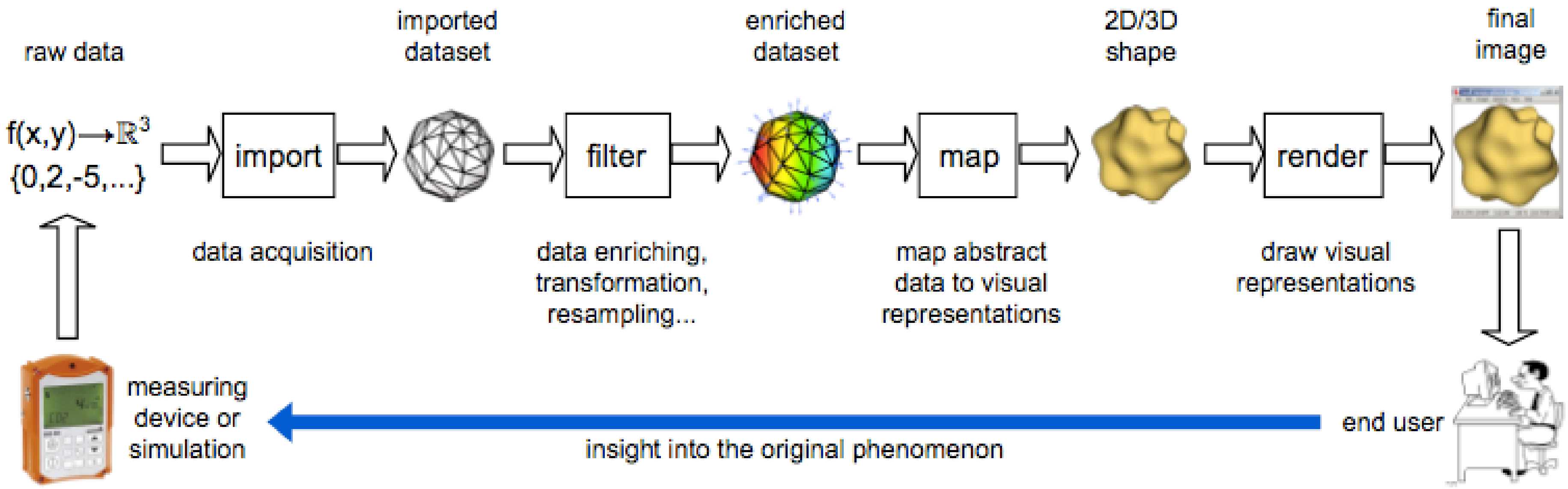
Steven Bergner
sbergner@sfu.ca

[based on slides by A. C. Telea]

The Visualization Pipeline - Recall

data formatting, data filtering, data mapping, 3D to 2D rendering

Dataset ➡ Process ➡ Dataset ➡ Process ➡ Dataset ➡ Process ➡ Dataset ➡ Process ➡ Dataset



[A.C Telea: Data Visualization, Principles and Practice, 2nd edition, CRC Press, 2014]

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

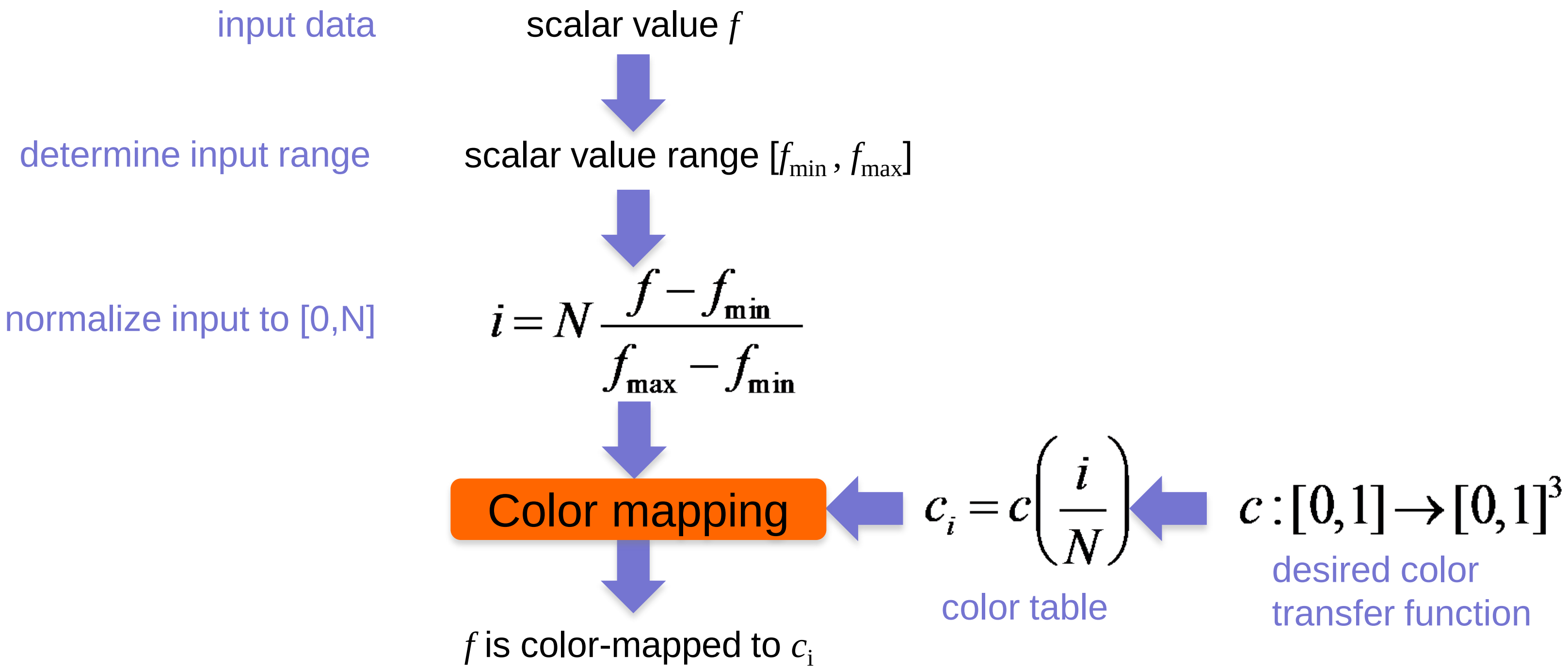
Color mapping

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] — $$i = N \frac{f - f_{\min}}{f_{\max} - f_{\min}}$$

Color mapping

$$c_i = c\left(\frac{i}{N}\right)$$

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?

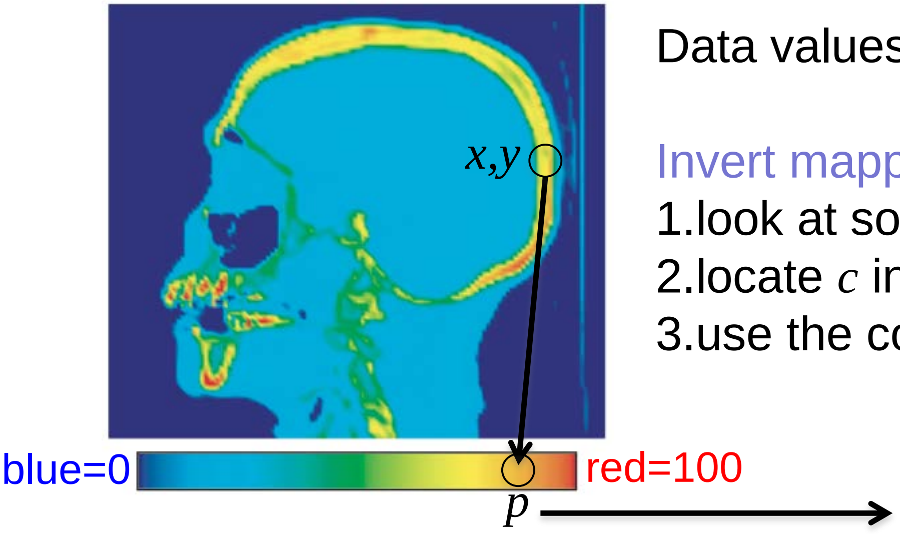
- map scalar values to colors *intuitively*…
- …so we can visually *invert* the mapping to tell scalar values from colors

Recall example in Module 1

Data values mapped to RGB colors via a colormap

Invert mapping:

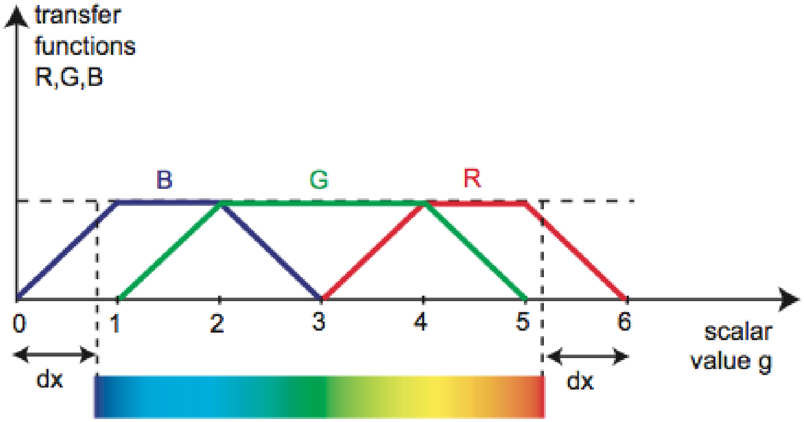
1. look at some point (x,y) in the image → color c
2. locate c in colormap at some position p
3. use the colormap legend to derive data value s from p

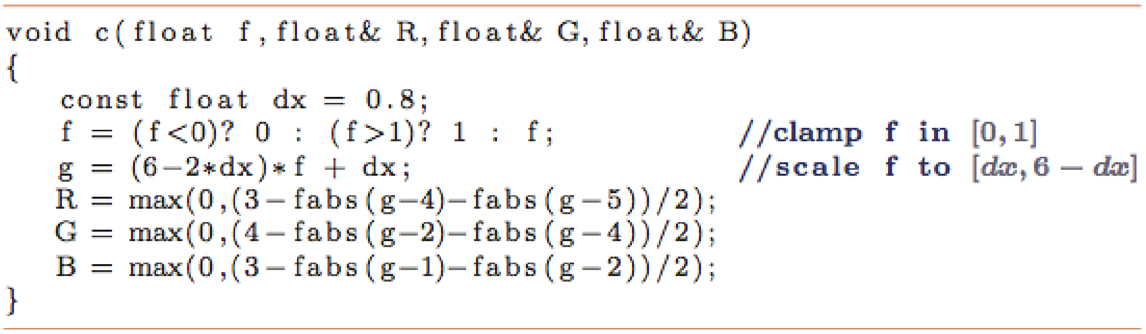


answer: $s = 90$

Rainbow colormap

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





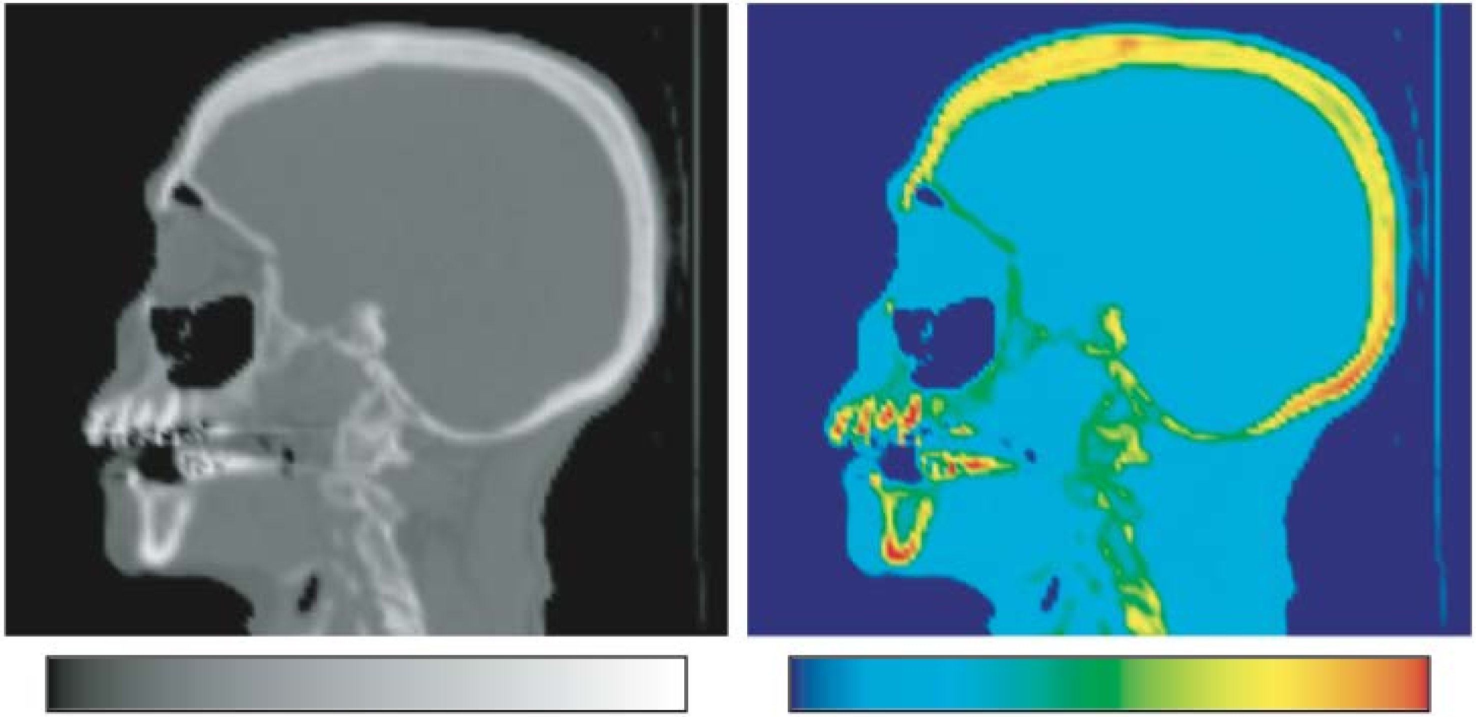
```
void c(float f,float& R,float& G,float& B)
{
   const float dx = 0.8;
   f = (f<0)? 0 : (f>1)? 1 : f;          //clamp f in [0,1]
   g = (6-2*dx)*f + dx;                  //scale f to [dx,6-dx]
   R = max(0,(3-fabs(g-4)-fabs(g-5))/2);
   G = max(0,(4-fabs(g-2)-fabs(g-4))/2);
   B = max(0,(3-fabs(g-1)-fabs(g-2))/2);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

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

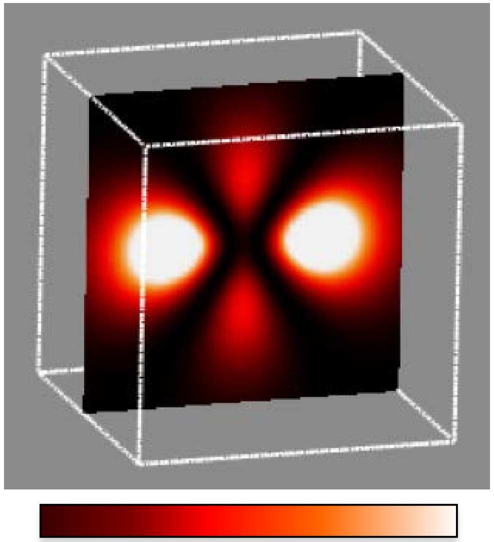


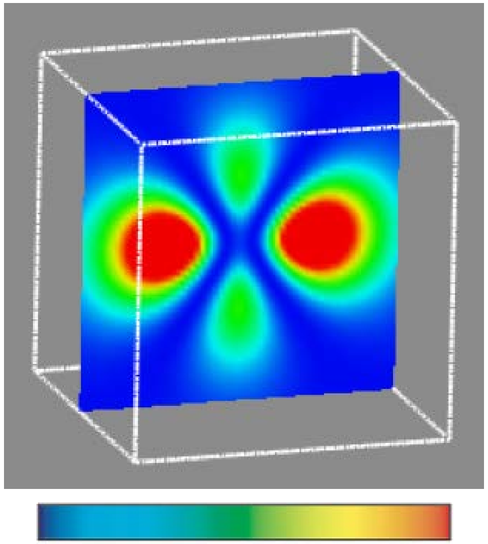
Gray-value colormap
•white = hard tissues (bone)
•gray = soft tissues (flesh)
•black = air

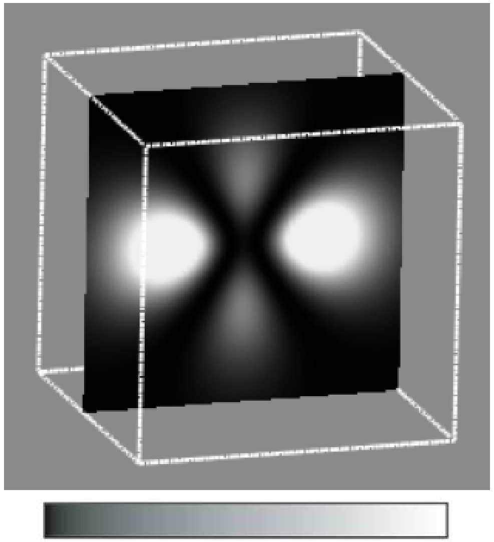
Rainbow colormap
•red = hard tissues (bone)
•blue = air
•other colors = soft tissues

Colormap comparison

2D slice in 3D hydrogen atom potential field







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

Heat colormap
- maxima not prominent
- lower values better
- separable

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

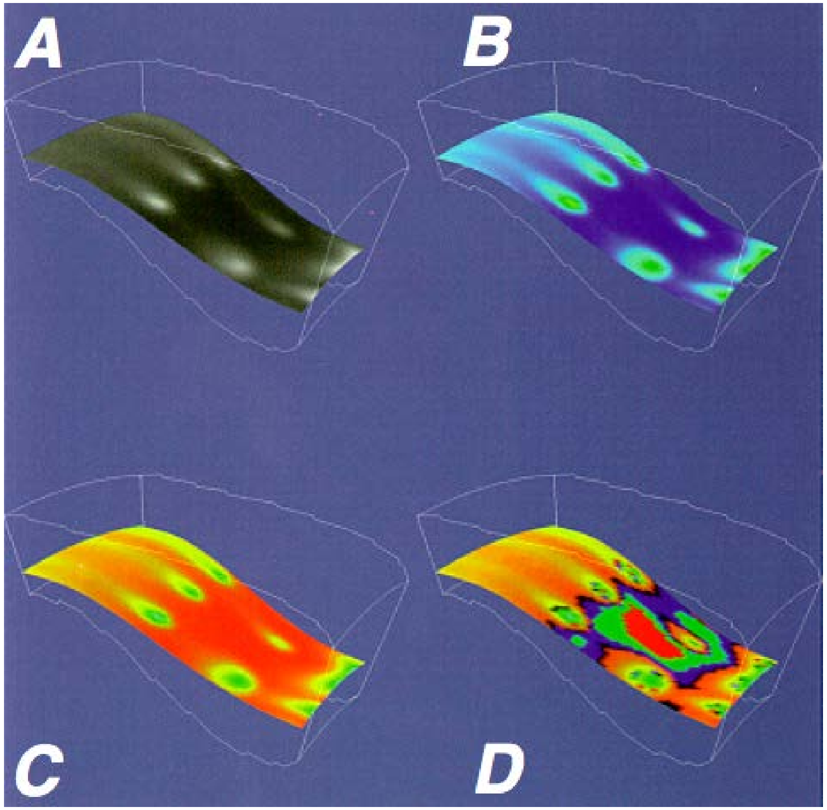
Which is the better colormap? Depends on the application context!

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)

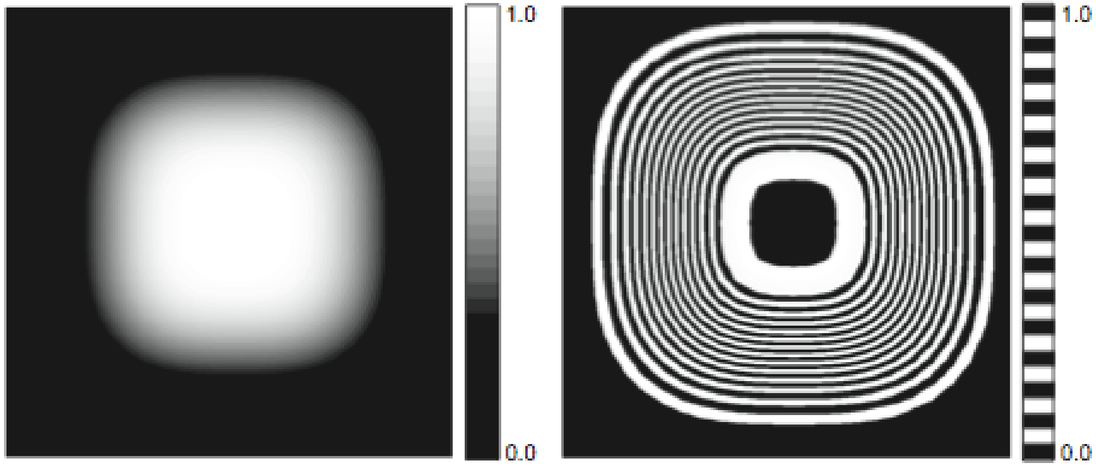
Good design guidelines: www.colorbrewer.org

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

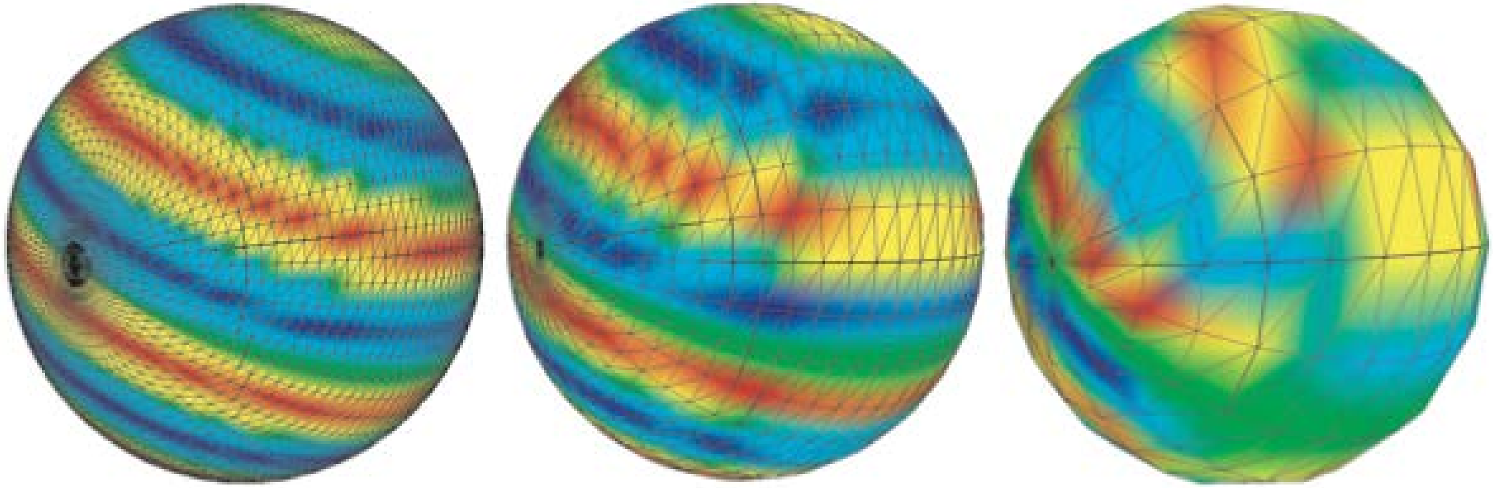
Zebra colormap
- highlights value variations (1st derivative)
- dense, thin bands: fast variation
- thick bands: slow variation

Colormap implementation details

Where to apply the colormap?

- per grid-cell vertex

2D periodic high-frequency function



64x64 points 32x32 points 16x16 points

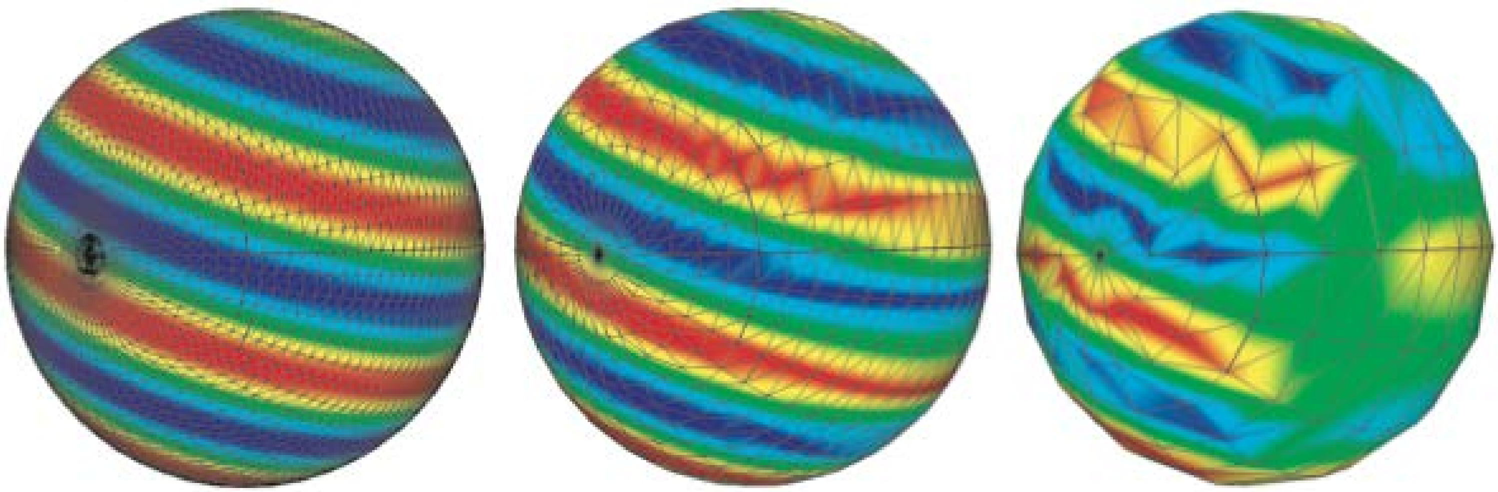
As we decrease the sampling frequency, strong colormapping artifacts appear
Why is this so?

Colormap implementation details

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: $f \to c(f) \to$ interpolation($c(f)$) color interpolation can fall outside colormap!
- per-pixel: $f \to$ interpolation(f) $\to c$(interpolation(f)) colors always stay in colormap

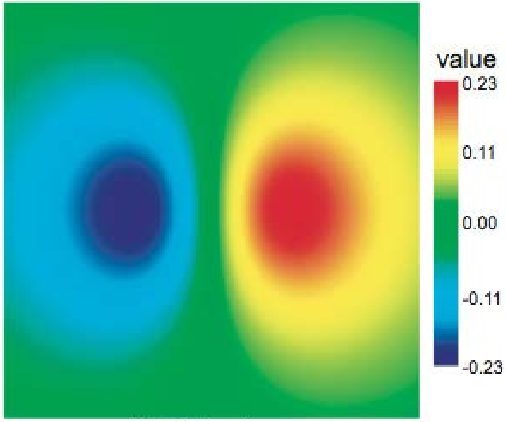
---

See Sec. 5.2 for details

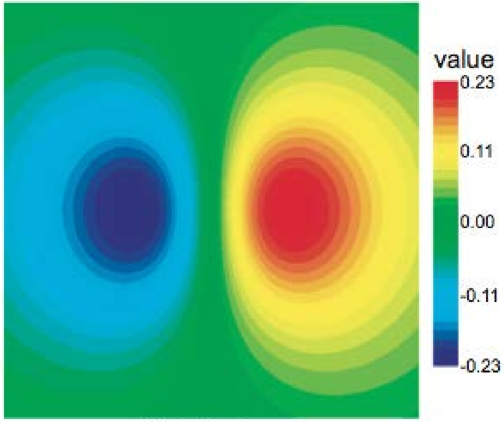
Color banding

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

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

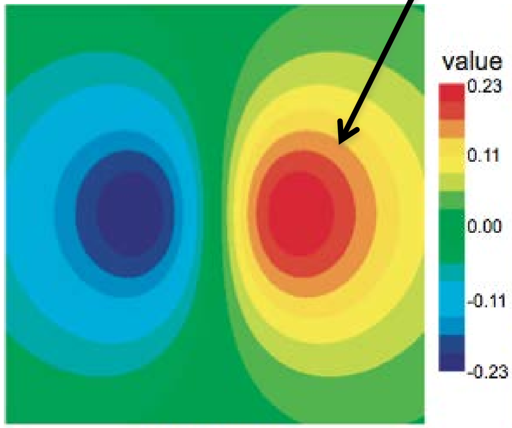


(a) 256 colors

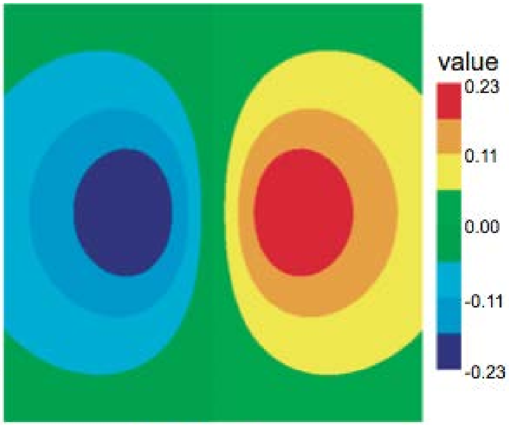


(b) 32 colors

color banding



(c) 16 colors



(d) 8 colors

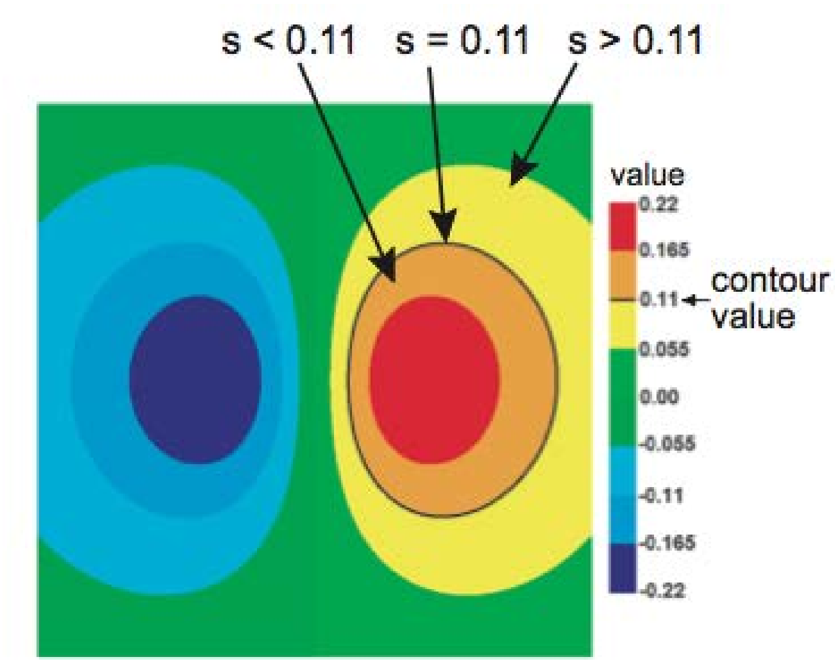
Question

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

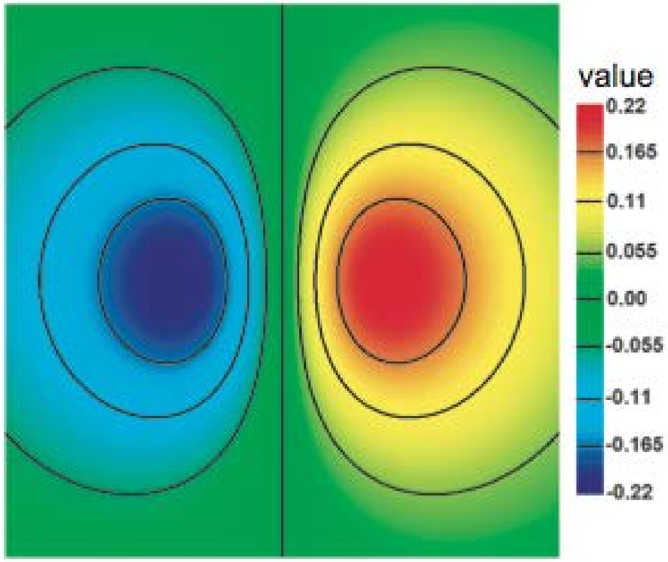
Contouring

How to see where some given values appear in a dataset?

- recall color banding
- a transition separating two consecutive bands = a contour



contour = all points having the scalar value $s = 0.11$

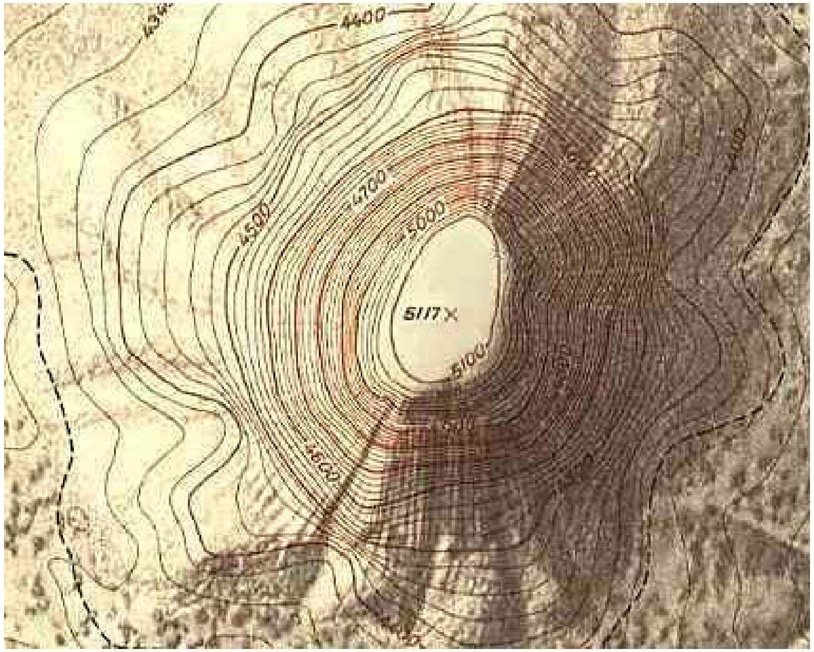


seven different contours, equidistant in value space

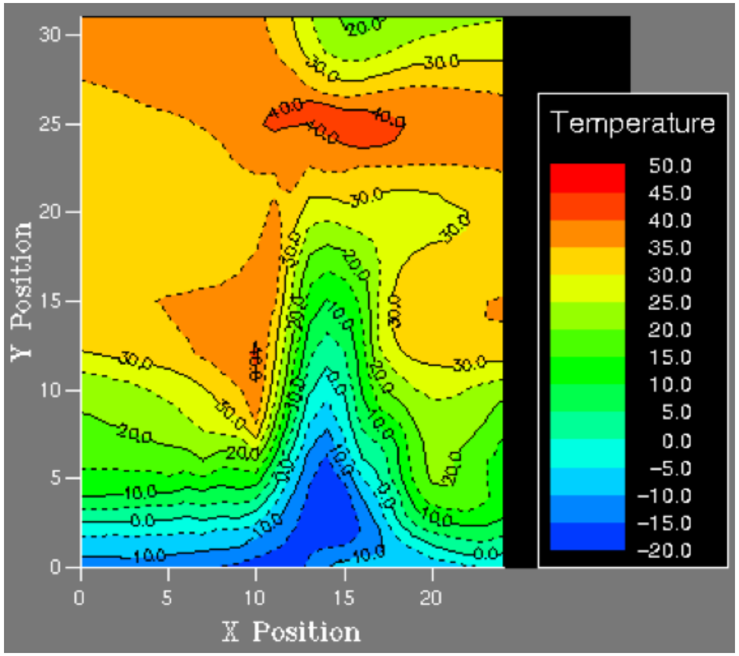
Contouring

Contours are known for hundreds of years in cartography

- also called *isolines* (‘lines of equal value’)



hand-drawn contours on geographical map



computer-generated contours of temperature map

How to compute contours?

Contour properties

Definition

$$I(f_0) = \{ x \in D \mid f(x) = f_0 \}$$

Contours are always closed curves (except when they exit D)

- why? Recall that f is C^0

Contours never (self-)intersect, thus are nested

- why? Think what would mean if a point belonged to two *different* contours

Contours cut D into values smaller resp. larger than the isovalue

- why? Think of definition

Contour properties

Contours are always orthogonal to the scalar value's gradient

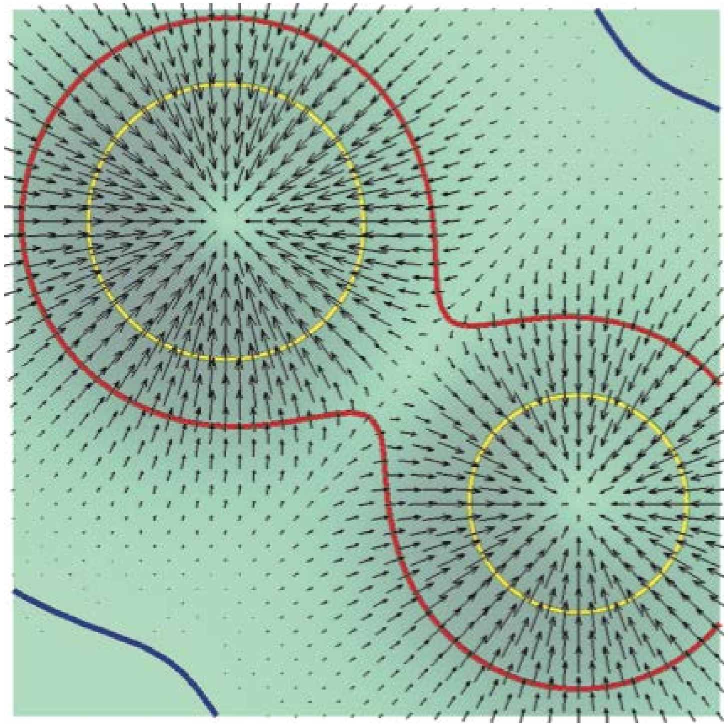
- why? Recall definitions

$$I(f_0)=\left\{x\in D \,\middle|\, f(x)=f_0\right\}$$

contour: $\frac{\partial f}{\partial I}=0$ since f constant along I

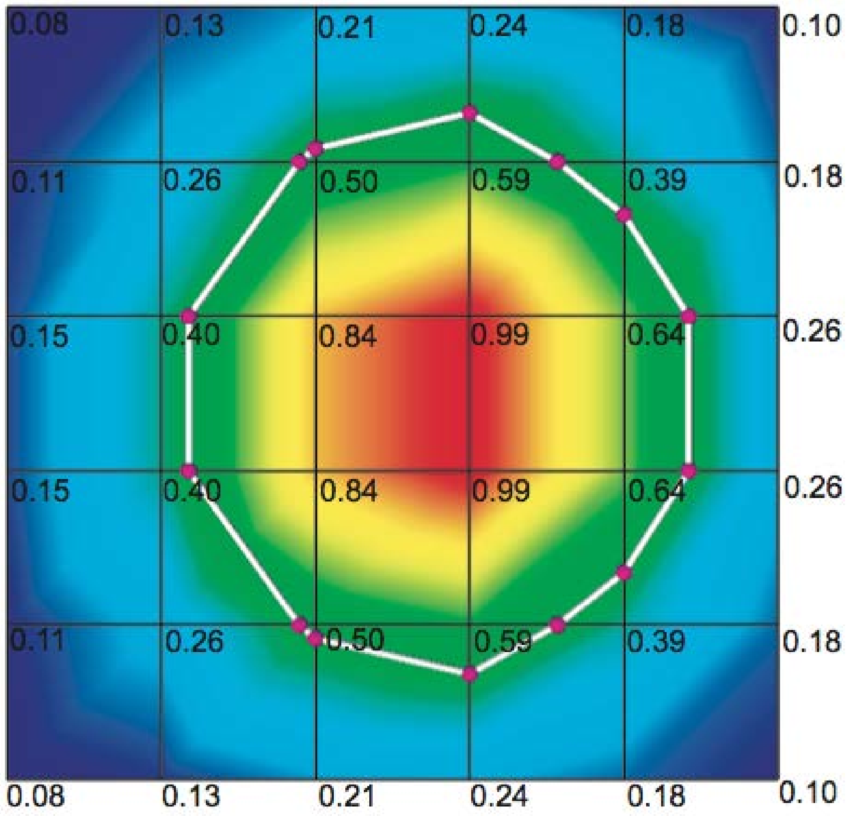
$$\nabla f=\left(\frac{\partial f}{\partial x},\frac{\partial f}{\partial y}\right)$$

gradient: $\frac{\partial f}{\partial(\nabla f)}=\max$ by definition of gradient



gradient of a scalar field (drawn with arrows) is orthogonal to contours

Basic contouring algorithm



```
for(each cell $c$ in $D$)
{
    $S = \emptyset$                  //no contour-edge cuts
    for(each edge $e=(p_i,p_j)$ of $c$)
    {
        if($f_i < v < f_j$)          //$e$ cuts contour
        {
            $q = \frac{p_i(v_j - v) + p_j(v - v_i)}{v_j - v_i}$
            $S = S \cup q$
        }
    }
    connect points in $S$ with lines to build contour;
}
```

Works OK but it is

- cumbersome: connecting contour-edge cuts into lines is not trivial to program
- slow: edges intersecting contours are processed twice

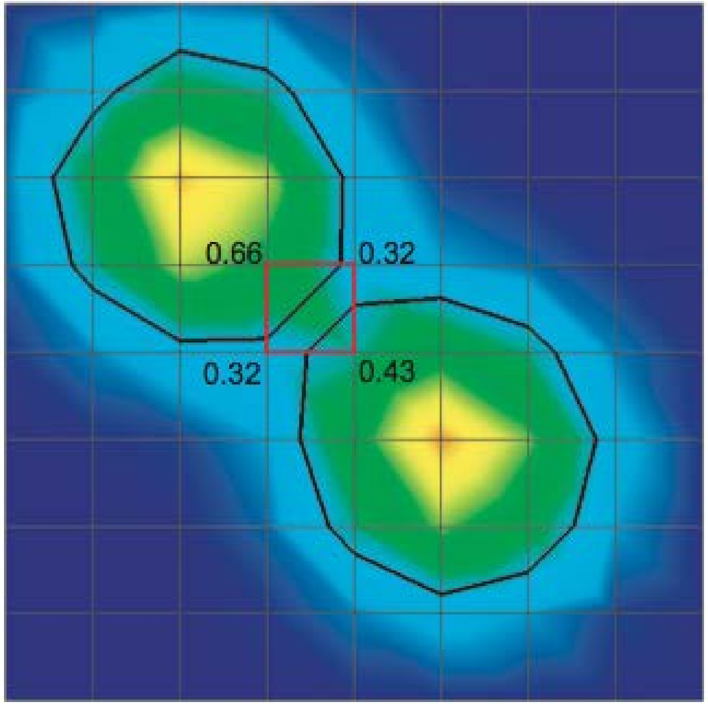
Question

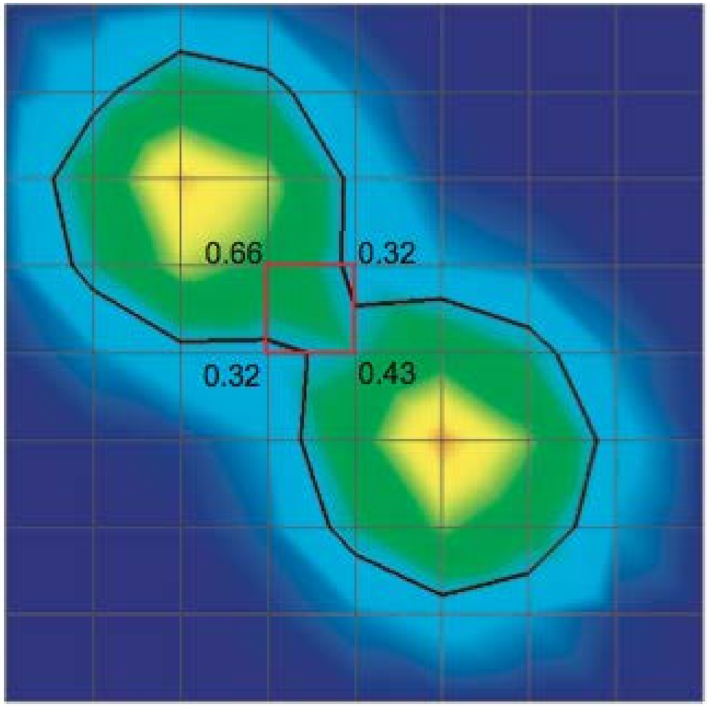
- Are contours piecewise-linear? Why (not)?

Contouring ambiguity

Each edge of the red cell intersects the contour

- which is the right contour result?





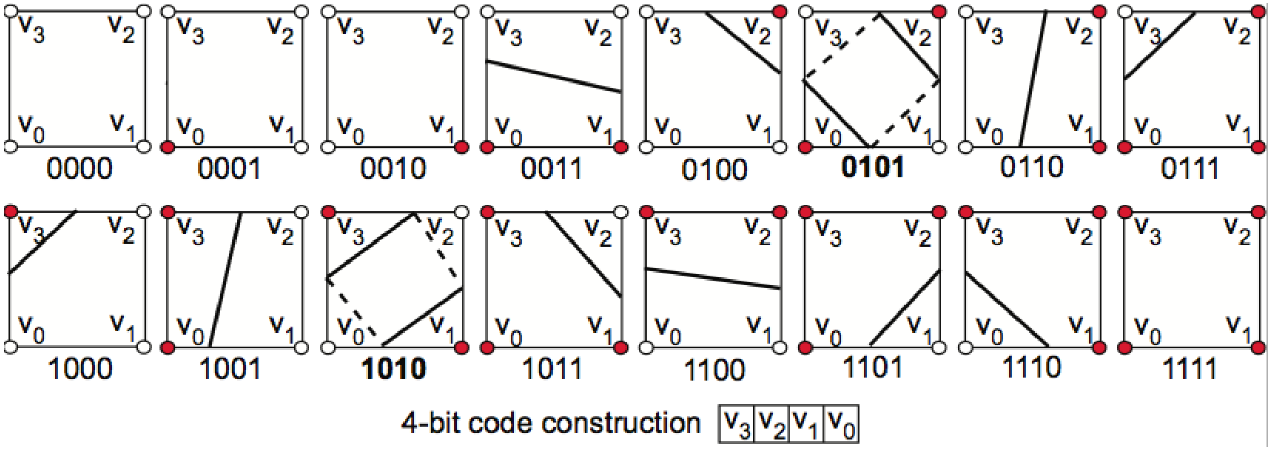
Both answers are equally correct!

- we could discriminate only if we had higher-level information (e.g. topology)
- at cell level, we cannot determine more
- same would happen if we first split quads into triangles (2 splits possible..)

Marching squares

Fast implementation of 2D contouring on quad-cell grids

1. Encode inside/outside state of each vertex w.r.t. contour in a 4-bit code



e.g.
inside: $f > f_0$
outside: $f \leq f_0$

2. Process all dataset cells

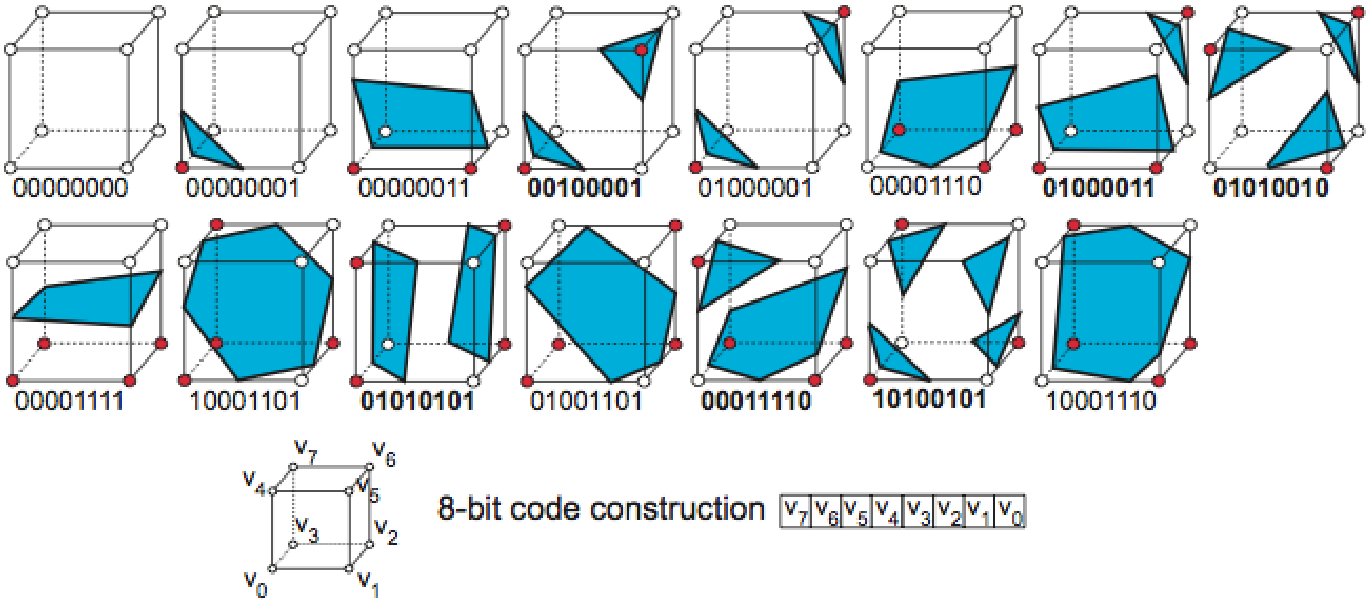
- for each cell, use codes as pointers into a jump-table with 16 cases
- each case has hand-optimized code to
  - compute only the existing edge-contour intersections
  - automatically create required contour segments (connect intersections)
  - reuse already-computed contour segment vertices from previous cells

Note: same can be done for triangles (‘marching triangles’)

Marching cubes

Fast implementation of 3D contouring (isosurfaces) on parallelepiped-cell grids

1. Encode inside/outside state of each vertex w.r.t. contour in a 8-bit code



e.g.
inside: $f > f_0$
outside: $f \leq f_0$

2. Process all dataset cells

- for each cell, use codes as pointers into a jump-table with 15 cases (reduce the 2^8=256 cases to 8 by symmetry considerations)

Marching cubes (cont'd)

- For each case
  - compute the cell-contour intersection → triangles, quads, pentagons, hexagons
  - triangulate these on-the-fly → triangle output only

3. Treat ambiguous cases

- 6 such cases (see **bold**-coded figures on previous slide)
- harder to solve than in 2D (need to prevent false cracks in the surface)
- see Sec. 5.3 for algorithmic details

4. Compute isosurface normals

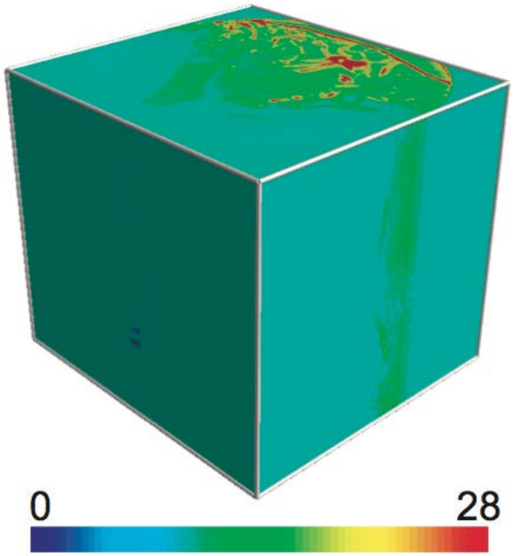
- by face-to-vertex normal averaging (see Module 2, Data resampling)
- directly from data

$$\forall x \in I,\ n_I(x) = -\frac{\nabla f(x)}{\left\|\nabla f(x)\right\|}$$

(gradient is normal to contours, see previous slides)

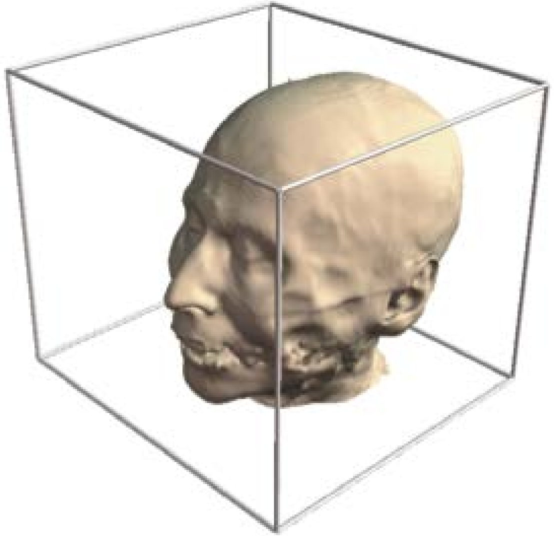
5. Draw resulting surface as a (shaded) unstructured triangle mesh

Marching cubes

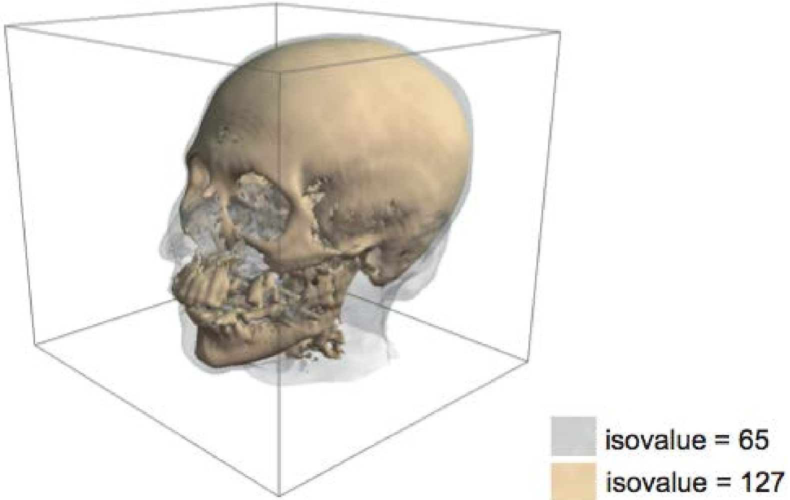


scalar CT volume
(tissue density)

isosurfaces



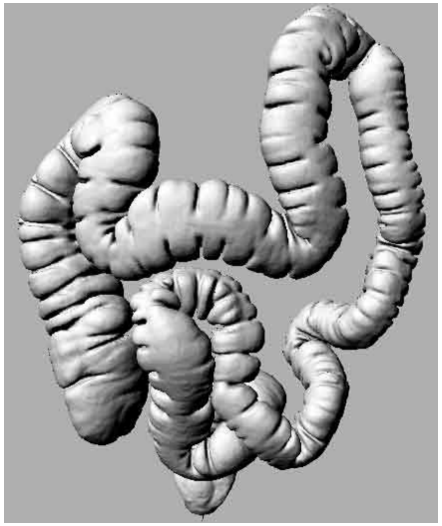
isosurface for scalar value
corresponding to skin



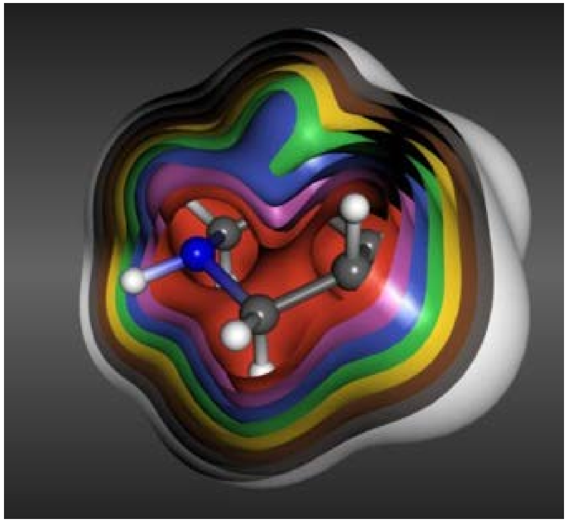
isosurfaces for skin and bone

- extremely simple to use tool
- insightful results

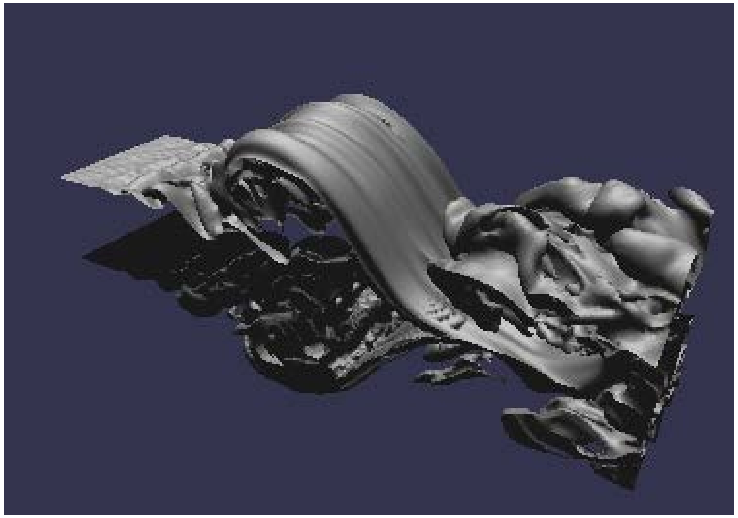
Isosurface examples



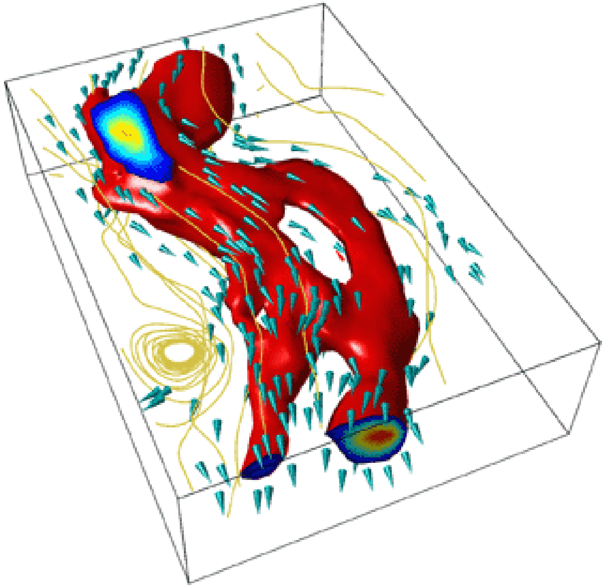
colon (CT dataset)



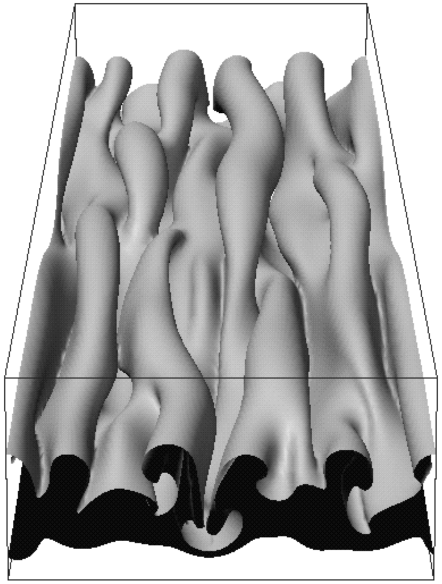
electron density in molecule



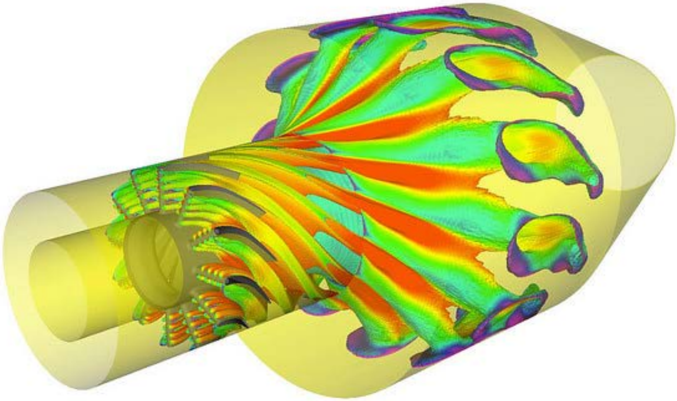
velocity in 3D fluid flow



velocity in 3D fluid flow



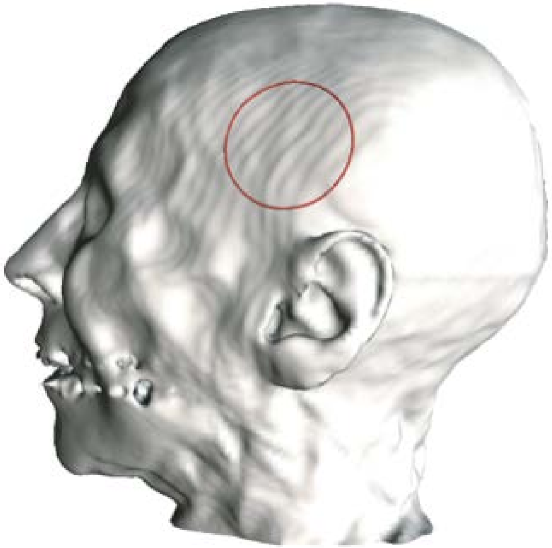
magnetic field in sunspots

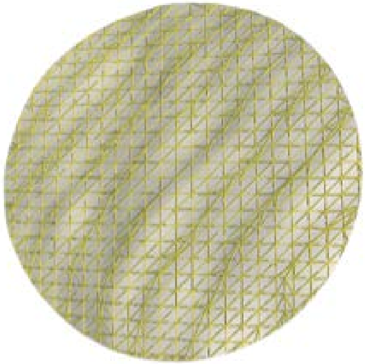


fuel concentration, colored
by temperature in jet engine

Marching cubes – technical points

overview





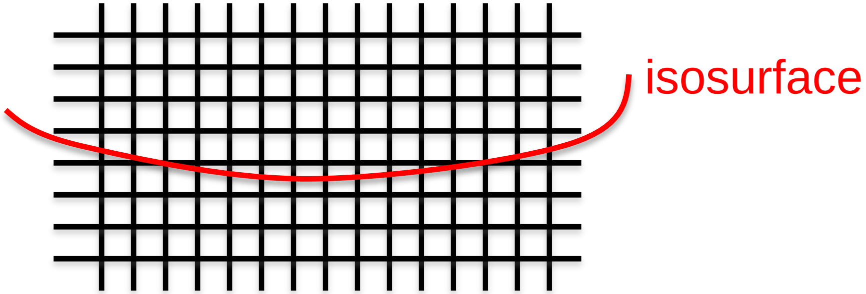
detail

Does this person have wavy wrinkles on his head’s skin?

•so it looks from the visualization…

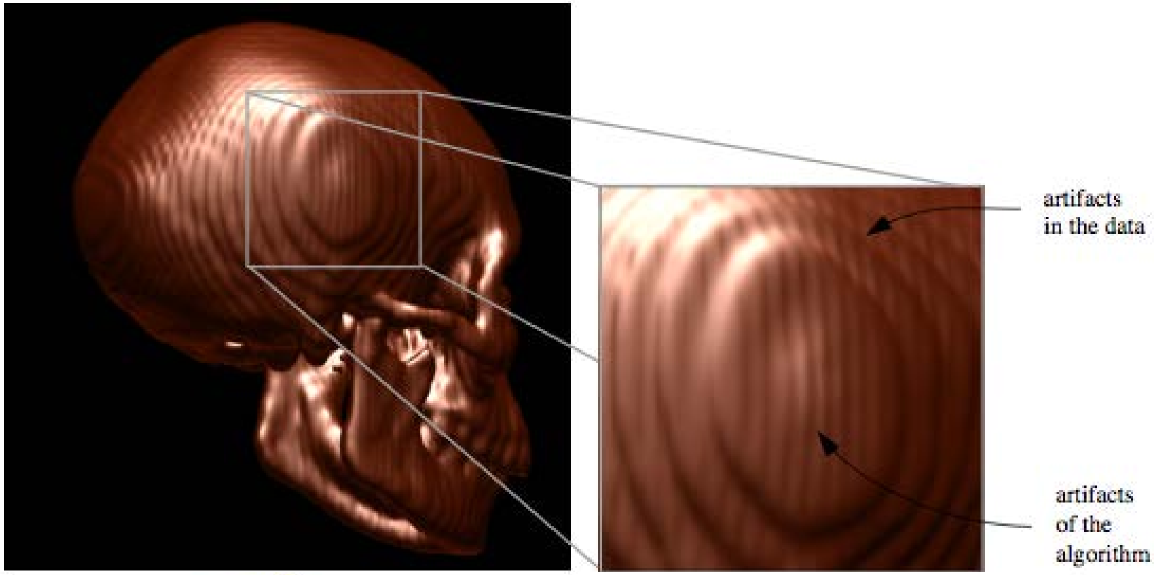
•these are so-called ‘ringing artifacts’

- due to the near-tangent orientation of the isosurface w.r.t. finite-resolution volume grid



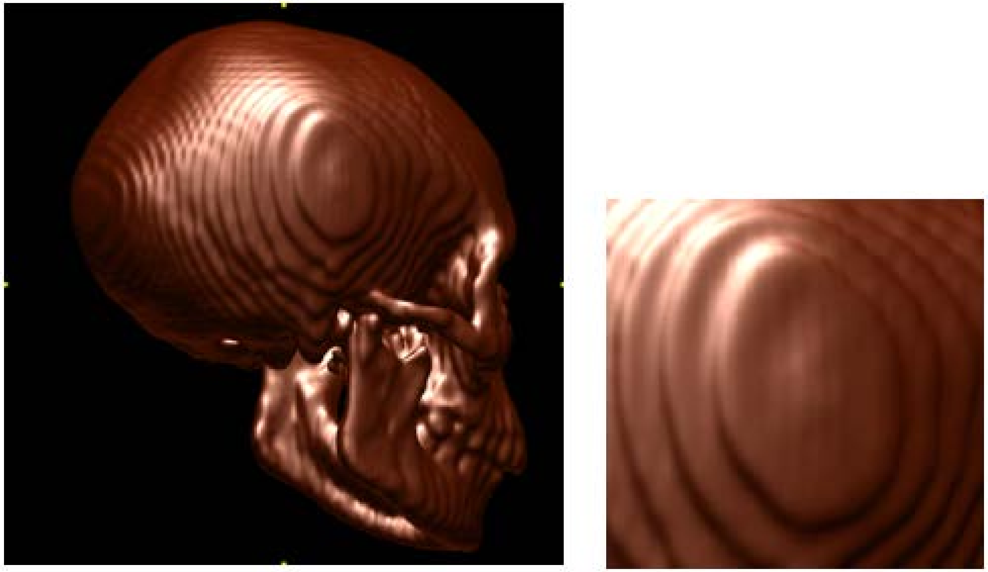
Marching cubes – technical points

A closer look at ringing artifacts



Two kinds of artifacts

- from data: cannot remove easily
- from algorithm (due to linear interpolation)



Removing algorithm artifacts

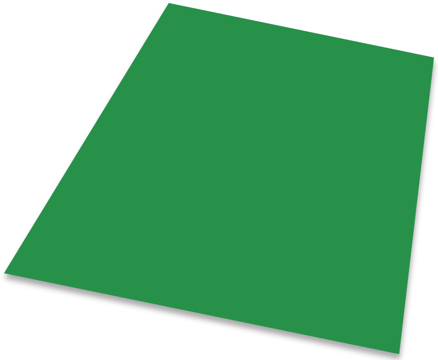
- use higher-order interpolants (e.g. splines)

E. C. LaMar, B. Hamann, K. Joy, High-Quality Rendering of Smooth Isosurfaces, JVCA vol. 10, 1999, 79-90

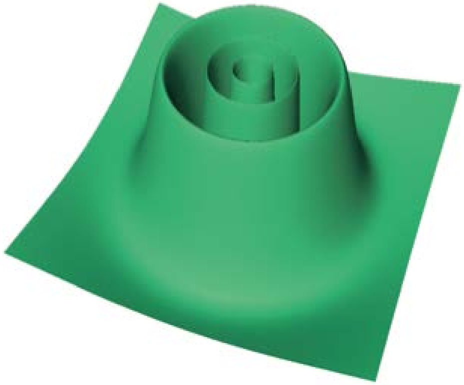
Height / displacement plots

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

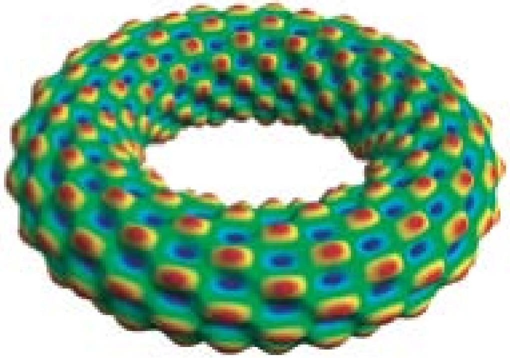


displaced surface S_{displ}

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

Summary

Scalar Algorithms (book Chapter 5)

- colormapping
- contouring (2D and 3D)
- height plots
- displacement plots
- read Ch. 5 *in detail* to understand all the algorithmic issues!

Next module

- vector visualization algorithms