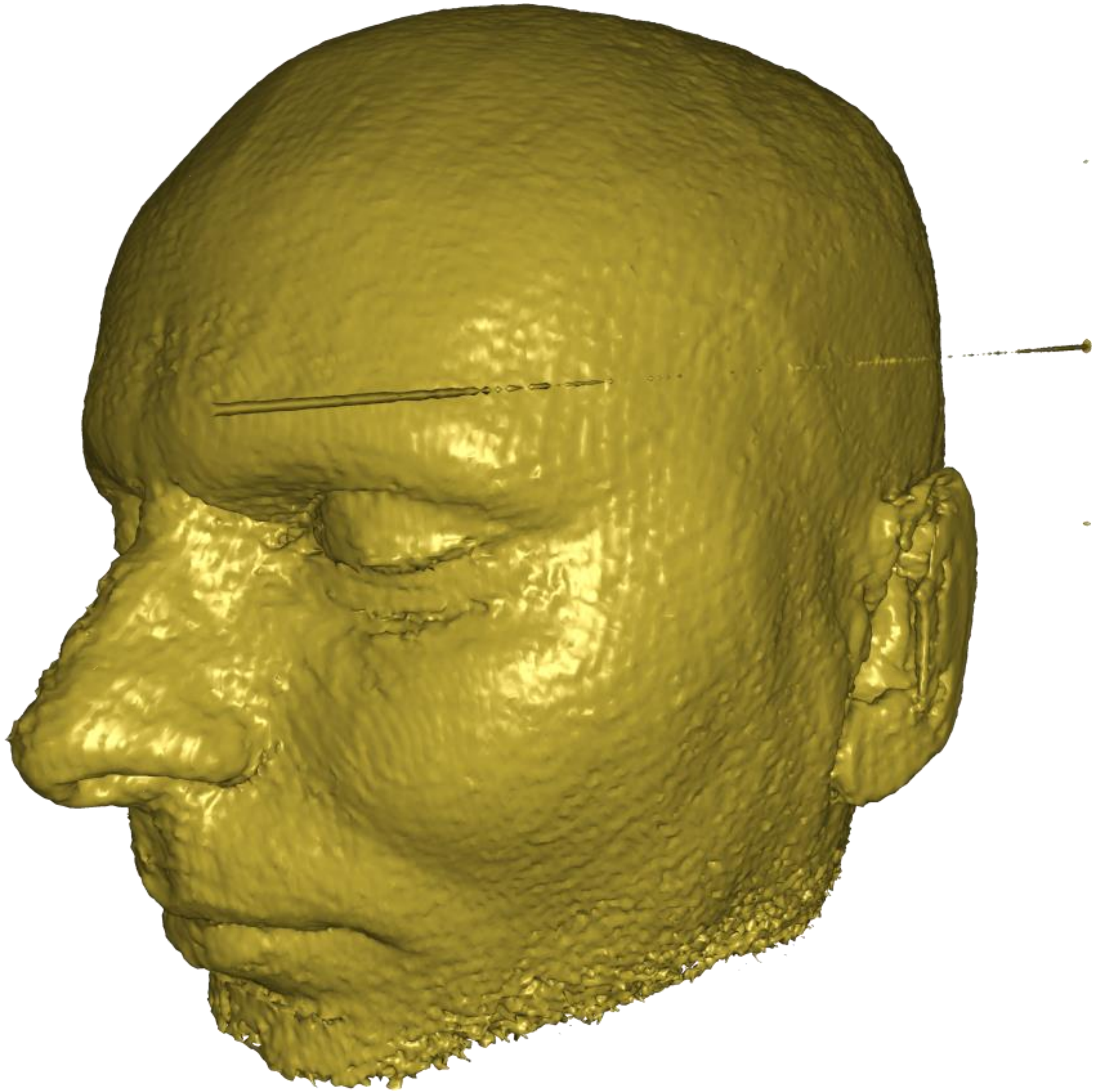
Visualization, DD2257

Prof. Dr. Tino Weinkauf

Geometry-based
Scalar Field Visualization



Function Plot

standard visualization of 1D scalar fields

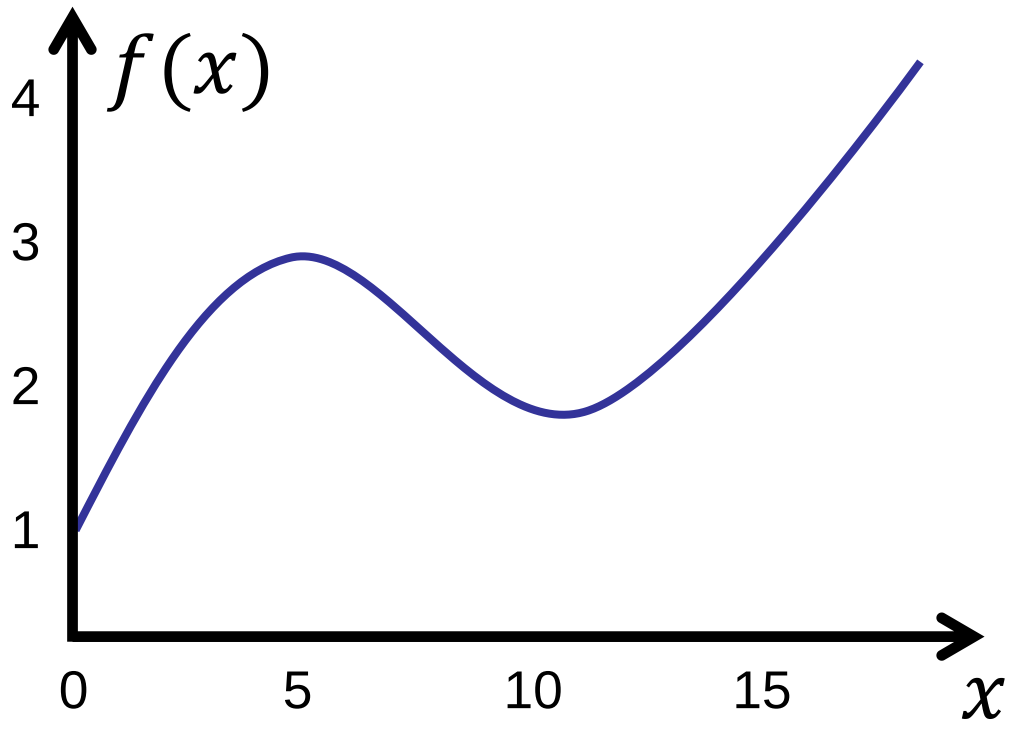
$$f : \mathbb{R} \rightarrow \mathbb{R}$$

sample function values

$\{(x, f(x)) \mid x \in \mathbb{R}\}$

connect neighboring samples

polyline



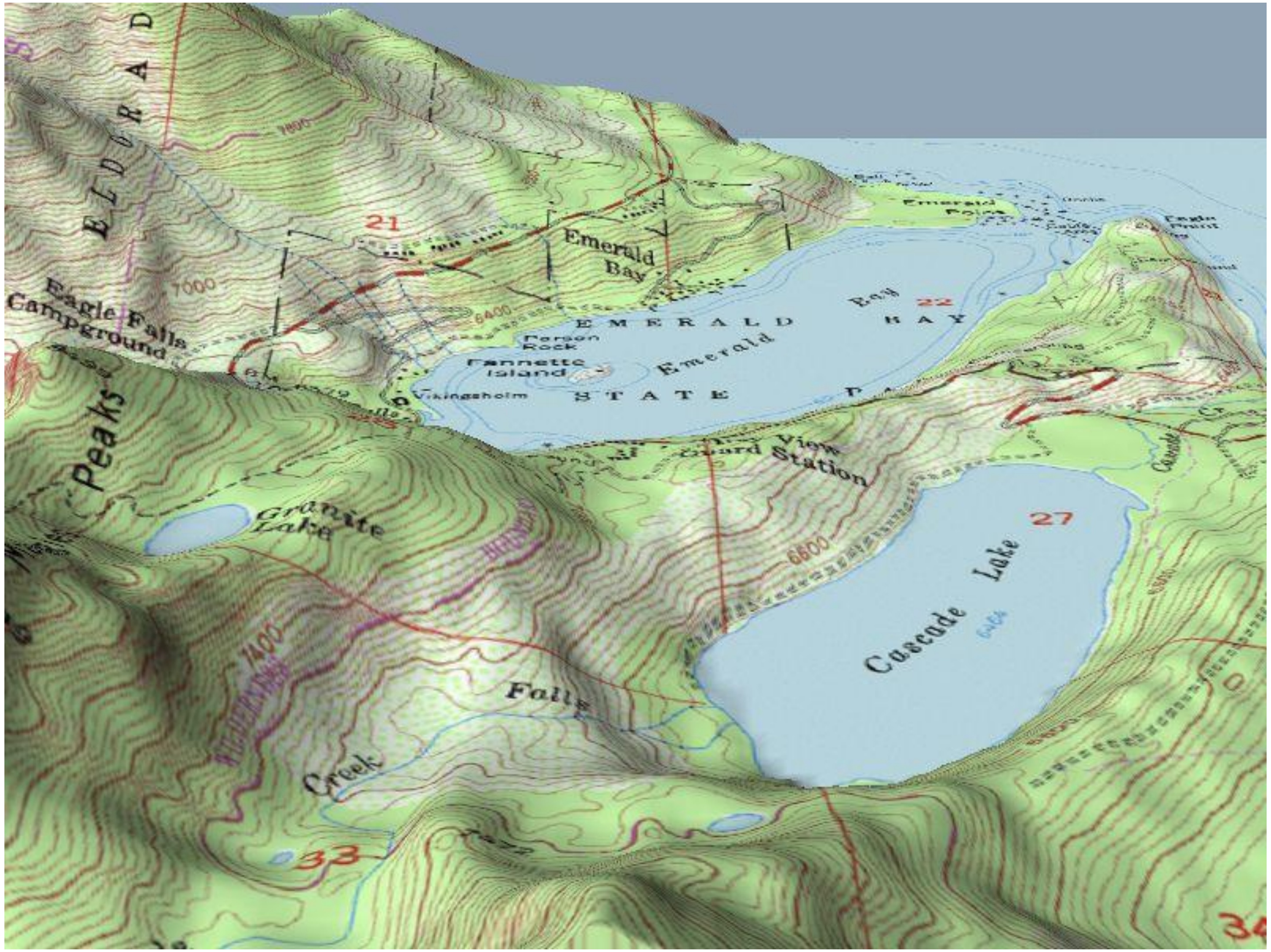
Height Plots

function plots for 2D scalar fields

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

sample function values
$\{(x, y, f(x, y)) \mid (x, y) \in \mathbb{R}^2\}$

connect neighboring samples
surface



very important

Isolines in 2D Scalar Fields

given:
scalar function $f\colon \mathbb{R}^2 \to \mathbb{R}$
isovalue $c \in \mathbb{R}$

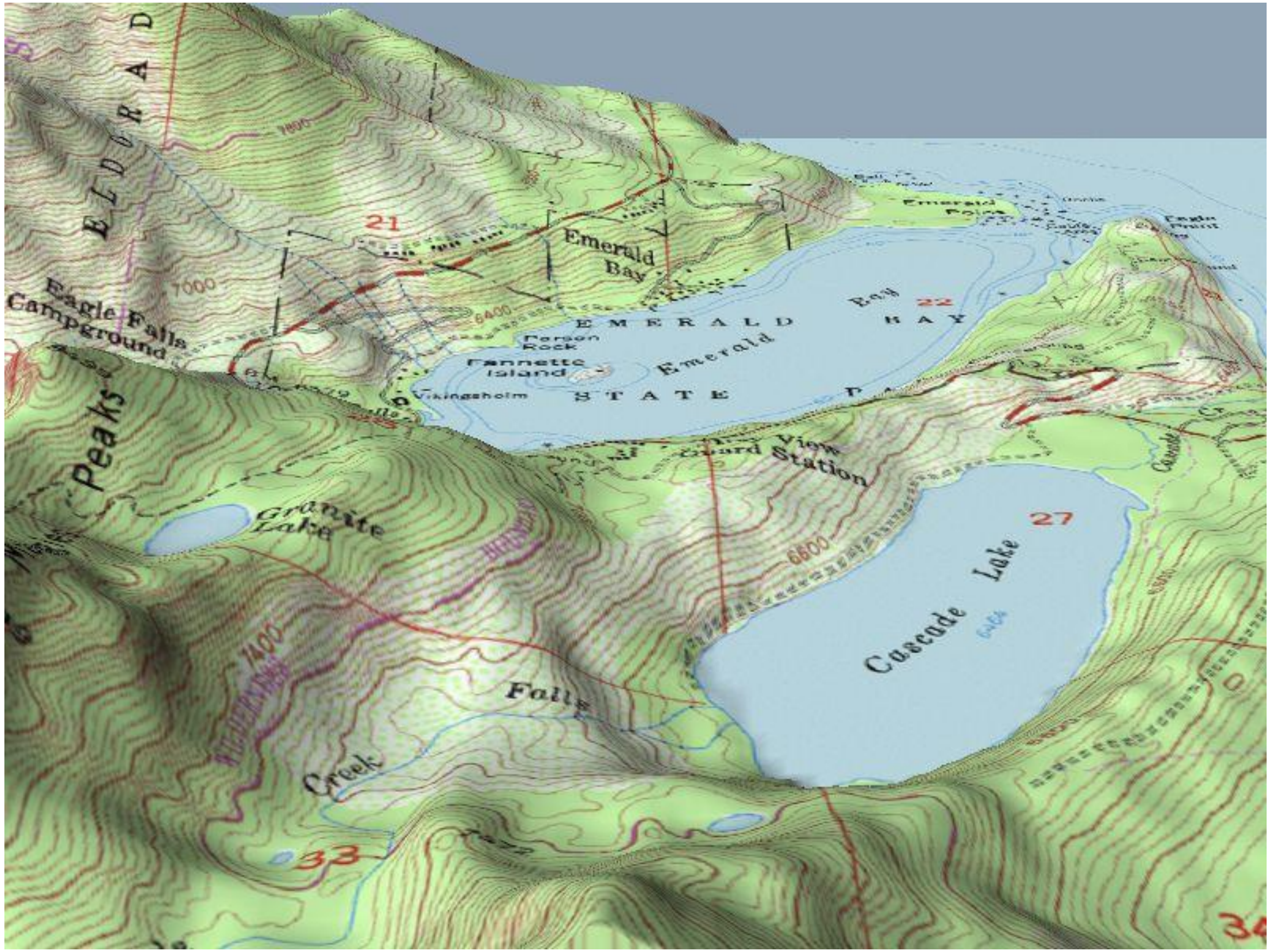
definition of **2D contour**:

$$\{(x,y) \mid f(x,y) = c\}$$

2D contours are curves

if f is differentiable and $\nabla f \neq \mathbf{0}$

common name: **isolines**



Properties of Isolines

closed curves

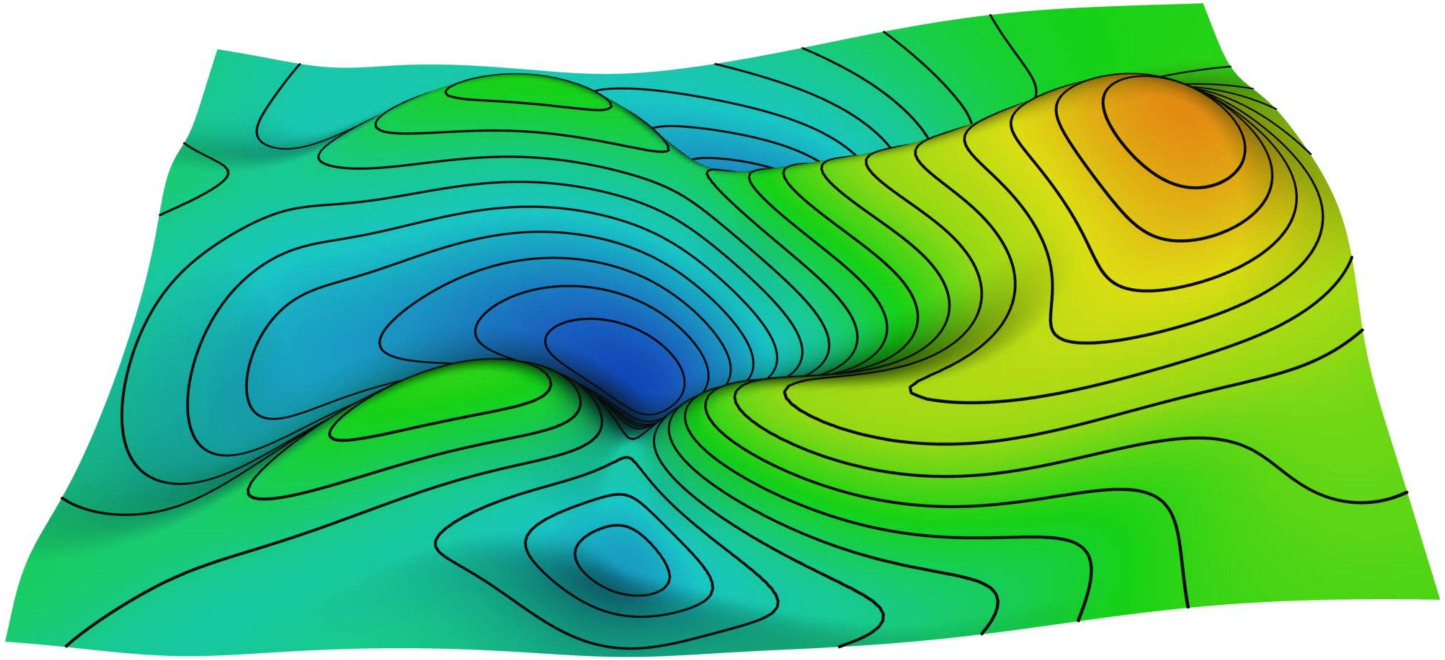
unless exiting the domain

cannot intersect each other

nested curves

points on isolines have similar semantics

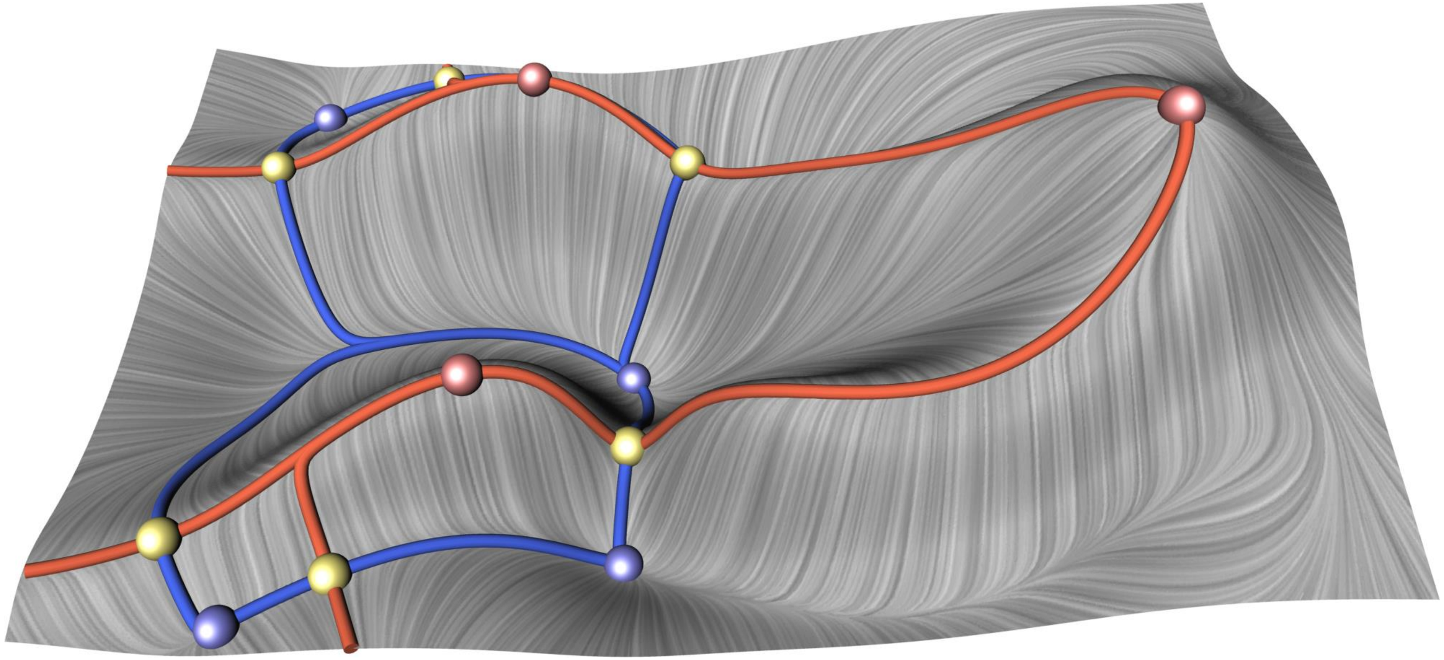
density of the lines reveals strength of the gradient



Properties of Isolines

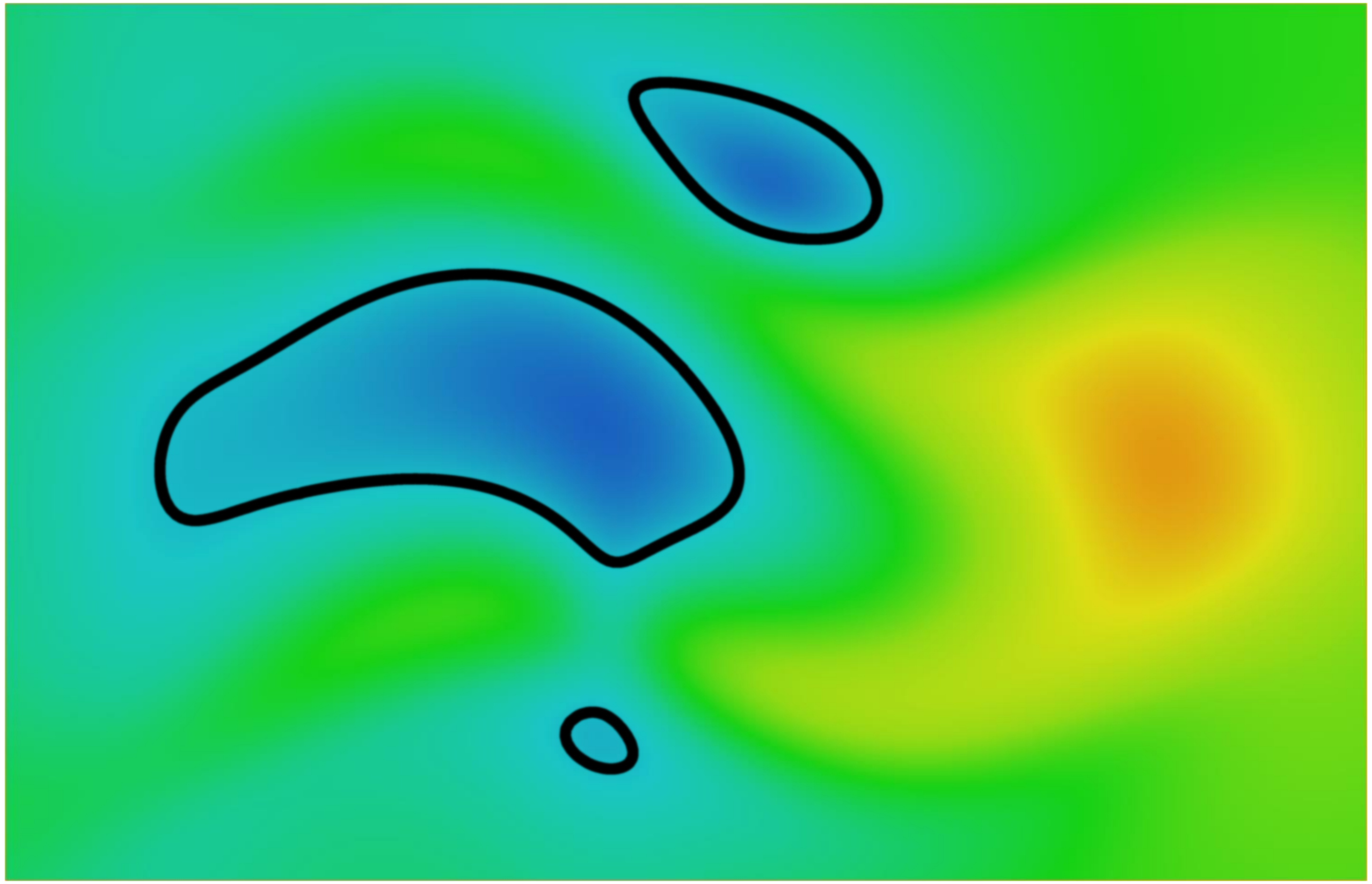
gradient vector is
perpendicular to the isolines

rate of change is zero along isolines



Properties of Isolines

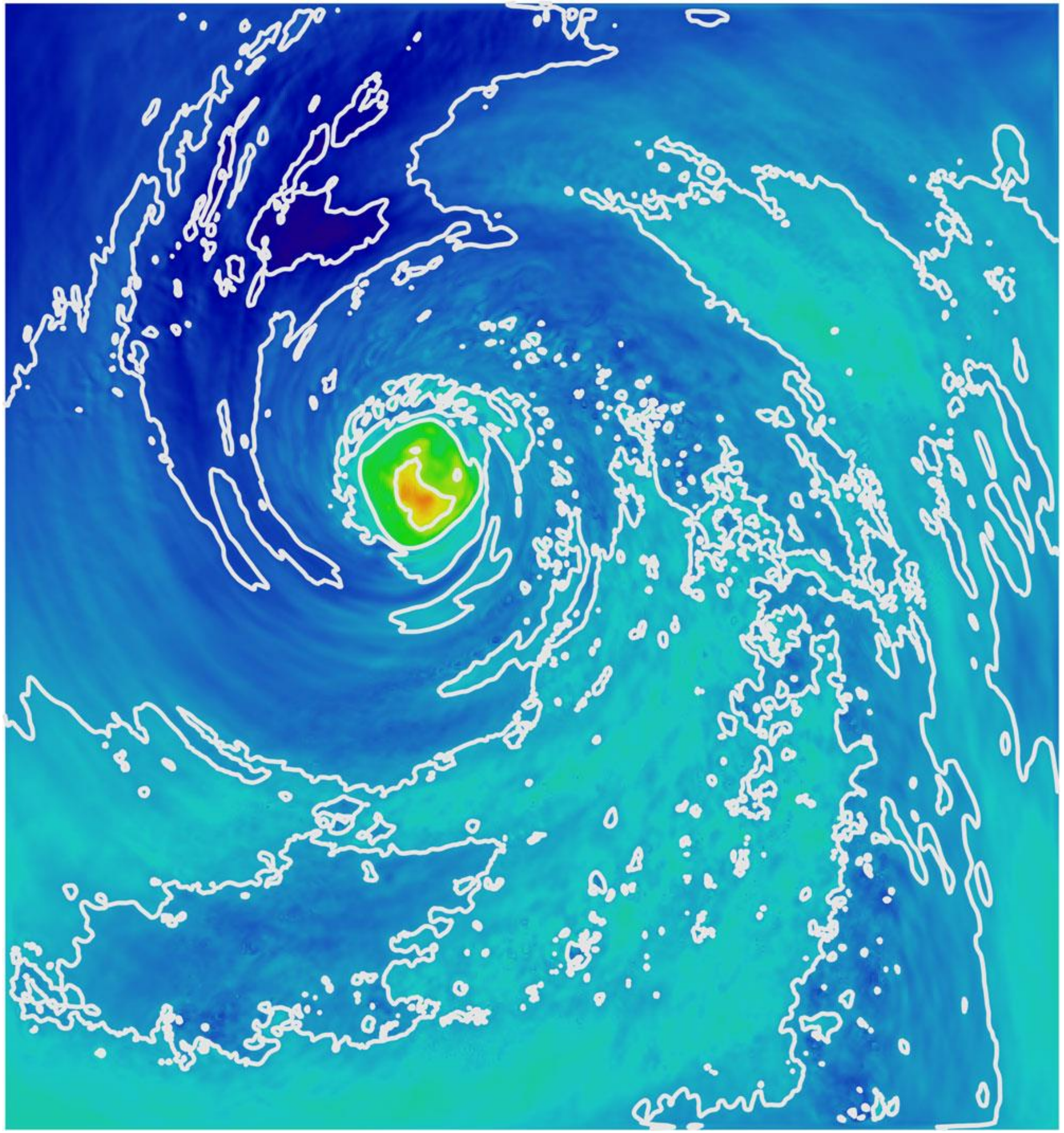
connected component:
a given isovalue produces one isocontour often consisting of several separate lines



three connected components making up one isocontour

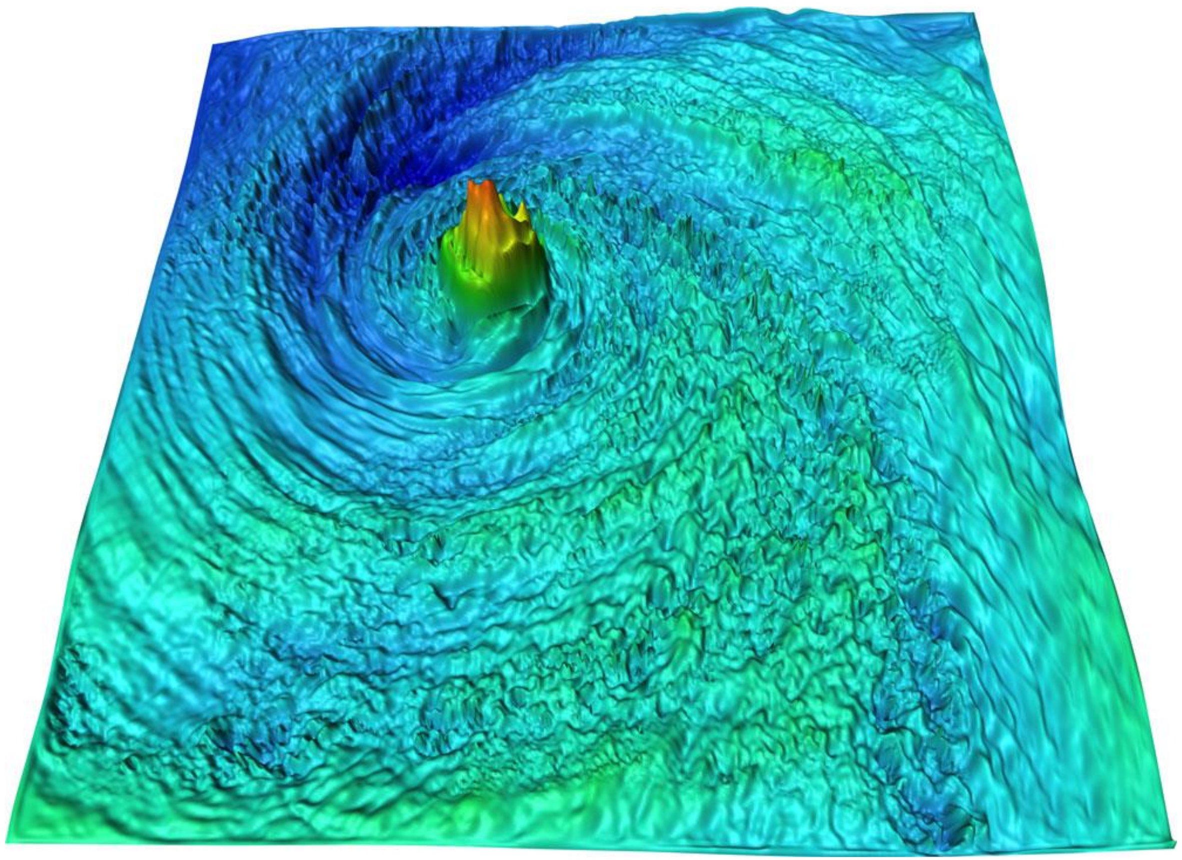
Properties of Isolines

many connected components
if data set is noisy

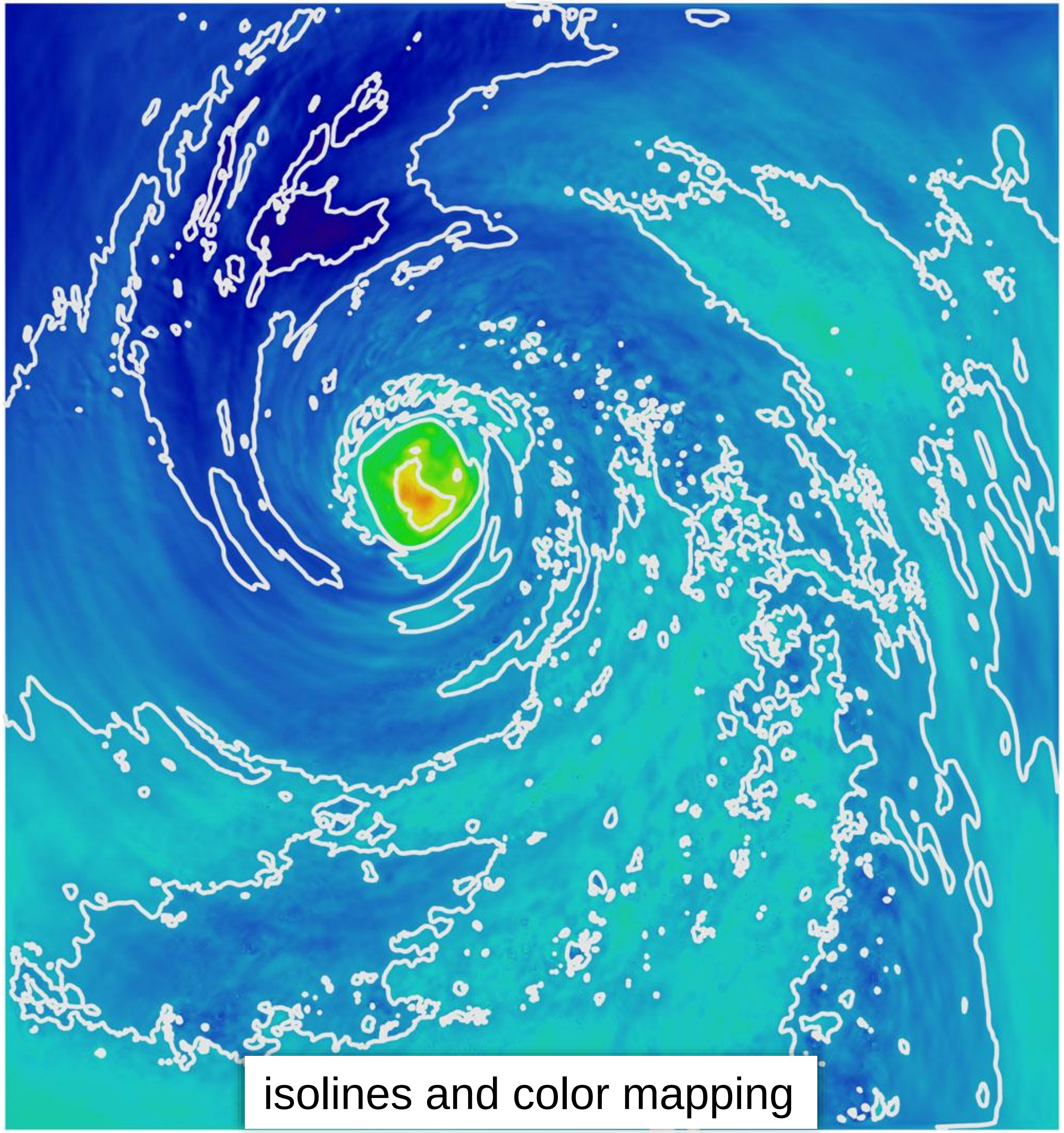


Isabel data set, NCAR, USA

no smoothing



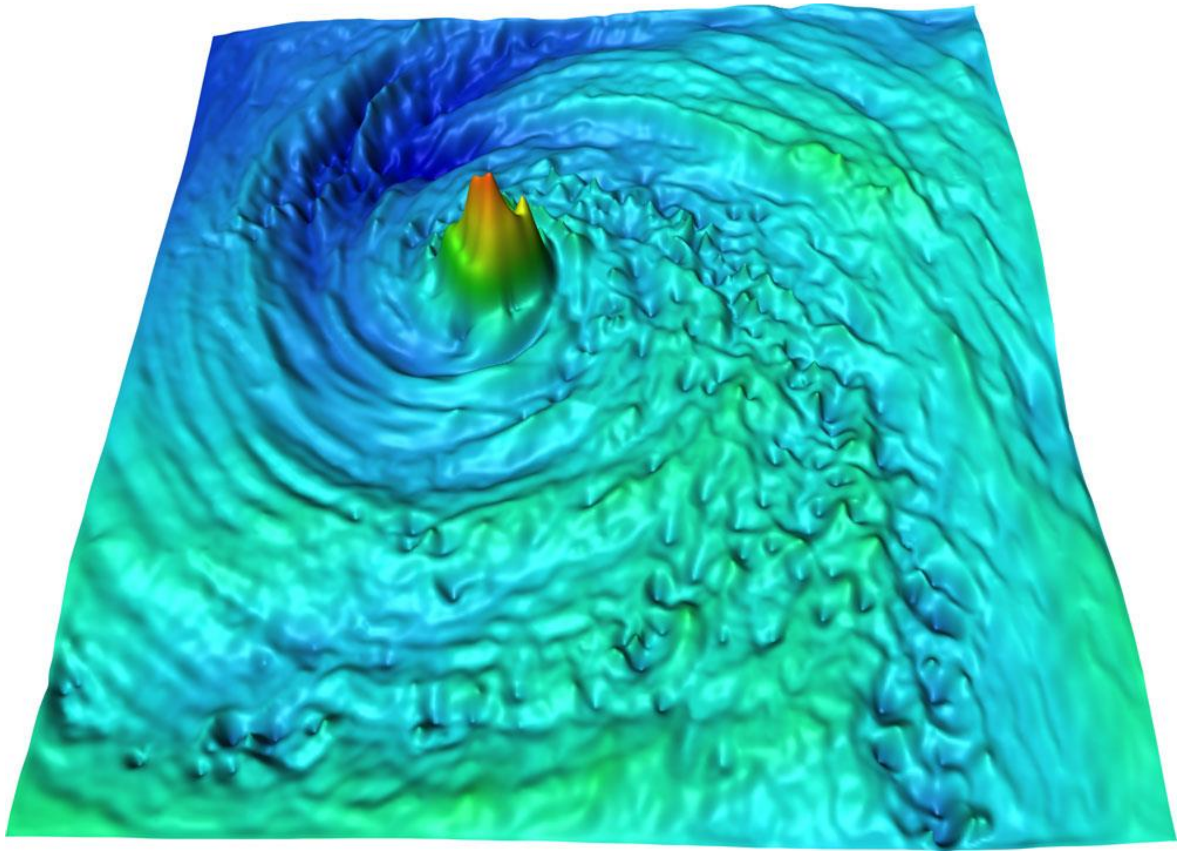
height field



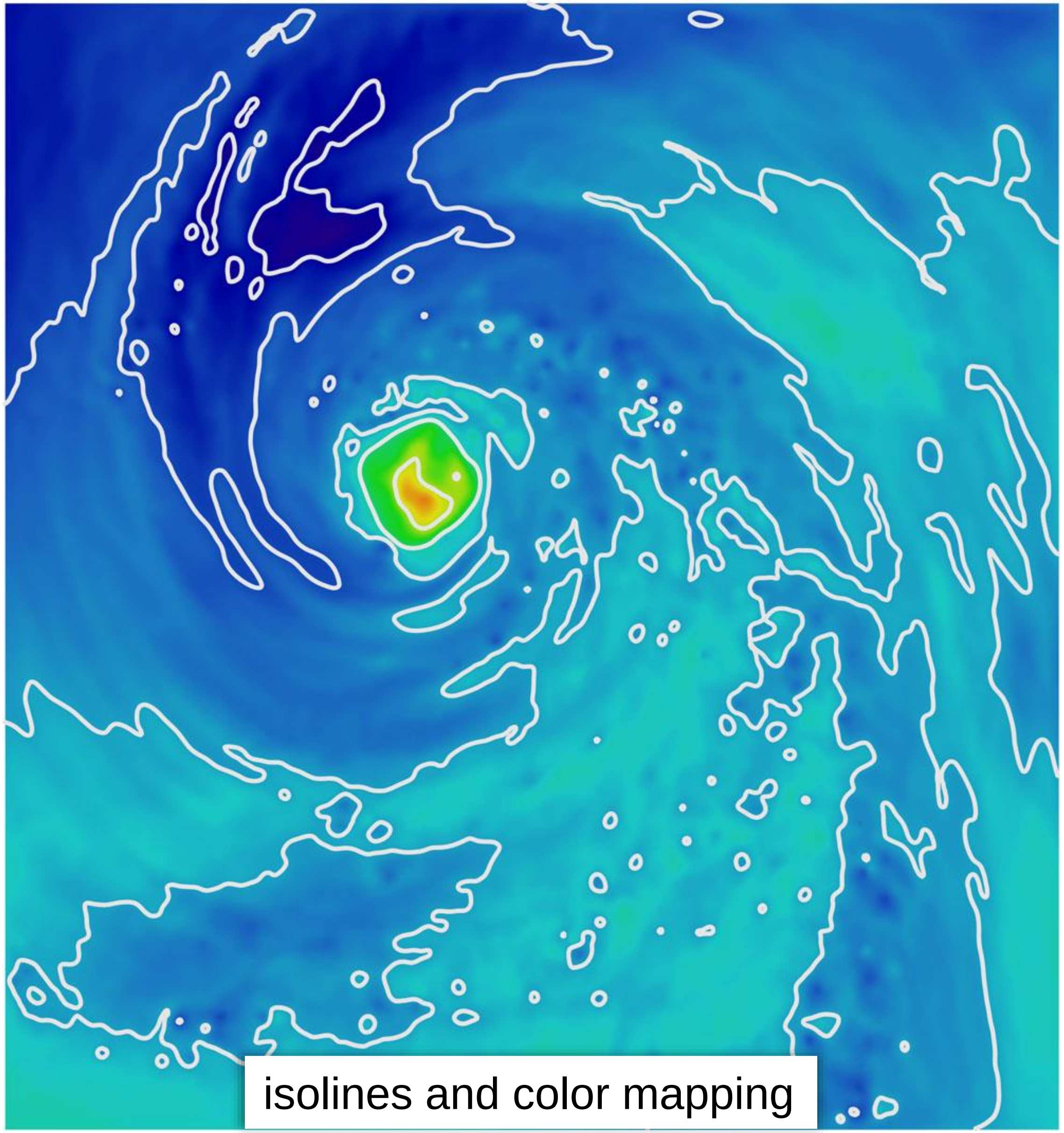
isolines and color mapping

Isabel data set, NCAR, USA

mild topological smoothing

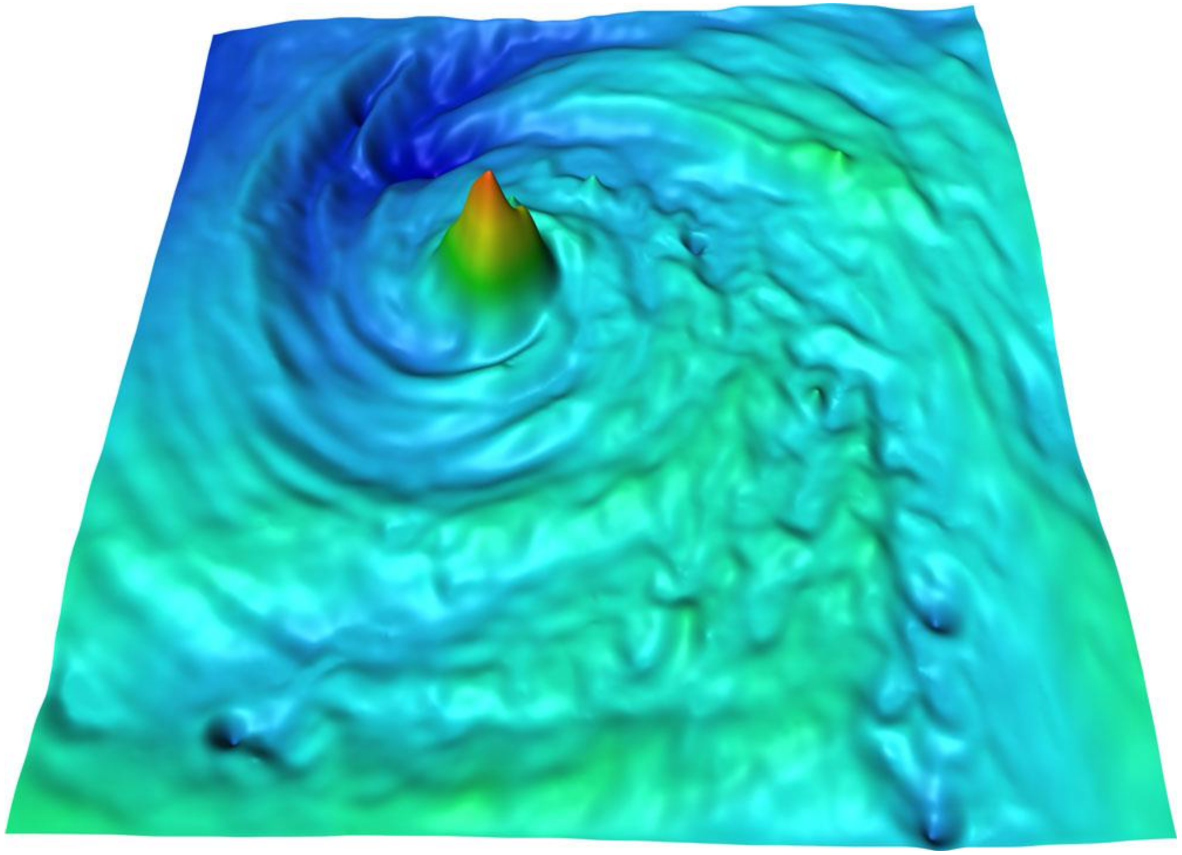


height field

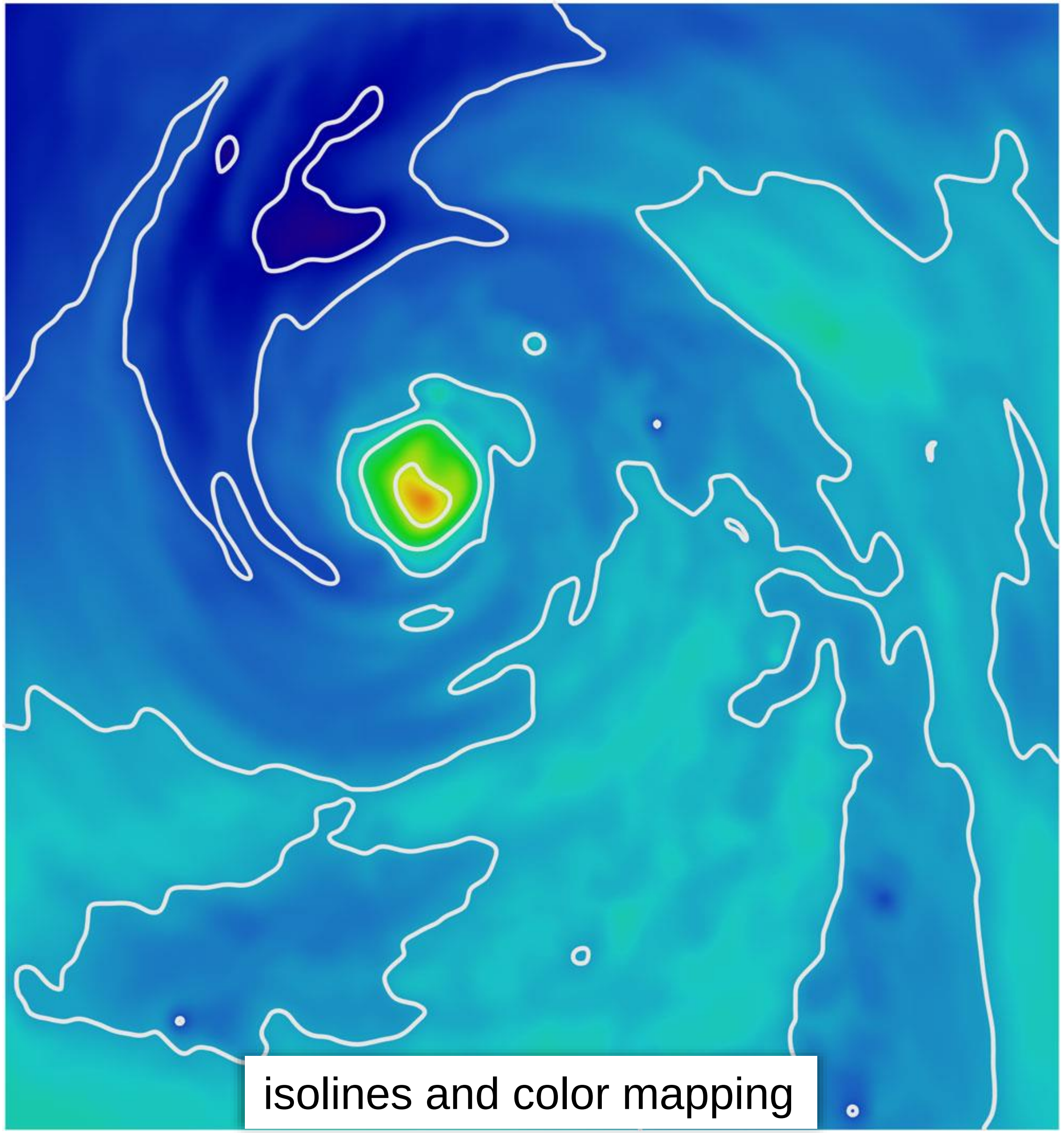


isolines and color mapping

Isabel data set, NCAR, USA. Smoothing method: Günther et al., VIS 2014 strong topological smoothing



height field

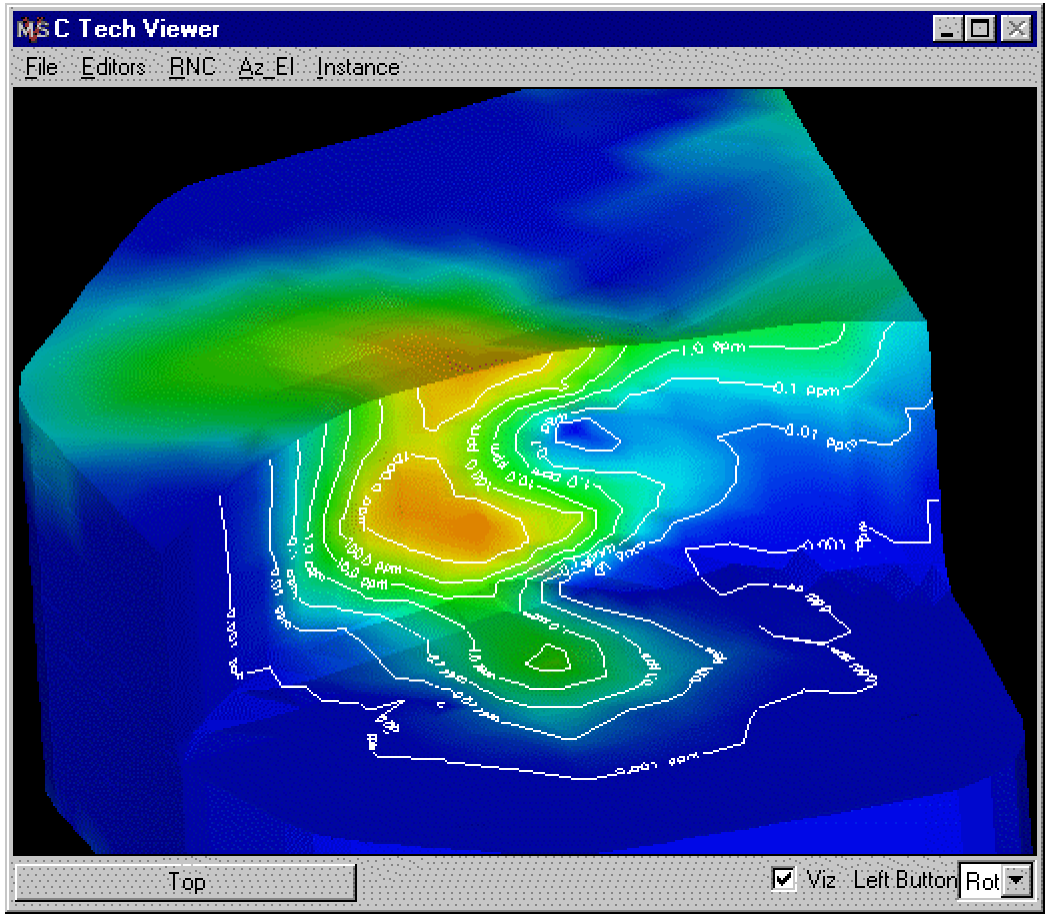


isolines and color mapping

Isabel data set, NCAR, USA. Smoothing method: Günther et al., VIS 2014

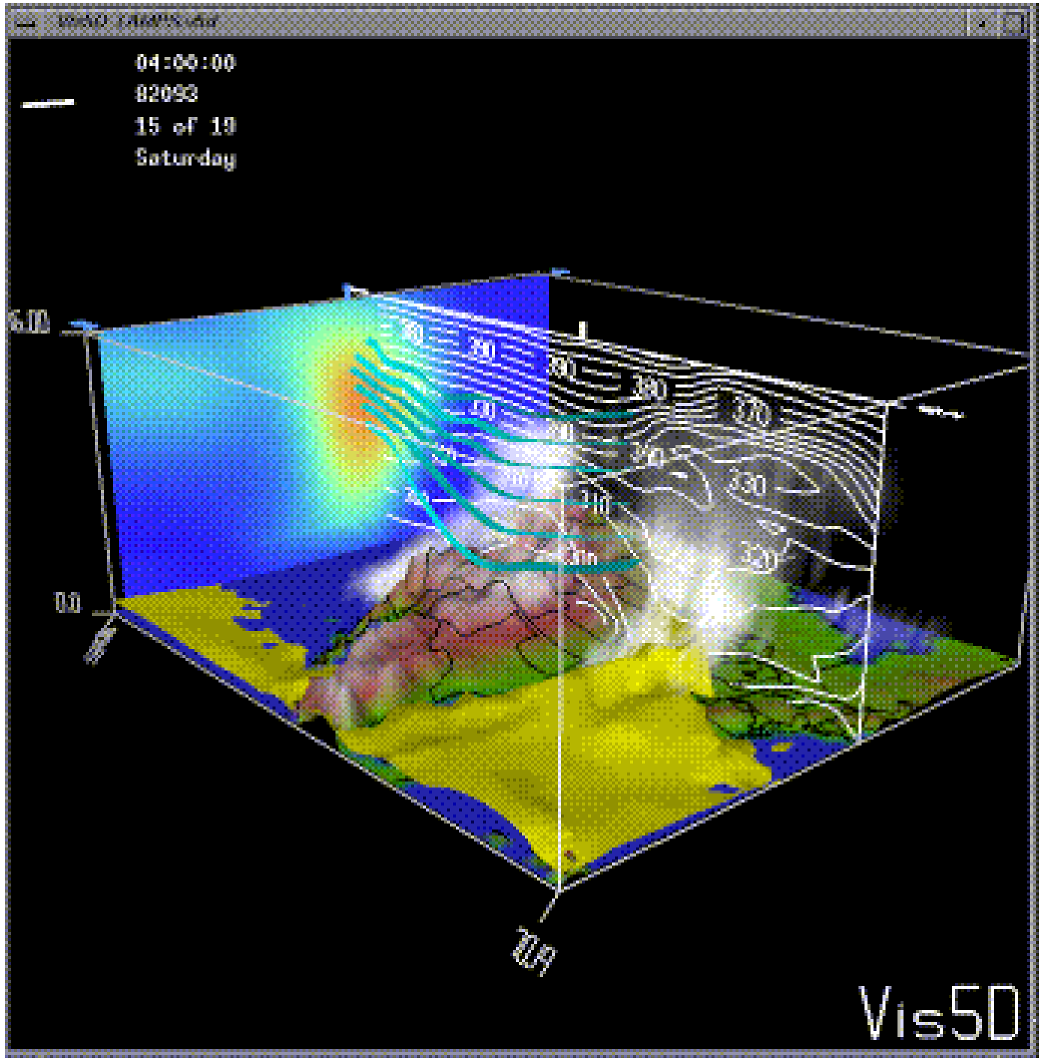
Applications of Isolines

annotate with isovalues



Applications of Isolines

can be applied to slices in 3D scalar fields



Contouring

- grid-based contouring
    - pixel-by-pixel contouring
    - marching squares
- grid-free contouring

Pixel-by-Pixel Contouring

Overlay a pixel grid onto the domain. For each pixel, $f(x, y)$ is computed.

➔ If $f(x, y)$ is within a tolerance of the isovalue, the pixel is part of the isoline.

advantages:

- reasonable image quality due to pixel-wise evaluation of function f
- different colors for different isovalues can easily be coded.

drawbacks:

- computationally intensive
- missing (parts of) isolines
- thickness of isoline varies

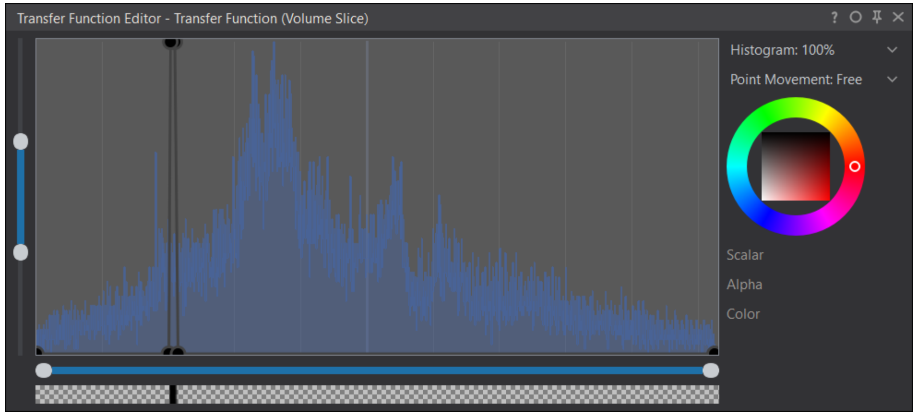
Pixel-by-Pixel Contouring

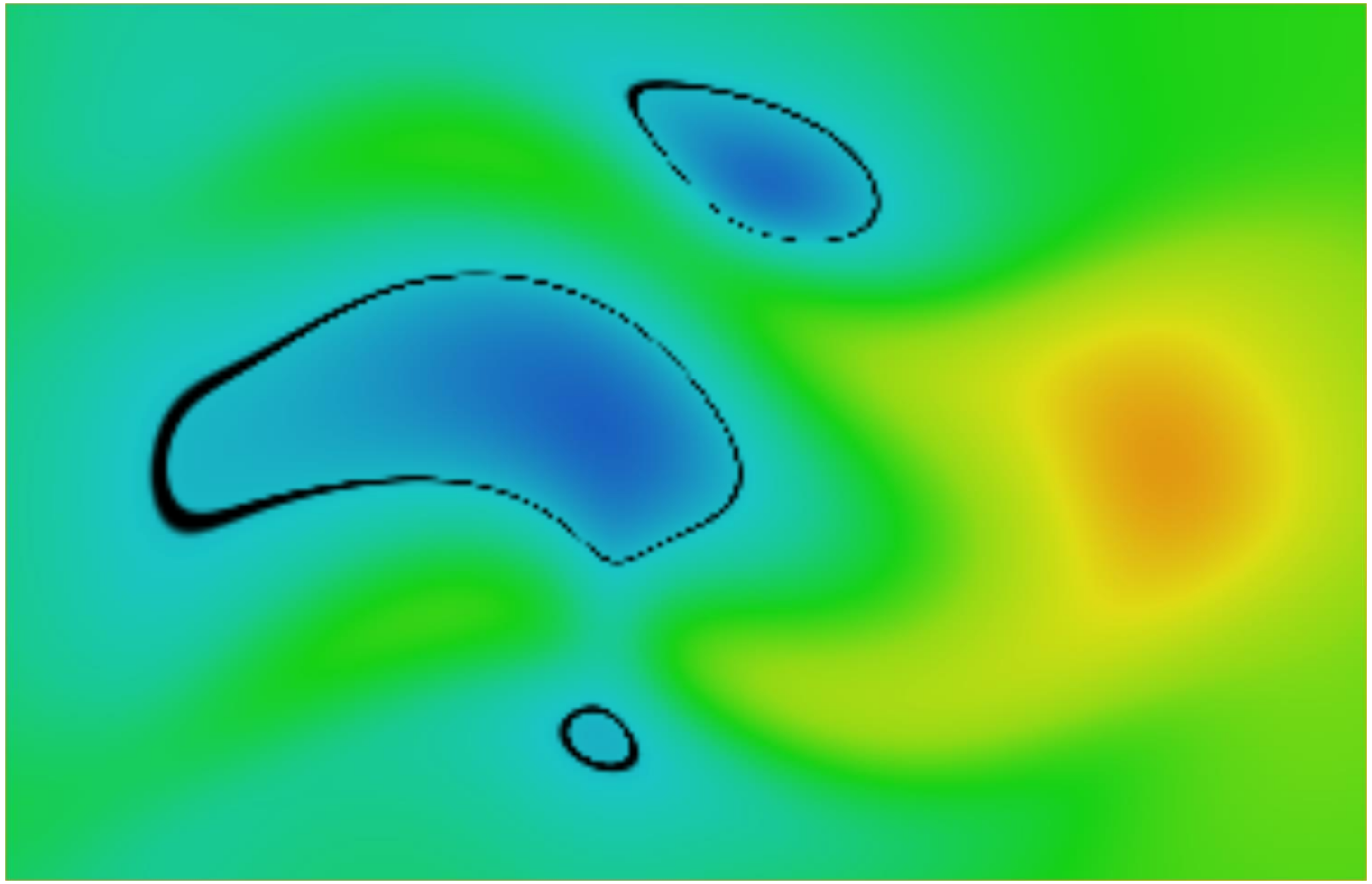
form of color mapping

transfer function has a peak

thickness varies

some parts interrupted





very important

Extraction of Isolines as Geometric Objects

- data grid is coarser than the pixel grid
- creating line segments by connecting intersection points of isolines and grid boundaries.

very important

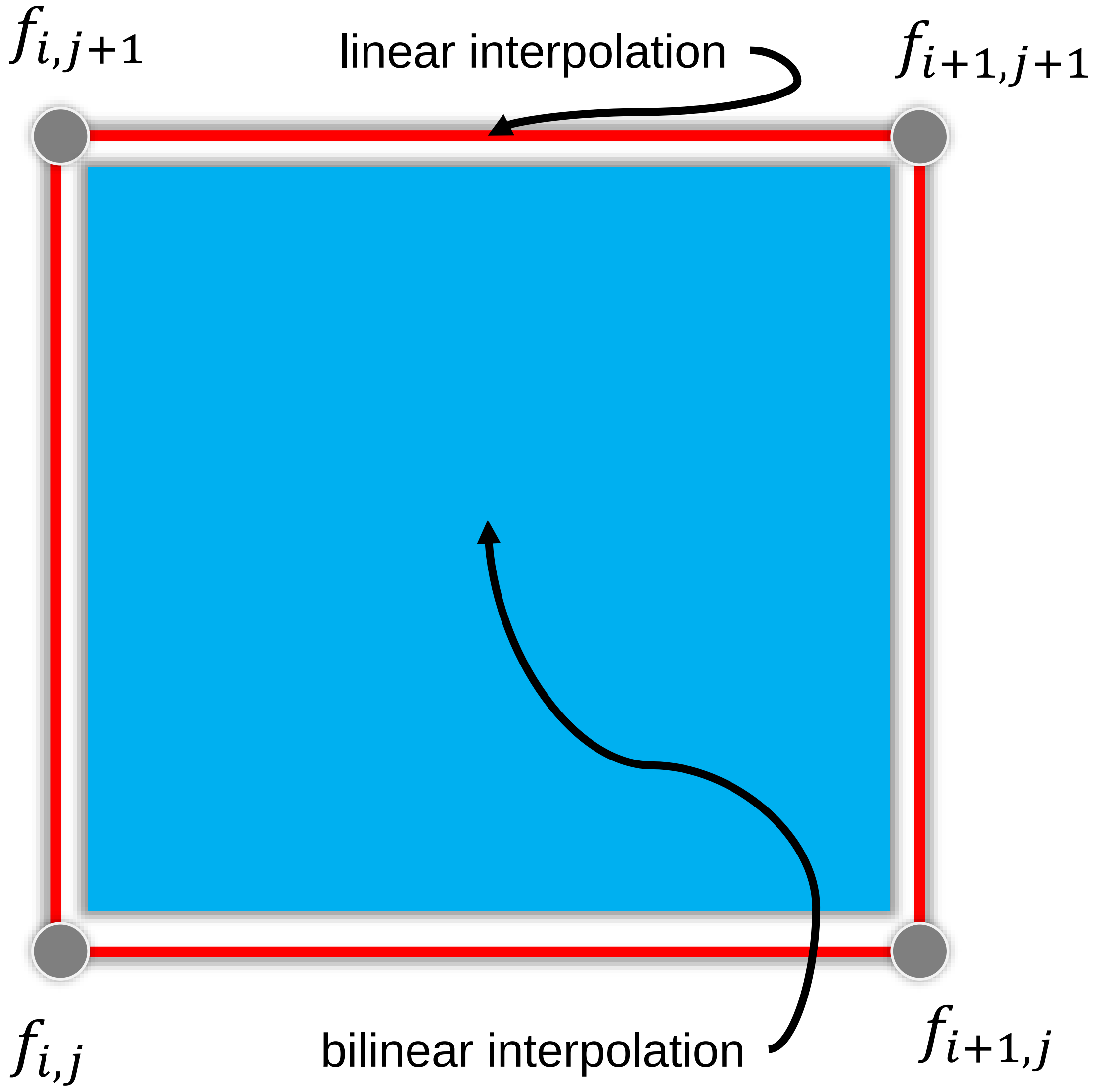
Marching Squares

input:

- data array
- isovalue c

output:

- line segments per grid cell



assumes bilinear interpolation

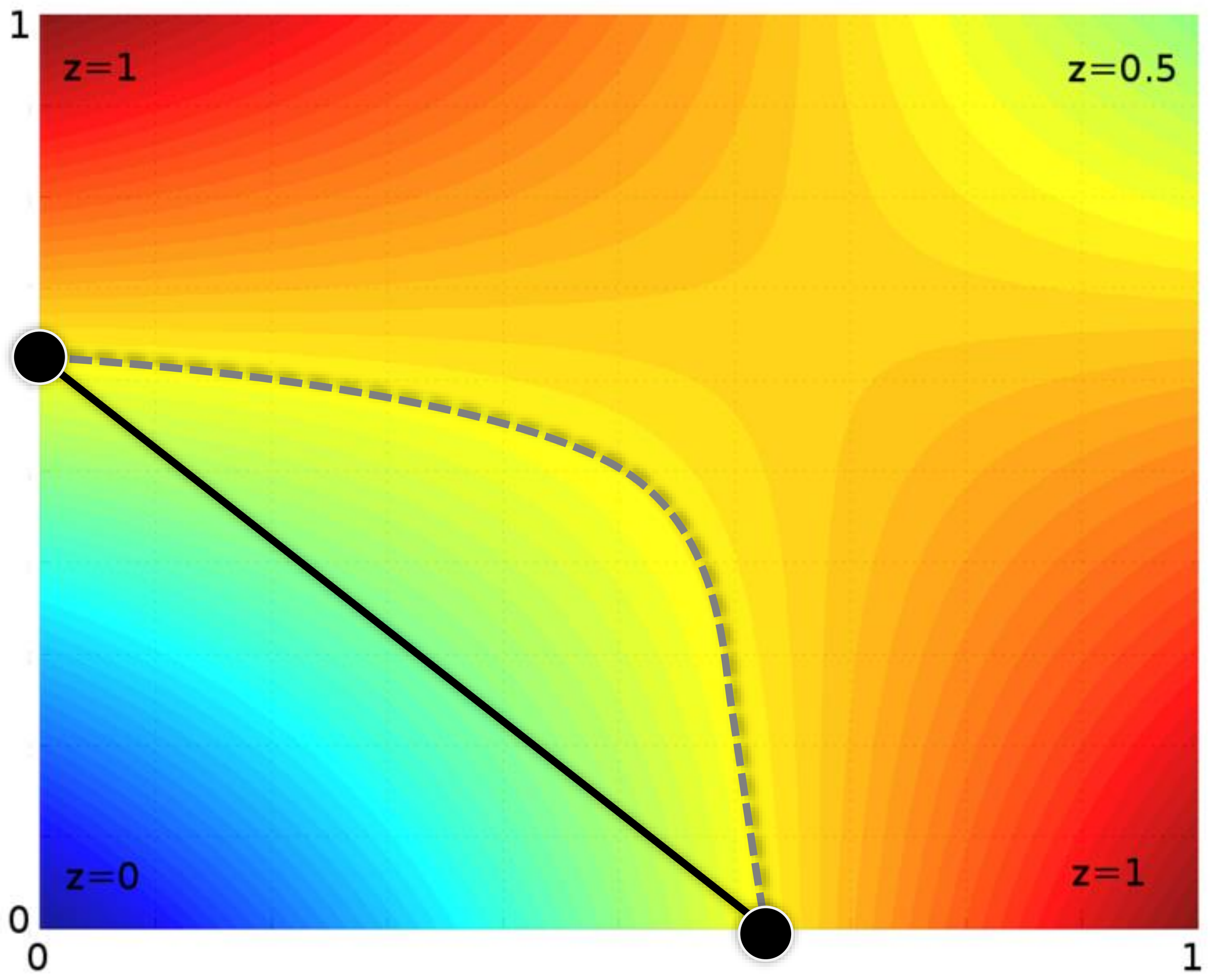
linear along grid edges

bilinear inside cells

Marching Squares

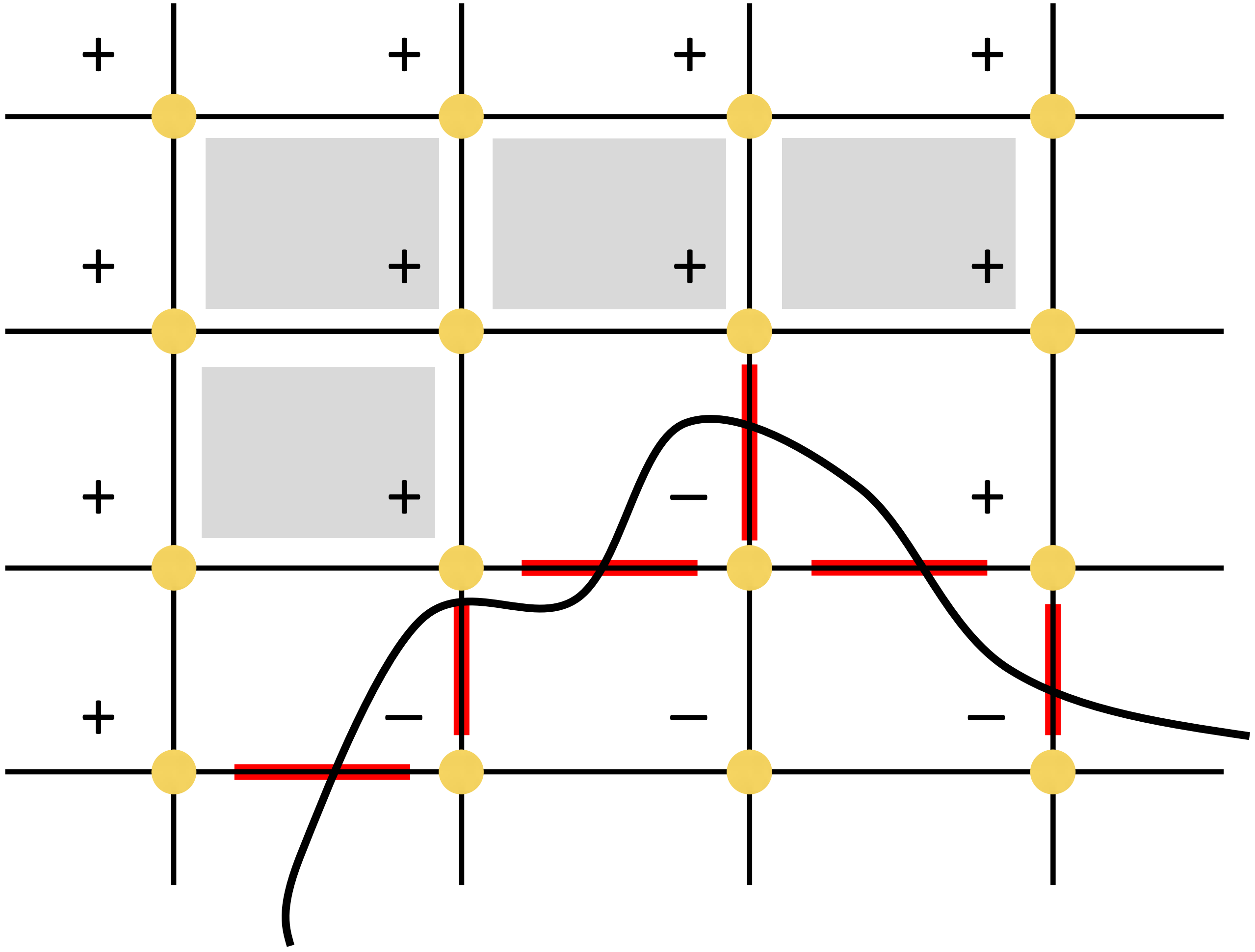
Isolines in a bilinear grid cell are hyperbolas

the Marching Squares algorithm approximates them as straight lines



Marching Squares

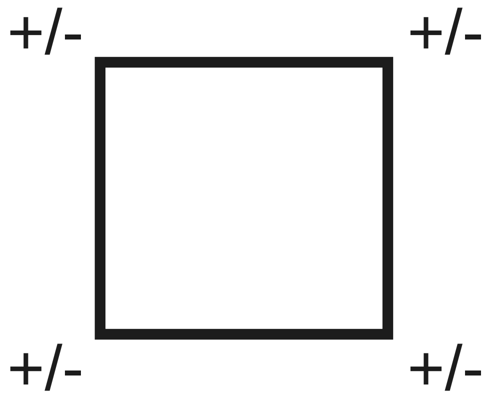
- Input: data array and isovalue c
- mark all vertices:
  $+ \Rightarrow f_{i,j} \geq c$
  $- \Rightarrow f_{i,j} < c$
- isoline passes only through cells with different signs at the four vertices (bilinear interpolation)
  $$f_{min} = \min(f_{i,j}\,,\ f_{i+1,j}\,,\ f_{i,j+1}\,,\ f_{i+1,j+1})$$
  $$f_{max} = \max(f_{i,j}\,,\ f_{i+1,j}\,,\ f_{i,j+1}\,,\ f_{i+1,j+1})$$
  $$f_{min} \leq c \leq f_{max}$$
- isoline can only intersect grid edges with different signs (property of linear interpolation)

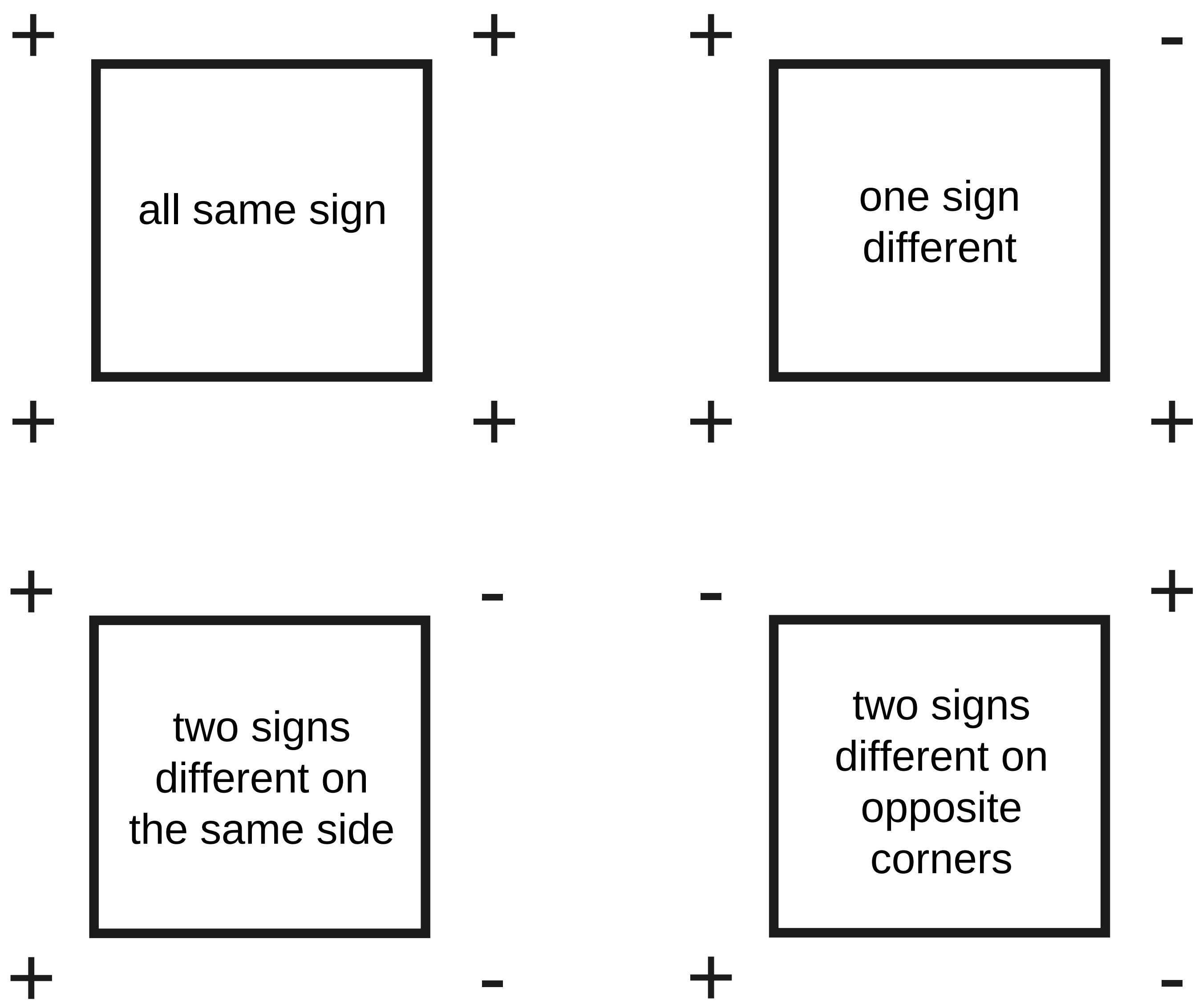


Marching Squares

Only 4 different cases of sign combinations

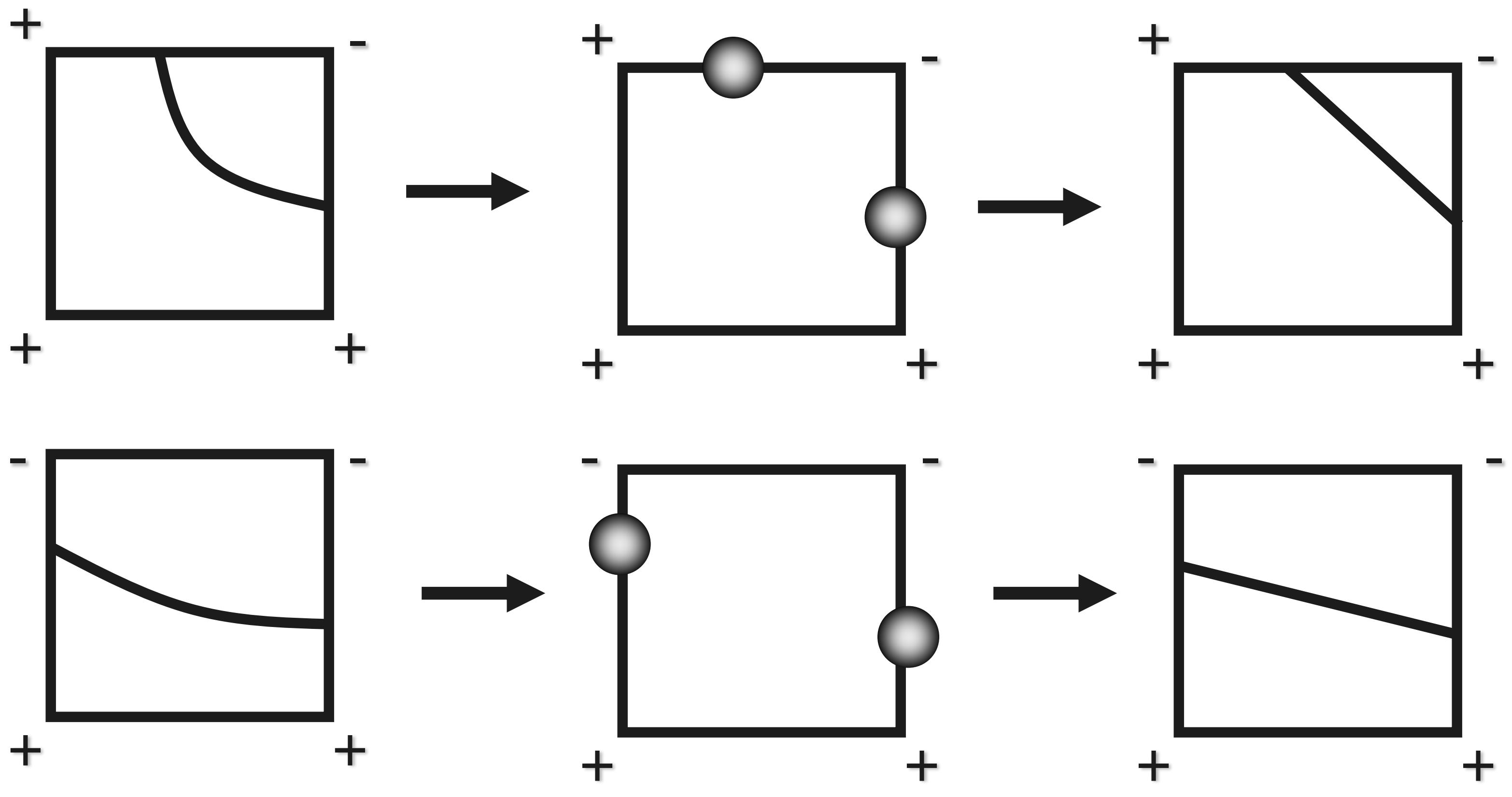
Symmetries: rotation, reflection, change + ↔ -



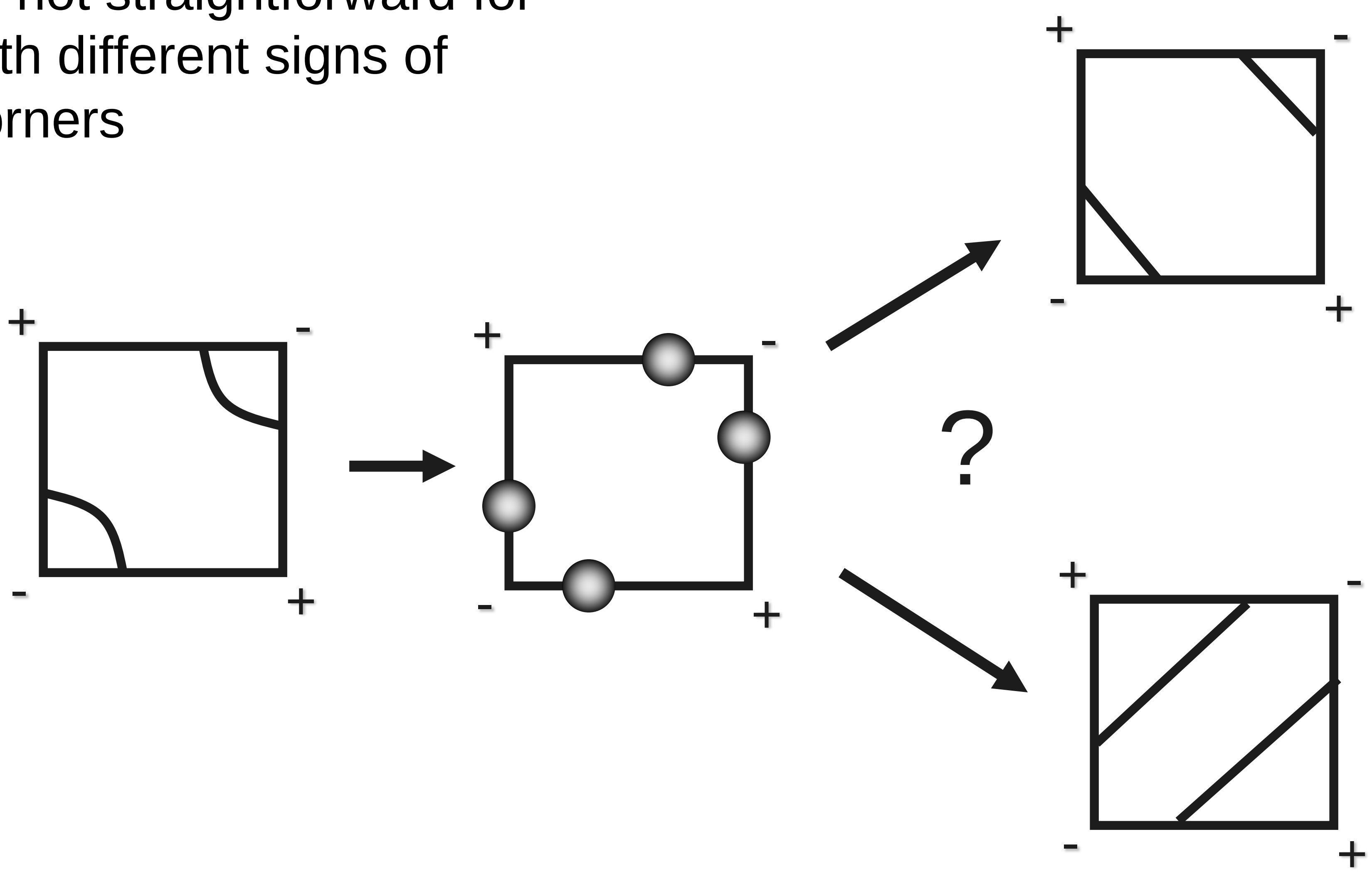


- Compute intersections between isoline and cell edge
  - Use linear interpolation along the cell edges

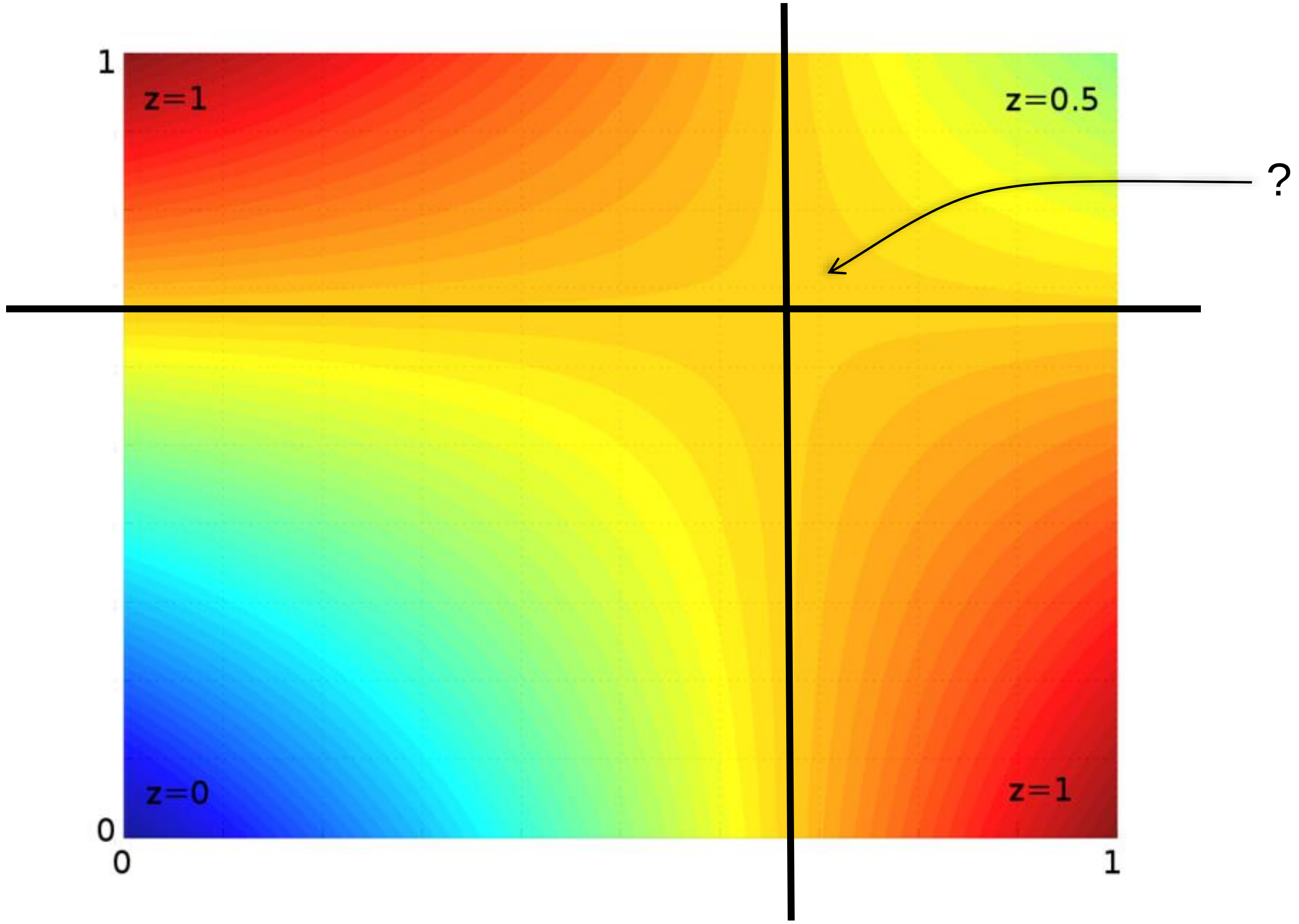
- Connect intersection points with straight line

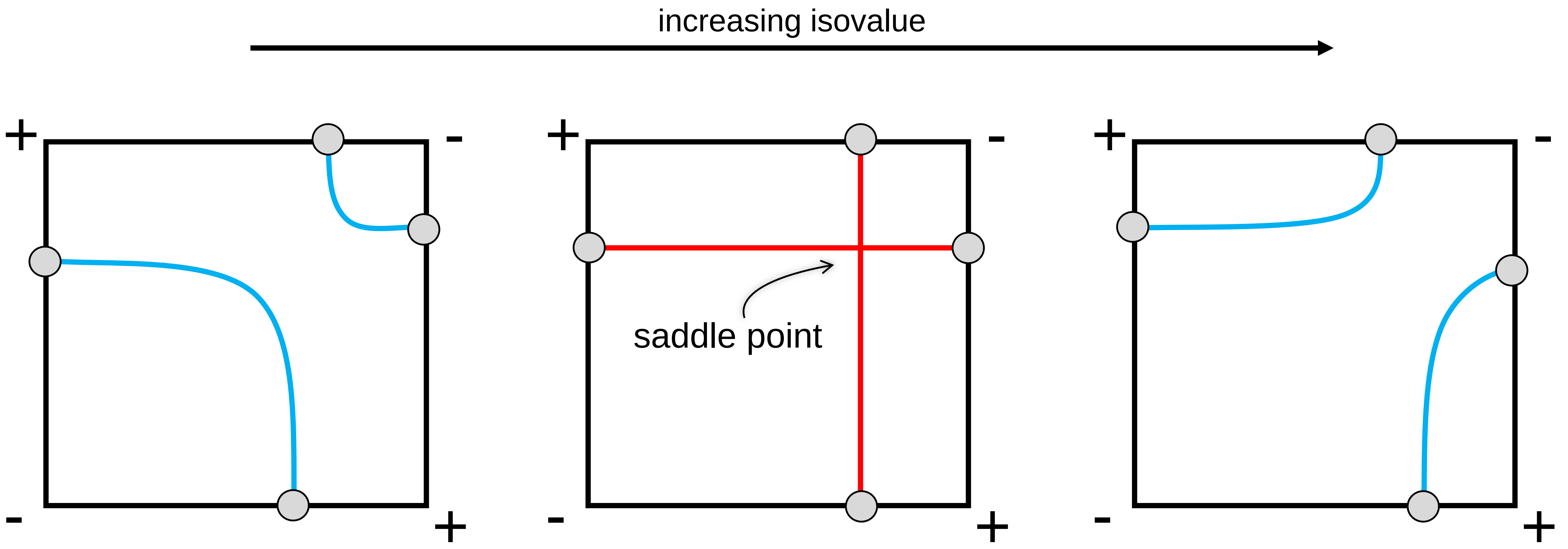


Connection not straightforward for the case with different signs of opposite corners



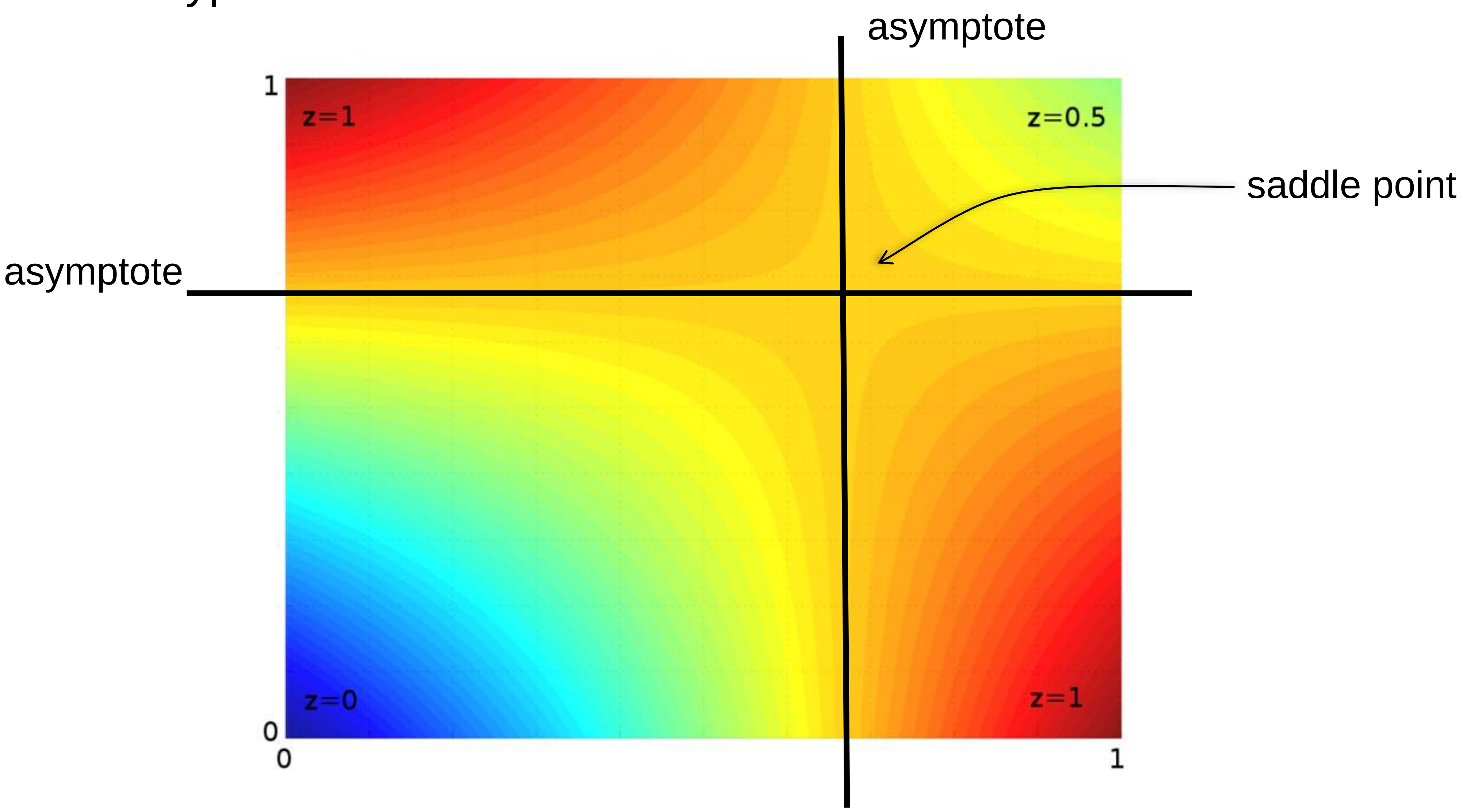
Bilinear isolines: hyperbolas





switch takes place
at the saddle's data value

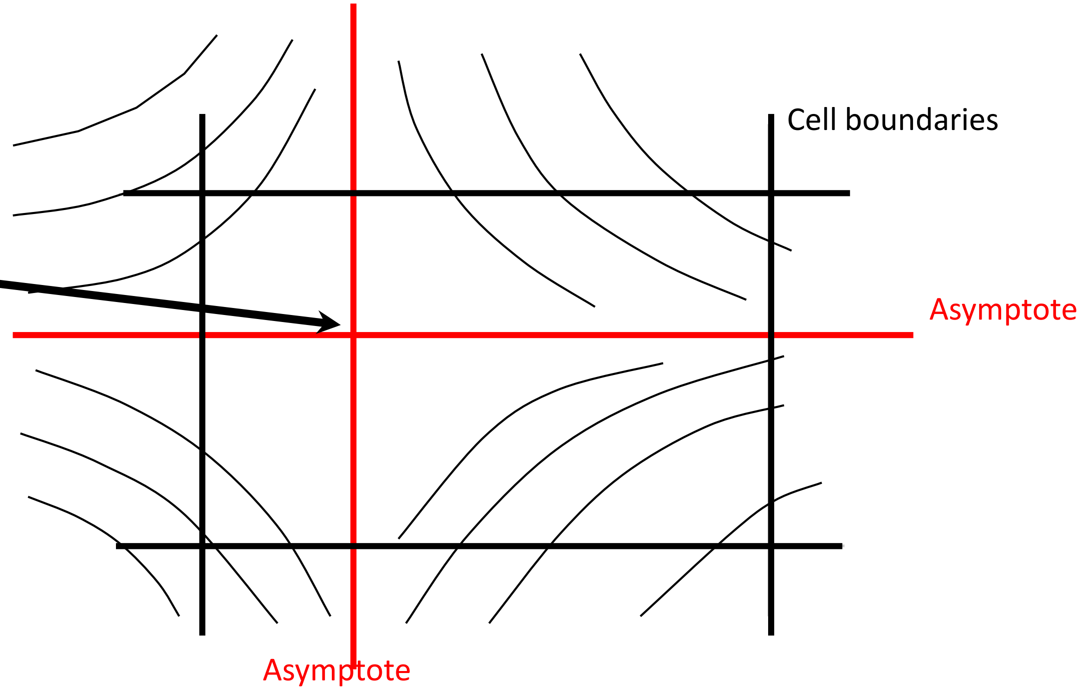
Bilinear isolines: hyperbolas



- Consider bi-linear interpolation

$$f(x,y) = f_{i,j} + (f_{i+1,j} - f_{i,j})\,x + (f_{i,j+1} - f_{i,j})\,y + (f_{i+1,j+1} + f_{i,j} - f_{i+1,j} - f_{i,j+1})\,xy$$

$$f(x_a, y_a) = \beta$$



$$\begin{aligned} f(x_a, y_a) =& f_{i,j} + (f_{i+1,j} - f_{i,j})\, x_a + (f_{i,j+1} - f_{i,j})\, y_a + \\ & (f_{i+1,j+1} + f_{i,j} - f_{i+1,j} - f_{i,j+1})\, x_a y_a = \beta \end{aligned}$$

Solve for x_a:

$$\Rightarrow \quad x_a = \frac{\beta - (f_{i,j+1} - f_{i,j})\, y_a - f_{i,j}}{(f_{i+1,j} - f_{i,j}) + (f_{i+1,j+1} + f_{i,j} - f_{i+1,j} - f_{i,j+1})\, y_a}$$

$$\xrightarrow[y_a \to \infty]{} \quad x_a = \frac{f_{i,j} - f_{i,j+1}}{f_{i+1,j+1} + f_{i,j} - f_{i+1,j} - f_{i,j+1}}$$

Similar for y_a:

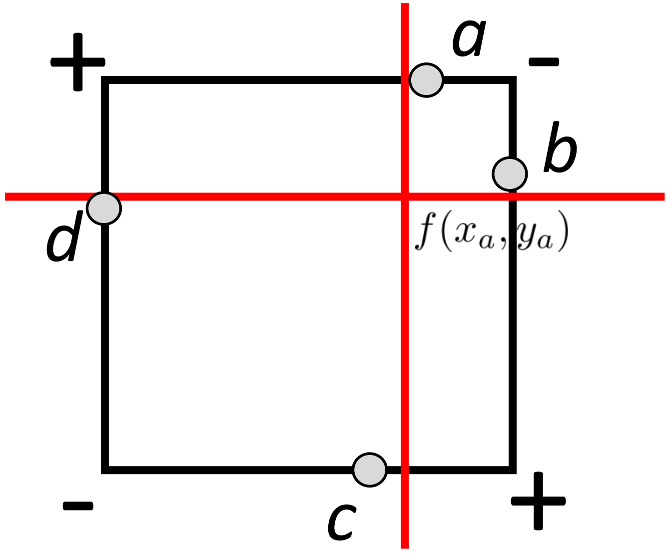
$$y_a = \frac{f_{i,j} - f_{i+1,j}}{f_{i+1,j+1} + f_{i,j} - f_{i+1,j} - f_{i,j+1}}$$

$$f(x_a, y_a) = f_{i,j} + (f_{i+1,j} - f_{i,j})\, x_a + (f_{i,j+1} - f_{i,j})\, y_a + (f_{i+1,j+1} + f_{i,j} - f_{i+1,j} - f_{i,j+1})\, x_a y_a = \beta$$

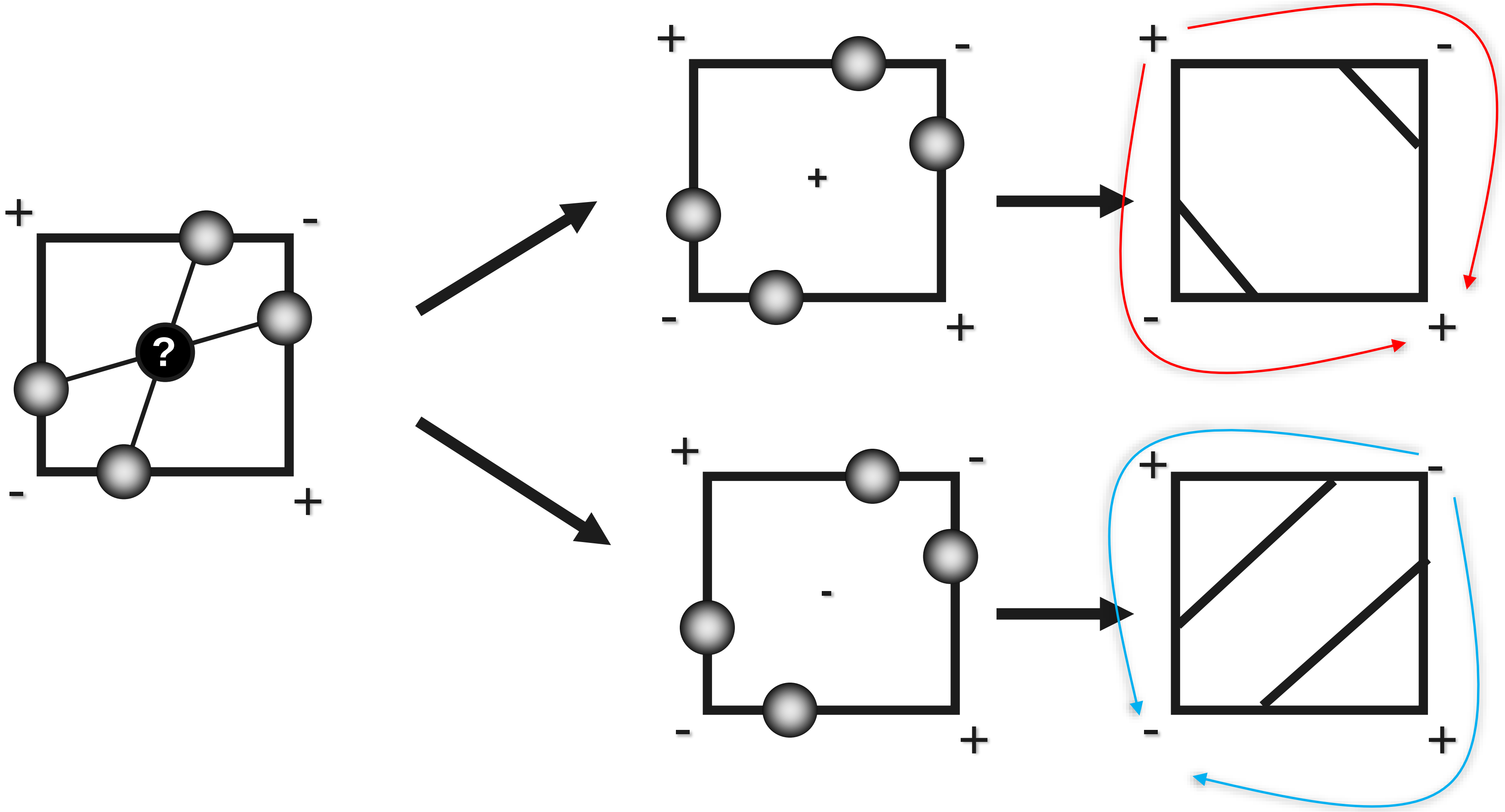
$$\Rightarrow \quad f(x_a, y_a) = \frac{f_{i,j}\, f_{i+1,j+1} - f_{i+1,j}\, f_{i,j+1}}{f_{i+1,j+1} + f_{i,j} - f_{i+1,j} - f_{i,j+1}}$$

"Asymptotic Decider"

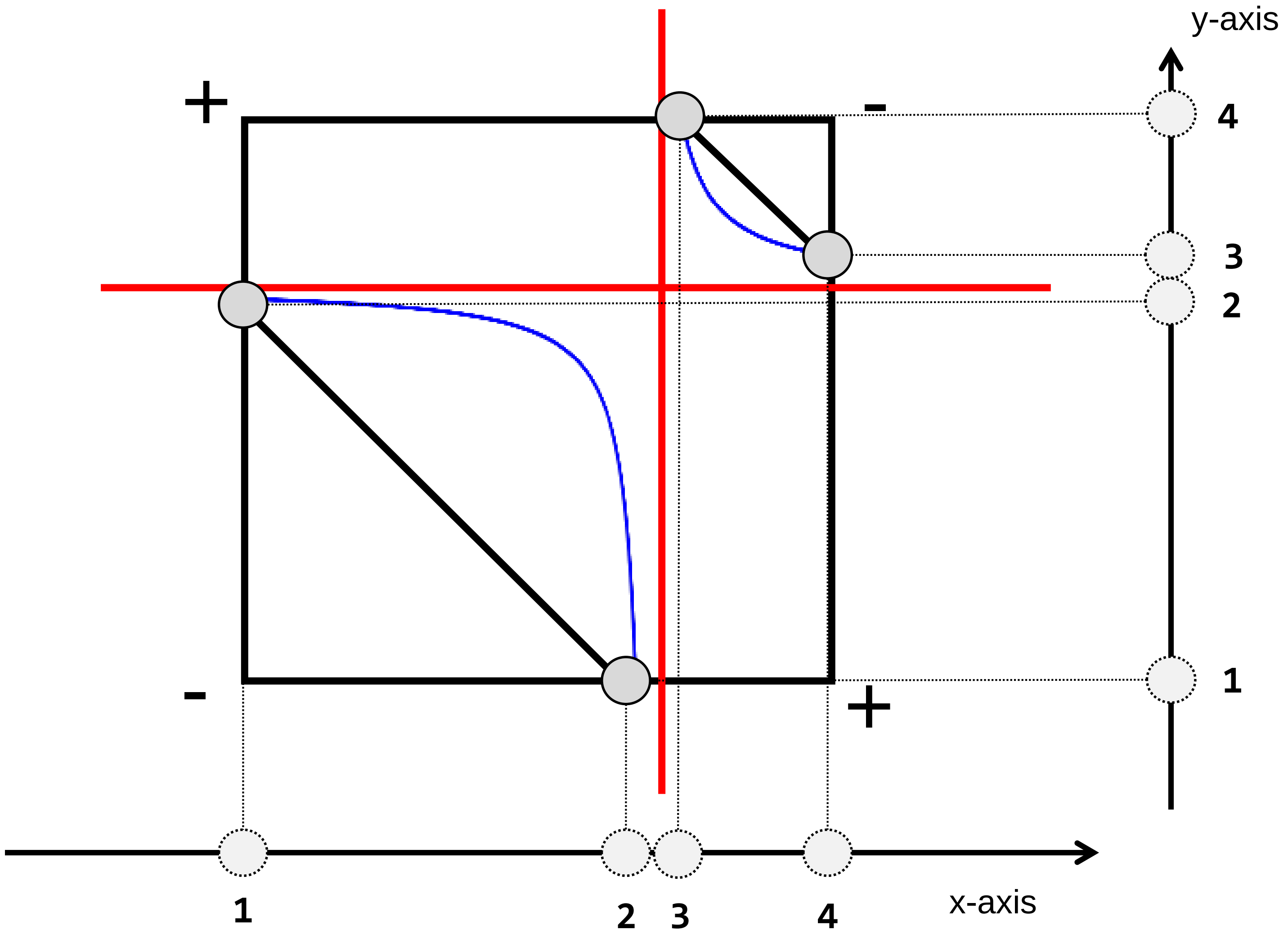
```
if f(x_a, y_a) ≥ c:
  connect (a,b) and (c,d)
else:
  connect (a,d) and (b,c)
```



- Decide based on value at saddle point



- Sort intersection points by their x or y coordinates
- Connect (1,2) and (3,4)



very important

Marching Squares

Input: data array and isovalue

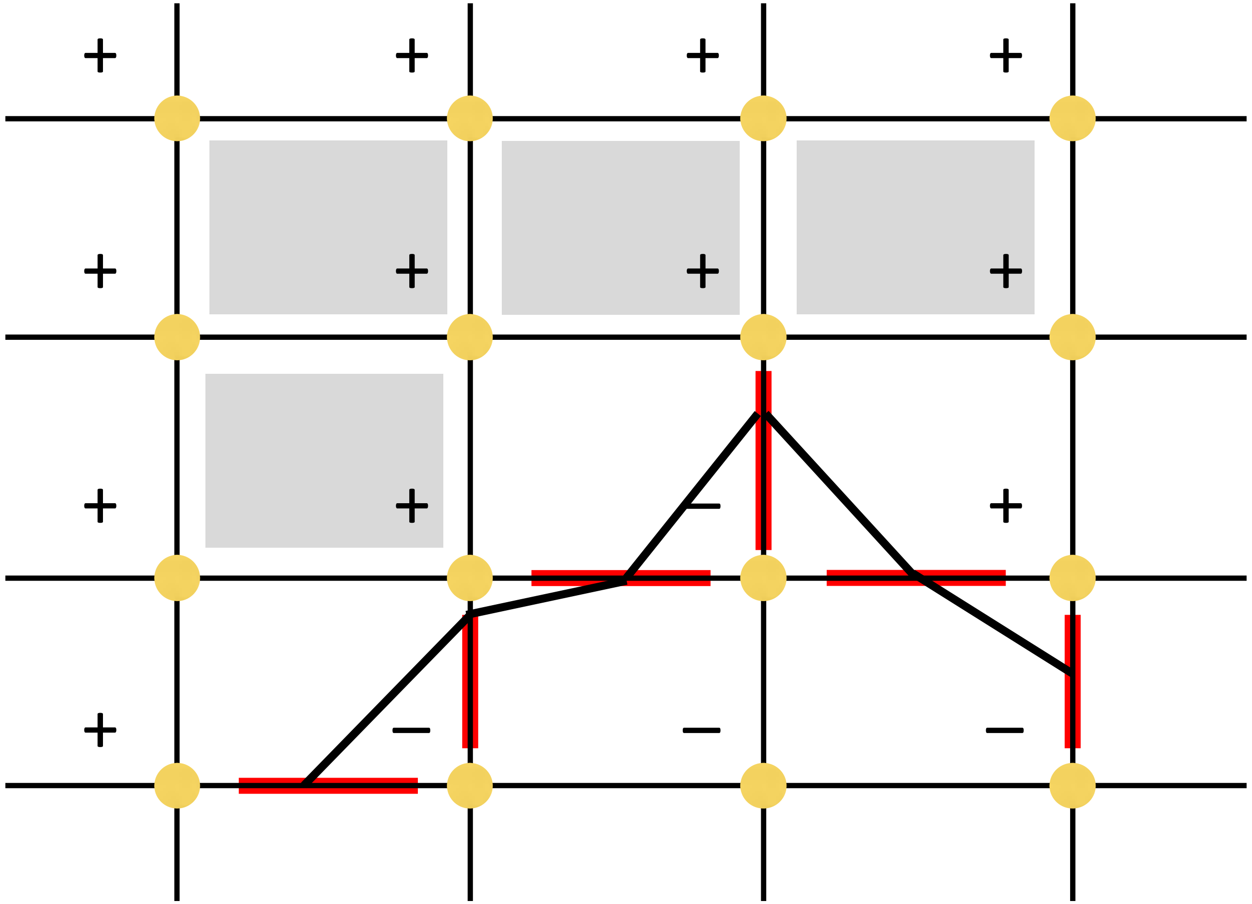
Iterate over all grid cells

4 possible cell cases

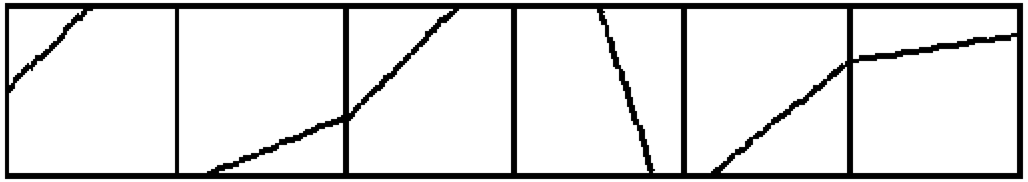
Find intersection points of grid edges and isoline

inverted linear interpolation

Draw isoline

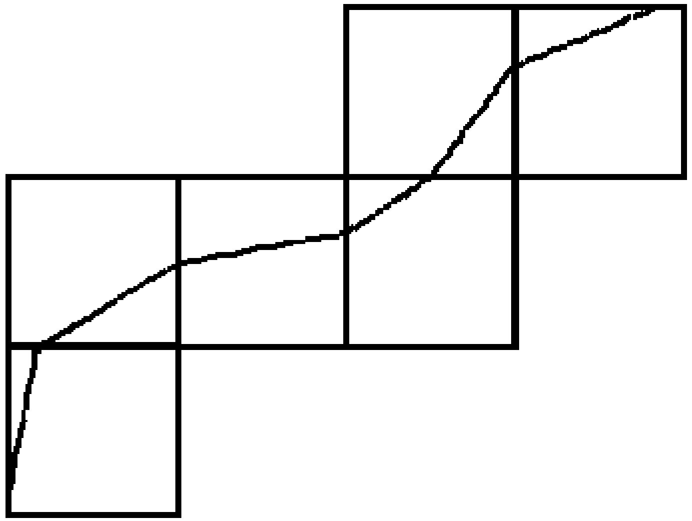


- Marching squares processes data in cell order
  - Traverse all cells of the grid
  - Apply marching squares technique to each single cell



- Disadvantage of cell order method
  - Every vertex (of the isoline) and every edge in the grid is processed twice
  - The output is just a collection of pieces of isolines which have to be post-processed to get (closed) polylines

- Contour tracing approach
  - Start at a seed point of the isoline
  - Move to the neighboring cell into which the isoline enters
  - Trace isoline until either
    - Bounds of the domain are reached, or
    - Isoline is closed



- Problem: How to find seed points efficiently?
  - In a preprocessing step, mark all cells which have a sign change
  - Remove marker from cells which are traversed during contour tracing (unless there are 4 intersection edges!)

very important

Isosurfaces in 3D Scalar Fields

given:
scalar function $f: \mathbb{R}^3 \to \mathbb{R}$
isovalue $c \in \mathbb{R}$

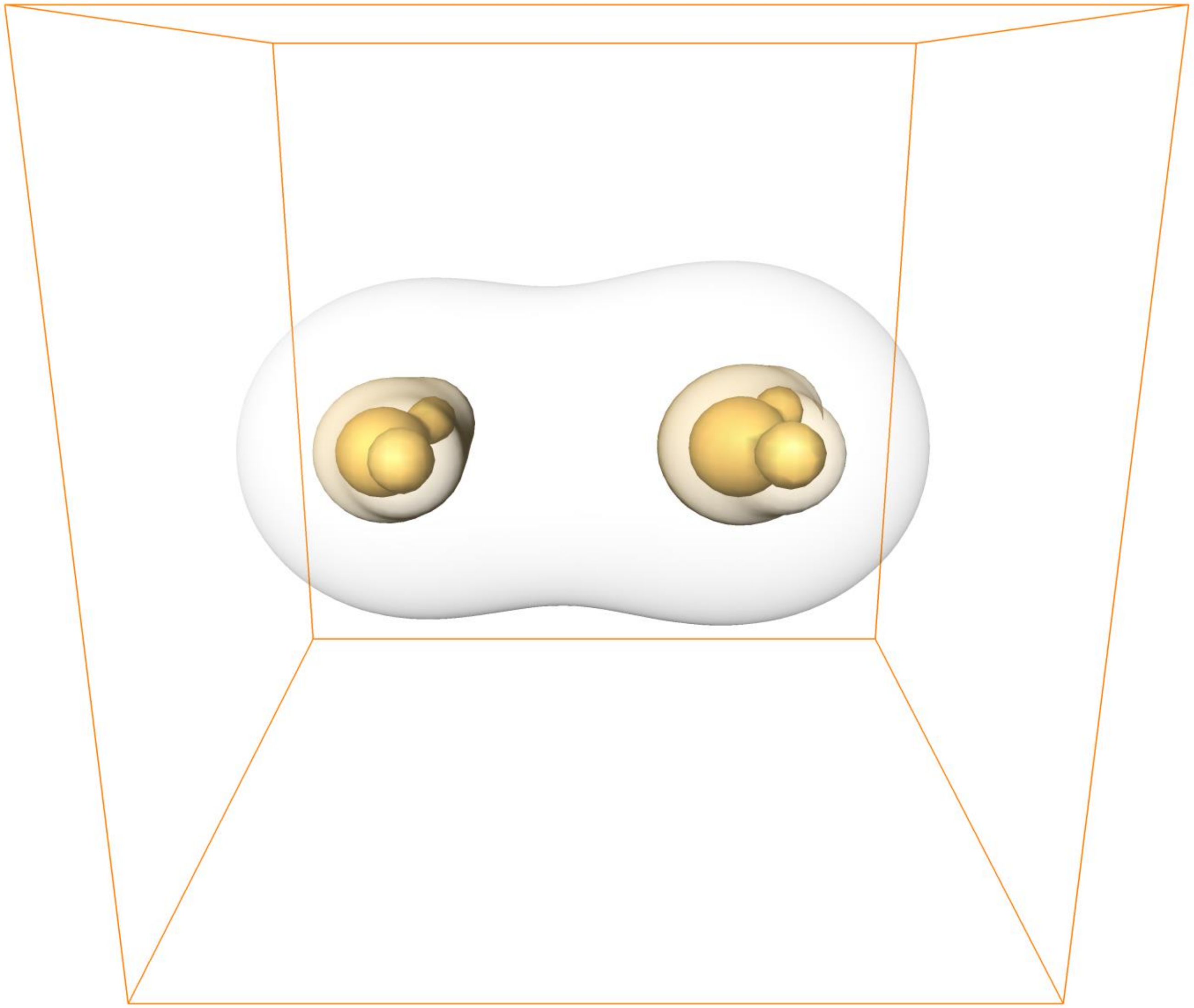
definition of **3D contour**:

$$\{(x, y, z) \mid f(x, y, z) = c\}$$

3D contours are surfaces

if f is differentiable and $\nabla f \neq \mathbf{0}$

common name: **isosurfaces**



Properties of Isosurfaces

closed surfaces

unless exiting the domain

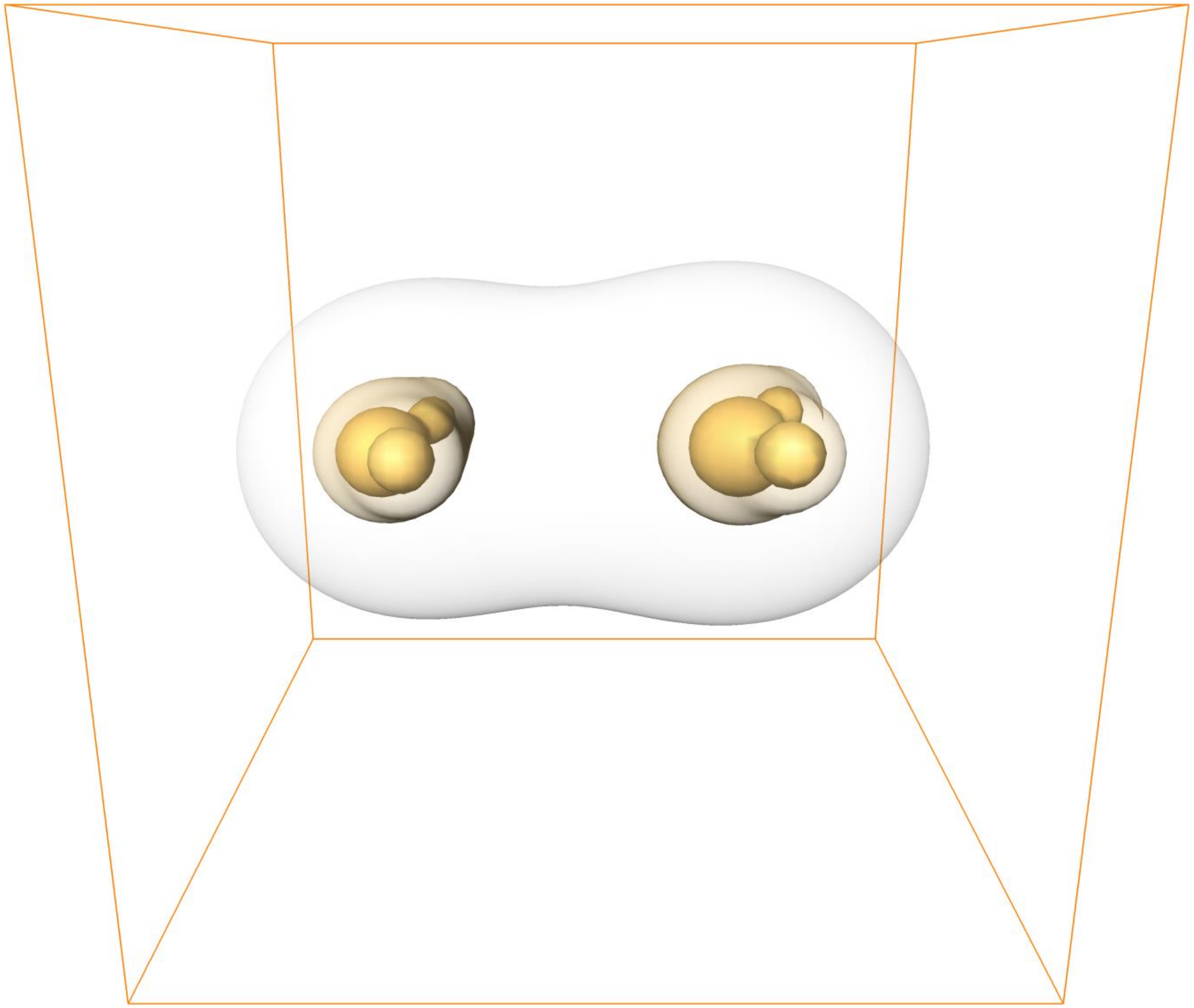
tunnels may occur

cannot intersect each other

nested surfaces

points on isosurfaces have similar semantics

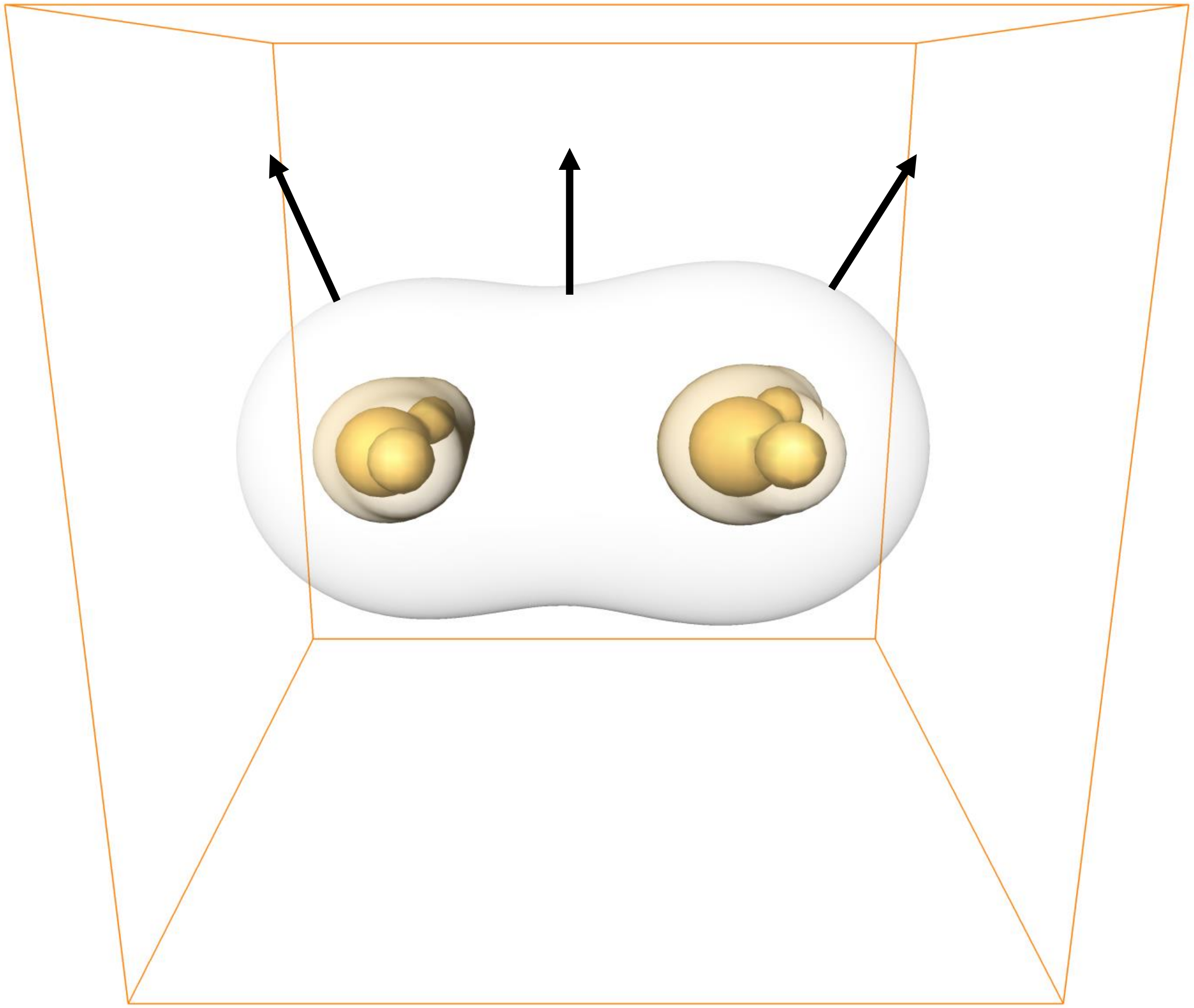
density of the surfaces reveals strength of the gradient



Properties of Isosurfaces

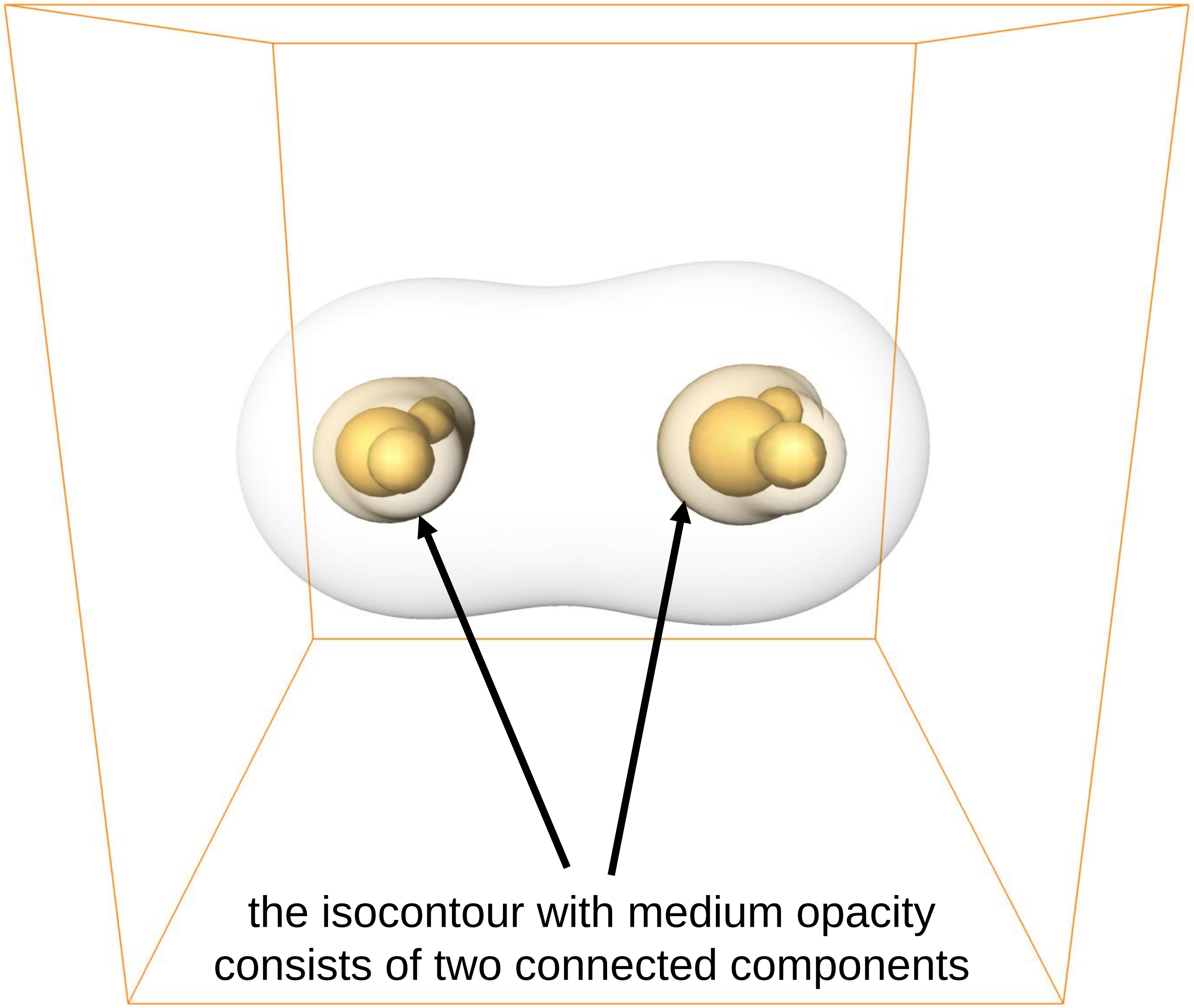
gradient is perpendicular to the isosurface

rate of change is zero along any isocontour



Properties of Isosurfaces

connected component:
a given isovalue produces one isocontour often consisting of several separate surfaces



very important

Isosurface Extraction

input:

- data array
- isovalue c

output:

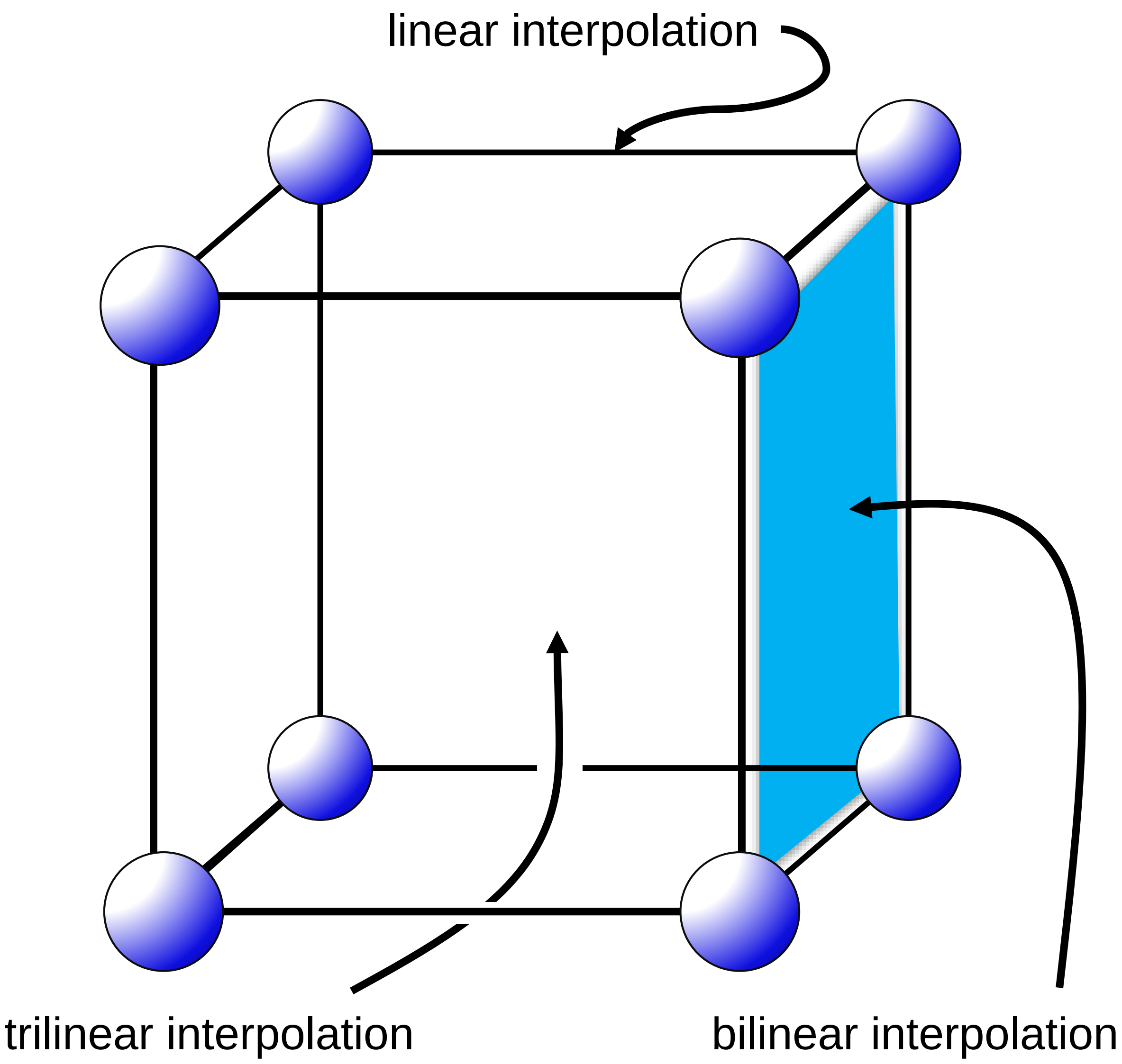
- triangles per grid cell

assumes trilinear interpolation

linear along grid edges

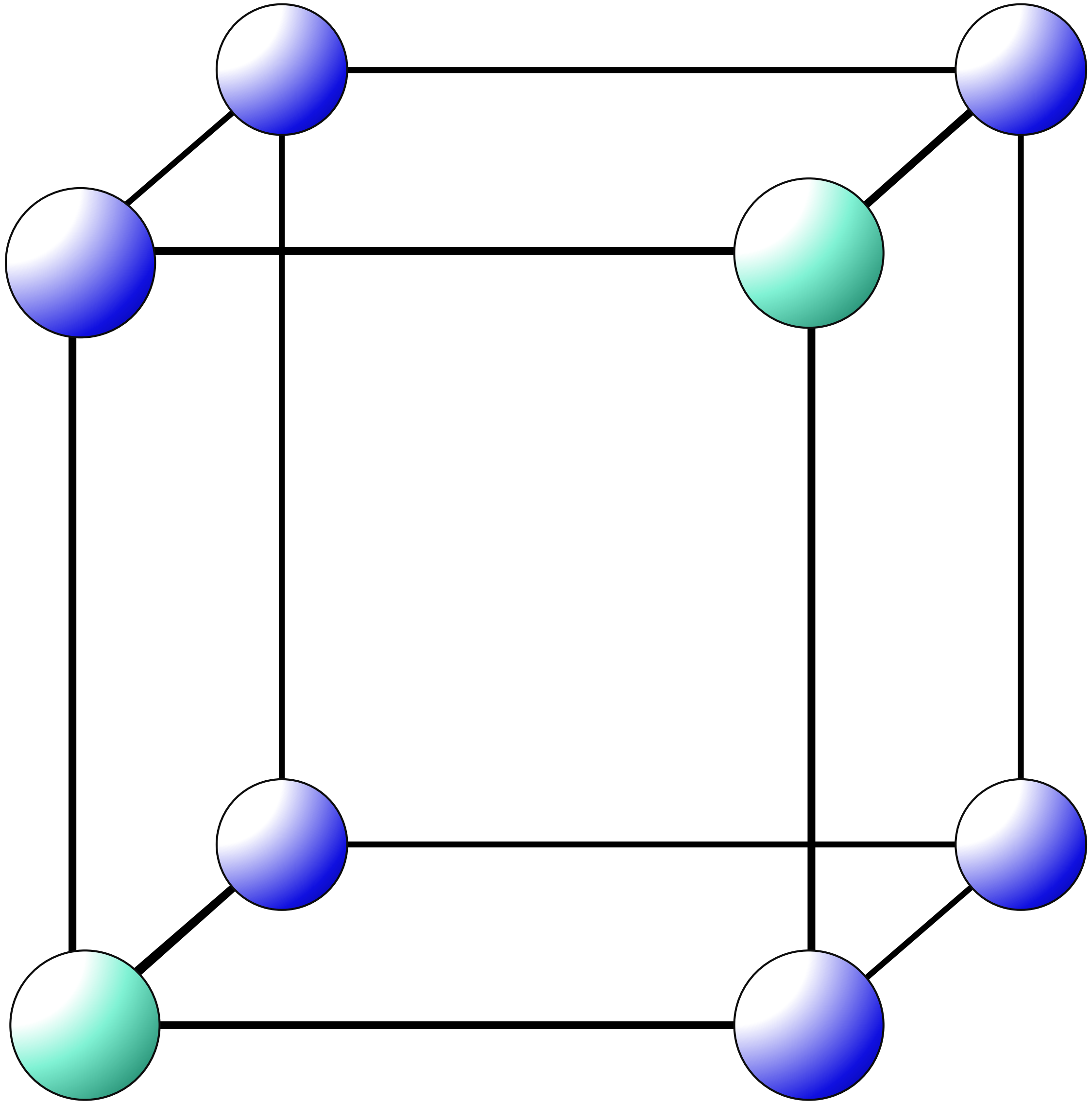
bilinear inside faces

trilinear inside voxel



Isosurface Extraction

- Input: data array and isovalue c
- mark all vertices:
  $+ \Rightarrow f_{i,j,k} \geq c$
  $- \Rightarrow f_{i,j,k} < c$
- tri-/bi-/linear interpolation:
  - isosurface passes only through voxels with different signs at the eight vertices
  - isosurface can only intersect grid faces with different signs at the vertices
  - isosurface can only intersect grid edges with different signs



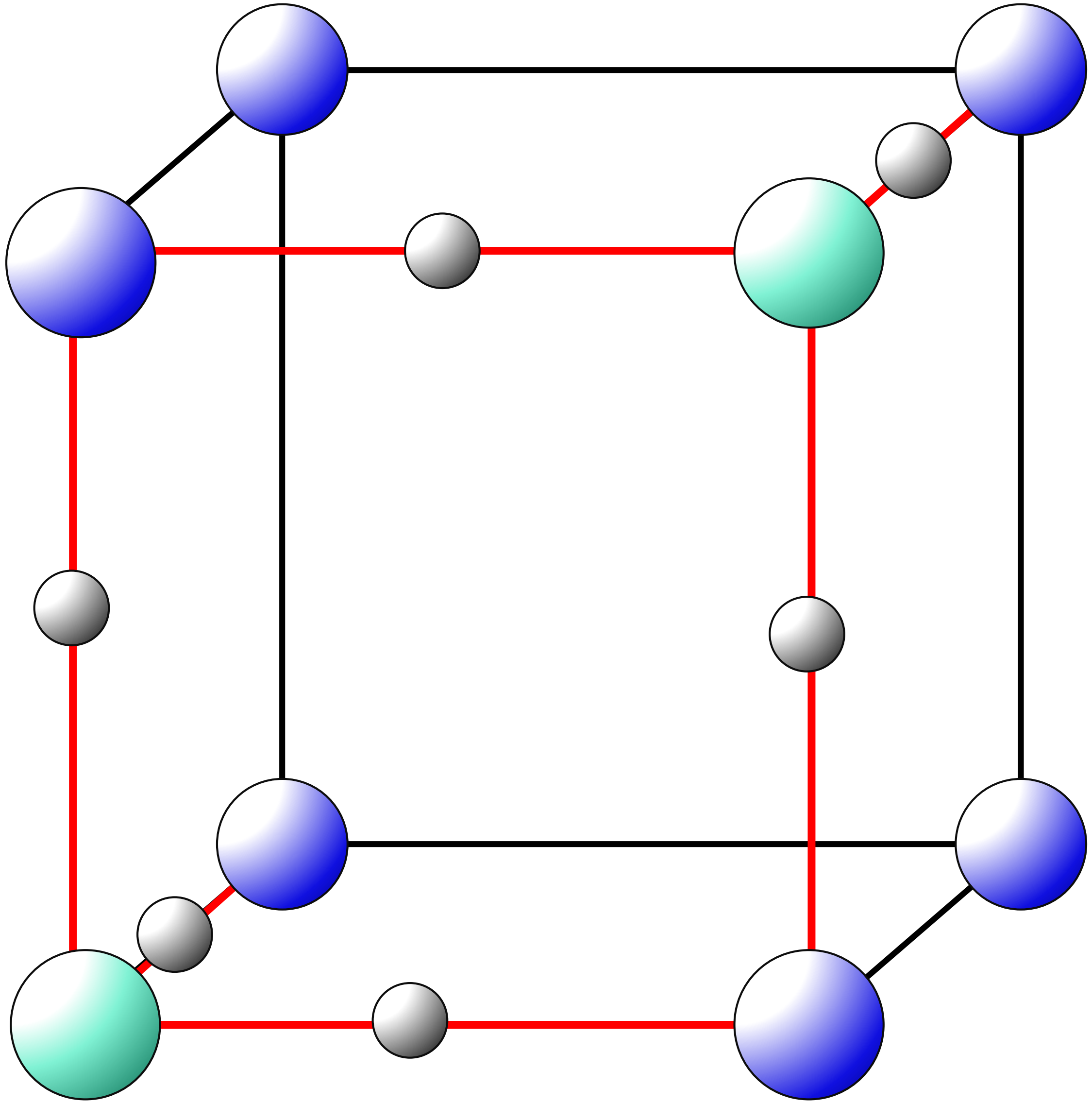
Isosurface Extraction

find edges with intersection

shown in red

compute edge intersections

inverted linear interpolation



Isosurface Extraction

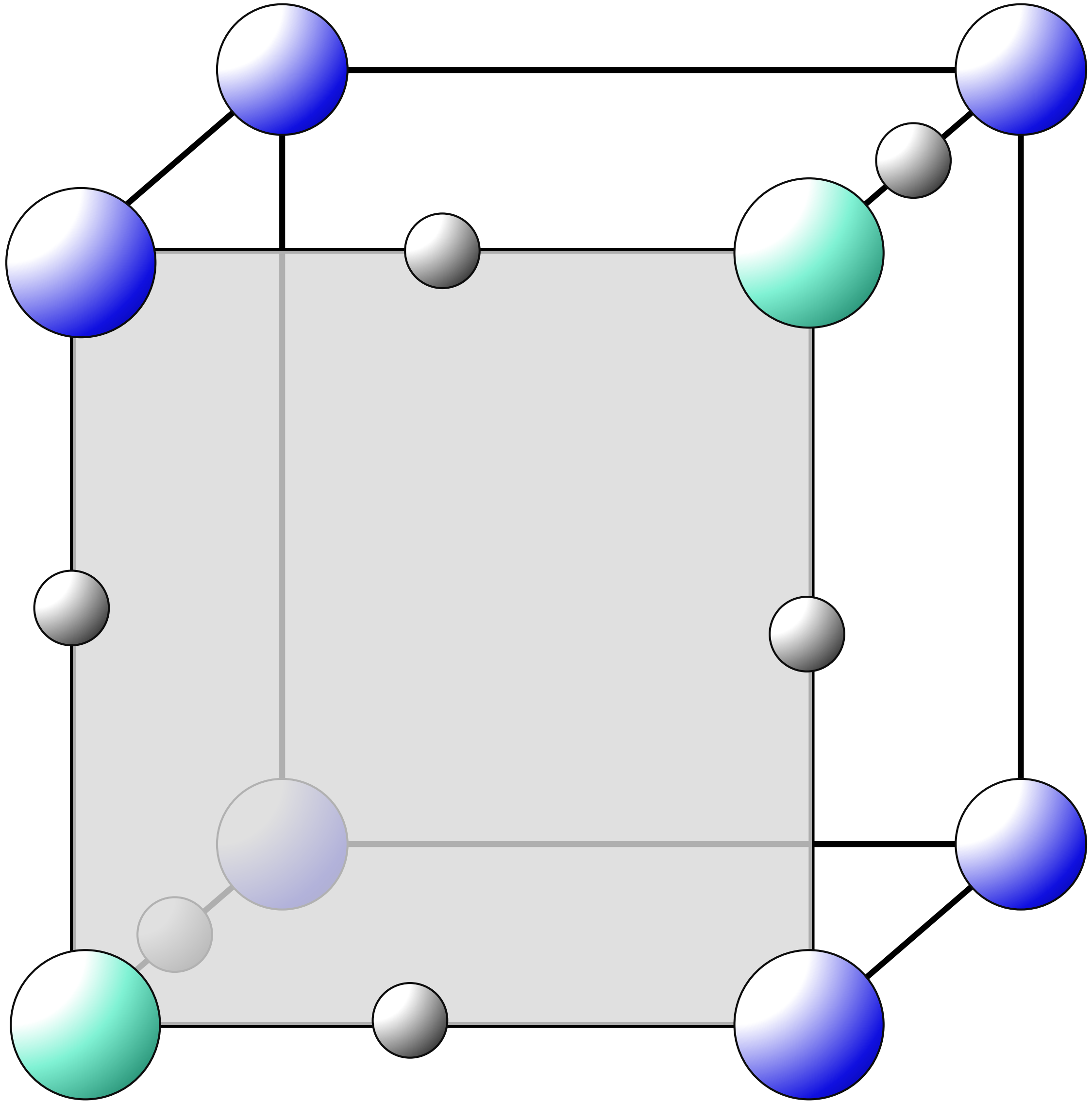
find edges with intersection

compute edge intersections

inverted linear interpolation

connect intersection points on each face

use asymptotic decider



Isosurface Extraction

find edges with intersection

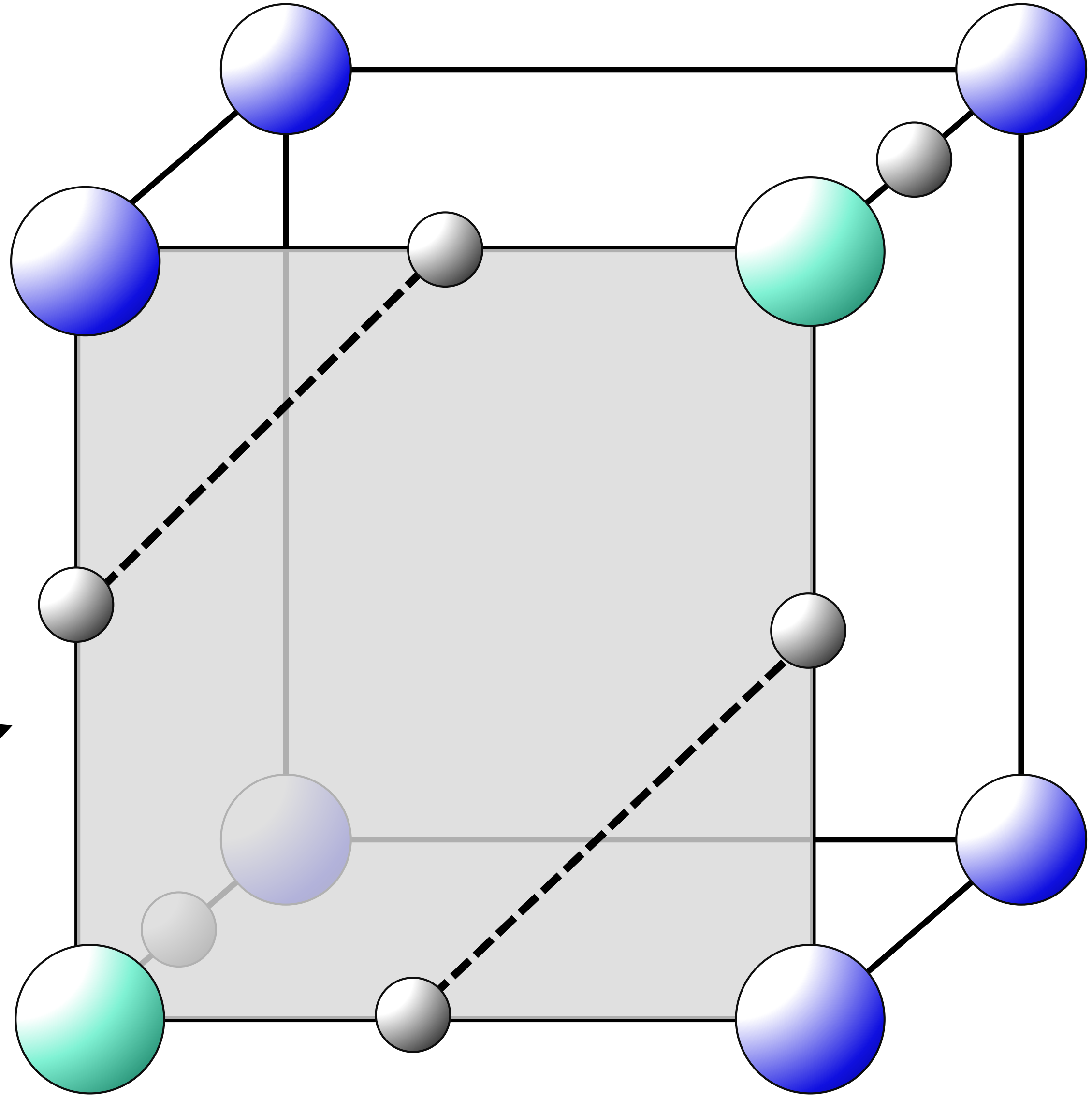
compute edge intersections

inverted linear interpolation

connect intersection points on each face

use asymptotic decider

A possible result of the asymptotic decider



Isosurface Extraction

find edges with intersection

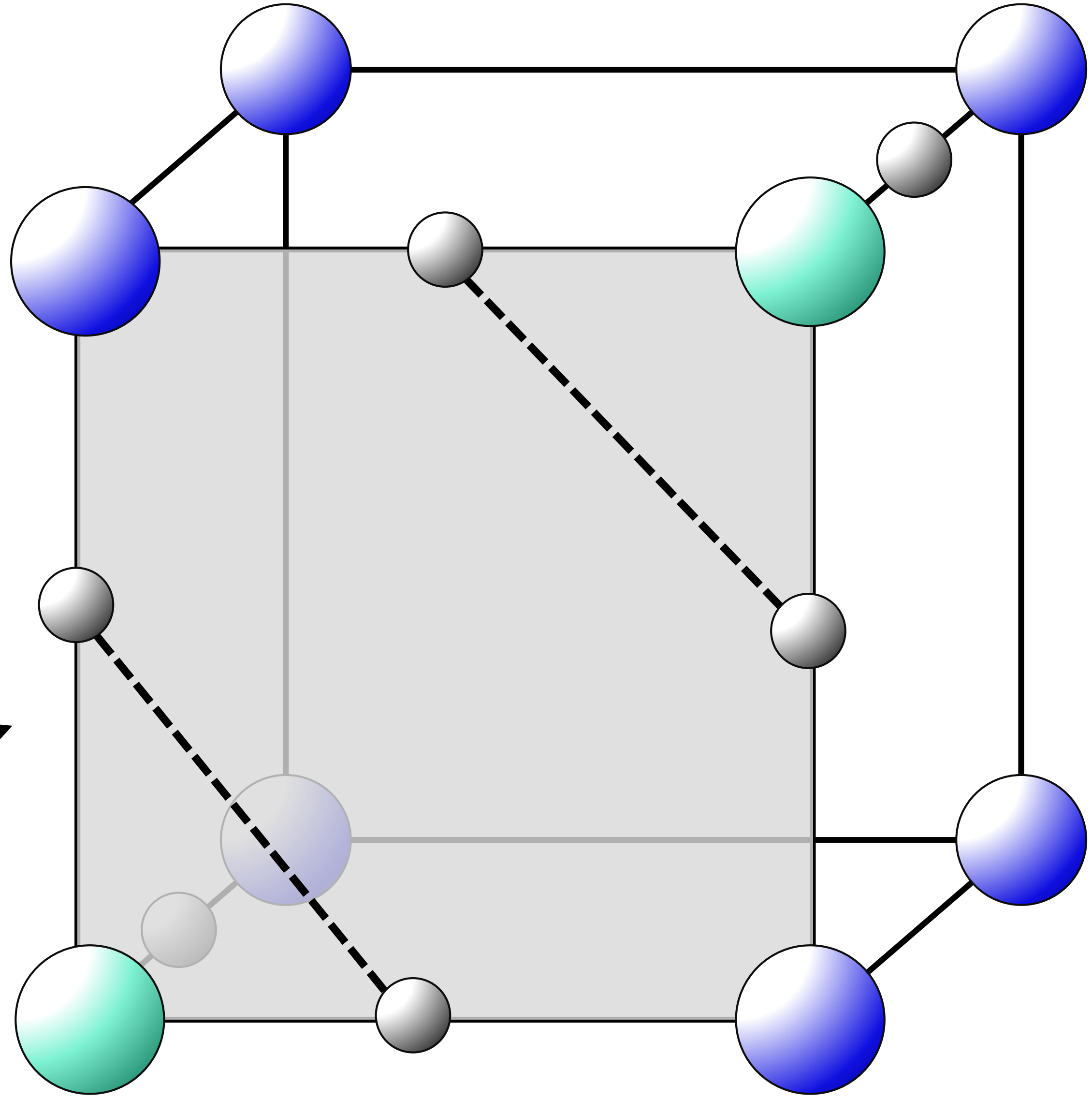
compute edge intersections

inverted linear interpolation

connect intersection points on each face

use asymptotic decider

Another possible result of the asymptotic decider



Isosurface Extraction

find edges with intersection

compute edge intersections

inverted linear interpolation

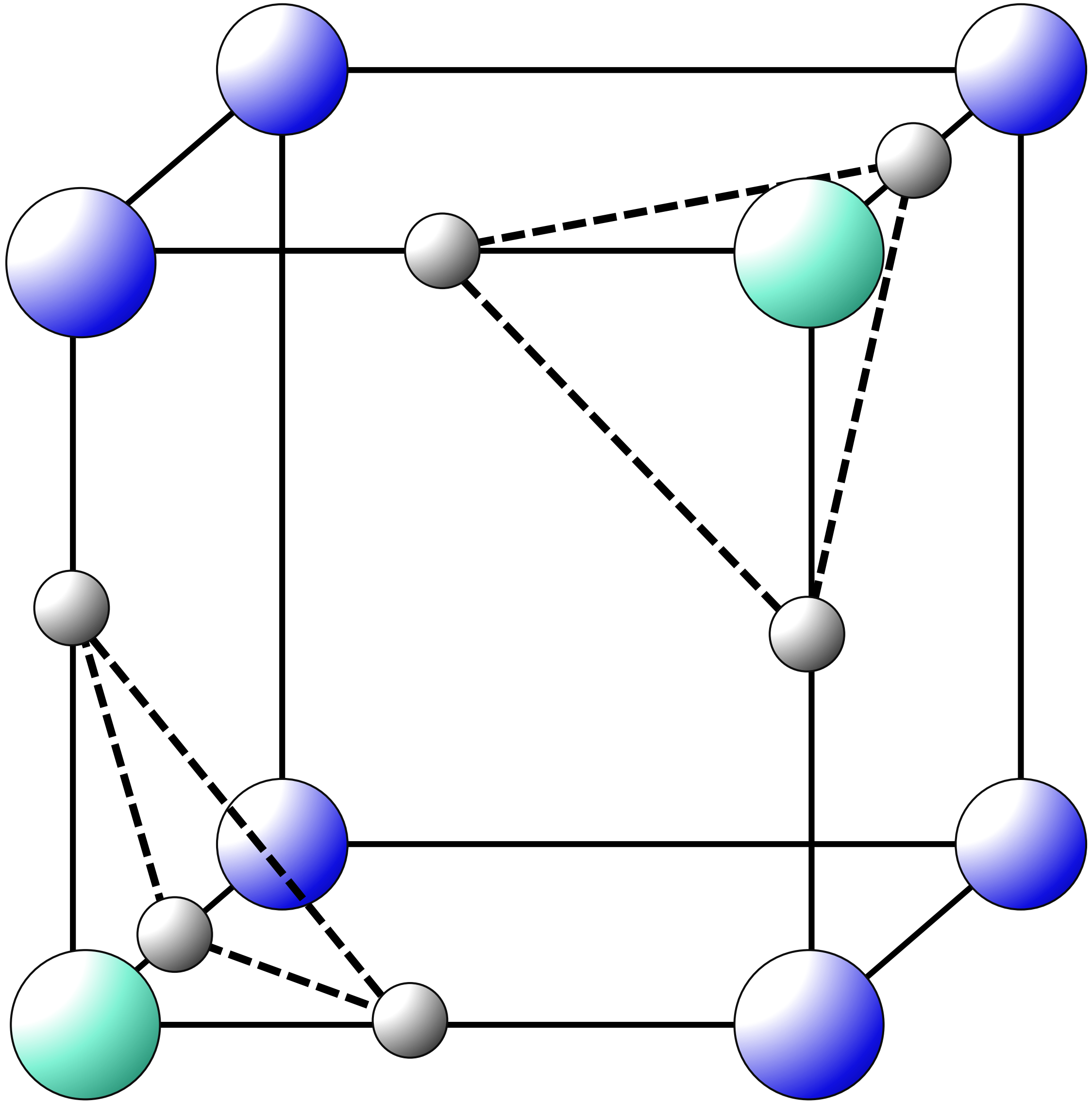
connect intersection points on each face

use asymptotic decider

establish connected components

*follow lines on the face*s

this ignores topology inside voxel



Isosurface Extraction

find edges with intersection

compute edge intersections

inverted linear interpolation

connect intersection points on each face

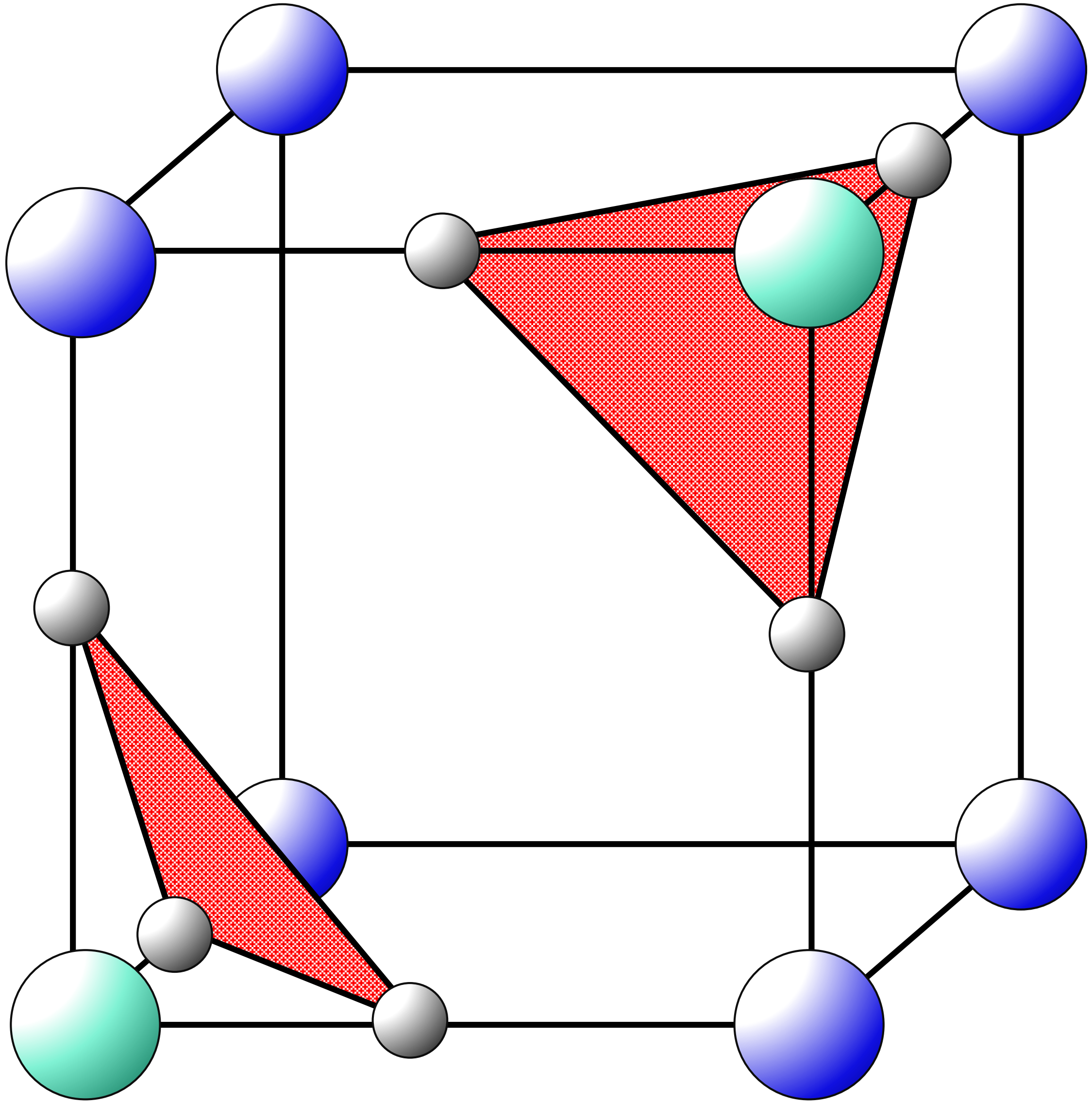
use asymptotic decider

establish connected components

follow lines on the faces

this ignores topology inside voxel

triangulate connected components



Isosurface Extraction

find edges with intersection

compute edge intersections

inverted linear interpolation

connect intersection points on each face

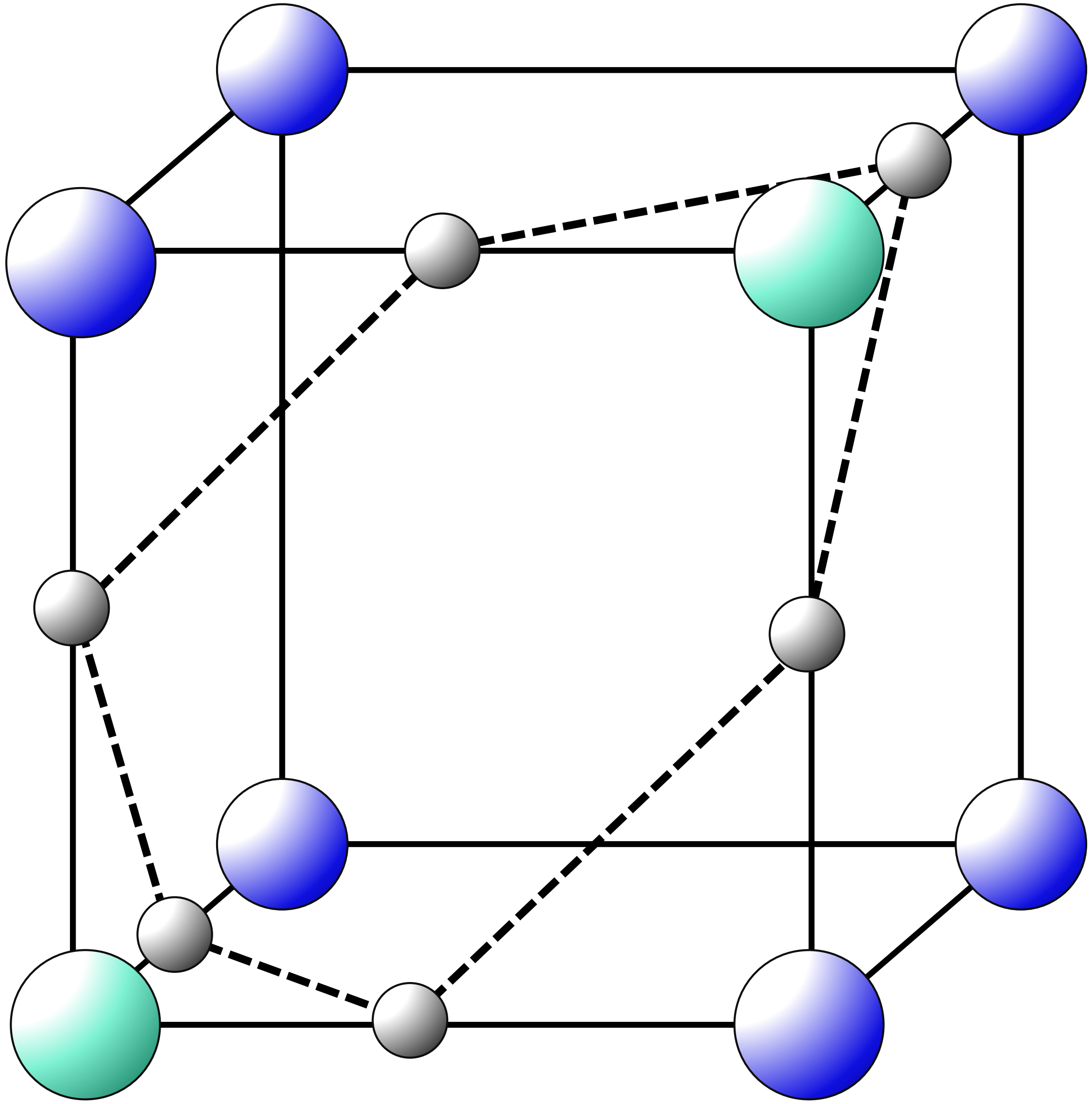
use asymptotic decider

establish connected components

follow lines on the faces

this ignores topology inside voxel

triangulate connected components



Isosurface Extraction

find edges with intersection

compute edge intersections

inverted linear interpolation

connect intersection points on each face

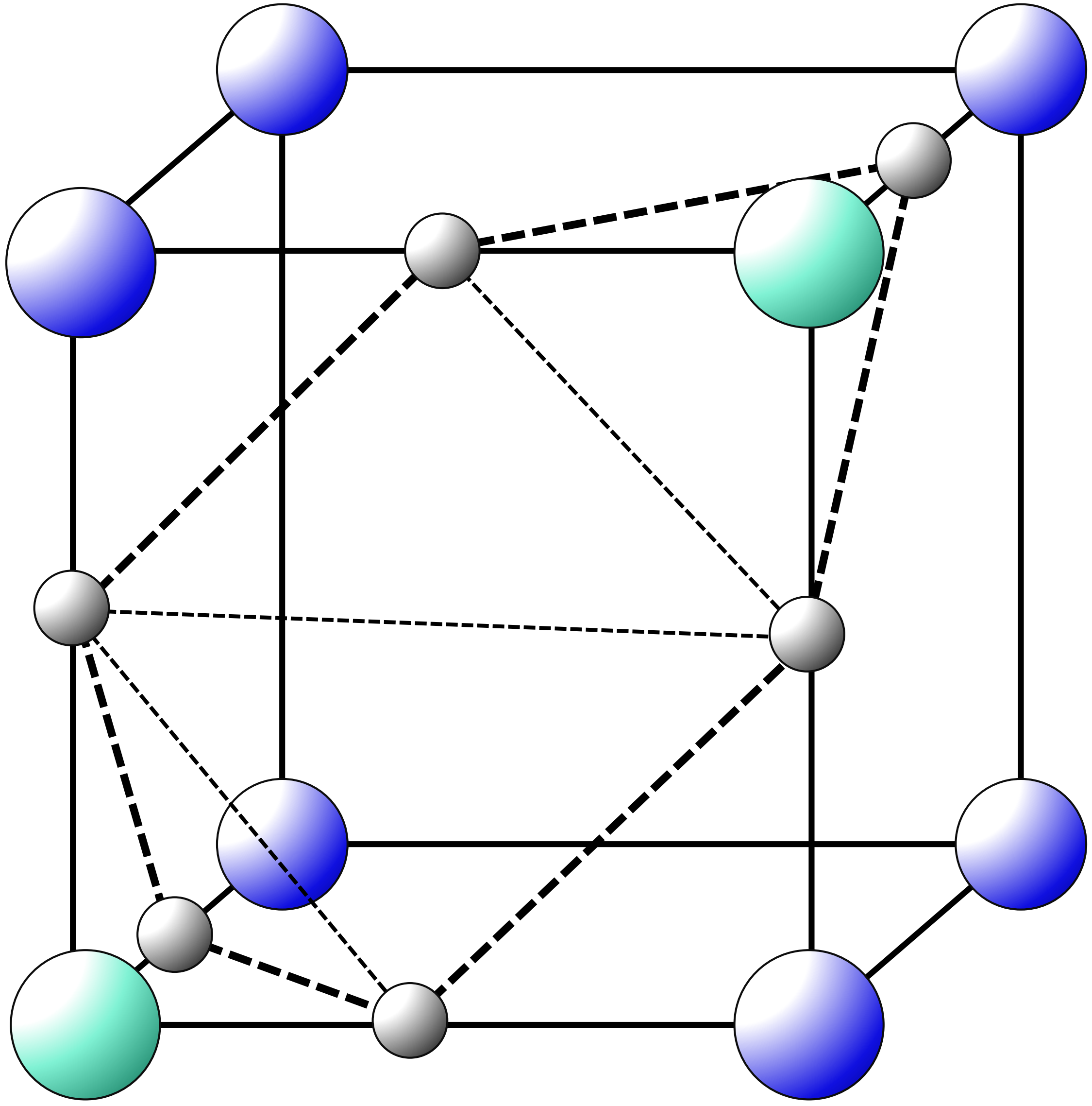
use asymptotic decider

establish connected components

follow lines on the faces

this ignores topology inside voxel

triangulate connected components



Isosurface Extraction

find edges with intersection

compute edge intersections

inverted linear interpolation

connect intersection points on each face

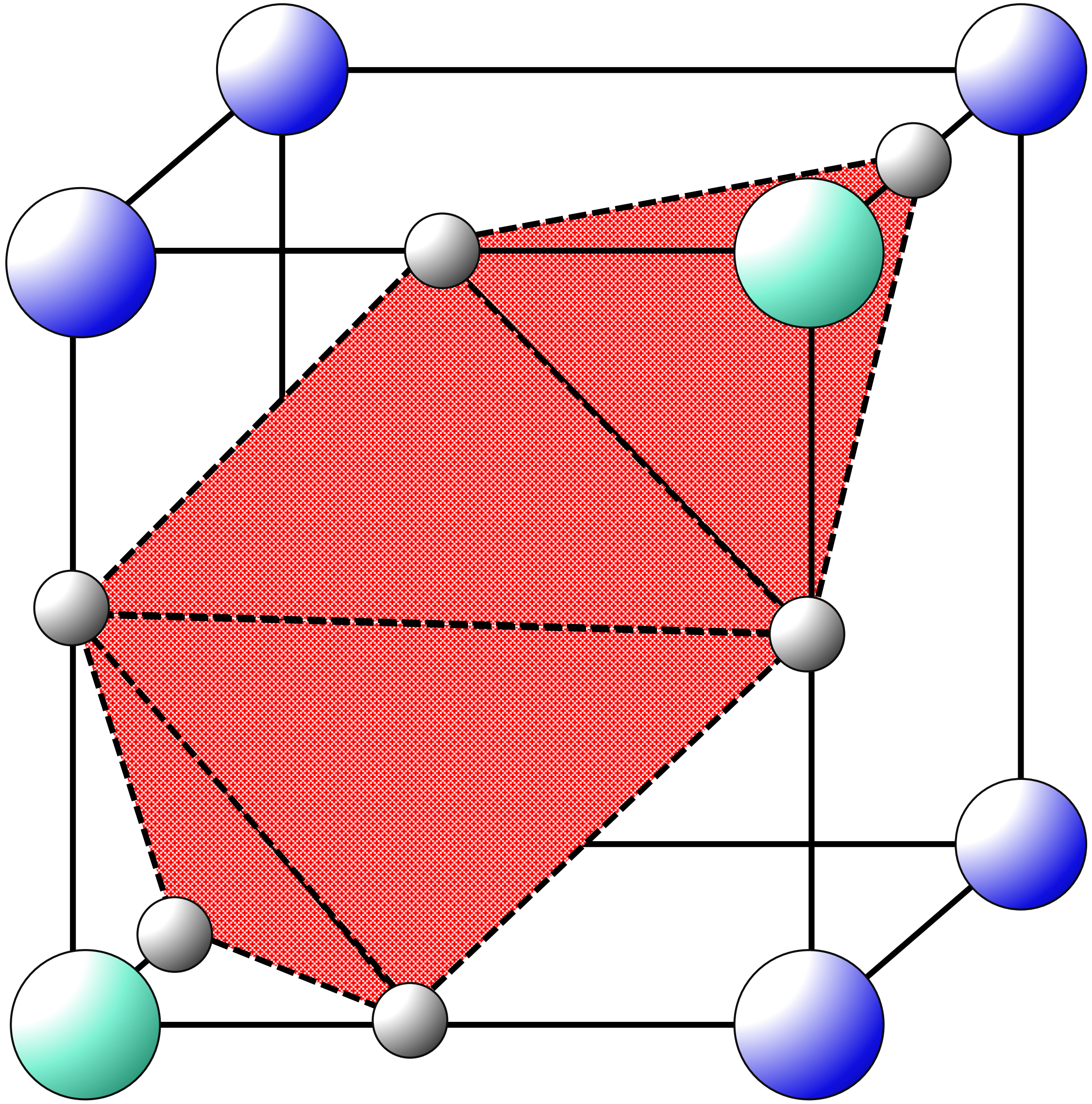
use asymptotic decider

establish connected components

follow lines on the faces

this ignores topology inside voxel

triangulate connected components



Isosurface Extraction

find edges with intersection

compute edge intersections

inverted linear interpolation

connect intersection points on each face

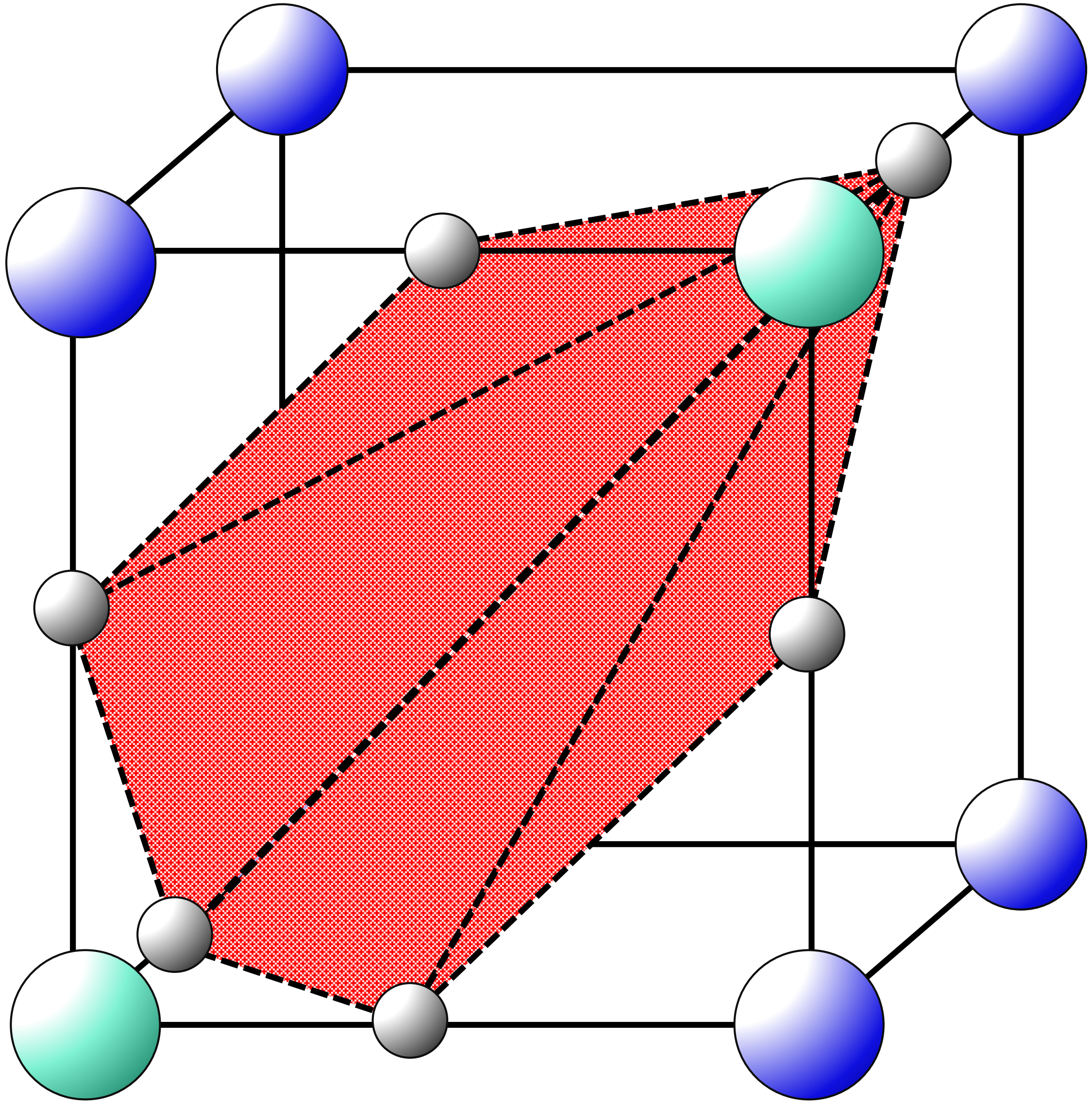
use asymptotic decider

establish connected components

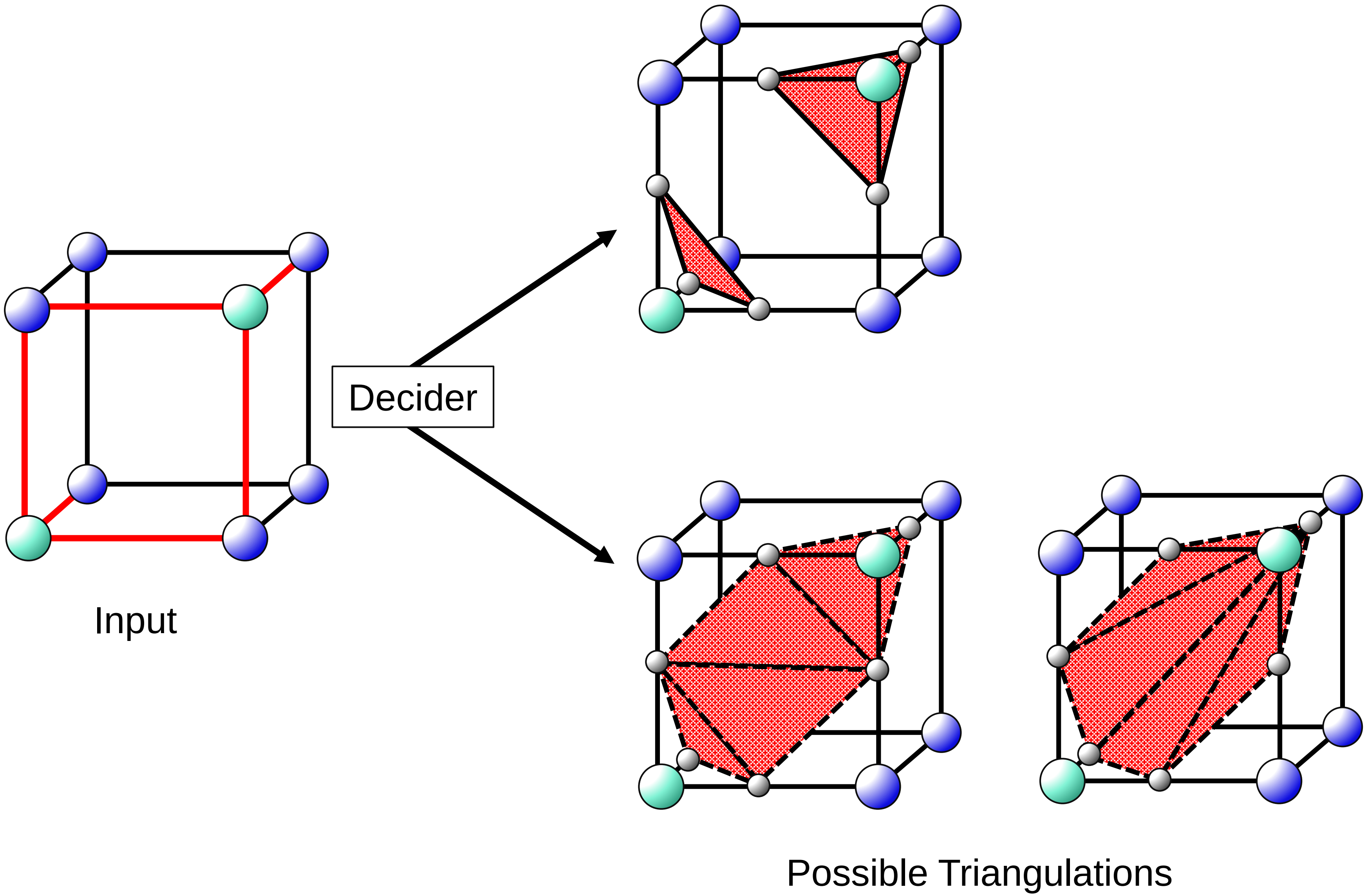
follow lines on the faces

this ignores topology inside voxel

triangulate connected components



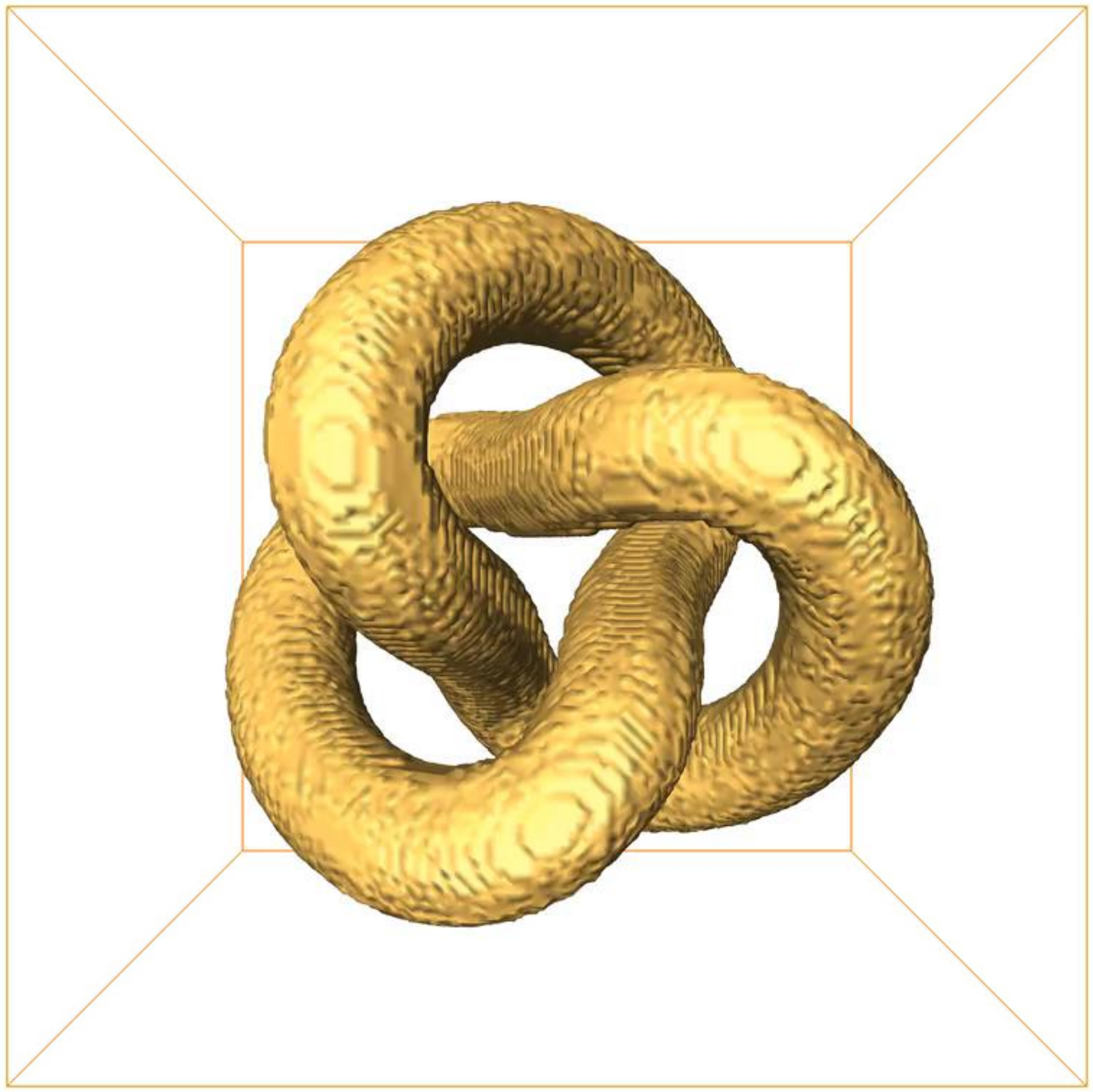
Isosurface Extraction: Overview



Isosurface Rendering

Use gradient of scalar field for the normals of the triangle mesh, since the gradient is perpendicular to the isosurfaces

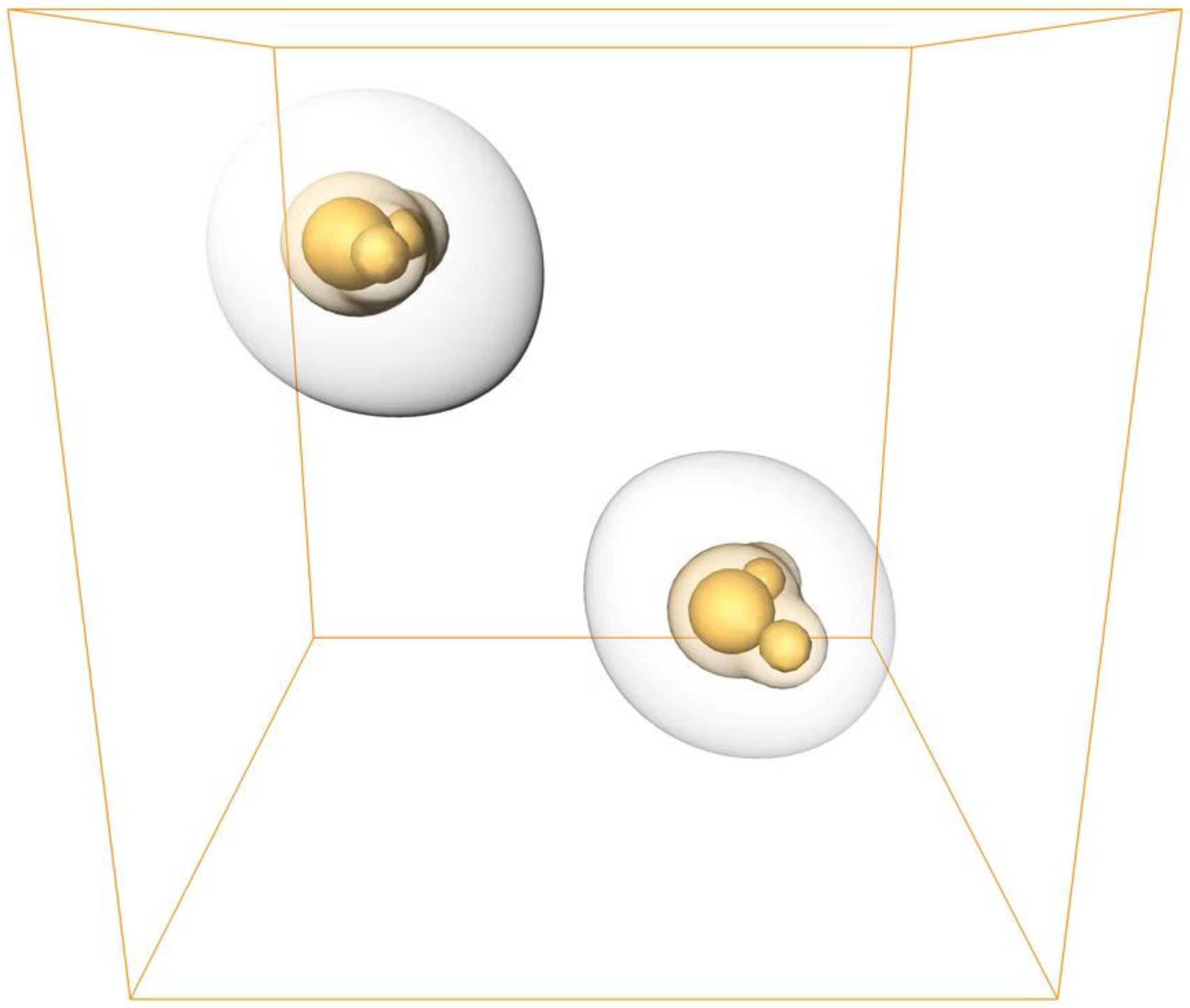
Higher-order derivatives pay off, since the human eye is very sensitive to lighting discontinuities



Trefoil data set from Candelaresi et al., Decay of trefoil and other magnetic knots. In Proc. Advances in Plasma Astrophysics, 2011

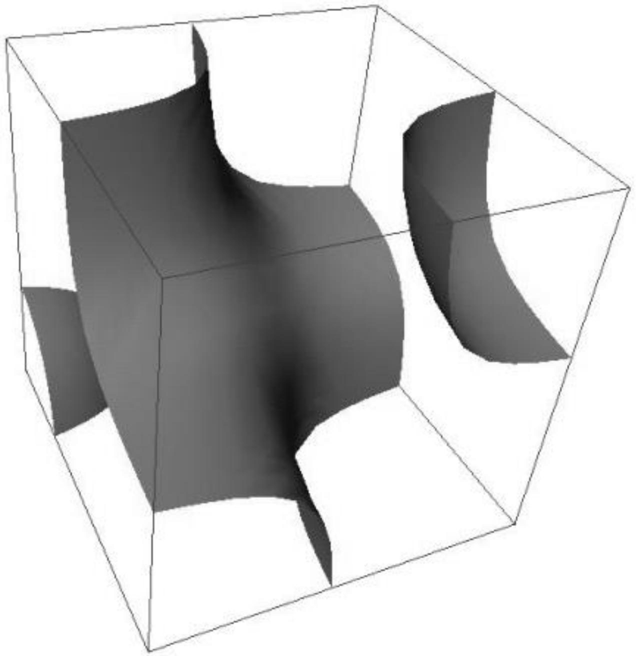
Isosurface Rendering

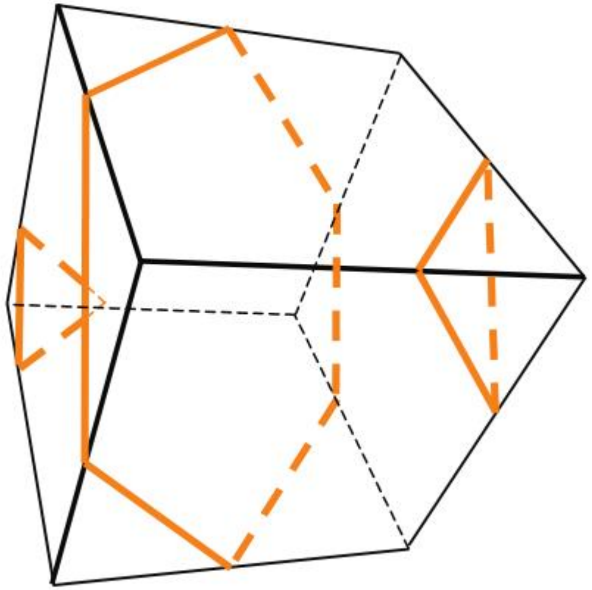
One can show several nested isosurfaces using varying levels of opacity

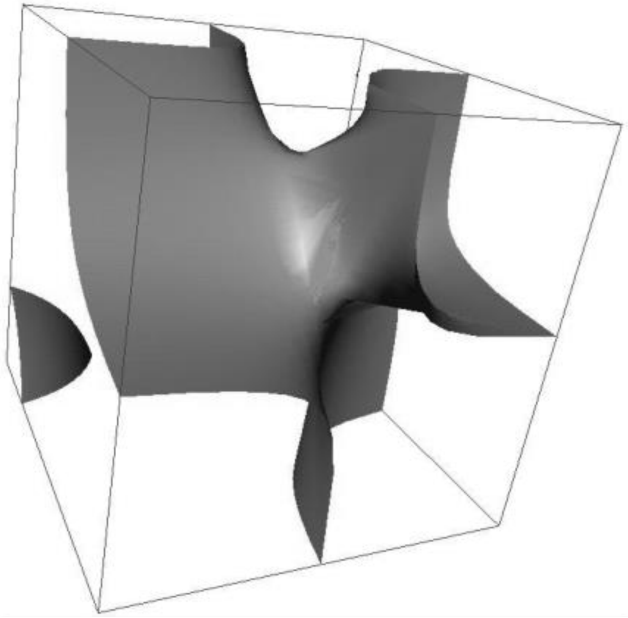


Inside the Trilinear Cell

possible isosurfaces
with the same
footprint on the boundary



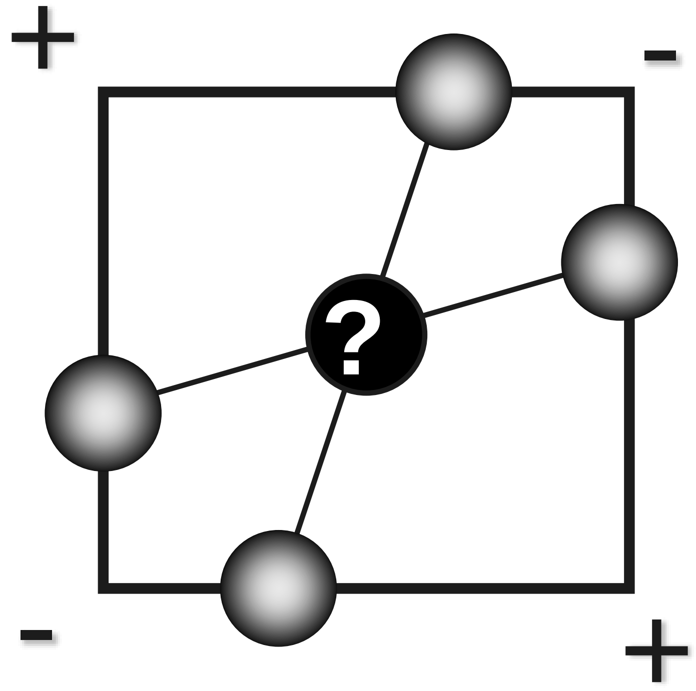


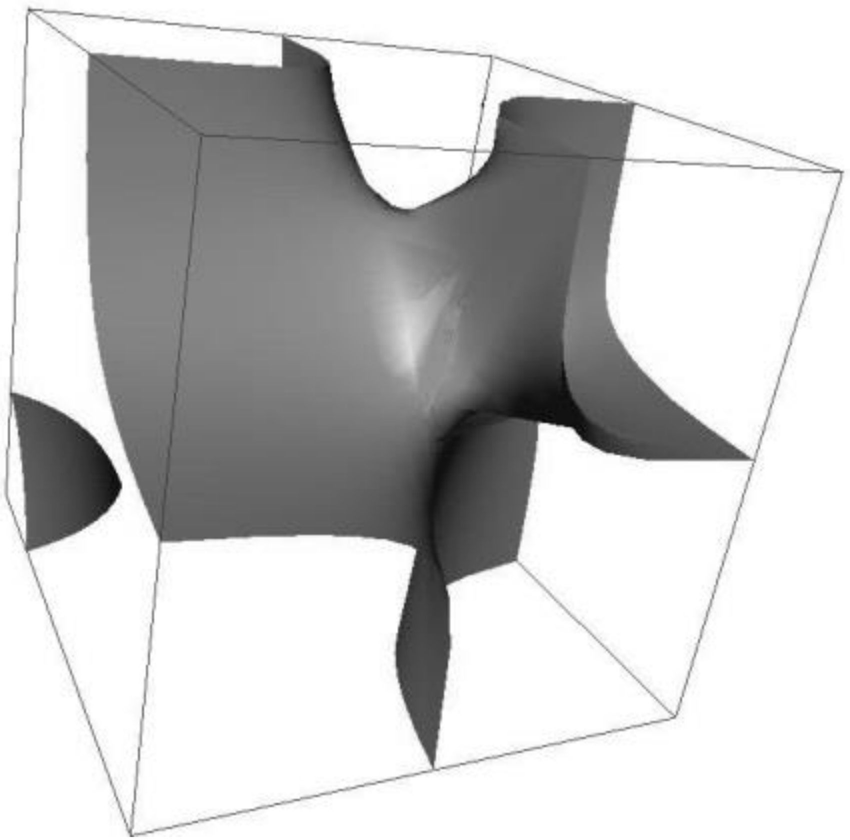


Details:
Habilitation Thesis of Holger Theisel
University of Rostock, 2001

Inside the Cell: Does it matter?

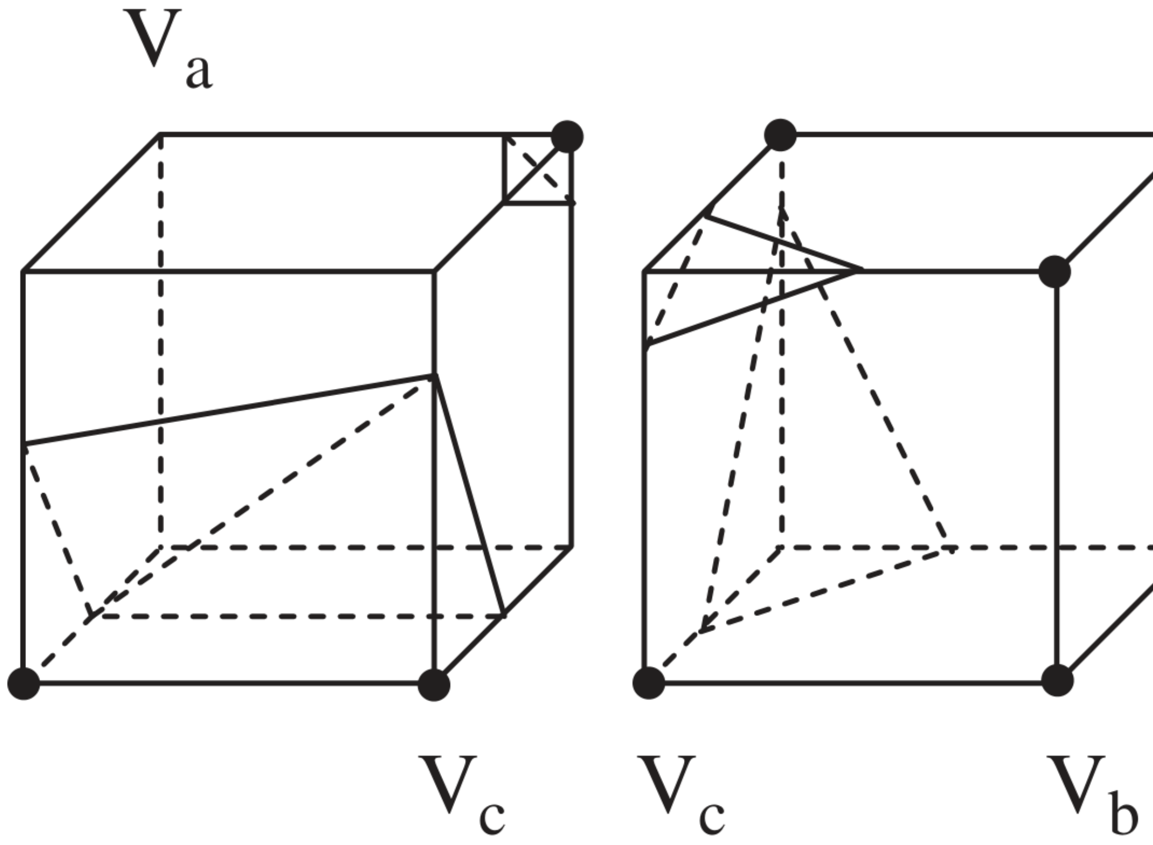
These details are below the sampling resolution of the data set.



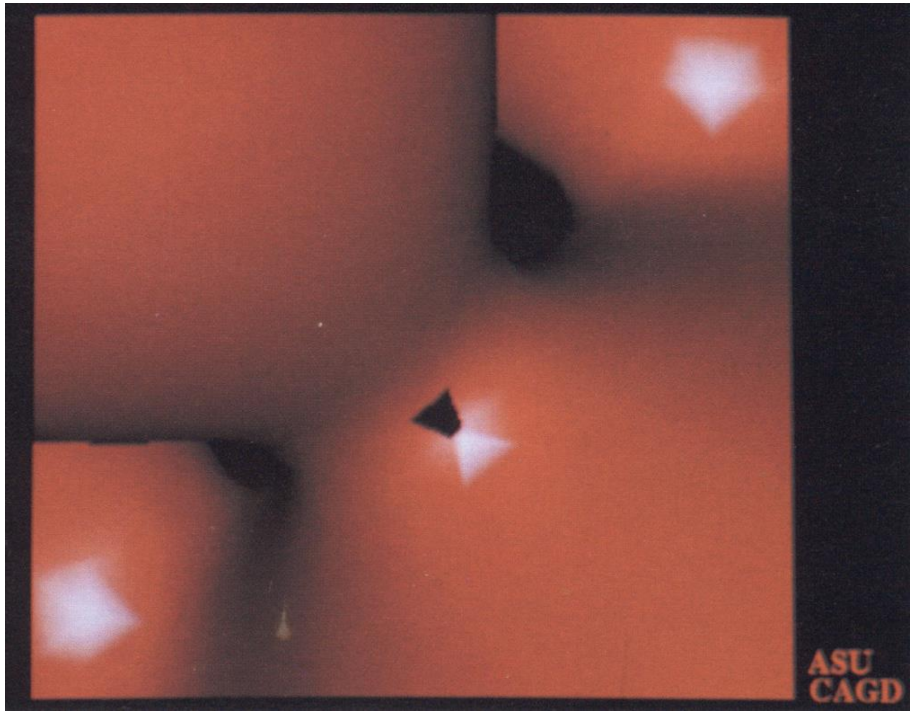


Between the Cells: Does it matter?

Important to not create arbitrary holes in the isosurface.



from Newman & Yi, A Survey of the Marching Cubes Algorithm, Computers & Graphics, 2006



from Nielsen & Hamann, The Asymptotic Decider, IEEE Vis 1991

very important

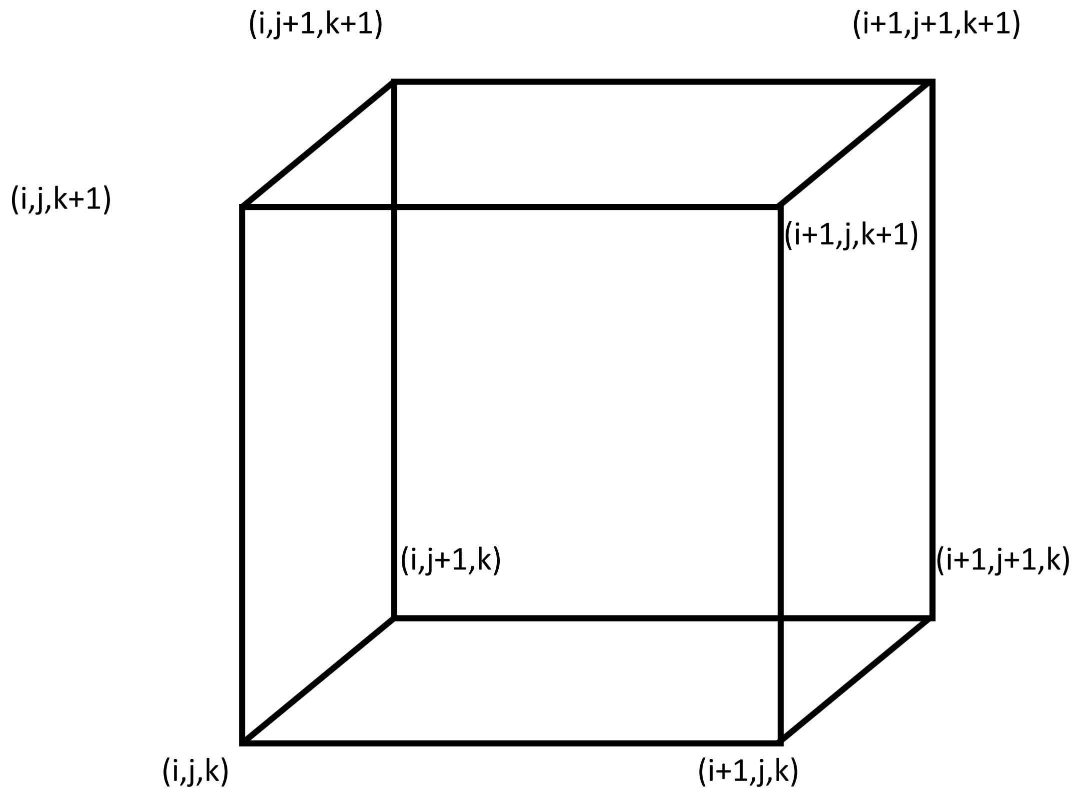
The Marching Cubes (MC) algorithm

- Invented by Lorensen & Cline 1987
  Ambiguities fixed by Nielson & Hamann 1991
- Addons, fixes, enhancements, history: Newman & Yi, A Survey of the Marching Cubes Algorithm, Computers & Graphics, 2006
- Approximates the surface using a triangle mesh; surface is found by linear interpolation along cell edges
- Triangulation using lookup tables
- Patented in the US 1985 – 2005
- THE standard geometry-based isosurface extraction algorithm

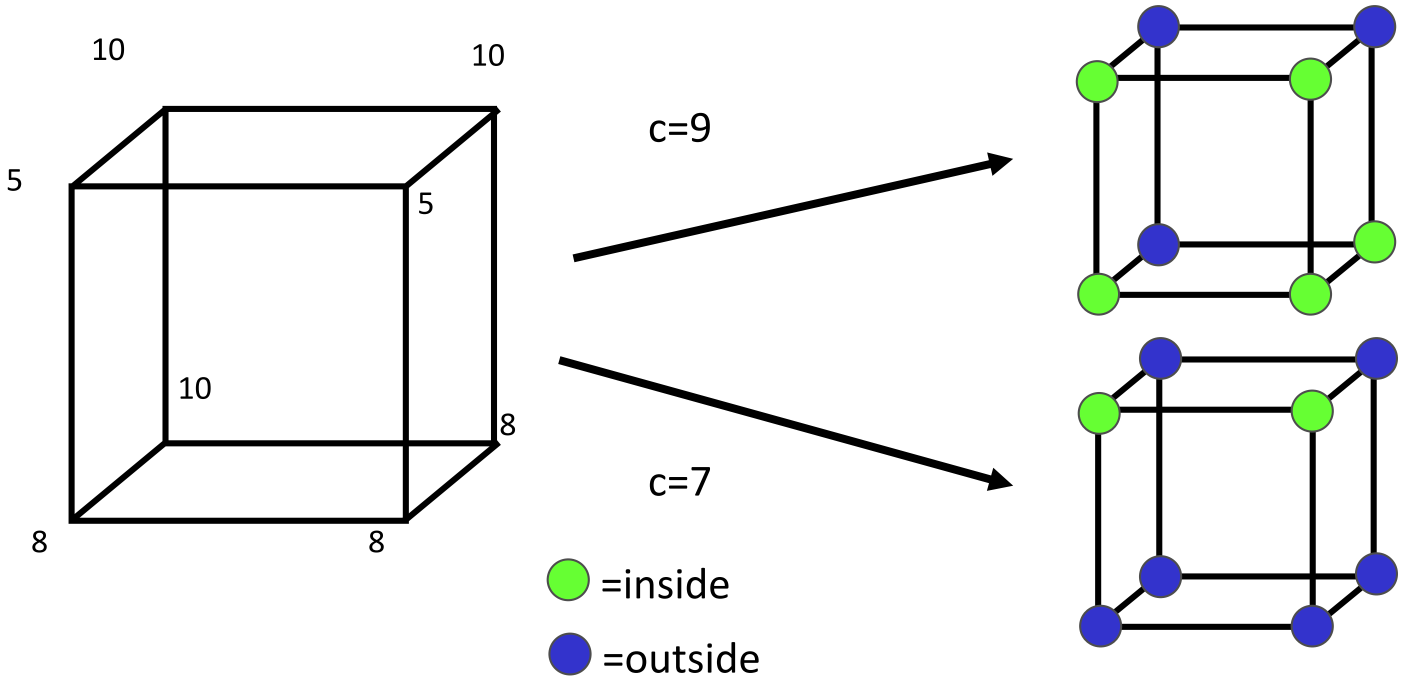
The Marching Cubes (MC) algorithm

1. Consider a cell
2. Classify each vertex as inside or outside
3. Build an index
4. Get per-cell triangulation from table[index]
5. Interpolate the edge location
6. Compute gradients (optional)
7. Consider ambiguous cases
8. Go to next cell

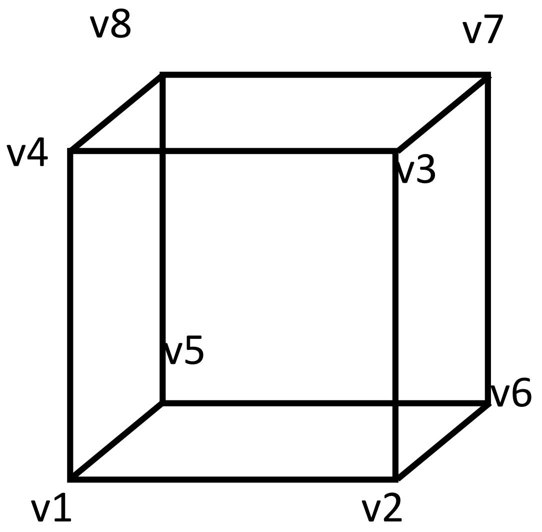
Consider a cell defined by eight data values



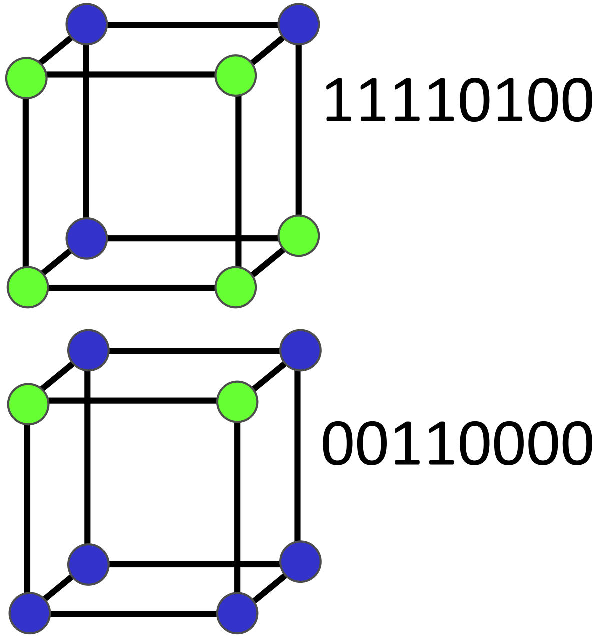
Classify each vertex according to whether it lies
- outside the surface (value > isovalue c)
- inside the surface (value <= isovalue c)



Use the binary labeling of each voxel to create an index



inside = 1
outside = 0



Index:

v1 v2 v3 v4 v5 v6 v7 v8

Marching Cubes: Step 4.1

index

+

lookup-table

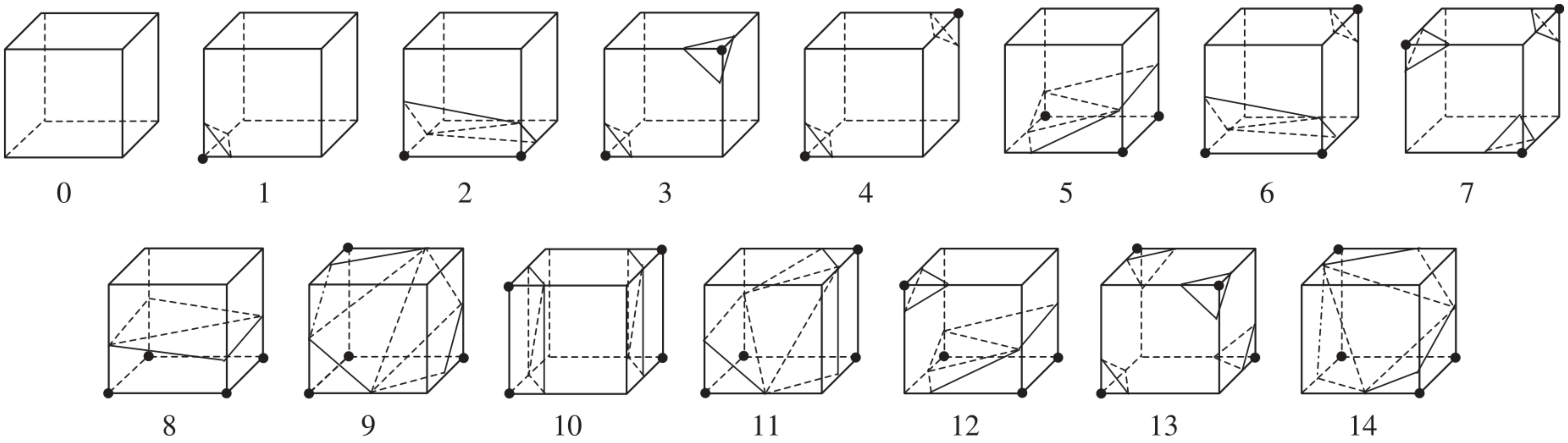
↓

intersected edges

resulting triangles

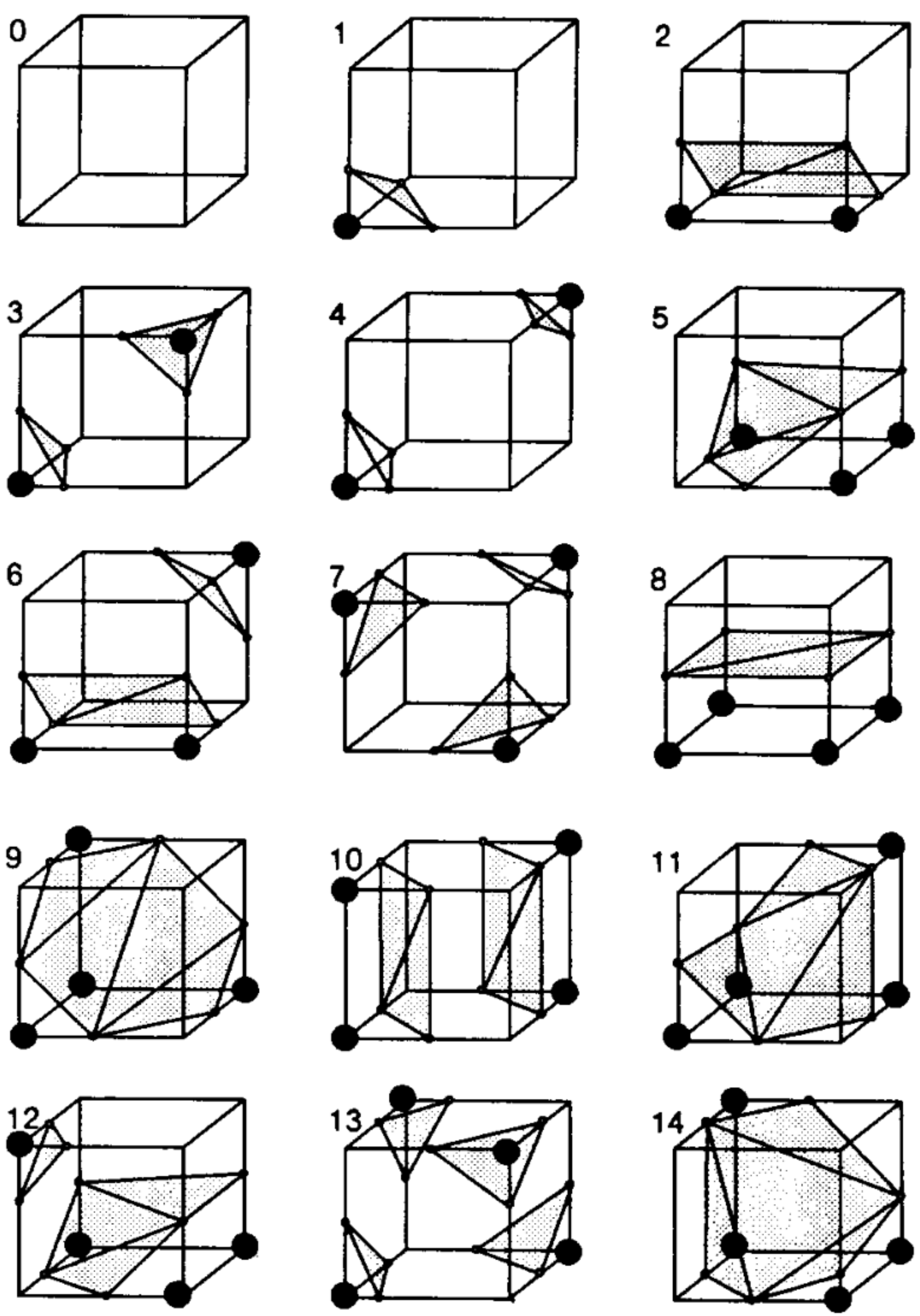
```
typedef struct {
   unsigned char nverts;          /* # vertices above threshold */
   unsigned char verts[8];        /* up to 8 vertices */
   unsigned char nedges;          /* # edges to be intersected */
   unsigned char edges[12];       /* up to 12 edges */
   unsigned char ntris;           /* # triangles to be generated */
   unsigned char tri_edges[15];   /* up to 5 triangles */
} lt;

static const lt LUT[256] =
   {
/*   0    00000000 */      {
    0,    {0, 0, 0, 0, 0, 0, 0, 0},
    0,    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    0,    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
               },
/*   1    00000001 */      {
    1,    {1, 0, 0, 0, 0, 0, 0, 0},
    3,    {1, 4, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    1,    {1, 4, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
               },
/*   2    00000010 */      {
    1,    {2, 0, 0, 0, 0, 0, 0, 0},
    3,    {1,10, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    1,    {1,10, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
               },
/*   3    00000011 */      {
    2,    {1, 2, 0, 0, 0, 0, 0, 0},
    4,    {2, 4, 9,10, 0, 0, 0, 0, 0, 0, 0, 0},
    2,    {2, 9,10, 2, 4, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0},
               },
```

For a given index, access an array storing a list of triangles and edges on which their vertices lie

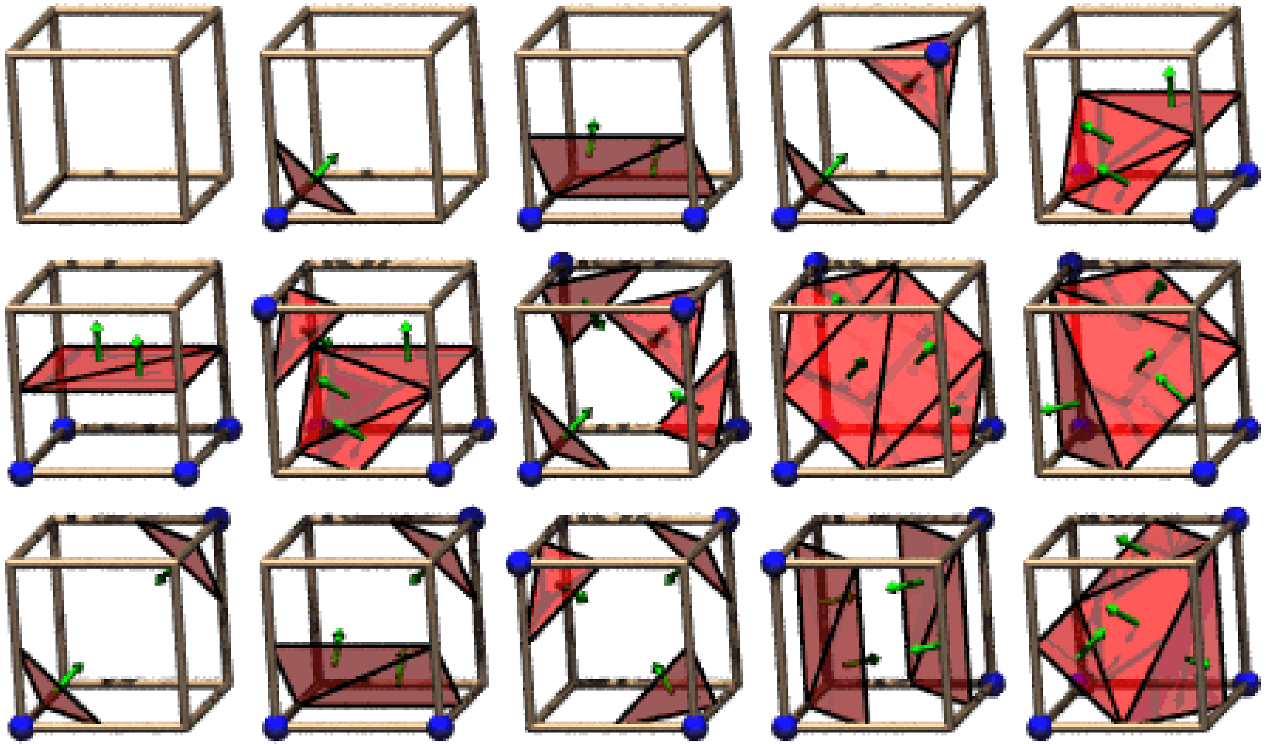


All 256 cases can be derived from 15 base cases due to symmetries

Image from Newman & Yi, A Survey of the Marching Cubes Algorithm, Computers & Graphics, 2006 original image from the MC paper for the triangulations



alternative pictures, different ordering



The 15 Cube Combinations

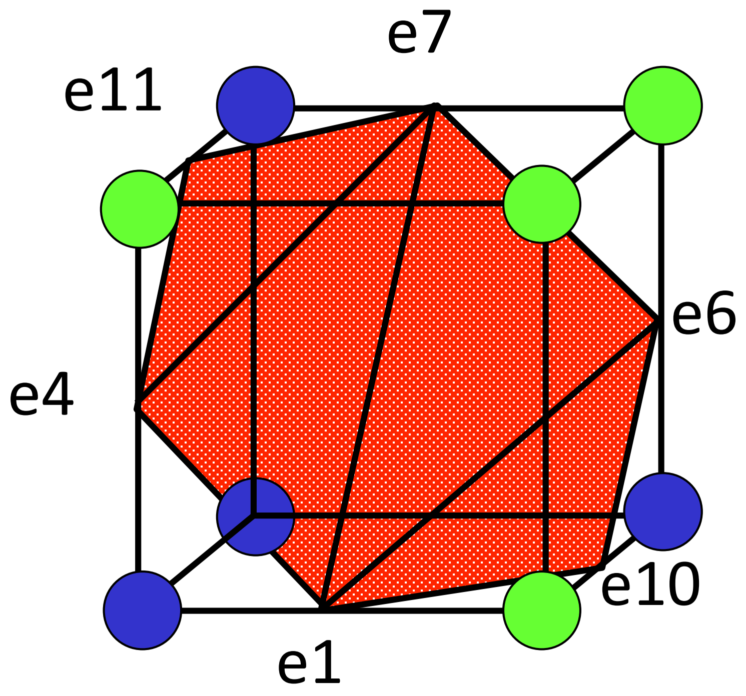
All 256 cases can be derived from 15 base cases due to symmetries

Get edge & triangle list from table

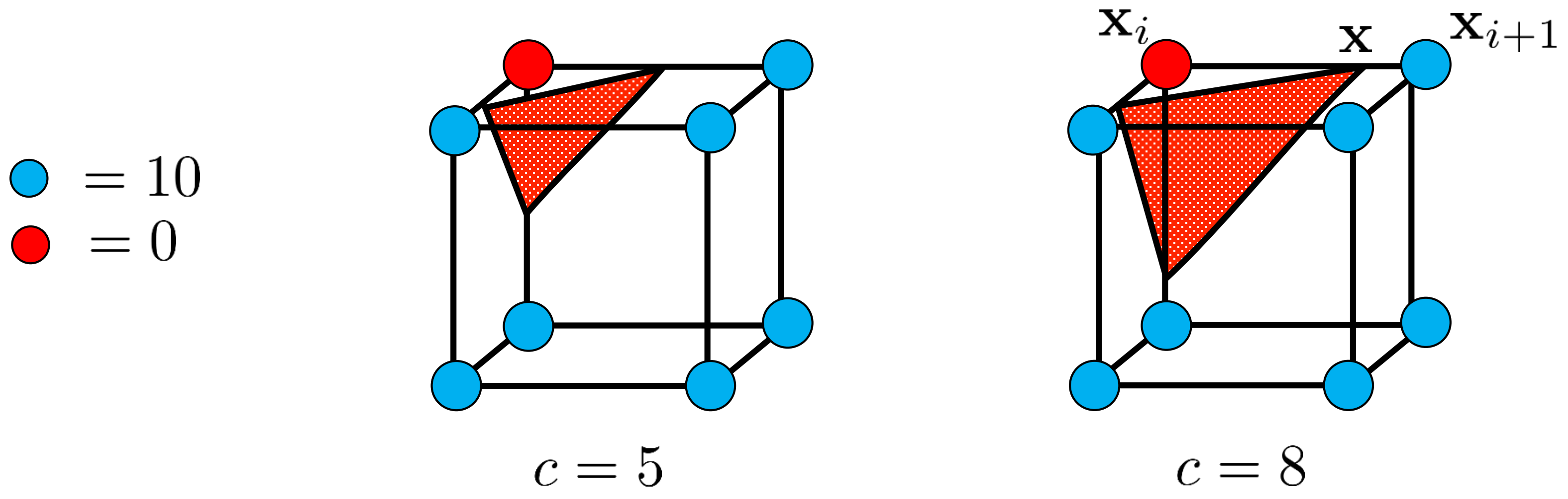
Example for

Index = 01110010

triangle 1 = e4, e7, e11
triangle 2 = e1, e7, e4
triangle 3 = e1, e6, e7
triangle 4 = e1, e10, e6



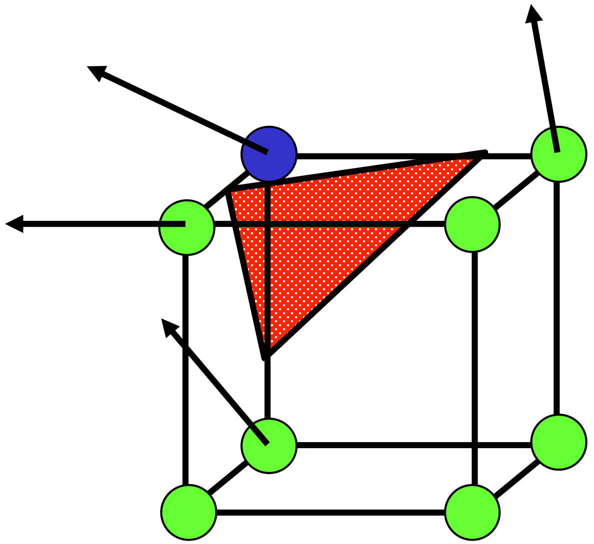
For each triangle edge, find the vertex location along the edge using linear interpolation of the voxel values



Calculate the normal at each cube vertex

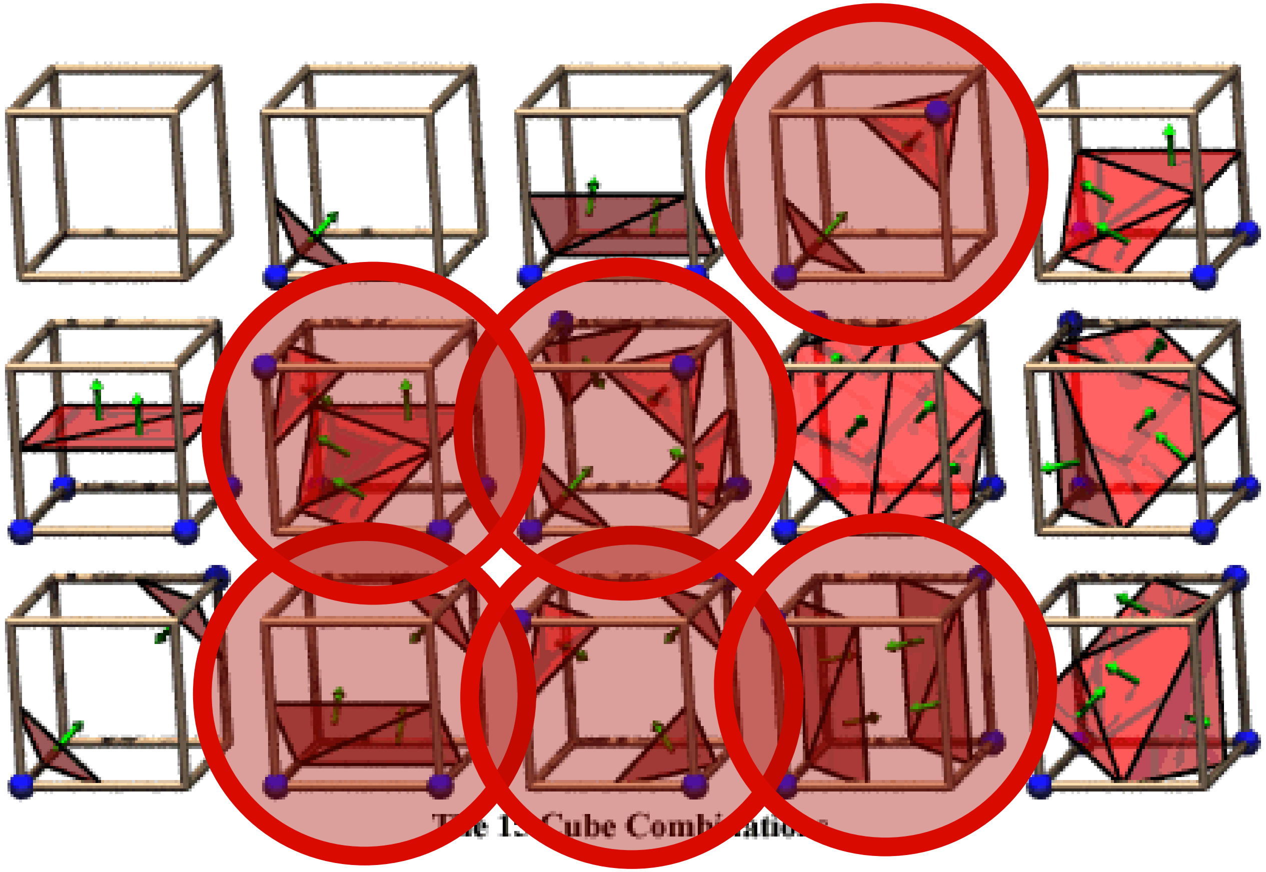
$$\tilde{\mathbf{g}}_{i,j,k} = \begin{pmatrix} \frac{\partial f(\mathbf{x}_{i,j,k})}{\partial x} \\ \frac{\partial f(\mathbf{x}_{i,j,k})}{\partial y} \\ \frac{\partial f(\mathbf{x}_{i,j,k})}{\partial z} \end{pmatrix} \approx \begin{pmatrix} f_{i+1,j,k} - f_{i-1,j,k} \\ f_{i,j+1,k} - f_{i,j-1,k} \\ f_{i,j,k+1} - f_{i,j,k-1} \end{pmatrix}$$

$$\mathbf{g}_{i,j,k} = \frac{\tilde{\mathbf{g}}_{i,j,k}}{||\tilde{\mathbf{g}}_{i,j,k}||}$$



- Use linear interpolation to compute the polygon vertex normal (of the isosurface)

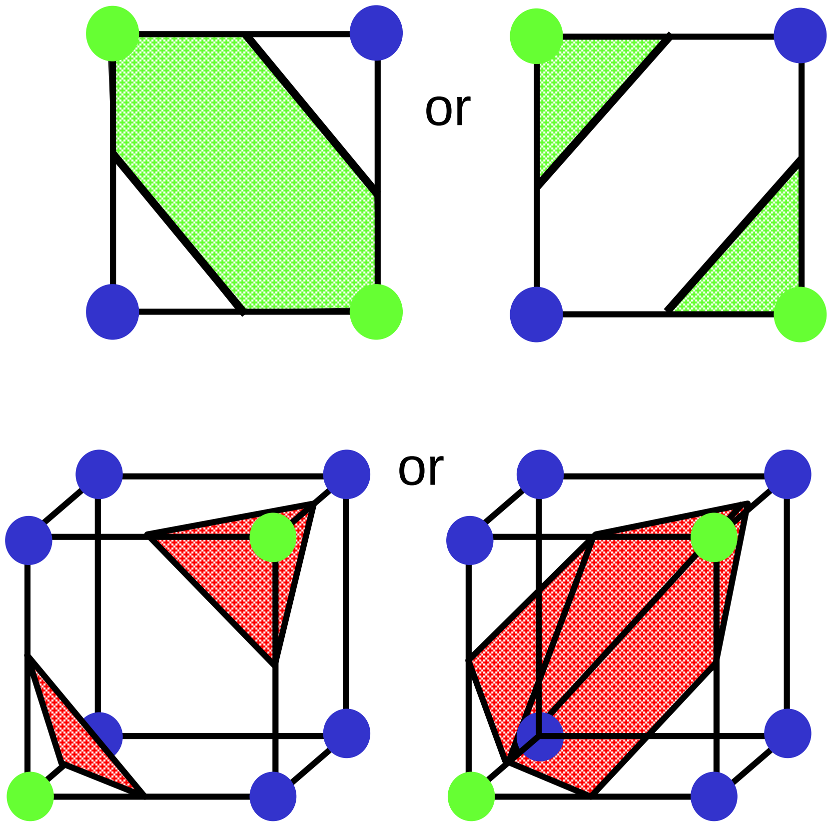
Consider ambiguous cases



- Solve ambiguities using asymptotic decider similar to marching squares

- Consider ambiguous cases
  - Ambiguous cases: 3, 6, 7, 10, 12, 13
  - Adjacent vertices: different states
  - Diagonal vertices: same state
  - Resolution: decide for one case

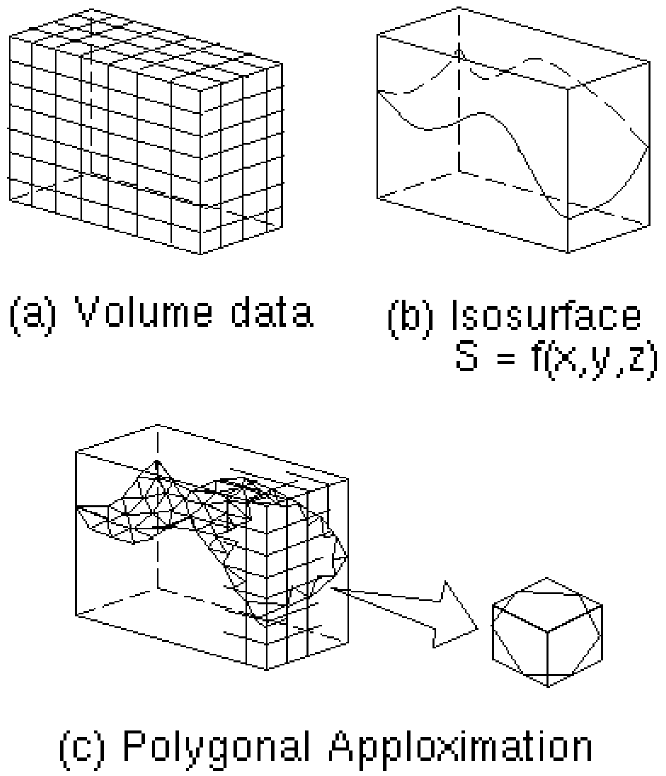
- due to “The Asymptotic Decider”, Nielson and Hamann, IEEE Vis 1991



- Marching Cubes: Summary
  - 256 Cases
  - Reduces to 15 cases by symmetry
  - Ambiguity resides in cases
    3, 6, 7, 10, 12, 13
  - Causes holes if arbitrary choices are made

- Up to 5 triangles per cube

- Dataset of 512^3 voxels can result in several million triangles
  -> many Mbytes!



Optimizations for Isosurface Extraction

- Contour Propagation
- Prevent vertex replication
- Mesh simplification, many more…

Contour Propagation

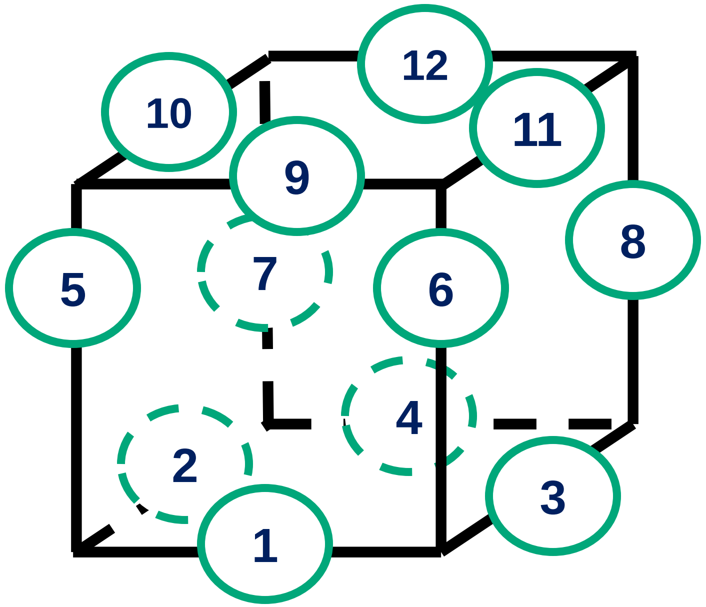
- Acceleration of cell traversal

- Algorithm:
  - Trace isosurface starting at a seed cell
  - Breadth-first traversal along adjacent faces
  - Finally, cycles are removed, based on marks at already traversed cells

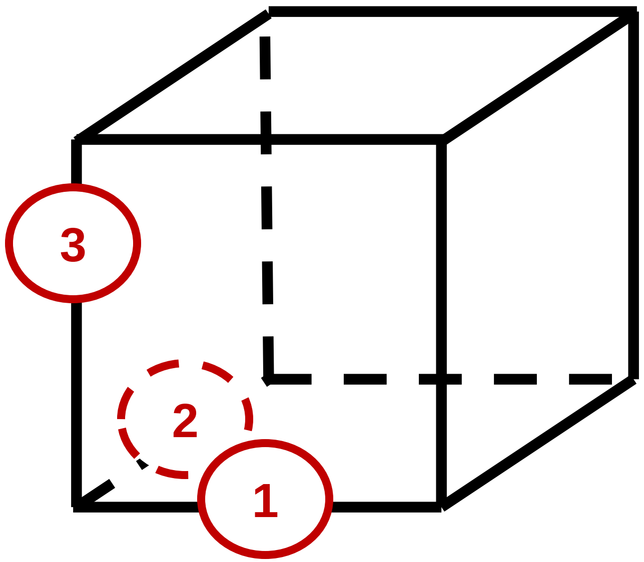
- Problems:
  - Find ALL connected components of the isosurface
  - What is the optimal seed set?

Preventing Vertex Replication

- Based on a unique representation of edges shared by multiple voxels
- Requires a „ghost“ layer of voxels along each axis

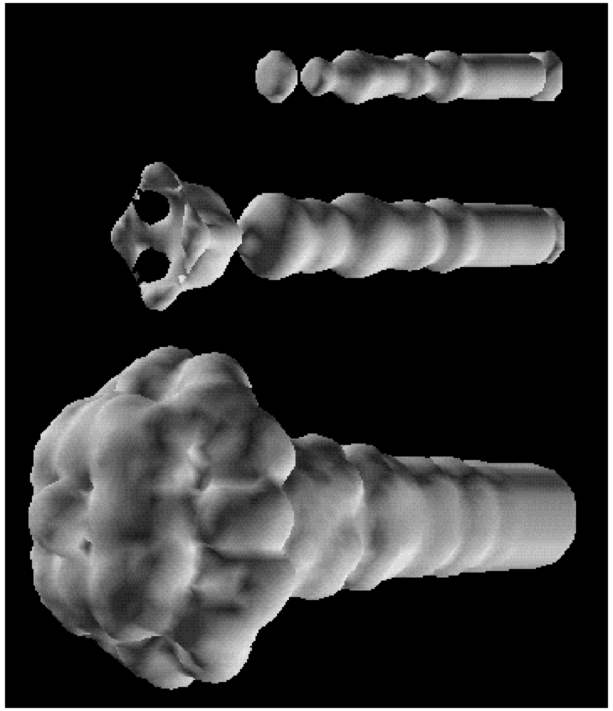


All edges (12)

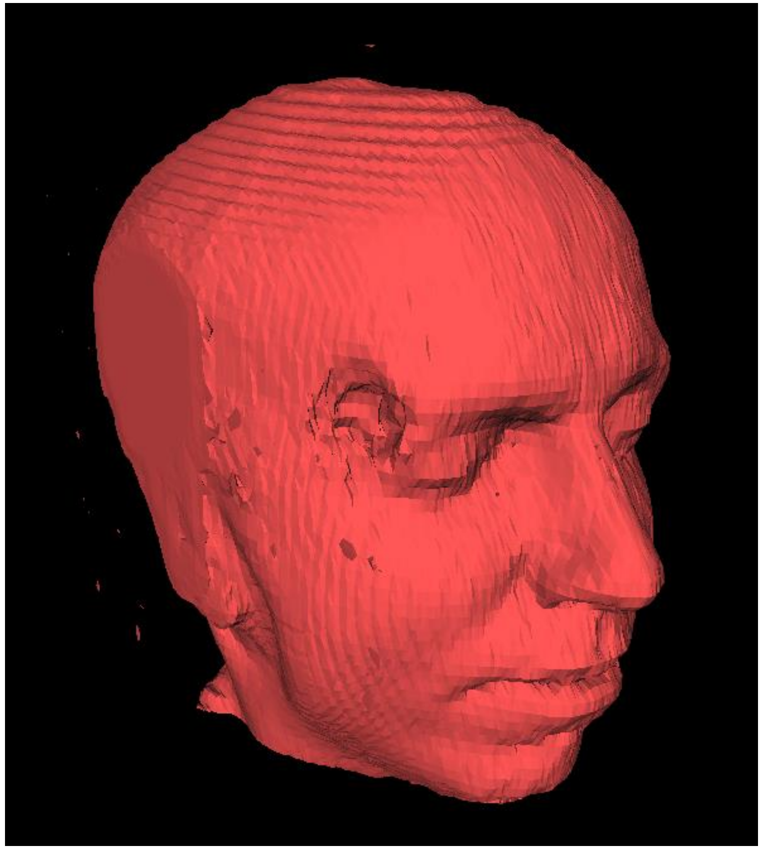


Unique edges (3)

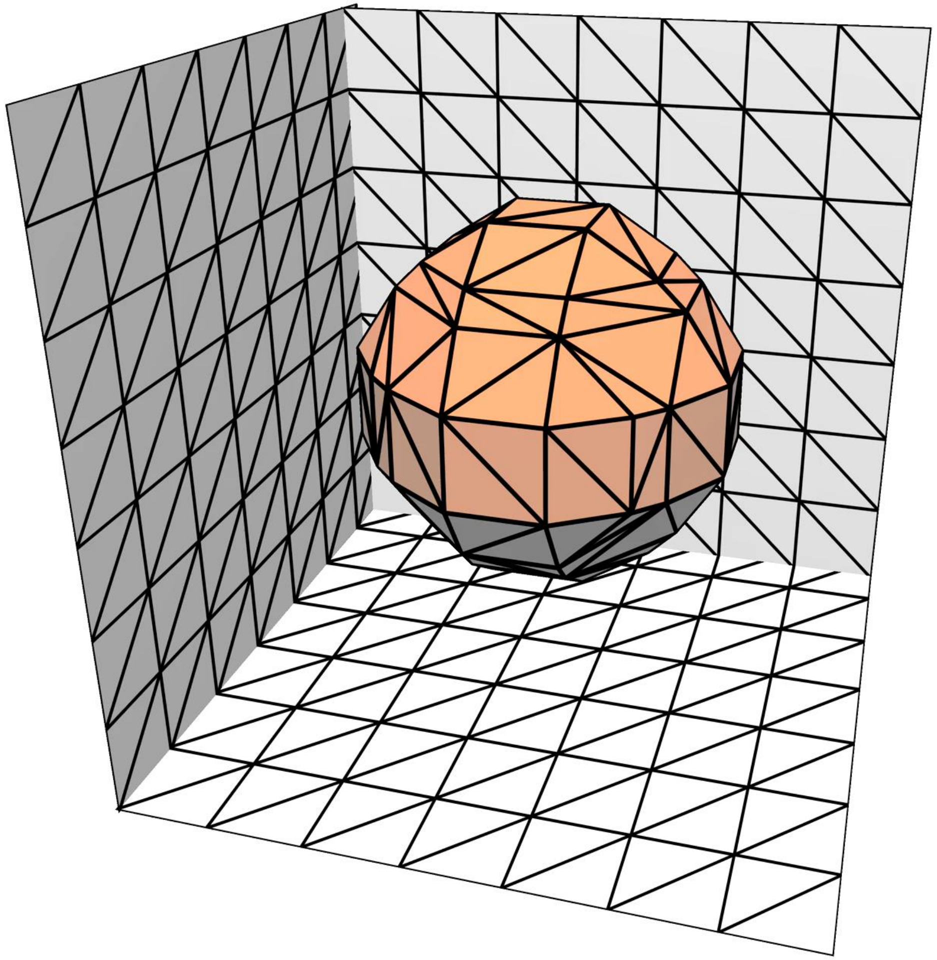
Isosurface at different time steps



Isosurface in 3D medical data set



Isosurface of a sphere in a low resolution grid



Marching Tetrahedra

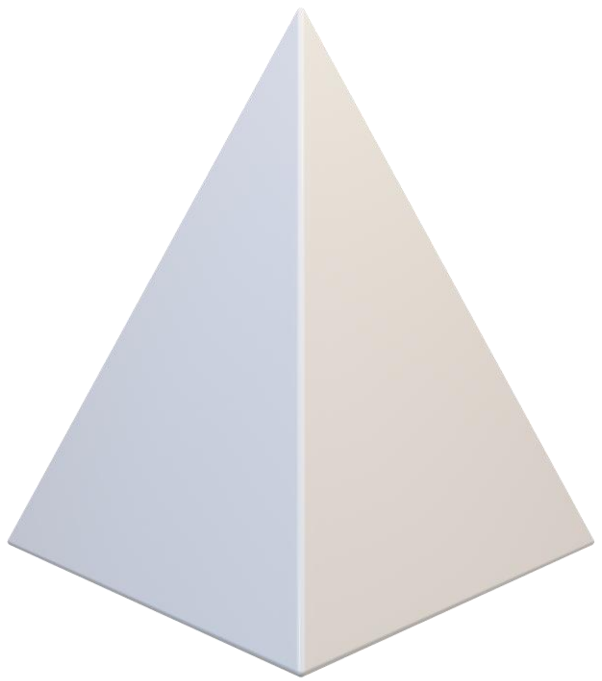
[Shirley et al. 1990]

Works on tetrahedral grids

Application to structured grids possible

split cuboid cells into tetrahedra

Process each tetrahedron similarly to the MC-algorithm

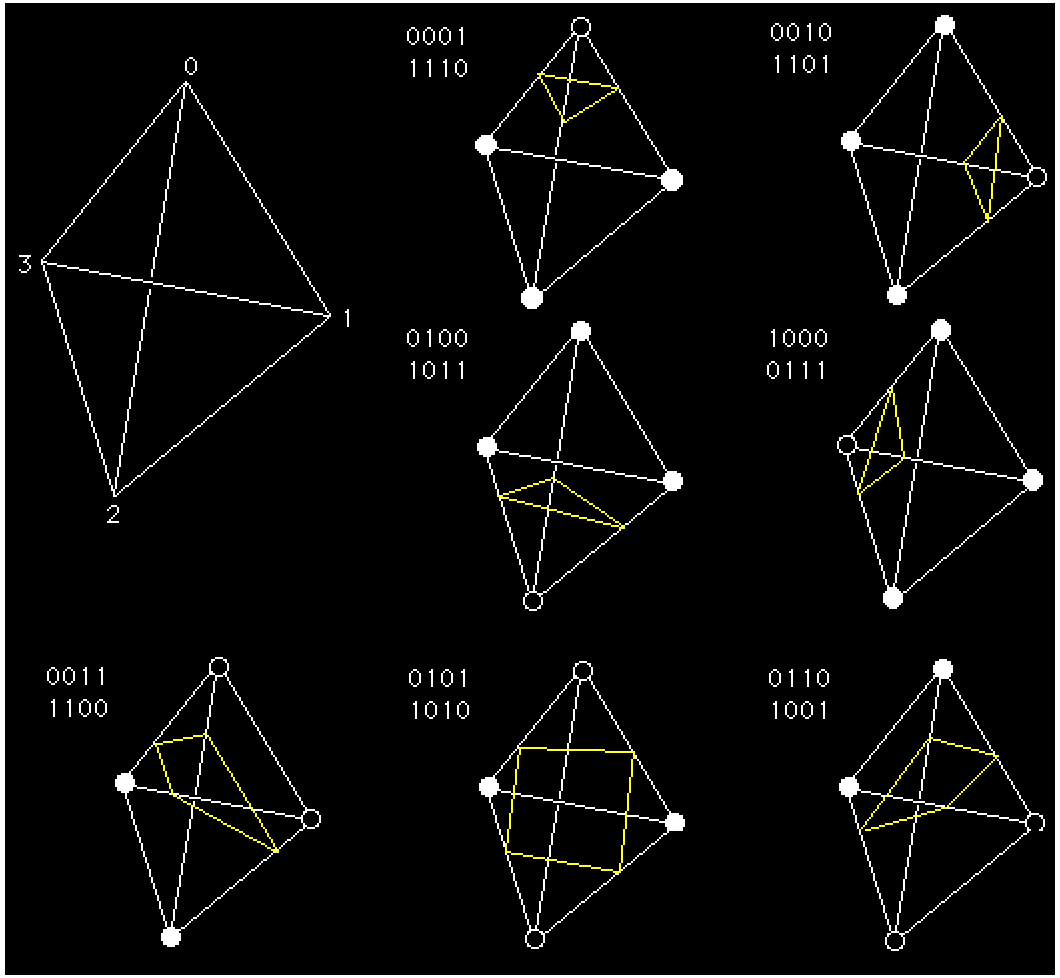


Marching Tetrahedra

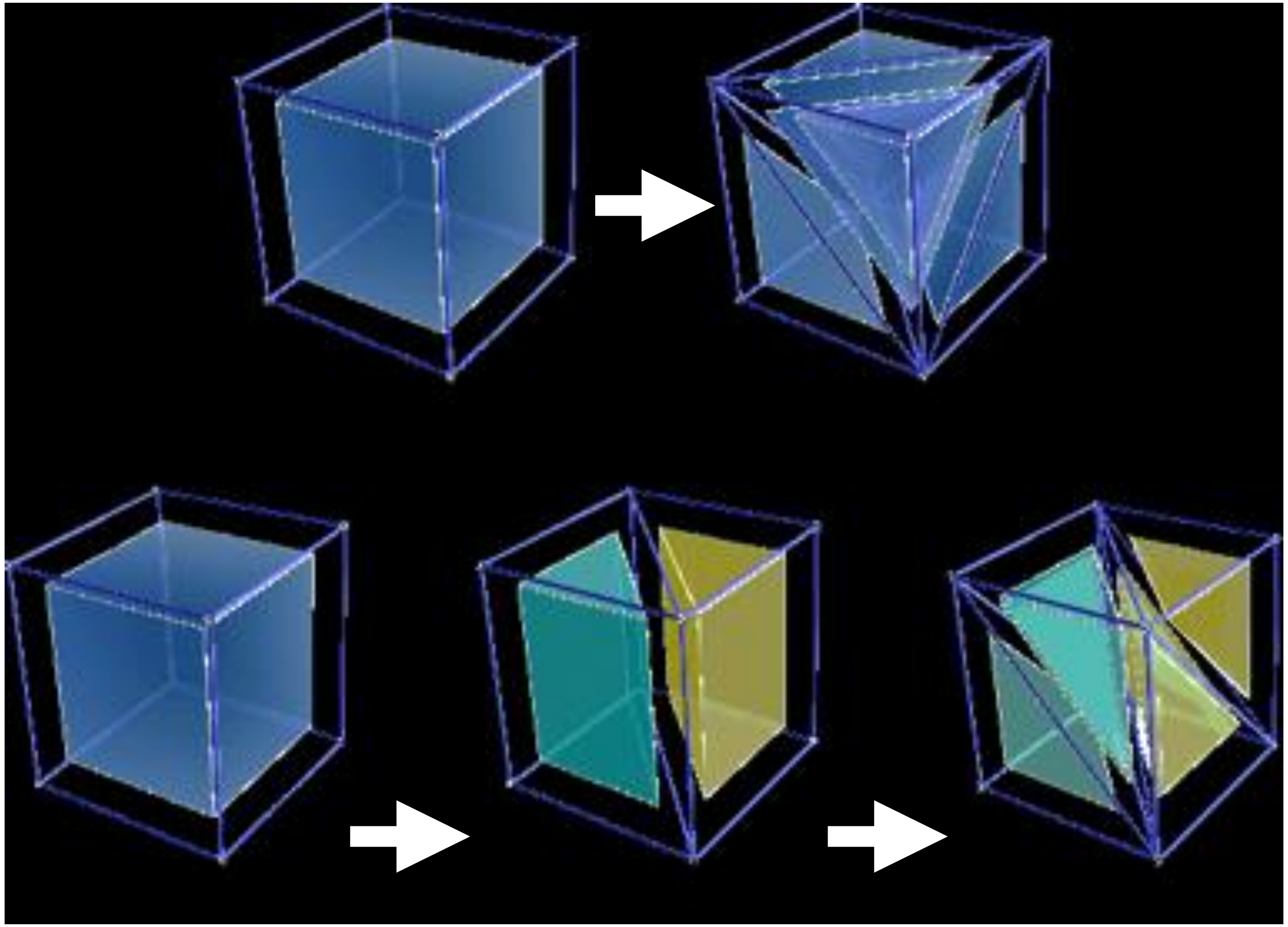
Two different scenarios:

- one „–“ and three „+“ (or vice versa)
  The surface is defined by one triangle

- two „–“ and two „+“
  Sectional surface given by a quadrilateral –
  split it into two triangles using the shorter diagonal



Initial Cube Five Tetrahedra



Initial Cube Two Prisms Six Tetrahedra

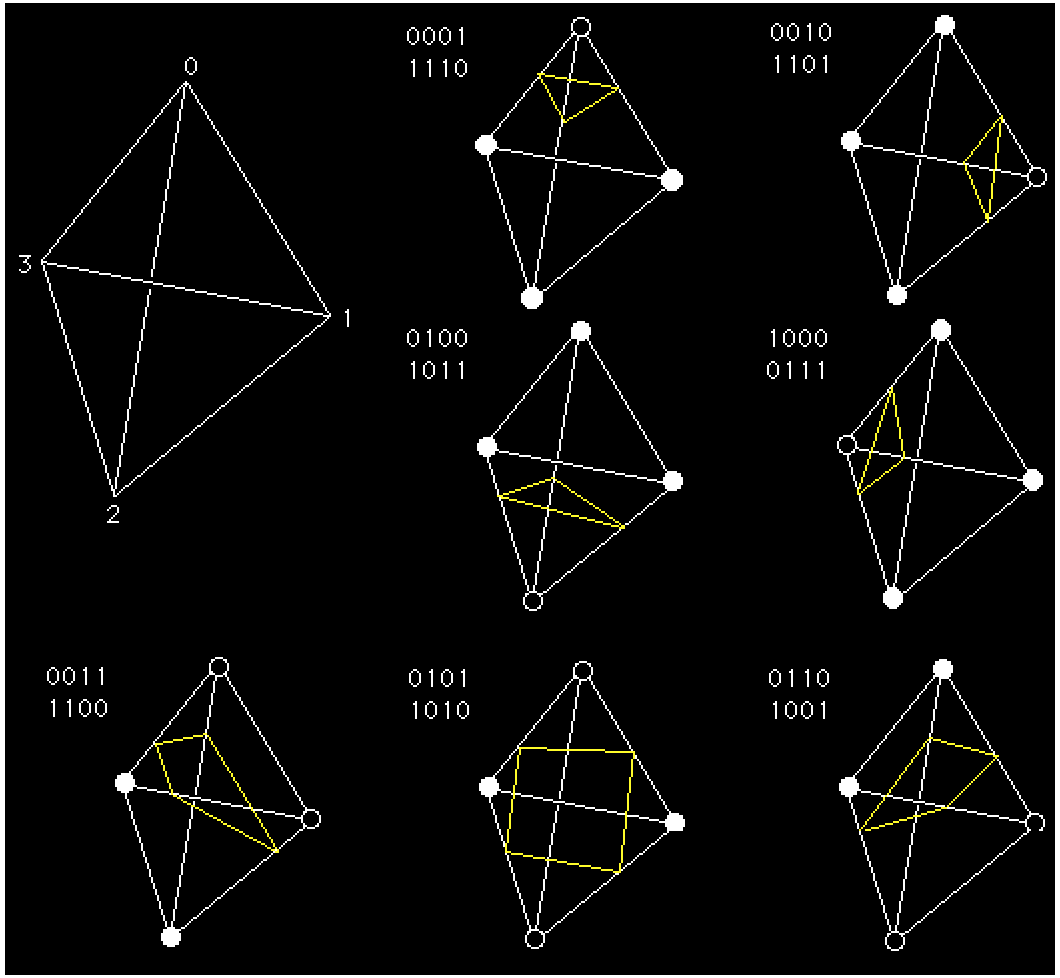
Marching Tetrahedra

Fewer cases than MC

3 cases instead of 15 cases for MC

no problems with consistency between adjacent cells

Number of generated triangles might increase considerably compared to the MC-algorithm when splitting voxels into tetrahedra



Summary

- Geometry-based Scalar Field Visualization
  - Contouring
- Properties of contours
  - closed, cannot intersect, nested, gradient is perpendicular, …
- 2D isoline extraction
  - Marching Squares
  - Asymptotic decider
- 3D isosurface extraction
  - Direct computation without lookup table
  - Marching Cubes
  - Marching Tetrahedra