Scalar Algorithms: Contouring

Computer Animation and Visualisation – Lecture 11

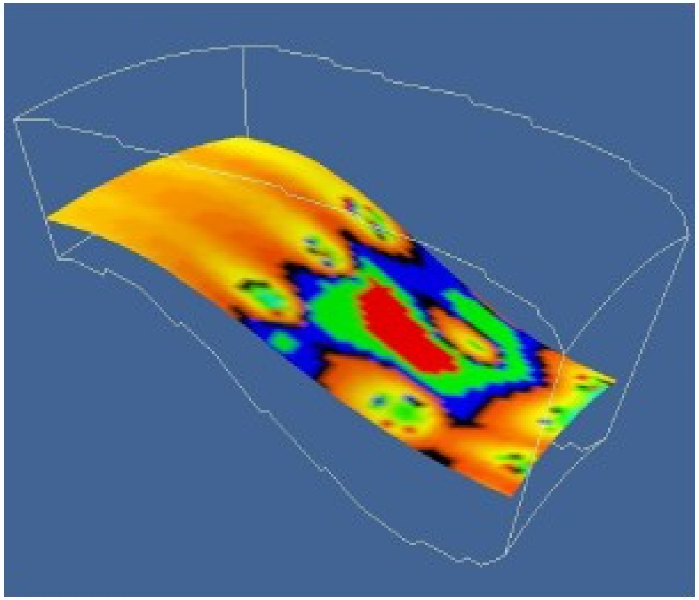
Taku Komura

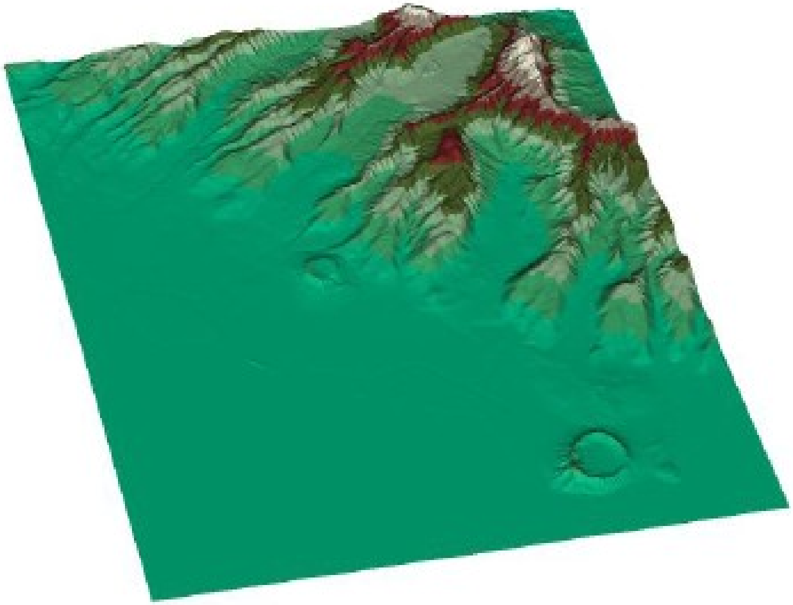
tkomura@inf.ed.ac.uk

Institute for Perception, Action & Behaviour
School of Informatics

Last Lecture ....

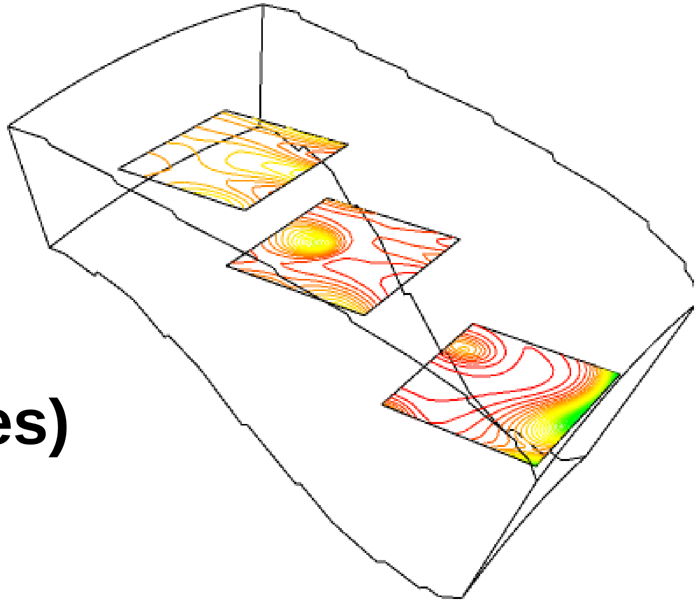
- **Colour mapping**
  - Designing the colour lookup table / transfer function
  - Interpolating in various colour spaces
    - HSV, RGB, greyscale





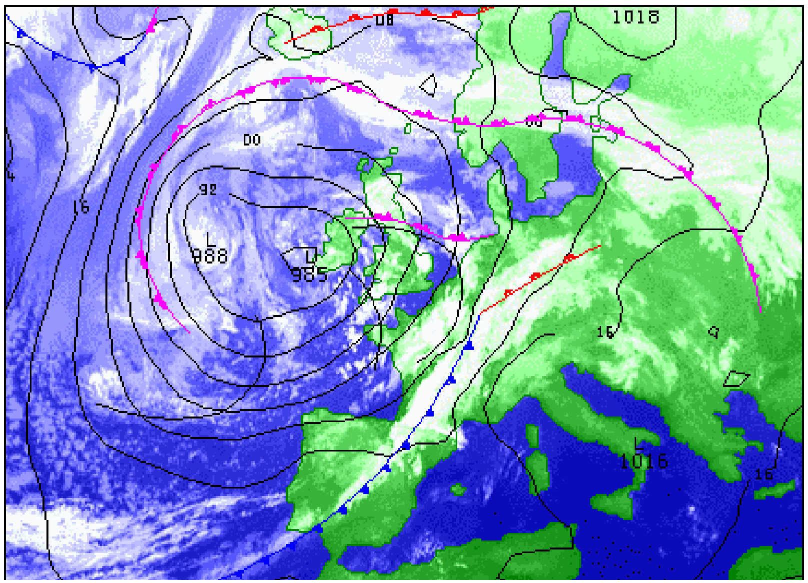
Contouring

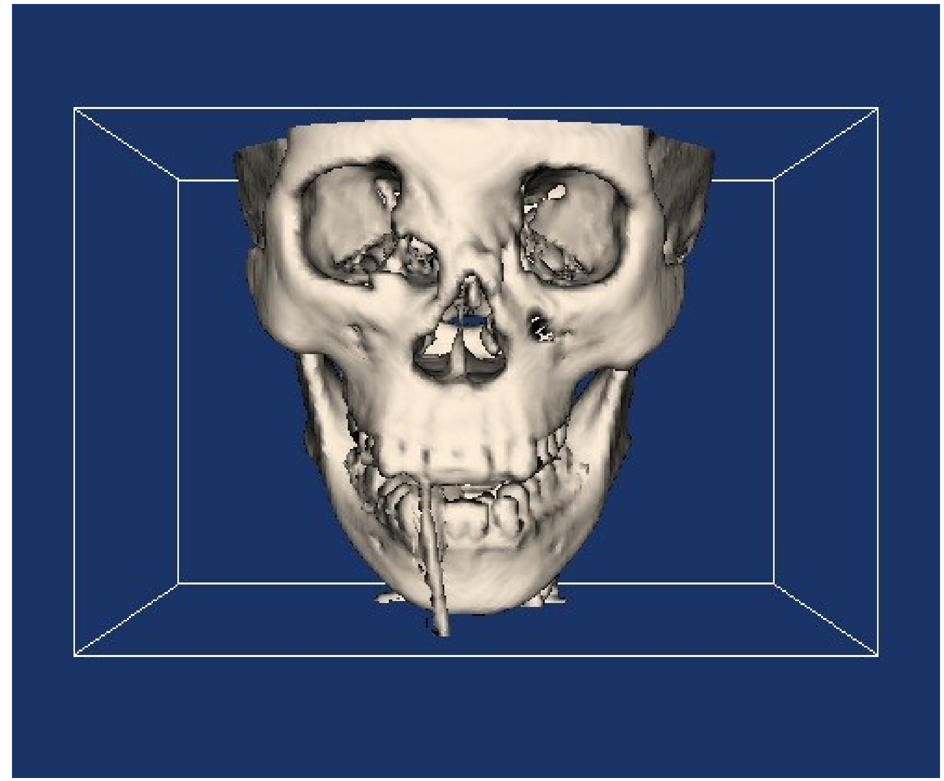
- Contours explicitly *construct* the **boundary between regions with values**



- Boundaries correspond to:
  - **lines in 2D**
  - **surfaces in 3D (known as isosurfaces)**
  - **of constant scalar value**

Example : contours

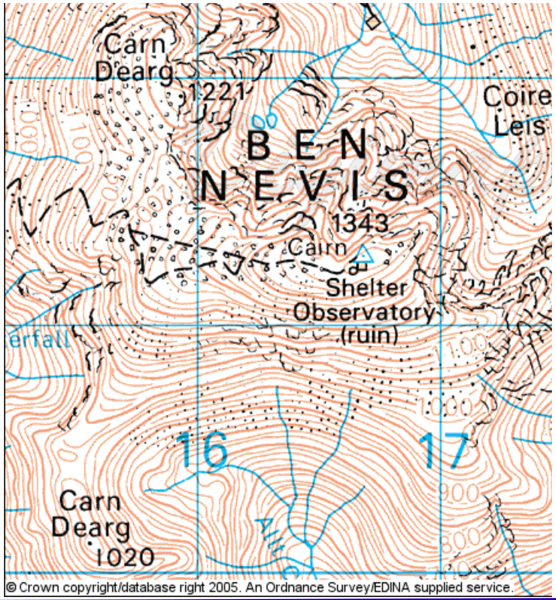




– lines of constant pressure on a weather map (isobars)
– surfaces of constant density in medical scan (isosurface)
  – “iso” roughly means equal / similar / same as

Contours

- Contours **are** boundaries between regions
  - they **DO NOT just** connect points of equal value
  - they **DO also indicate a TRANSITION from a value below the contour to a value above the contour**



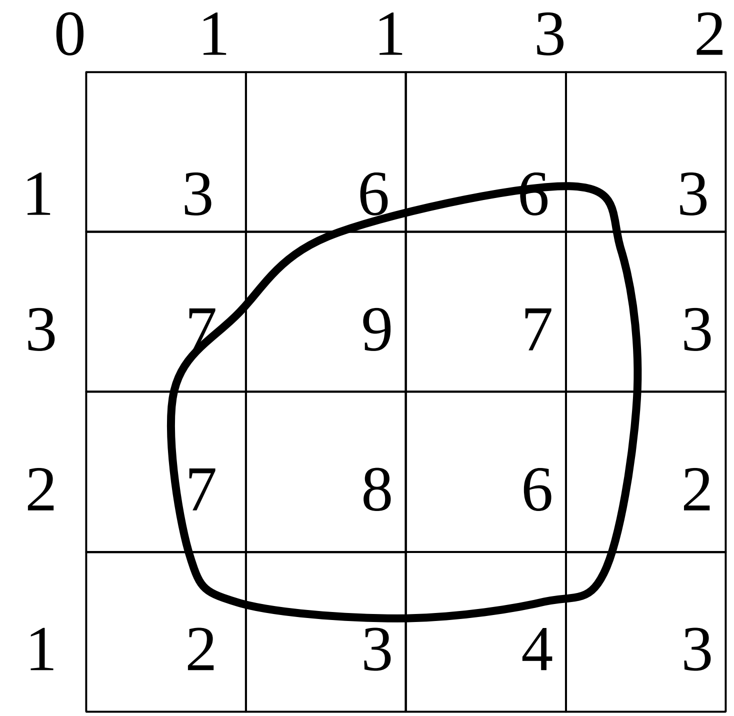
2D contours

- **Data :** 2D structured grid of scalar values

0	1	1	3	2
1	3	6	6	3
3	7	9	7	3
2	7	8	6	2
1	2	3	4	3

- Difficult to visualise transitions in data
  - use **contour** at specific scalar value to highlight **transition**
- What is the contour of 5**?**

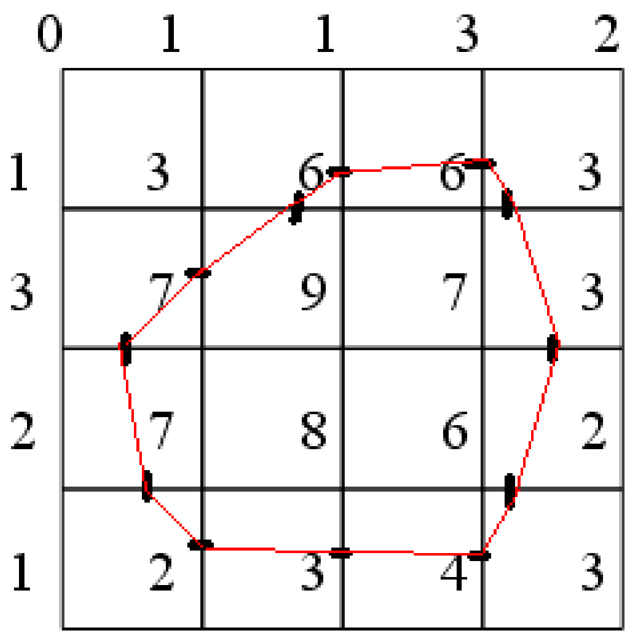
2D contours : line generation



- **Select scalar value**
  - corresponds to contour line
    - i.e. contour value, e.g. 5 (right)
- **Interpolate contour line** through the grid corresponding to this value
  - must interpolate as **scalar values at finite point locations**
  - true **contour transition may lie in-between point values**
  - **simple linear interpolation along grid edges**

Methods of Contour Line Generation

- **Approach 1 : Tracking**
  - find contour intersection with an edge
  - track it through the cell boundaries
    - if it enters a cell then it must exit via one of the boundaries
    - track until it connects back onto itself or exits dataset boundary
  - If there is known to be only one contour, stop
  - *otherwise*
  - Check every edge



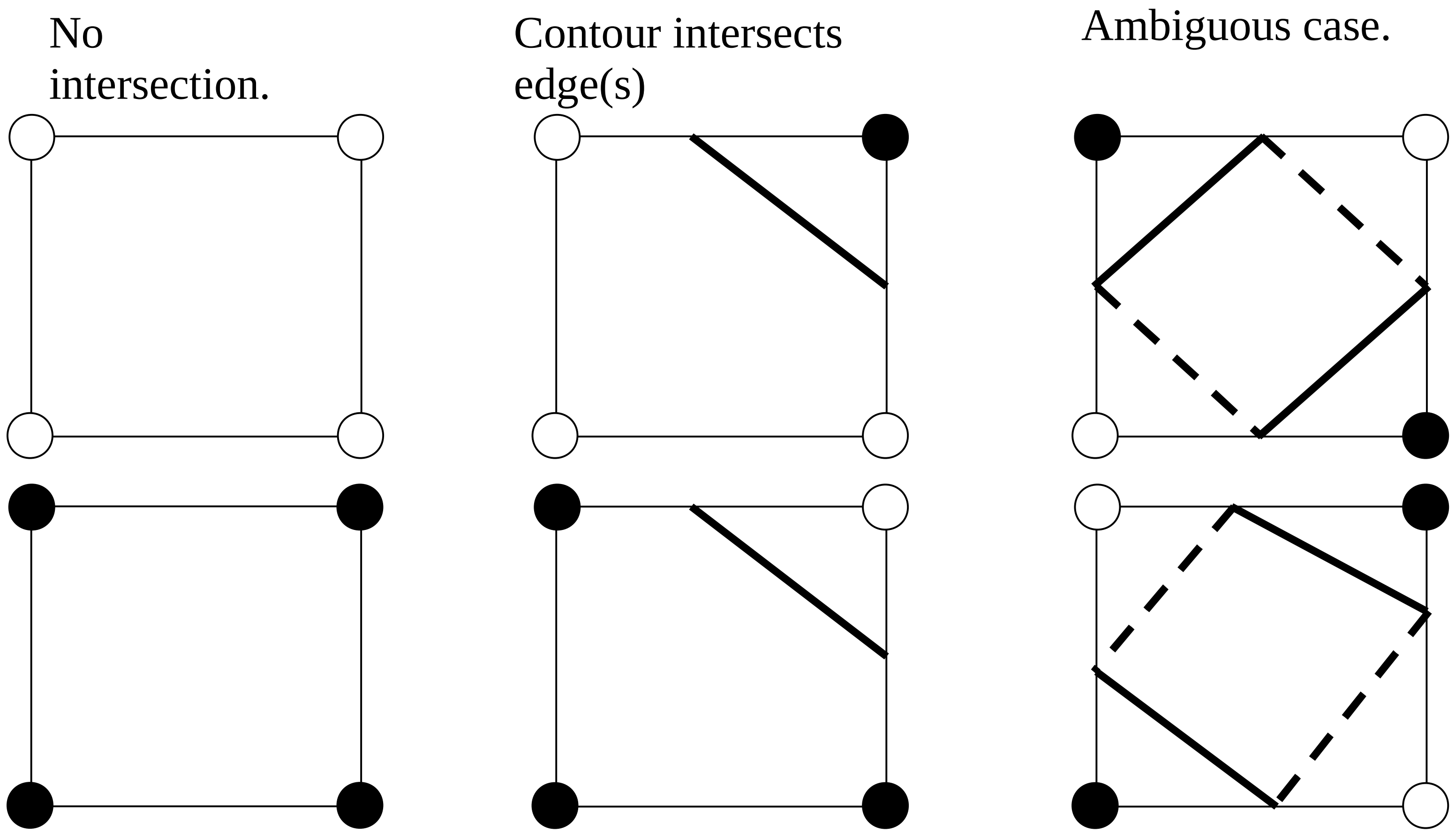
- **Approach 2 : Marching Squares Algorithm**

Marching Squares Algorithm

- Focus : intersection of contour and cell edges
  - how the contour passes through the cell
- Assumption: a contour can pass through a cell in only a finite number of ways
  - cell vertex is inside contour if scalar value > contour

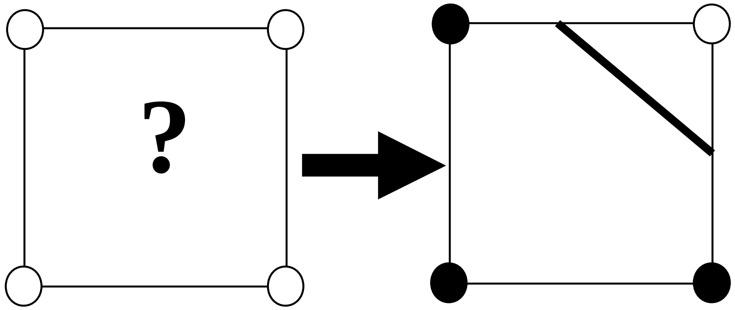
    outside contour if scalar value < contour
  - 4 vertices, 2 states (in or out)

Marching Squares



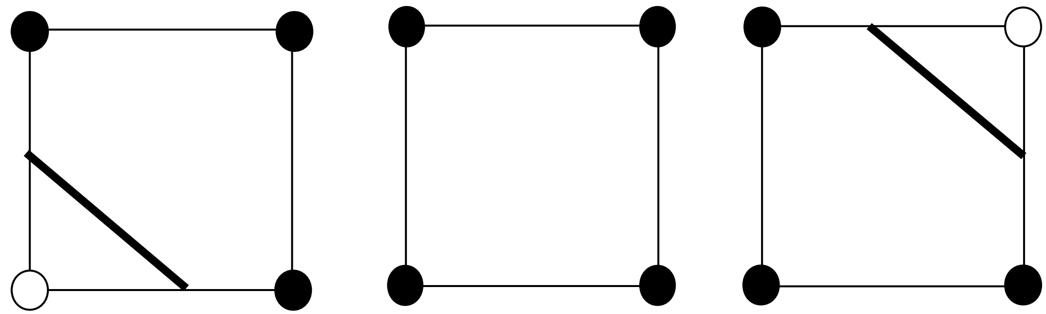
- $2^4 = 16$ possible cases for each square
  - small number so just treat each one separately

MS Algorithm Overview



- **Main algorithm**
  1. Select a cell
  2. Calculate inside/outside state for each vertex
  3. Look up topological state of cell in state table
     - determine which edge must be intersected (i.e. which of the 16 cases)
  1. Calculate contour location for each intersected edge
  2. Move (or **march) onto next cell**
     - **until all cells are visited GOTO 2**
- **Overall : contour intersections for each cell**

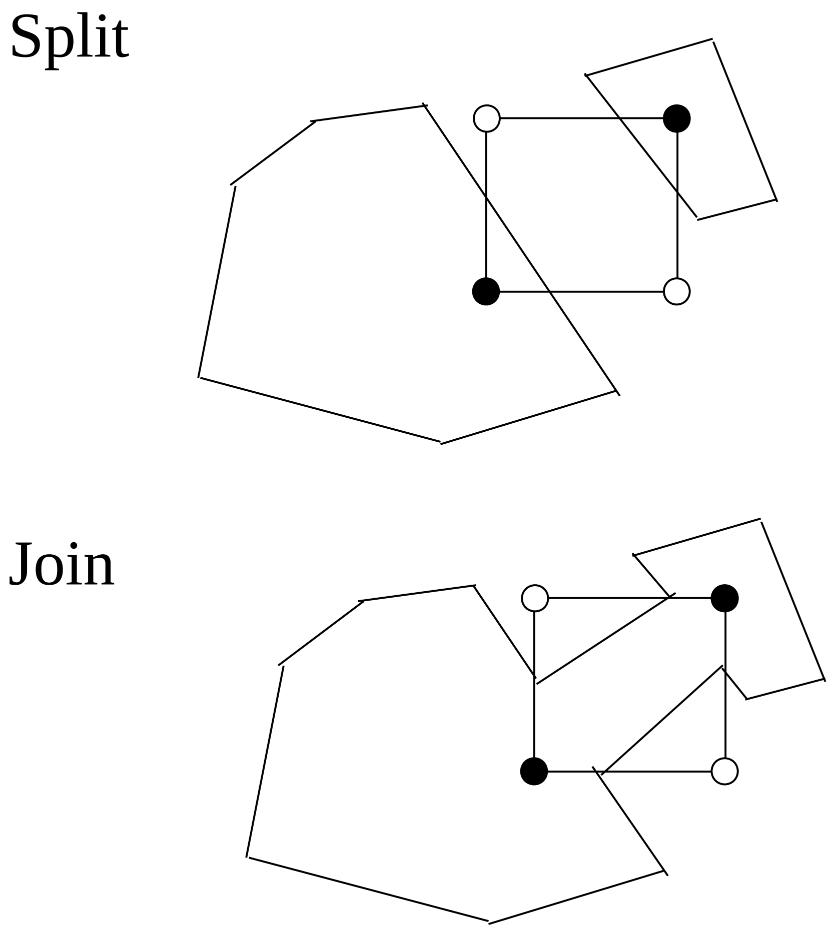
MS Algorithm - notes

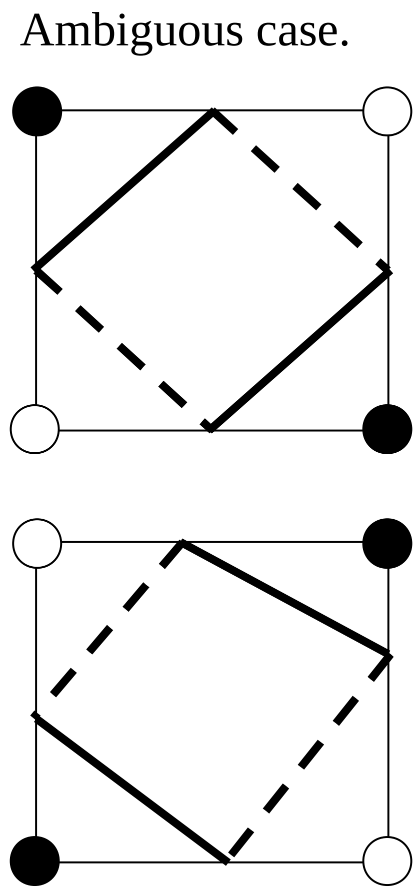


- Intersections for each cell must be merged to form complete contour
  - cells processed independently
  - further **“merging” computation required**
  - disadvantage over tracking (continuous tracked contour)

- easy to implement (also to extend to 3D)
- Easy to parallelise

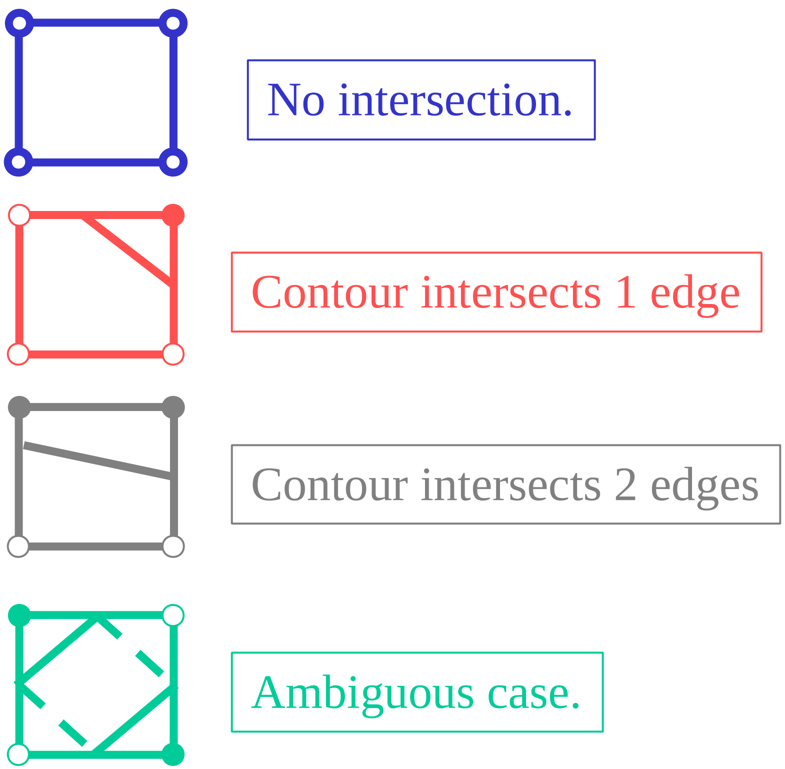
MS : Dealing with ambiguity ?

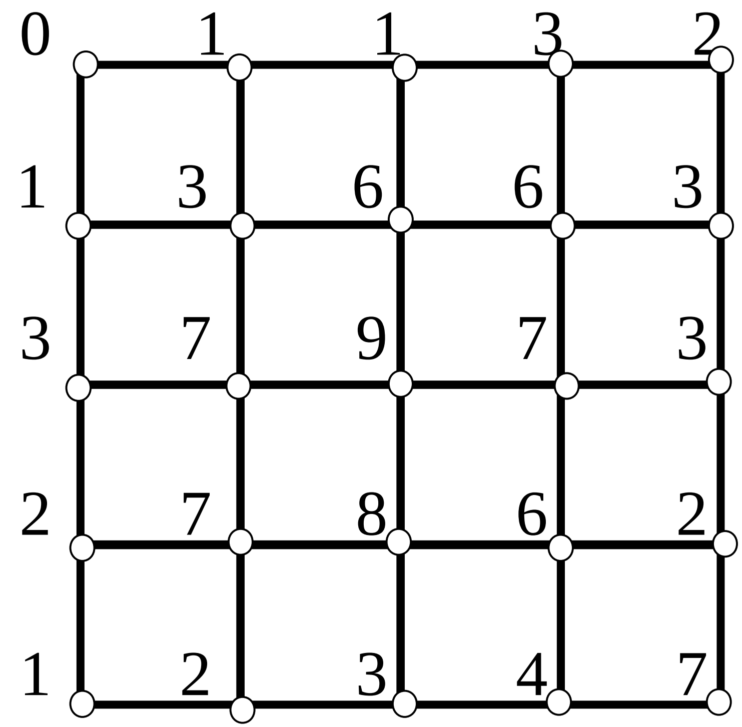




- Choice independent of other choices
  - either valid : both give continuous and closed contour

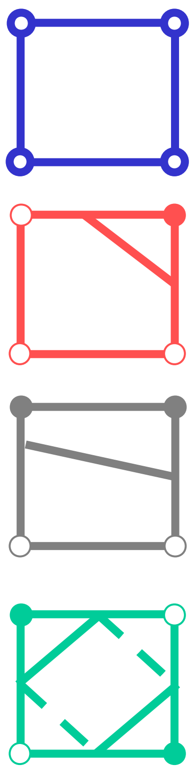
Example : Contour Line Generation





- 3 main steps for each cell
  - here using simplified summary model of cases

Step 1 : classify vertices

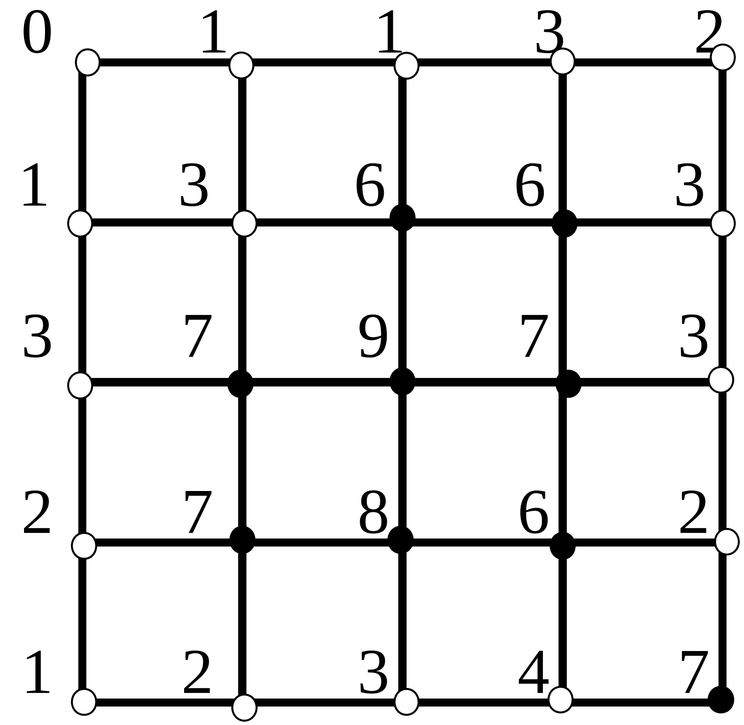


No intersection.

Contour intersects 1 edge

Contour intersects 2 edges

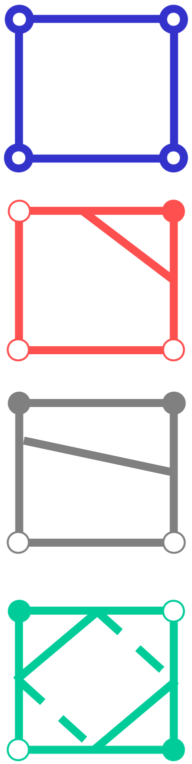
Ambiguous case.



Contour value =5

- Decide whether each vertex is inside or outside contour

Step 2 : identify cases

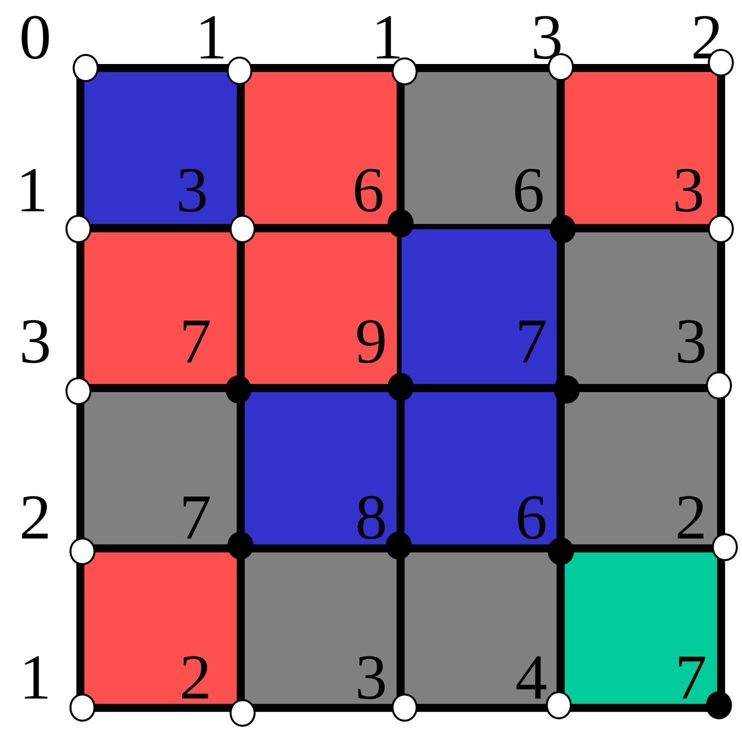


No intersection.

Contour intersects 1 edge

Contour intersects 2 edges

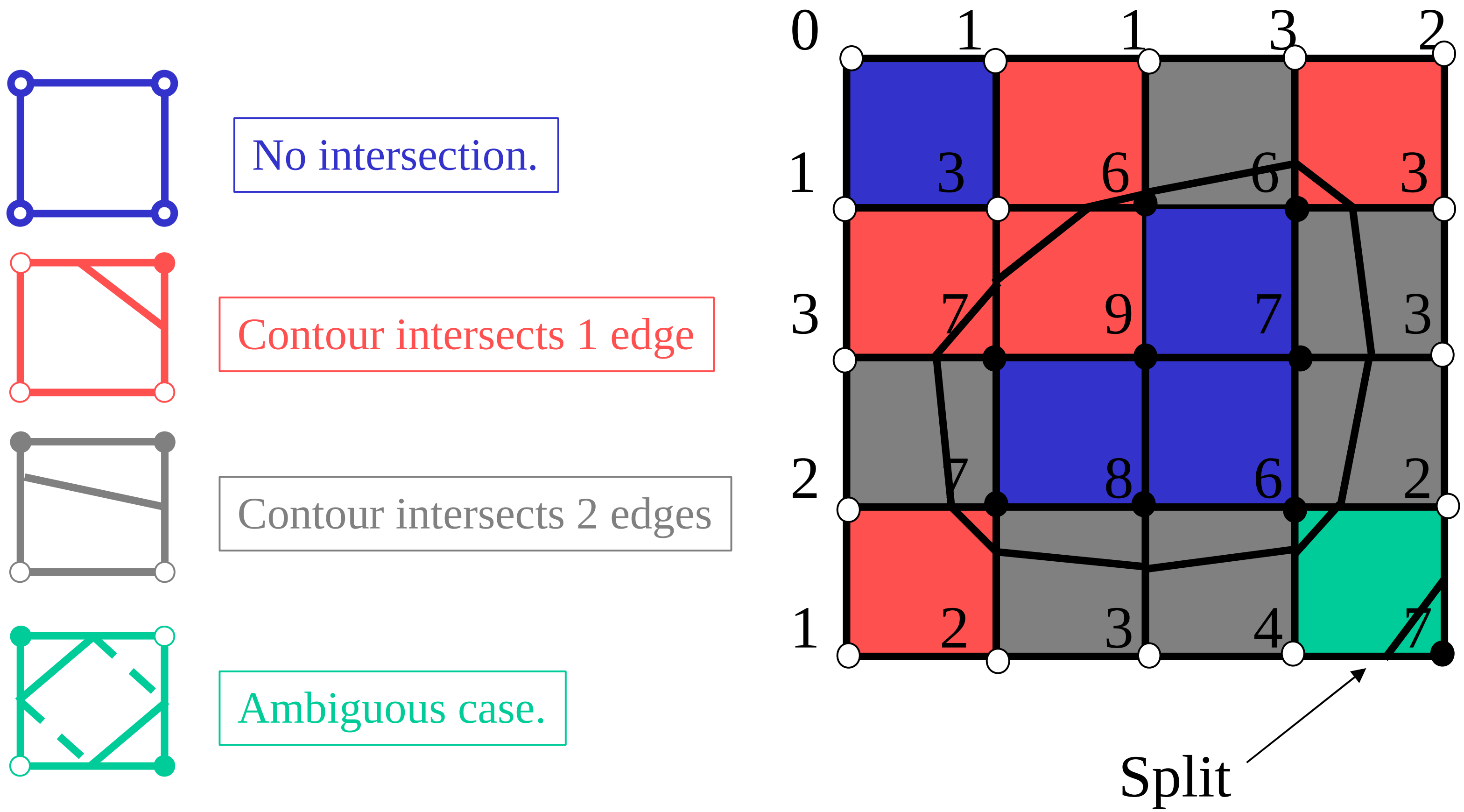
Ambiguous case.



Contour value
=5

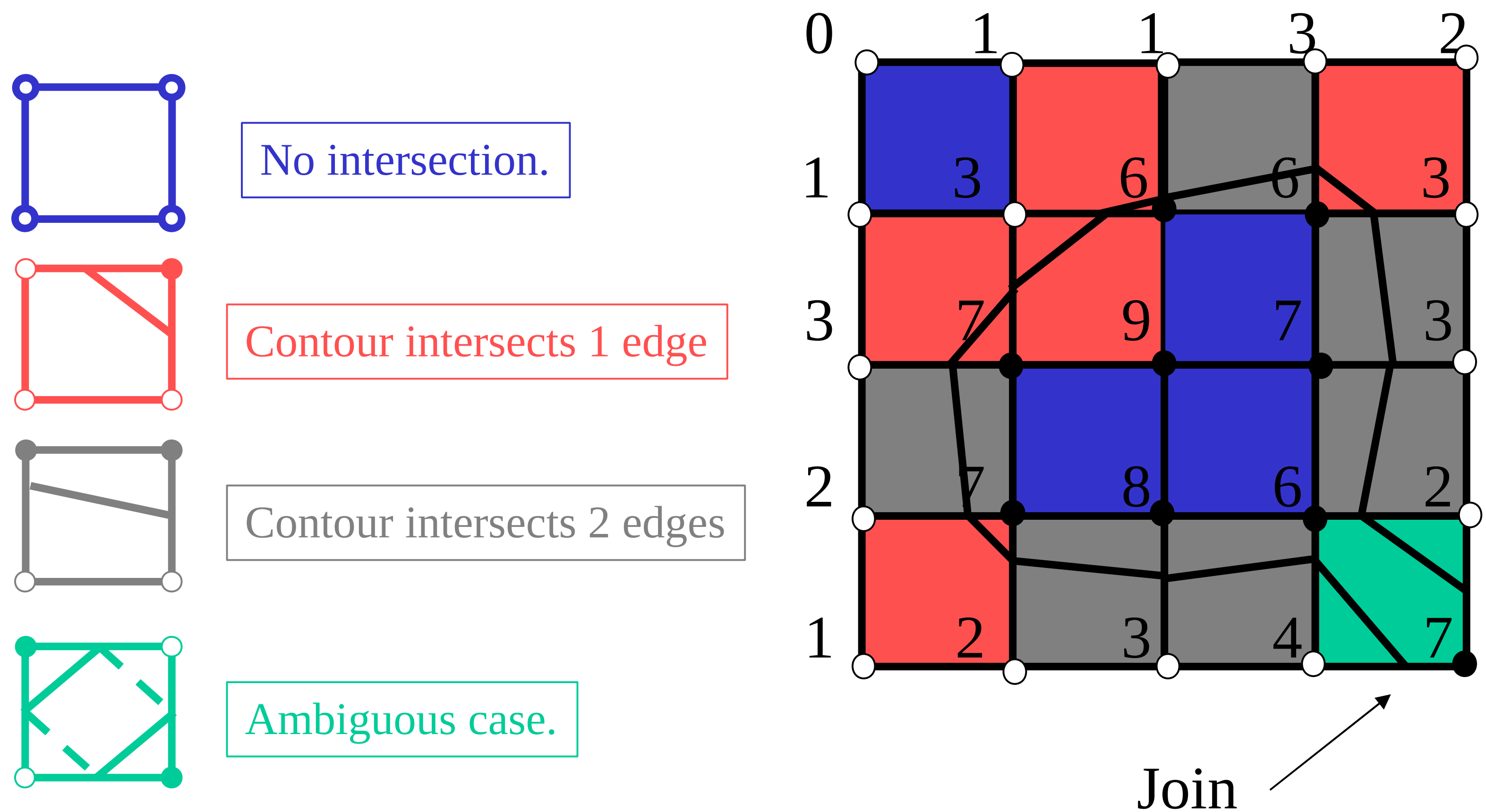
- Classify each cell as one of the cases

Step 3 : interpolate contour intersections



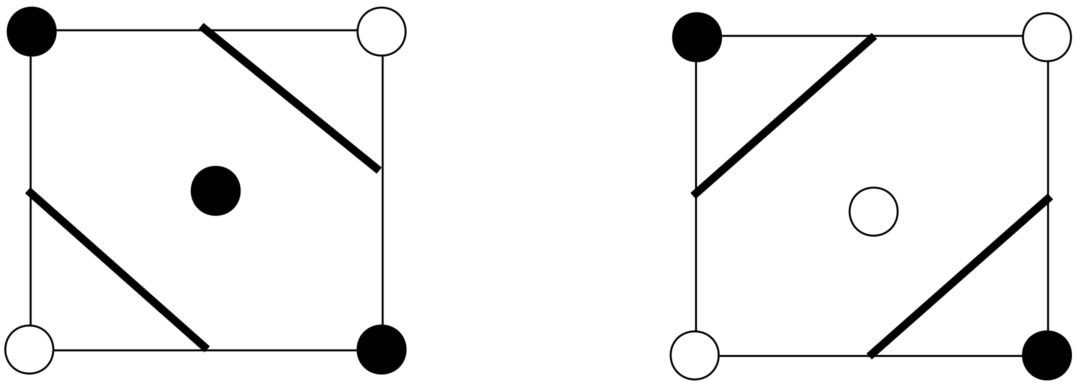
- Determine the edges that are intersected
  - compute contour intersection with each of these edges

Ambiguous contour



- Finally : resolve any ambiguity
  - here choosing “join” (example only)

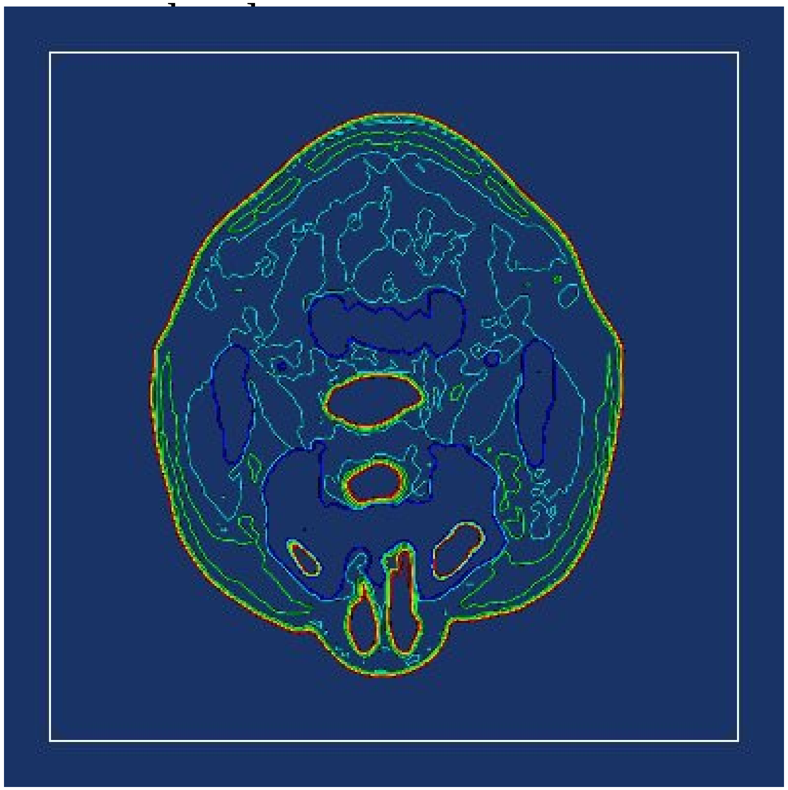
MS : Dealing with ambiguity ? One solution



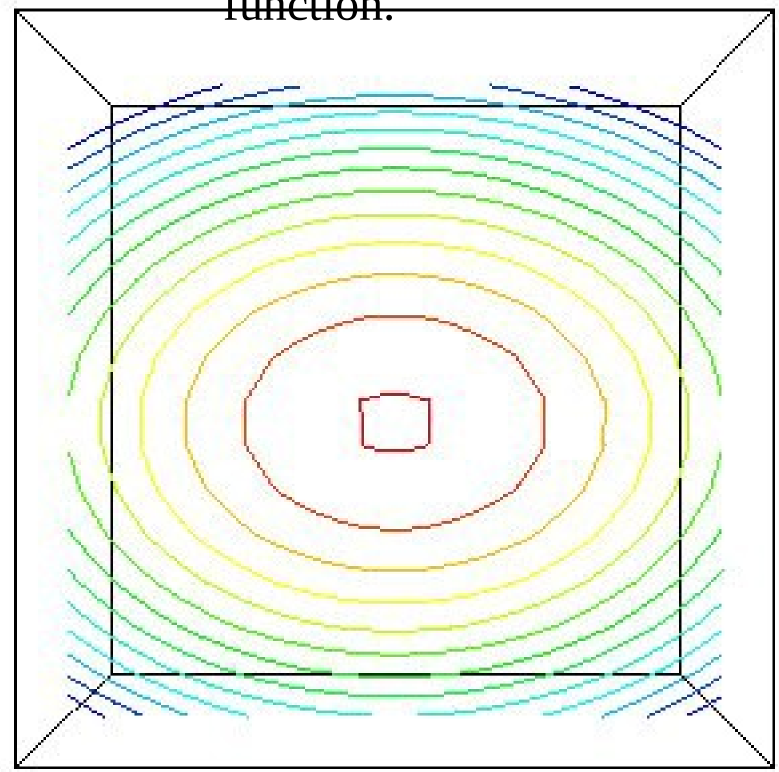
- Calculate the value at the middle of the square by interpolation
- Check if it is under or above the threshold value
- Choose the pattern that matches

2D : Example contour

A slice through the



A Quadric function.



(with colour mapping added)

3D surfaces : marching cubes

- Extension of Marching Squares to **3D**
  - **data** : 3D regular grid of scalar values
  - **result :** 3D surface boundary instead of 2D line boundary
  - 3D cube has 8 vertices → $2^8 = 256$ cases to consider
    - use symmetry and complementary set to reduce to 15
- **Problem : ambiguous cases**
  - cannot simply choose arbitrarily as choice is determined by neighbours
  - poor choice may leave hole artefact in surface

Marching Cubes - cases

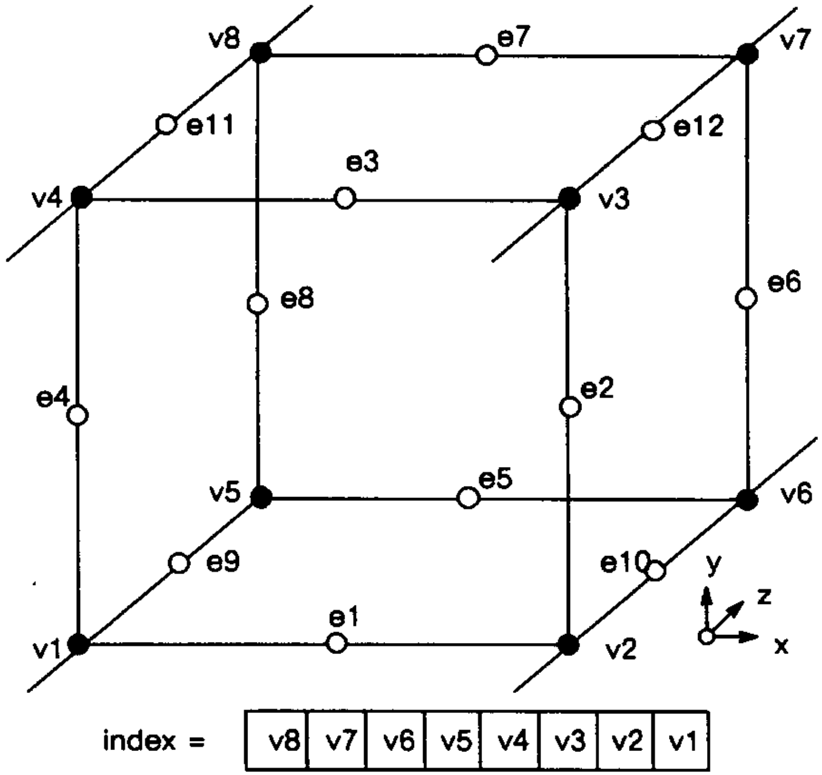


Figure 4. Cube Numbering.

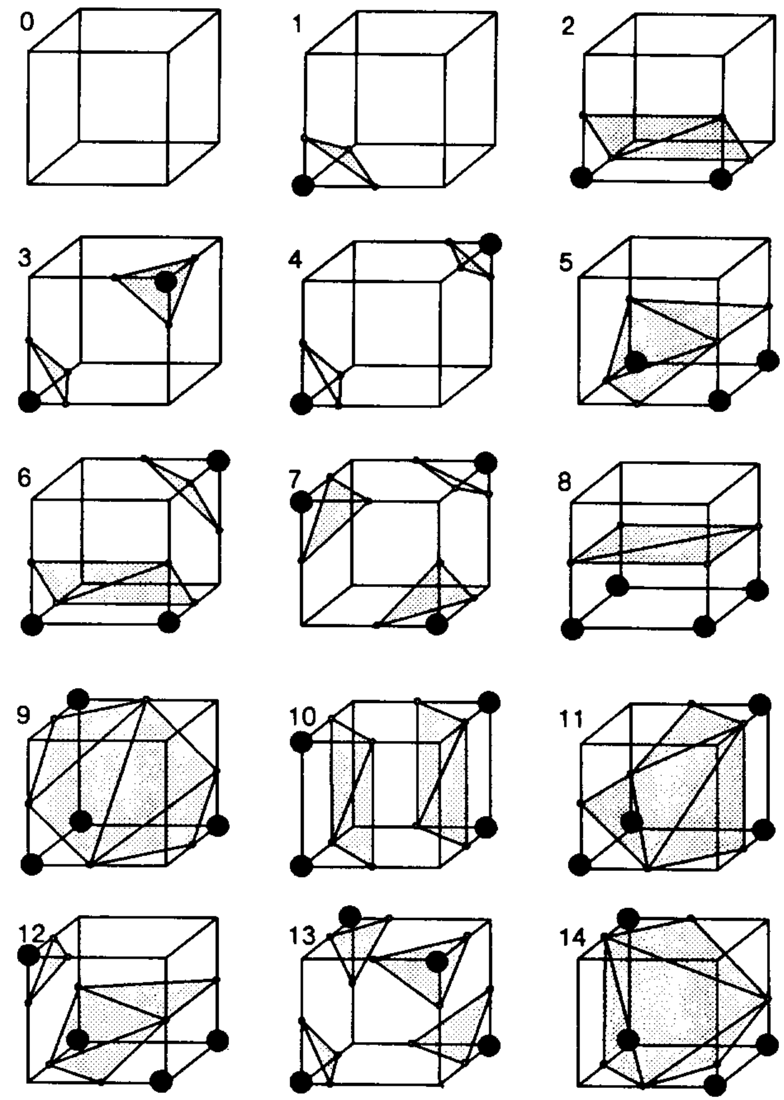
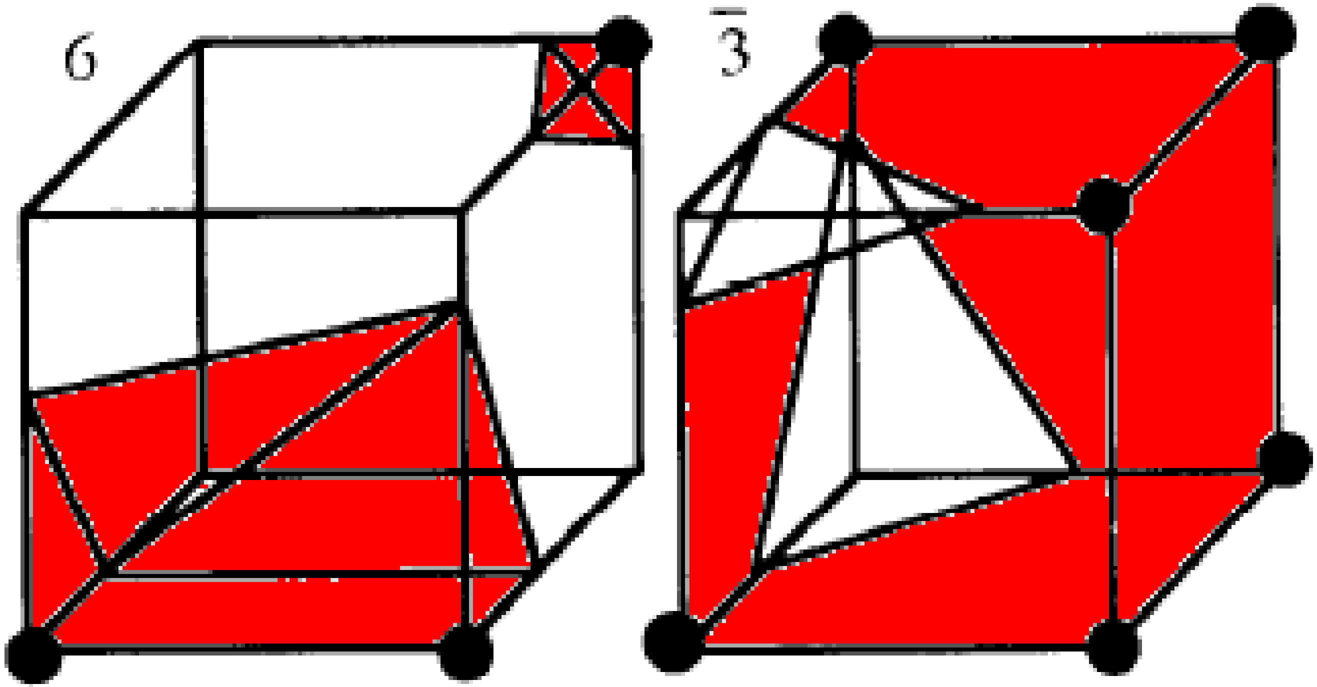


Figure 3. Triangulated Cubes.

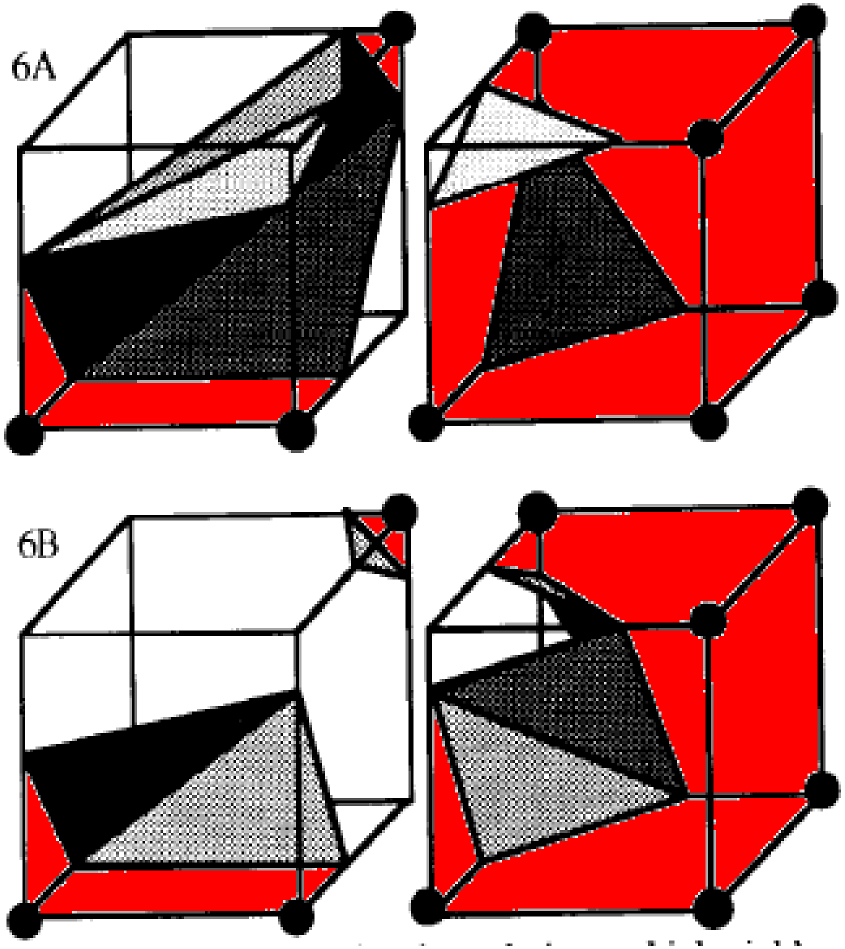
- Ambiguous cases
  - 3,6,10,12,13 – split or join ?

Example of bad choices



- The dark dots are the interior
- There are edges which are not shared by both cubes
- Need to make sure there is no contradiction with the neighbors

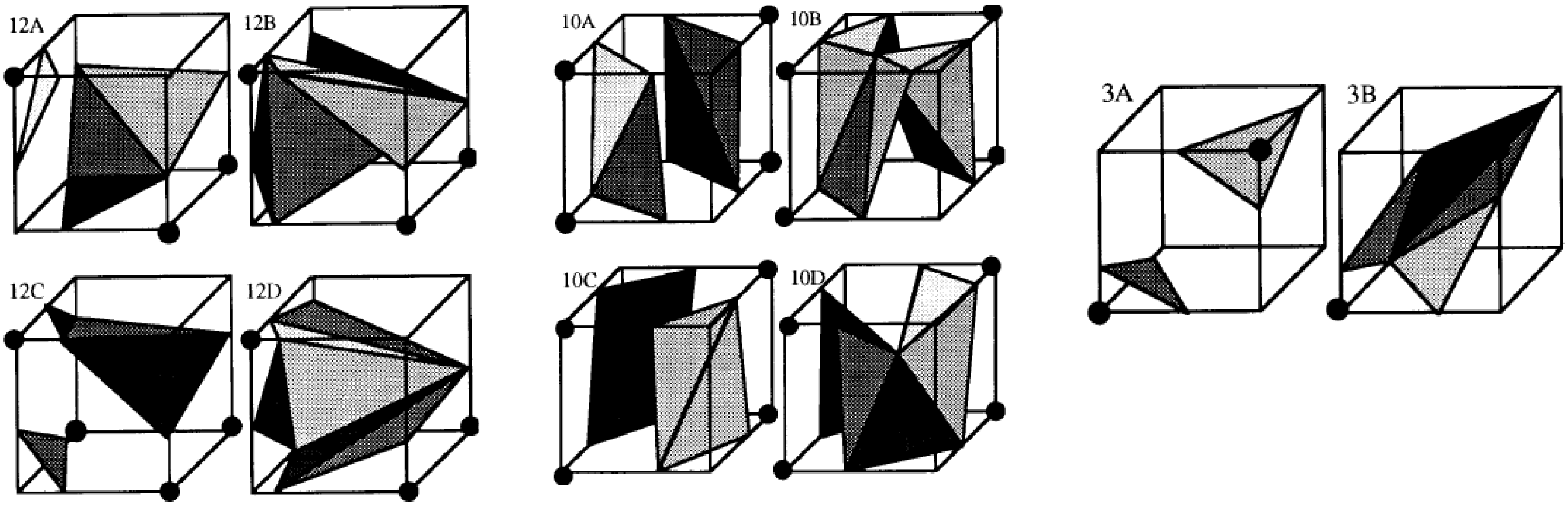
Other two possible triangulations



- Need to decide how the faces are intersected by the contours

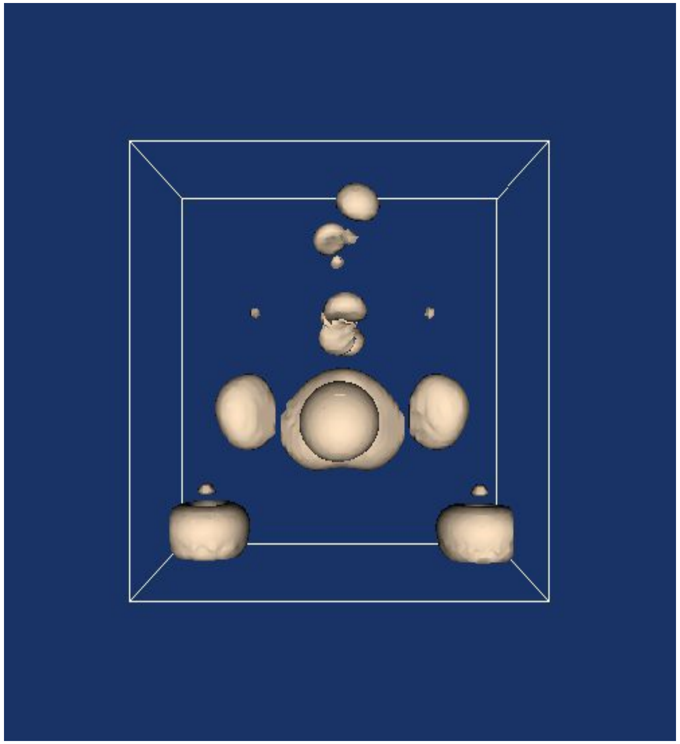
Adding more patterns

- Adding more patterns for 3,6,10,12,13 [Neilson '91]
- Compute the values at the middle of the faces and the cubes
- Selecting the pattern that matches

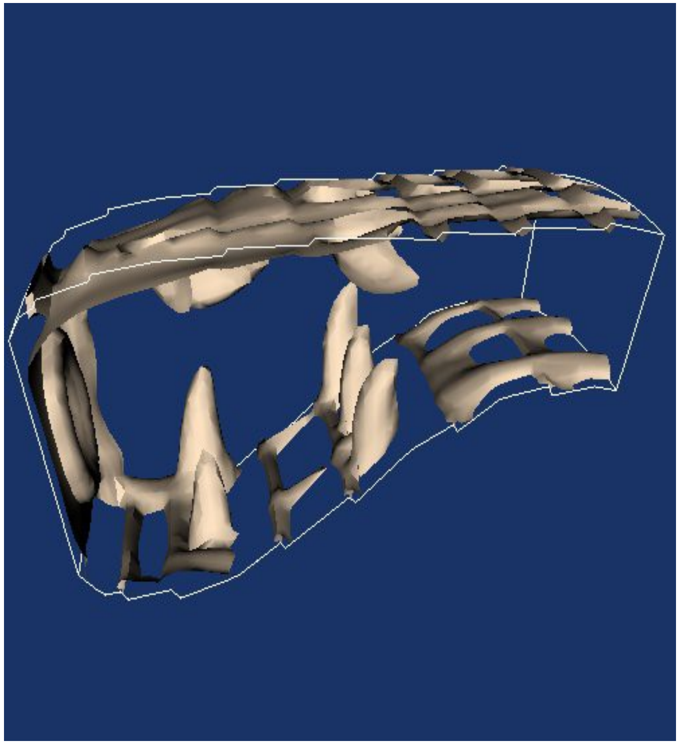


Results : isosurfaces examples

isosurface of Electron potential



isosurface of flow density



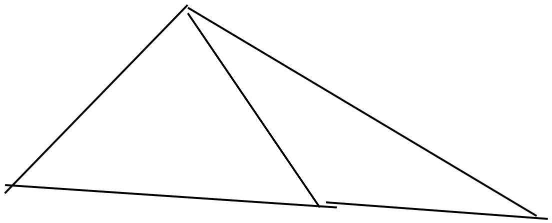
- white outline shows bounds of **3D data grid**
- surface = **3D contour** (i.e. isosurface) **through grid**
- **method :** Marching Cubes

Marching Cubes by CUDA

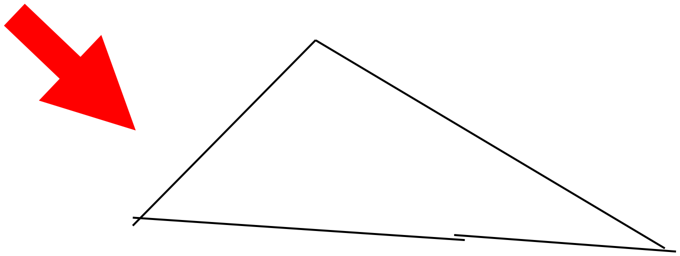
- http://www.youtube.com/watch?v=Y5URxpX8q8U

Problems with Marching Cubes

- Generates **lots of polygons**
  - 1-4 triangles per cell intersected
  - many unnecessary
    - e.g. co-planar triangles
    - lots of work extra for rendering!



  - As with marching squares **separate merging required**
    - need to perform explicit search



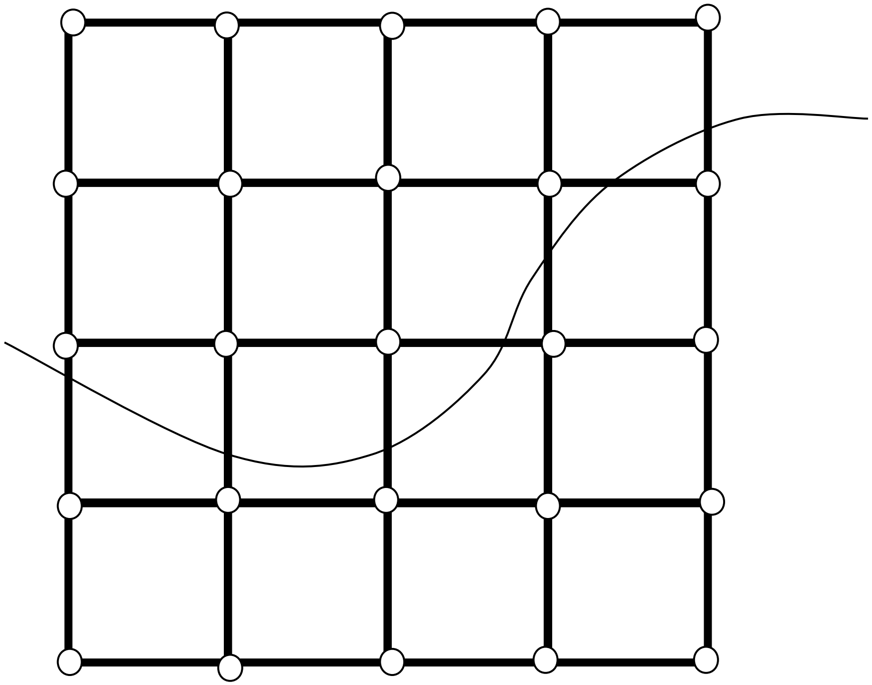
Dividing Cubes Algorithm

- **Marching cubes**
  - often **produces more polygons than pixels** for given rendering scale
  - **Problem :** causes **high rendering overhead**

- **Solution : Dividing Cubes Algorithm**
  - **draw points instead of polygons** ***(faster rendering)***
  - **Need 1: efficient method to find points on surface**

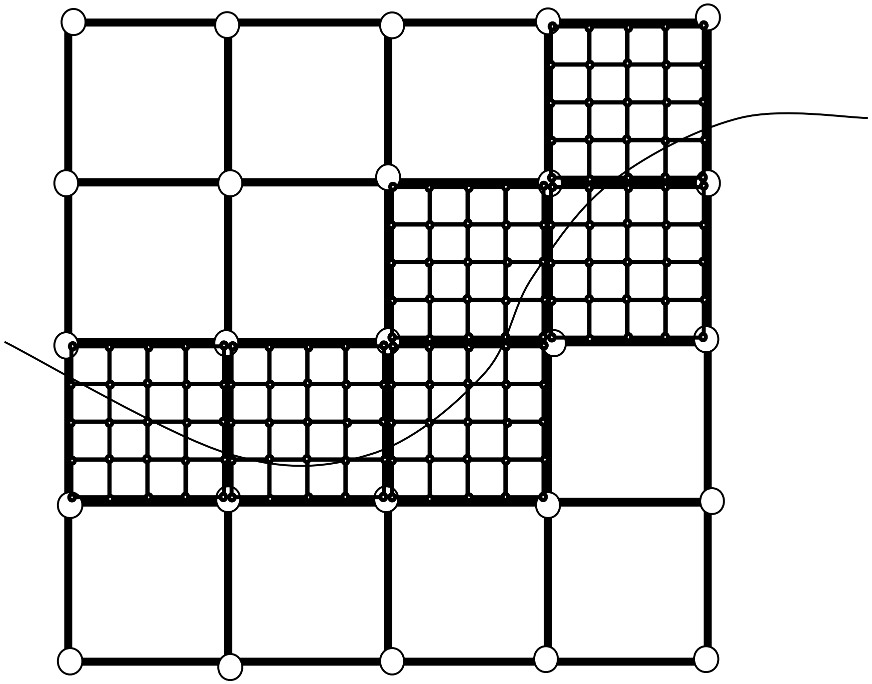
  2: method to shade points

Example : 2D divided squares for 2D lines



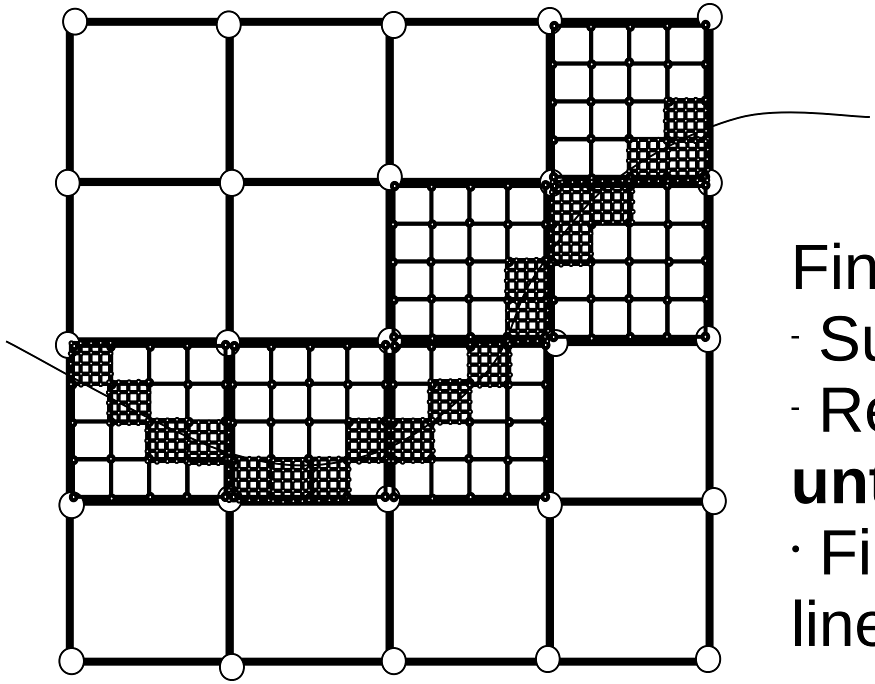
Find pixels that intersect contour
- Subdivide them

2D "Divided Cubes" for lines



Find pixels that intersect line
- **Subdivide them** (usually in 2x2)
- **Repeat recursively**

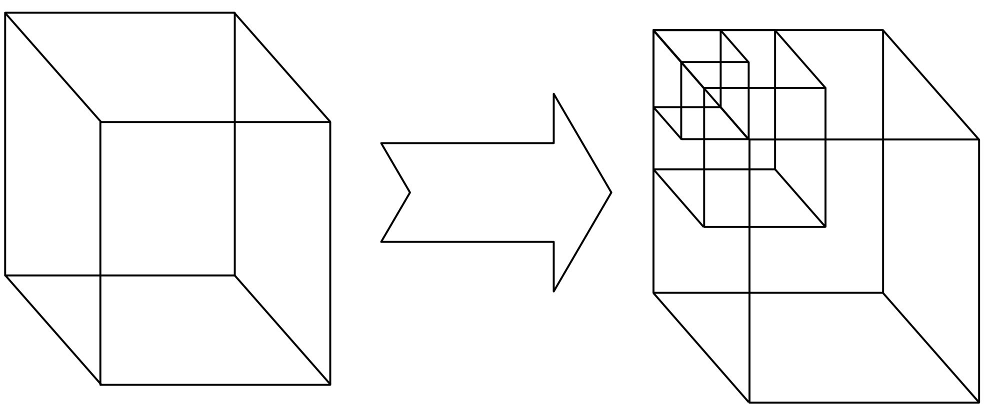
2D “Divided cubes” for lines

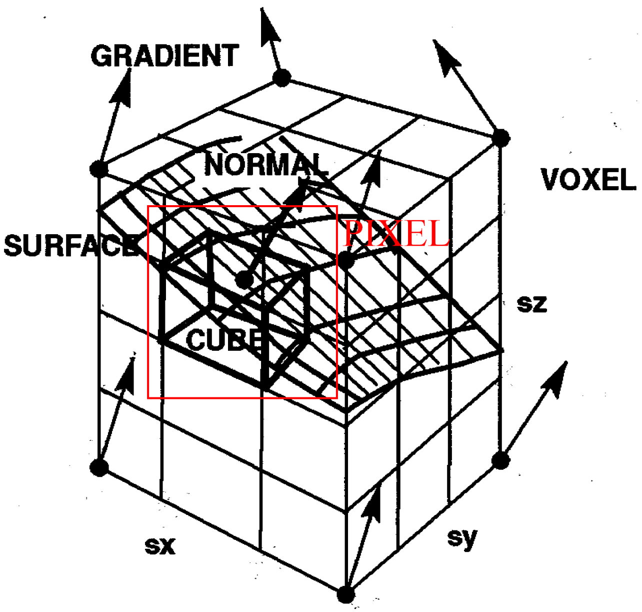


Find pixels that intersect line
- Subdivide them
- Repeat recursively
until screen resolution reached
· Fill in the pixel with the color of the line

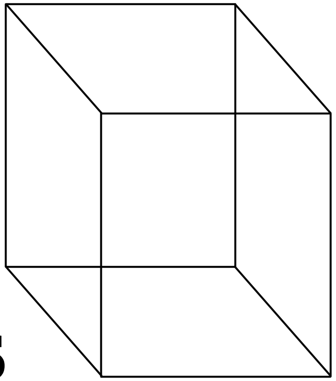
Extension to 3D

- Find **voxels** which intersect **surface**
- Recursively subdivide the voxels that intersect the contour
- Calculate **mid-points of voxels**
- Project **points and draw pixels**





Drawing divided cubes surfaces



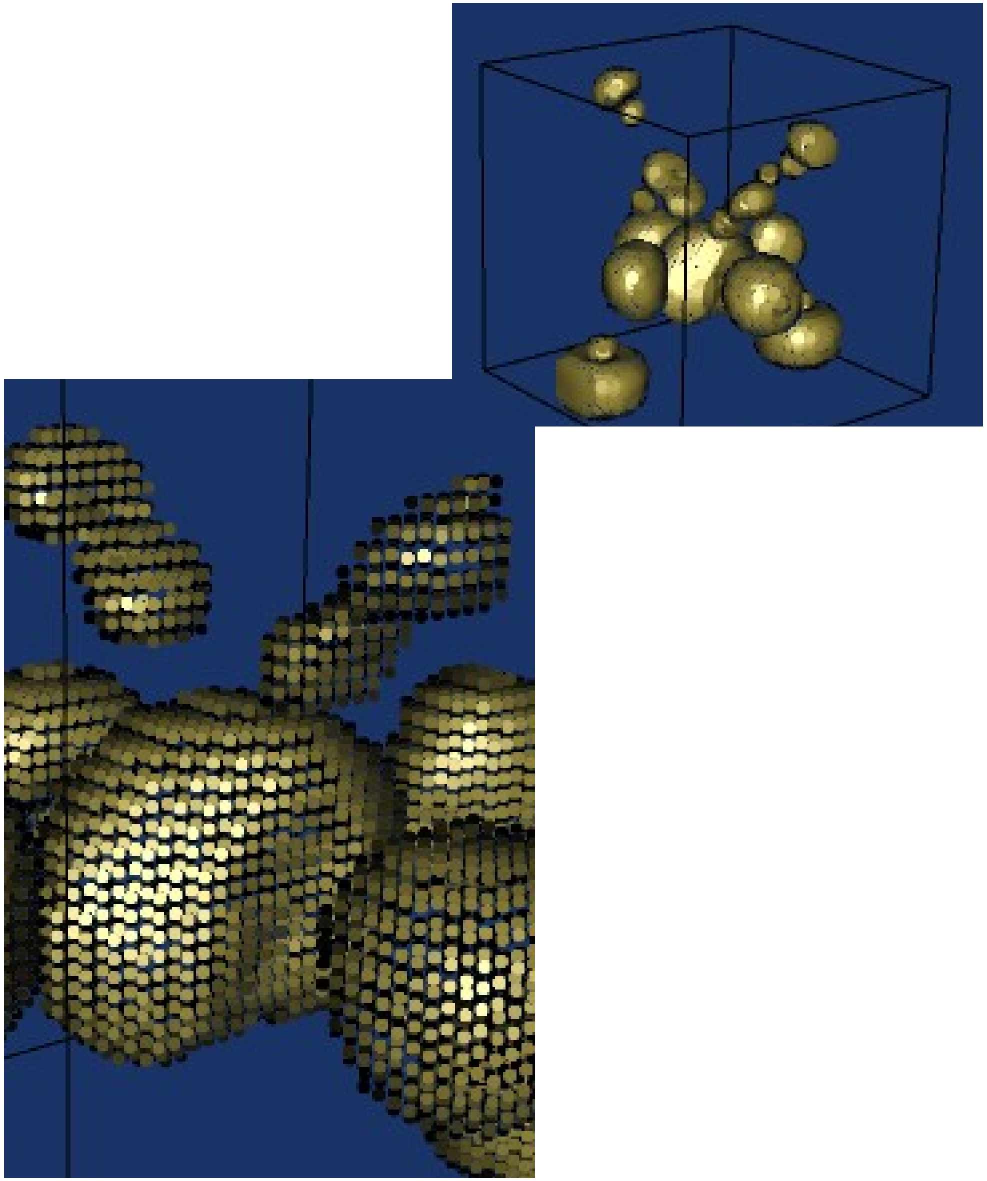
- **surface normal** for lighting calculations
  - interpolate **from voxel corner points**

$$\left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}\right) \approx \left(\frac{F^{x+\Delta x} - F^{x-\Delta x}}{2\Delta x}, \frac{F^{y+\Delta y} - F^{y-\Delta y}}{2\Delta y}, \frac{F^{z+\Delta z} - F^{z-\Delta z}}{2\Delta z}\right)$$

- **problem with camera zoom**
  - ideally dynamically re-calculate points
  - not always computationally possible

Dividing Cubes : Example

- 50,000 points
- when sampling less than screen resolution structure of surface can be seen



Summary

- **Contouring Theory**
  - 2D : **Marching Squares Algorithm**
  - 3D : **Marching Cubes Algorithm** [Lorensen '87]
    - marching tetrahedra, ambiguity resolution
    - limited to regular structured grids
  - 3D Rendering : **Dividing Cubes Algorithm** [Cline '88]

Readings

- G.M. Nielson, B Hamann, “The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes”
- W.E. Lorensen, H.E. Cline, “Marching Cubes: A high resolution 3D surface construction algorithm”
- H.E. Cline, W.E. Lorensen and S. Ludke, “Two algorithms for the three-dimensional reconstruction of tomograms”