Big Data Visual Analytics (CS 661)

Instructor: Soumya Dutta

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur (IITK)

email: soumyad@cse.iitk.ac.in

Acknowledgements

- Some of the following slides are adapted from the excellent course materials and tutorials made available by:
  - Prof. Han-Wei Shen (The Ohio State University)
  - Prof. Klaus Mueller (State University of New York at Stony Brook)
  - David DeMarle (Intel)

Announcements

- Please form **groups of 2** among yourselves for assignments
  - Deadline Jan 23rd
- Groups for projects will be formed later in the course
  - Sometime around the midsem when the project will be assigned

Study Materials for Lecture 5

- William E. Lorensen and Harvey E. Cline. 1987, "*Marching cubes: A high resolution 3D surface construction algorithm*". SIGGRAPH Comput. Graph. 21, 4 (July 1987), 163–169. https://doi.org/10.1145/37402.37422.
- Reference: "Resolving the Ambiguity in Marching Cubes" by Nielson and Hamman, IEEE VIS'91.
- The Visualization Toolkit by Will Schroeder, Ken Martin, Bill Lorensen
  - Chapter 6

VTK Examples

- Show2DData.py
- ExtractCell.py
- LoadPolyData.py
- Polyline.py

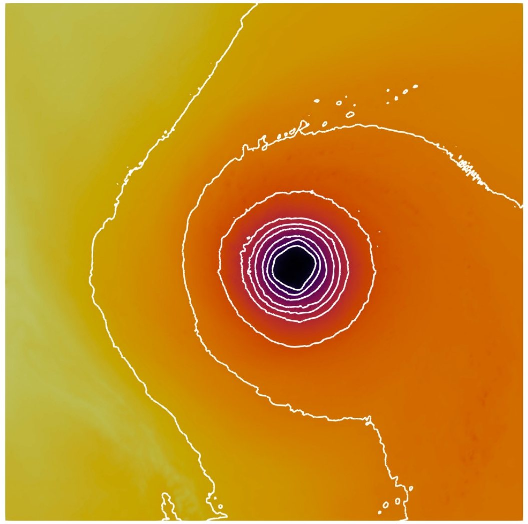
Isocontour Algorithm (2D and 3D)

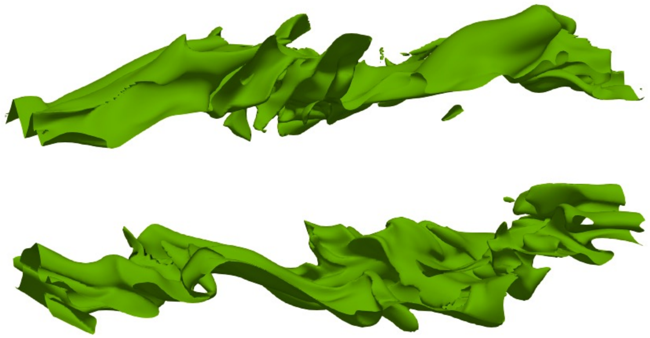
What is an Isocontour?

- An Isocontour is a curve(2D)/surface(3D) in a scalar field where the value of the scalar function is constant across the domain
  – 2D: isoline
  – 3D: Isosurface

– A technique for analyzing and visualizing scalar field data or scalar functions

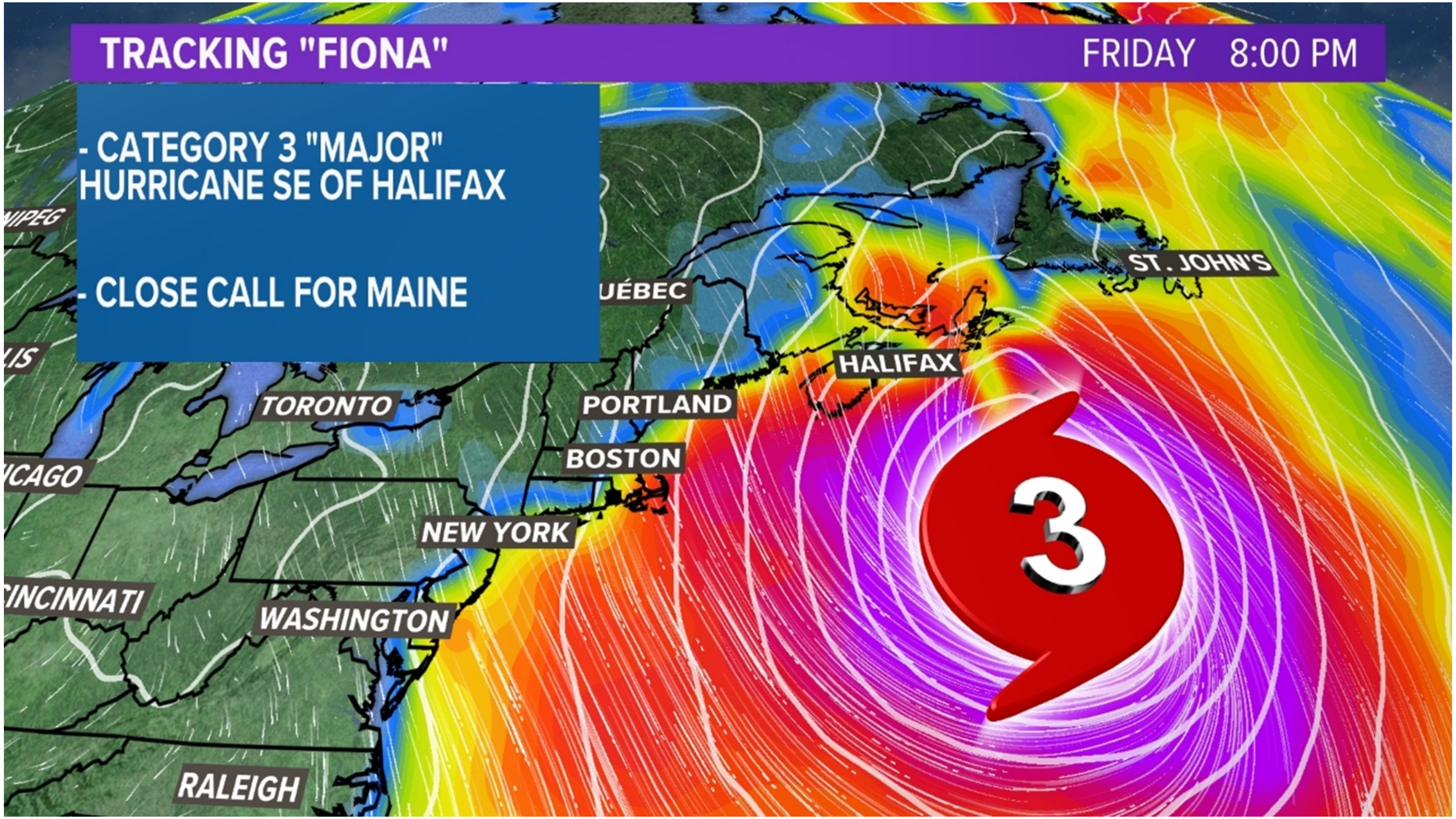
2D isocontour: Isoline





3D isocontour: Isosurface

Isocontour: Isobar – Lines with Equal Pressure

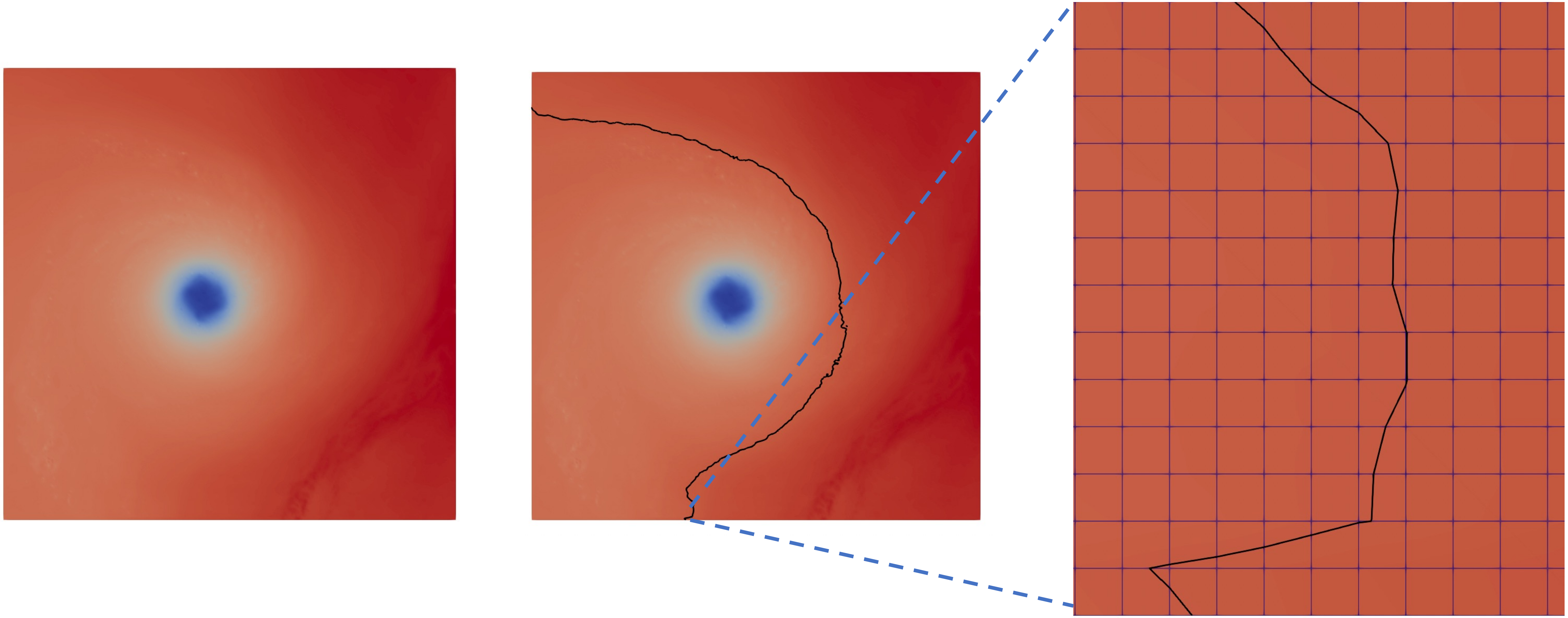


Isocontour

Application in Medical Science:
Isosurface of bone and skin

https://web.cs.ucdavis.edu/~okreylos/PhDStudies/Spring2000/ECS277/FootIso.png

Isocontour

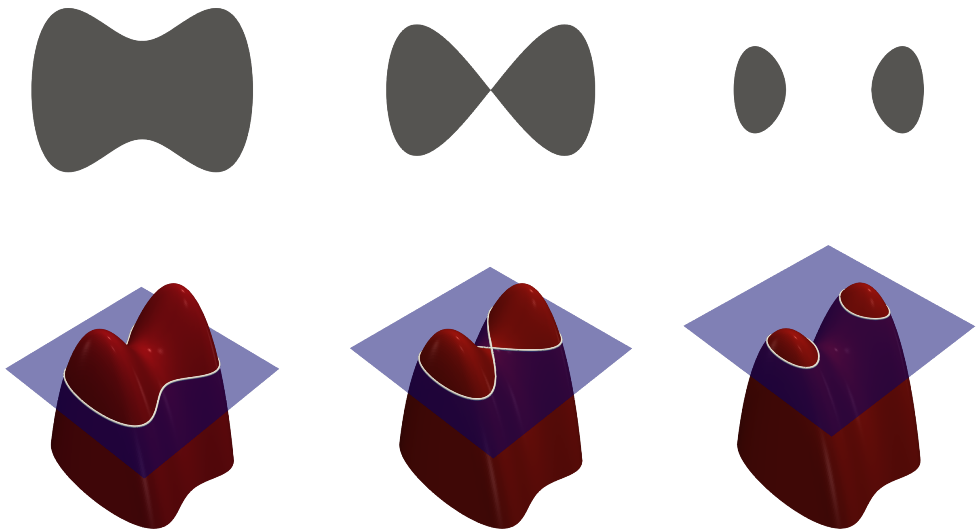


Isocontour (Isobar) at Pressure=250

Isocontour Demo with ParaView

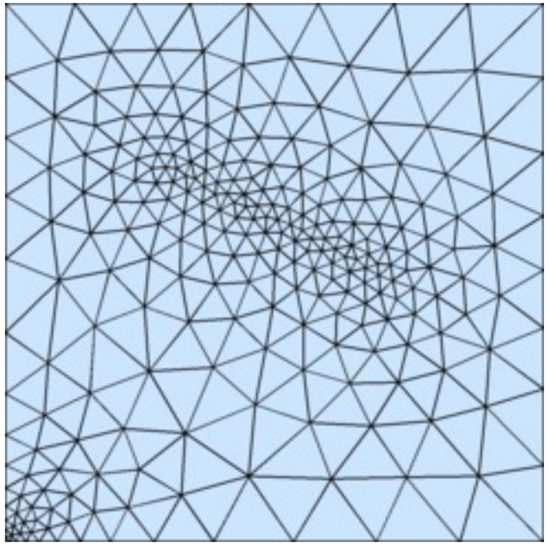
Isocontour also Known As 'Level Set'

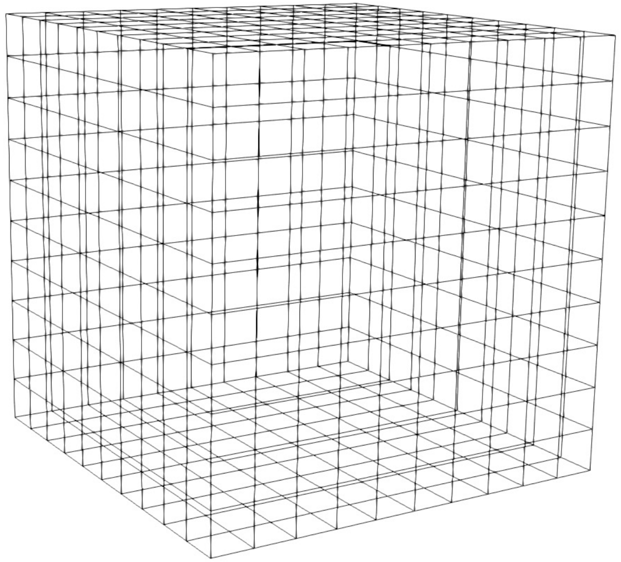
- Suppose that $F : R^n \rightarrow R$ is a function and *C* is in the range of F.
- A **level set** corresponding to an output *C* is a set of all points **x** in the domain of F with the property that F(**x**)=C
- In other words, all the points in the level set are assigned the same value, *C*, by the function F, and any point in the domain of F with output *C* is in that level set



Scalar Data

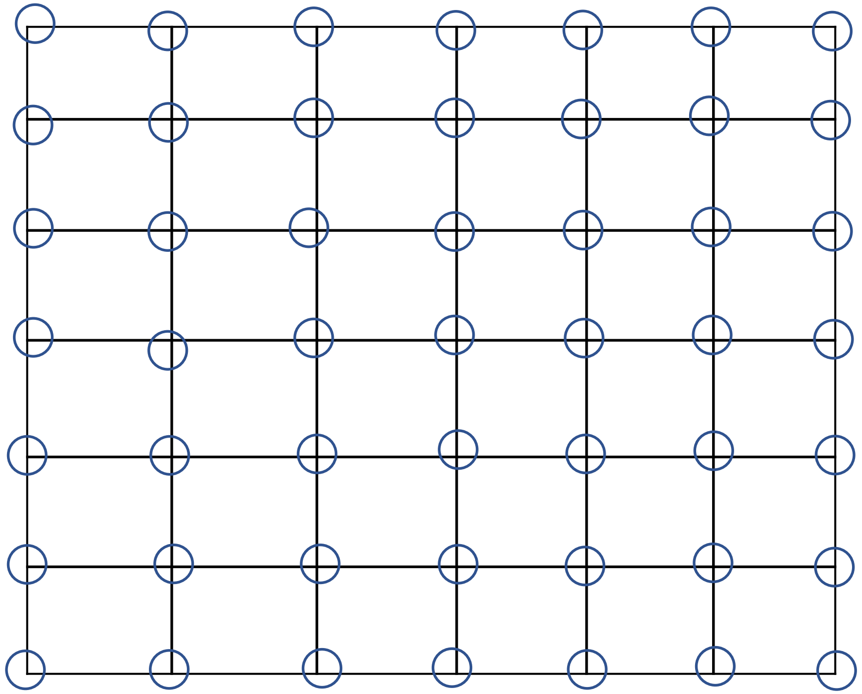
- Data is sampled from a continuous domain
- Discrete sampled domain is represented as a grid/mesh
  - Triangular mesh, cube mesh etc.
- The function value (scalar value) specified at mesh/grid vertices
- Values can be interpolated within the mesh to get value at a query location

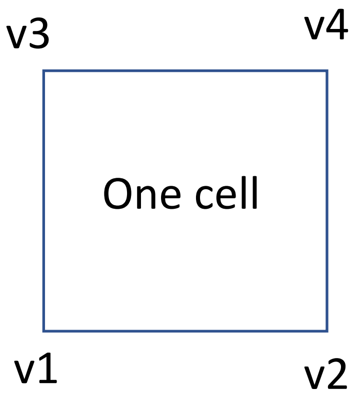




2D Isocontour Extraction

- Given a 2D scalar field, compute isocontour (isoline) for isovalue = C

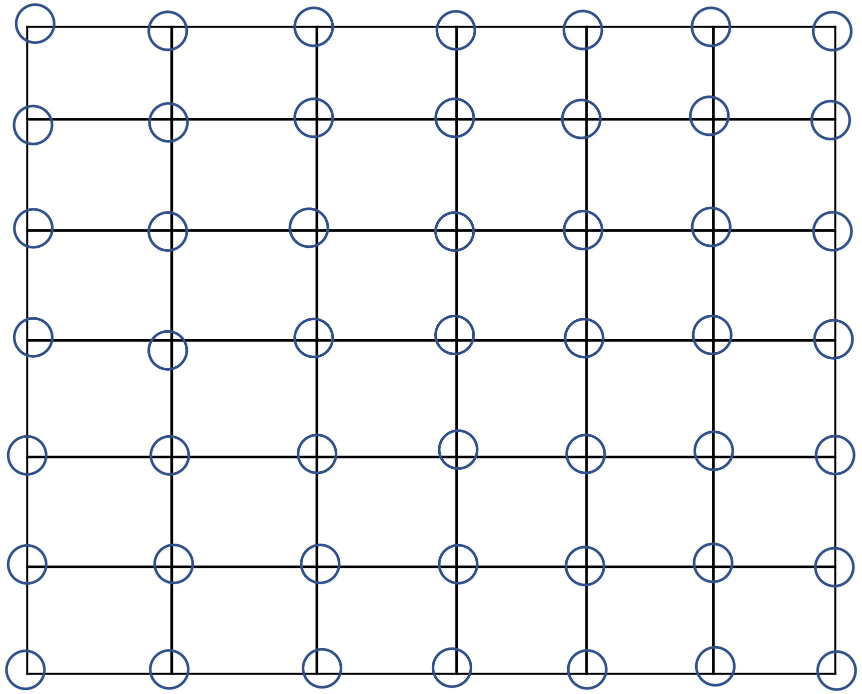


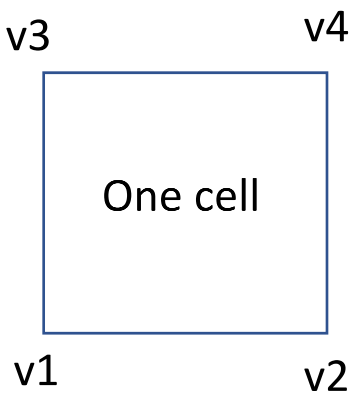


v1,v2,v3,v4 all are > C
No isoline in this cell

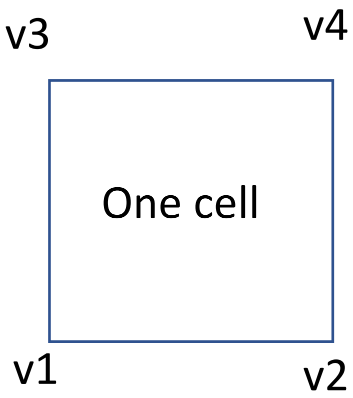
2D Isocontour Extraction

- Given a 2D scalar field, compute isocontour (isoline) for isovalue = C





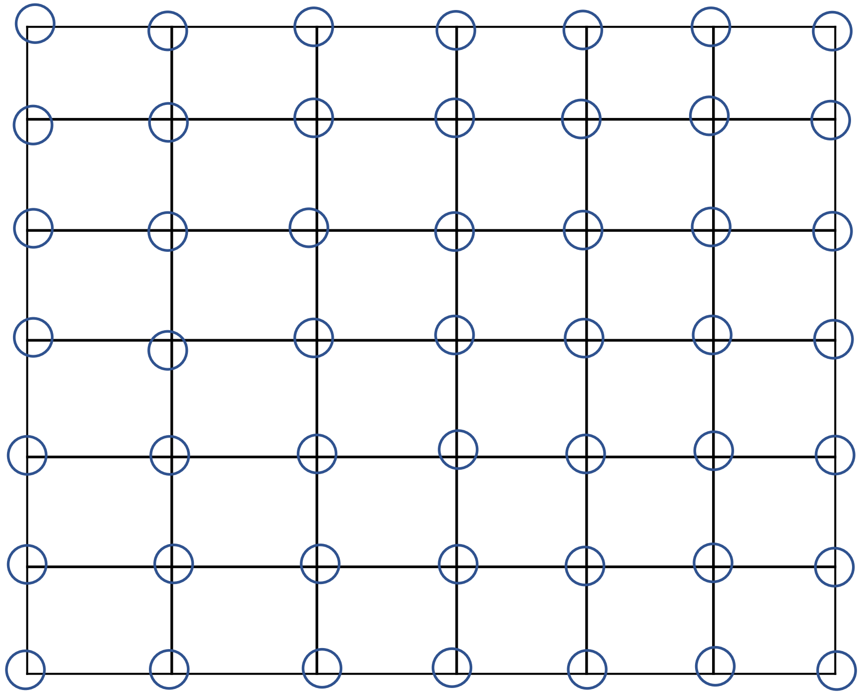
v1,v2,v3,v4 all are > C
No isoline in this cell

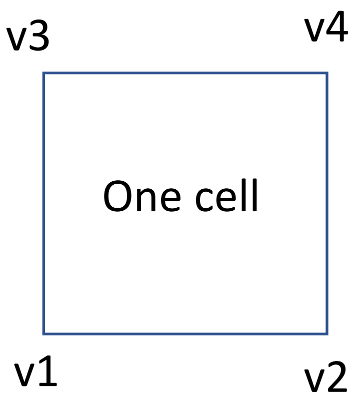


v1,v2,v3,v4 all are < C
No isoline in this cell

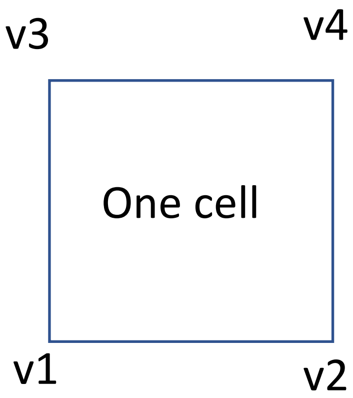
2D Isocontour Extraction

- Given a 2D scalar field, compute isocontour (isoline) for isovalue = C

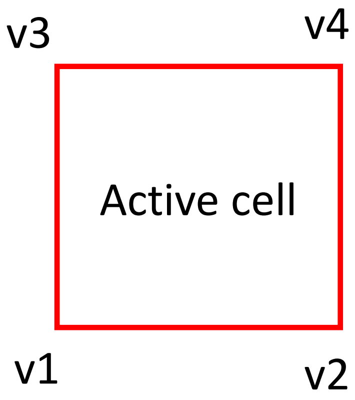




v1,v2,v3,v4 all are > C
No isoline in this cell



v1,v2,v3,v4 all are < C
No isoline in this cell



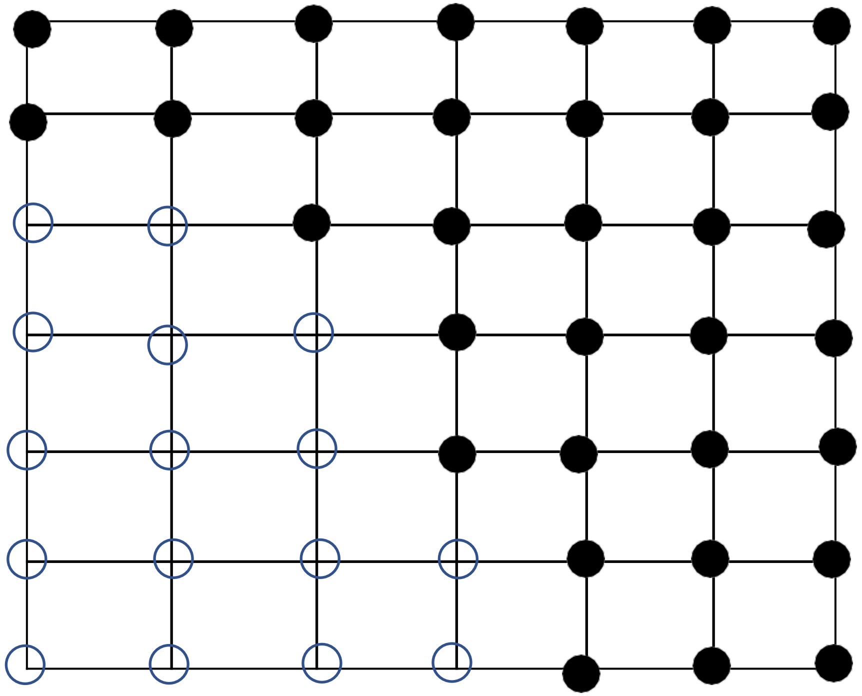
Otherwise, cell contains isocontour segment

2D Isocontour Extraction: Marching Squares

- Given a 2D scalar field, compute isocontour (isoline) for isovalue = C
- This is usually done in a cell-by-cell manner using **<u>Marching Squares algorithm</u>**

2D Isocontour Extraction: Marching Squares

- Given a 2D scalar field, compute isocontour (isoline) for isovalue = C
- This is usually done in a cell-by-cell manner using **Marching Squares algorithm**

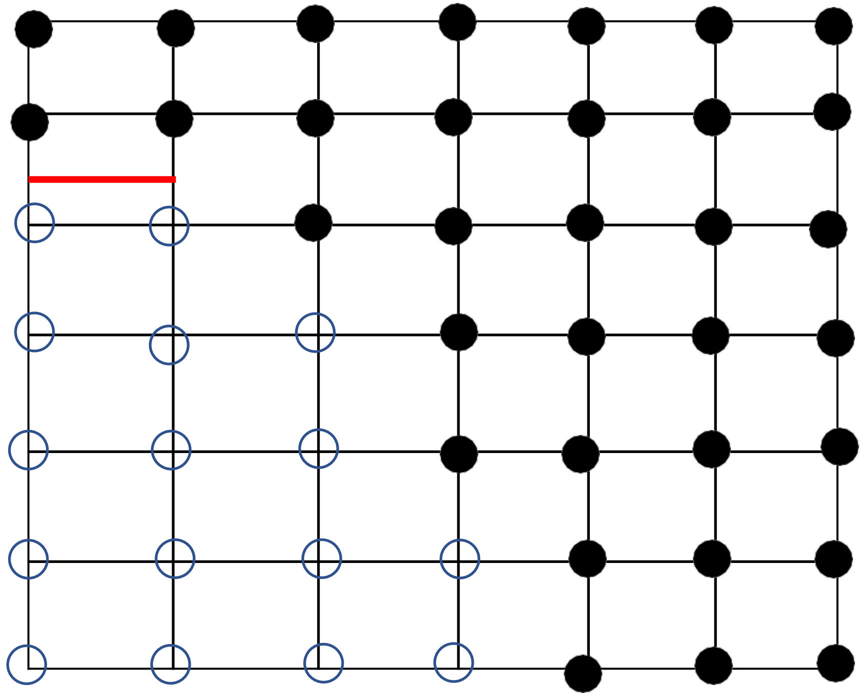


Contour value (isovalue) = C

- ● Value > C
- ○ Value < C

2D Isocontour Extraction: Marching Squares

- Given a 2D scalar field, compute isocontour (isoline) for isovalue = C
- This is usually done in a cell-by-cell manner using Marching Squares algorithm

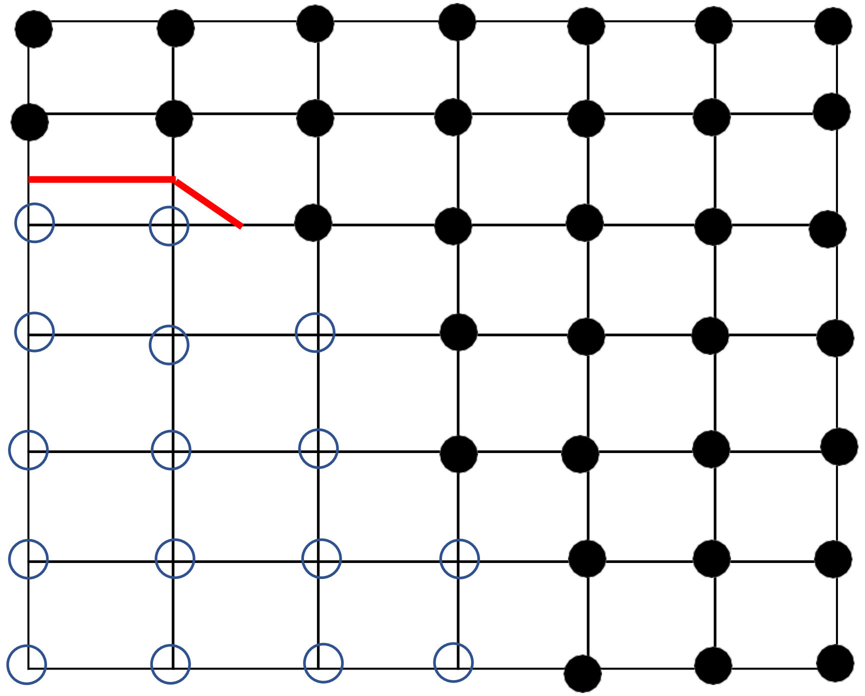


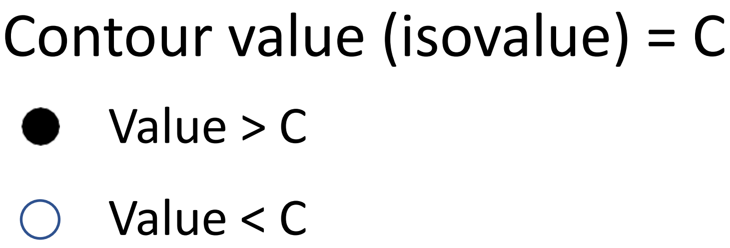
Contour value (isovalue) = C

- ● Value > C
- ○ Value < C

2D Isocontour Extraction: Marching Squares

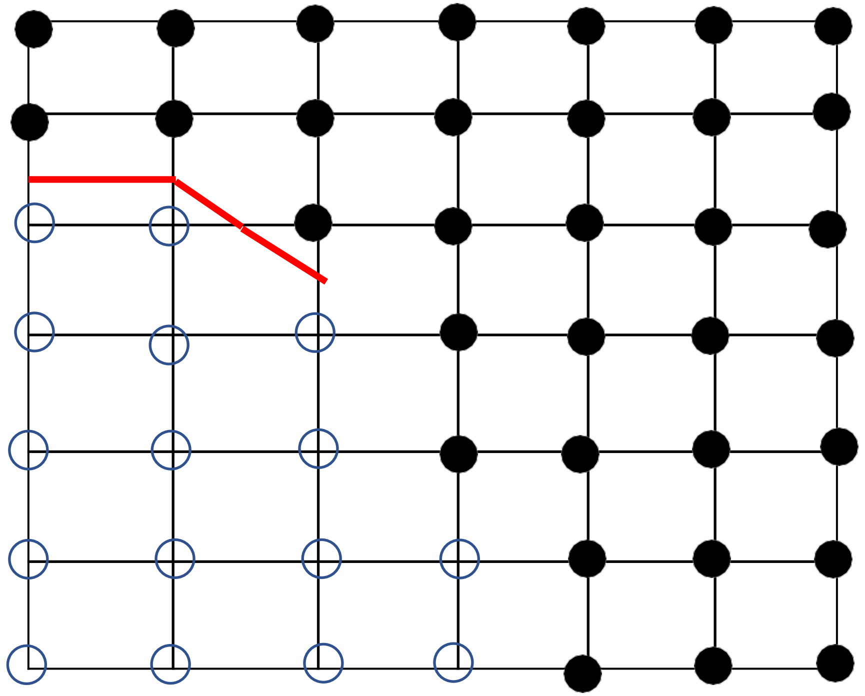
- Given a 2D scalar field, compute isocontour (isoline) for isovalue = C
- This is usually done in a cell-by-cell manner using Marching Squares algorithm



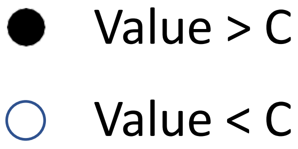


2D Isocontour Extraction: Marching Squares

- Given a 2D scalar field, compute isocontour (isoline) for isovalue = C
- This is usually done in a cell-by-cell manner using Marching Squares algorithm

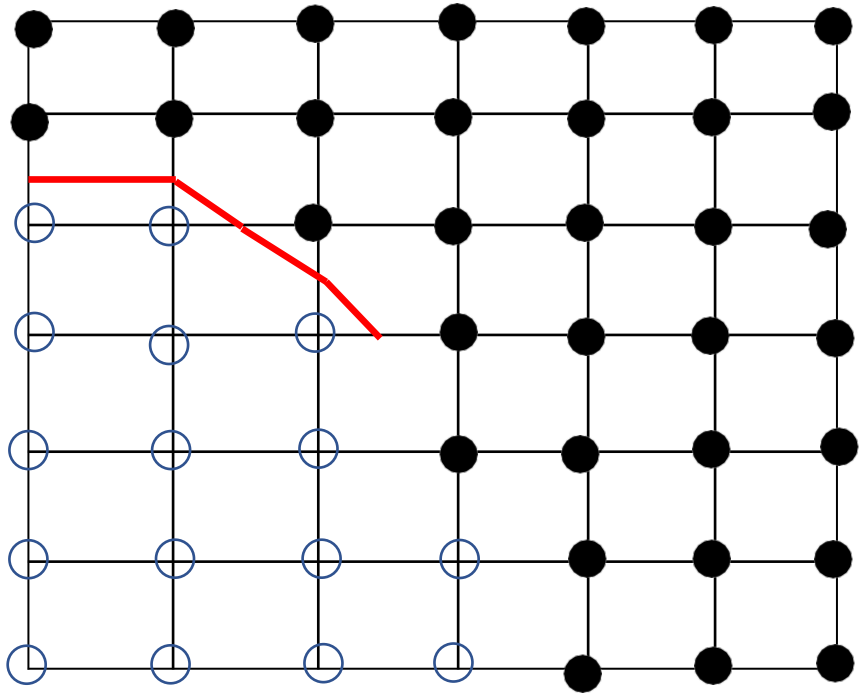


Contour value (isovalue) = C



2D Isocontour Extraction: Marching Squares

- Given a 2D scalar field, compute isocontour (isoline) for isovalue = C
- This is usually done in a cell-by-cell manner using Marching Squares algorithm



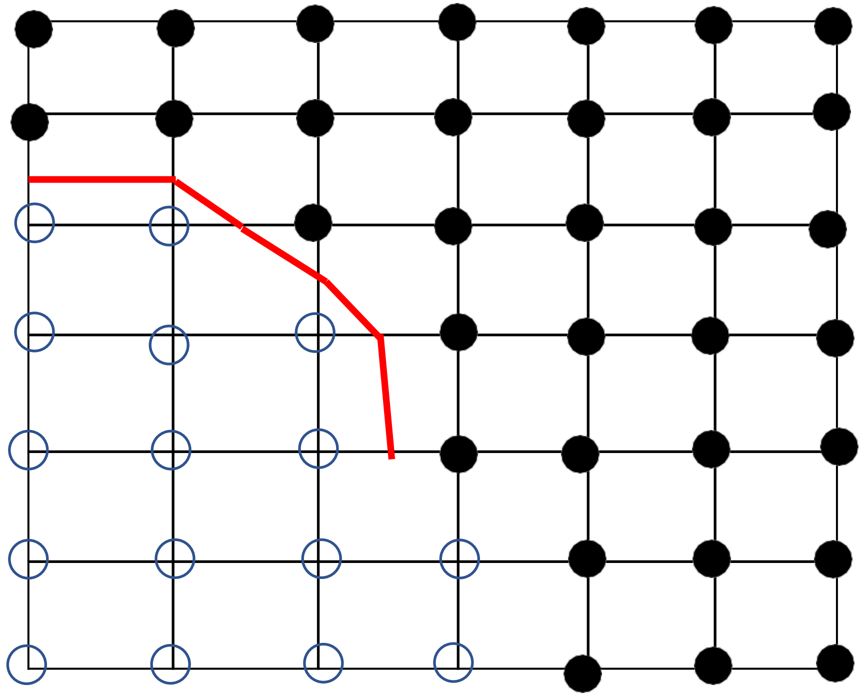
Contour value (isovalue) = C

● Value > C

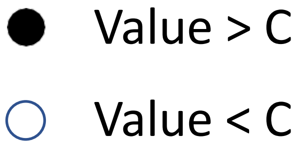
○ Value < C

2D Isocontour Extraction: Marching Squares

- Given a 2D scalar field, compute isocontour (isoline) for isovalue = C
- This is usually done in a cell-by-cell manner using Marching Squares algorithm

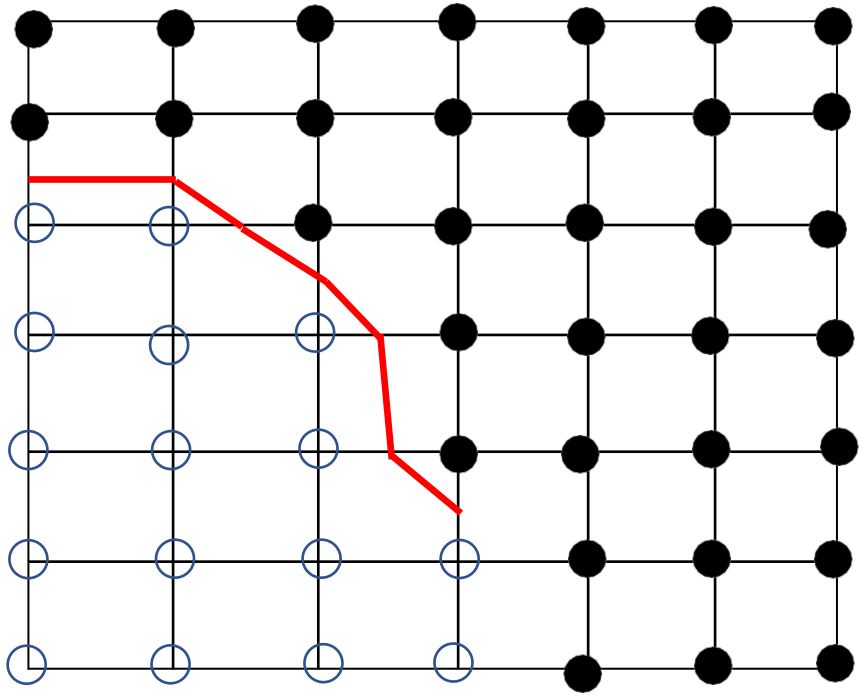


Contour value (isovalue) = C



2D Isocontour Extraction: Marching Squares

- Given a 2D scalar field, compute isocontour (isoline) for isovalue = C
- This is usually done in a cell-by-cell manner using Marching Squares algorithm

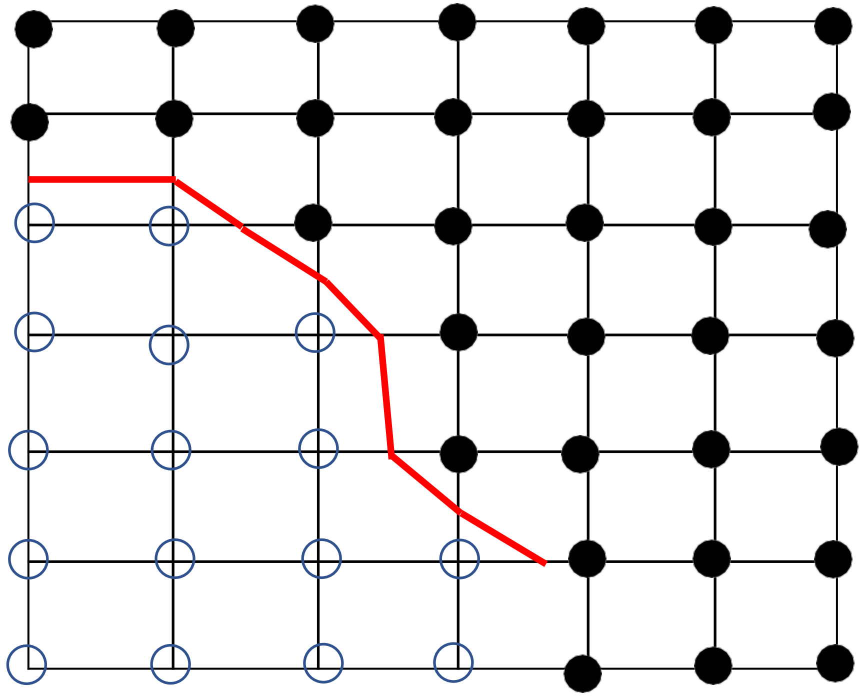


Contour value (isovalue) = C

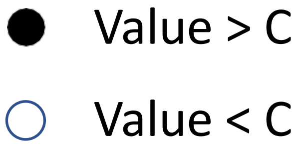
- ● Value > C
- ○ Value < C

2D Isocontour Extraction: Marching Squares

- Given a 2D scalar field, compute isocontour (isoline) for isovalue = C
- This is usually done in a cell-by-cell manner using Marching Squares algorithm

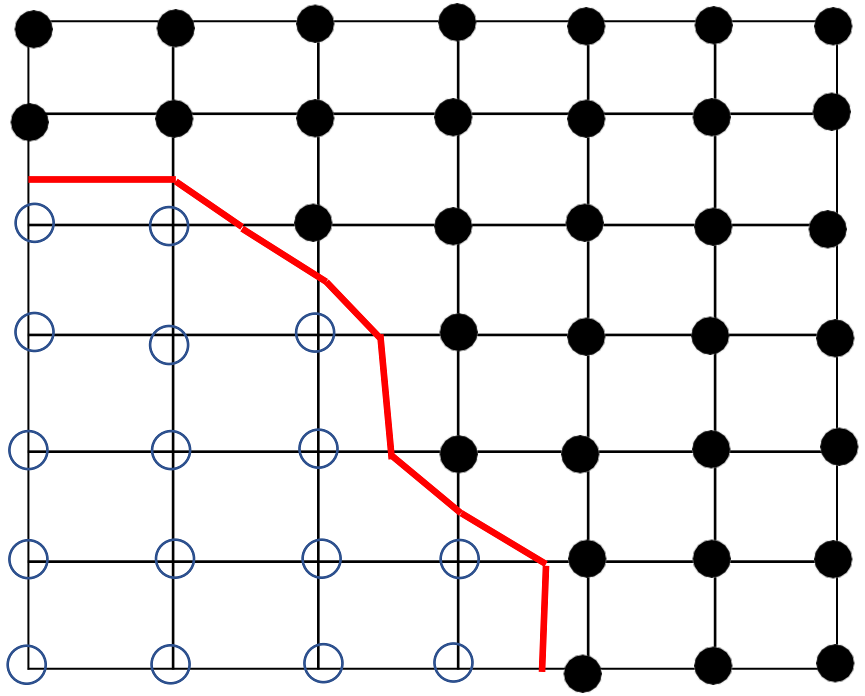


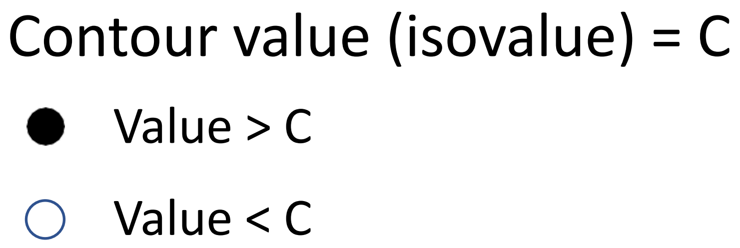
Contour value (isovalue) = C



2D Isocontour Extraction: Marching Squares

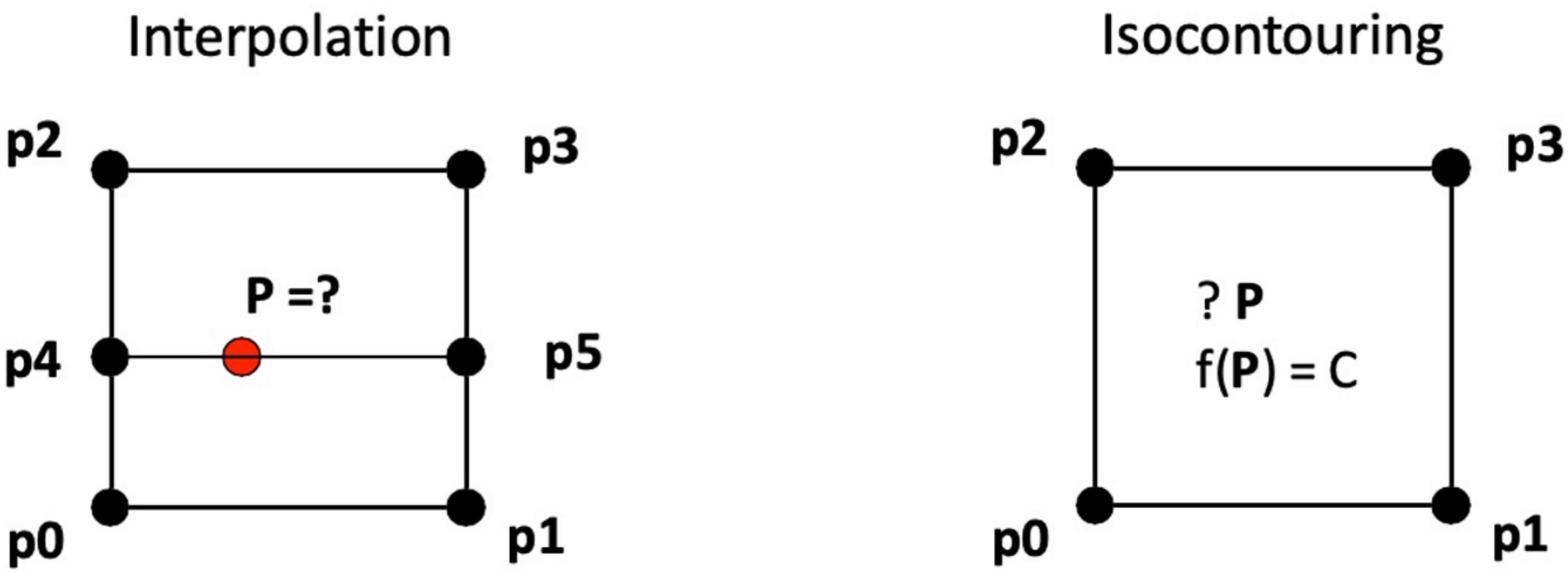
- Given a 2D scalar field, compute isocontour (isoline) for isovalue = C
- This is usually done in a cell-by-cell manner using Marching Squares algorithm





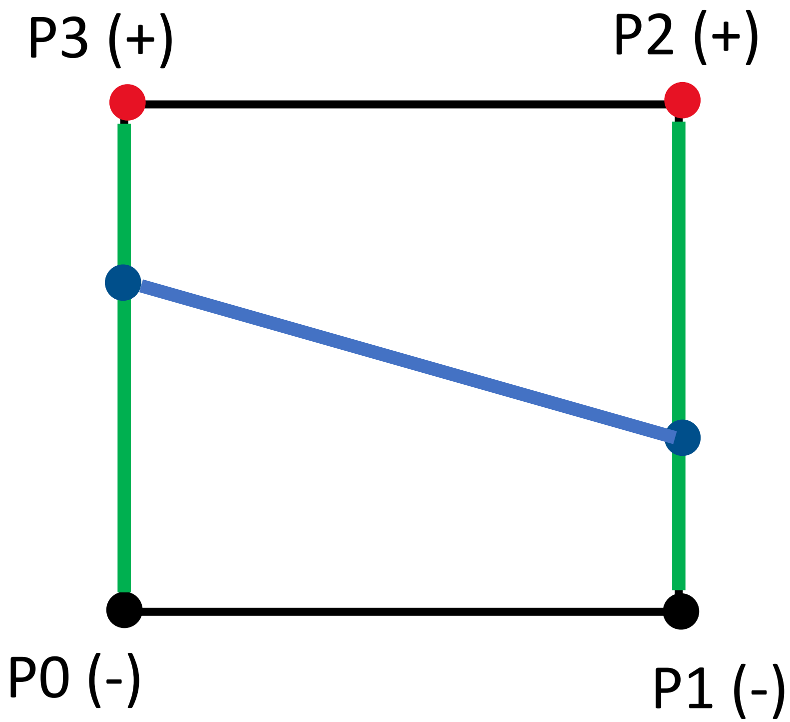
Isocontour in a 2D Cell

- Finding Isocontour in a cell is an **inverse problem of value interpolation**



Isocontouring by Linear Interpolation

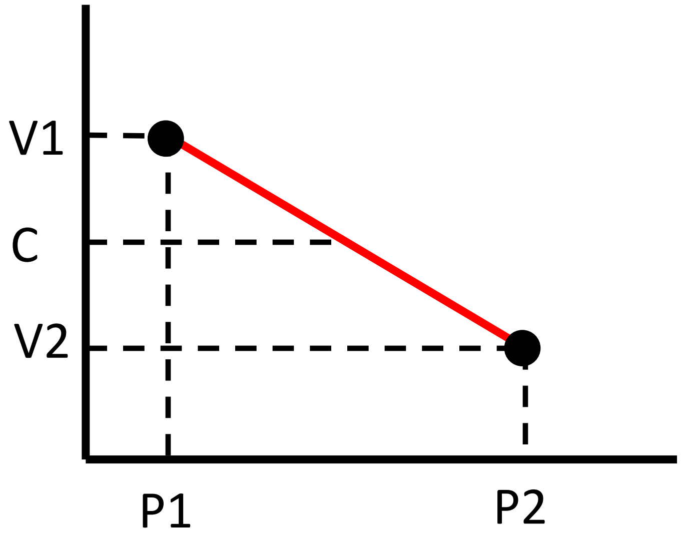
- Compute isocontour within a cell based on linear interpolation



- Identify edges that are 'zero crossing'
  - Values at the two end points are greater (+) and smaller (–) than the contour value
- Calculate the positions in those edges that has value equal to isovalue
- Connect the points with a line

Step 1: Identify Edges

- Edges that have values greater (+) and less (–) than the contour values must contain a point P that has f(p) = C
  - This is based on the assumption that values vary linearly and continuously across the edge

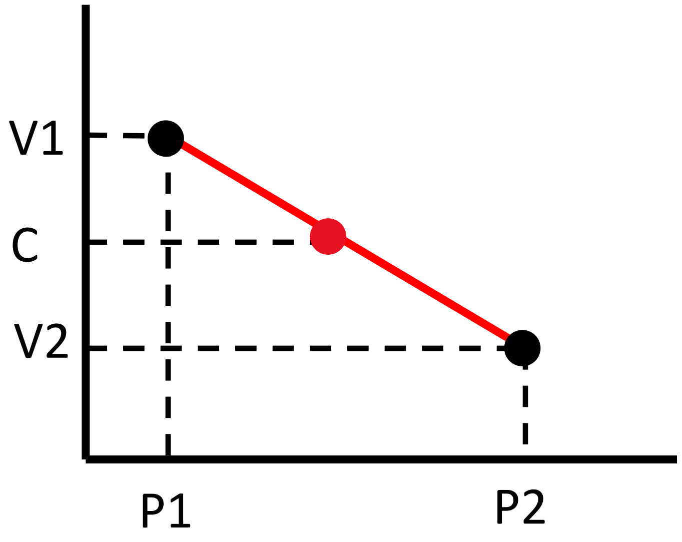


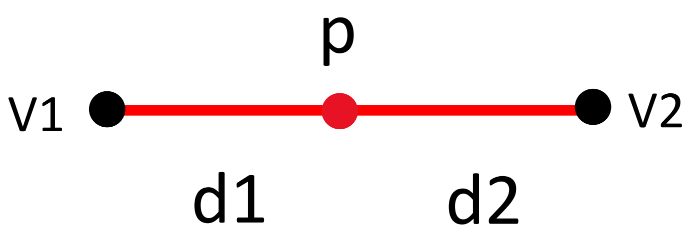
Step 2: Compute Intersection

- The intersection point **f(p) = C** on the edge can be computed by linear interpolation

$d1/d2 = (v1-C) / (C - v2)$ ⇨ $(\mathbf{p} - \mathbf{p}1)/(\mathbf{p}2 - \mathbf{p}1) = (v1-C) / (v1 - v2)$

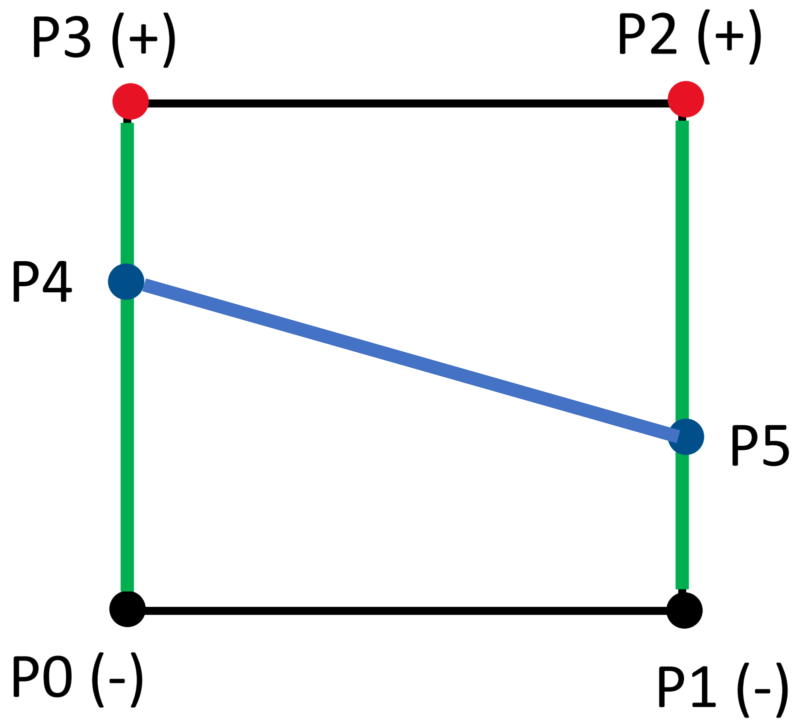
$\mathbf{p} = (v1-C)/(v1 - v2) * (\mathbf{p}2 - \mathbf{p}1) + \mathbf{p}1$





Step 3: Connect the Dots

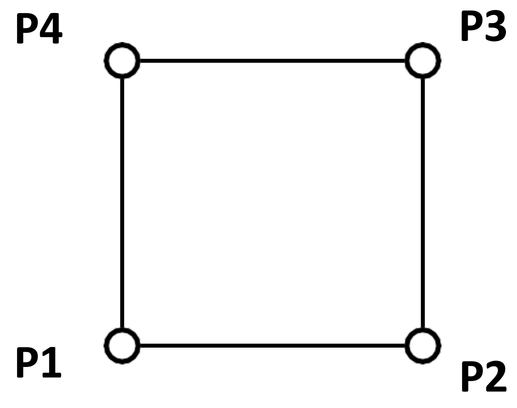
- Based on the principle of linear interpolation, all points along the line **P4P5** have values equal to C (isovalue)

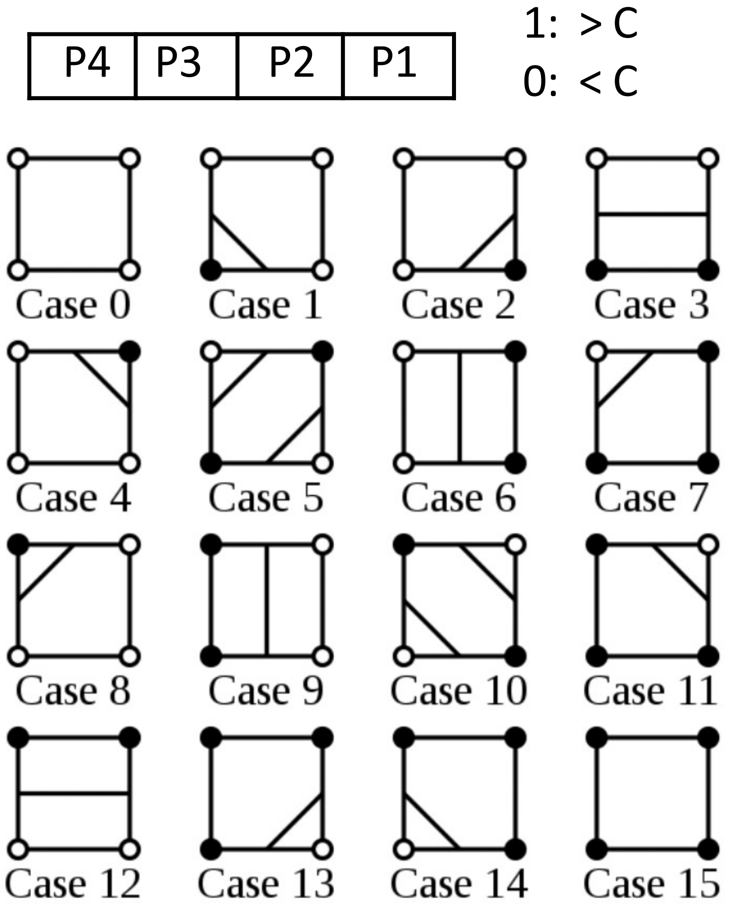


Repeat Step1 – Step 3 for all cells

Isocontour Cases

- How many ways can an isocontour intersect a rectangular cell?

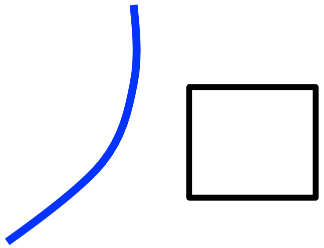




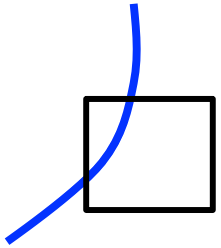
- The value at each vertex can be either greater or less than the contour value
- So, there are 2 x 2 x 2 x 2 = 16 cases

Unique Topological Cases

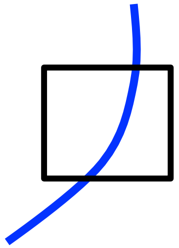
- There are only four unique topological cases



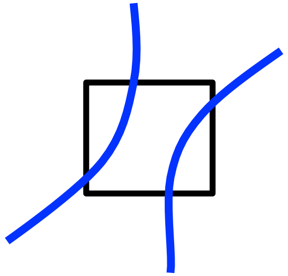
(1) No intersection



(2) Intersect with two adjacent edges



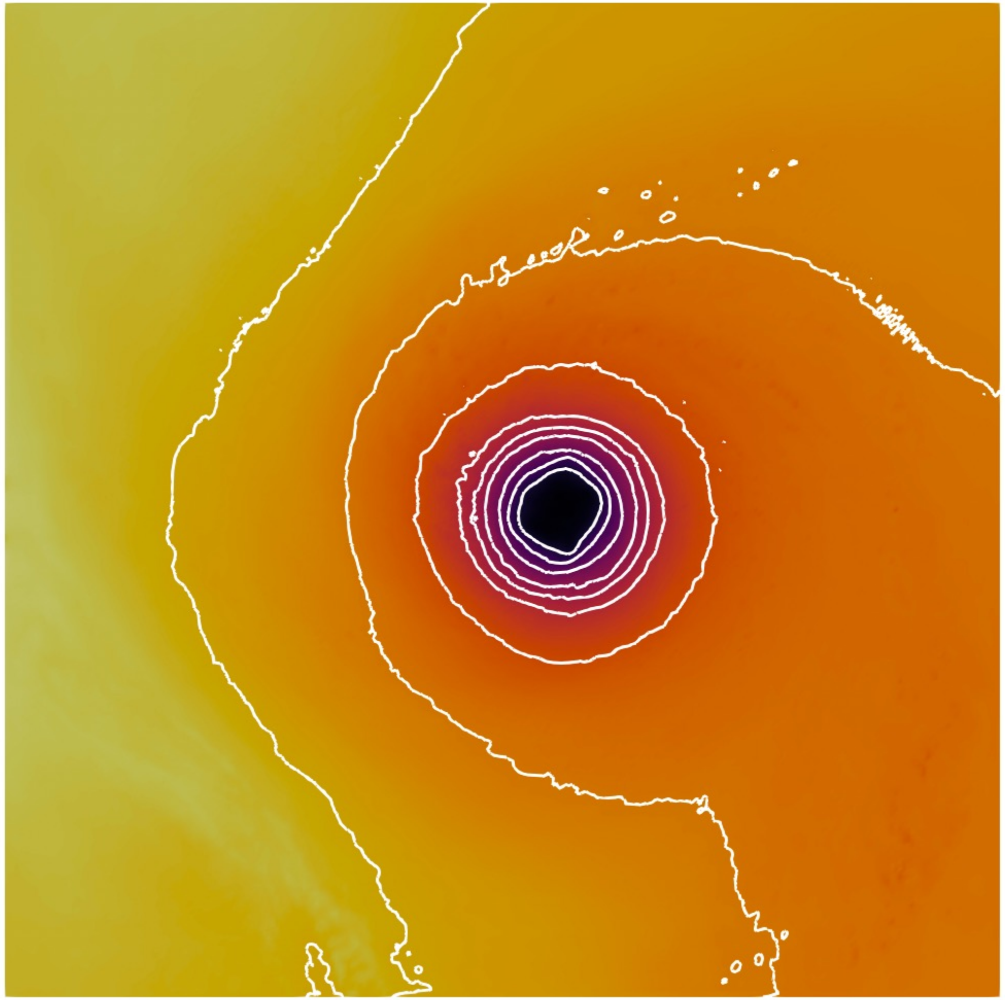
(3) Intersect with two opposite cases



(4) Two contours pass through the cell

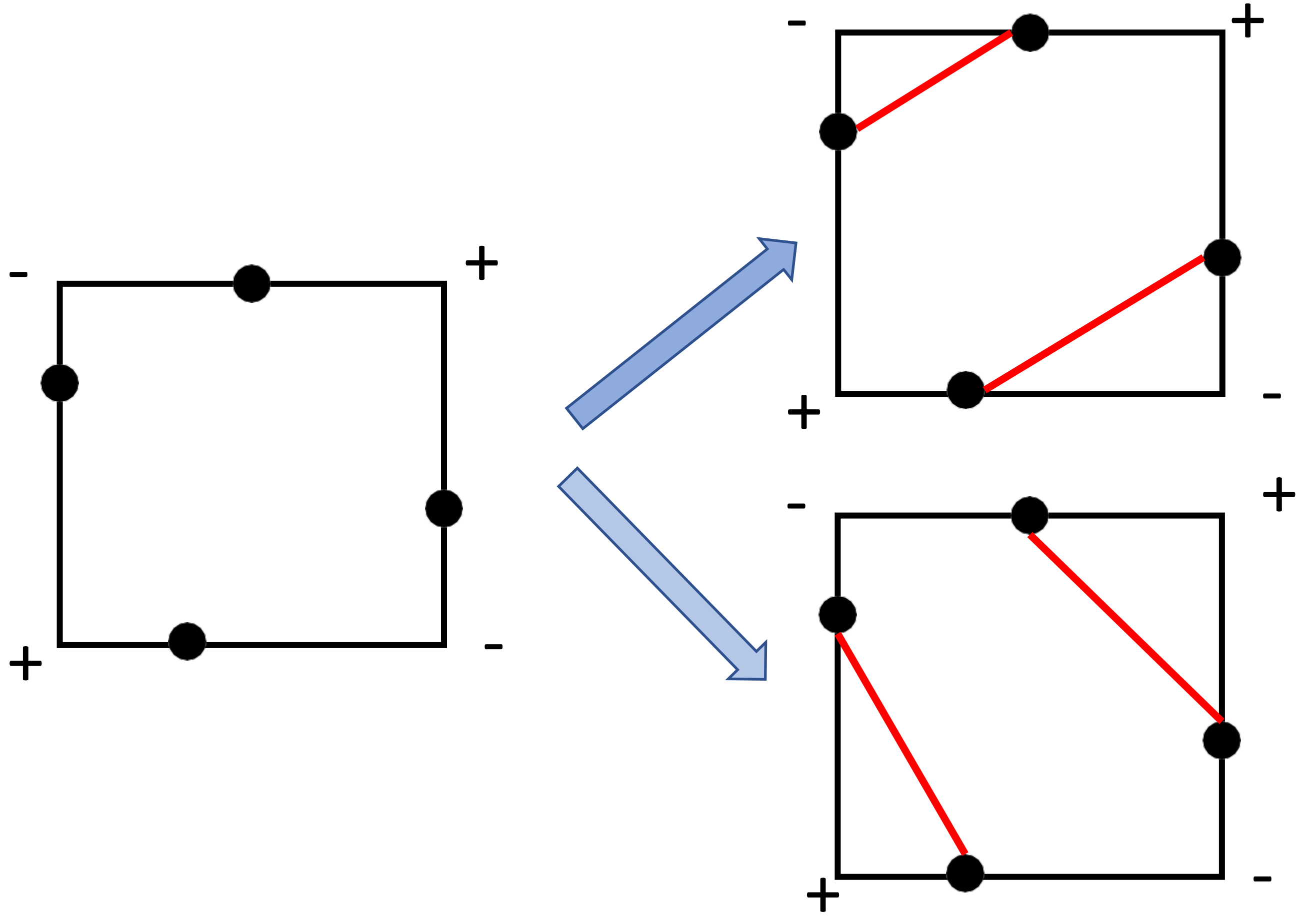
Putting it All Together

- 2D Isocontouring algorithm for square meshes:
  - Process one cell at a time
  - Compare the values at 4 vertices with the contour value C and identify intersected edges
  - Linearly interpolate along the intersected edges
  - Connect the interpolated points together



Dealing with Ambiguous Cases

- Ambiguous face: A face that has two diagonally opposite points with the same sign

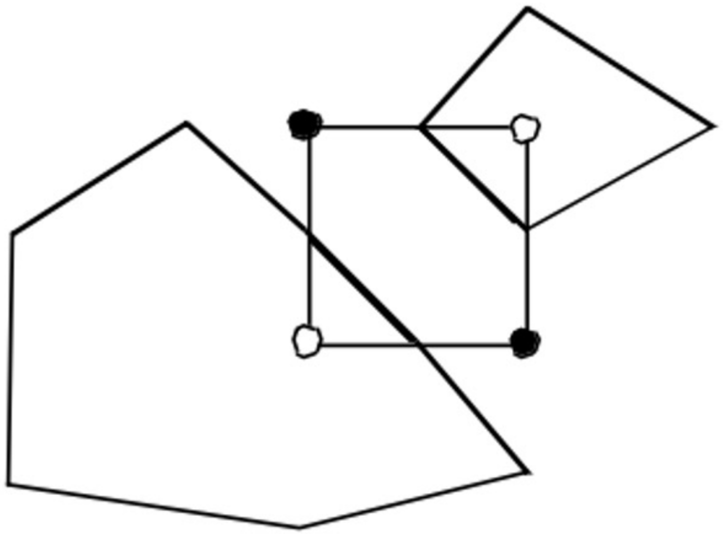


How to connect?

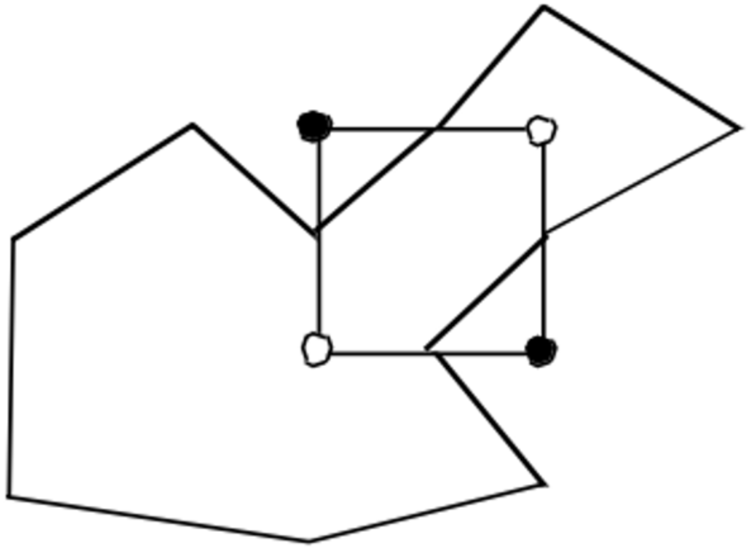
Both configurations are possible!

Dealing with Ambiguous Cases

- Ambiguous face: A face that has two diagonally opposite points with the same sign



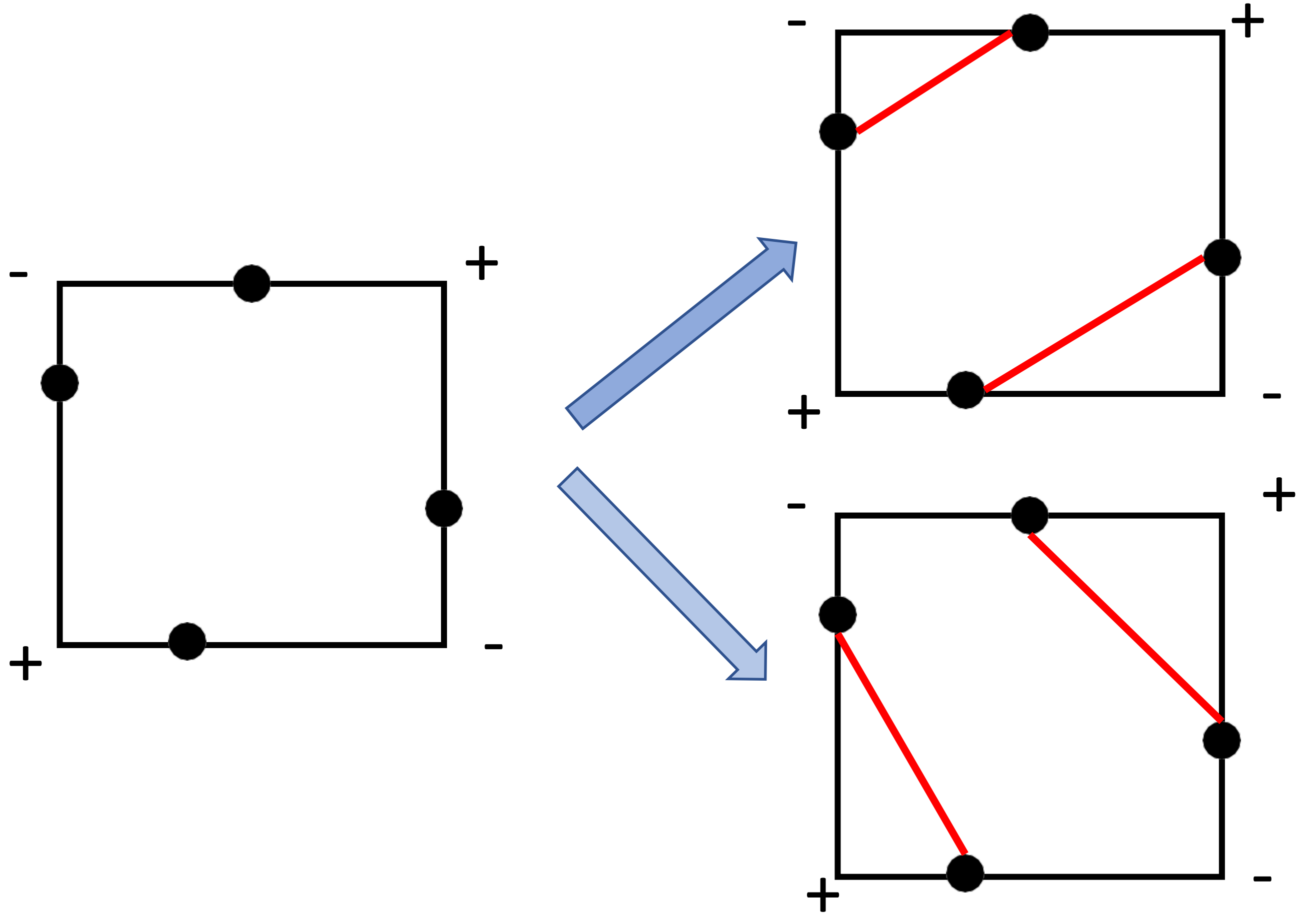
Broken Contour



Connected Contour

Dealing with Ambiguous Cases

- Ambiguous face: A face that has two diagonally opposite points with the same sign



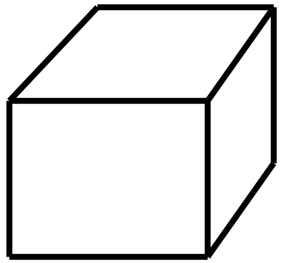
How to connect?
Both configurations are possible!

- One way to resolve: Use Asymptotic decider

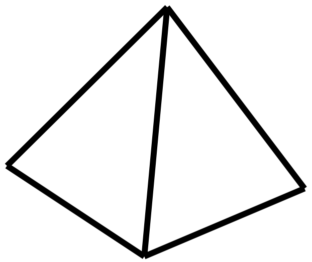
The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes by Nielson and Hamman, IEEE VIS'91

3D Isocontour: Isosurface

- The 2D algorithm extends naturally to 3D where the data will have 3D cells
- Identify 'active cells': cells that intersect with the Isosurface
- Linear interpolation along edges in active cells
- Compute surface patches within each cell based on the edges that have intersected with the Isosurface



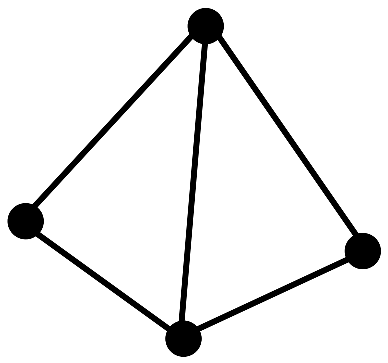
Cube/Rectangular cell



Tetrahedron cell

Tetrahedral Cell

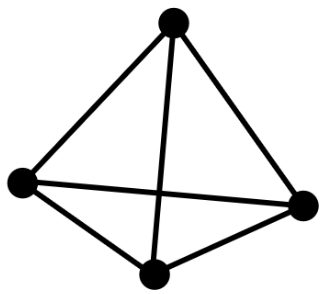
- Active cells: min value < C < max value
- Mark cell vertices that are greater than C with "+" and smaller than C with "-"
- Each cell has 4 vertices
  - Each vertex can have value greater or less than C
  - Hence, 2x2x2x2 = 16 possible combinations
  - Only <u>three unique topological cases</u>



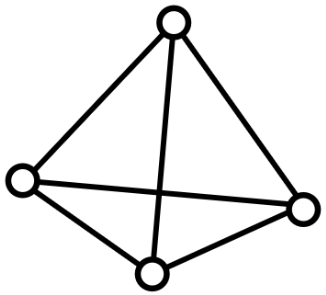
Tetrahedron cell

Tetrahedral Cell: Case 1

- Case 1: No intersection (all vertices are either outside or inside)
- Values at all cell vertices are either larger or smaller than the isovalue C
  - If we assume that cell values greater than the contour value C as ‘outside’ and smaller as ‘inside’, then all cell vertices are either completely inside or outside of the isosurface



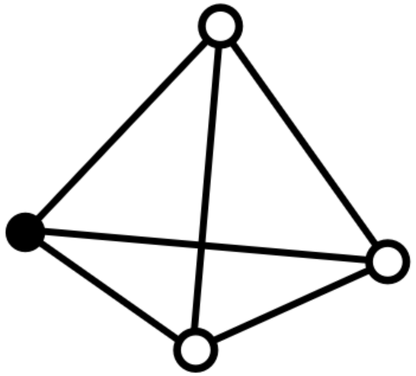
All Vertices Outside



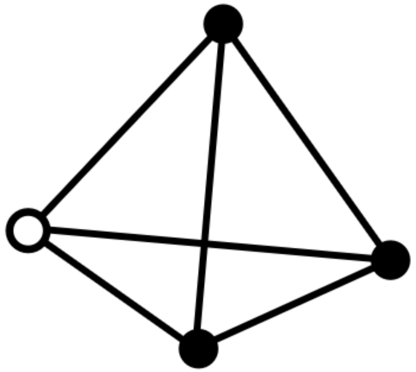
All Vertices Inside

Tetrahedral Cell: Case 2

- Case 2: One vertex outside (or inside)
- Isosurface only intersects with edges that have '+' and '–' vertices at two ends



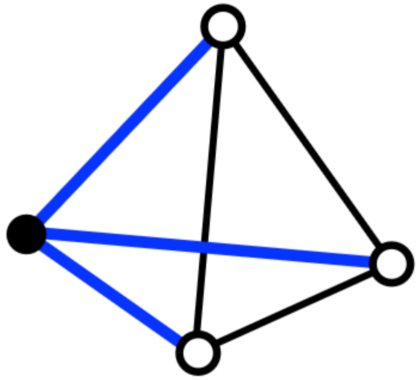
One Outside



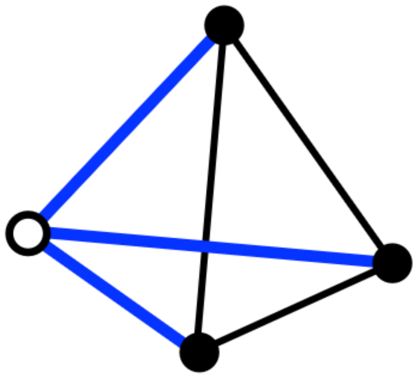
One Inside

Tetrahedral Cell: Case 2

- Case 2: One vertex outside (or inside)
- Isosurface only intersects with edges that have '+' and '–' vertices at two ends



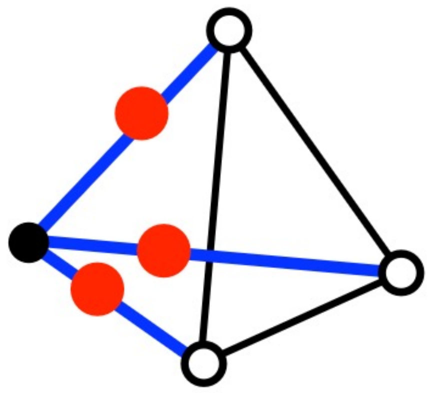
One Outside



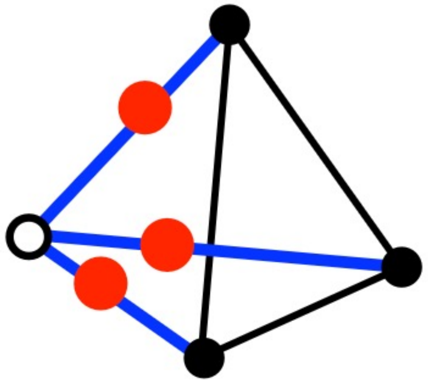
One Inside

Tetrahedral Cell: Case 2

- Case 2: One vertex outside (or inside)
- Compute intersection points on active edges



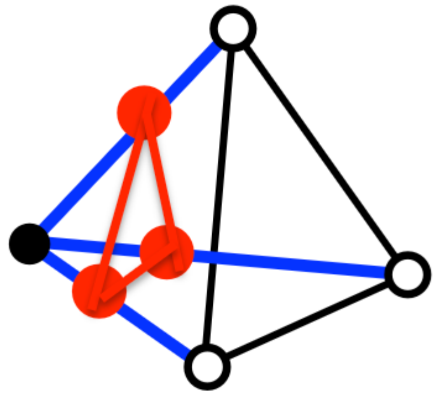
One Outside



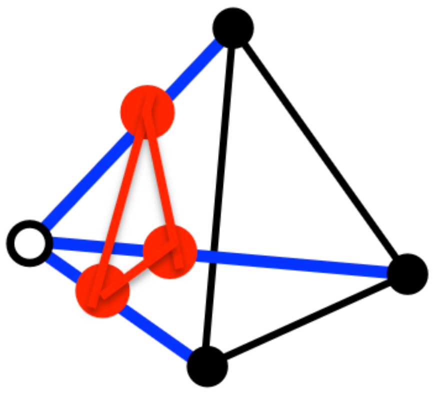
One Inside

Tetrahedral Cell: Case 2

- Case 2: One vertex outside (or inside)
- Connect intersection points into a triangle



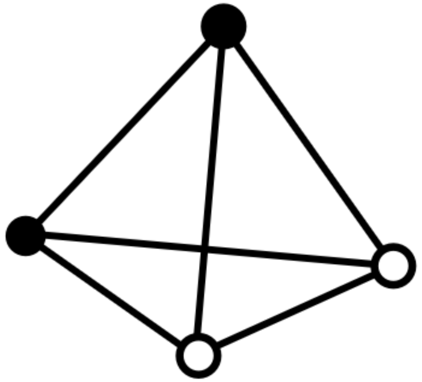
One Outside



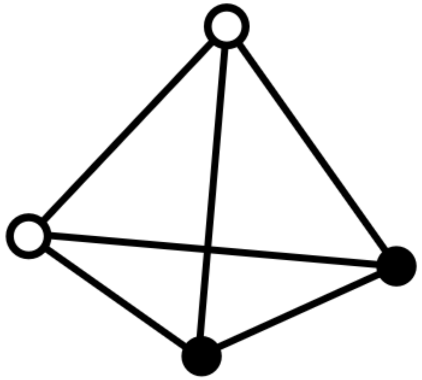
One Inside

Tetrahedral Cell: Case 3

- Case 3: Two vertices outside (or inside)
- Isosurface only intersects with edges that have '+' and '–' vertices at two ends



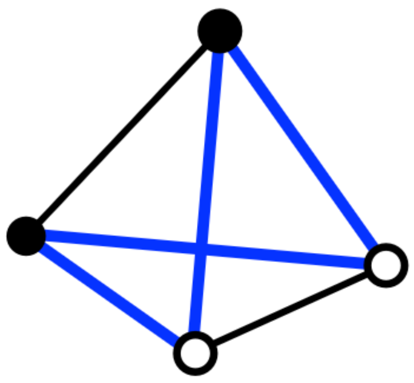
One Outside



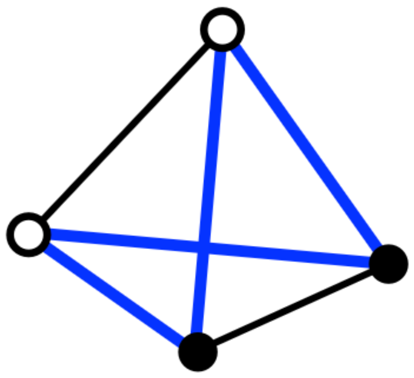
One Inside

Tetrahedral Cell: Case 3

- Case 3: Two vertices outside (or inside)
- Isosurface only intersects with edges that have '+' and '–' vertices at two ends



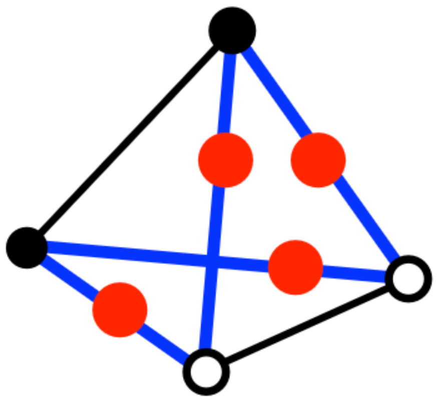
One Outside



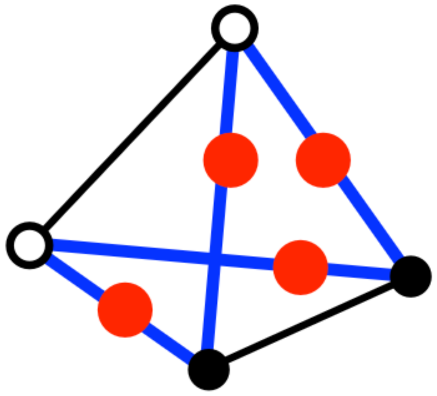
One Inside

Tetrahedral Cell: Case 3

- Case 3: Two vertices outside (or inside)
- Compute intersection points on active edges



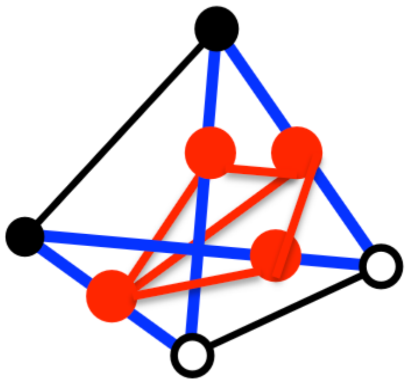
One Outside



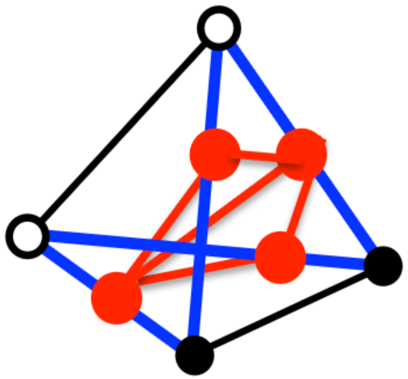
One Inside

Tetrahedral Cell: Case 3

- Case 3: Two vertices outside (or inside)
- Connect intersection points into a triangle

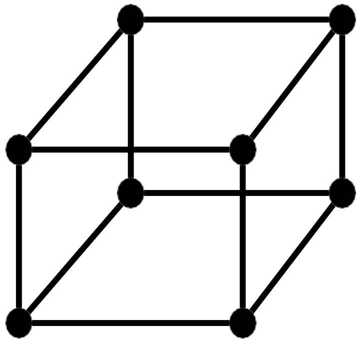


One Outside



One Inside

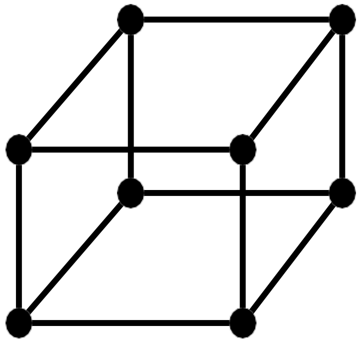
3D Isocontour: Cube/Rectangular Cells



Cube/Rectangular cell

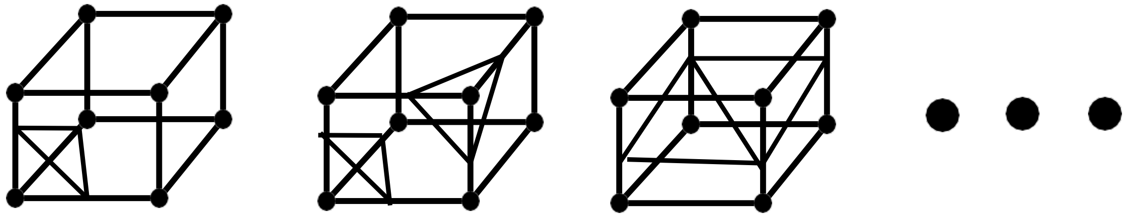
- With 8 vertices in a cell, each having a value greater or smaller than the contour value, there can be $2^8 = 256$ possible cases

3D Isocontour: Cube/Rectangular Cells



Cube/Rectangular cell

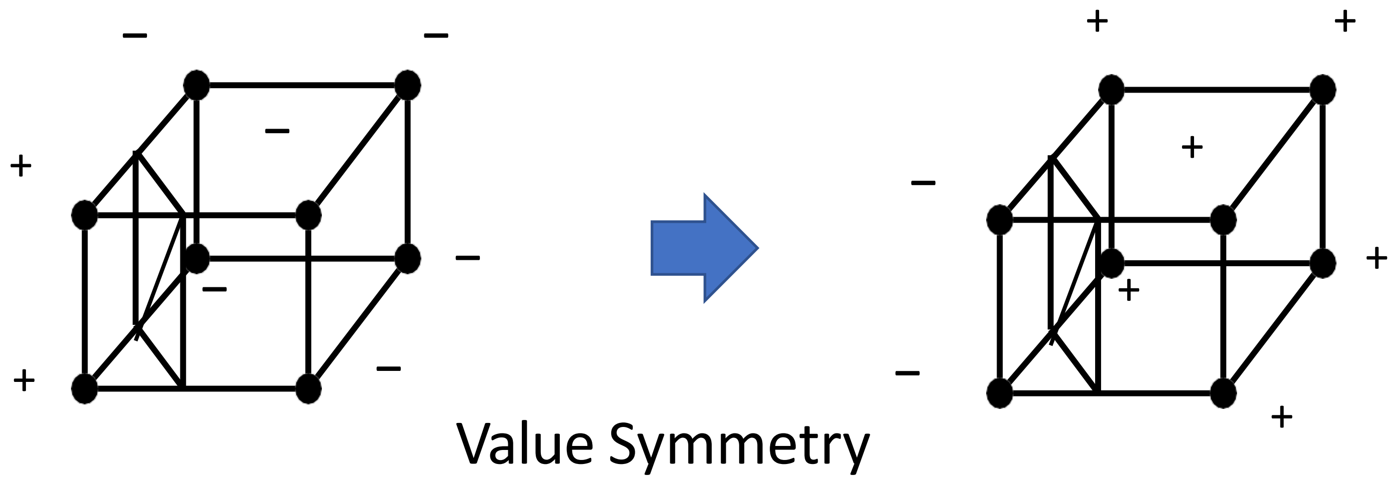
- With 8 vertices in a cell, each having a value greater or smaller than the contour value, there can be $2^8 = 256$ possible cases



But the total number of unique topological cases is much less than 256

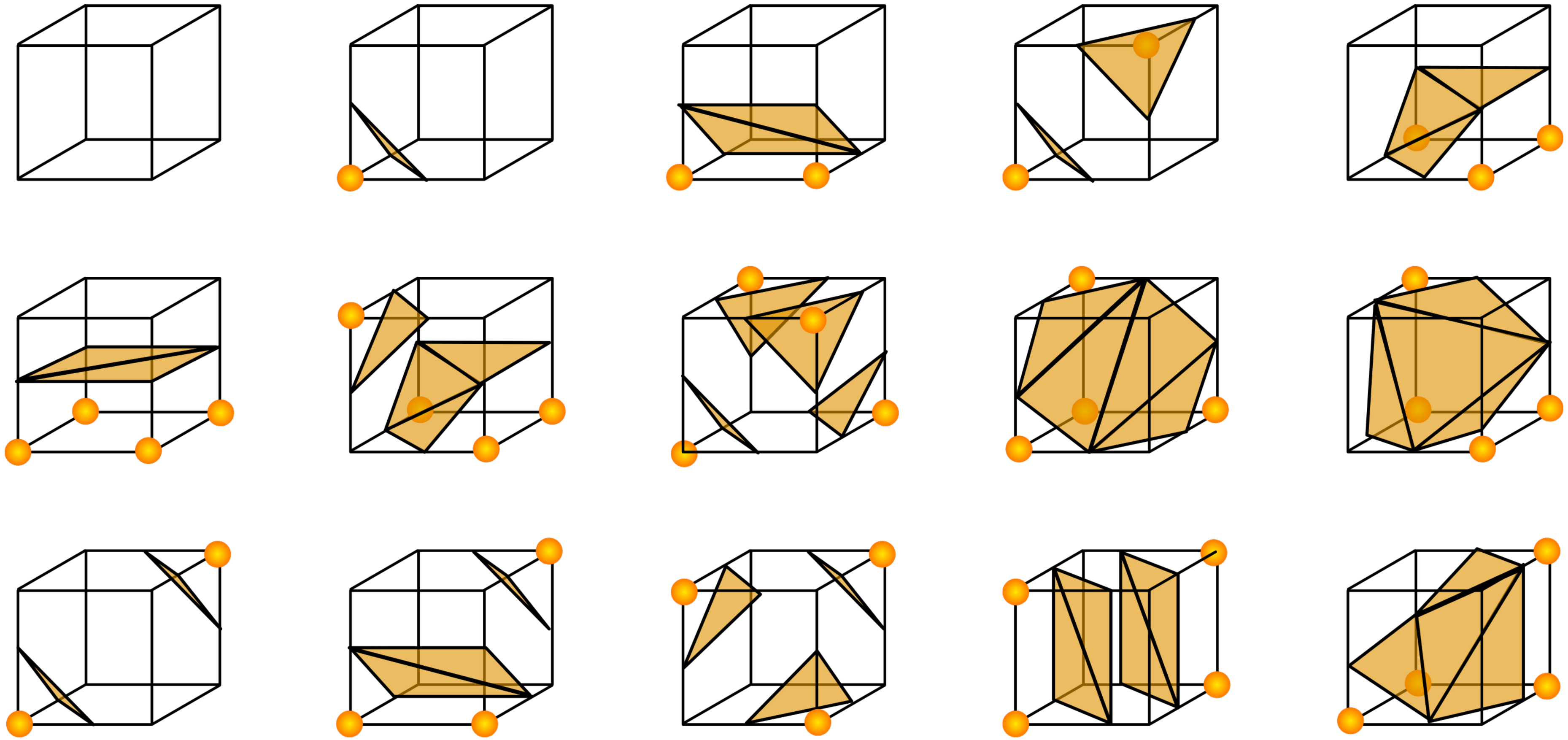
Case Reduction

- The topology of the surface does not change, and the unique number of cases reduces to 15 from 256
  - Value Symmetry
  - Rotational Symmetry



3D Isosurface Unique Cases

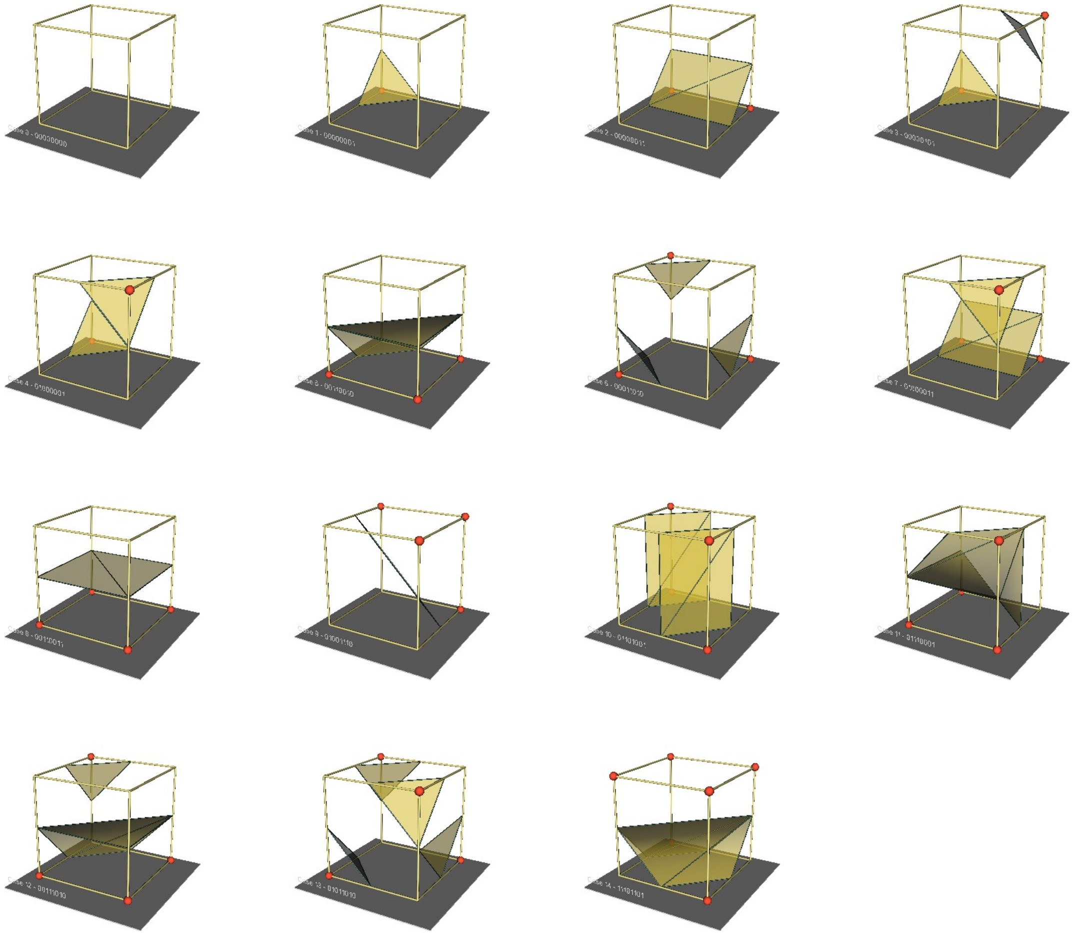
- 15 Topologically Unique Cases



Wikipedia

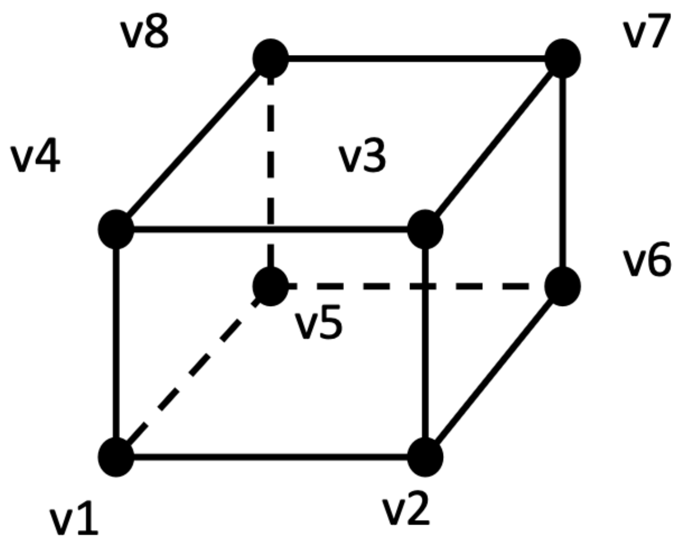
VTK Demo: Marching Cubes Cases

Source: https://gitlab.kitware.com/vtk/vtk-examples/-/tree/master/src/Python/VisualizationAlgorithms/MarchingCases.py



Marching Cubes Algorithm

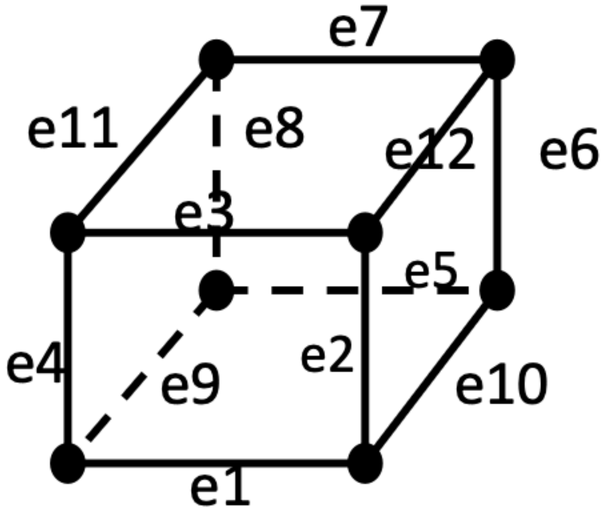
- Lorensen and Cline in 1987
- Mark each cell corner with a bit
  - V_i is 1 if value > C (C=isovalue)
  - V_i is 0 if value < C
- Each cell has an index mapped to a value ranged [0,255]



Index =	v8	v7	v6	v5	v4	v3	v2	v1

Marching Cubes Algorithm

- Based on the values at the vertices, map the cell to one of the 15 cases
- Perform a table lookup to see what edges have intersections



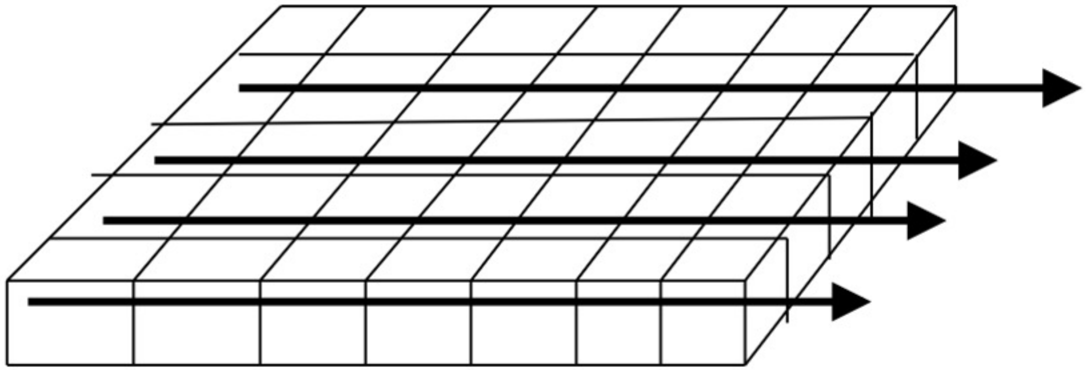
Index =

v8	v7	v6	v5	v4	v3	v2	v1

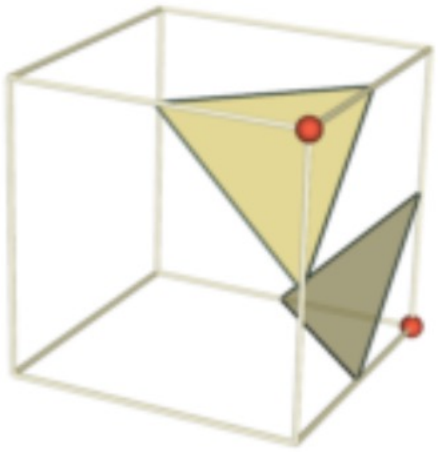
Index	intersection edges
0	e1, e3, e5
1	...
2	
3	
	• • •
14	

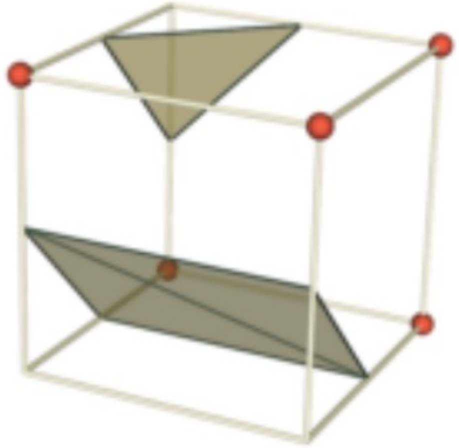
Marching Cubes Algorithm

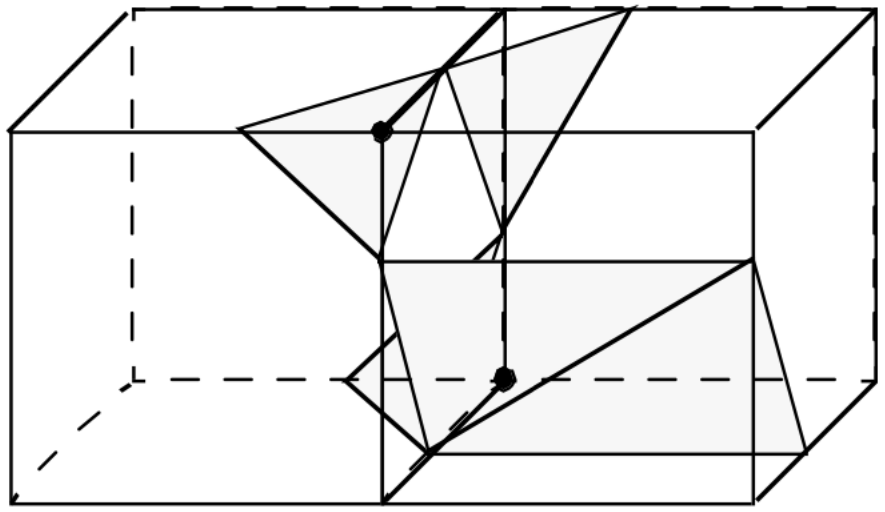
- Perform linear interpolation to compute the intersection points at the edges
- Connect the points to form surface patches
- Sequentially scan through the cells – row by row, layer by layer



Ambiguity in Marching Cubes


Case 3


Case 6c



Arbitrarily choosing marching cubes cases leads to holes in the isosurface

Dealing With Ambiguity

- Use ‘Asymptotic decider’
  - Idea is similar and extends to 3D
  - More details can be found is the paper: “Resolving the Ambiguity in Marching Cubes” by Nielson and Hamman, IEEE VIS’91

Marching Cubes Algorithm: Animation