2

Contouring and Isosurfaces

What are contours?

Set of points where the scalar field *s* has a given value *c*:

$$\left\{ \mathbf{x} \in \mathbb{R}^n : s\left(\mathbf{x}\right) = c \right\}$$

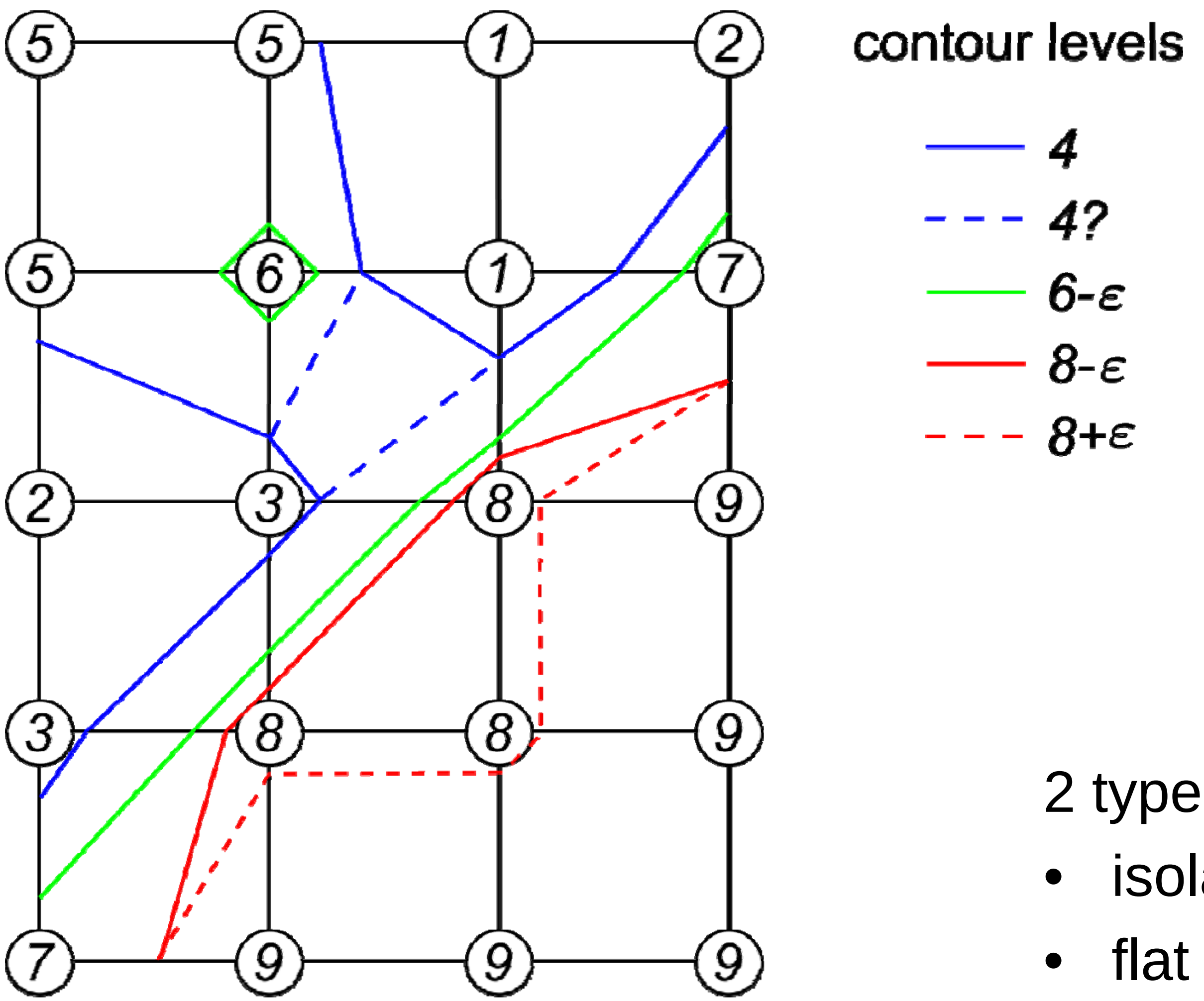
Examples in 2D:

- height contours on maps
- isobars on weather maps

Contouring algorithm:

- find intersection with grid edges
- connect points in each cell

Example



2 types of degeneracies:

- isolated points (*c*=6)
- flat regions (*c*=8)

Topological consistency

To avoid degeneracies, use symbolic perturbations:

If level *c* is found as a node value, set the level to $c-\varepsilon$ where ε is a symbolic infinitesimal.

Then:

- contours intersect edges at some (possibly infinitesimal) distance from end points
- flat regions can be visualized by pair of contours at $c-\varepsilon$ and $c+\varepsilon$
- contours are topologically consistent, meaning:

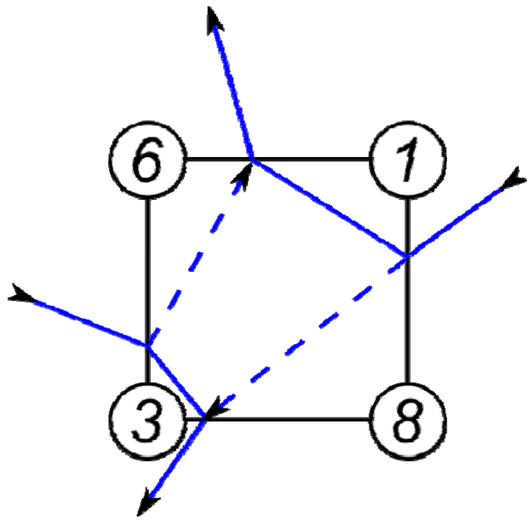
Contours are closed, orientable, nonintersecting lines.

Ambiguities of contours

What is the correct contour of c=4?

Two possibilities, both are orientable:

- values $s(\mathbf{x})>c$ are on the left side
- values $s(\mathbf{x})<c$ are on the right side



Answer: correctness depends on interior values of $s(\mathbf{x})$.

But different interpolation schemes are possible.

Better question: What is the correct contour with respect to bilinear interpolation?

Contours in a quadrangle cell

- local coordinates: $(0,0),\ (1,0),\ (0,1),\ (1,1)$
- function values: $s_{00}\ ,\ s_{10}\ ,\ s_{01}\ ,\ s_{11}$
- bilinear interpolant:

$$s = (1-x)(1-y)s_{00} + x(1-y)s_{10} + (1-x)y\,s_{01} + x\,y\,s_{11}$$

$$= Axy + Bx + Cy + D$$

If *A*=0, contour equation is $c = Bx + Cy + D$

contours are straight lines, all parallel

If *A*≠0, contour equation is $c = A\left(x + \frac{C}{A}\right)\left(y + \frac{B}{A}\right) + D - \frac{BC}{A}$

contours are hyperbola, except for level $c = D - \frac{BC}{A}$

Contour equation for special level: $$0 = A\left(x + \frac{C}{A}\right)\left(y + \frac{B}{A}\right)$$

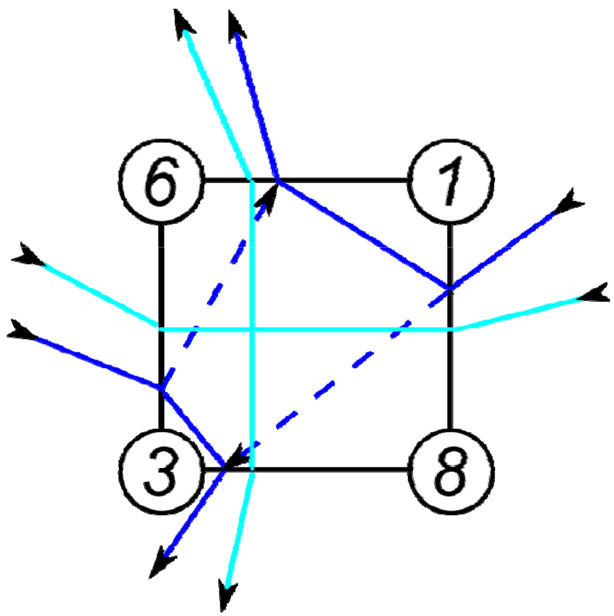
Contour is a pair of axis-aligned straight lines $x = -C/A$ and $y = -B/A$.

Applied to example:

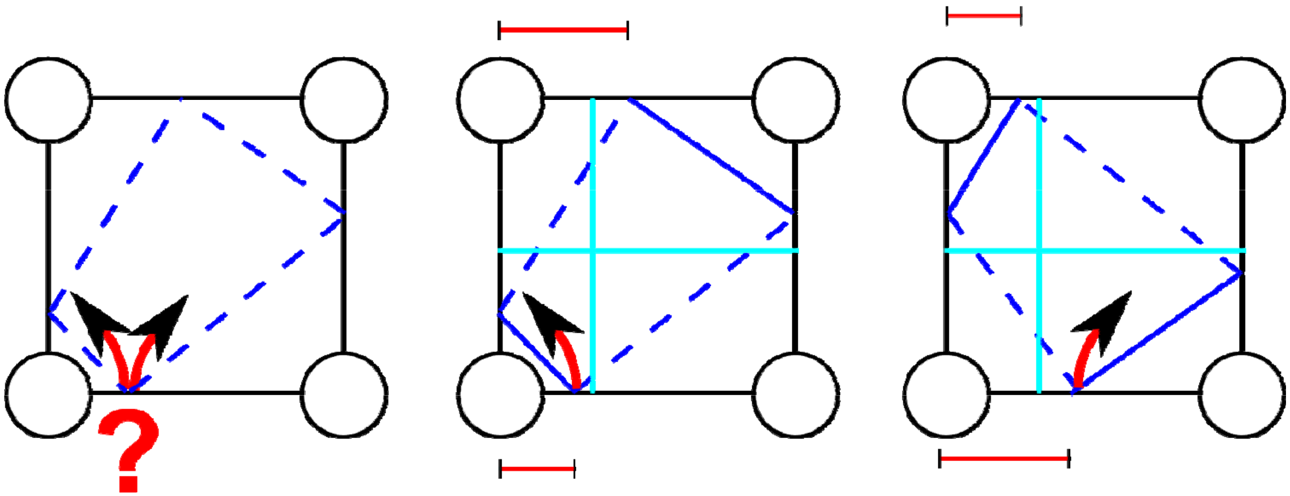
- contour equation:

$$c = -10\left(x - 0.3\right)\left(y - 0.5\right) + 4.5$$

- special level c=4.5
- saddle point at (0.3, 0.5)



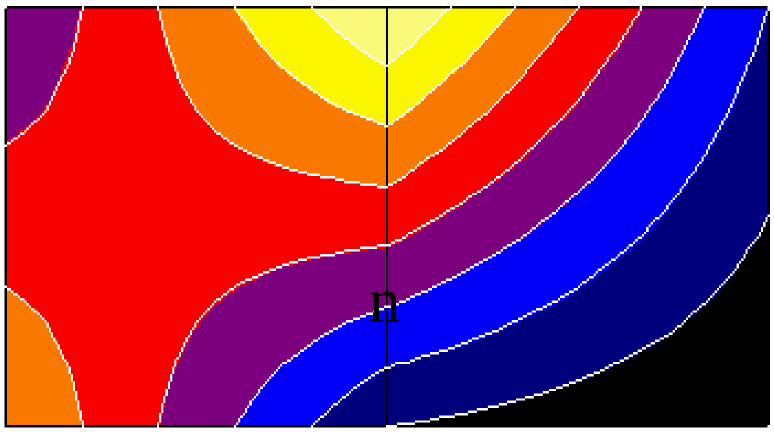
Decision can be made without computing special level or saddle point, by comparing fractions of edges:



Using local coordinates, this works also for curvilinear and unstructured grids.

Note: For drawing, straight lines are sufficient.

Drawing hyperbola does not lead to better contours:



Reason: piecewise bilinear function is not C^1.

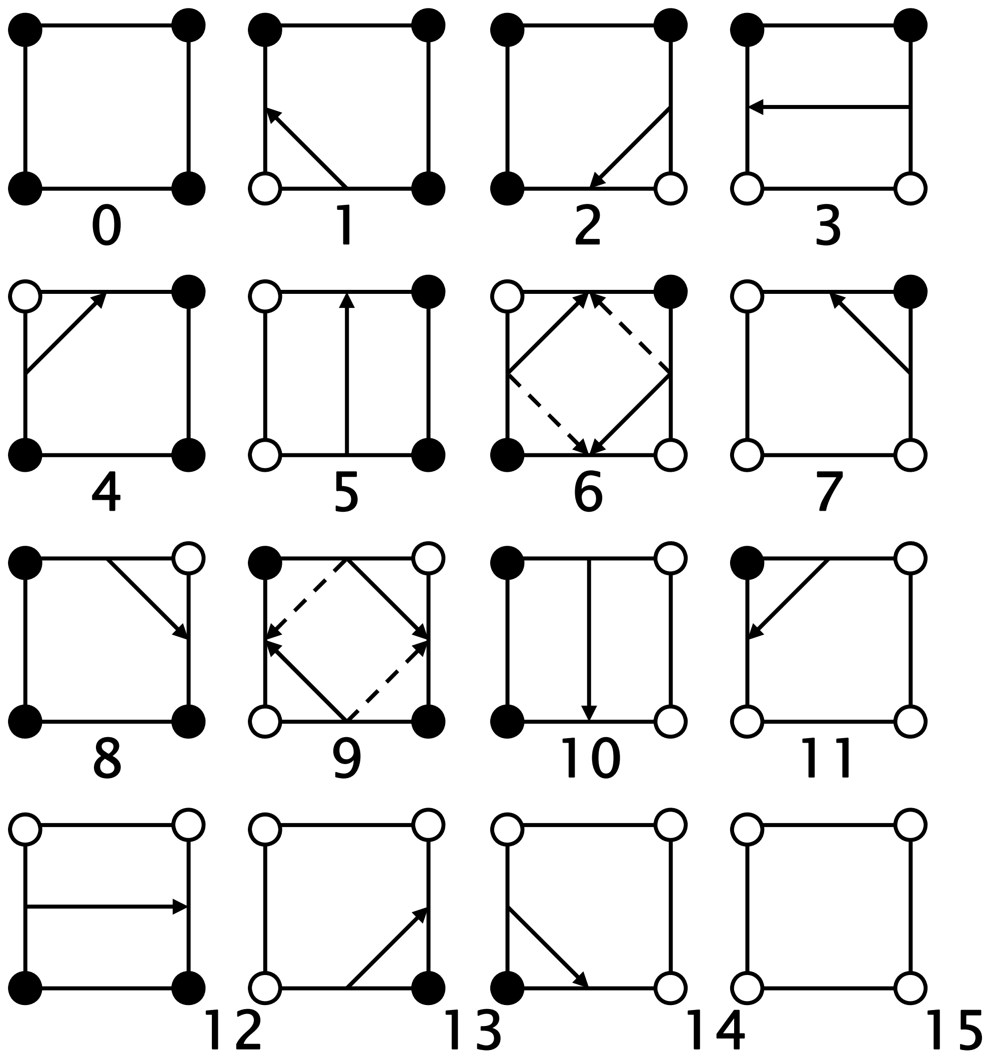
Basic contouring algorithms:

- cell-by-cell algorithms: simple structure, but generate disconnected segments, require post-processing
- contour propagation methods: more complicated, but generate connected contours

"Marching squares" algorithm (systematic cell-by-cell):

- process nodes in ccw order, denoted here as $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$
- compute at each node $\mathbf{x}_i$ the reduced field $\tilde{s}(\mathbf{x}_i) = s(\mathbf{x}_i) - (c - \varepsilon)$ (which is forced to be nonzero)
- take its sign as the i^{th} bit of a 4-bit integer
- use this as an index for lookup table containing the connectivity information:

Contours in a quadrangle cell



● $\tilde{s}(\mathbf{x}_i) < 0$

○ $\tilde{s}(\mathbf{x}_i) > 0$

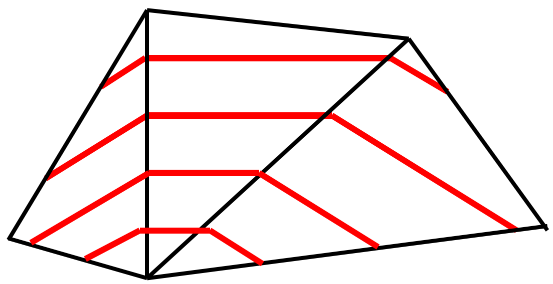
Alternating signs exist in cases 6 and 9.

Choose the solid or dashed line?

Both are possible for topological consistency.

This allows to have a fixed table of 16 cases.

Contours in triangle/tetrahedral cells



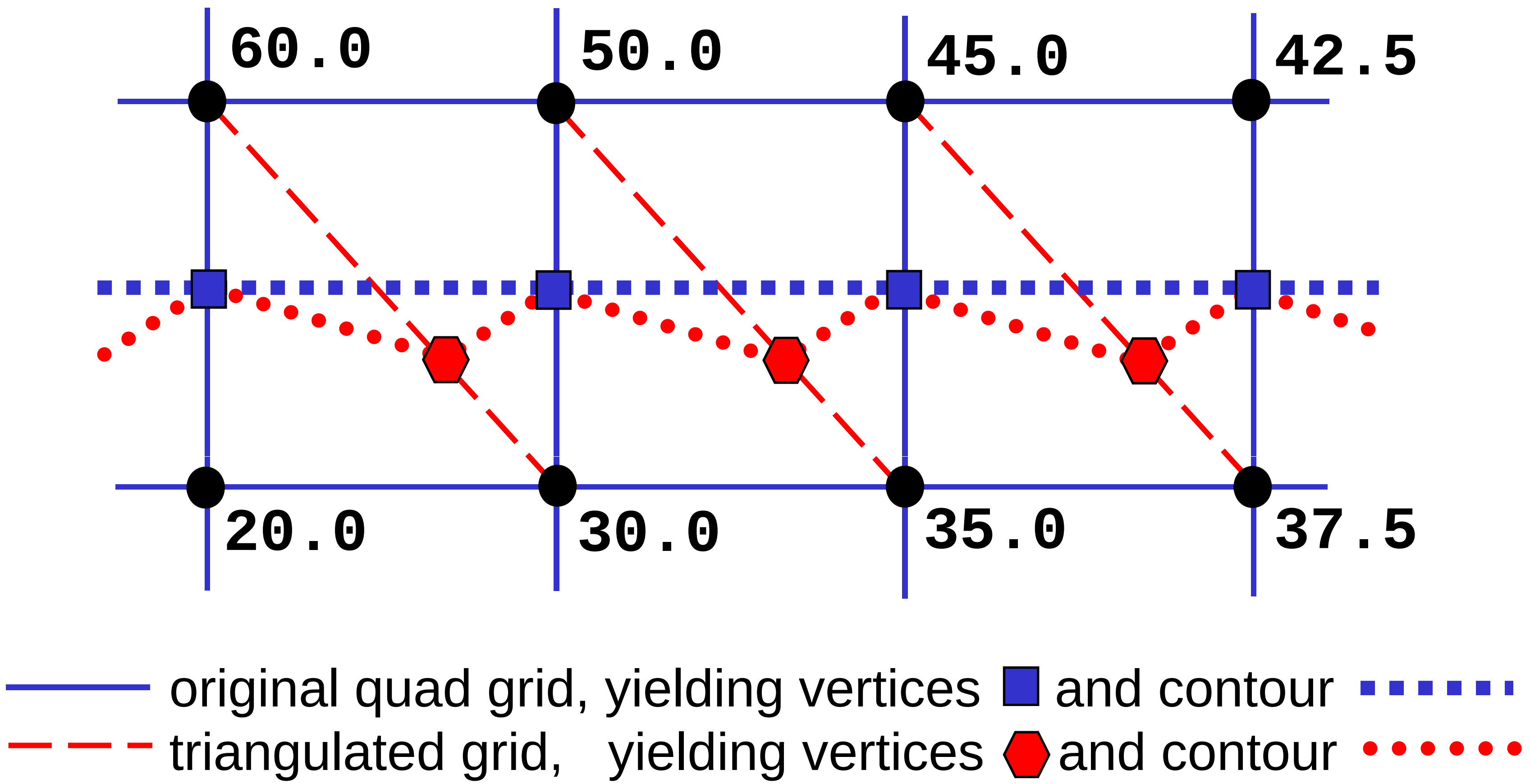
Linear interpolation of cells implies piece-wise linear contours.

Contours are unambiguous, making "marching triangles" even simpler than "marching squares".

Question: Why not split quadrangles into two triangles (and hexahedra into five or six tetrahedra) and use marching triangles (tetrahedra)?

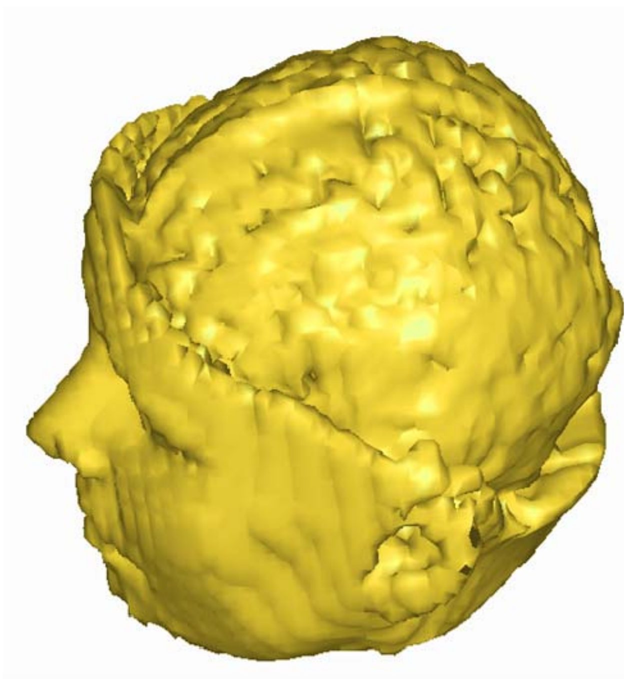
Answer: This can introduce periodic artifacts!

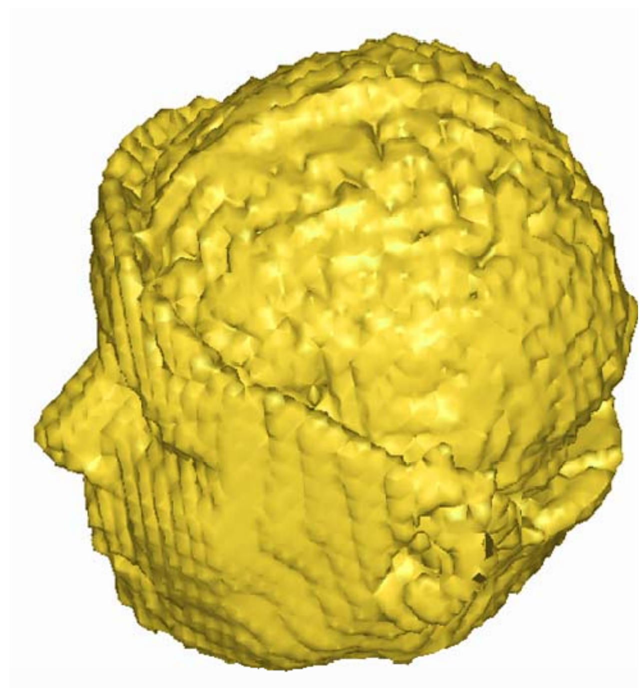
Illustrative example: Find contour at level *c*=40.0 !



3D example based on real (downsampled) dataset.
Contour (=isosurface) in

original hexahedral grid vs. in tetrahedrized grid:





The marching cubes algorithm

Contours of 3D scalar fields are known as isosurfaces.

Before 1987, isosurfaces were computed as

- contours on planar slices, followed by
- "contour stitching".

The marching cubes algorithm computes contours directly in 3D.

- Pieces of the isosurfaces are generated on a cell-by-cell basis.
- Similar to marching squares, a 8-bit number is computed from the 8 signs of $\tilde{s}(\mathbf{x}_i)$ on the corners of a hexahedral cell.
- The isosurface piece is looked up in a table with 256 entries.

How to build up the table of 256 cases?

Lorensen and Cline (1987) exploited 3 types of symmetries:

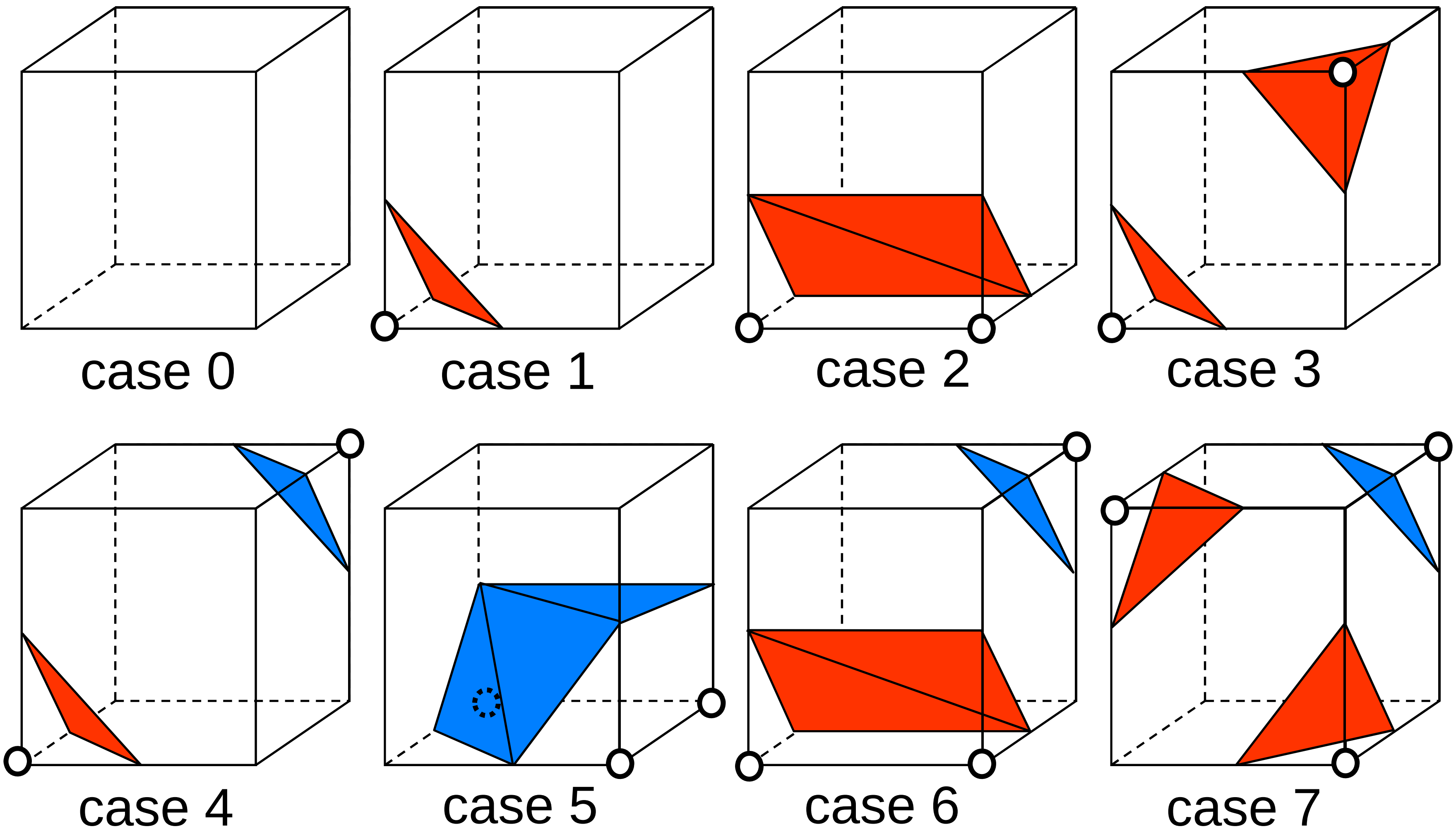
- rotational symmetries of the cube
- reflective symmetries of the cube
- sign changes of $\tilde{s}(\mathbf{x})$

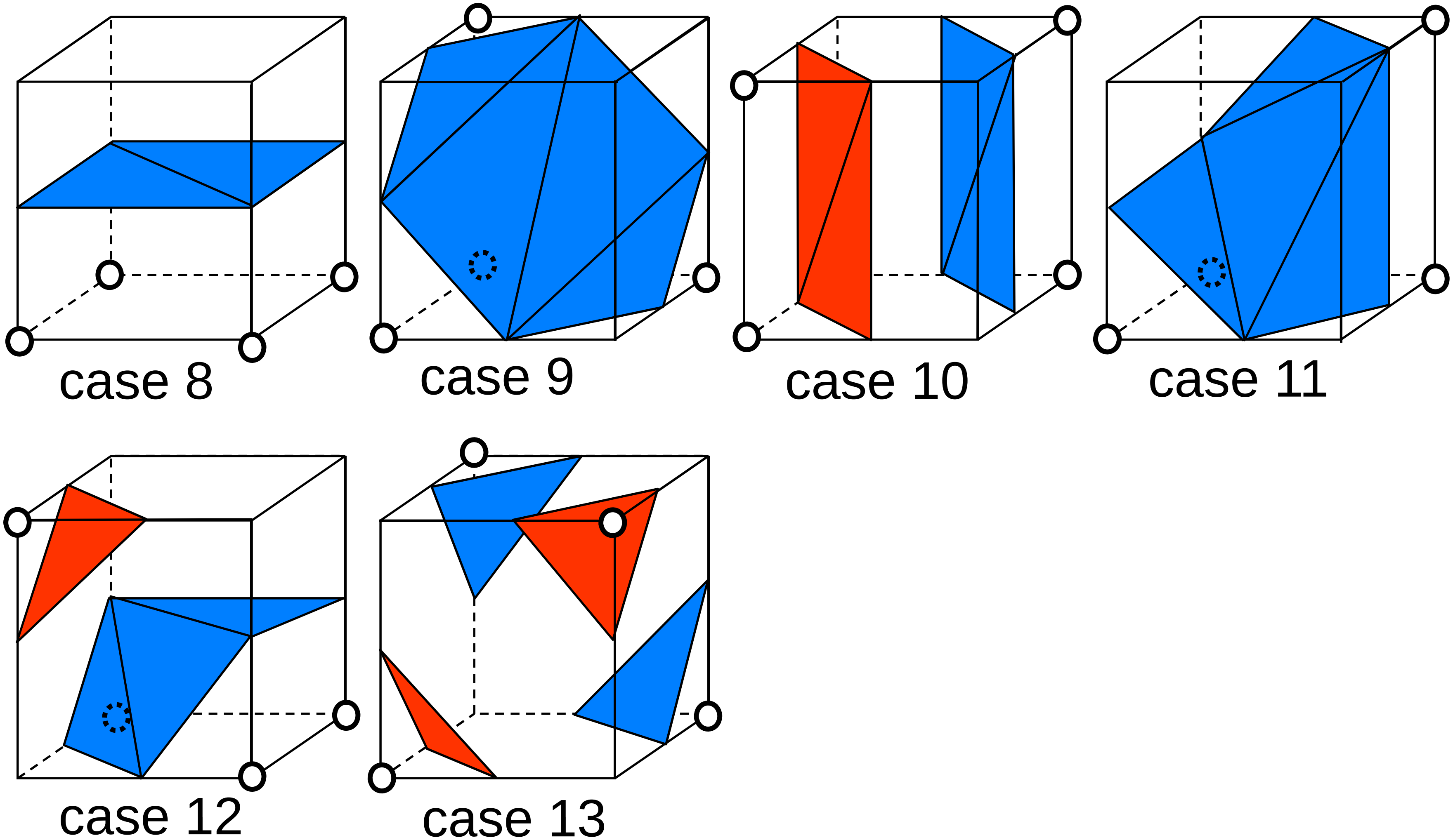
They published a reduced set of 14*) cases shown on the next slides where

- white circles indicate positive signs of $\tilde{s}(\mathbf{x})$
- the positive side of the isosurface is drawn in red, the negative side in blue.

*) plus an unnecessary "case 14" which is a symmetric image of case 11.

The marching cubes algorithm




case 8
case 9
case 10
case 11
case 12
case 13

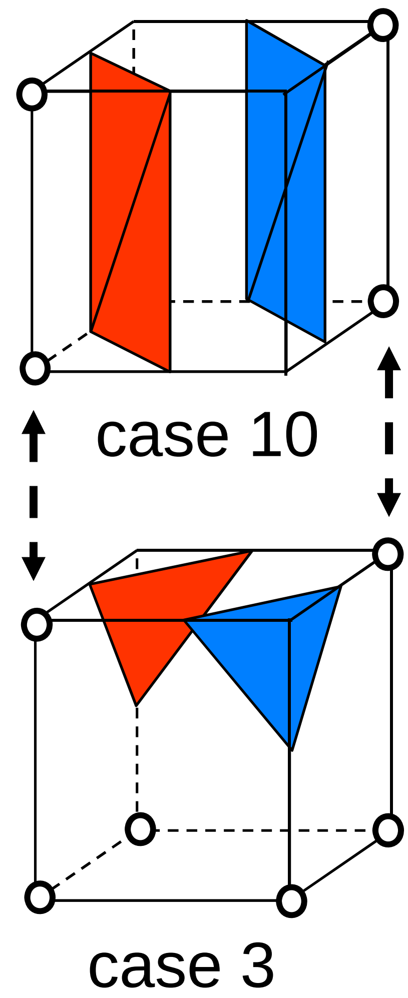
Do the pieces fit together?

- The correct isosurfaces of the trilinear interpolant would fit (trilinear reduces to bilinear on the cell interfaces)
- but the marching cubes polygons don't necessarily fit.

Example

- case 10, on top of
- case 3 (rotated, signs changed)

have matching signs at nodes but polygons don't fit.

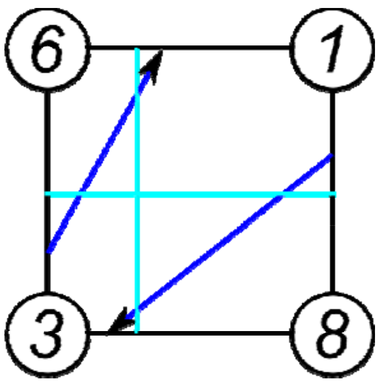


Reason for failure:

Topology decision on faces with alternating signs.

Decision by original MC algorithm is not correct w.r.t. the interpolant, and not consistent.

A consistent decision would be: always cut off the positive corners!

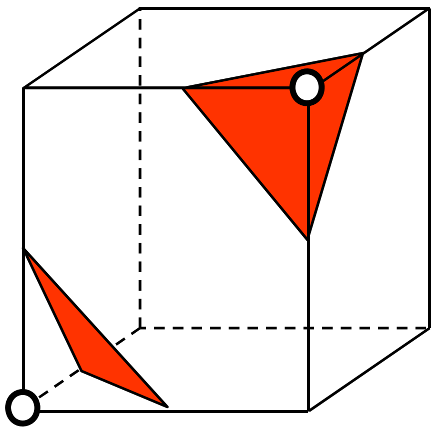


Original MC table obeys this rule, but:

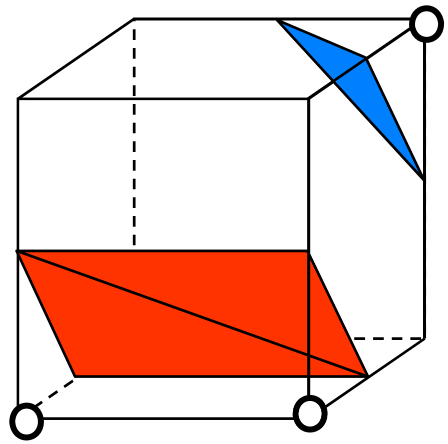
It is lost when sign change is applied!

Consequence:

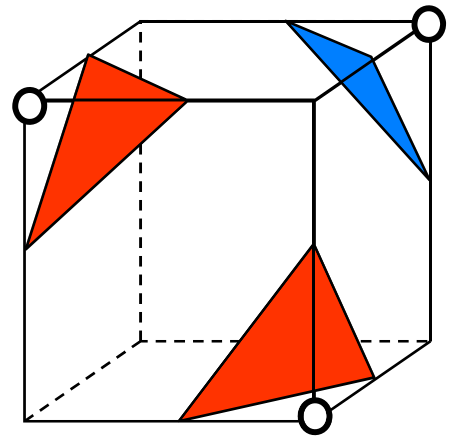
Extend table by 14 complementary cases for changed signs!



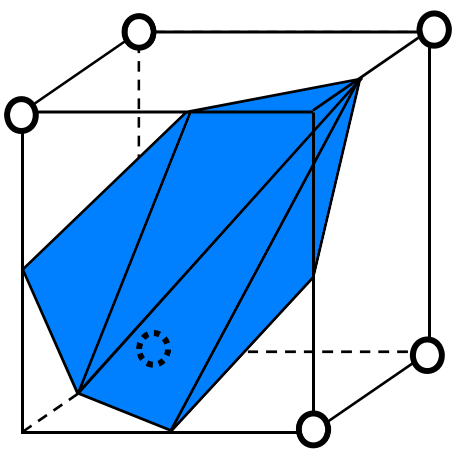
case 3



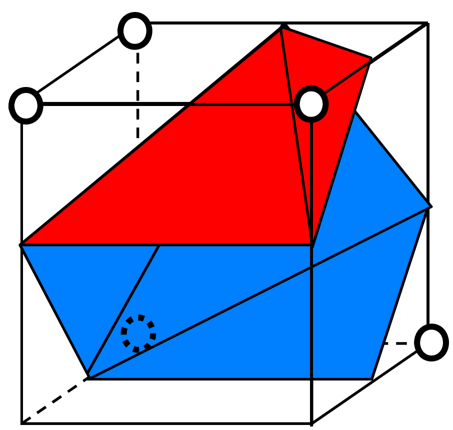
case 6



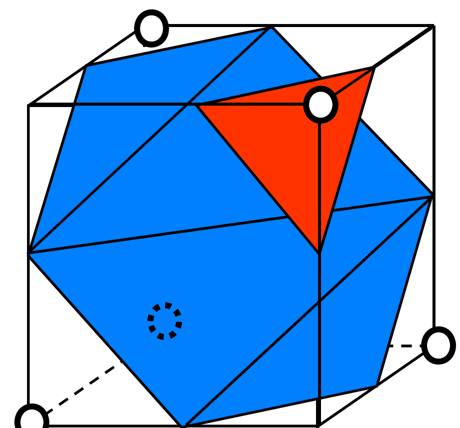
case 7



case 3c



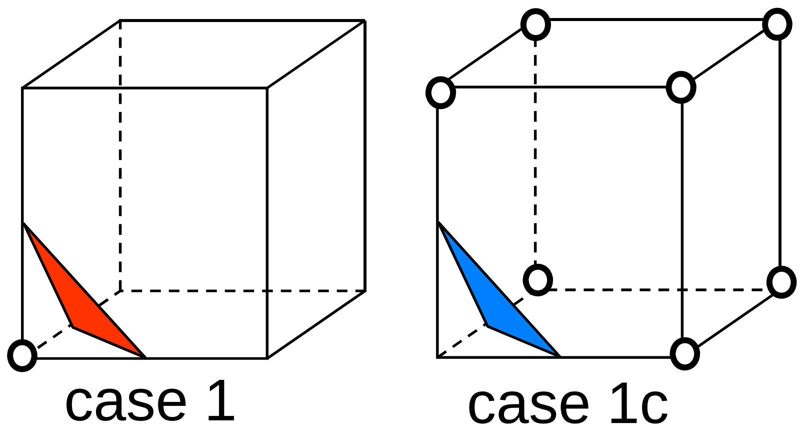
case 6c



case 7c

The remaining complementary cases are obtained simply by changing the orientation.

Example:



Based on the 28 cases, the full 256 cases are obtained by

- rotations of the cube
- reflections of the cube (and re-orienting of triangles)

Summary of marching cubes algorithm:

Pre-processing steps:

- build a table of the 28 cases
- derive a table of the 256 cases, containing info on
  - intersected cell edges, e.g. for case 3/256 (see case 2/28): (0,2), (0,4), (1,3), (1,5)
  - triangles based on these points, e.g. for case 3/256: (0,2,1), (1,3,2).

Loop over cells:

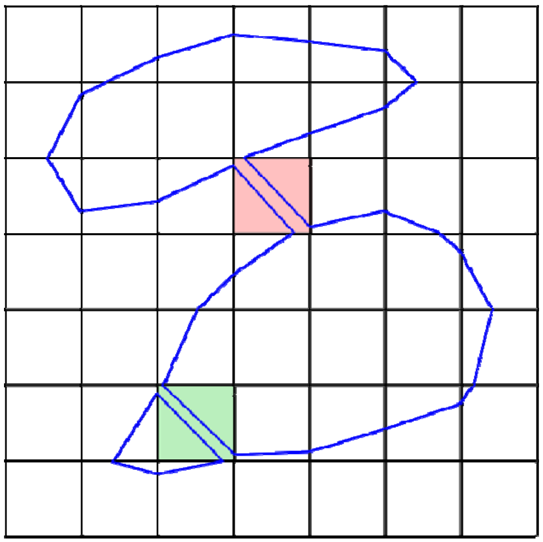
- find sign of $\tilde{s}(\mathbf{x})$ for the 8 corner nodes, giving 8-bit integer
- use as index into (256 case) table
- find intersection points on edges listed in table, using linear interpolation
- generate triangles according to table

Post-processing steps:

- connect triangles (share vertices)
- compute normal vectors
  - by averaging triangle normals (problem: thin triangles!)
  - by estimating the gradient of the field $s(\mathbf{x})$ (better)

The asymptotic decider algorithm

Motivation for a different isosurface algorithm:



Marching cubes can produce "bad" topology.

2D example (marching squares):

Asymptotic decider algorithm (Nielson and Hamann 1991) :

- generate topologically correct contours (as oriented straight line segments) on the cell interfaces
- connect these around the cell, resulting in one or more polygons
- triangulate the polygons

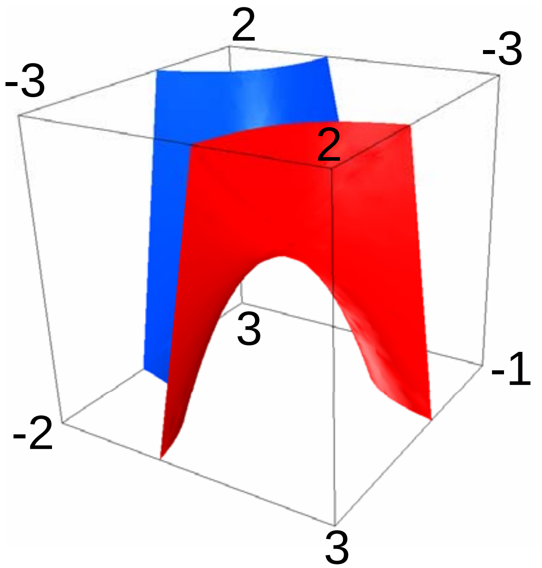
~/avs/networks/SciVis/MCandAD*.net

In general, the AD algorithm generates better isosurfaces.

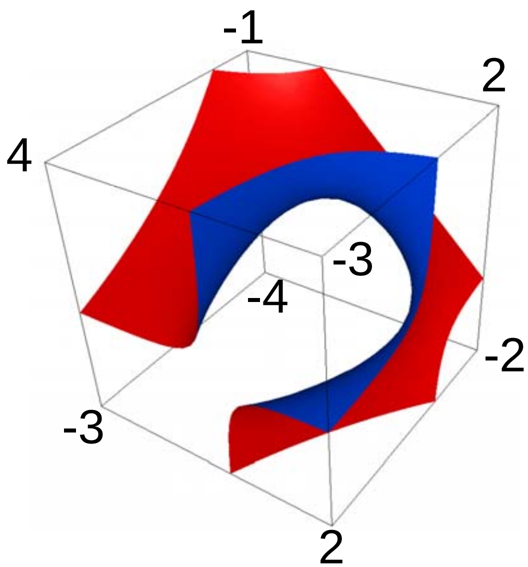
However,

- it cannot be easily implemented with a table like MC (too many cases)
- it generates polygons with up to 12 sides (MC: up to 7)
- the topology is correct w.r.t the trilinear interpolant, but the geometry can deviate
- some polygons cannot be "cleanly" triangulated

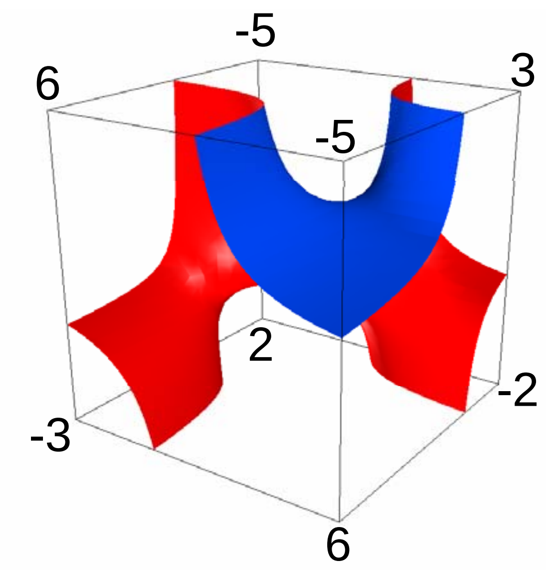
A few examples are given on the next slide, showing isosurfaces of the trilinear interpolant.



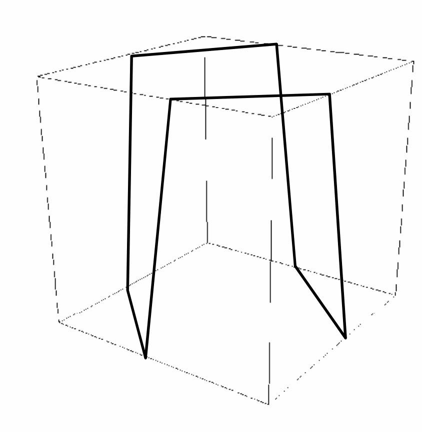
8-sided polygon



9-sided polygon



12-sided polygon



The 8-sided polygon has no valid triangulation!

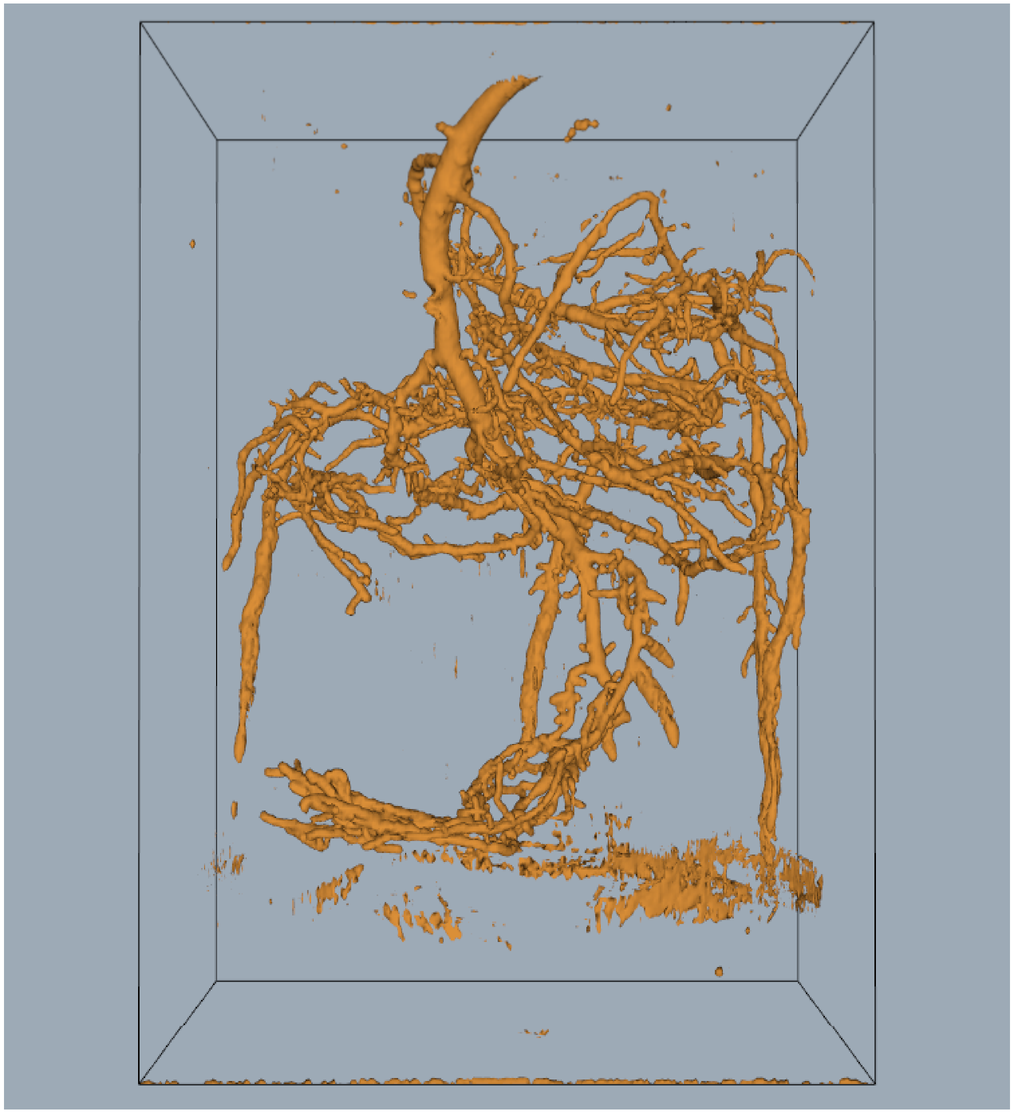
- either some triangles lie on faces of the cell
- or an extra vertex has to be used

~/avs/networks/SciVis/AD*net

Post-processing of isosurfaces

Example (VTK demo):
pine root dataset

(1) unprocessed
MC isosurface

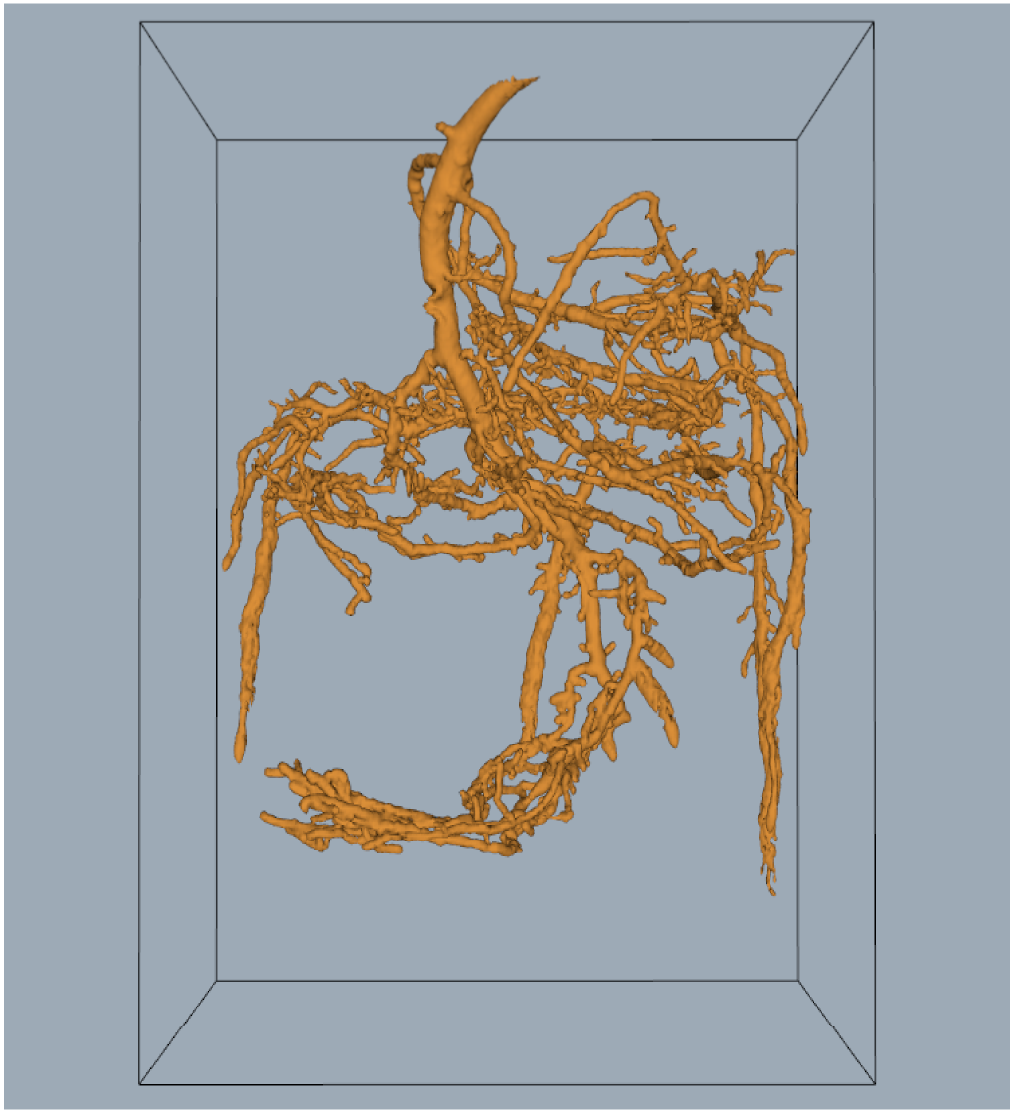


Data: J. McFall, Center for In Vivo Microscopy, Duke University

Example (VTK demo): pine root dataset

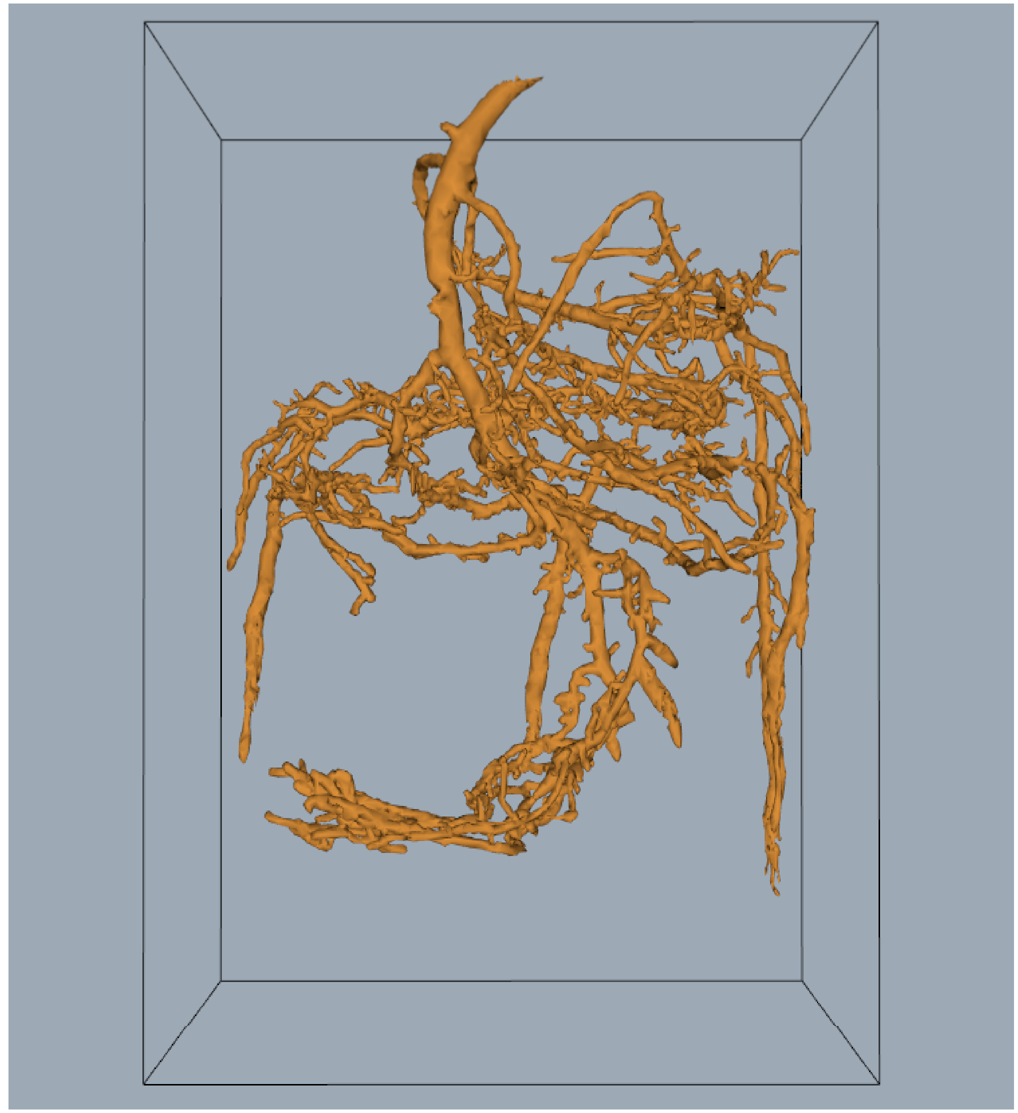
(2) largest connected component only

Algorithm: connected component labeling



Example (VTK demo): pine root dataset

(3) decimated from
351,118 to
81,111 triangles



Purpose of decimation:

- data reduction
- improve mesh quality (thin/small triangles)

Algorithm (Schroeder):

- vertex removal
- feature edges kept

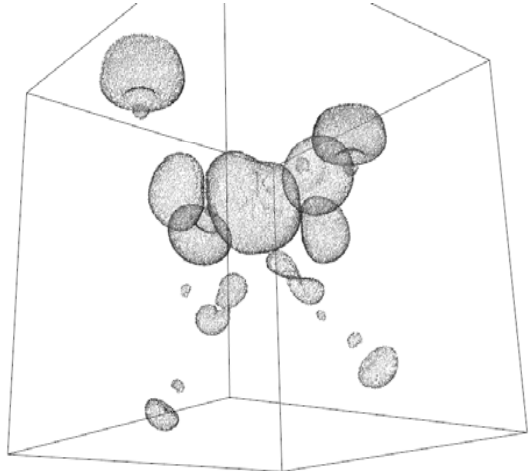
The dividing cubes algorithm

An early point-based algorithm (Crawford et al. '87): For each cell

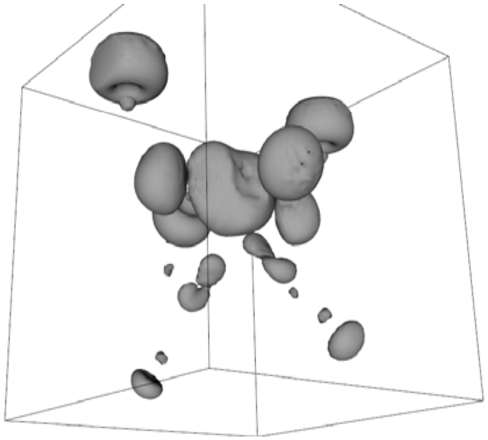
- check whether it is intersected by the isosurface:
  $\min_{i \in cell} s_i < c < \max_{i \in cell} s_i$
- subdivide intersected cell into $m \times m \times m$ subcells using trilinear interpolation
- draw the centers of all intersected subcells

Points can be lit:

- estimate the gradient and use it as the normal vector



50’078 and
2’506’989 points



Optimized isosurface algorithms

Approaches to speeding up isosurface computation:

View dependent algorithms

- occluded triangles not computed
- GPU-based isosurface computation and rendering

Data preprocessing for fast computation of multiple isosurfaces (multiple levels), e.g. for interactive exploration of the data.

- many methods: octree, extrema graph, span space
- common goal: avoid computation in non-intersected cells.

The octree-based algorithm

Method by Wilhelms and van Gelder (1992) for (block-)structured grids.

Pre-processing:

- recursively split the grid in two subgrids, building up a binary tree of subgrids, stop splitting when single cells are reached.
- compute minimum and maximum of $s(\mathbf{x})$ per subgrid, store as an interval [*min*, *max*] in the tree.

Computing the isosurface for a level c:

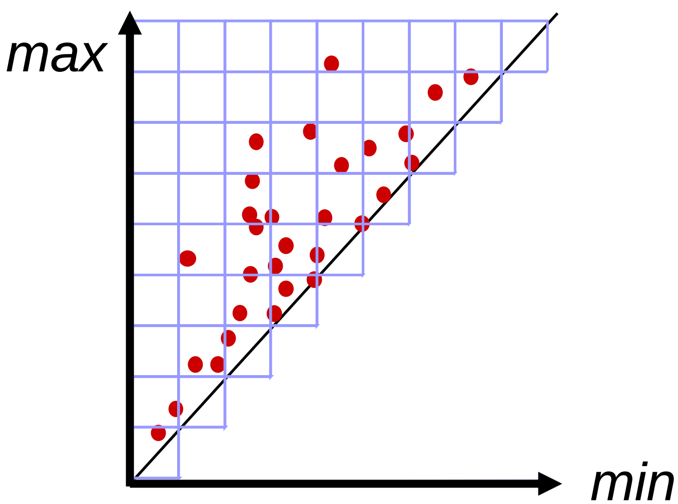
- starting at the root,
- descend recursively to subtrees if $min<c<max$
- if a leaf is reached, generate the isosurface for the respective cell with MC or AD.

The span-space algorithm

Method by Livnat (1996).

Pre-processing:

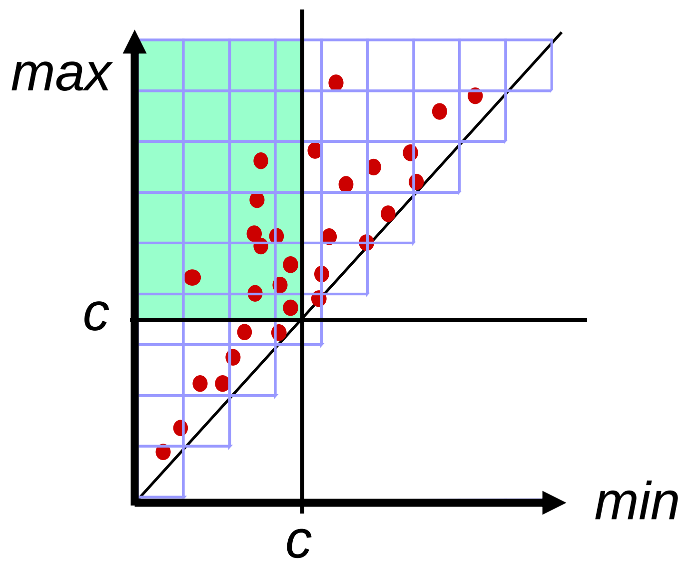
- for each cell compute *min* and *max*,
- treat (*min*,*max*) as a point in the span space (Euclidean plane)
- store points in boxes, non-empty boxes organized as linked list



Computing the isosurface for a level *c*:

- Find the intersected cells in the quadrant *min*<*c*, *max*>*c*

Performance gain for datasets with small local variation,
i.e. points in span space distributed mostly near diagonal

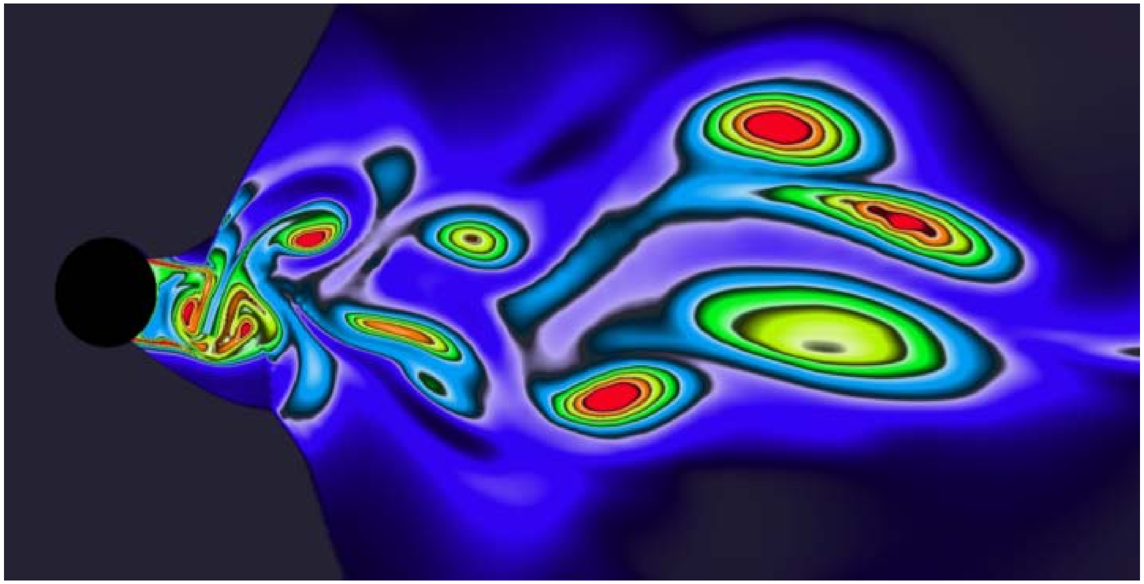


Limitations of isosurfaces

Isosurfaces represent only a single level within the data range.

In practial data, there is often not a single "interesting" level.

Example: Von Kármán vortex street, colored by entropy.



"interesting" level: red on the left, green on the right.

How should a 3D version of these data be visualized?

Limitations of isosurfaces

Transparent rendering of multiple isosurfaces is possible, but:

- limited to a small number by visibility
- alpha-blending requires depth sorting

Alternatives:

- feature extraction methods, e.g. detecting "blobs" (maximal ellipse-like contours).
- volume rendering can show ranges of "interesting" levels of the field and/or its gradient.