CS 519: Scientific Visualization

Scalar Data Visualization: Isosurfaces

Eric Shaffer

Some slides adapted from Tao Ju, Washington University St. Louis and Alexandru Telea, *Data Visualization Principles and Practice*

Colormaps: Transfer Functions versus Lookup Tables

Transfer Functions Basic idea

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

Color Tables Basic Idea

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

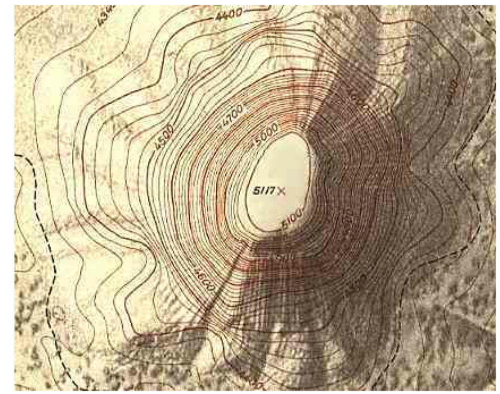
Color mapping ← $c_i = c\left(\frac{i}{N}\right)$ (color table) ← $c : [0,1] \rightarrow [0,1]^3$ (desired color transfer function)

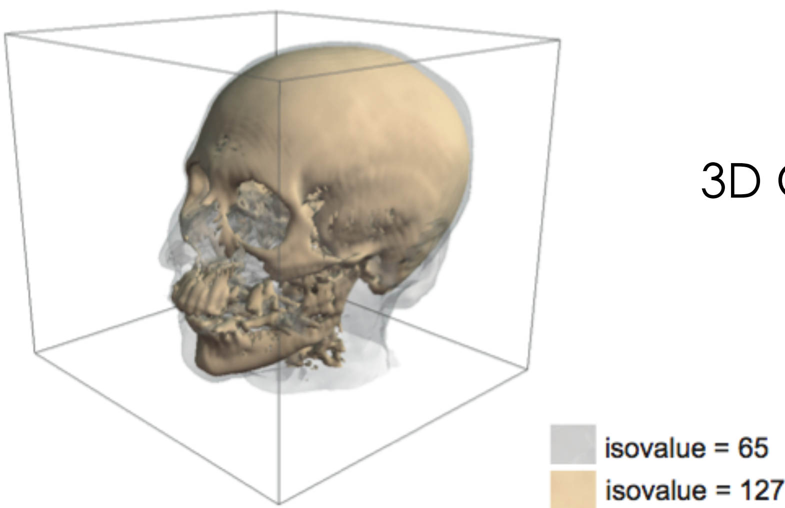
f is color-mapped to c_i

Contouring

Contours have been used for hundreds of years in cartography

- also called *isolines* ('lines of equal value')





3D Contouring: Marching Cubes:

"*Marching cubes: A high resolution 3D surface construction algorithm*", by Lorensen and Cline (1987)

>6000 citations on Google Scholar

Contour Properties

Definition

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

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

- why? Recall that *f* is C^0

Two different contour lines never intersect, thus are nested

- why? What would it mean if a point belonged to two *different* contours

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

- why?

Contour Properties

Contours are always orthogonal to the scalar value's gradient

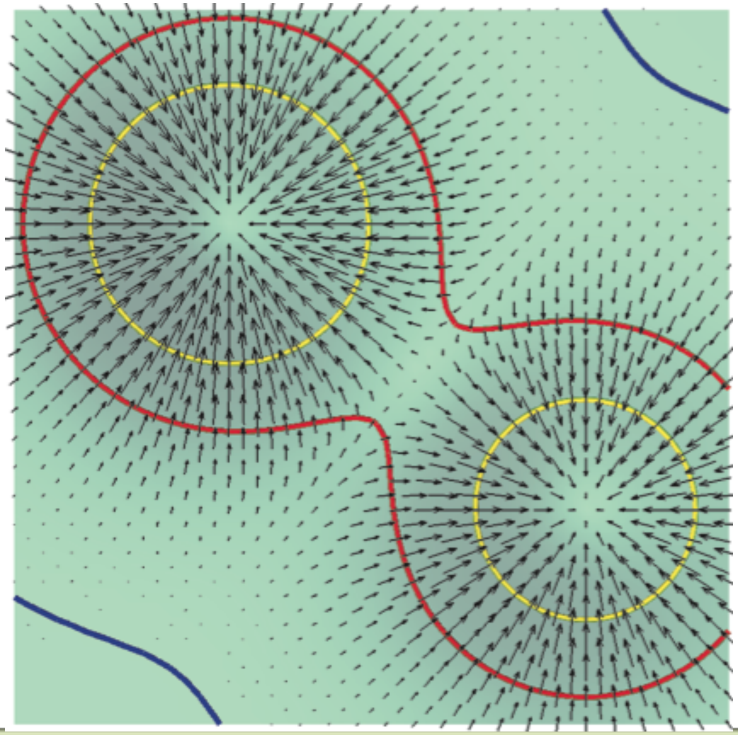
- why? Recall definitions

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

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

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

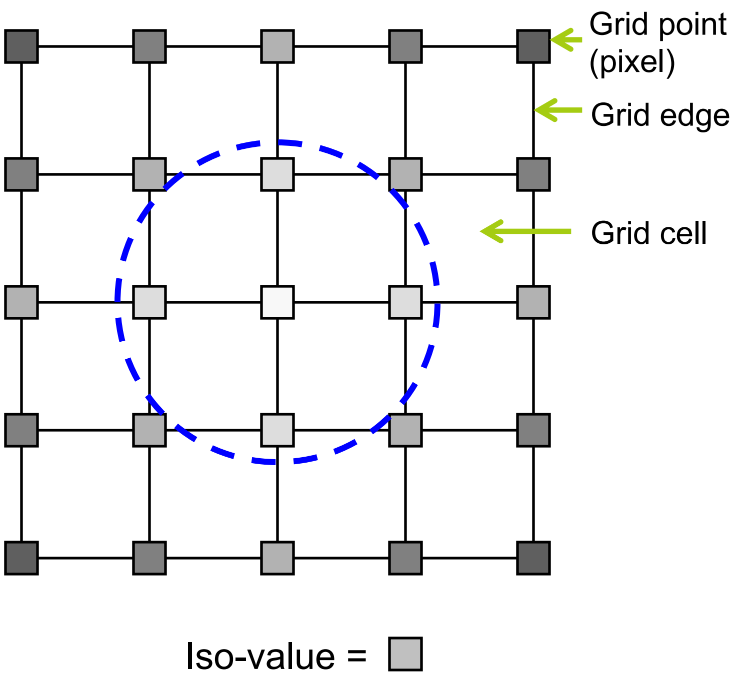
gradient: $\frac{\partial f}{\partial(\nabla f)} = \max$ by definition of gradient direction of greatest increase in f



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

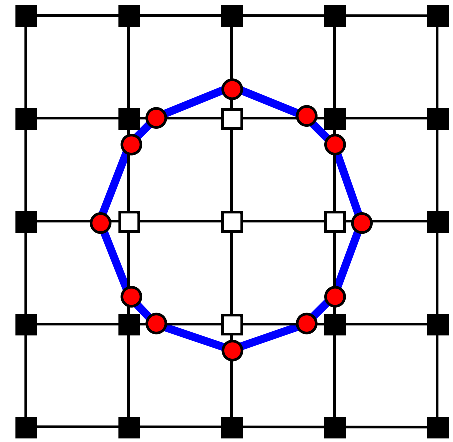
Contouring (On A Grid)

- Input
  - A grid where each grid point (pixel or voxel) has a value (color)
  - An iso-value (threshold)
- Output
  - A closed polyline (2D) or mesh (3D) that separates grid points above or below the iso-value

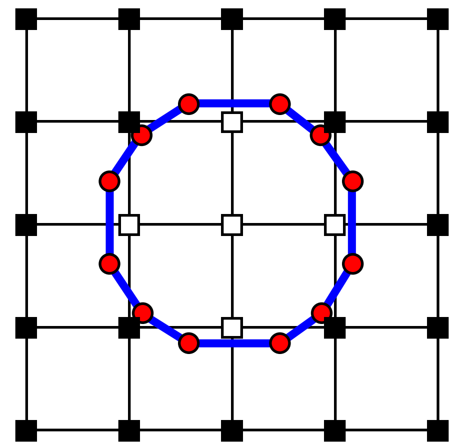


Algorithms

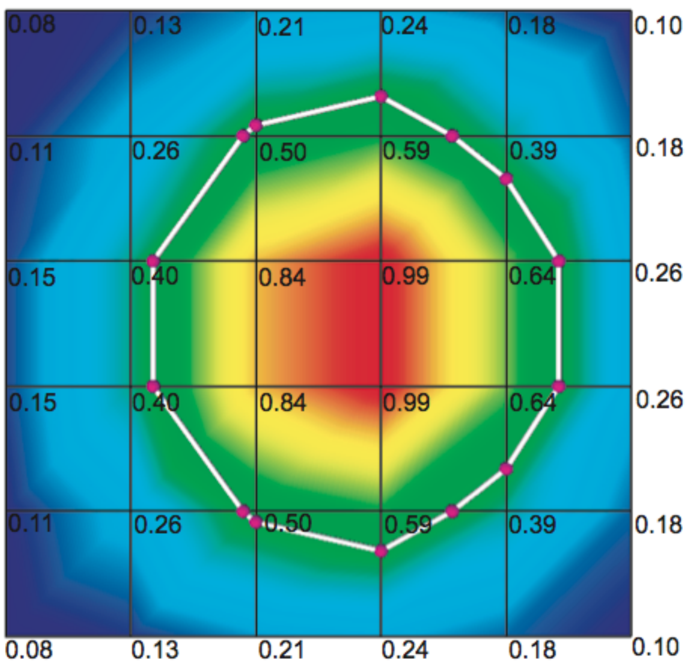
- **Primal methods**
  - Marching Squares (2D), Marching Cubes (3D)
  - Placing vertices on grid edges



- **Dual methods**
  - Dual Contouring (2D,3D)
  - Placing vertices in grid cells



Contouring in 2D

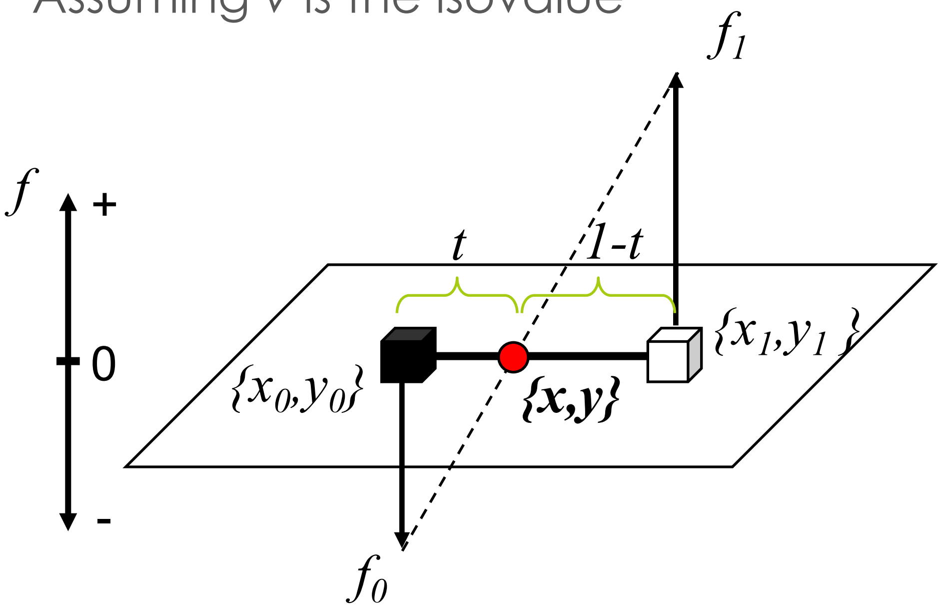


$S = \emptyset$

for(each cell c in D)
{
  for(each edge $e=(p_i,p_j)$ of c)
  {
    if($f_i < v < f_j$)
    {
      $$q = \frac{p_i(v_j - v) + p_j(v - v_i)}{v_j - v_i}$$
      $S = S \cup q$
    }
  }
  connect points in S with lines to build contour;
}

Marching Squares (2D)

- Creating vertices
  - Assuming the underlying, continuous function is linear on the grid edge
  - Linearly interpolate the positions of the two grid points
    Assuming v is the isovalue

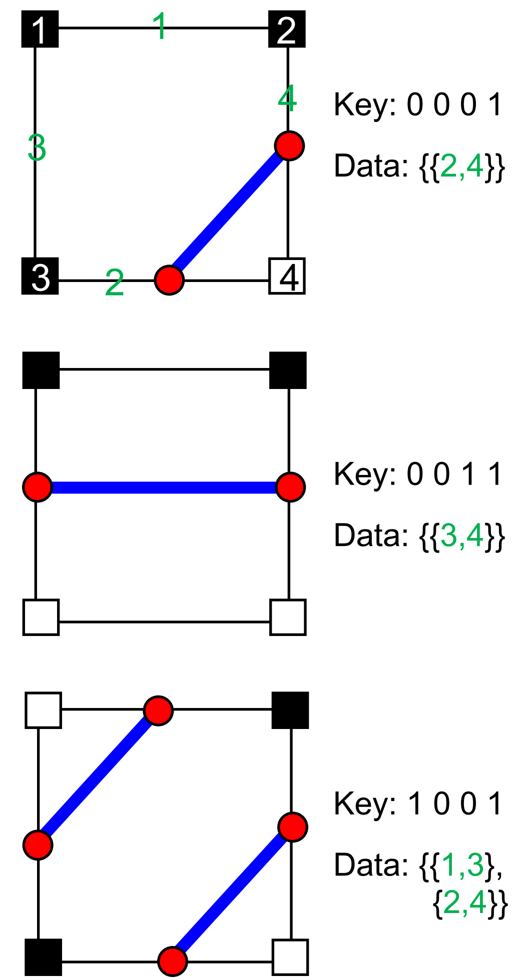


$$t = \frac{v - f_0}{f_1 - f_0}$$

$$x = x_0 + t(x_1 - x_0)$$
$$y = y_0 + t(y_1 - y_0)$$

Marching Squares (2D)

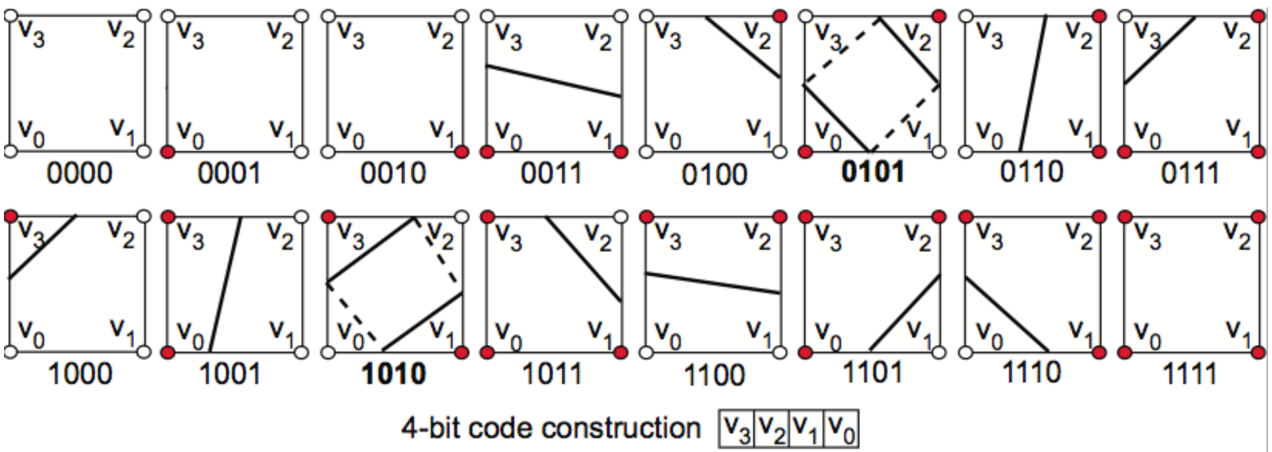
- Connecting vertices by lines
  - Lines shouldn't intersect
  - Each vertex is used once
    - So that it will be used exactly twice by the two cells incident on the edge
- Two approaches
  - Do a walk around the grid cell
    - Connect consecutive pair of vertices
  - Or, using a pre-computed look-up table
    - 2^4=16 entries
    - For each sign combination at the 4 corners, it stores the indices of the grid edges whose vertices make up the lines.



Marching Squares

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 table with 16 cases
- each case has code to
  - compute the existing edge-contour intersections
  - connect to already-computed contour vertices from previous cells

Marching Squares

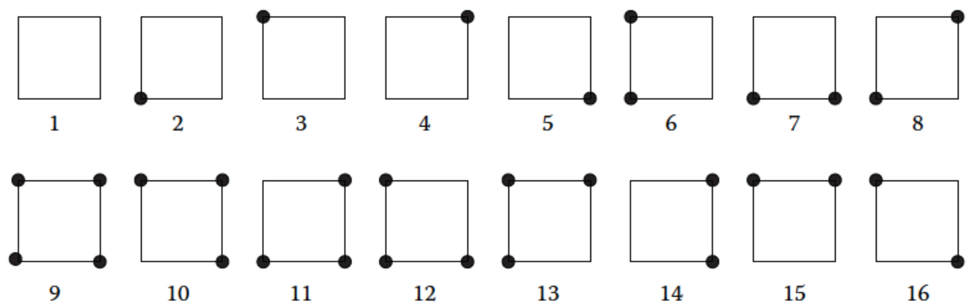


Figure 2.2. Square configurations. Black vertices are positive.

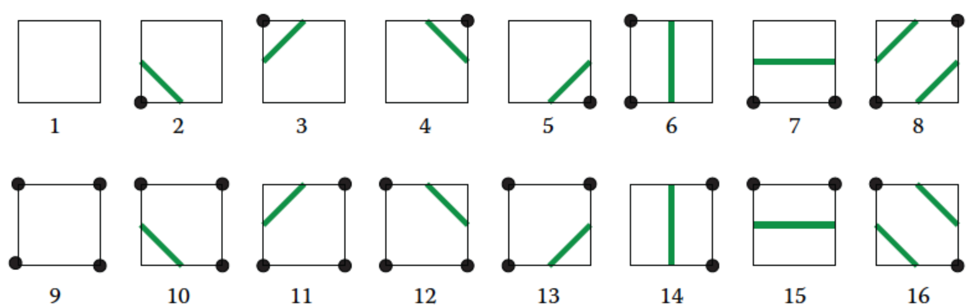
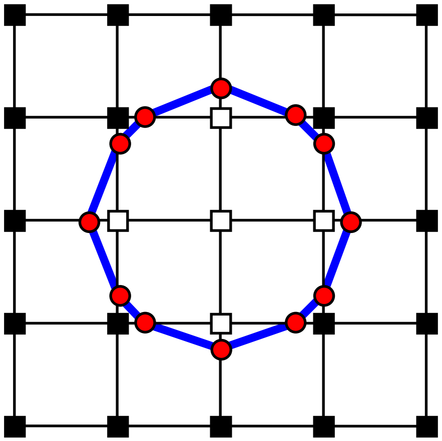


Figure 2.3. Square isocontours. Configurations 1 and 9 have no isocontour. Isocontours for configurations 2–7 and 10–15 are single line segments. Isocontours for configurations 8 and 16 are two line segments.

Implementation Notes

- Avoid computing one vertex multiple times
  - Compute the vertex location once, and store it in a hash table

- When the grid point’s value is same as the iso-value
  - Treat it either as “above” or “below”, but be consistent.



Generate Isocontours for $f_0=5$

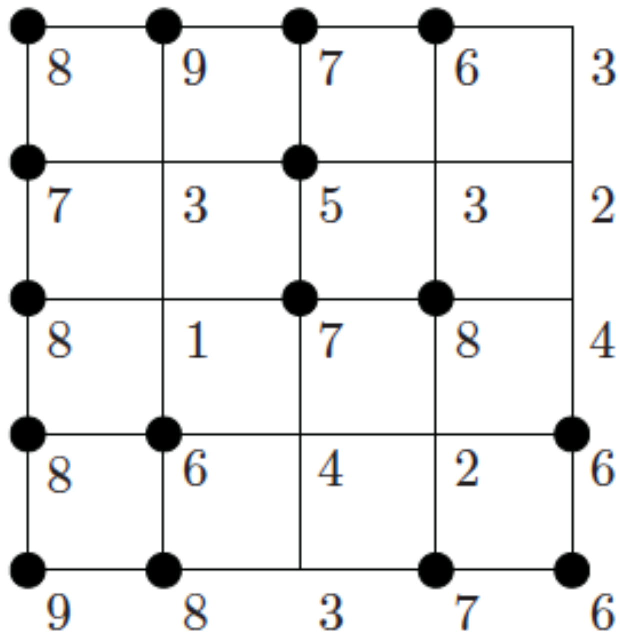
Scalars associated with point to the upper left

Classify points with value 5 as positive

8	9	7	6	3
7	3	5	3	2
8	1	7	8	4
8	6	4	2	6
9	8	3	7	6

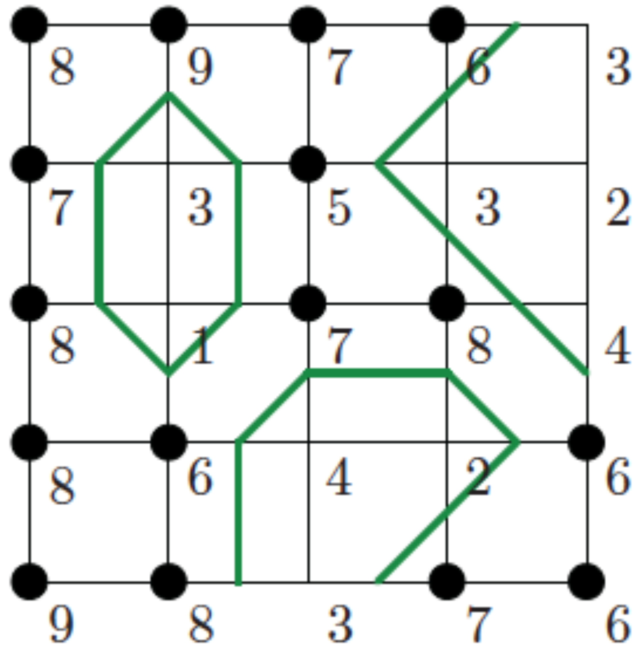
(a) Scalar grid.

+/- Grid for T=5



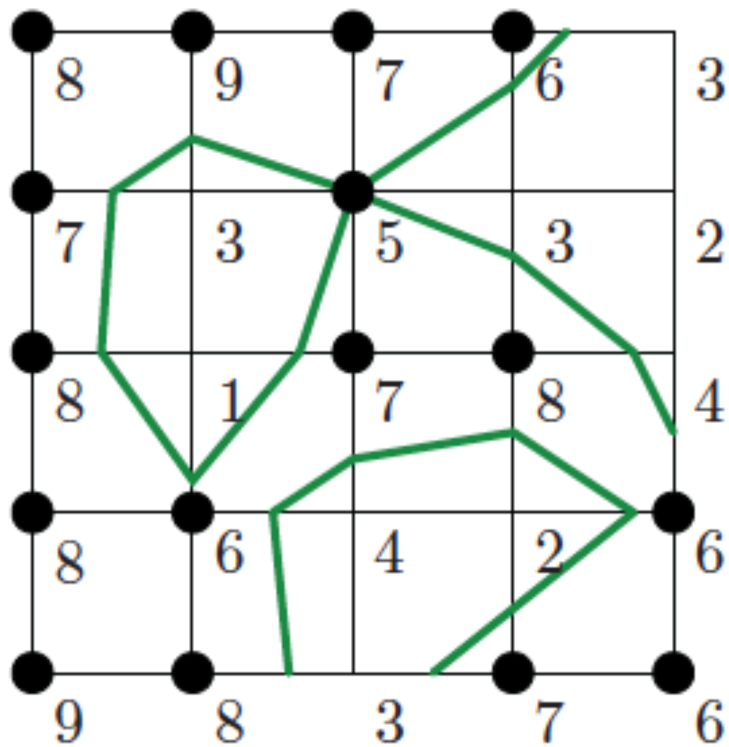
(b) The +/− grid.

Isocontours using midpoints



(c) Midpoint vertices.

Isocontours using Interpolation

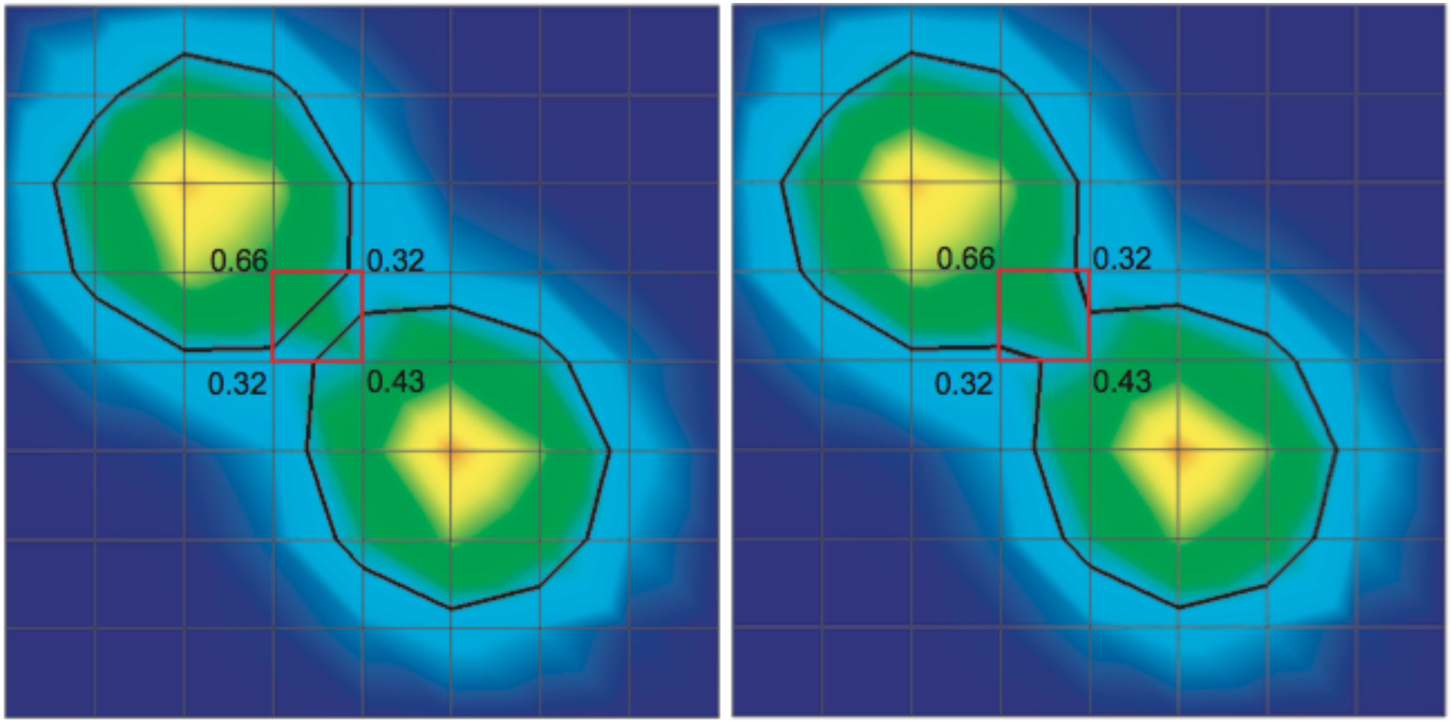


(d) Isocontour.

Contouring Ambiguity

Each edge of the red cell intersects the contour

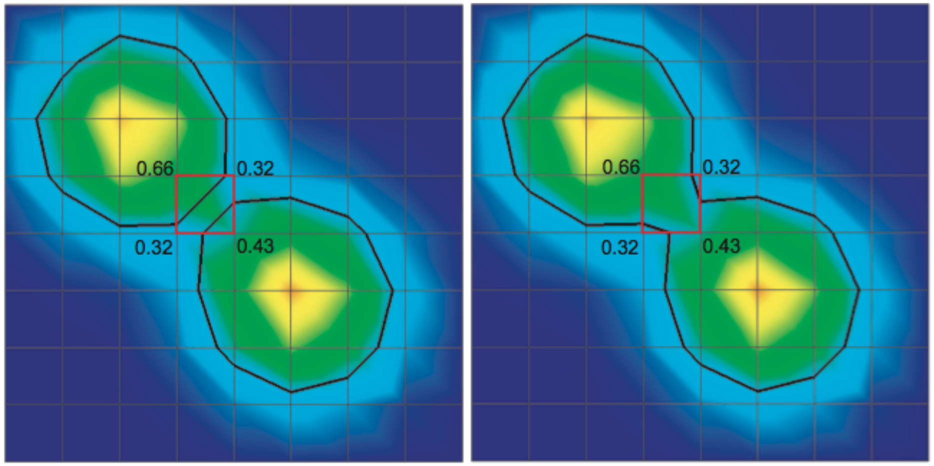
- which is the right contour result?



Contouring Ambiguity

Each edge of the red cell intersects the contour

- which is the right contour result?

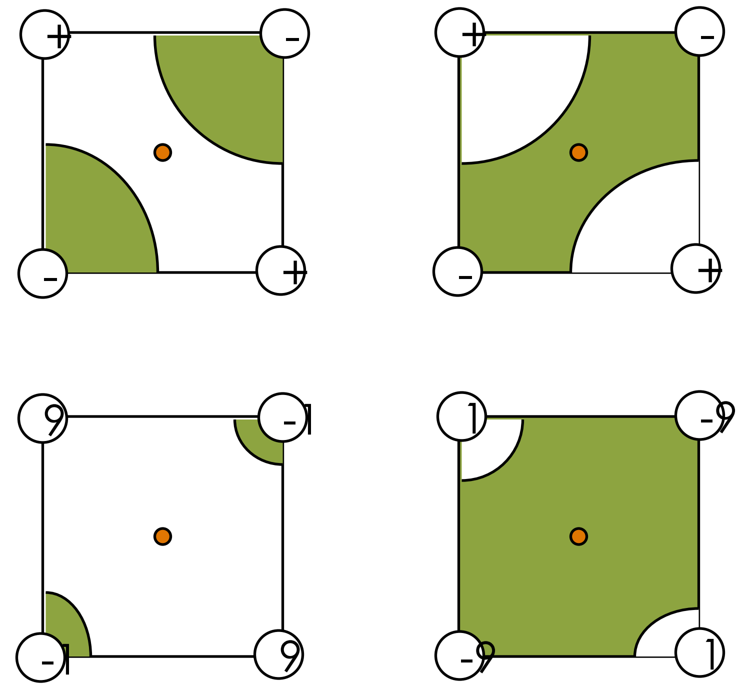


Both answers are equally correct!

- we could discriminate only if we had higher-level information (e.g. topology)
- at cell level, we cannot determine more unless we increase sampling rate

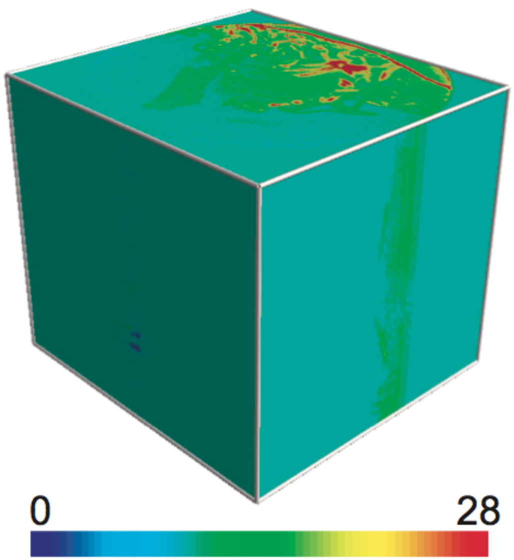
Problem: Ambiguity

- Some cell corner value configurations yield more than one consistent polygon
- In 3-D can yield holes in surface!
- How can we resolve these ambiguities?
- Topological Inference
  - Sample a point in the center of the ambiguous face
  - If data is discretely sampled, bilinearly interpolate samples



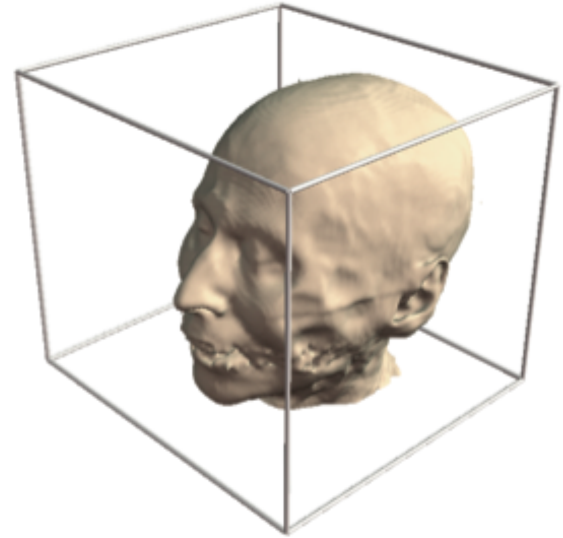
$$p(s,t) = (1-s)(1-t)\,a + s\,(1-t)\,b + (1-s)\,t\,c + s\,t\,d$$

Marching Cubes

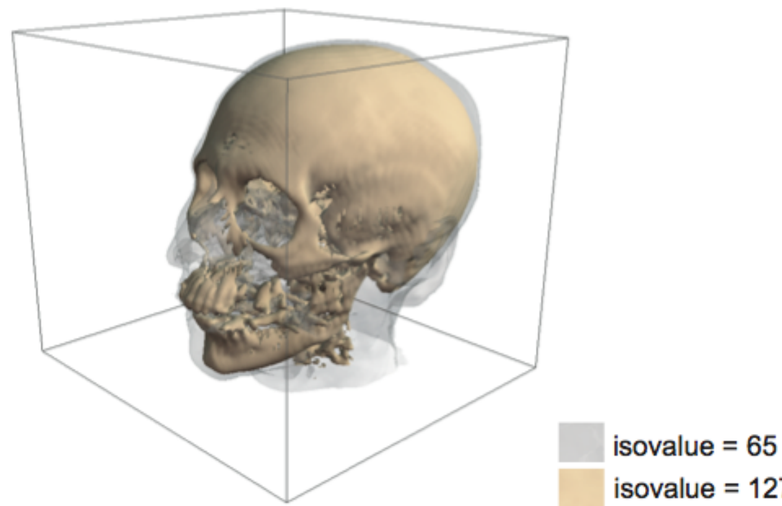


scalar CT volume
(tissue density)

isosurfaces



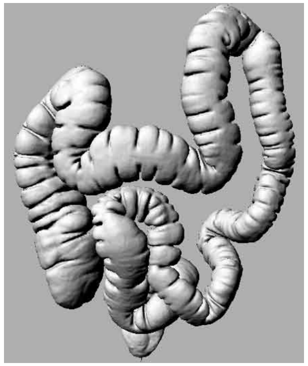
isosurface for scalar value
corresponding to skin



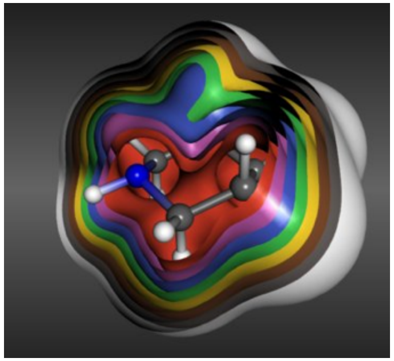
isosurfaces for skin and bone

- extremely simple to use tool
- insightful results

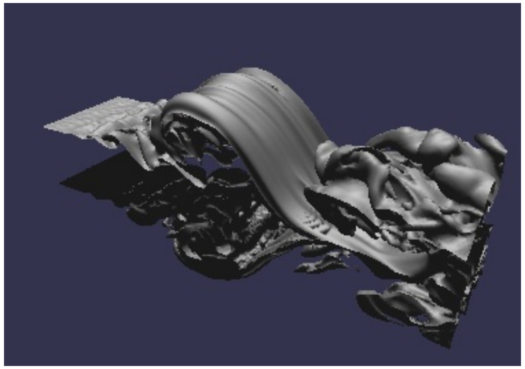
Examples



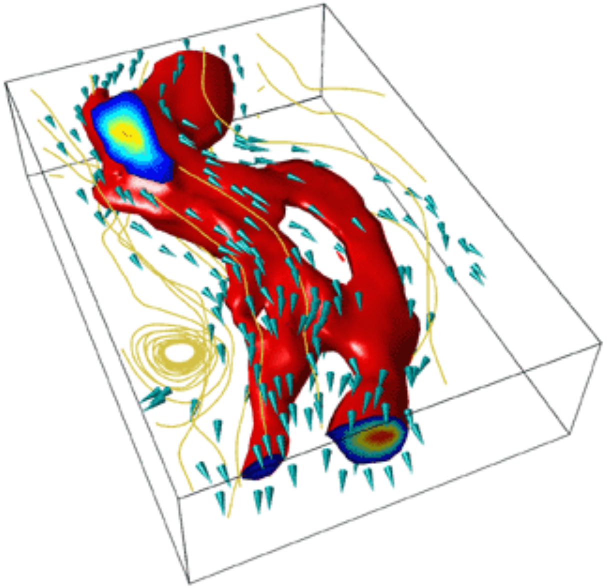
colon (CT dataset)



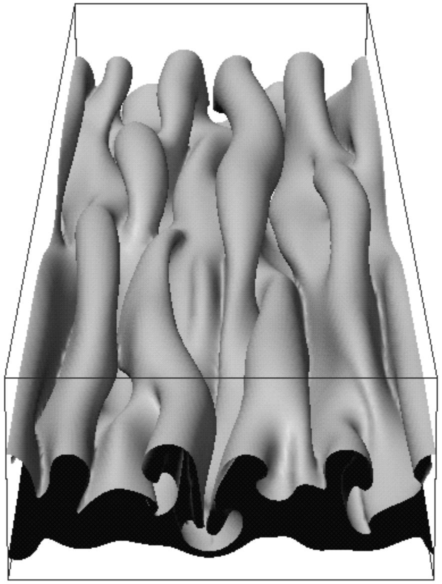
electron density in molecule



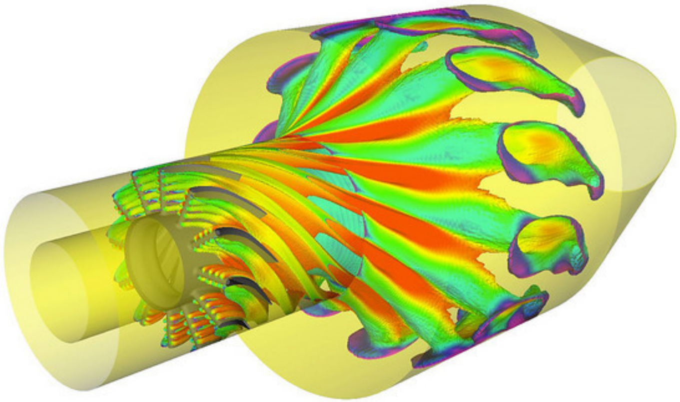
velocity in 3D fluid flow



velocity in 3D fluid flow



magnetic field in sunspots

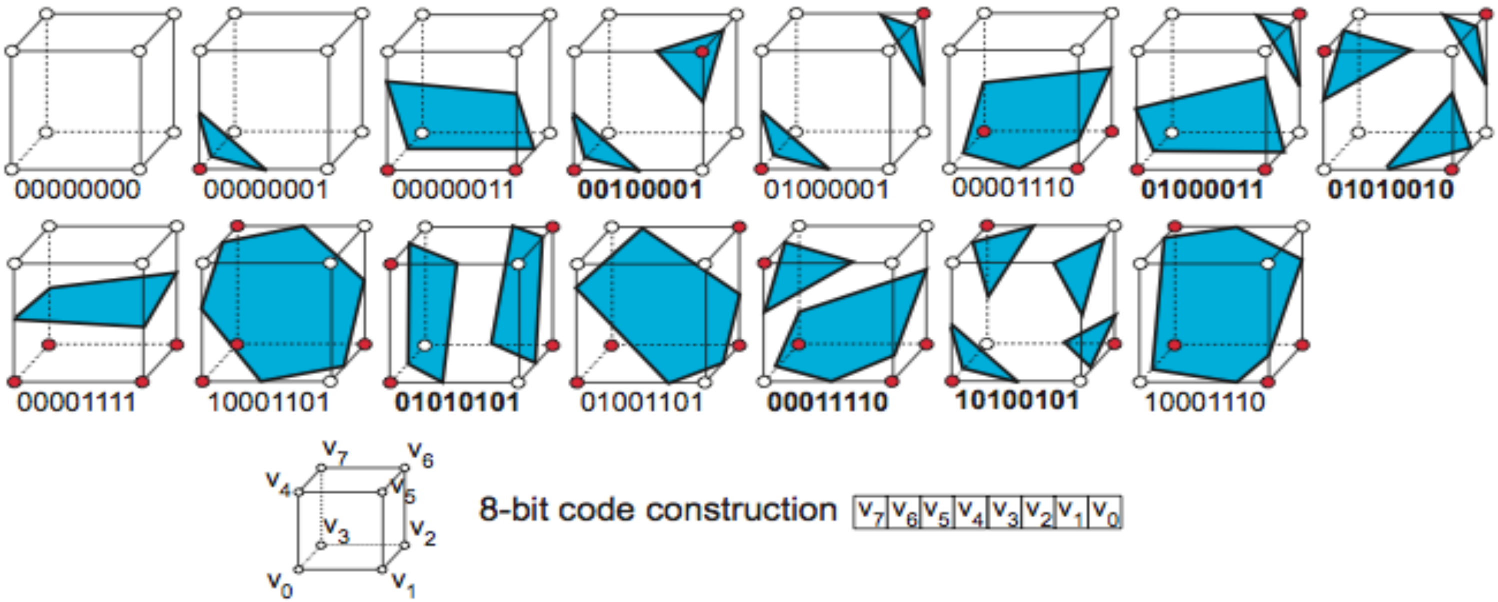


fuel concentration, colored by temperature in jet engine

3D: 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

- For each case
  - compute the cell-contour intersection → triangles, quads, pentagons 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 of the book for algorithmic details

4. Compute isosurface normals

- by face-to-vertex normal averaging
- directly from data

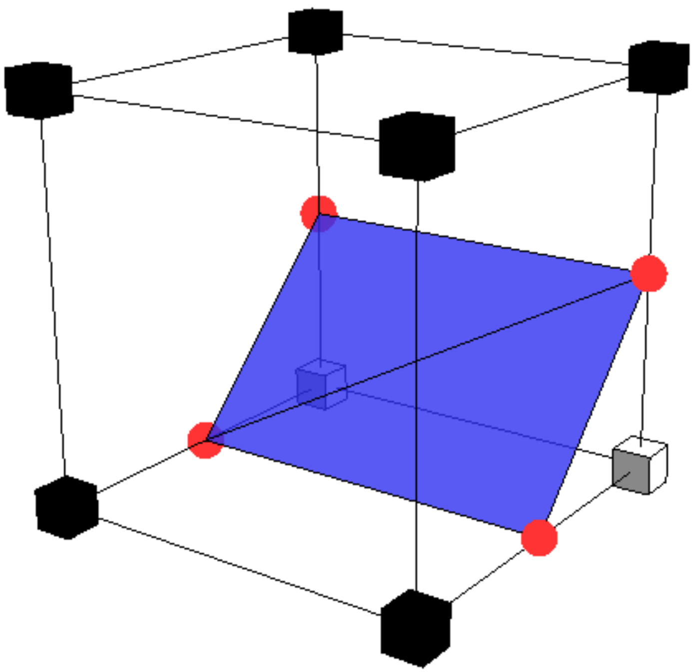
(gradient is normal to contours, see previous slides)

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

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

Marching Cubes (3D)

- For each grid cell with a sign change
  - Create one vertex on each grid edge with a sign change (using linear interpolation)
  - Connect vertices into triangles

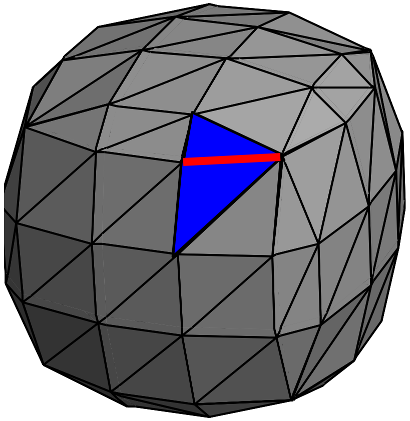


Surface Approximations

- In 2D, the piecewise linear surface approximation is a polyline
- In 3D, it is a triangle mesh

<u>Desired Qualities of the Approximation</u>

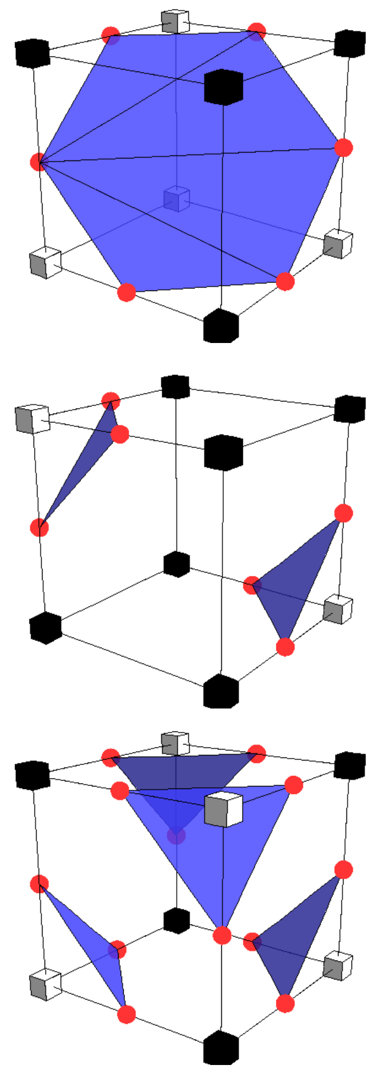
- Closed (with inside and outside)
  - Polyline: a vertex is shared by even # of edges
  - Mesh: an edge is shared by even # of polygons
- Manifold
  - Polyline: a vertex is shared by 2 edges
  - Mesh: an edge is shared by 2 polygons, and a vertex is contained in a ring of polygons
- Non-intersecting



A closed, manifold triangular mesh

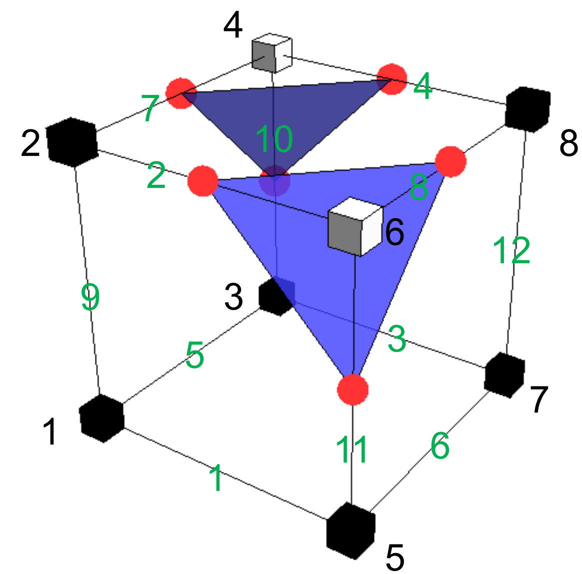
Marching Cubes (3D)

- Connecting vertices by triangles
  - Triangles shouldn't intersect
  - To be a closed manifold:
    - Each vertex used by a triangle "fan"
    - Each mesh edge used by 2 triangles (if inside grid cell) or 1 triangle (if on a grid face)



Marching Cubes (3D)

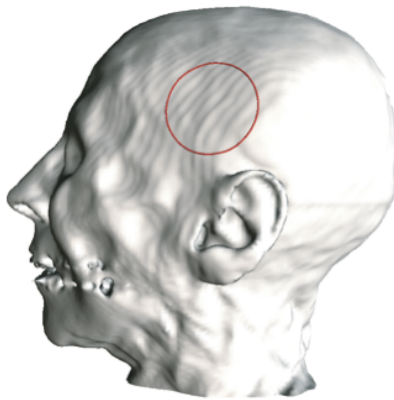
- Connecting vertices by triangles
  - Triangles shouldn't intersect
  - To be a closed manifold:
    - Each vertex used by a triangle "fan"
    - Each mesh edge used by 2 triangles (if inside grid cell) or 1 triangle (if on a grid face)
    - Each mesh edge on the grid face is shared between adjacent cells
- Look-up table
  - 2^8=256 entries
  - For each sign configuration, it stores indices of the grid edges whose vertices make up the triangles

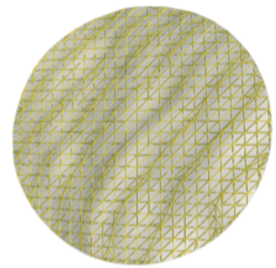


Sign: "0 0 0 1 0 1 0 0"
Triangles: {{2,8,11},{4,7,10}}

Marching Cubes: Artifacts

overview

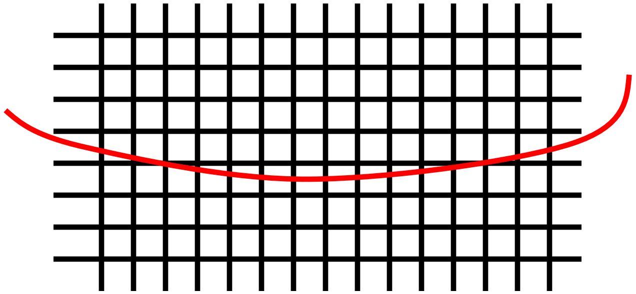




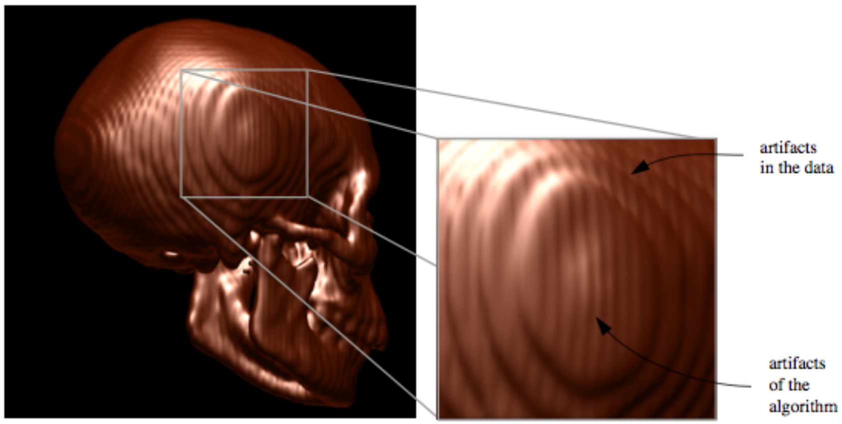
detail

Does this person have wavy wrinkles on his skin?

- these are ‘ringing artifacts’
  - due to the near-tangent orientation of the isosurface w.r.t. finite-resolution volume grid

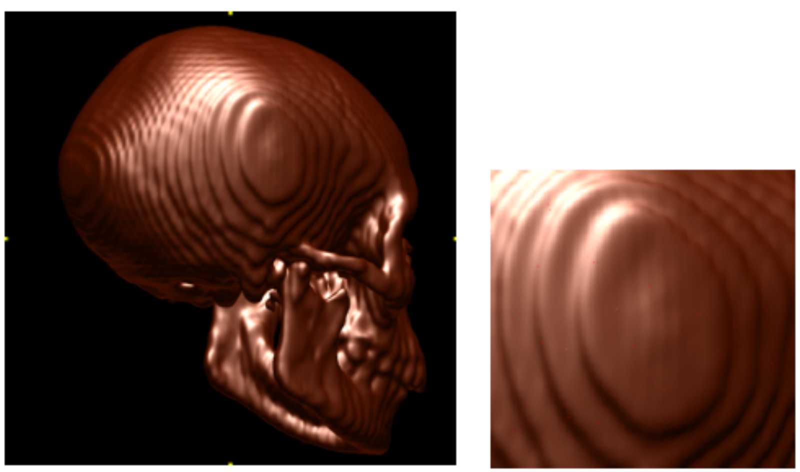


Marching Cubes: Artifacts



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)

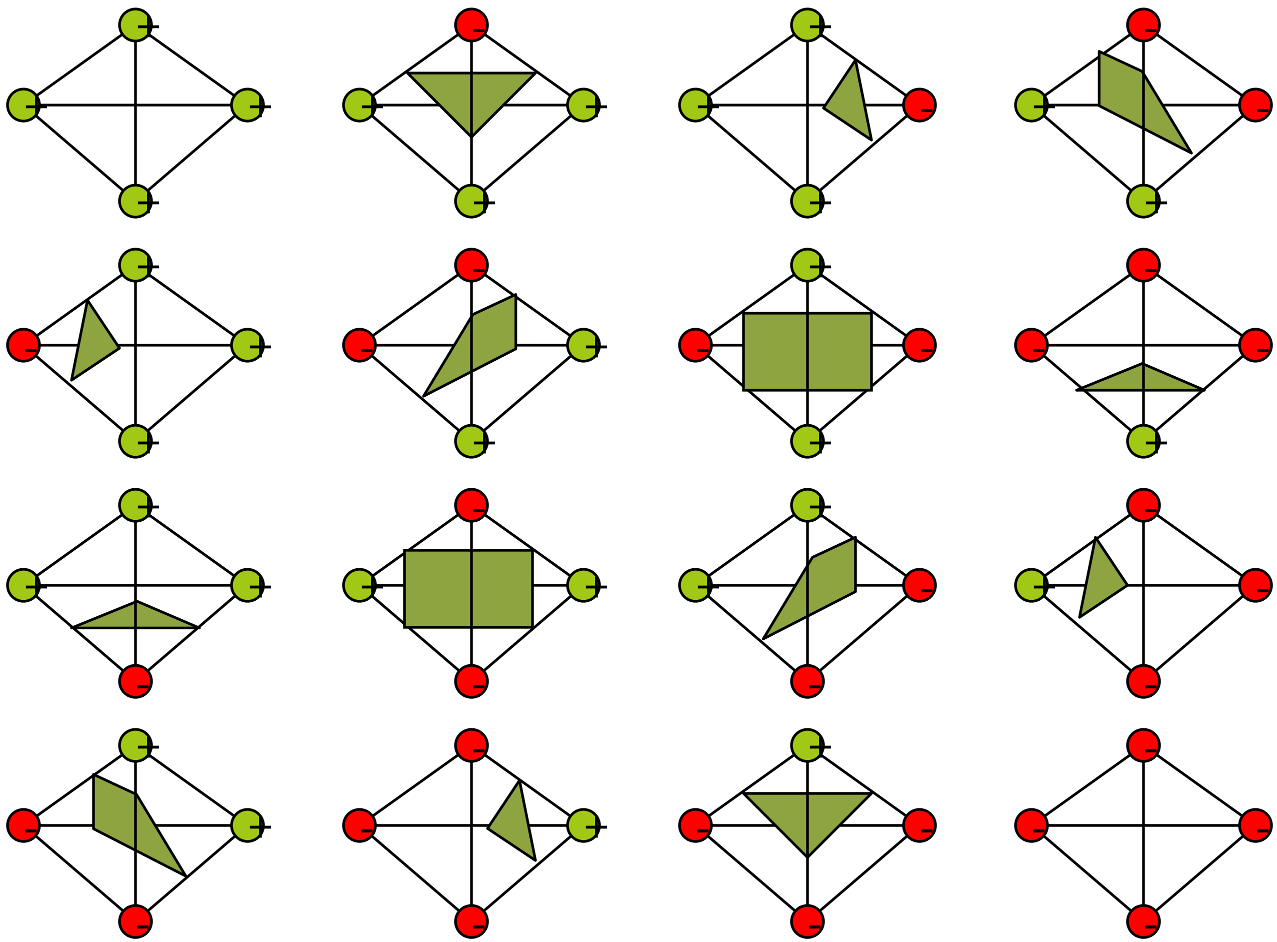
E. C. LaMar, B. Hamann, K. Joy, High-Quality Rendering of Smooth Isosurfaces, JVCA vol. 10, 1999, 79-90

Marching Cubes: Scalability to Big Data Sets

- Running time on a grid with **n** cells?
- How much data must be kept in memory?
- Suitability for parallel processing?

Marching Tet Cases

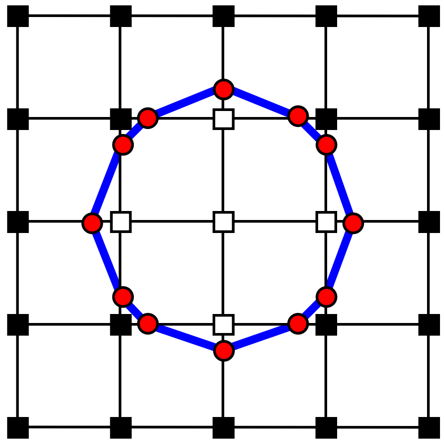
16 in all
3 modulo symmetry



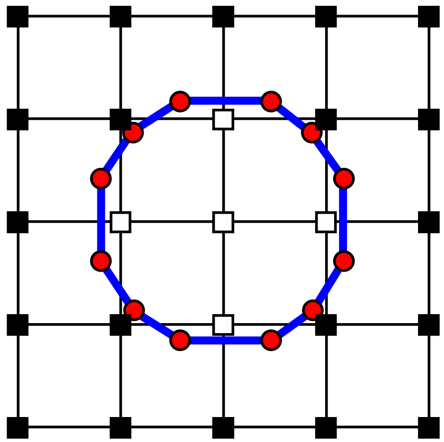
Marching Tetrahedra: Advantages/Disadvantages

Alternative Algorithms

- Primal methods
  - Marching Squares (2D), Marching Cubes (3D)
  - Placing vertices on grid edges

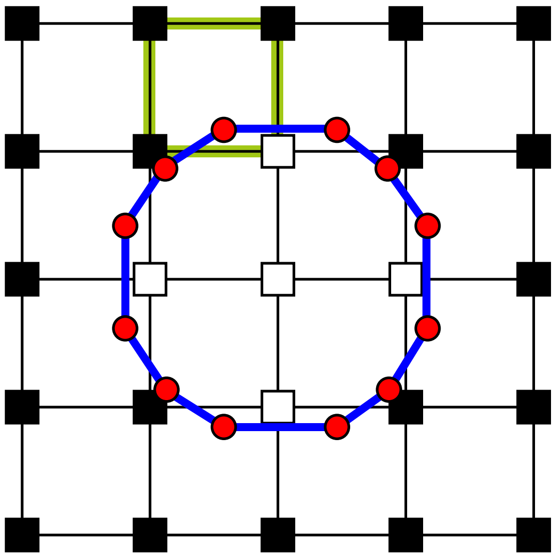


- Dual methods
  - Dual Contouring (2D,3D)
  - Placing vertices in grid cells



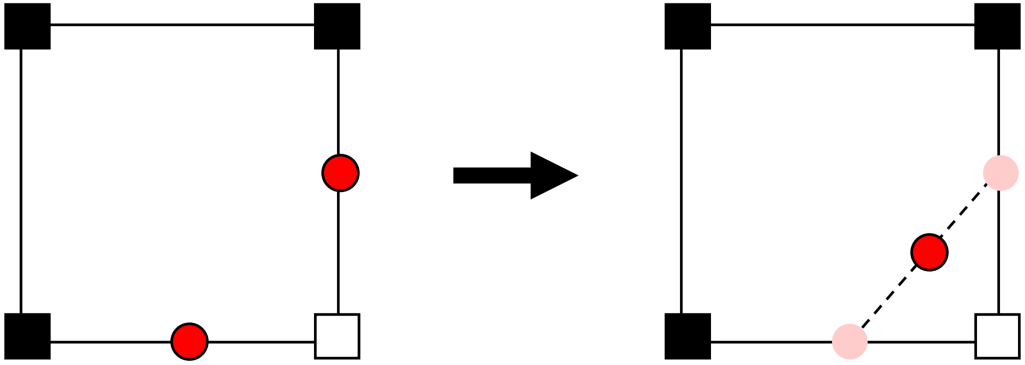
Dual Contouring (2D)

- For each grid cell with a sign change
  - Create one vertex
- For each grid edge with a sign change
  - Connect the two vertices in the adjacent cells with a line



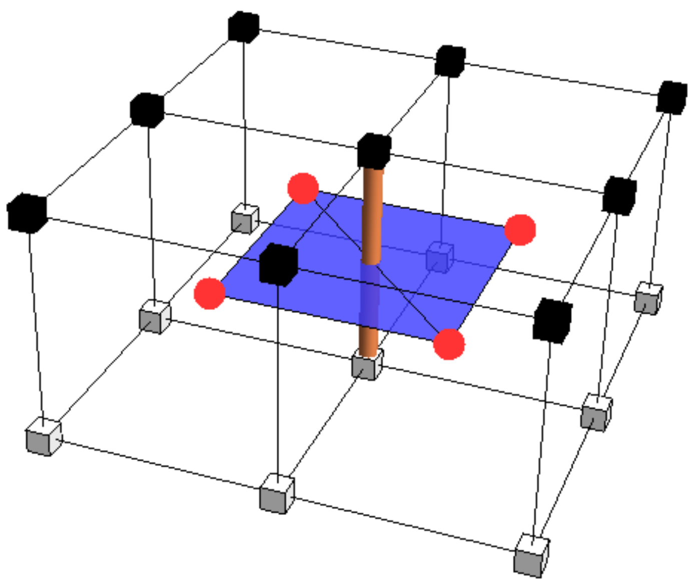
Dual Contouring (2D)

- Creating the vertex within a cell
  - Compute one point on each grid edge with a sign change (by linear interpolation, as in Marching Squares/Cubes)
    - There could be more than two sign-changing edges, so >2 points possible
  - Take the centroid of these points



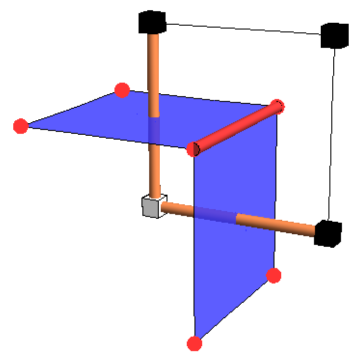
Dual Contouring (3D)

- For each grid cell with a sign change
  - Create one vertex (same way as 2D)
- For each grid edge with a sign change
  - Create a quad (or two triangles) connecting the four vertices in the adjacent grid cubes
  - No look-up table is needed!

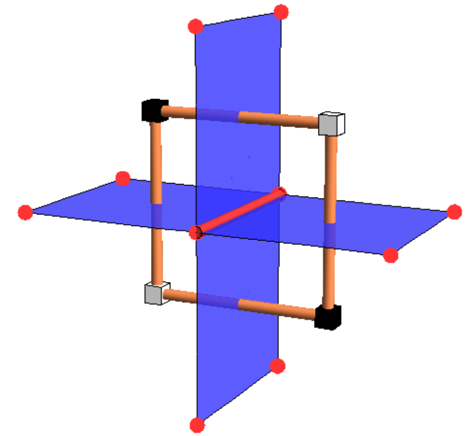


Dual Contouring: Discussion

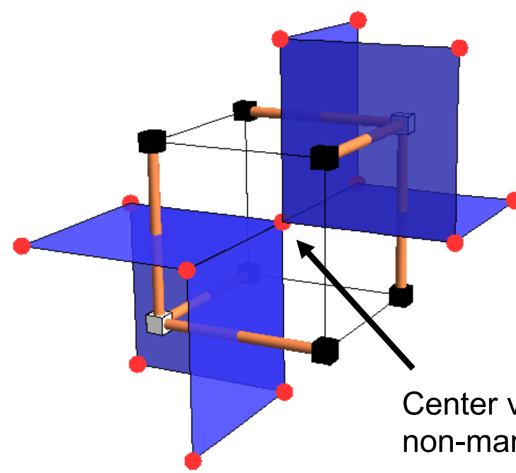
- Result is closed, but possibly non-manifold
  - Each mesh edge is shared by even number of quads
  - An edge may be shared by 4 quads
  - A vertex may be shared by 2 rings of quads
- Can be fixed
  - But with more effort…



Red edge is shared by 2 quads



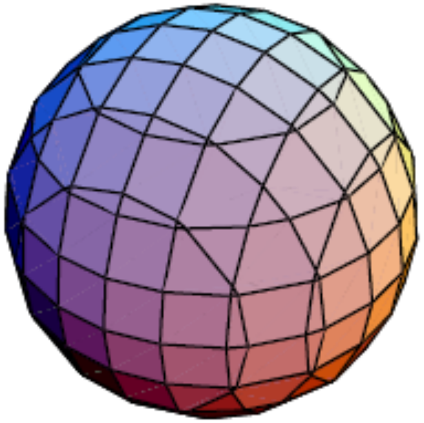
Red edge is shared by 4 quads (non-manifold)

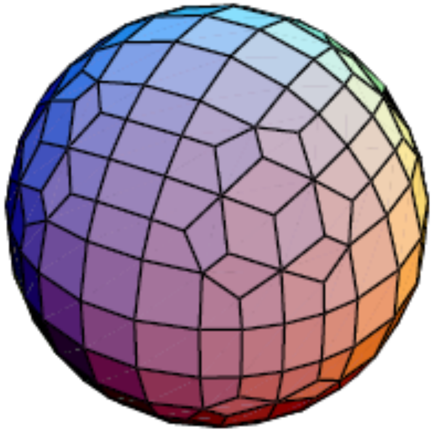


Center vertex is non-manifold

MC vs. DC: Mesh quality

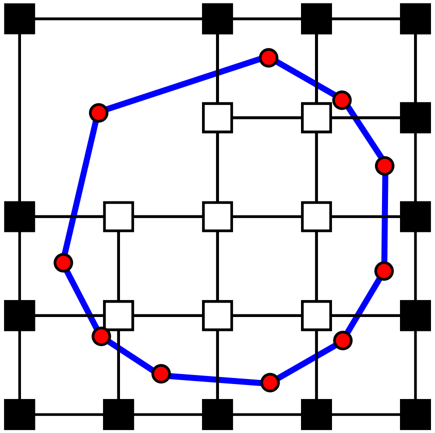
- Marching Cubes
  - ✓ Closed, manifold, and intersection-free
  - ✗ Often generates thin and tiny polygons
- Dual Contouring
  - ✓ Closed and intersection-free
  - ✓ Generates better-shaped polygons
  - ✗ Can be non-manifold


Marching Cubes

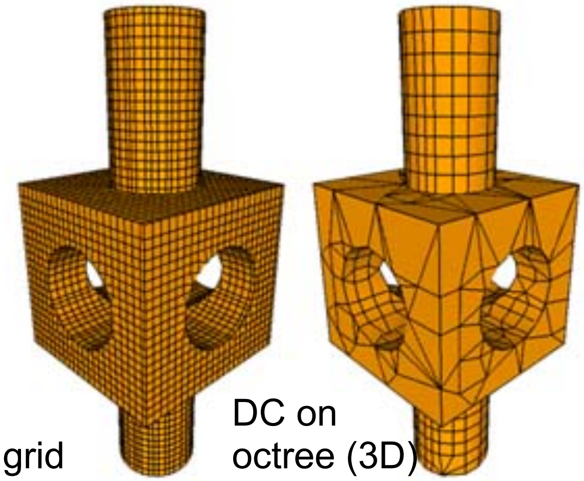

Dual Contouring

MC vs. DC: Implementation

- Marching Cubes
  - ✗ Creating the triangulation table is non-trivial
    - Although table-lookup is straight forward
  - ✗ Restricted to uniform grids
- Dual Contouring
  - ✓ Simple to implement
    - No look-up table is needed
  - ✓ Can be applied to any type of grid
    - Good for generating anisotropic meshes



DC on a Quadtree (2D)



DC on uniform grid

DC on octree (3D)

References

- Marching Cubes:
  - "*Marching cubes: A high resolution 3D surface construction algorithm*", by Lorensen and Cline (1987)
    - >6000 citations on Google Scholar
  - "*A survey of the marching cubes algorithm*", by Newman and Yi (2006)

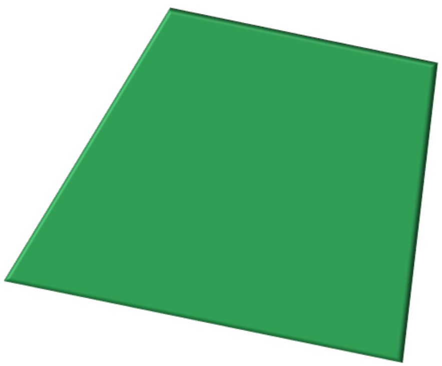
- Dual Contouring:
  - "*Dual contouring of hermite data*", by Ju et al. (2002)
    - >300 citations on Google Scholar
  - "*Manifold dual contouring*", by Schaefer et al. (2007)

Displacement Plots

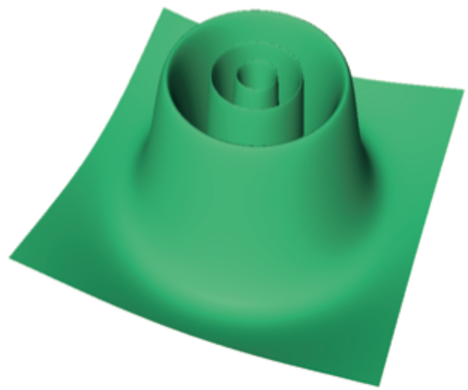
Displace a given surface $S \subseteq D$ in the direction of its normal

Displacement value encodes the scalar data f

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



input surface S

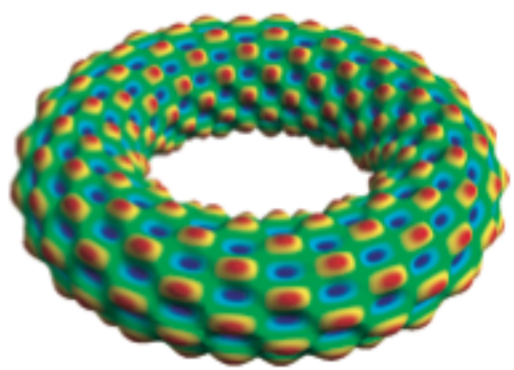


displaced surface S_{displ}

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