Scientific Visualization

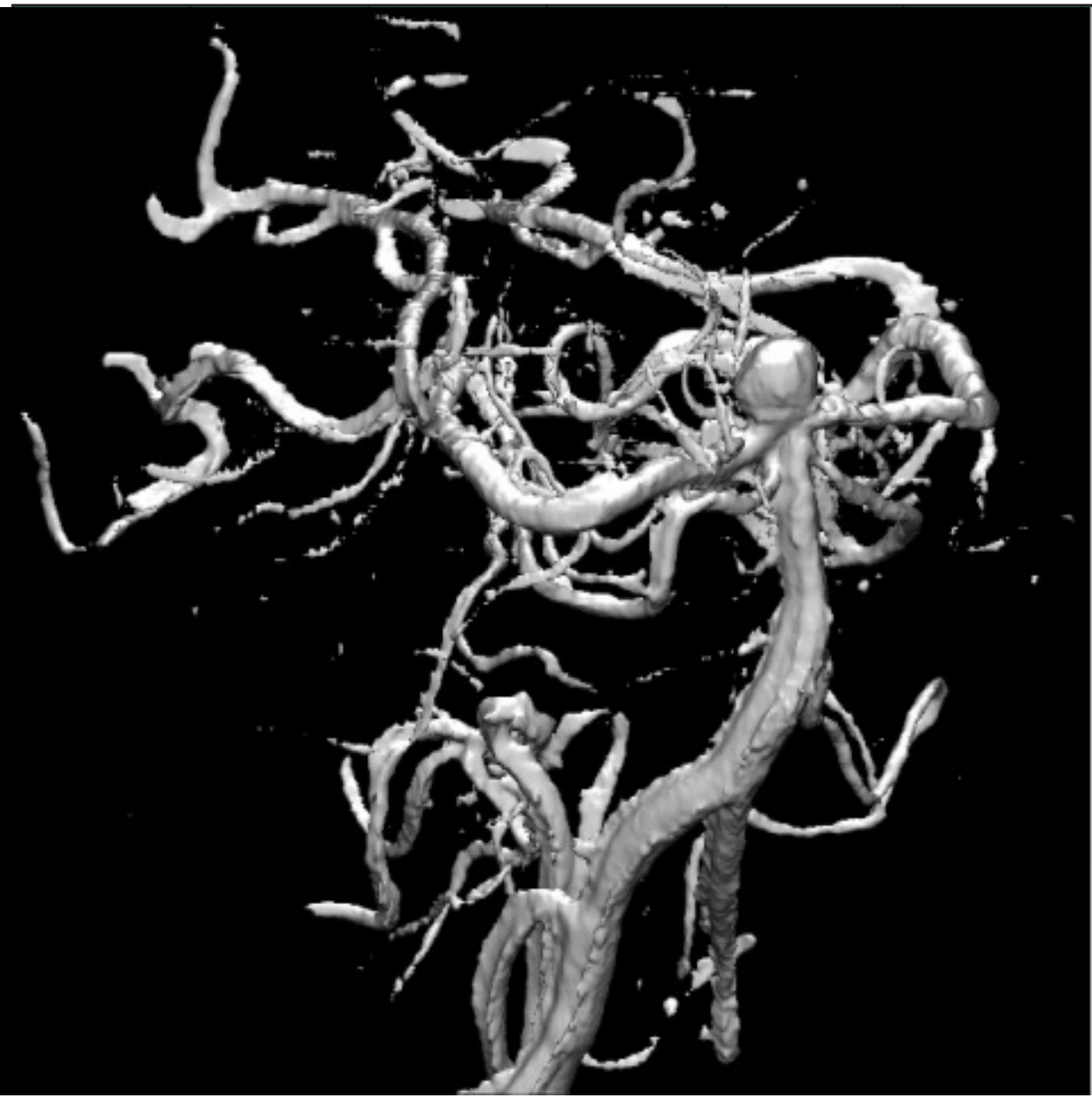
Hai Lin
State Key Lab of CAD&CG

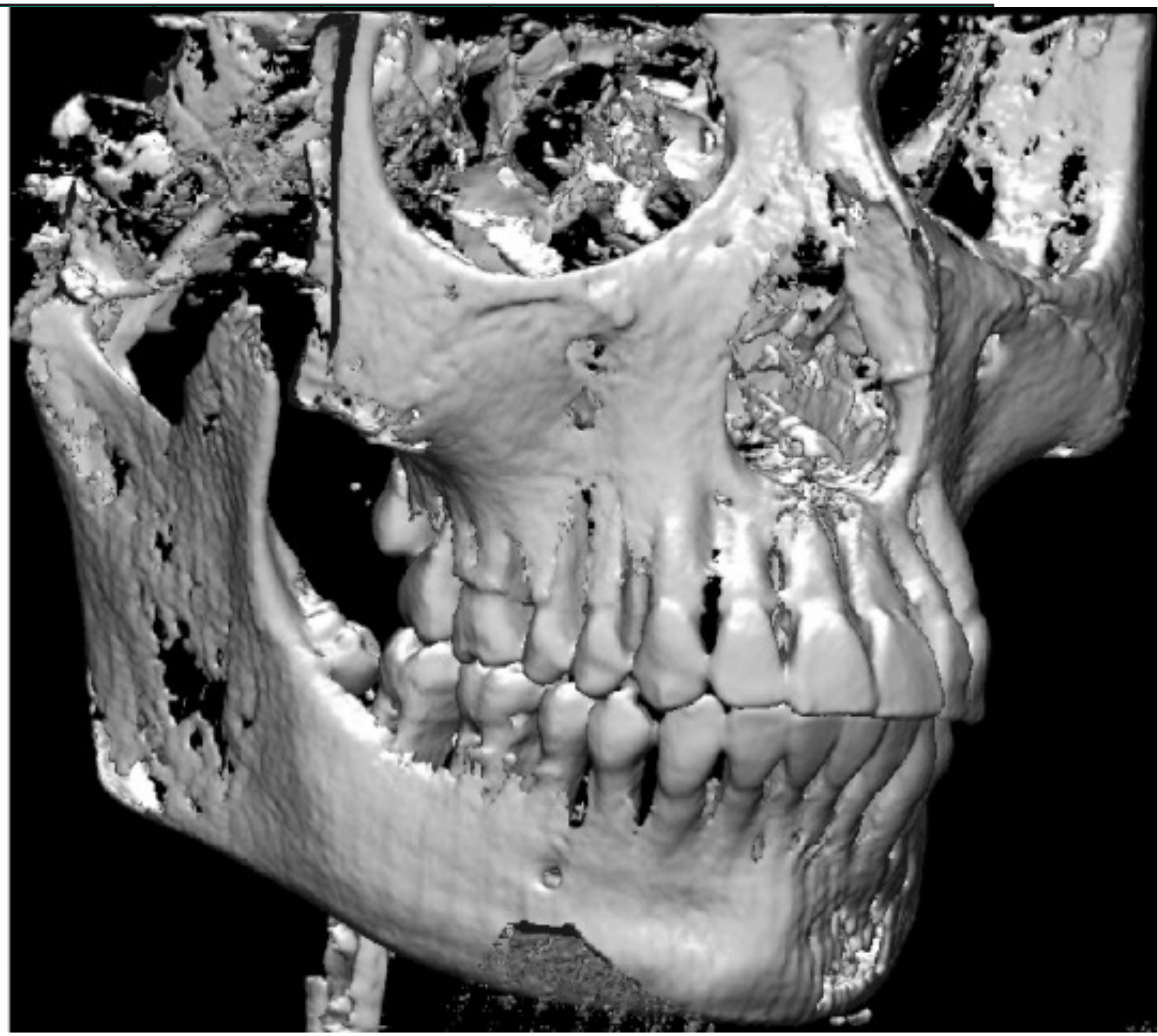
Lecture 3
Contour and Isosurface

Lecture 3

- Contour
- Isosurface

Mean sea level pressure **Annual mean**





More Formally

- A scalar visualization technique that creates curves(in 2D) or surfaces (in 3D) representing a constant scalar value across a scalar field.
- Contour lines are called isovalue lines or isolines
- Contour surfaces are called isovalue surfaces or isosurfaces

1

Contour

- Definition
- Characteristic
- Pipeline
- Grid Sequence Method
- Grid Free Method

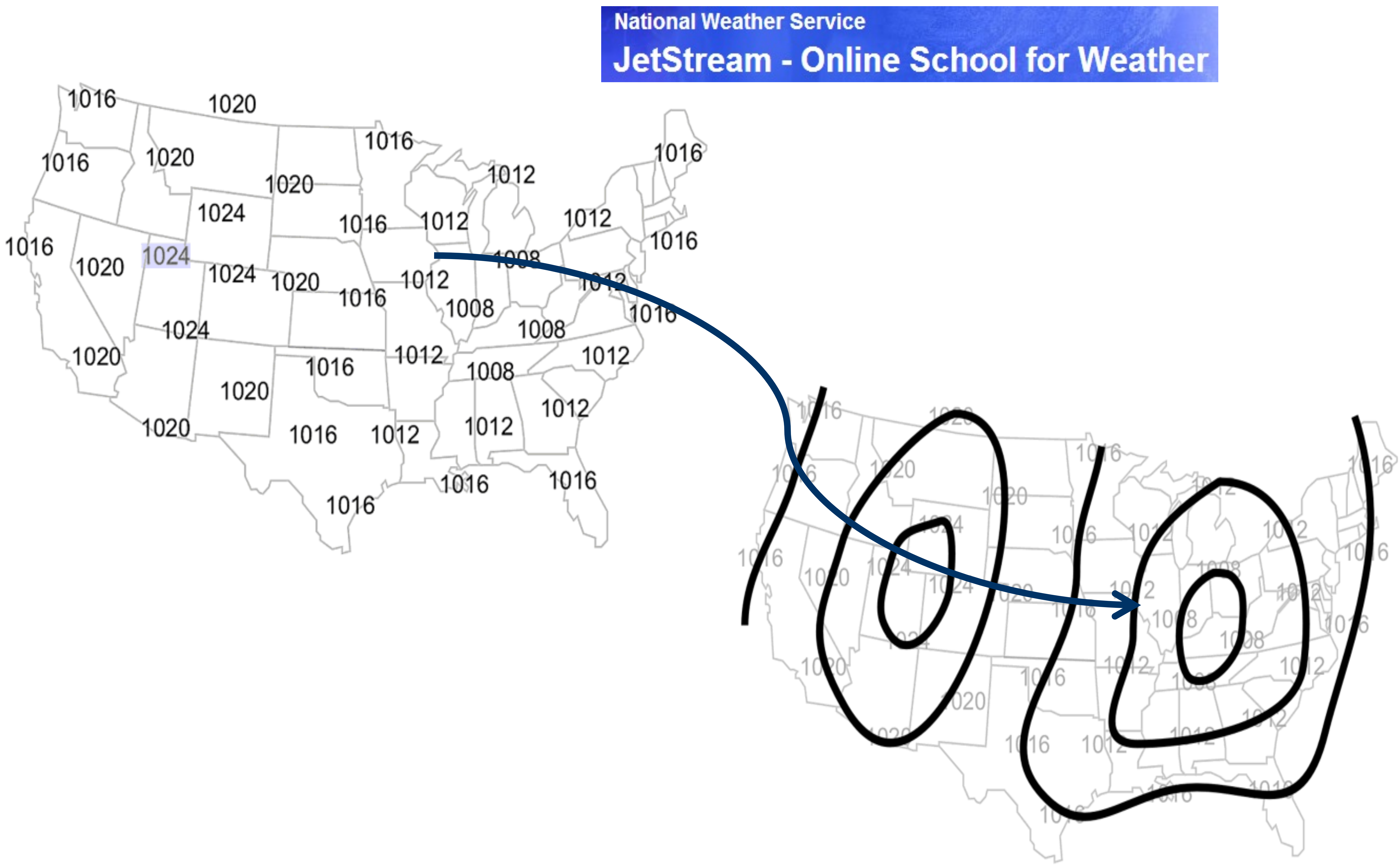
- Definition
- Characteristic
- Pipeline
- Grid Sequence Method
- Grid Free Method

What are the contours?

- Triple point dataset
  - $((x, y), value)$
- 2D scalar fied
  - $F = F(D)$
    1D-scalar function defined on the surface D
- Contours
  - Set of points where the scalar field F has a given value c:
    $$\{(x \in D: F(x) = c\ )\}$$

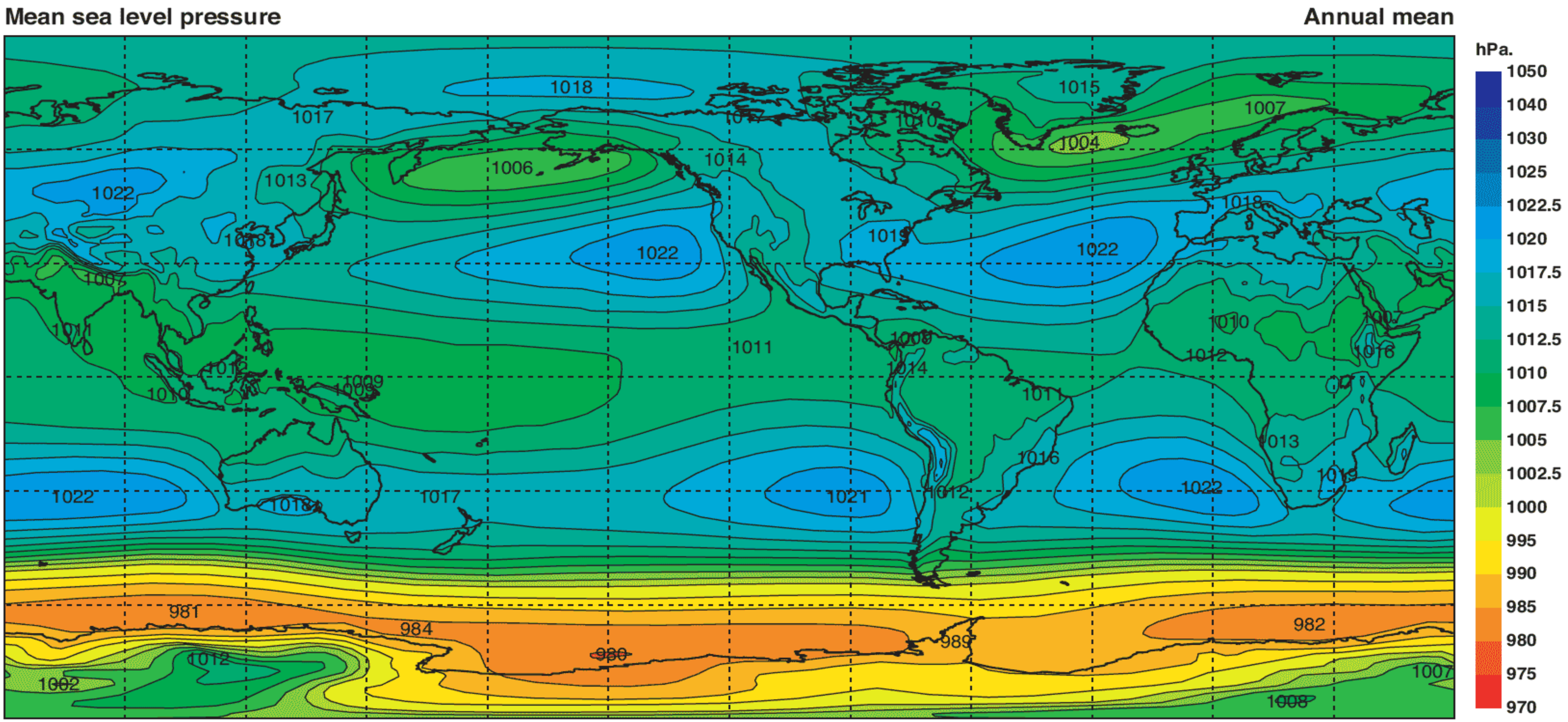
What are the contours?

- Conversion of a scalar field into contours



Pressure Contours

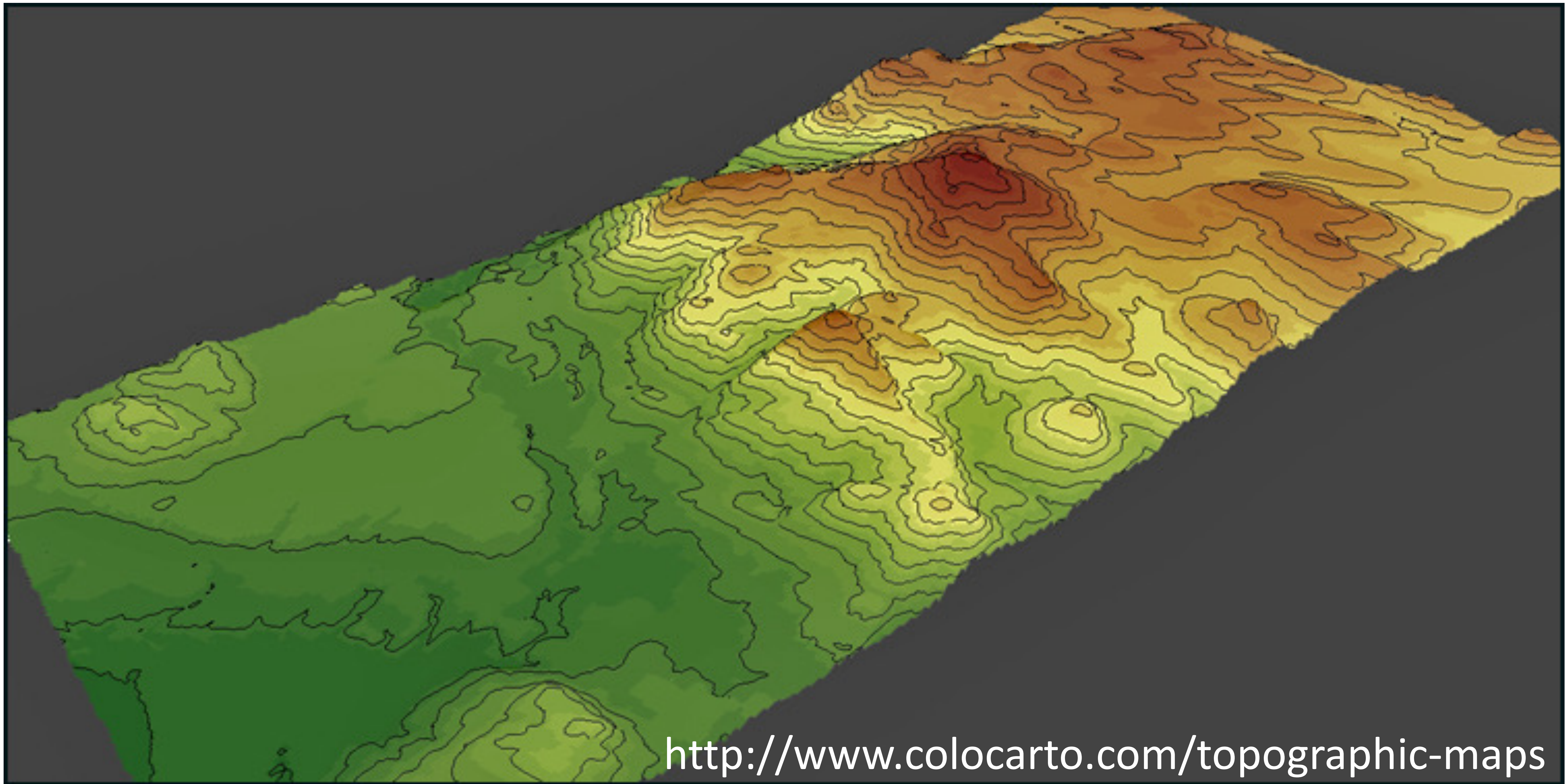
- Pressure contours from meteorology



http://oceanworld.tamu.edu

Elevation Contours

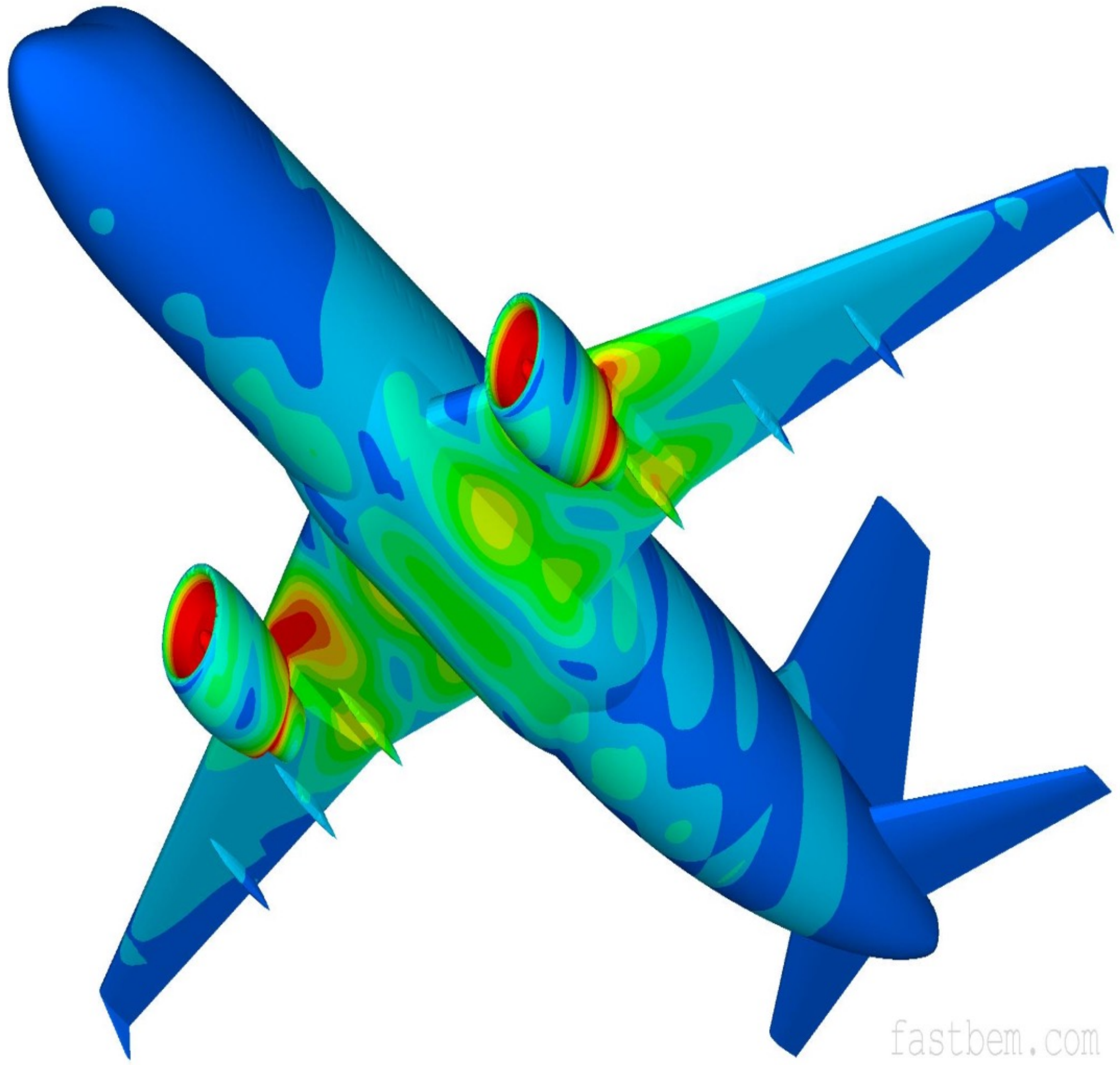
- Elevation contours from topography
  - An area of western Massachusetts, with contours at 20-foot intervals.
  - The greener depict a valley.
  - The darkest red area is the highest point in the area



The Acoustic Pressure Fields

- The acoustic pressure fields on the surface of the Airbus A320 airplane

FastBEM Acoustics®



- Definition
- Characteristic
- Pipeline
- Grid Sequence Method
- Grid Free Method

Characteristic

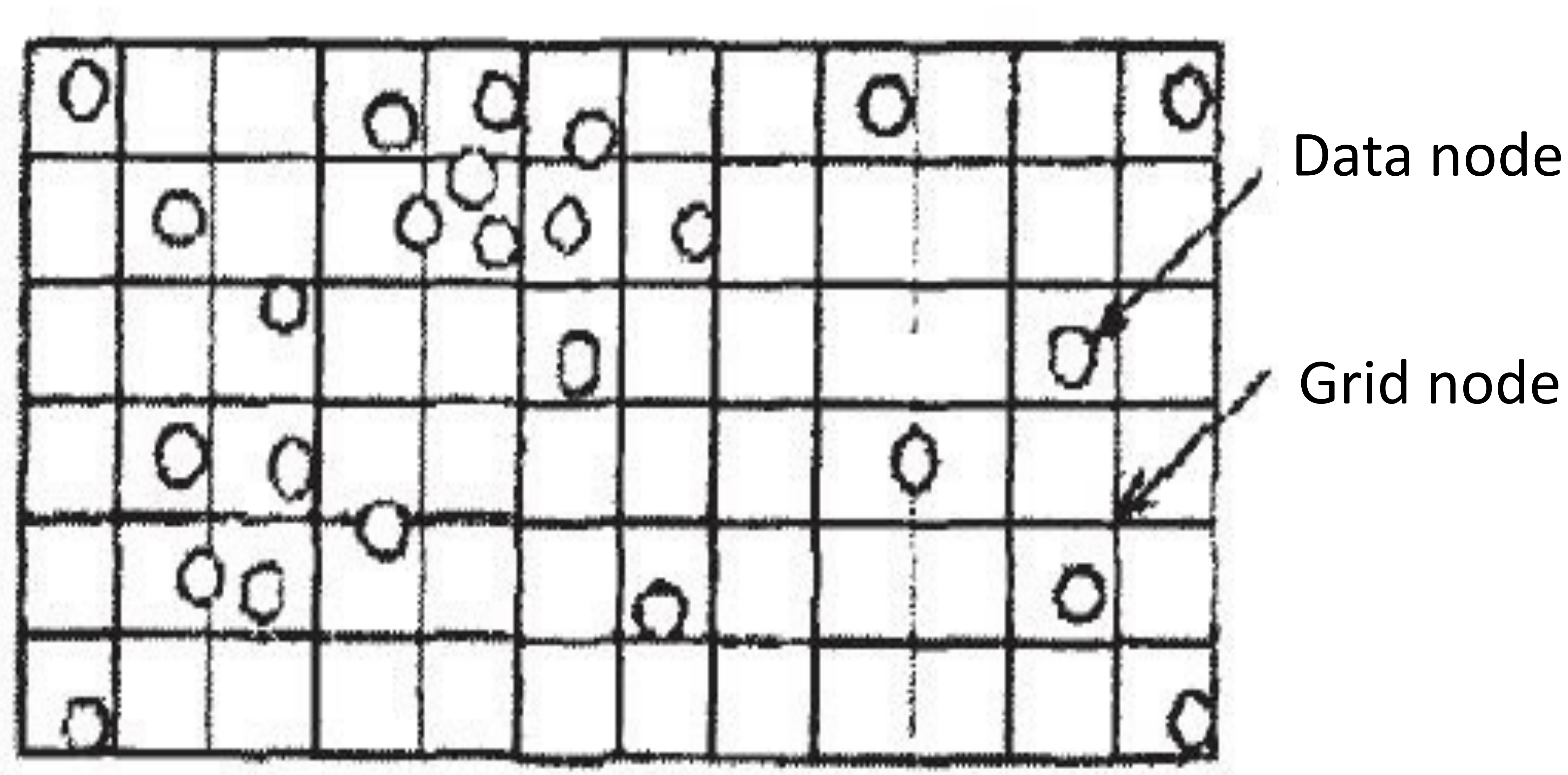
- Contour lines do not intersect each other
- For a given value C, there may exist more than one the corresponding contour line
- The domain is bounded, so the contour could be closed or not

- Definition
- Characteristic
- Pipeline
- Grid Sequence Method
- Grid Free Method

Pipeline

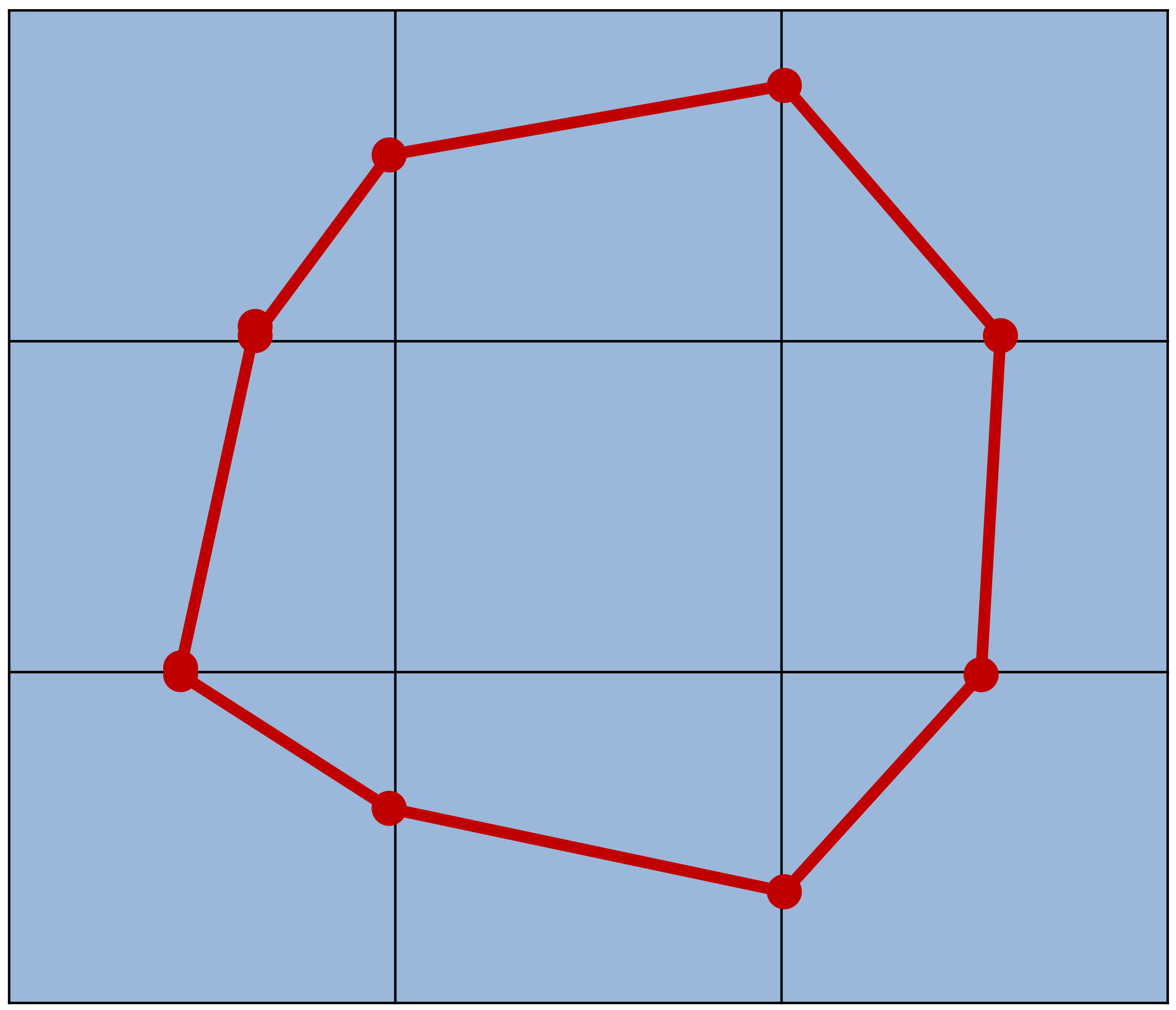
- Discrete data gridding
- Contour generation
- Color filling

Discrete Data Gridding



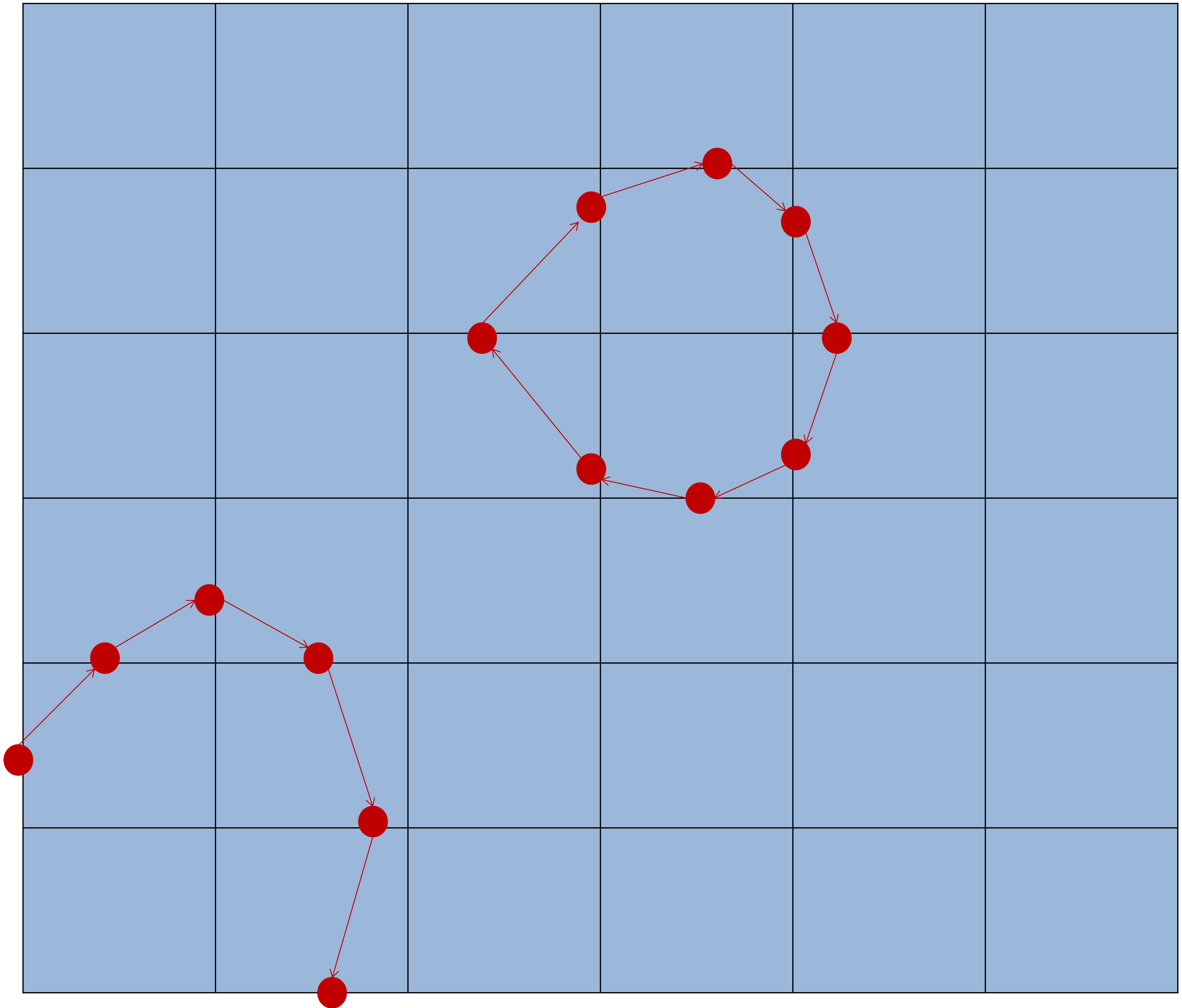
Contour Generation

- Grid sequence method
- Grid free method



Contour generation

- Grid free method



Contour Generation

- Grid sequence method
  - Rectangular grid method
  - Krige
  - Triangle-meshed method
- Grid free method
  - marching
  - adaptive
  - recursive

Color Filling

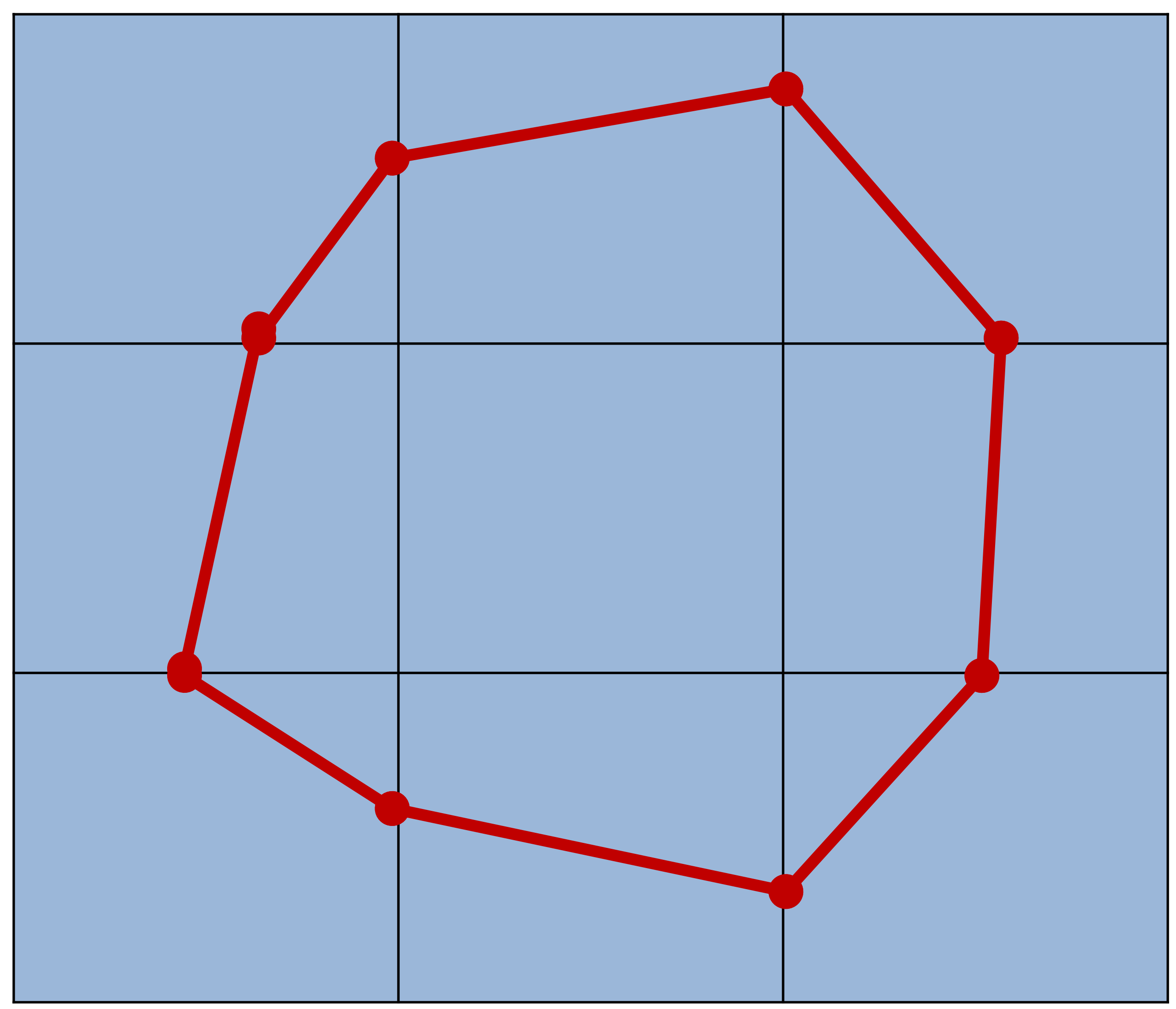
	Scan Filling	Region Filling
Basic idea	➢ Compute the color of each pixel ➢ Fill color at the level of pixel	➢ Find enclosed polygons between the con tours ➢ Fill color at the level of polygons
Advantage	simple	better visual effect
Disadvantage	slow	complex computation

- Definition
- Characteristic
- Pipeline
- Grid Sequence Method
- Grid Free Method

Grid sequence method

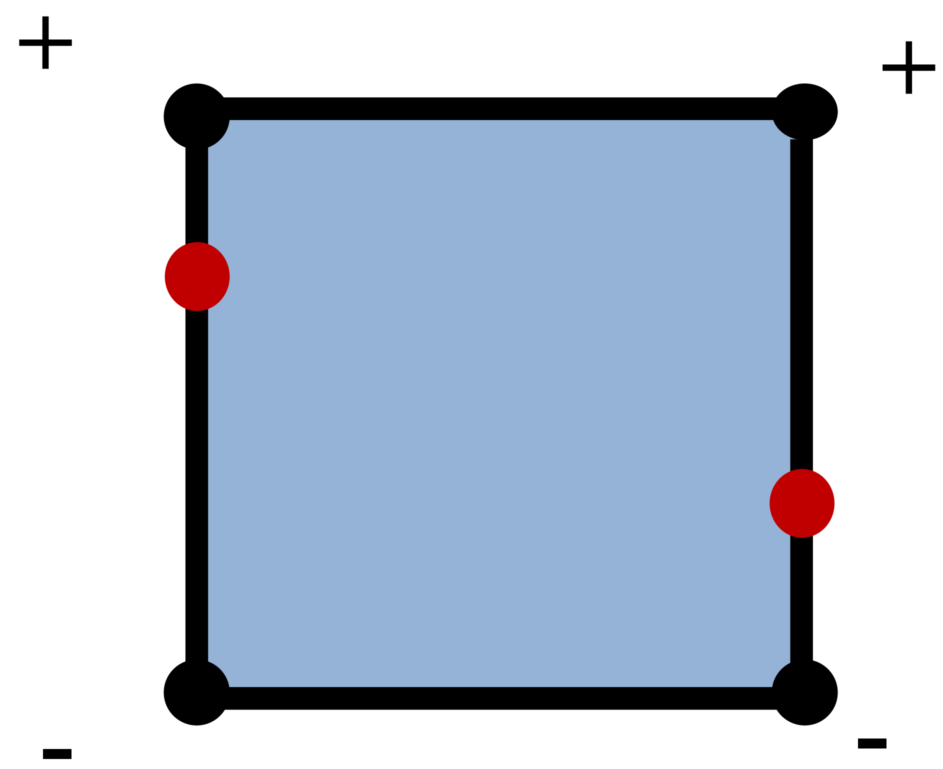
- find intersection with grid edges for each cell
- connect points in each cell to generate a segment of contour
- Generate a complete contour automatically

Marching squares



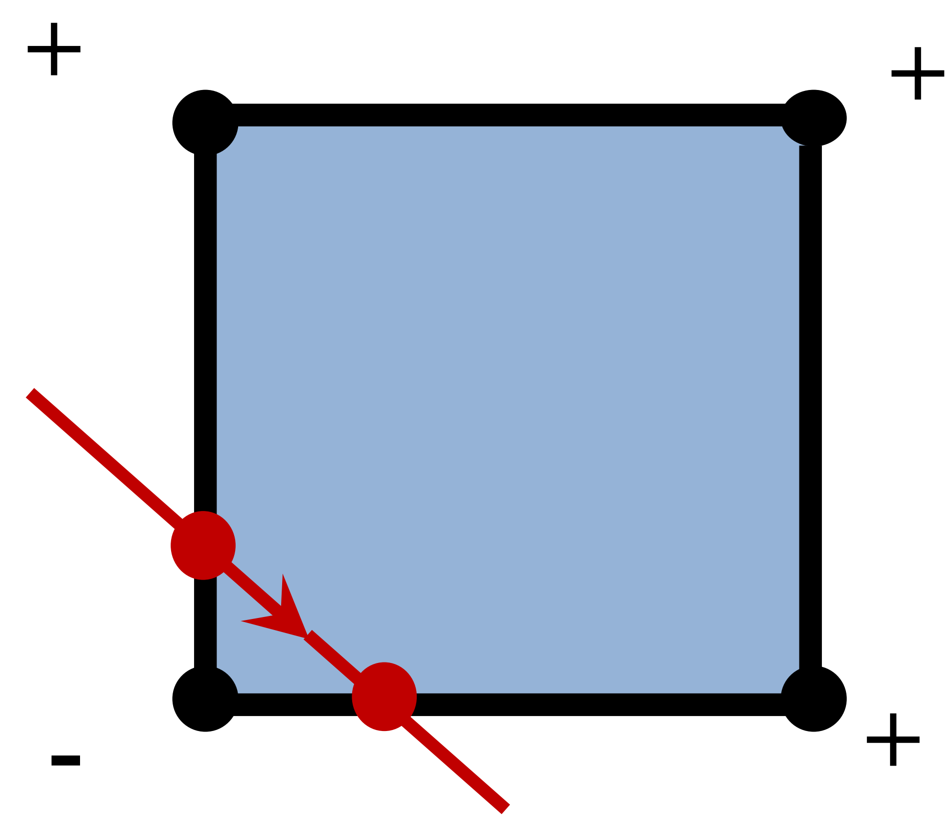
Name Convention of *In* or *Out*

- If a value is smaller than the iso-value, we call it "In"
- If a value is greater than the iso-value, we call it "Out"



Name Convention of direction

- If a vertex is smaller than the iso-value, it is on the right of the contour
- If a vertex is greater than the iso-value, it is on the left of the contour



Marching Squares

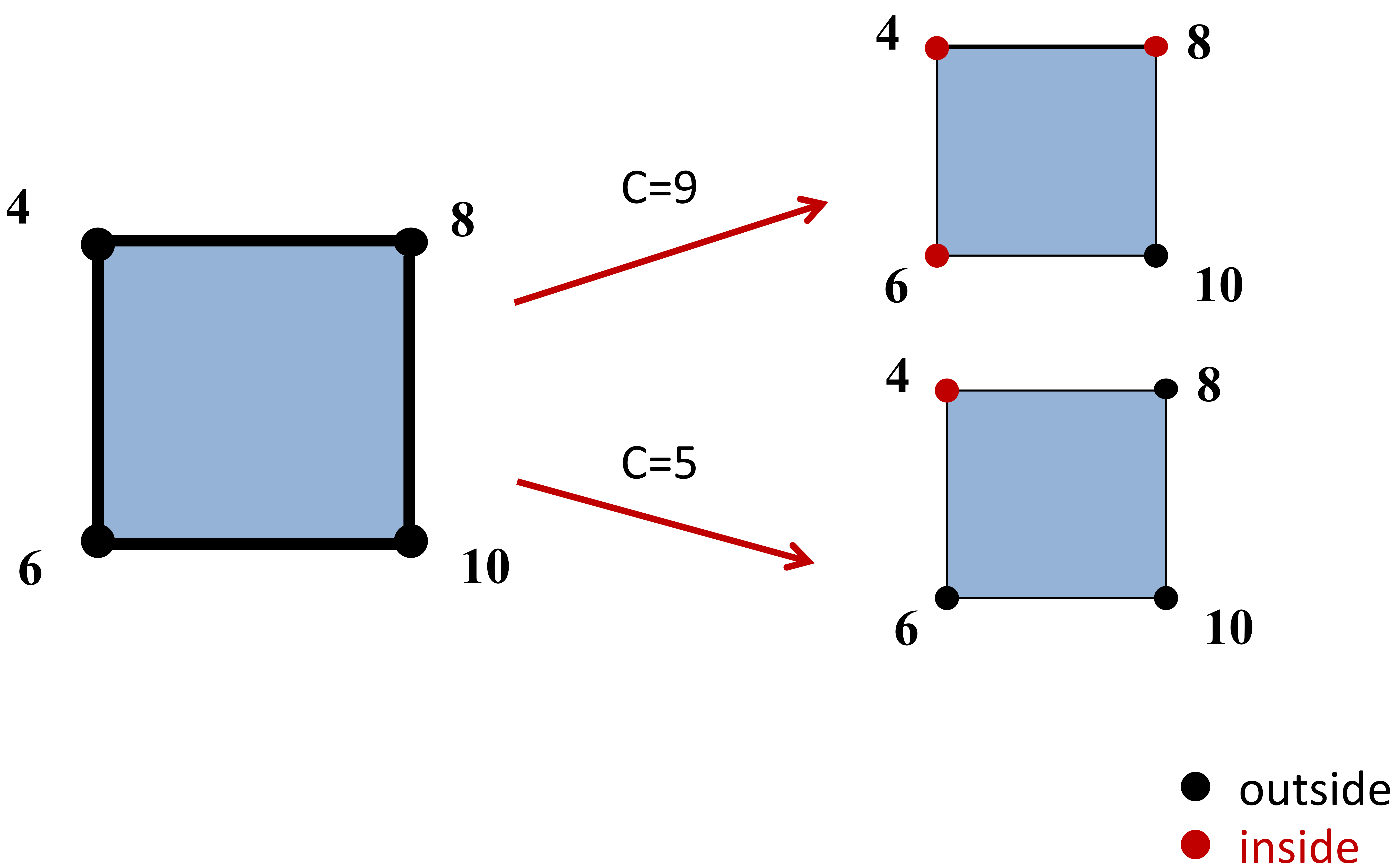
- Computes contour lines in 2D data set
- Treats each cell independently
- Assumption : contour can pass a cell in a finite number of ways, depending on inside/outside relationship with scalar value
- Table (case table)contains all possible topological states

Marching Squares

- For every cell
  - Calculate inside/outside state of each vertex of cell
  - Create an index by storing binary state of each vertex in a separate bit
  - Use index to look up the topological state of the cell in a case table
  - Calculate the contour location (via linear interpolation)for each edge in the case table
  - Connect contour locations by line

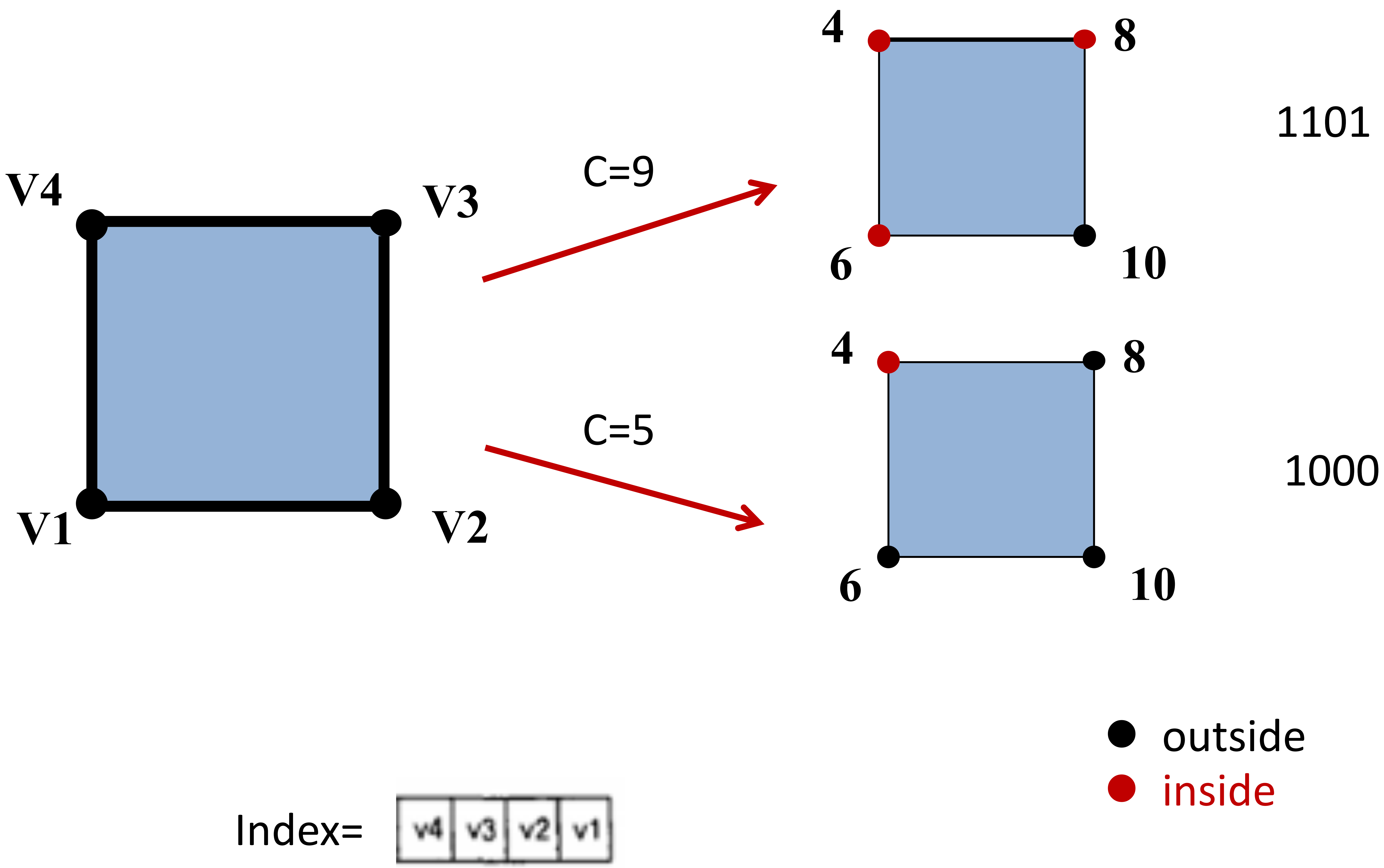
Classify Each Cell

- Binary classification of each cell



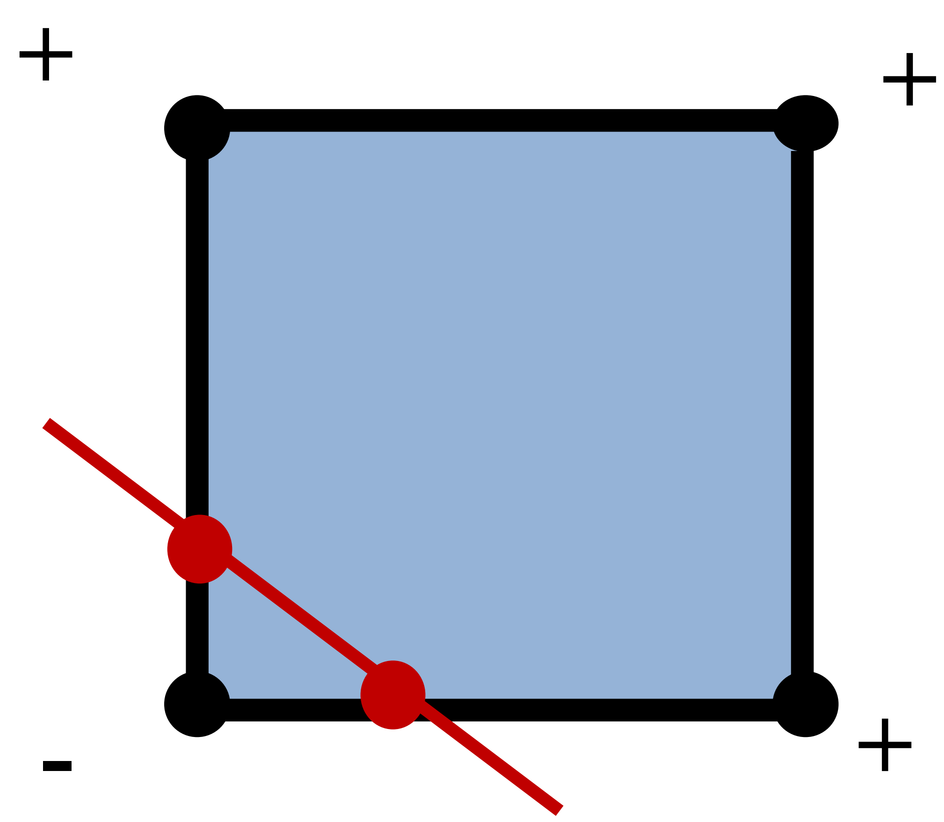
Build an Index

- Binary classification of each cell



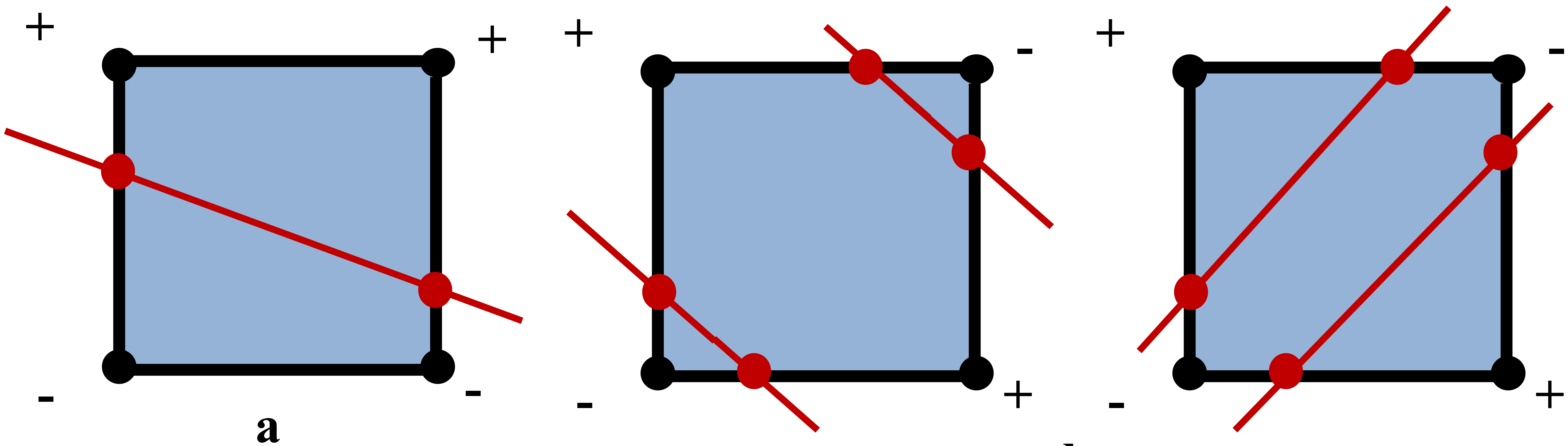
the Case Table

- Four vertexes in each cell
  - Cell with four vertices ‘+’ or ‘-’
    no intersection
  - Edge of cell with one ‘+‘ and one ‘-’



the Case Table

- vertices of the cell: two ‘+’ or two‘-’
  - One contour segment: Fig. a
  - two contour segments: fig. b

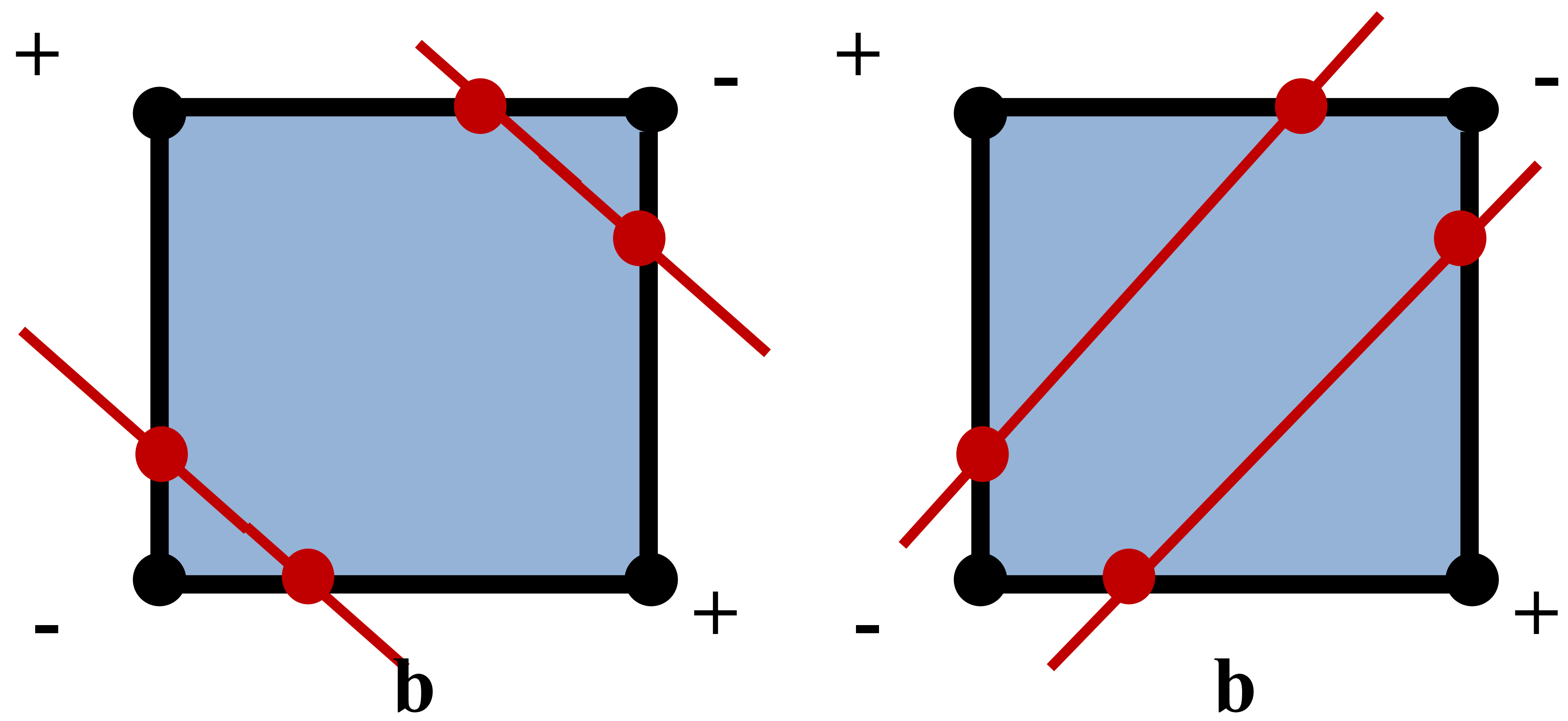


a

b

the Case Table

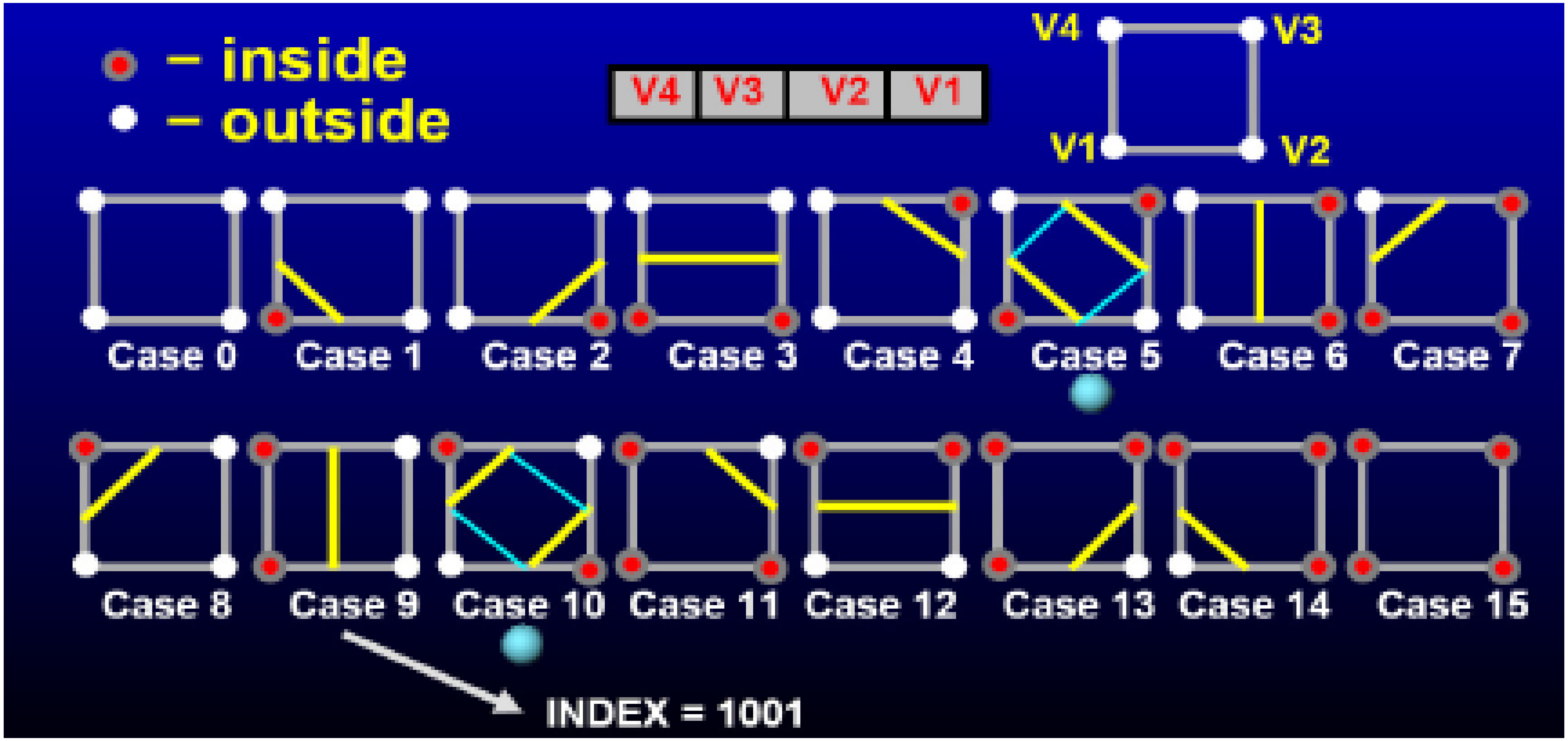
- vertices of the cell: two ‘+’ or two‘-’
  - ➢ One contour segment: fig. a
  - ➢ two contour segments: fig. b
    - Direction
    - No direction



- ..

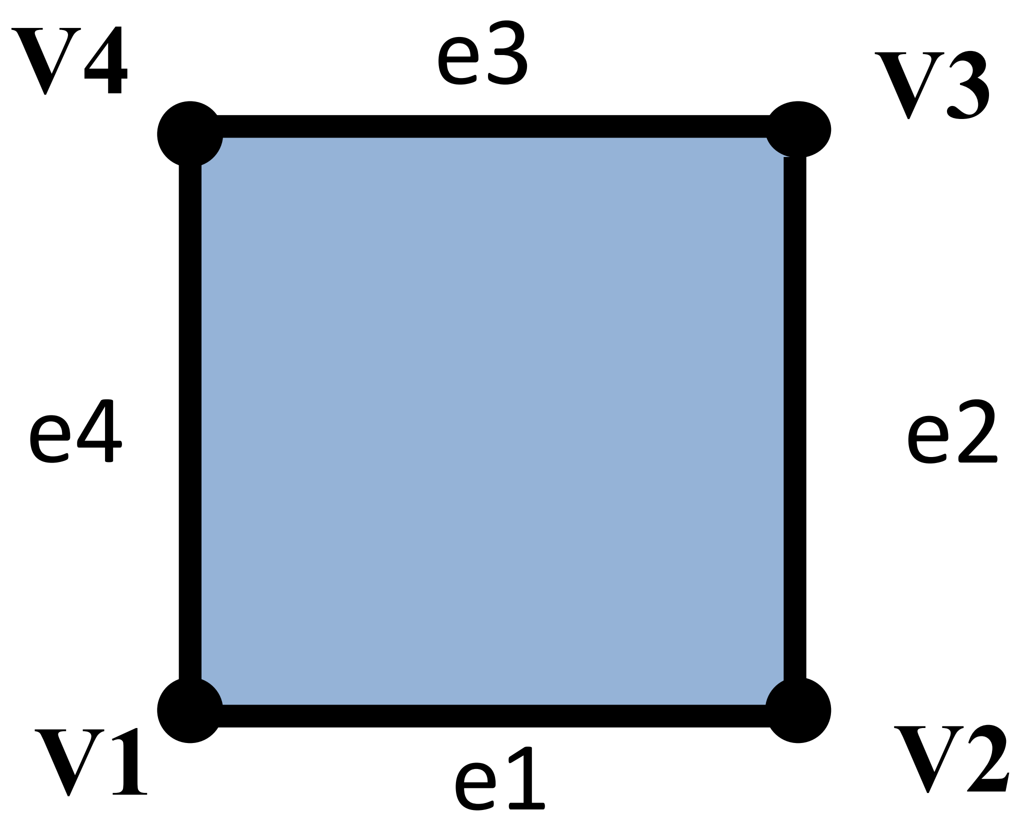
Look-up the Case Table

- Case table



Look-up the Case Table

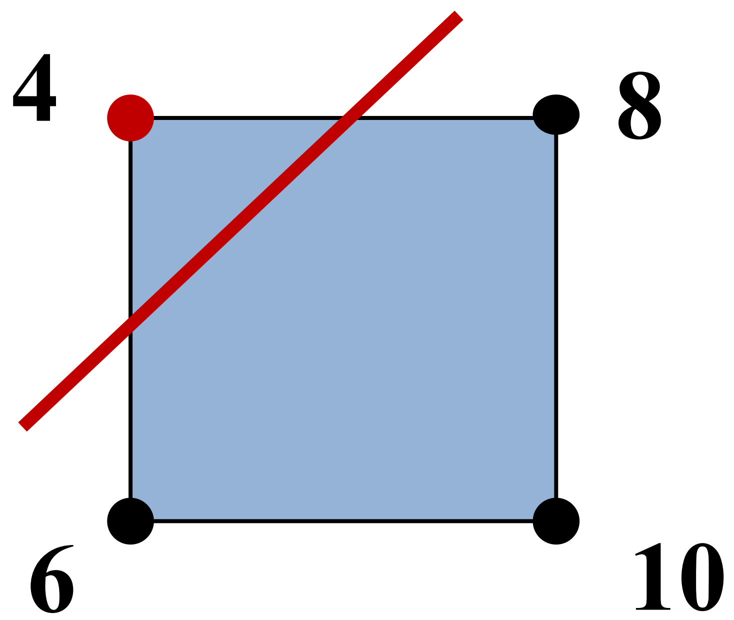
- Given the index for each cell, a table lookup is performed to identify the edges that has intersections with the iso-line



index	Intersection edges
0	NULL
1	e3,e4
…	…

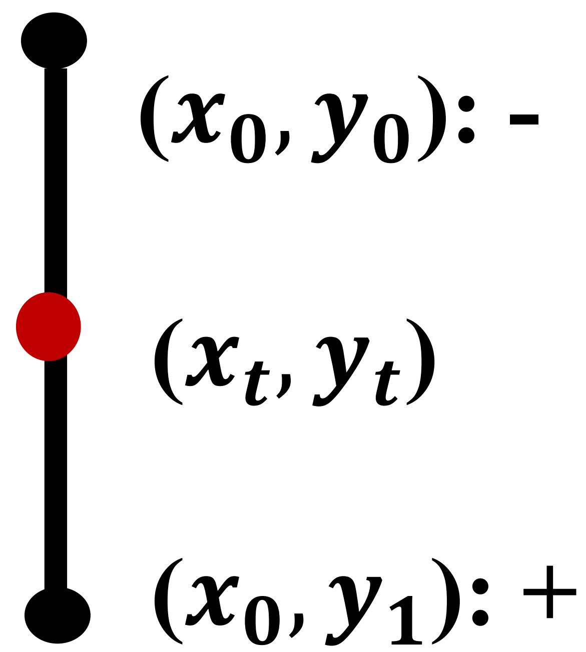
Example

- Index=0001



Interpolation

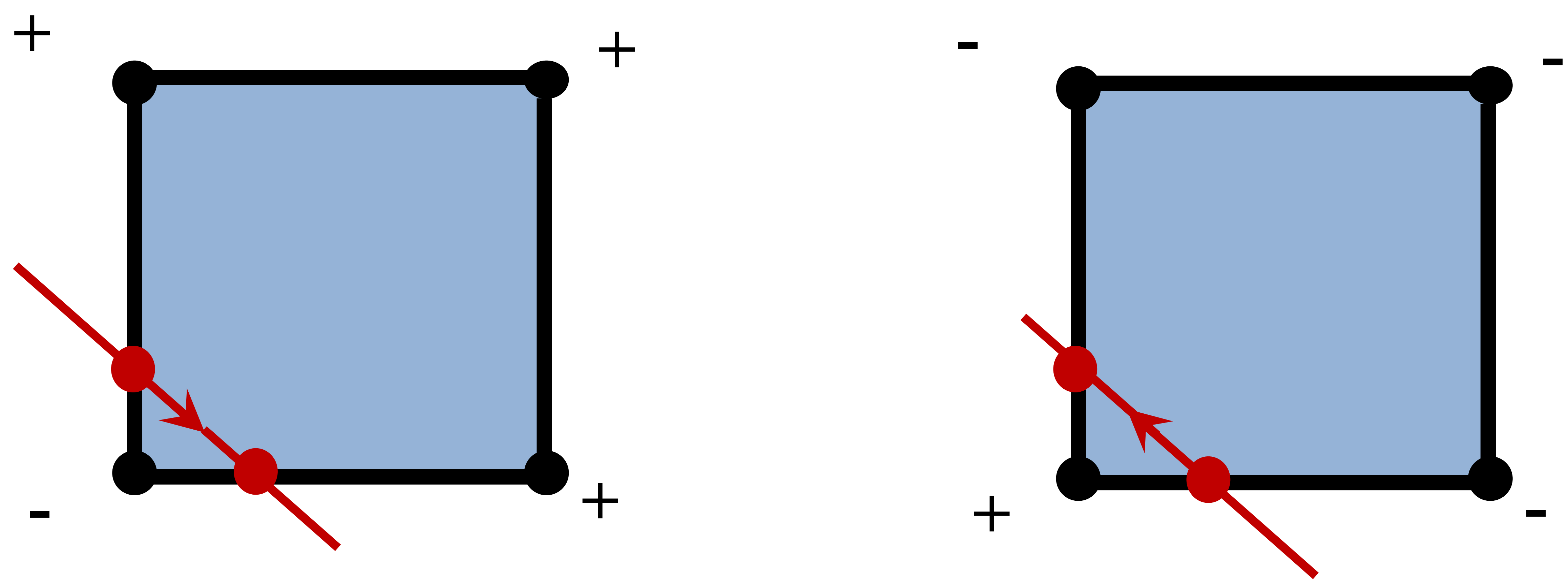
- For each point, find the intersection point using linear interpolation



$$\begin{cases} x_t = x_0 \\ y_t = \dfrac{y_0 * (F_{01} - F_t) + y_1 * (F_t - F_{00})}{F_{01} - F_{00}} \end{cases}$$

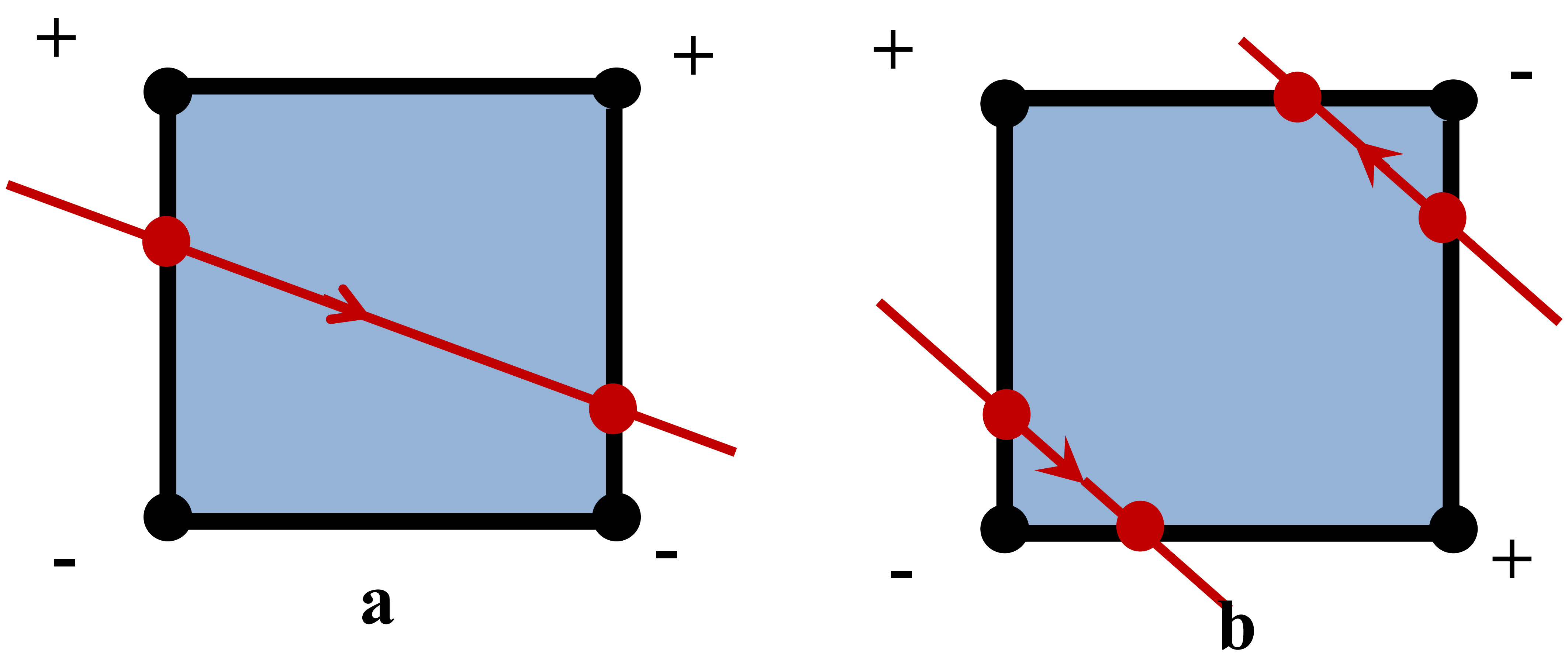
Connect

- vertices of the cell： with only one’+’ or’-’
  - One contour segment



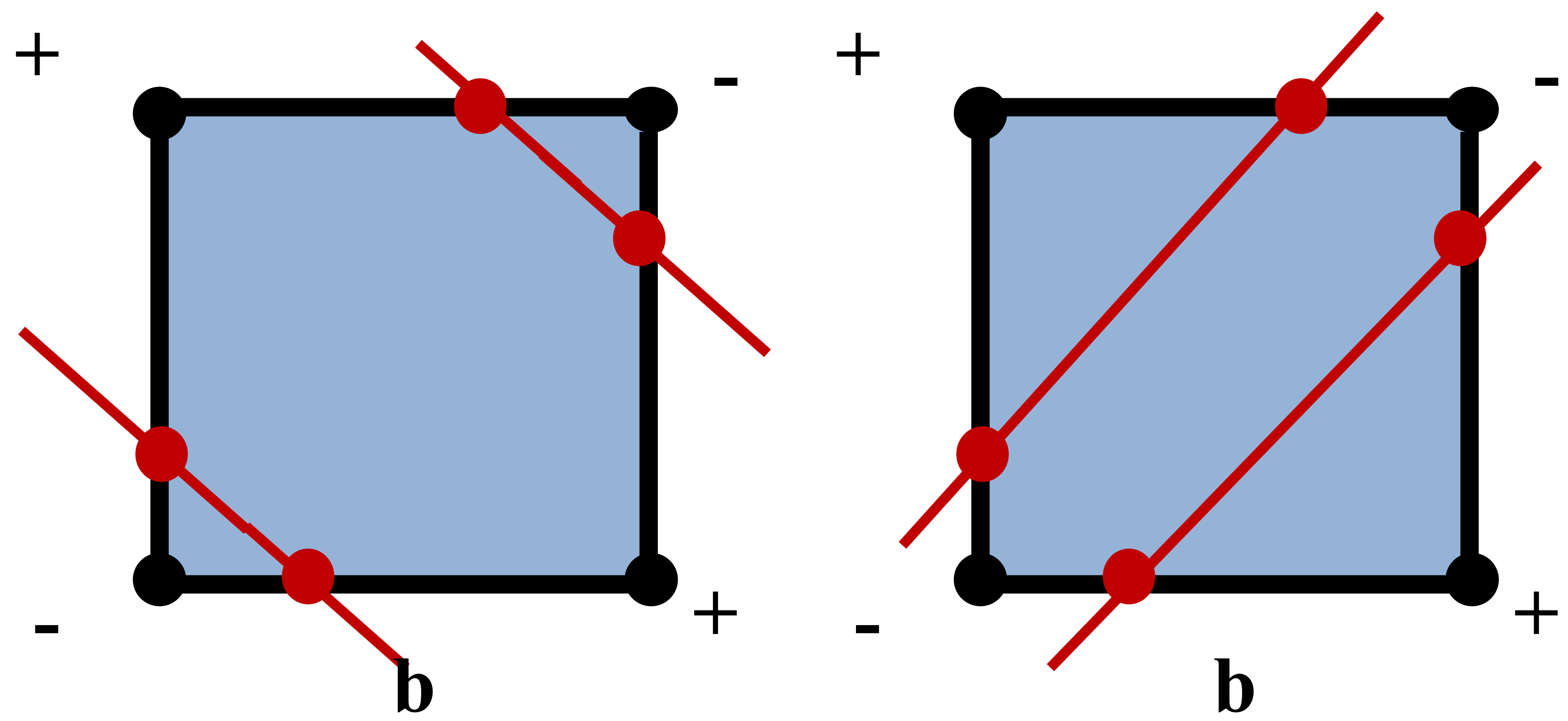
Connect

- vertices of the cell: two ‘+’ or two‘-’
  - One contour segment: Fig. a
  - two contour segments: fig. b
    - Direction

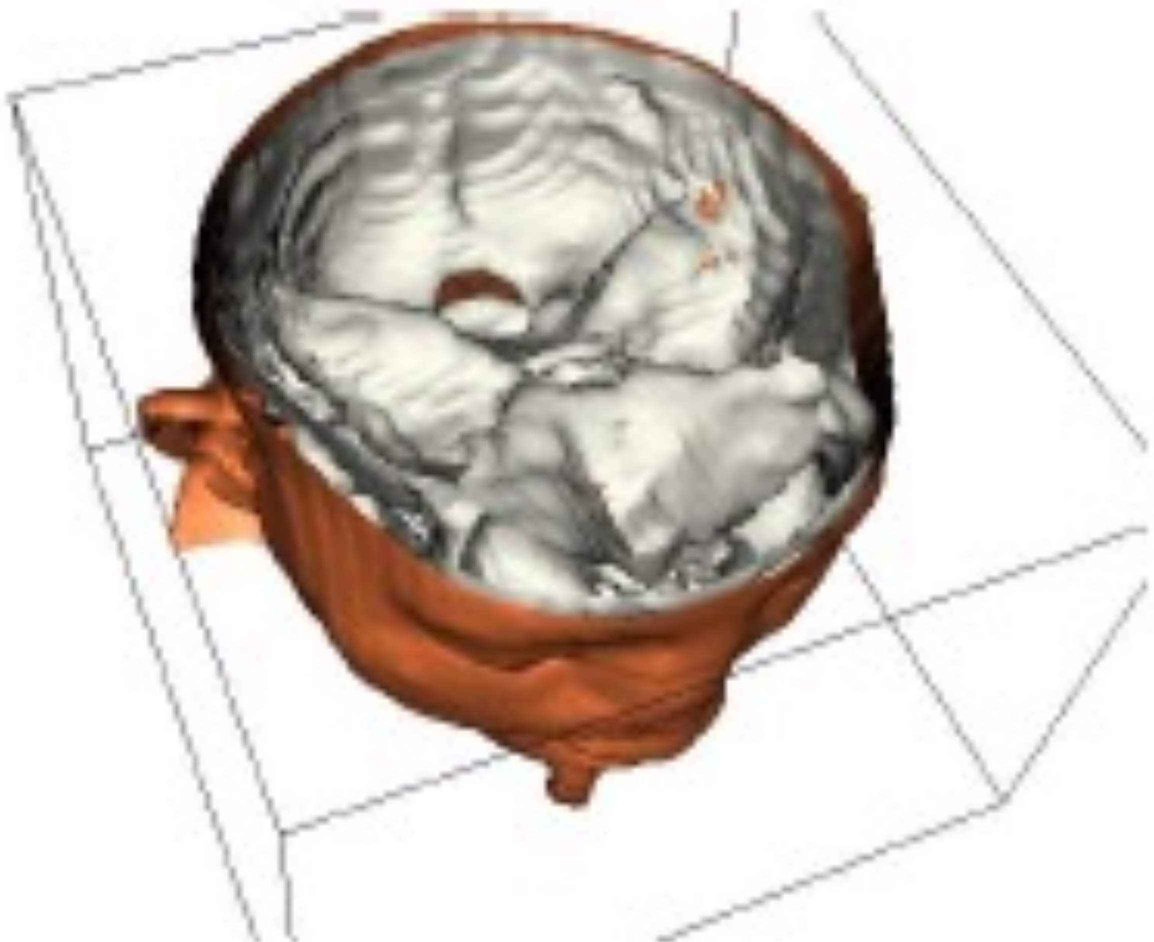


Connect

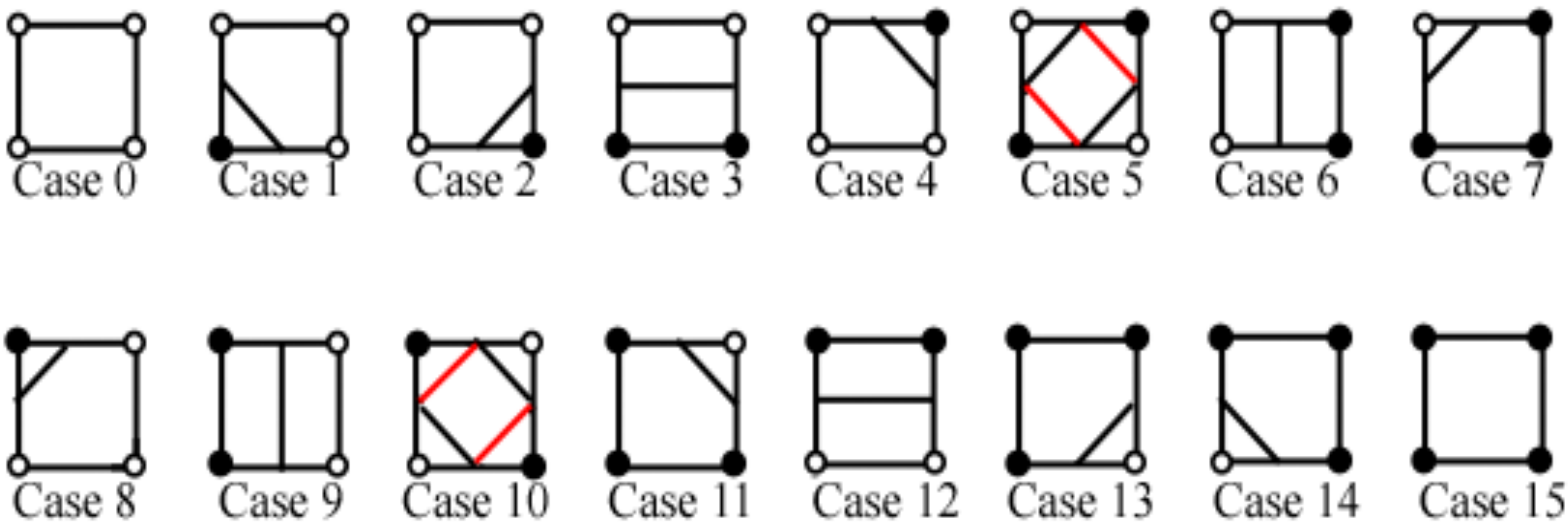
- vertices of the cell: two '+' or two'-'
  - One contour segment: fig. a
  - two contour segments: fig. b
    - Direction
    - No direction



Marching squares

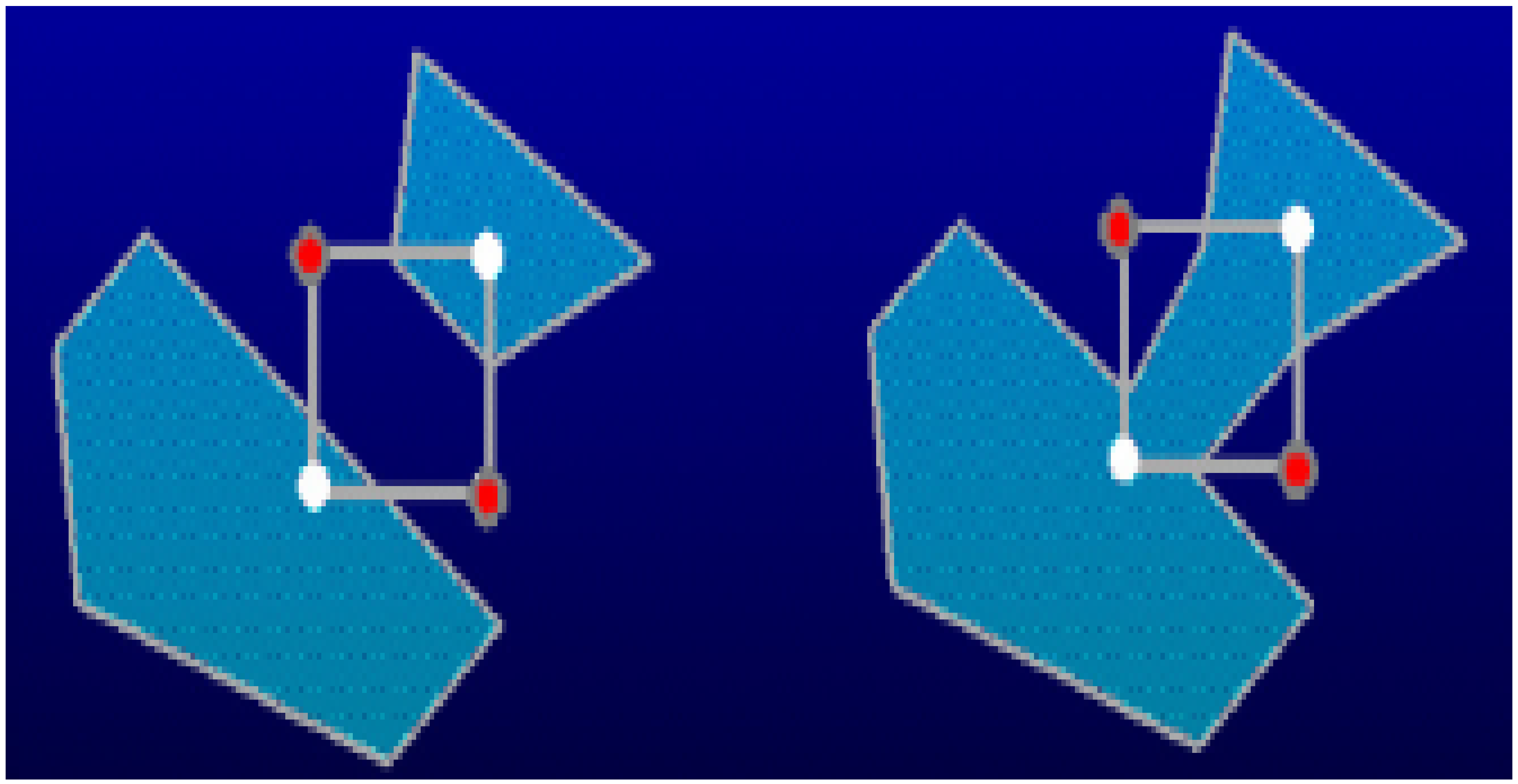


Ambiguities of Contours



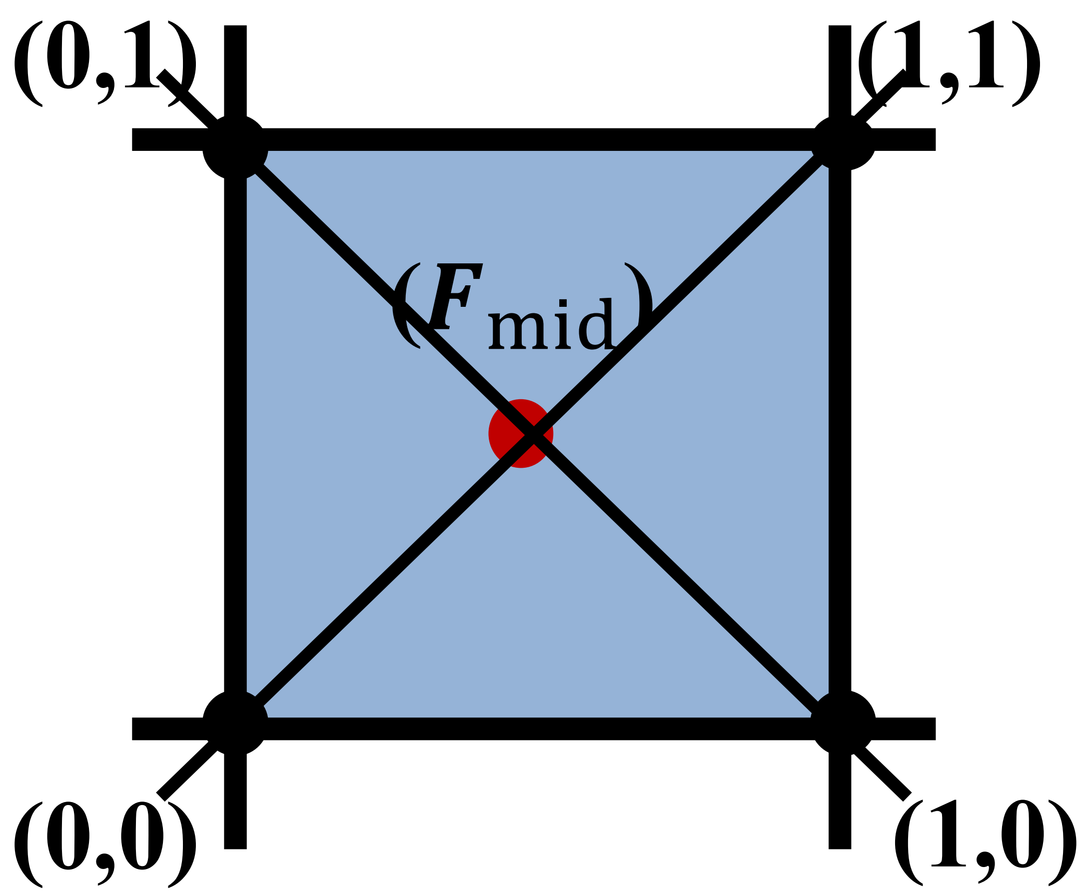
Ambiguities of Contours

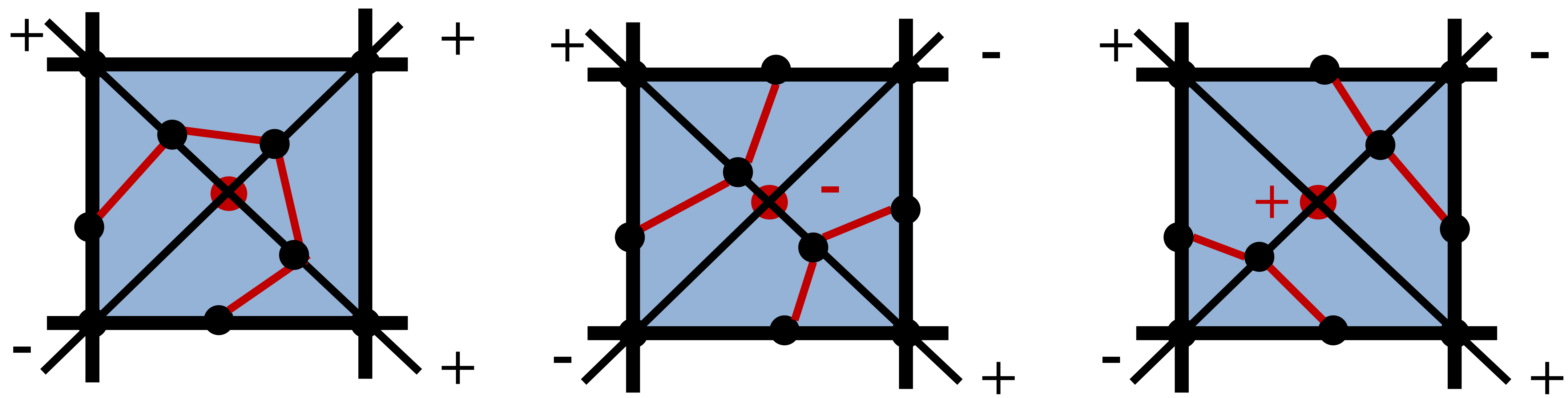
- Contour ambiguity (case 5 and 10)



Solution

- Andrew subdivision
  - Compute $\boldsymbol{F}_{\text{mid}}$





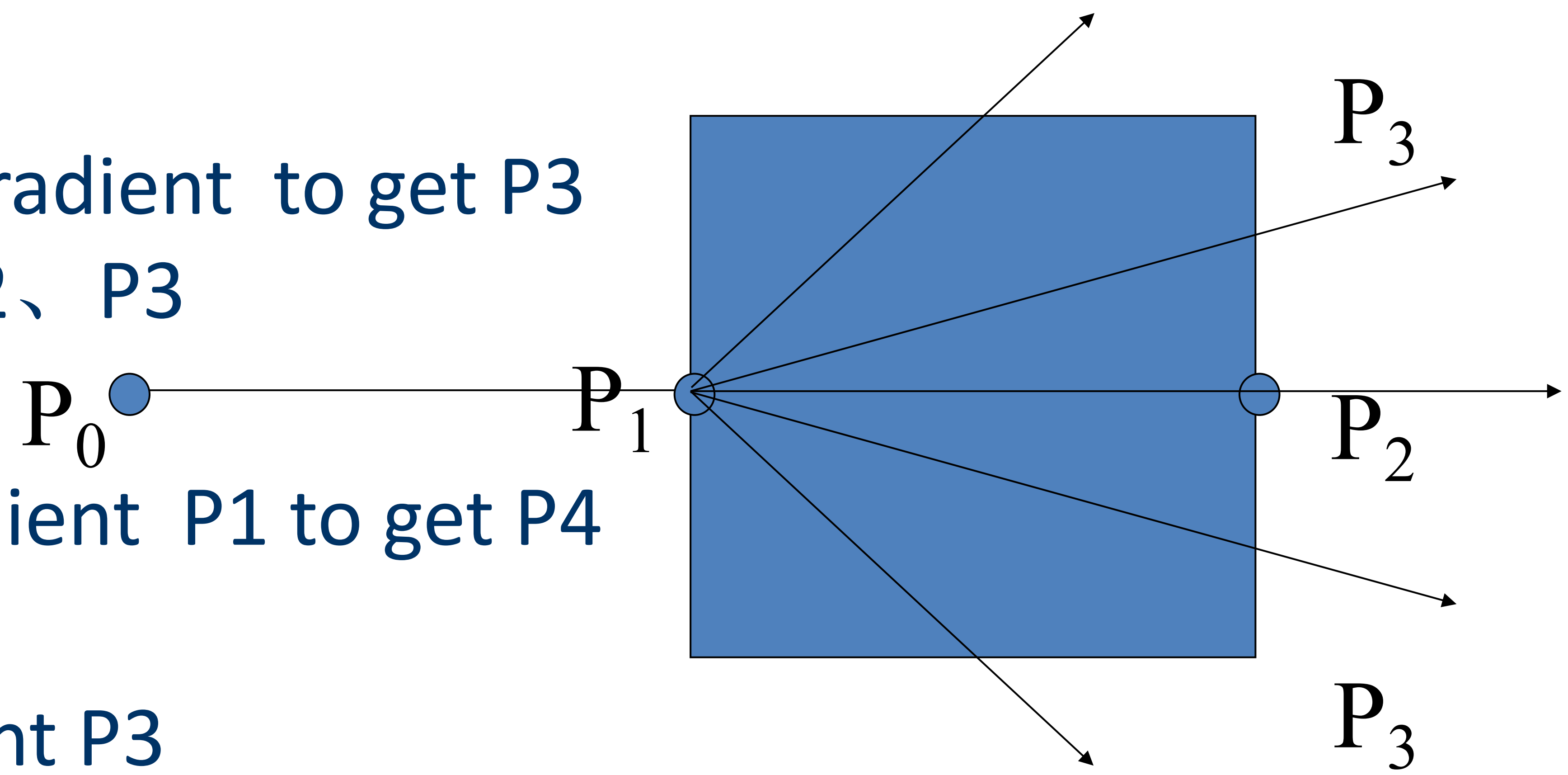
- Definition
- Characteristic
- Pipeline
- Grid Sequence Method
- Grid Free Method

■Grid free method—marching

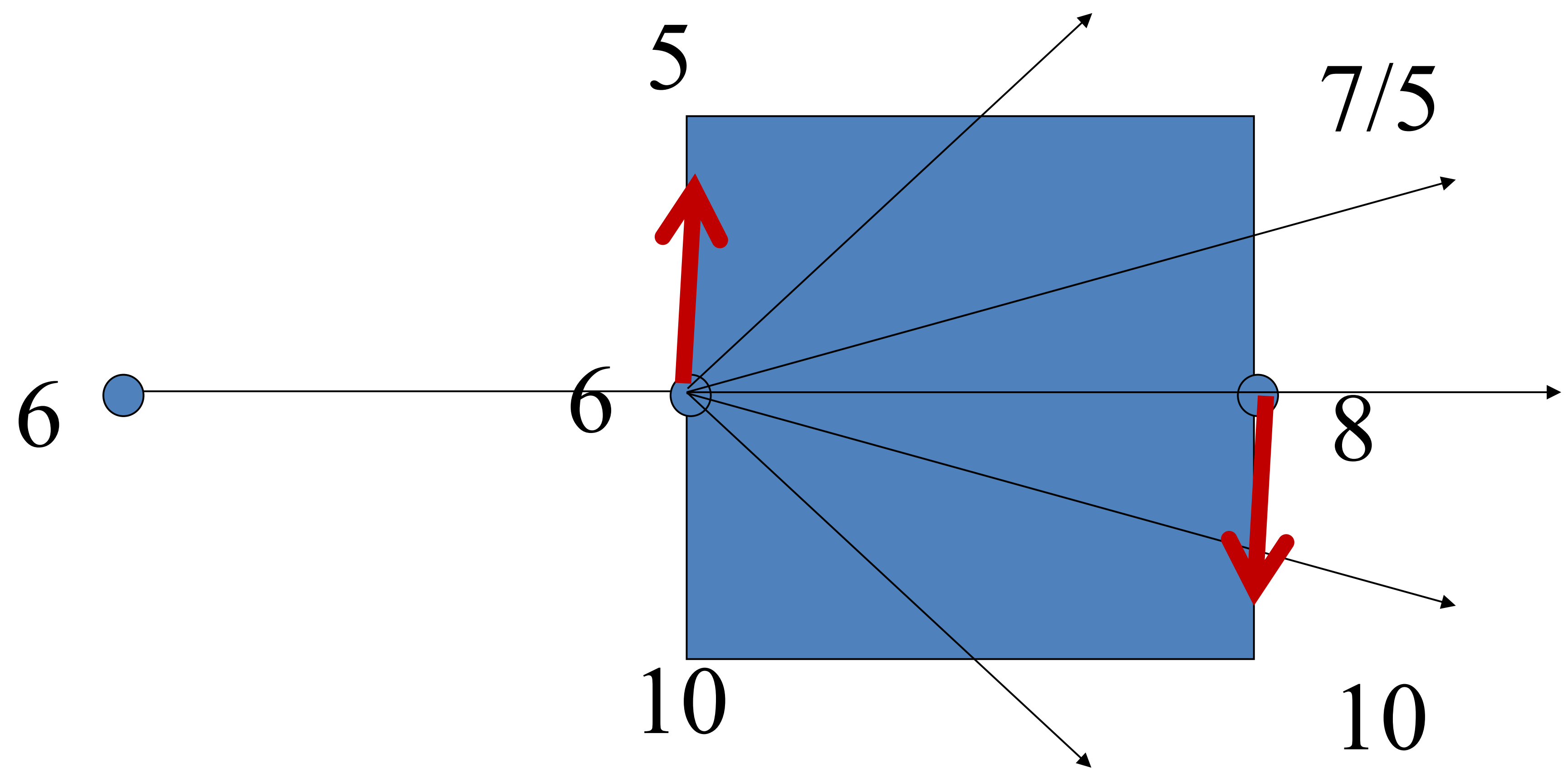
- Given start point P0,|P0P1|=step
- Marching |P2P1|=step
- Classify P2 signal

P2 ‘+’
→ opposite direction of gradient to get P3
P3 ‘-’ ： P is between P2、P3
P3 ‘+’ ：
opposite direction of gradient P1 to get P4

P2 ‘-’
→ the direction of gradient P3



■ Grid free method—marching



grid sequence	gird free
Be suitable for less cell situation	Be suitable for more cell situation
	Depends on the start point selection
Low efficiency	High efficiency

2

Isosurface

- Definition
- Data Acquisition
- Goal
- Marching Cubes

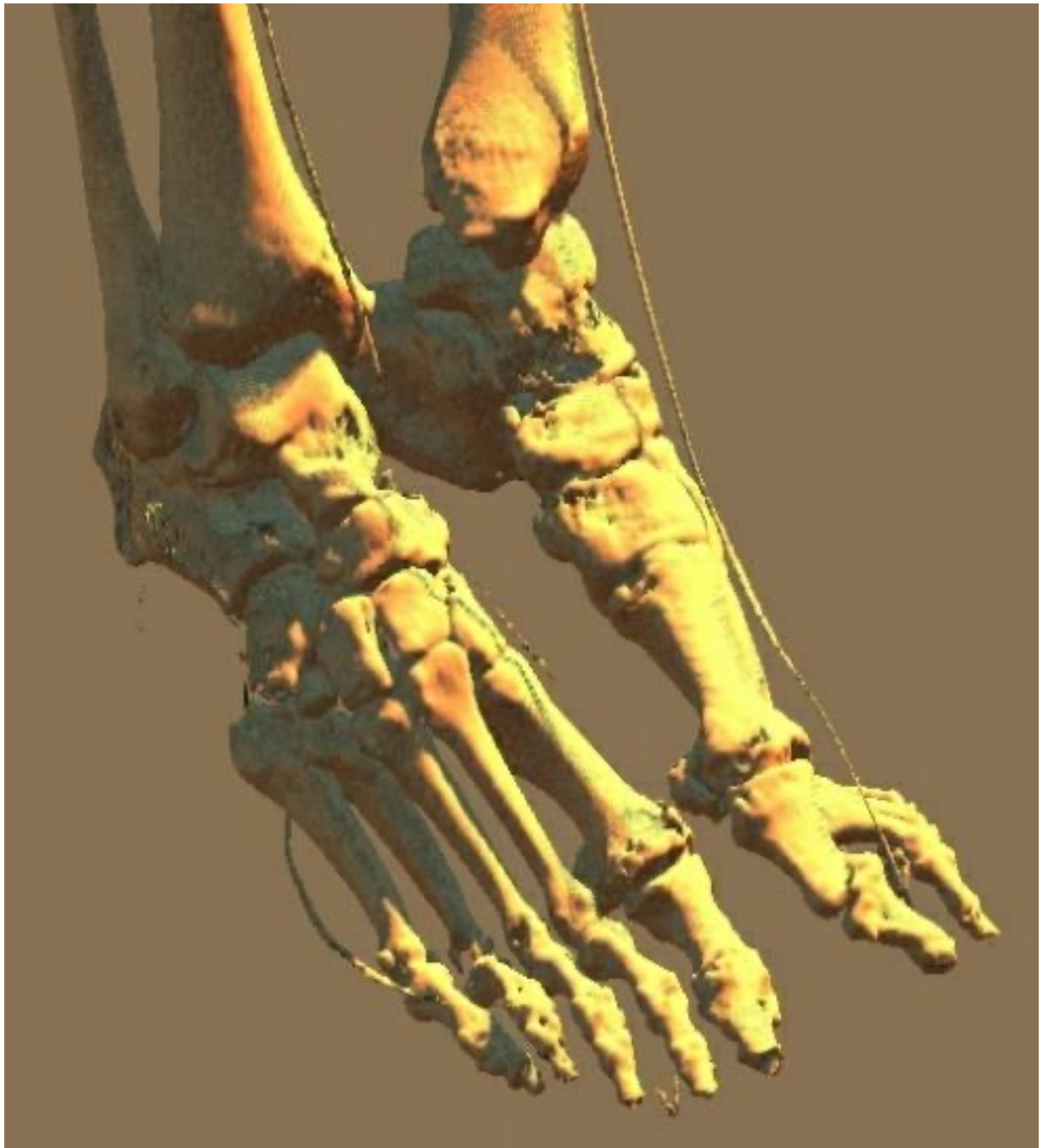
- Definition
- Data Acquisition
- Goal
- Marching Cubes

What is the isosurface

- tetrad point dataset
  - $((x, y, z), value)$
- 3D scalar field
  - $F = F(D)$
    1D-scalar function defined on the volume of 3D space D
- isosurface
  - Set of points where the scalar field F has a given value c:
    $$\{(x \in D: F(x) = c\ )\}$$

What is the isosurface?

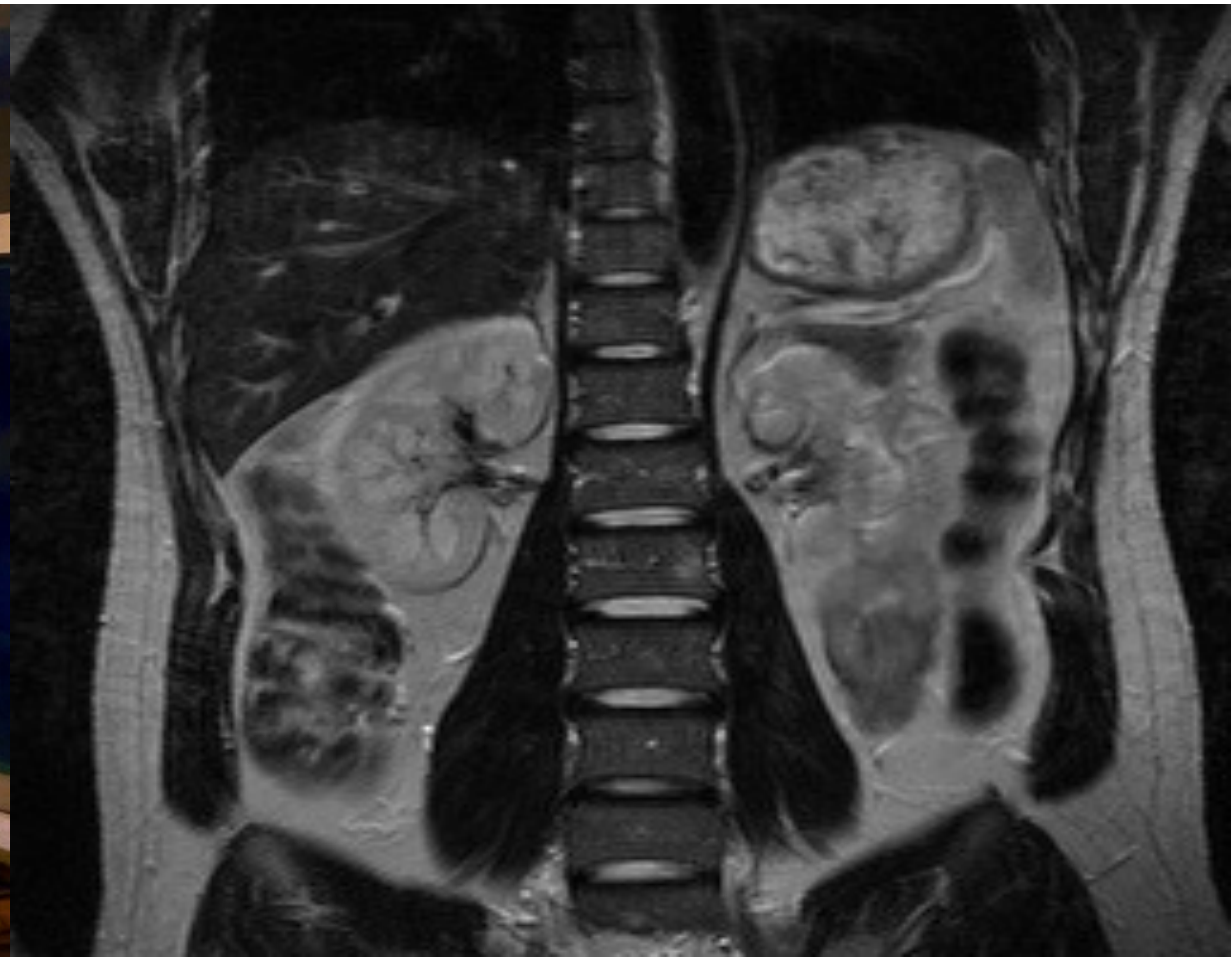
- Conversion of a 3D scalar field into isosurface



- Definition
- Data Acquisition
- Goal
- Marching Cubes

Data Acquisition

- MRI(Magnetic Resonance Imaging)



Data Acquisition

- CT scan (Computer Tomography)

Data acquisition

- Ultrasound

- Definition
- Data Acquisition
- Goal
- Marching Cubes

Goal

- Display iso-surface
  - Surface of constant density
  - Medical visualization
    - Bone , flesh organ densities differ
    - Operator selects desired density

- Definition
- Data Acquisition
- Goal
- Marching Cubes

Marching Cubes

Marching cubes: A high resolution 3D surface construction algorithm

WE Lorensen, HE Cline - ACM Siggraph Computer Graphics, 1987 - dl.acm.org

Abstract We present a new algorithm, called marching cubes, that creates triangle models of constant density surfaces from 3D medical data. Using a divide-and-conquer approach to generate inter-slice connectivity, we create a case table that defines triangle topology. The ...

被引用次数：10081　相关文章　所有 18 个版本　导入BibTeX　更多▾

Marching Cubes

- Motivation
  - Visualization for medical aps
- input
  - Regular grid of points
  - Density values at each point

Marching Cubes

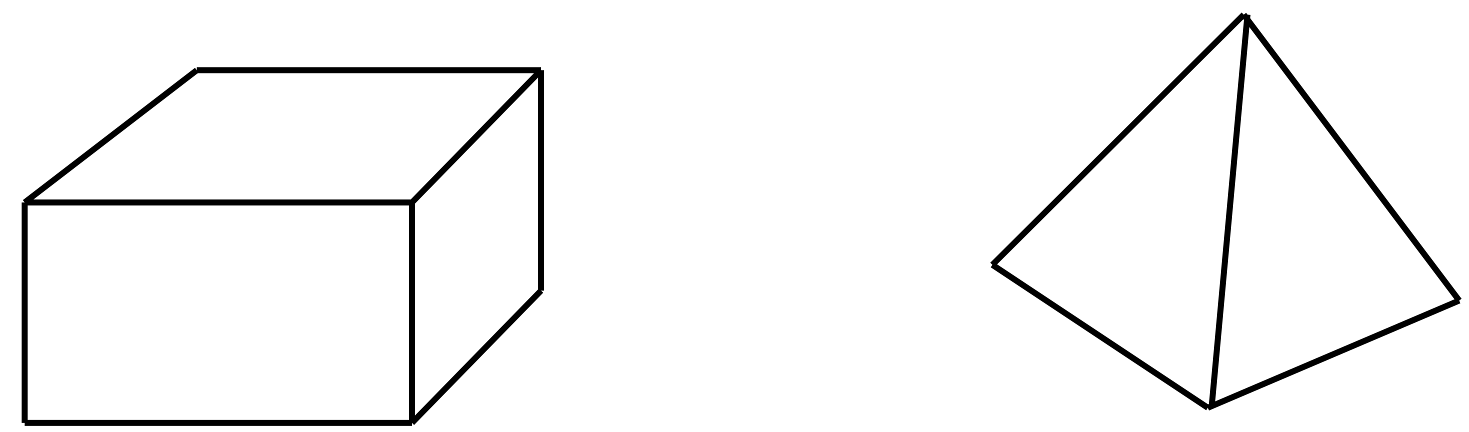
- Outputs triangles
- Surface normal for each vertex
  - Improve rendered appearance

Marching Cubes

- Create a cube
- Classify each voxel
- Build an index
- Lookup edge list
- Interpolate triangle vertices
- Calculate and interpolate normals

Marching Cubes

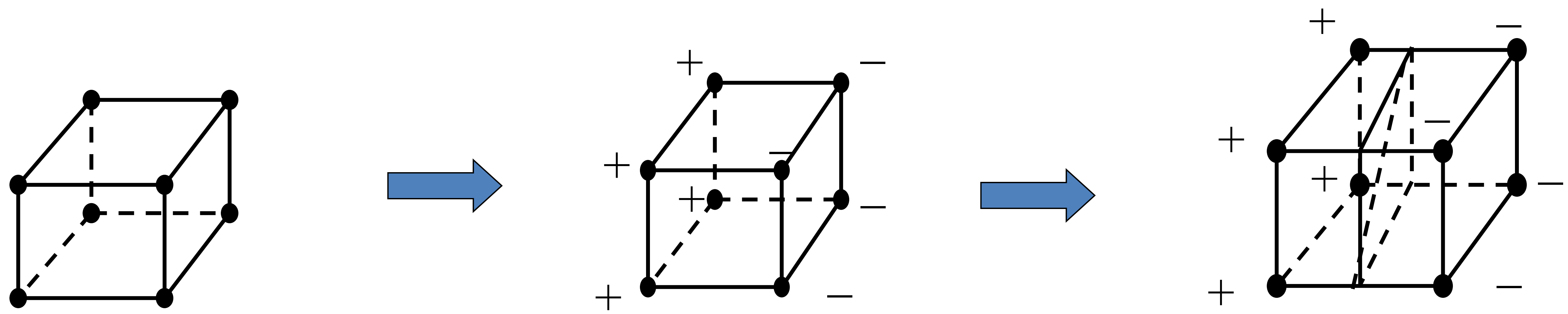
- Extend the 2D marching square algorithm to three dimension
  - 3D cells:



  - Look at one cell at a time

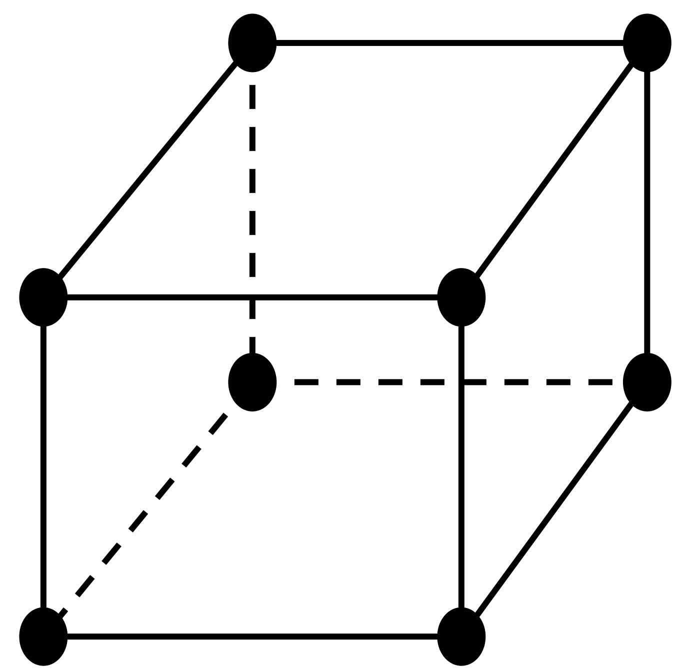
Marching Cubes

- Extend the 2D marching square algorithm to three dimension



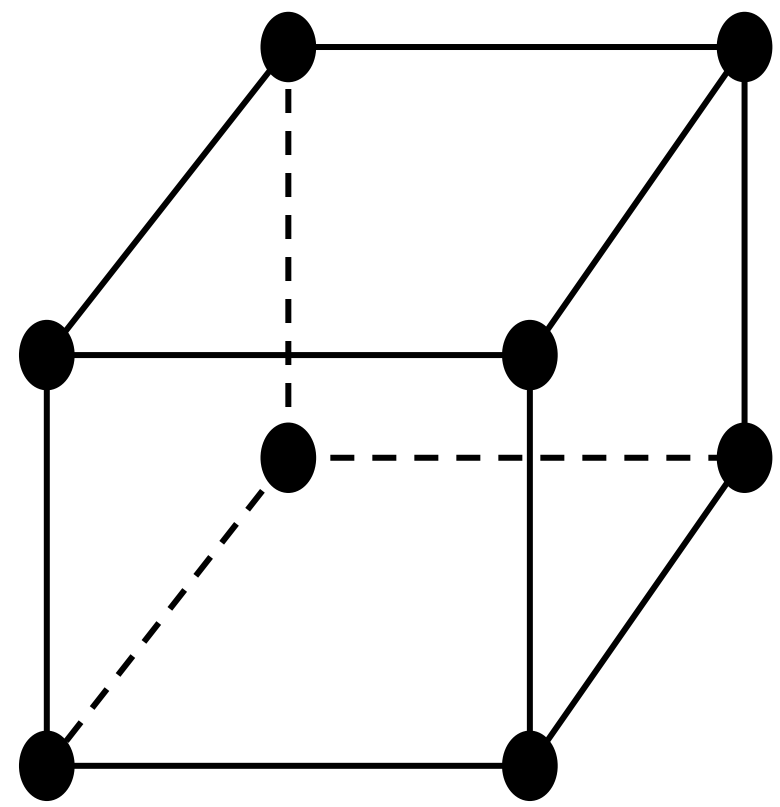
Name Convention of In or Out

- If a value is smaller than the iso-surface value, we call it “Inside”
- If a value is greater than the iso-surface value, we call it “Outside”



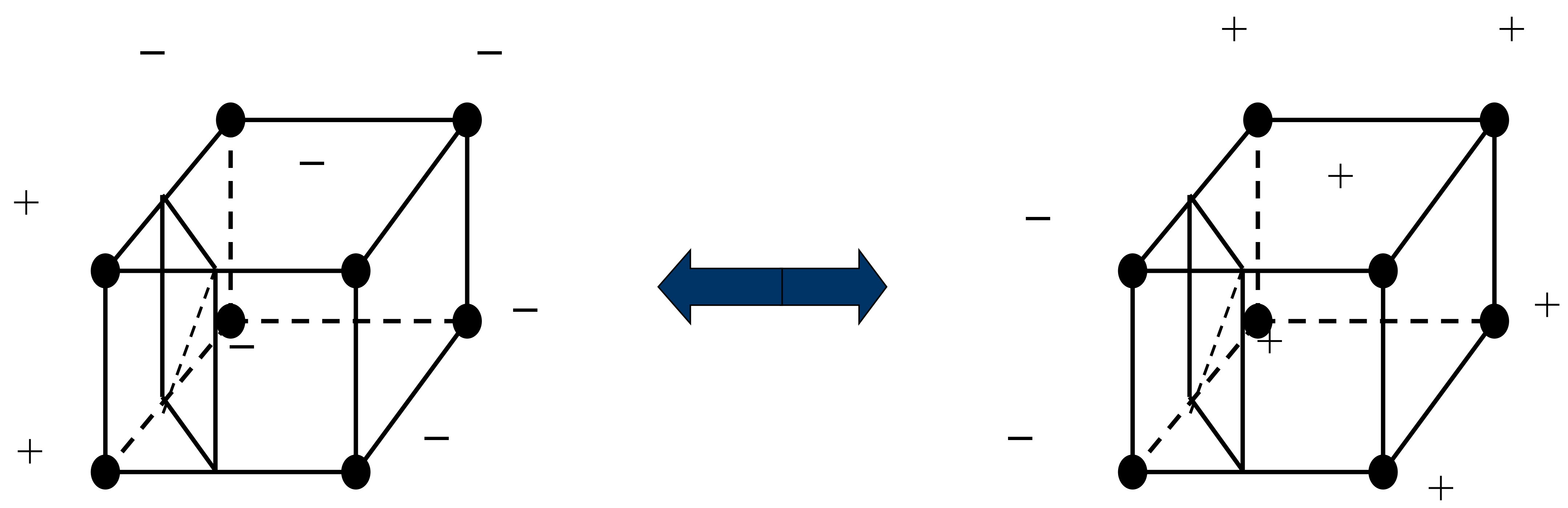
Marching Cubes

- 8 vertices in each voxel
- it is: 2^8 =256 cases
- Build a look-up table



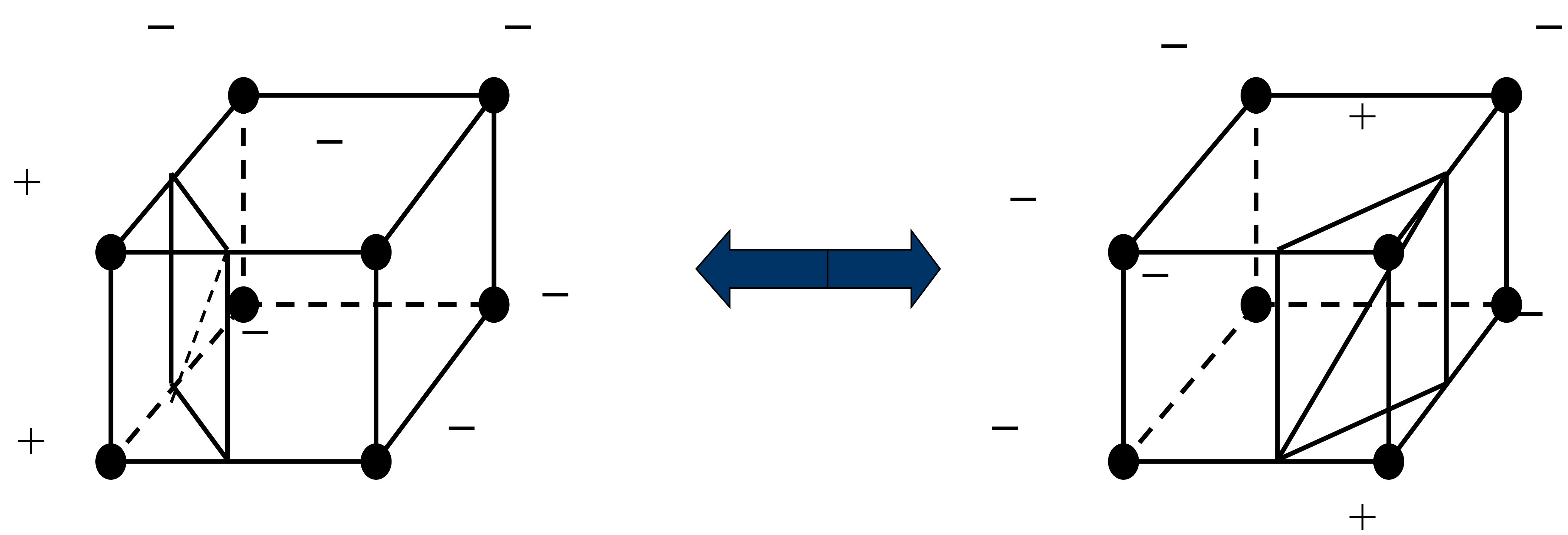
Marching Cubes

- Value Symmetry



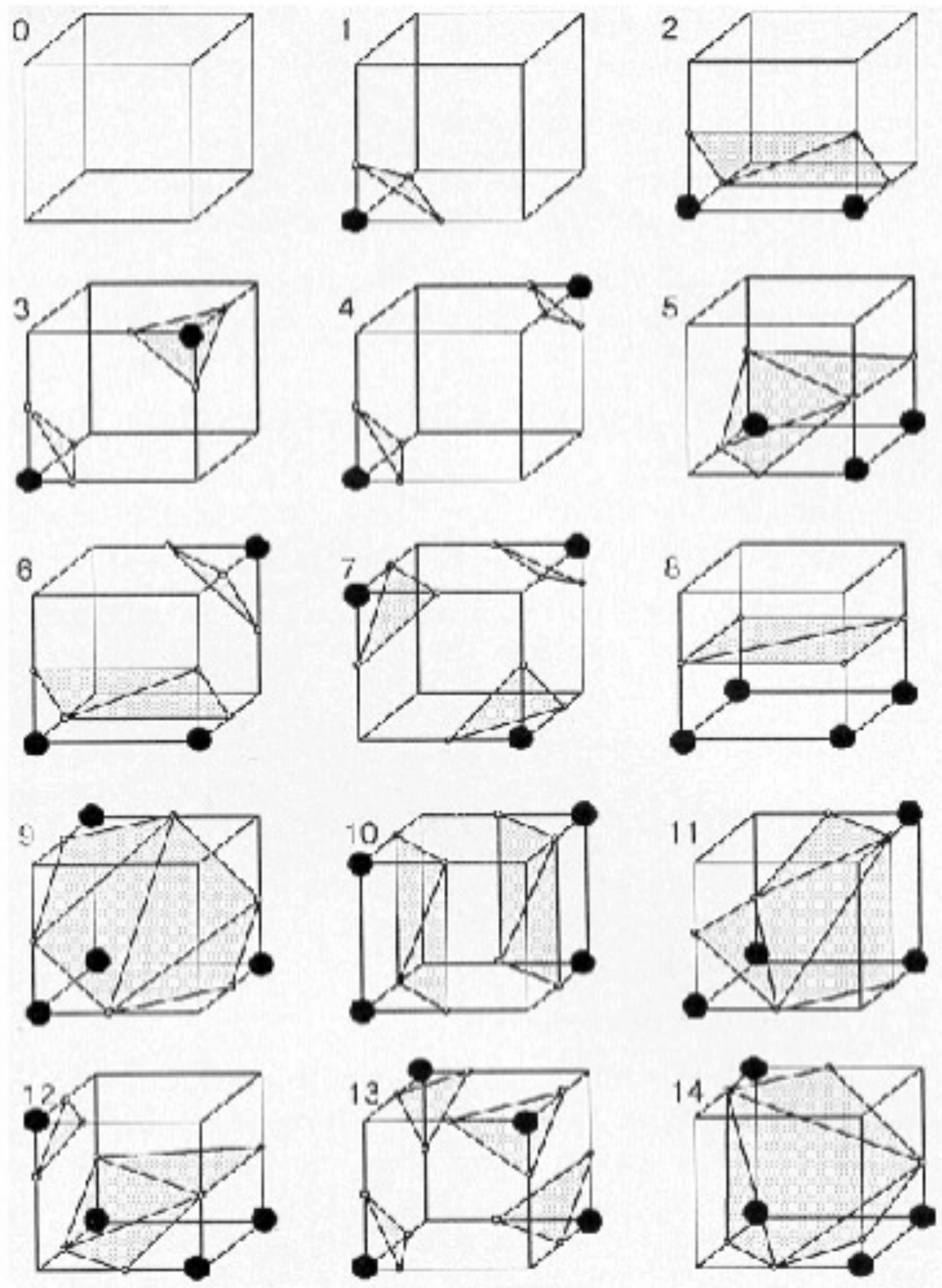
Marching Cubes

- Rotation Symmetry



Marching Cubes

- Case table： 256 →15



Marching Cubes

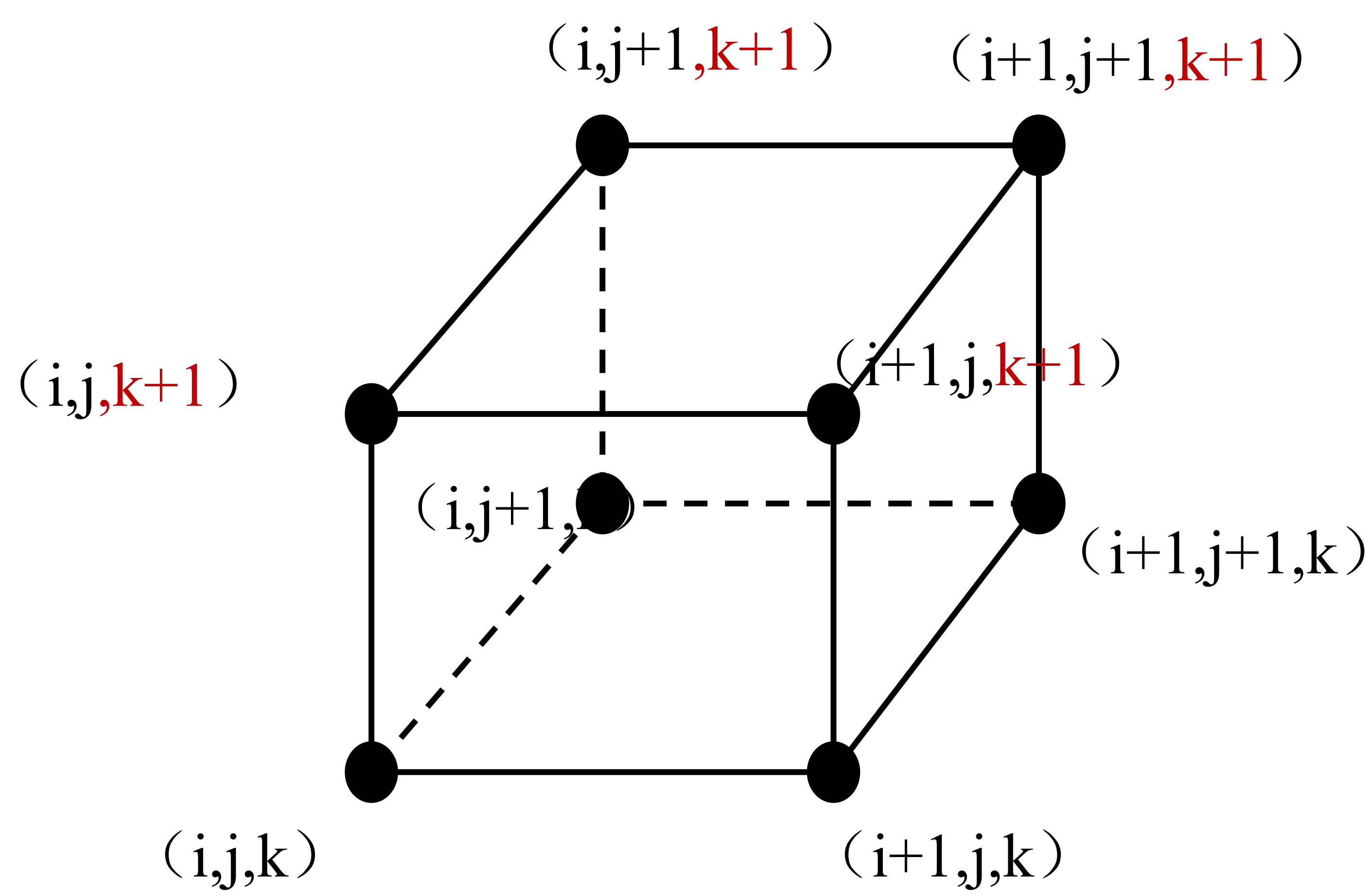
- Create a cube
- Classify each voxel
- Build an index
- Lookup edge list
- Interpolate triangle vertices
- Calculate and interpolate normals

Create a Cube

- From medical data slice

  Four vertices from slice K

  And the other four vertices from slice K+1

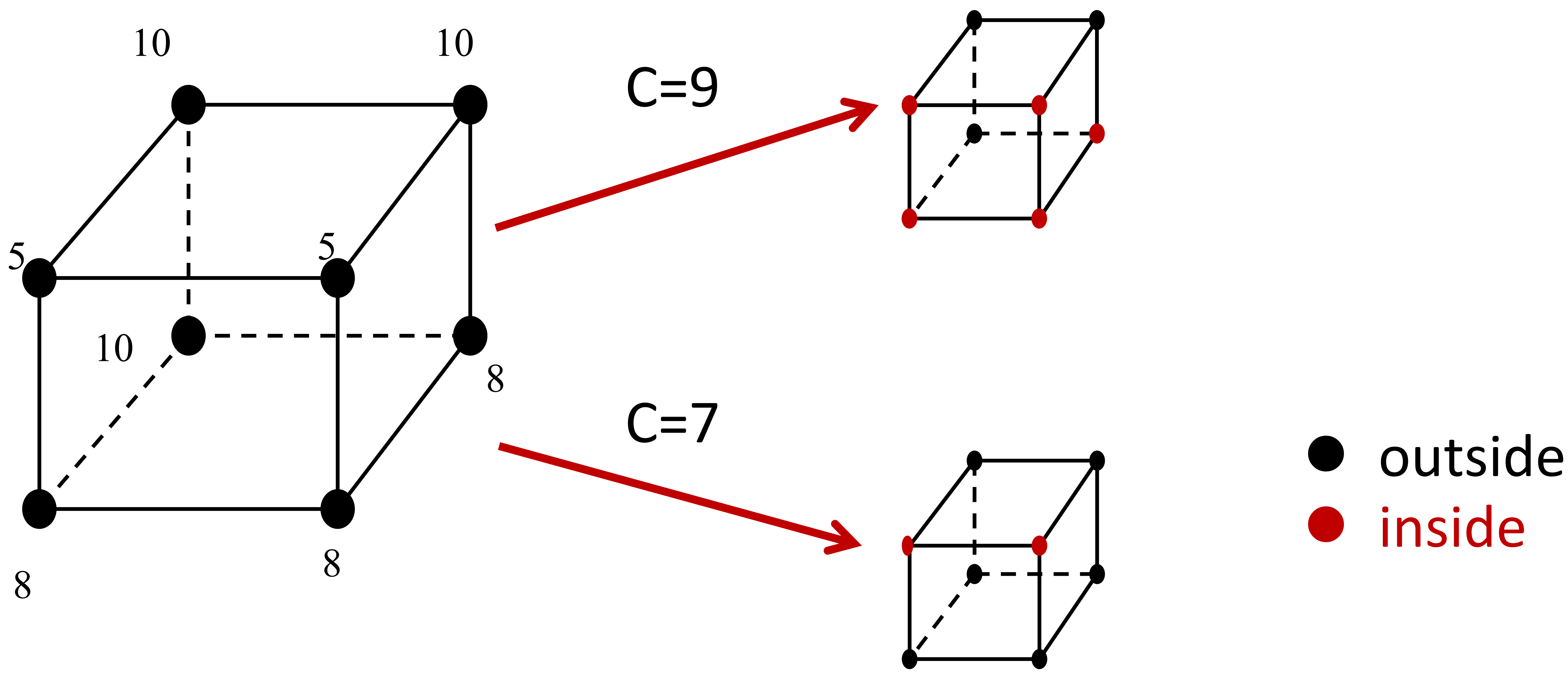


Classify Each Voxel

- Binary classification of each vertex of the cube

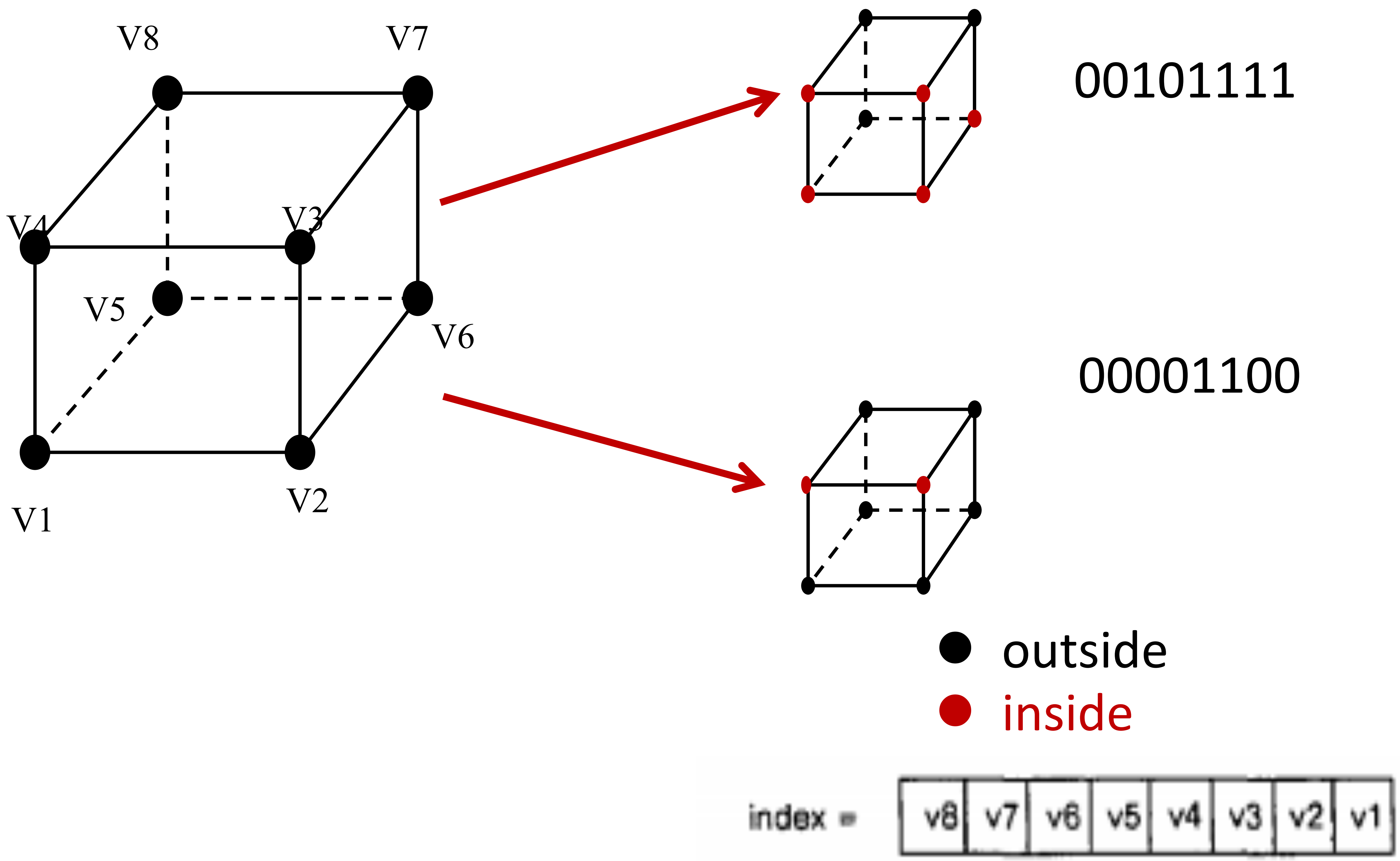
  Outside the surface

  Inside the surface



Build an Index

- Use the binary labeling of each voxel to create an 8-bit index



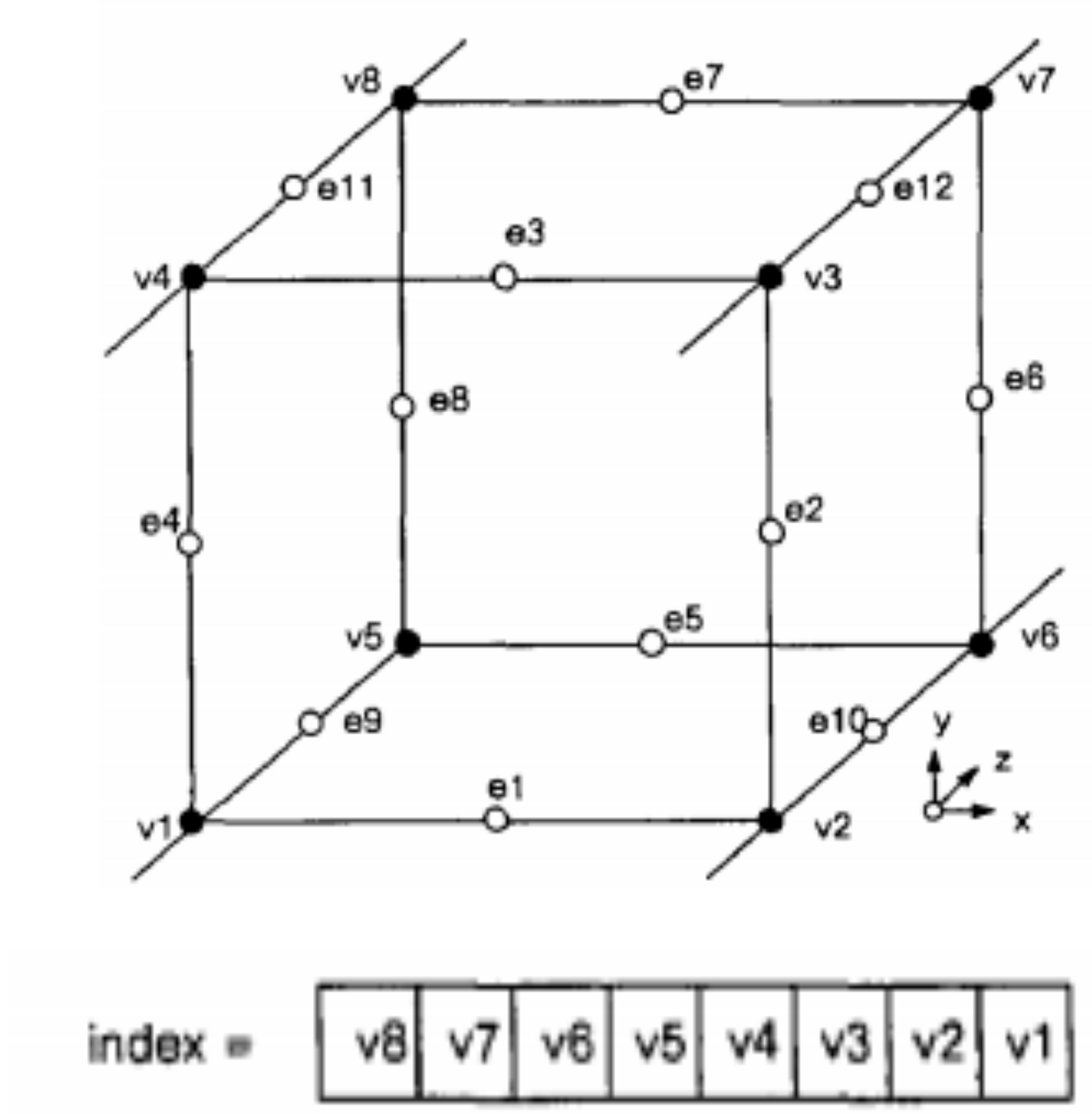
Look-up the table

- Given the index for each cell, a table lookup is performed to identify the edges that has intersections with the iso-surface

- Only 15 patterns

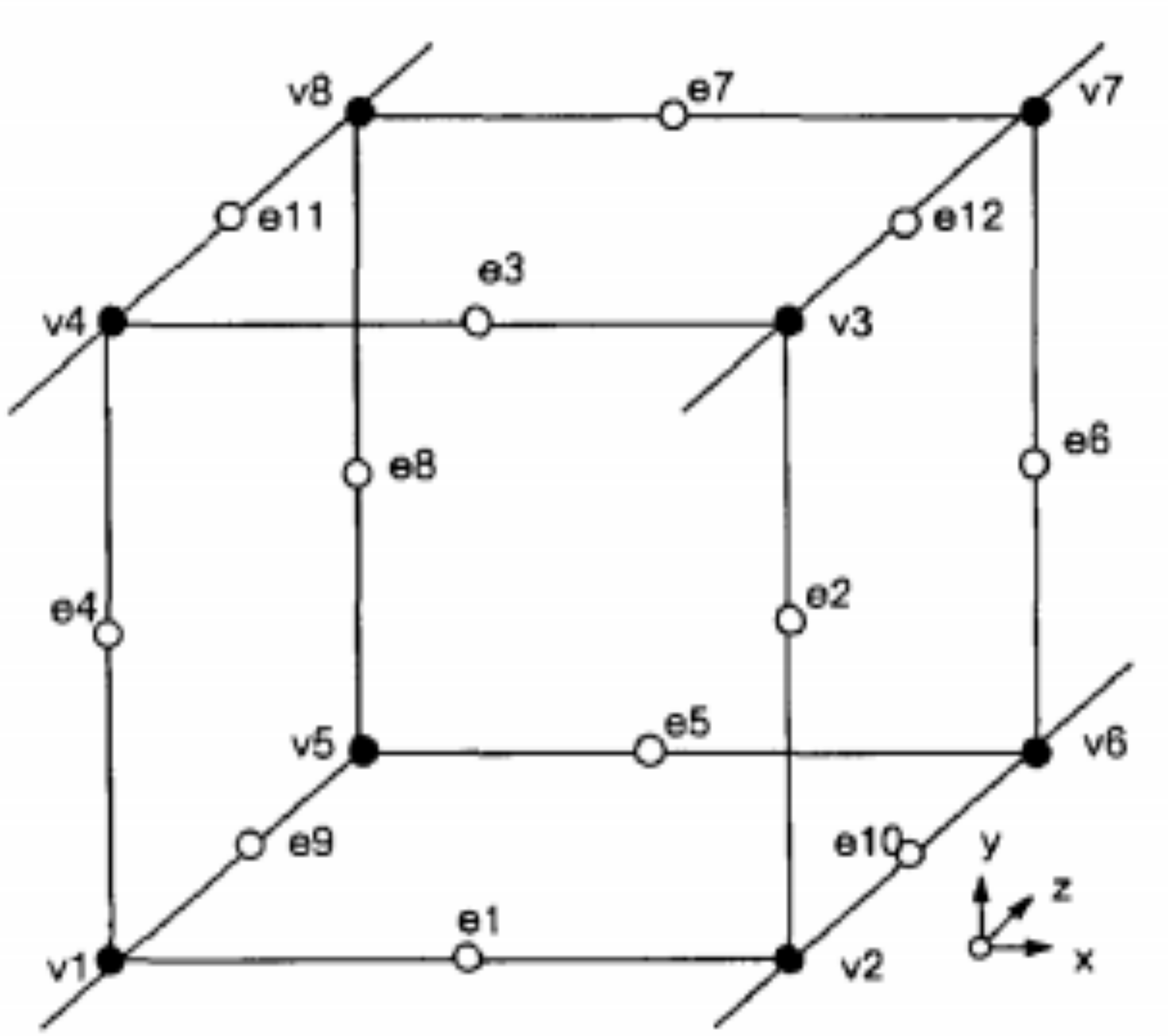
Index : ordered 8-bits of in/out labels from cube corners

Output: which edges intersected, triangles formed



Look-up the table

- Given the index for each cell, a table lookup is performed to identify the edges that has intersections with the iso-surface



index	Intersection edges
0	NULL
1	
…	

example

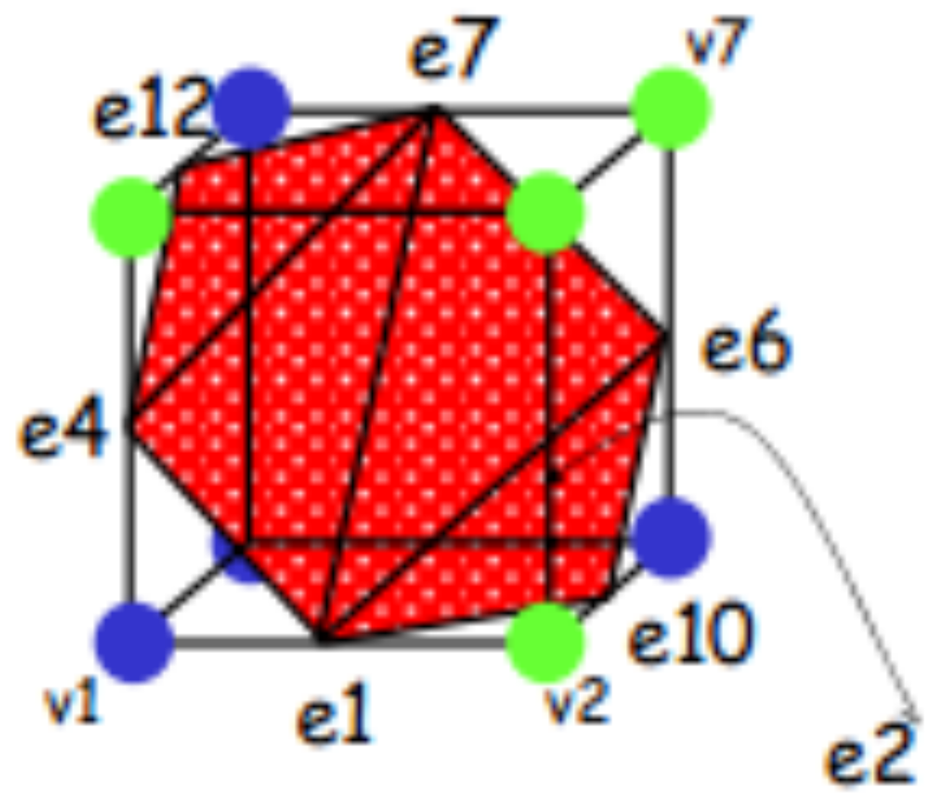
- Index=10001101

  Triangle 1=$e_{4,}e_7, e_{12}$

  Triangle 1=$e_{1,}e_7, e_4$

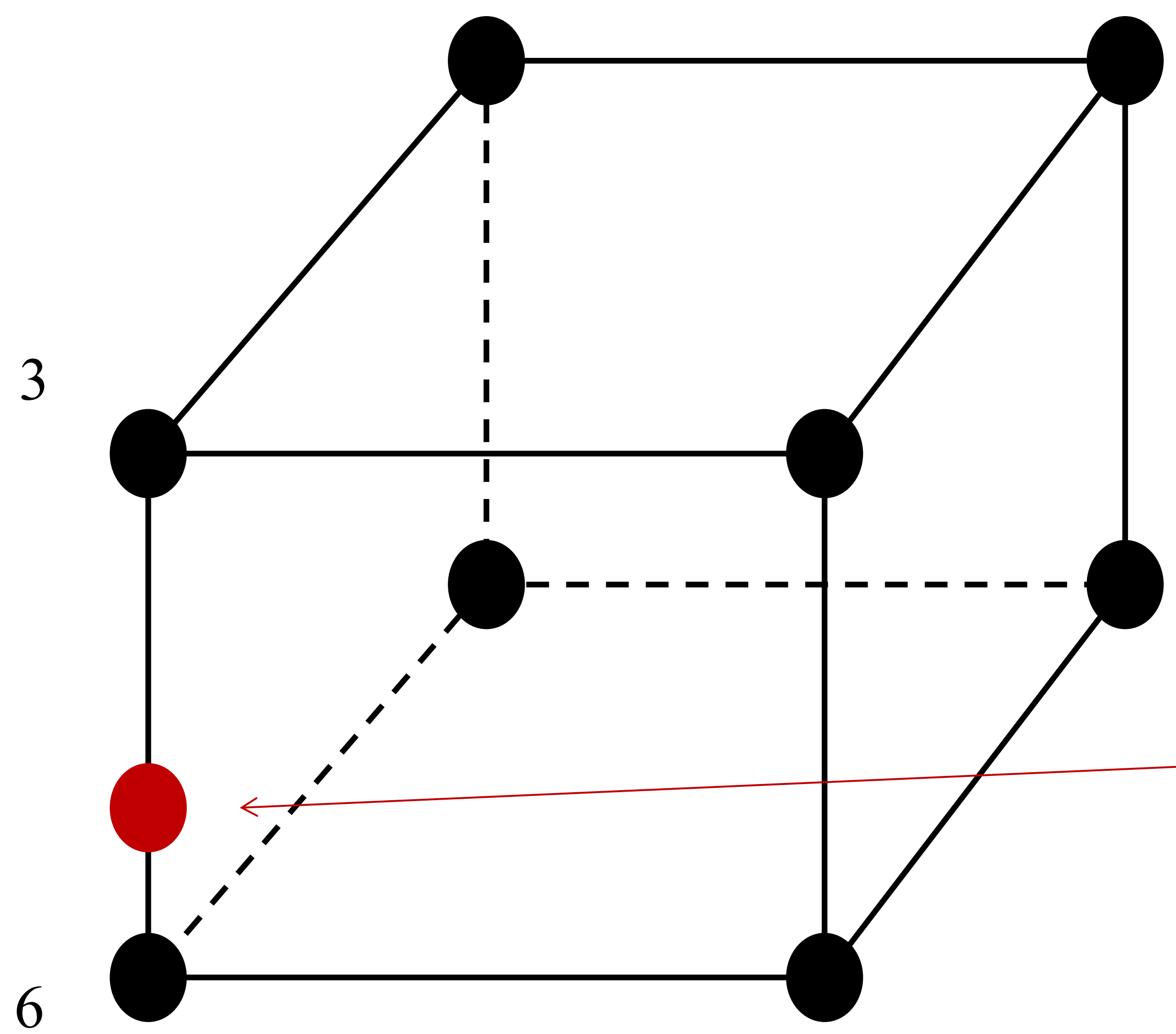
  Triangle 1=e_1, e_6, e_7

  Triangle 1=e_1, e_{10}, e_6



interpolation

- For each edge , find the vertex location using linear interpolation

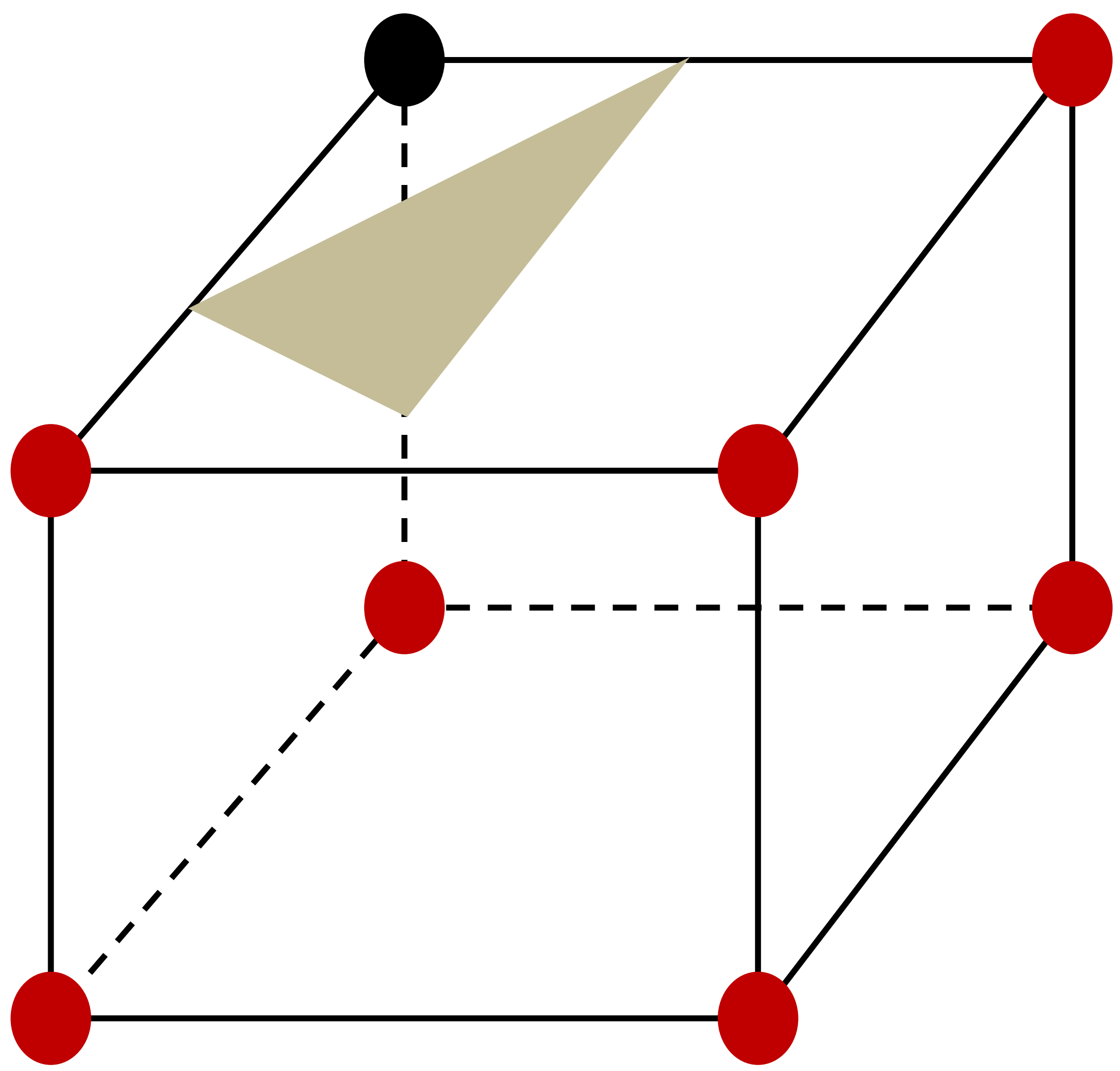


$$x = i + \left(\frac{T - V[i]}{V[i+1] - V[i]}\right)$$

Intersection position
for isosurface density 5

interpolation

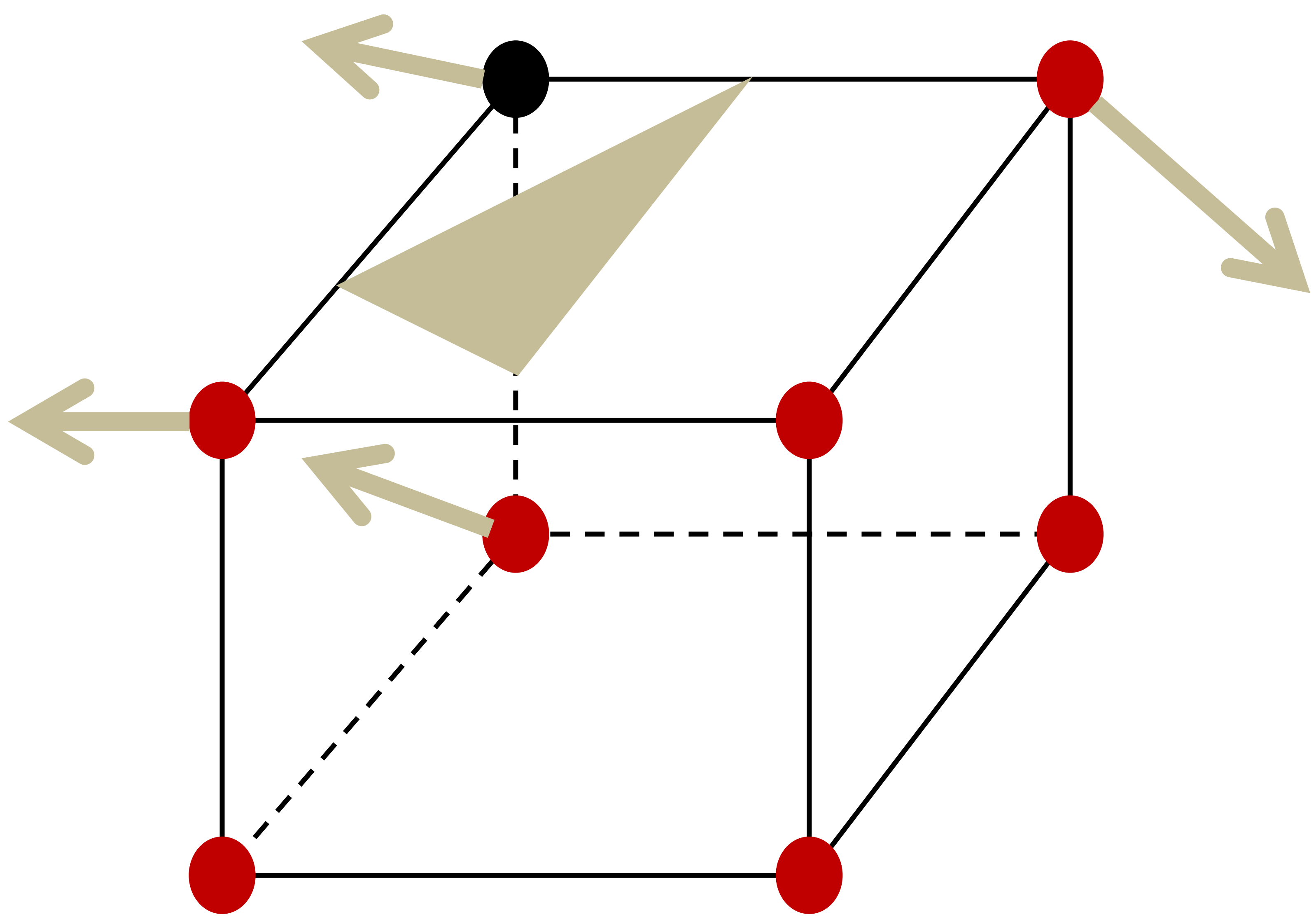
- For each edge , find the vertex location using linear interpolation



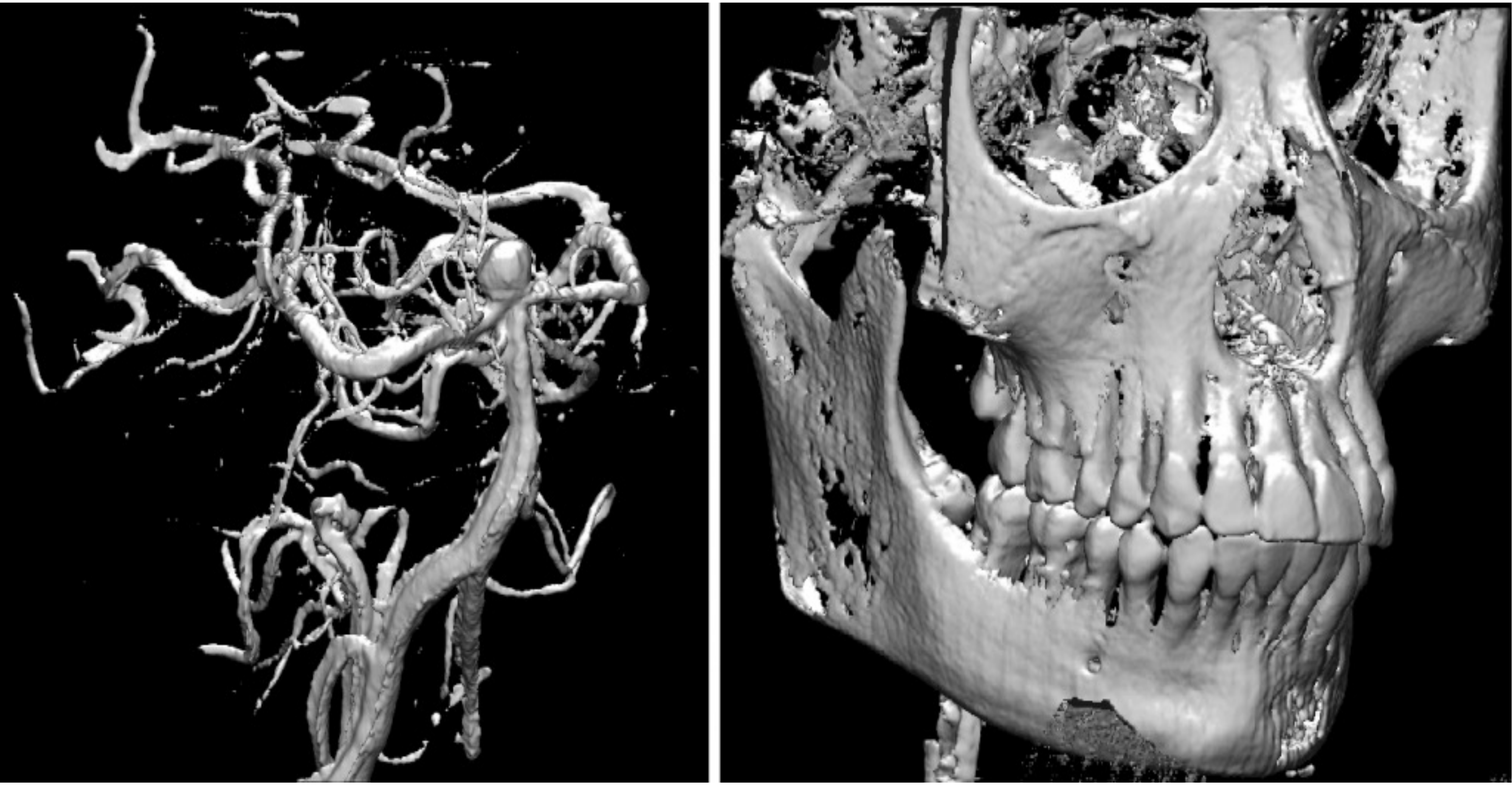
$$x = i + \left(\frac{C - V[i]}{V[i+1] - V[i]}\right)$$

Compute Normals

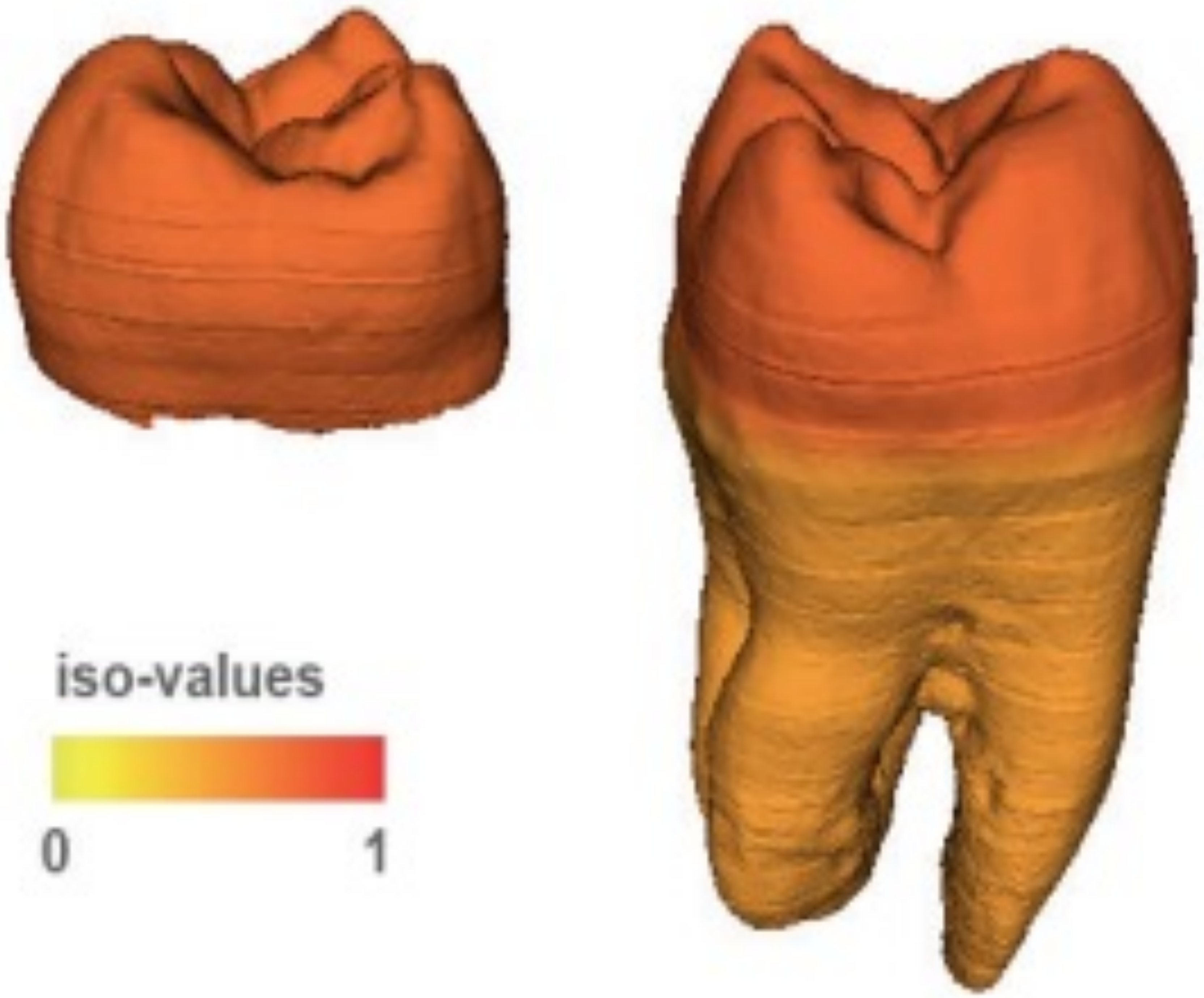
- Want to set to normal to iso-surface
- Gradient direction= normal direction for iso-surfaces
- Calculate gradient at each corner
- Interpolate to edge intersection

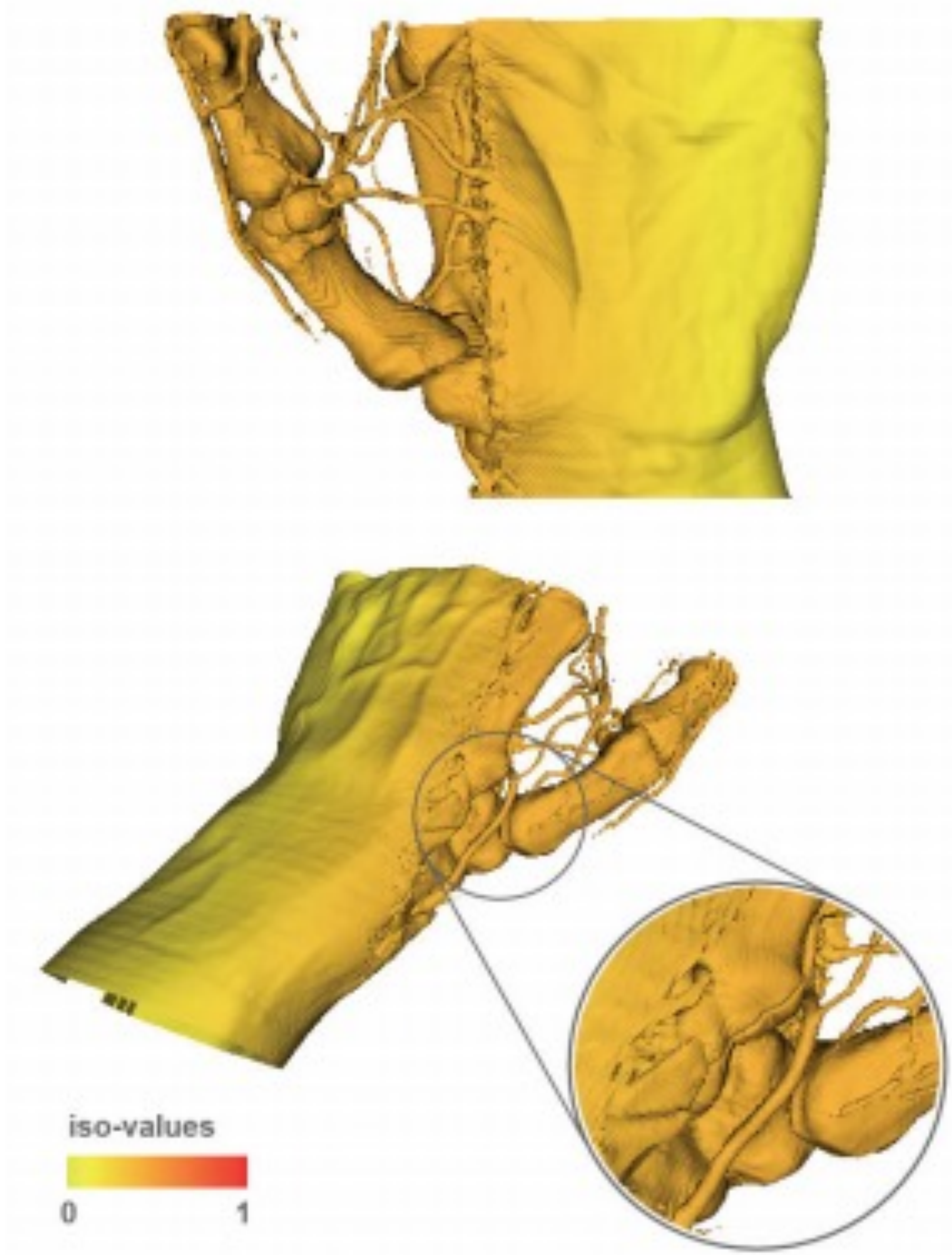


$$Gx = V_{i-1,j,k} - V_{i+1,j,k}$$
$$Gy = V_{i,j-1,k} - V_{i,j+1,k}$$
$$Gz = V_{i,j,k-1} - V_{i,j,k+1}$$

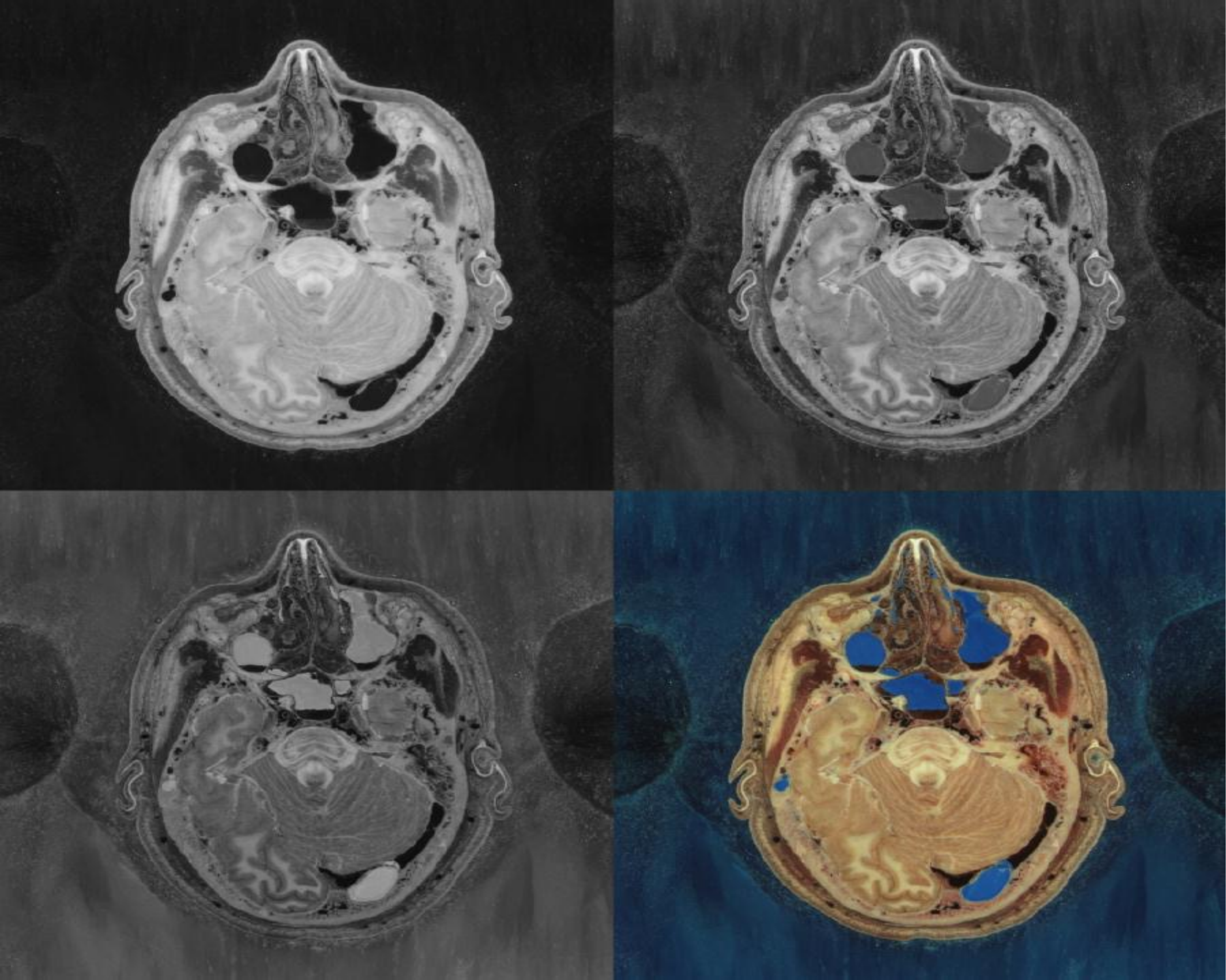


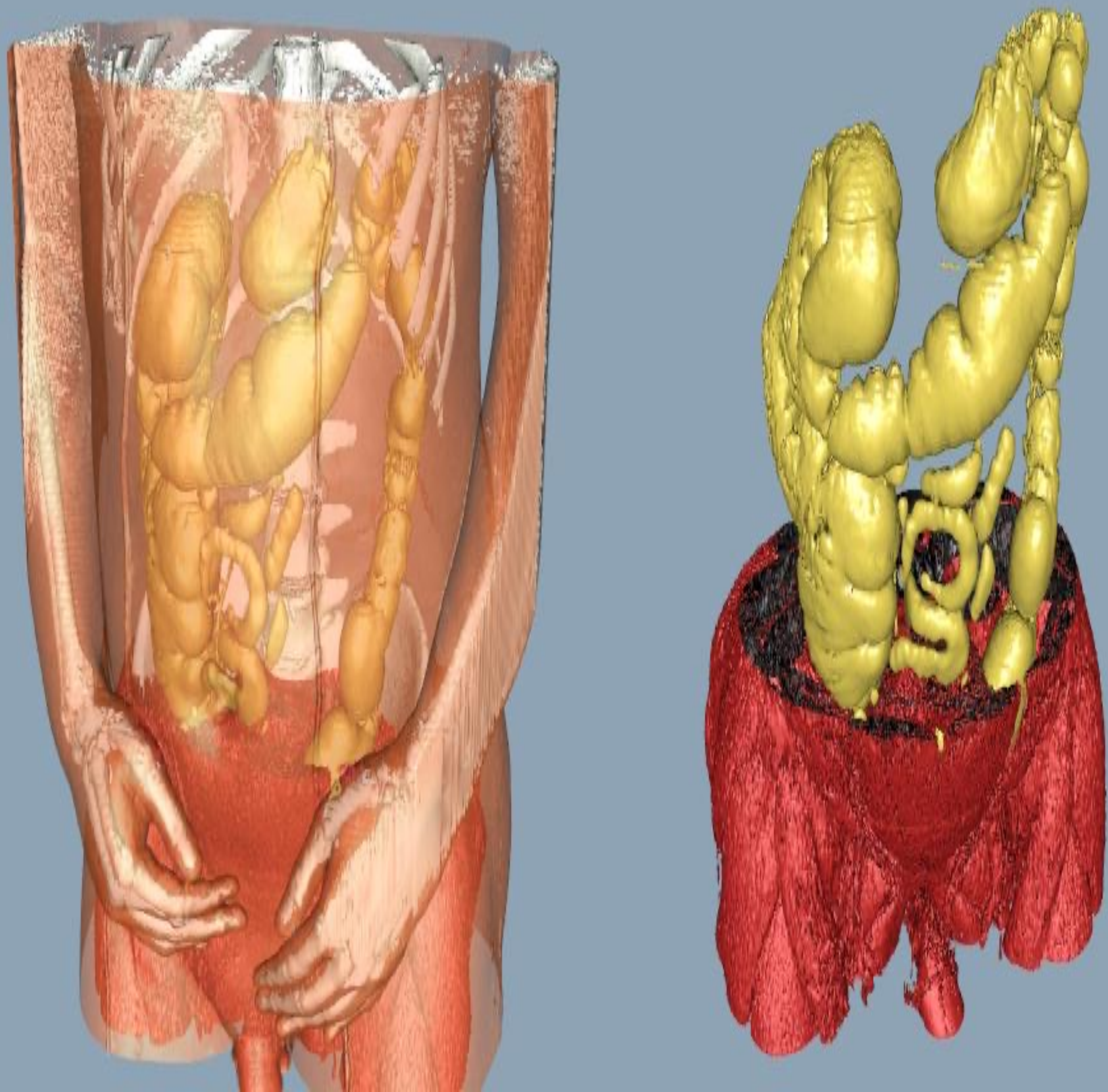
*Marching Cubes implementation using OpenCL and OpenGL





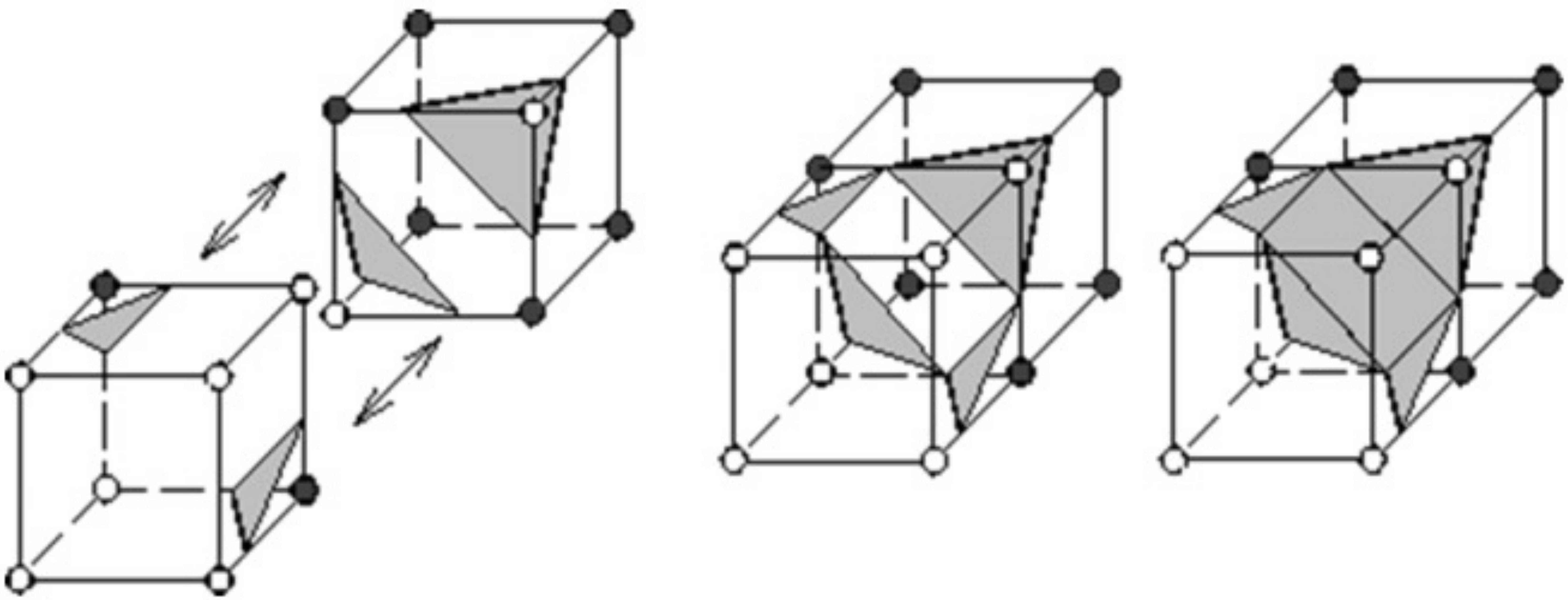
* Locally adaptive marching cubes through iso-value variation





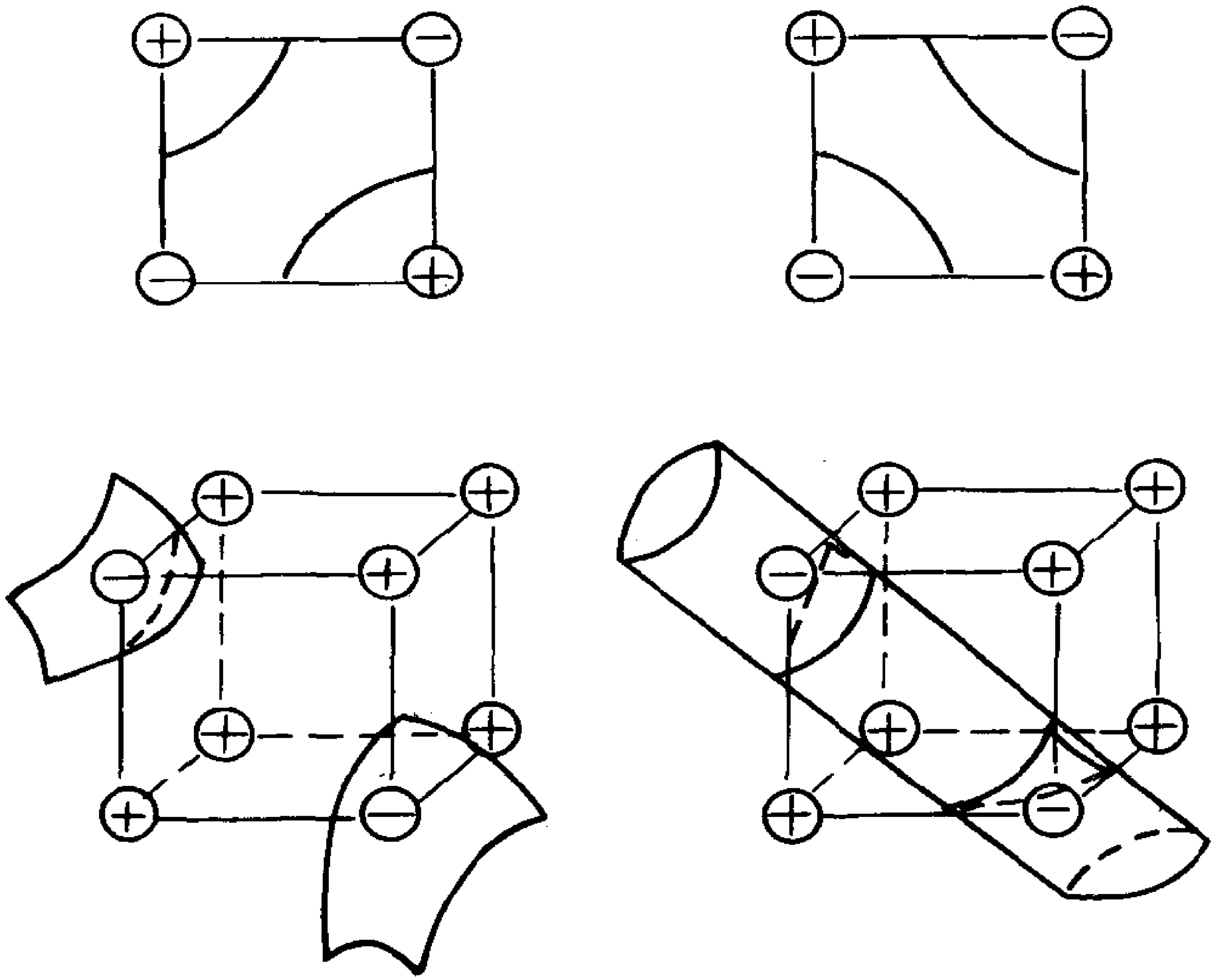
Topology Problems

- E.g. holes in output



Ambiguous Cases

- 3,6,7,10,12,13



Solutions

- There are many solutions available – we present a method called:

Asymptotic Decider by Nielson and Hamann (IEEE Vis'91)

Marching Cubes

```
static int const HexaEdges[12][2] =
{   {0,1}, {1,2}, {2,3}, {3,0},
    {4,5}, {5,6}, {6,7}, {7,4},
    {0,4}, {1,5}, {3,7}, {2,6}
};
typedef struct {
    EDGE_LIST HexaEdges[16];
} HEXA_TRIANGLE_CASES;

/* Edges to intersect.  Three at a time form a triangle. */
static const HEXA_TRIANGLE_CASES HexaTriCases[] = {
    {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, /*   0 */
    { 0,  8,  3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, /*   1 */
    { 0,  1,  9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, /*   2 */
    { 1,  8,  3,  9,  8,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, /*   3 */
    //...
}
```

Marching Cubes

```
/* Determine the marching cubes index */
for ( i=0, index = 0; i < 8; i++)
    if (val1[nodes[i]] >= thresh) /* If the nodal value is above the     */
        index |= CASE_MASK[i]; /* threshold, set the appropriate bit. */

triCase = HexaTriCases[index]; /* triCase indexes into the MC table. */
edge = triCase->HexaEdges;   /* edge points to the list of intersected edges */

for ( ; edge[0] > -1; edge += 3 )  /* stop if we hit the -1 flag */
{
     for (i=0; i<3; i++)     /* Calculate and store the three edge intersections */
     {
            vert = HexaEdges[edge[i]];
            n0 = nodes[vert[0]];
            n1 = nodes[vert[1]];
            t = (thresh - val1[n0]) / (val1[n1] - val1[n0]);
            tri_ptr[i] = add_intersection( n0, n1, t );    /* Save an index to the pt. */
     }
     add_triangle( tri_ptr[0], tri_ptr[1], tri_ptr[2], zoneID); /* Store the triangle */
}
```

Thanks !