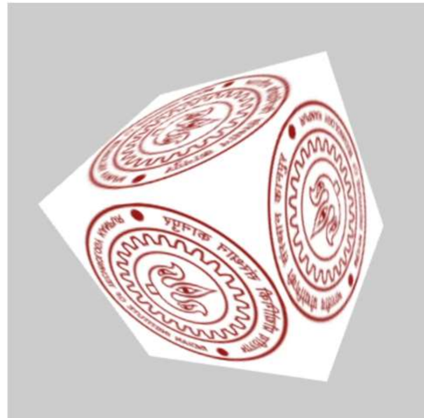


Introduction to Computer Graphics (CS360A)

Instructor: Soumya Dutta

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur (IITK)

email: soumyad@cse.iitk.ac.in

Acknowledgements

- A subset of the slides that I will present throughout the course are adapted/inspired by excellent courses on Computer Graphics offered by Prof. Han-Wei Shen, Prof. Wojciech Matusik, Prof. Frédo Durand, Prof. Abe Davis, and Prof. Cem Yuksel

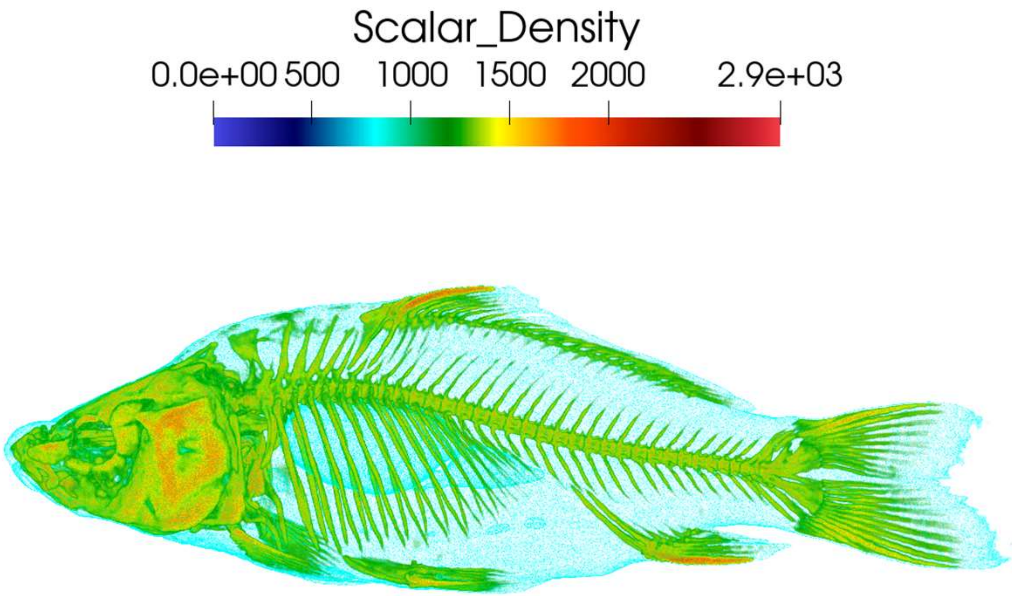
Course Feedback/Survey

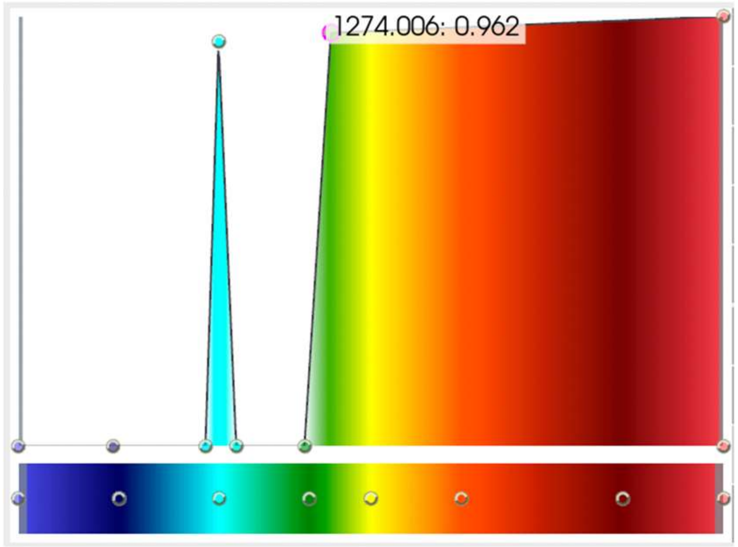
IITK CS360A

- Please fill out the online survey for CS360A
- Online Student Reaction Survey (SRS)
    - Ends on Nov 15, 2023 (23:59 hrs.)
- Please fill up the online survey form using the following link: https://oars.iitk.ac.in/survey/survey.php
- Your login details are:
    - User Name: Roll No
    - Password: DOB (DDMMYYYY) format
- For any issue while logging in, send an email to courses@iitk.ac.in

Transfer Function Design

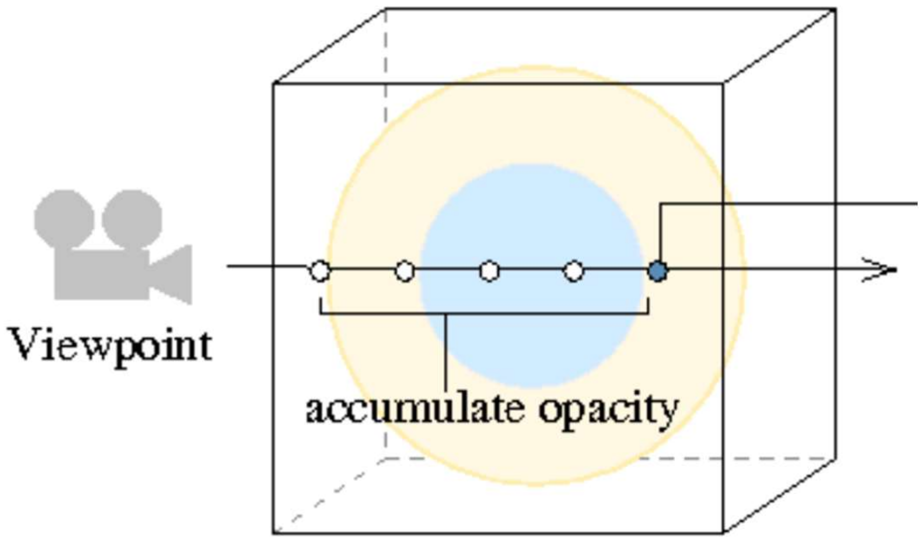
- Distinguish between different materials or features in the data





Visibility Histogram Guided Transfer Functions

- A semi-automatic approach for generating opacity transfer function
- The visibility of a sample refers to the contribution of a sample to the final image, in terms of opacity
- Visibility depends on
  - Opacity of the sample
  - The viewpoint which affects the accumulated opacity in front of the sample



Visibility Histogram Guided Transfer Functions

IITK CS360A

- Visibility Histogram: Distribution of the visibility function in relation to the domain values of the volume

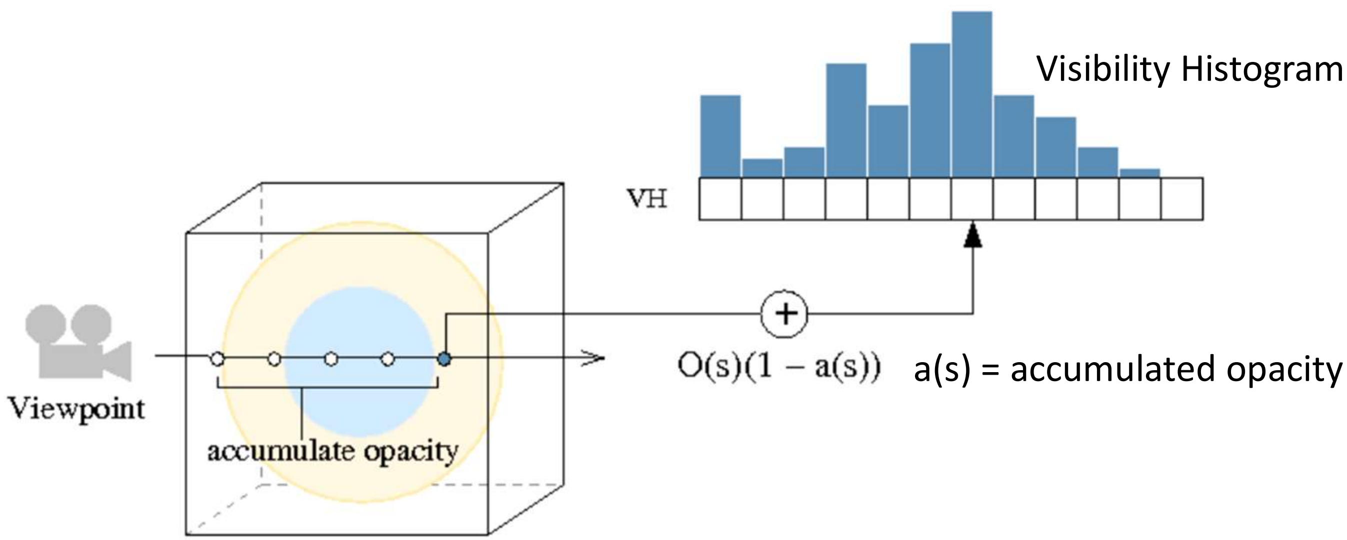
$$VH(x) = O(x) \int_{s \in \Omega} \delta(s,x)(1-\alpha(s))ds$$

$$\delta(s,x) = \begin{cases} 1 & V(s) = x \\ 0 & otherwise \end{cases}$$

x = data value at sample s
VH(x) = Visibility Histogram
O(x) = Opacity of value x
V = Volume
$\alpha(s)$ = accumulated opacity in front of the sample s

Visibility Histogram Guided Transfer Functions

- Visibility Histogram: Distribution of the visibility function in relation to the domain values of the volume

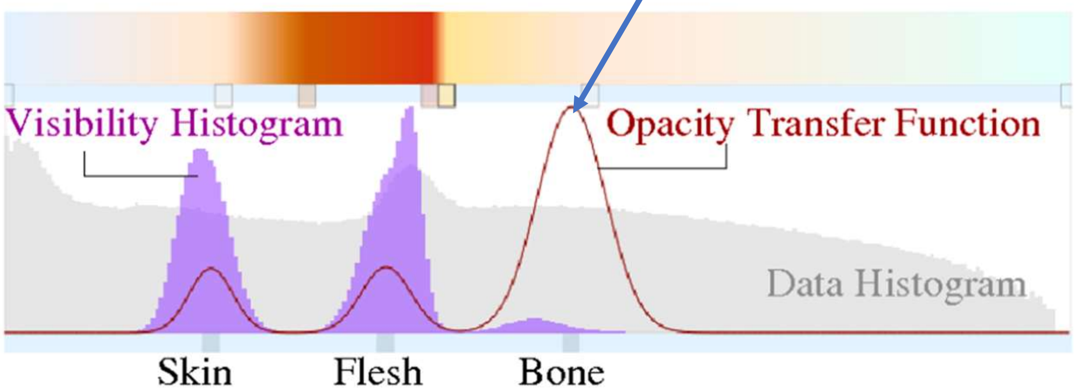


Visibility Histogram Guided Transfer Functions

- Visibility Histogram: Distribution of the visibility function in relation to the domain values of the volume

But we can't see bons as flesh and skin is blocking it

User wants to see bones with clarity

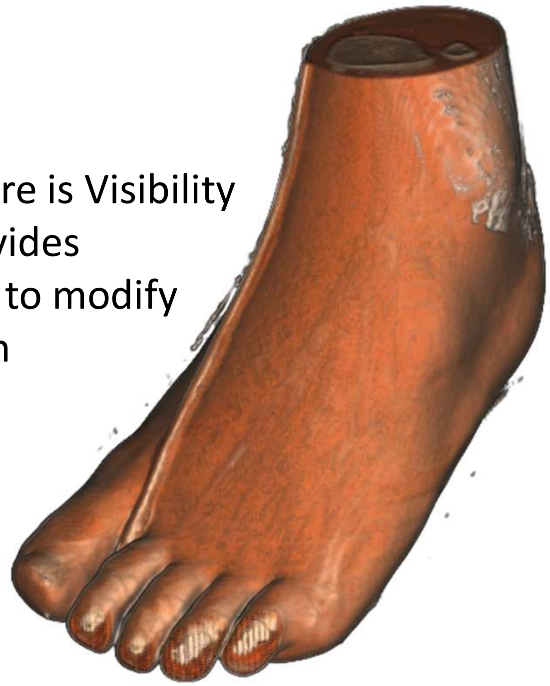


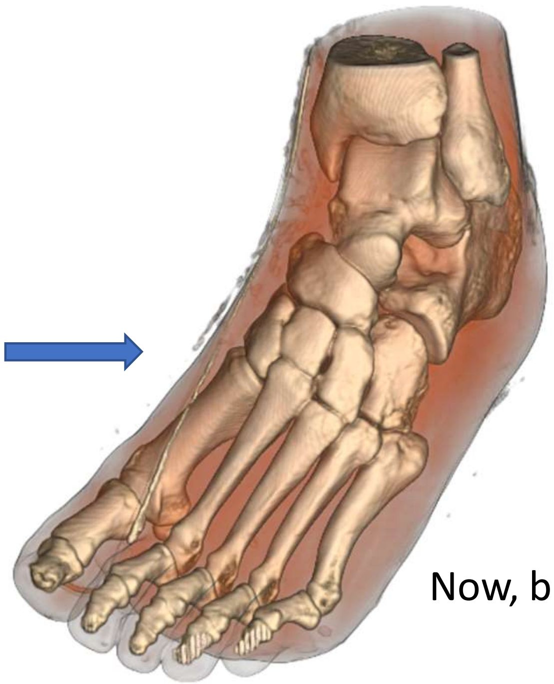
Visibility Histogram can provide the guidance

Visibility Histogram Guided Transfer Functions

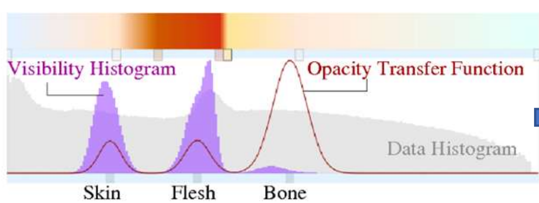
Still manual but there is Visibility Histogram that provides guidance as to how to modify the opacity function

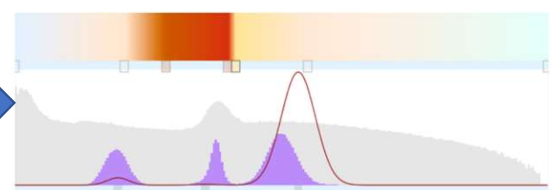
Can we do better?





Now, bones are clearly seen



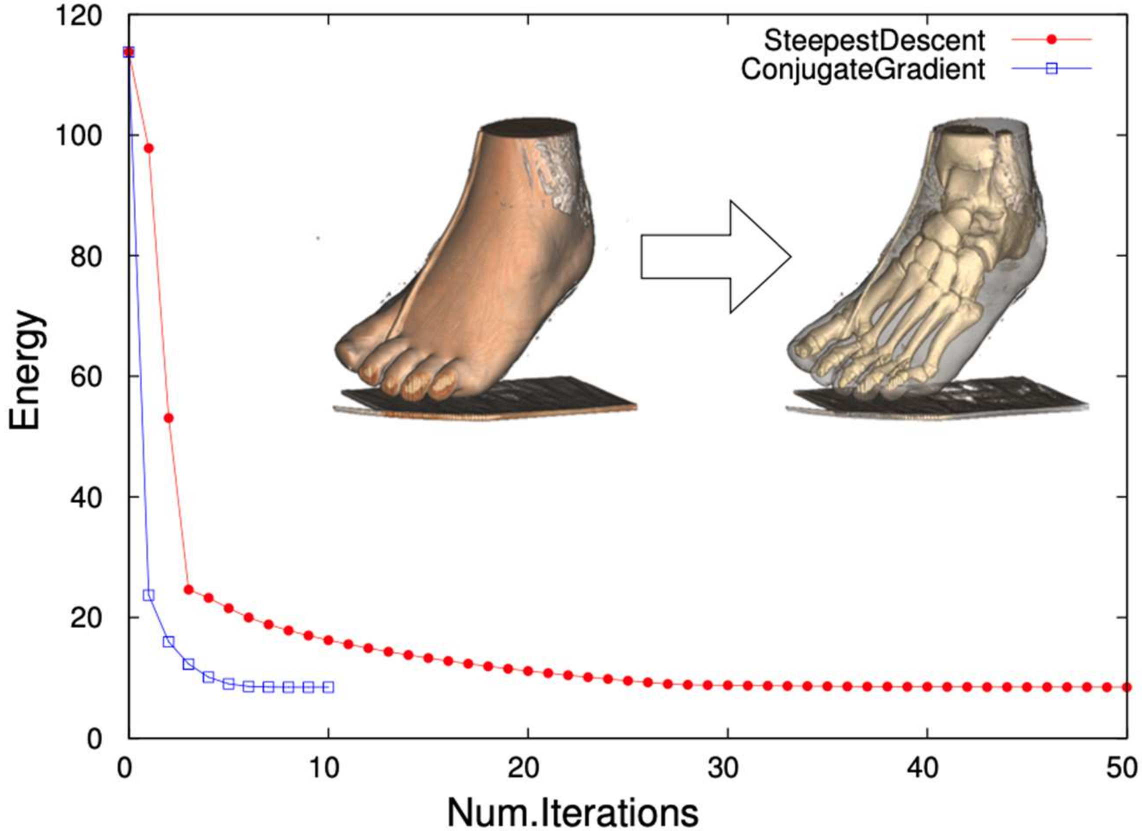


Visibility Histogram Guided Transfer Functions

IITK CS360A

- Semi-automatic transfer function design using Visibility Histogram
- Use Optimization techniques
  - selection of the best solution, with regard to some criterion, from some set of available alternatives by minimizing(maximizing) an energy function
- Energy function here should consider characteristics of a good opacity transfer function
  - User satisfaction: minimize mismatch between user provided initial and computer transfer function
  - Visibility: maximize visibility of samples
  - Constraints: constraints for opacity transfer function parameters

Visibility Histogram Guided Transfer Functions



Isocontour Algorithm (2D and 3D)

What is an Isocontour?

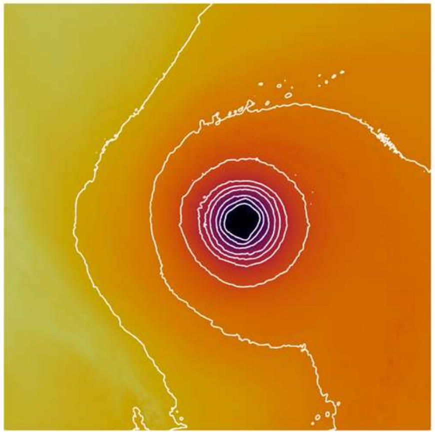
IITK CS360A

- An isocontour is a curve(2D)/surface(3D) in a scalar field where the value of the scalar function is constant across the domain
  - Scalar fields: pressure, temperature, etc.
  - 2D: isoline
  - 3D: Isosurface
- A technique for analyzing and visualizing scalar field data or scalar functions

What is an Isocontour?

- A contour is a curve(2D)/surface(3D) in a scalar field where the value of the scalar function is constant across the domain
  - Scalar fields: pressure, temperature, etc.
  - – 2D: isoline
  - – 3D: Isosurface
- – A technique for analyzing and visualizing scalar field data or scalar functions

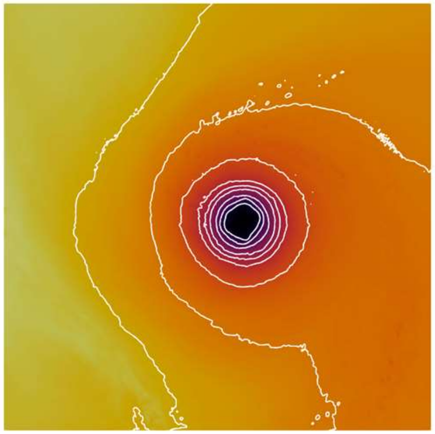
2D isocontour: Isoline

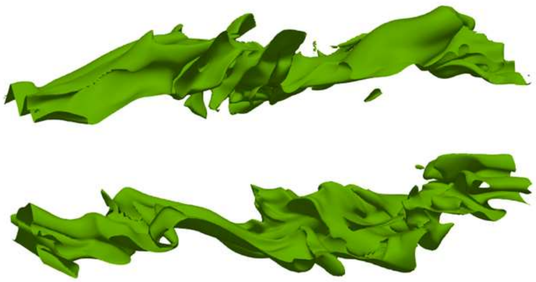


What is an Isocontour?

- A contour is a curve(2D)/surface(3D) in a scalar field where the value of the scalar function is constant across the domain
  - Scalar fields: pressure, temperature, etc.
  - – 2D: isoline
  - – 3D: Isosurface
- – A technique for analyzing and visualizing scalar field data or scalar functions

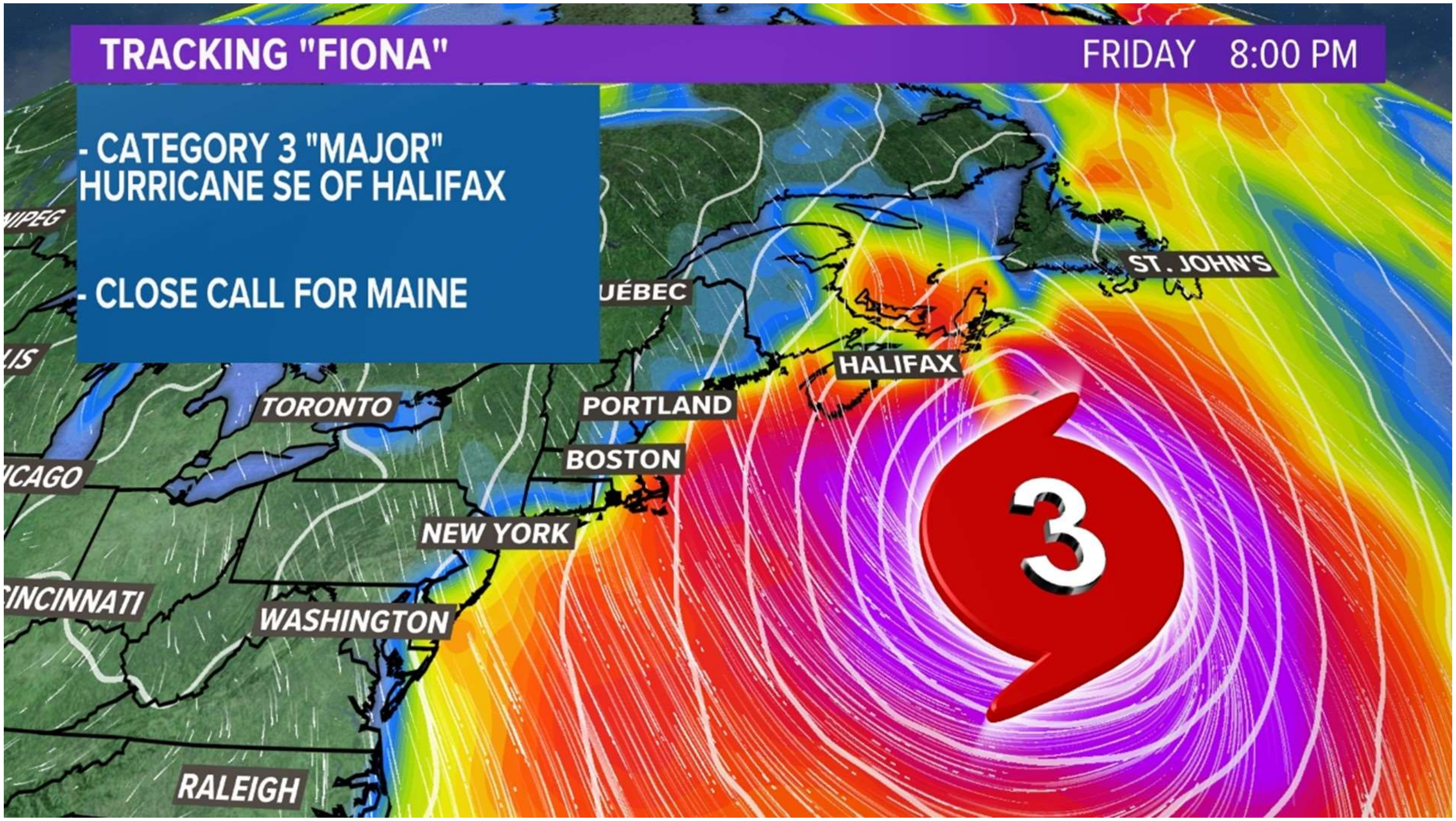
2D isocontour: Isoline





3D isocontour: Isosurface

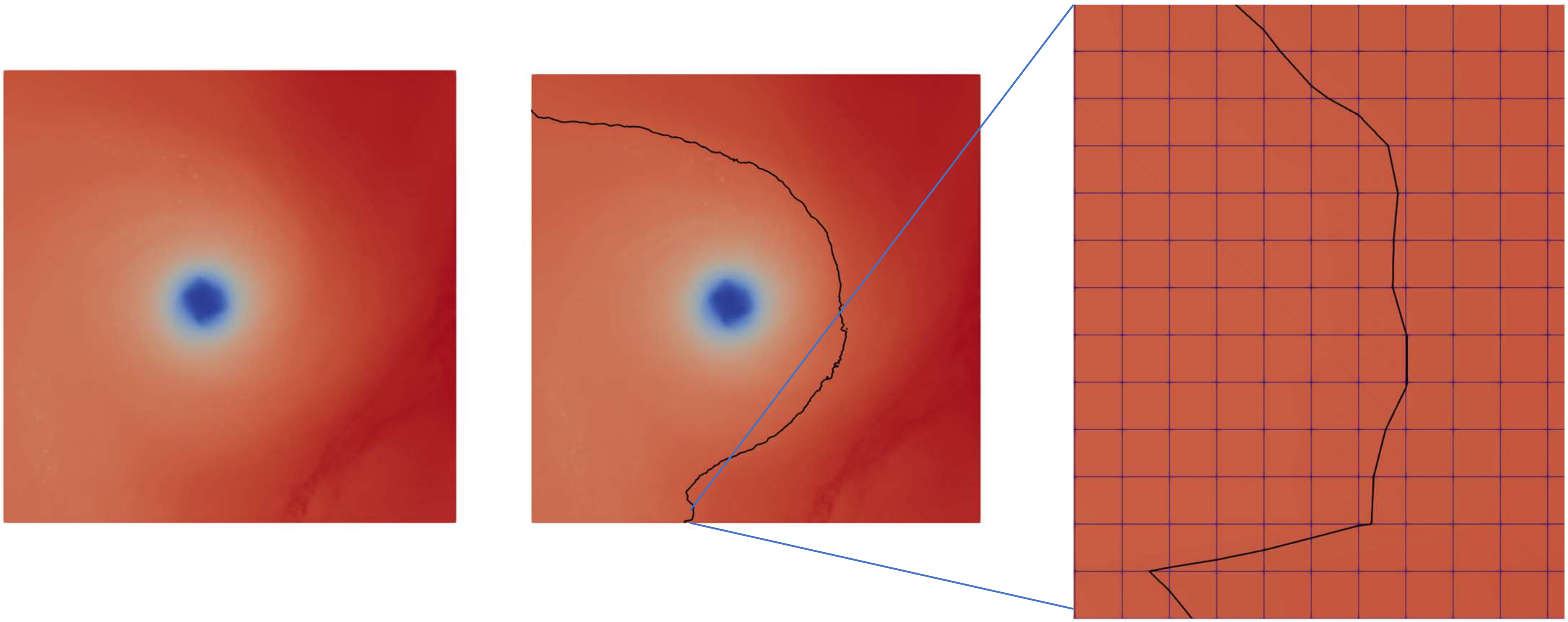
Isocontour: Isobar – Lines with Equal Pressure



Isocontour

Isosurface of bone and skin

Isocontour



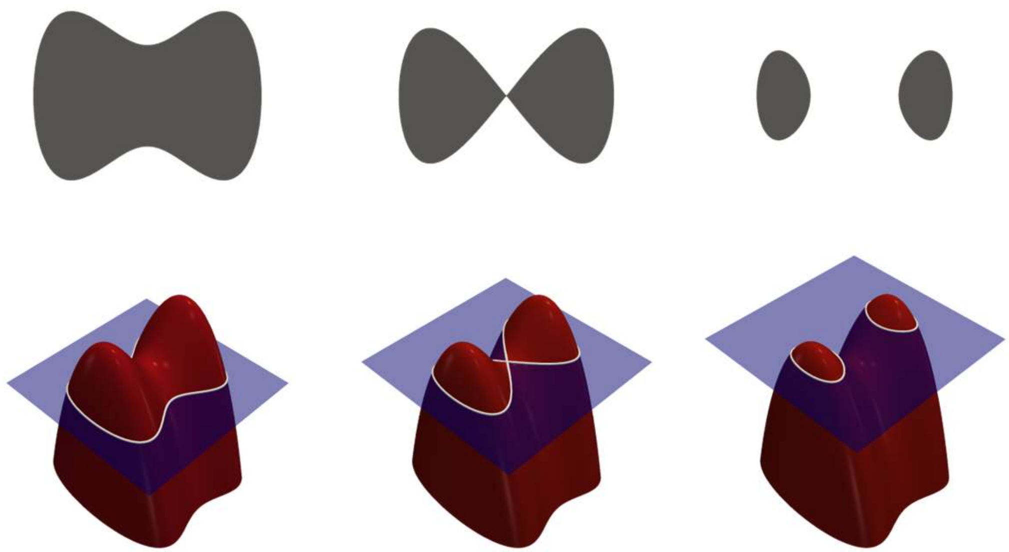
Isocontour (Isobar) at Pressure=250

Isocontour Demo with ParaView

Isocontour also Known As 'Level Set'

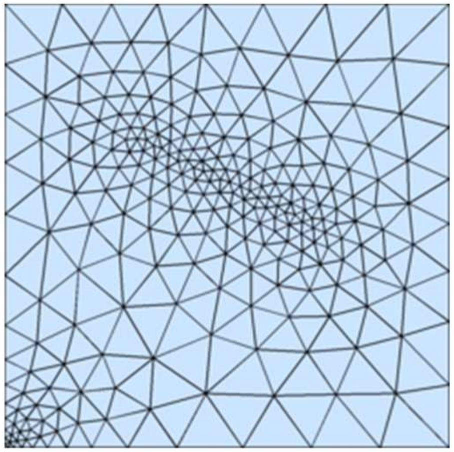
- Isocontour is also known as level set of a function in Mathematics
- A level set is defined as the set for real valued function of n variables, where the value of the function is a constant value

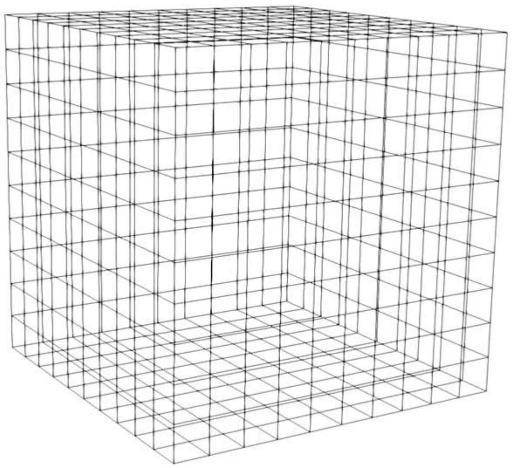
$$L_f(c) = \{x \mid f(x) = c\}$$



Scalar Data

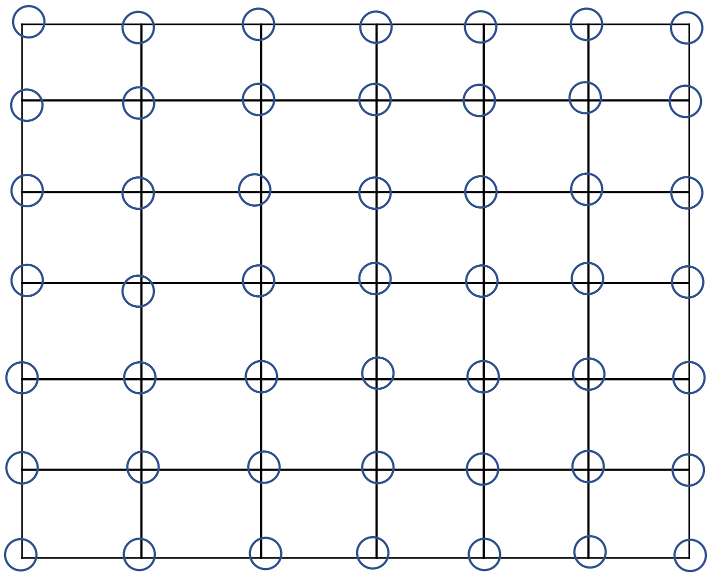
- Data is sampled from a continuous domain
- Discrete sampled domain is represented as a grid/mesh
  - Triangular mesh, cube mesh etc.
- The function value (scalar value) specified at mesh/grid vertices
- Values can be interpolated within the mesh to get value at a query location

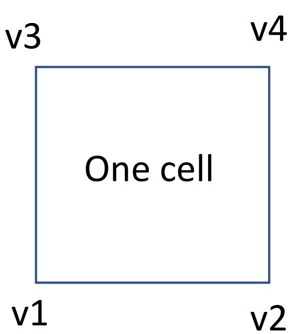




2D Isocontour Extraction

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

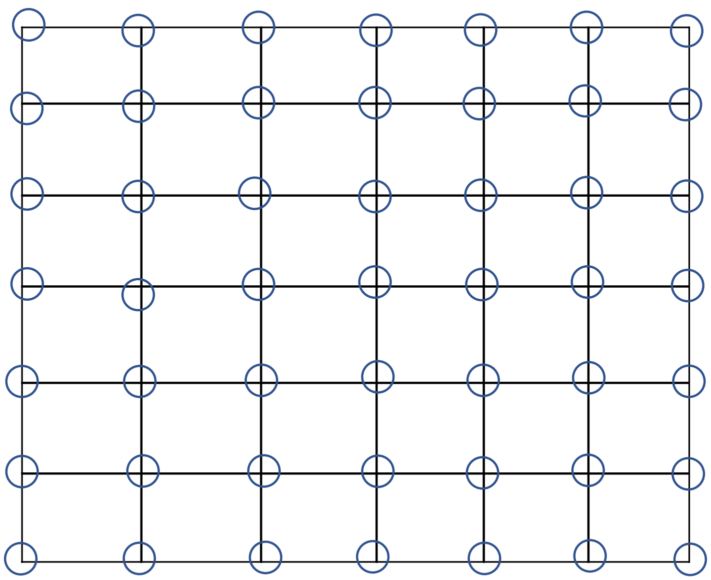


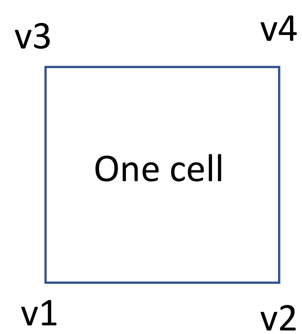


v1,v2,v3,v4 all are > C
No isoline in this cell

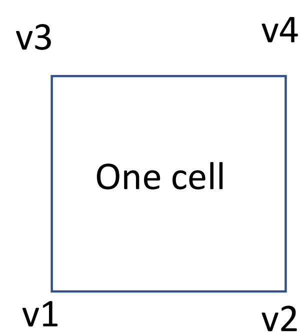
2D Isocontour Extraction

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





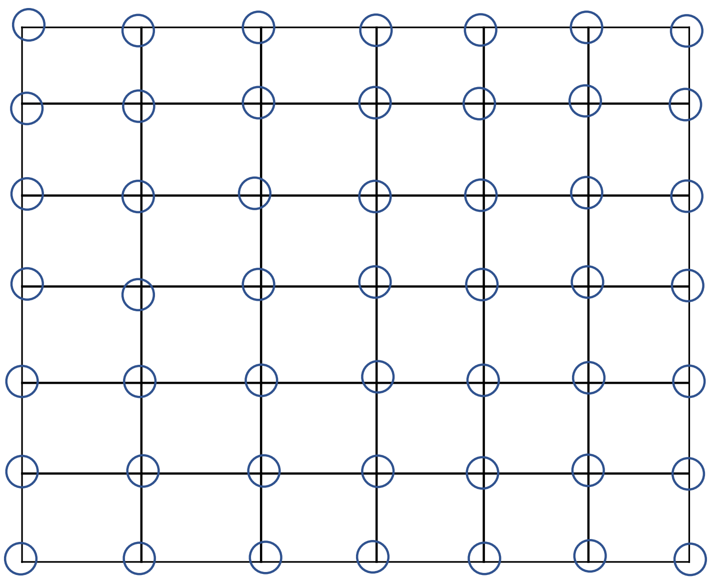
v1,v2,v3,v4 all are > C
No isoline in this cell

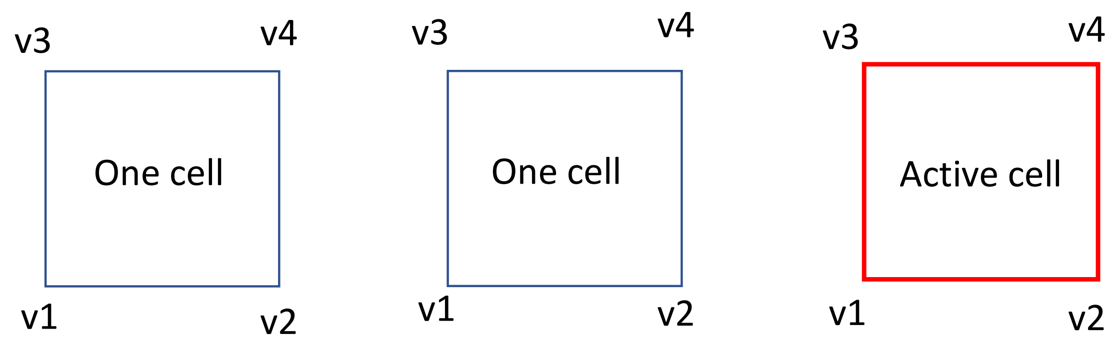


v1,v2,v3,v4 all are < C
No isoline in this cell

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

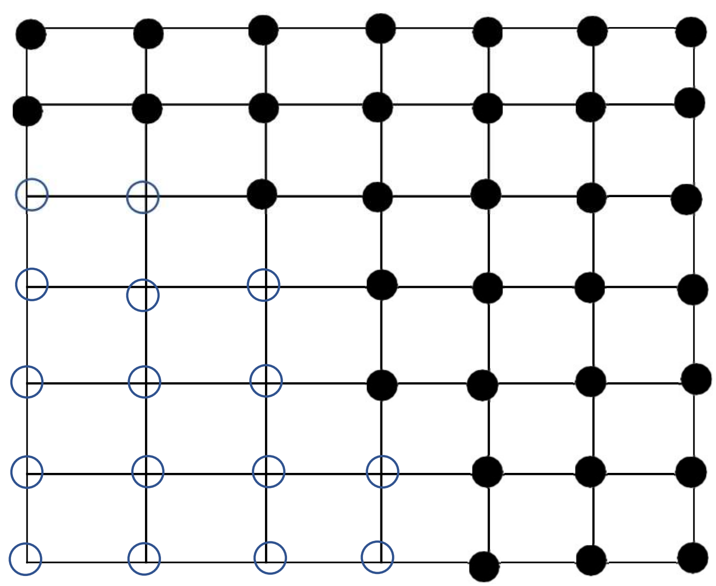
Otherwise, cell contains isocontour segment

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 **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**

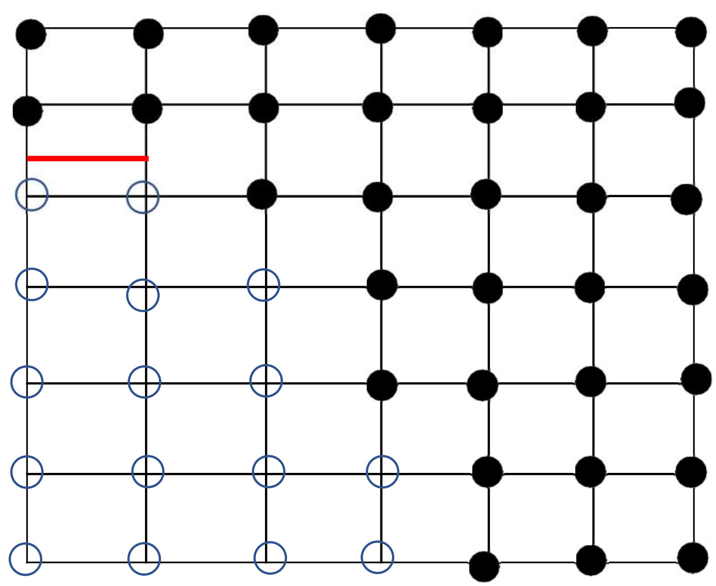


Contour value (isovalue) = C

- ● 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



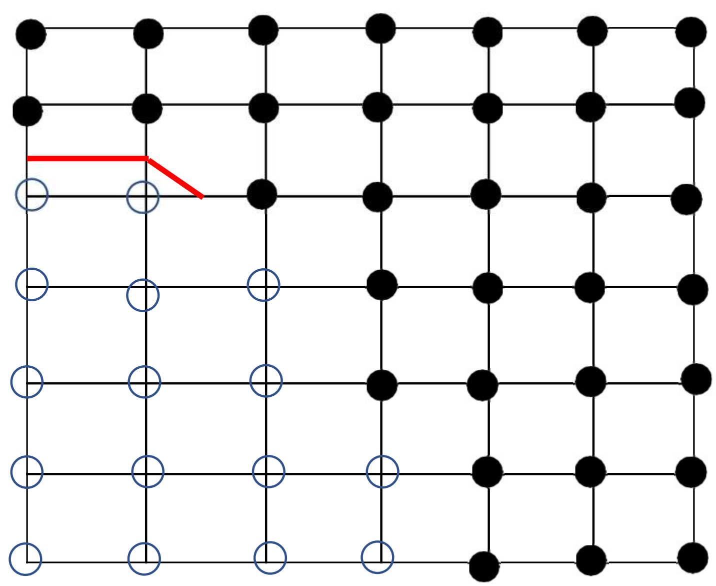
Contour value (isovalue) = C

● 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



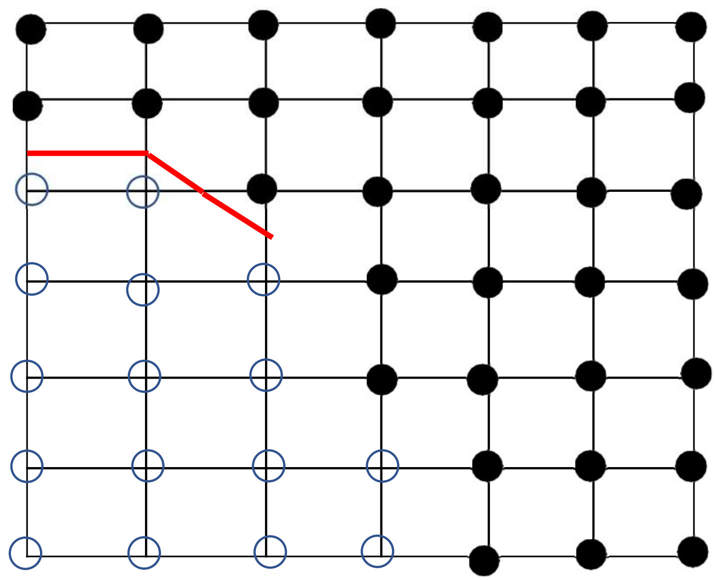
Contour value (isovalue) = C

- ● Value > C
- ○ Value < C

2D Isocontour Extraction: Marching Squares

IITK CS360A

- 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

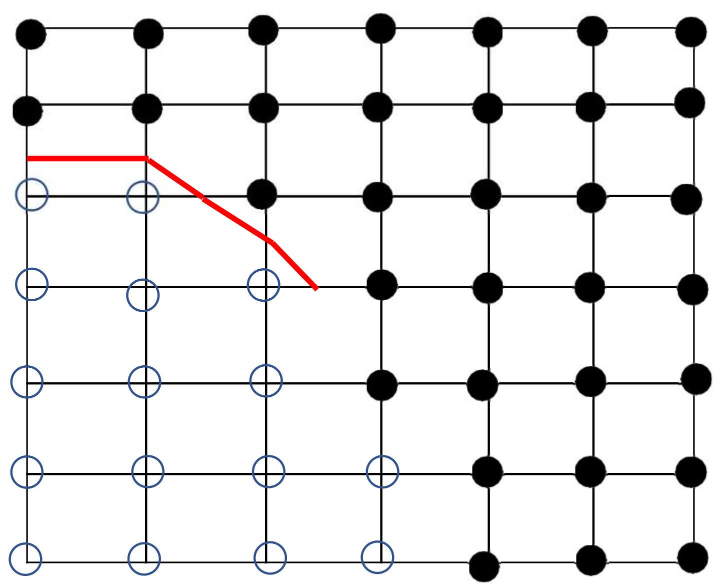
● Value > C

○ Value < C

2D Isocontour Extraction: Marching Squares

IITK CS360A

- 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

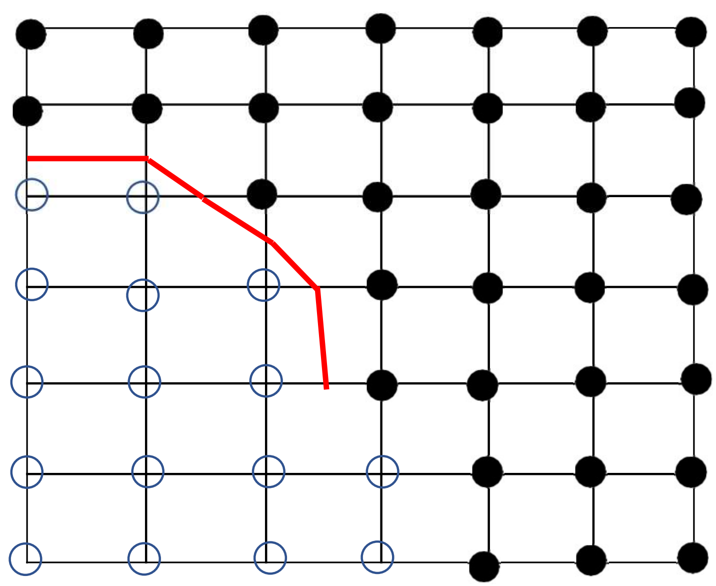


Contour value (isovalue) = C

- ● 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

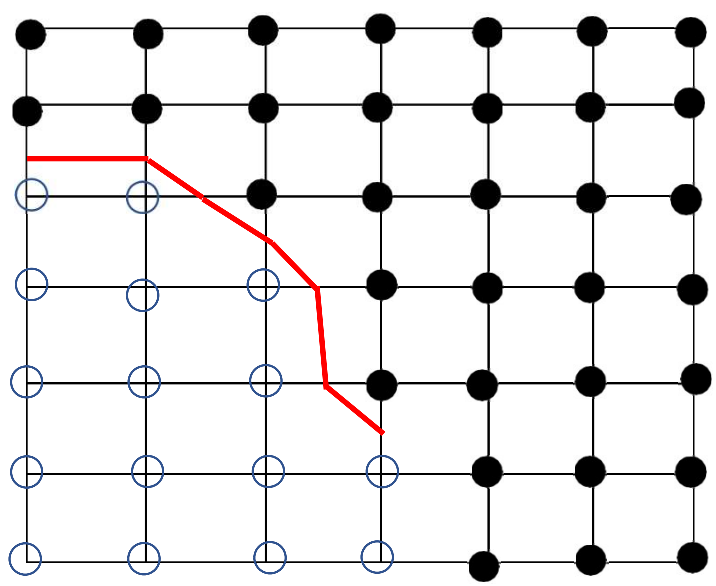


Contour value (isovalue) = C

- ● 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

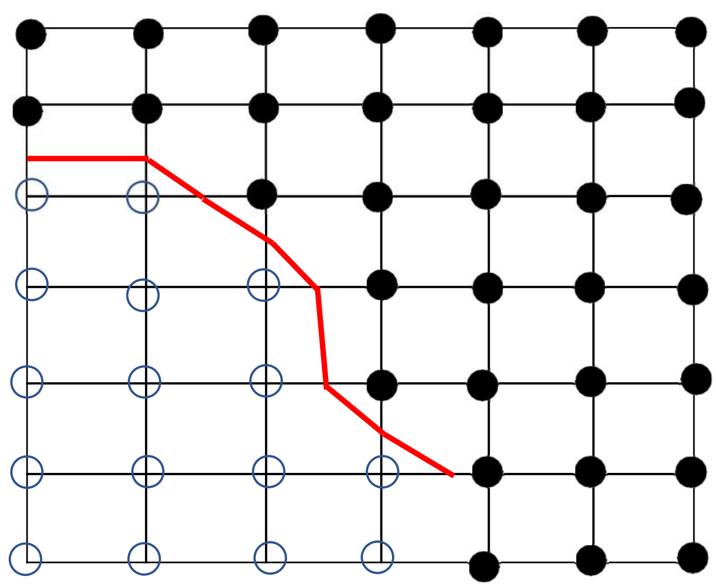


Contour value (isovalue) = C

- ● 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

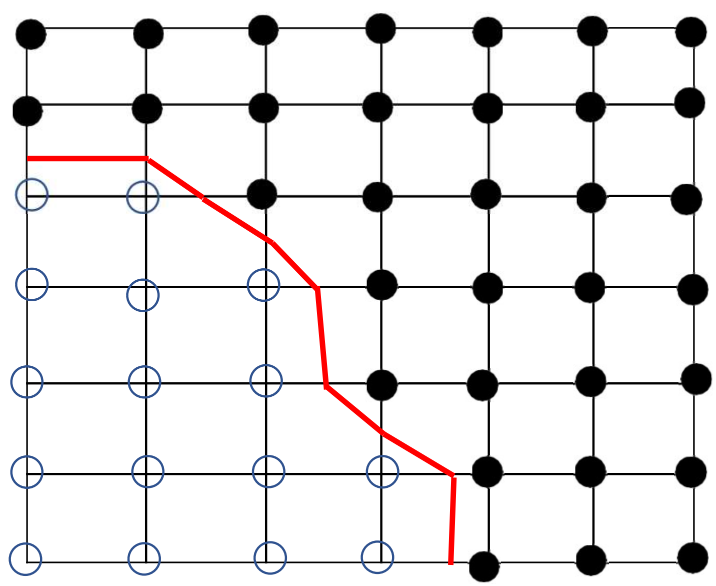
● Value > C

○ Value < C

2D Isocontour Extraction: Marching Squares

IITK CS360A

- 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

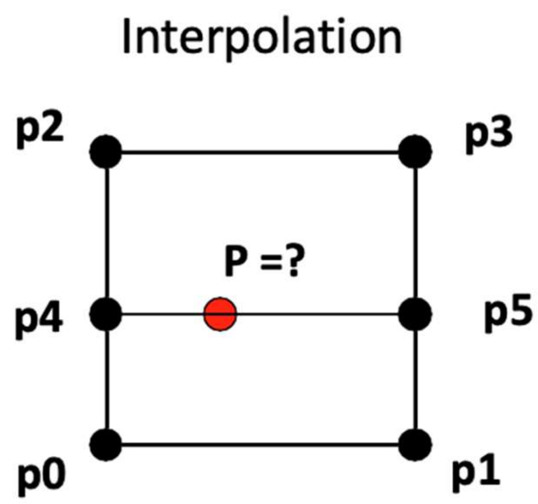


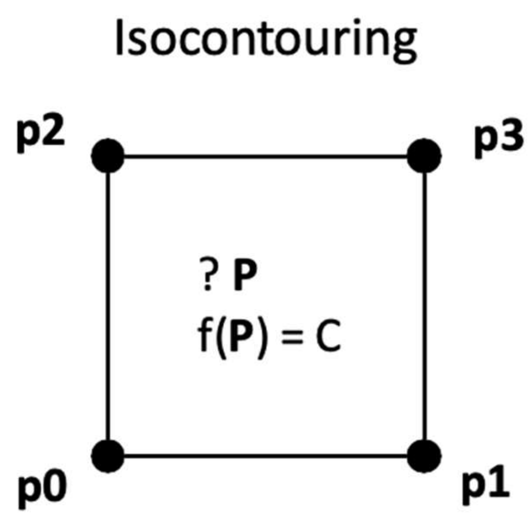
Contour value (isovalue) = C

- ● Value > C
- ○ Value < C

Isocontour in a 2D Cell

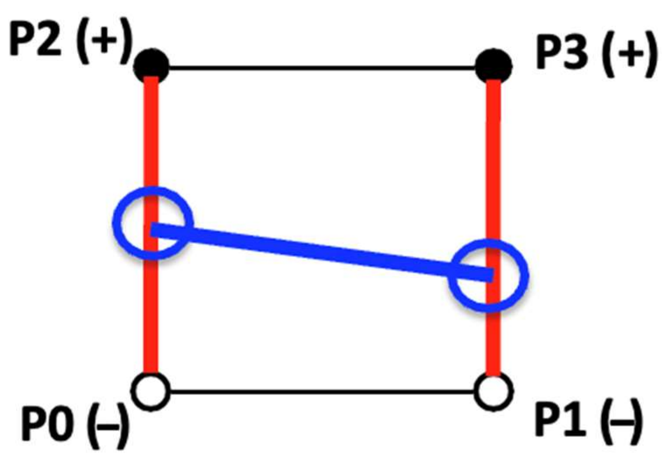
- Finding Isocontour in a cell is an inverse problem of value interpolation





Isocontouring by Linear Interpolation

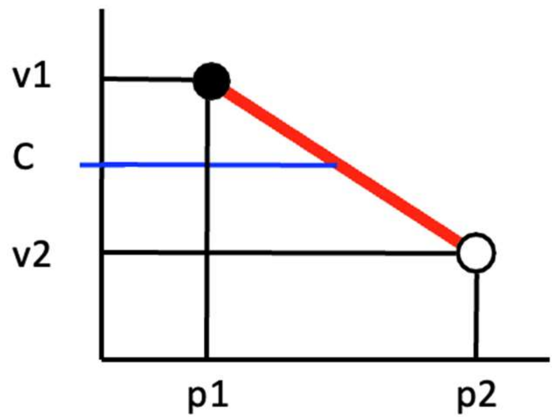
- Compute isocontour within a cell based on linear interpolation



- Identify edges that are ‘zero crossing’
  - Values at the two end points are greater (+) and smaller (–) than the contour value
- Calculate the positions of **P** in those edges
- 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

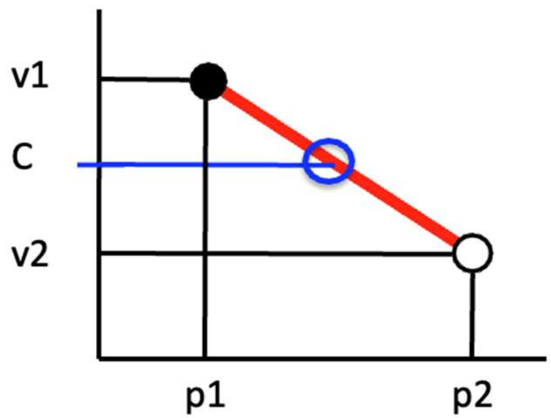


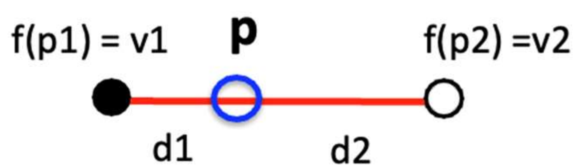
Step 2: Compute Intersection

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

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

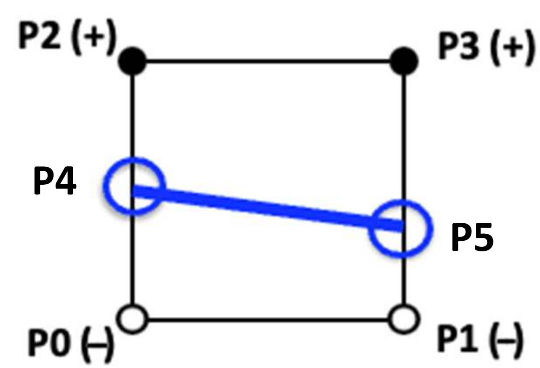
$\mathbf{p} = (v1-C)/(v1-v2) * (\mathbf{p}2 - \mathbf{p}1) + \mathbf{p}1$





Step 3: Connect the Dots

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

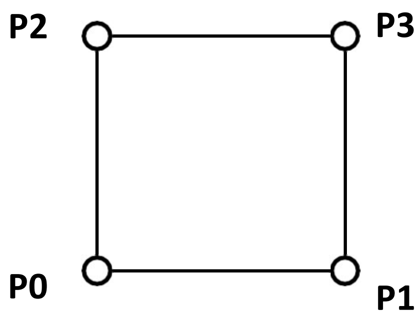


Repeat Step1 – Step 3 for all cells

Isocontour Cases

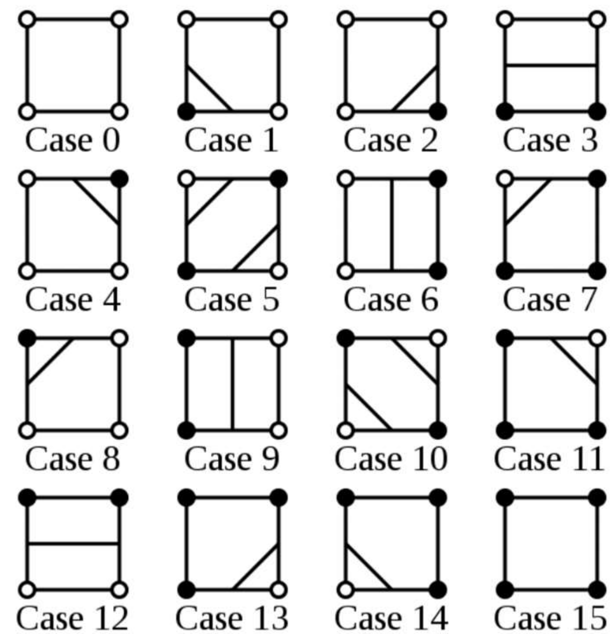
IITK CS360A

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



P4	P3	P2	P1

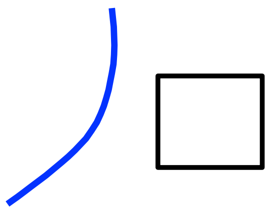
1: > C
0: < C



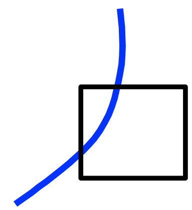
- The value at each vertex can be either greater or less than the contour value
- So, there are 2 x 2 x 2 x 2 = 16 cases

Unique Topological Cases

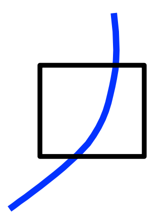
- There are only four unique topological cases



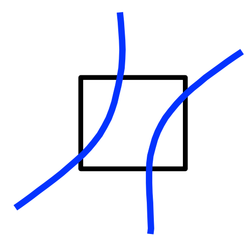
(1) No intersection



(2) Intersect with two adjacent edges



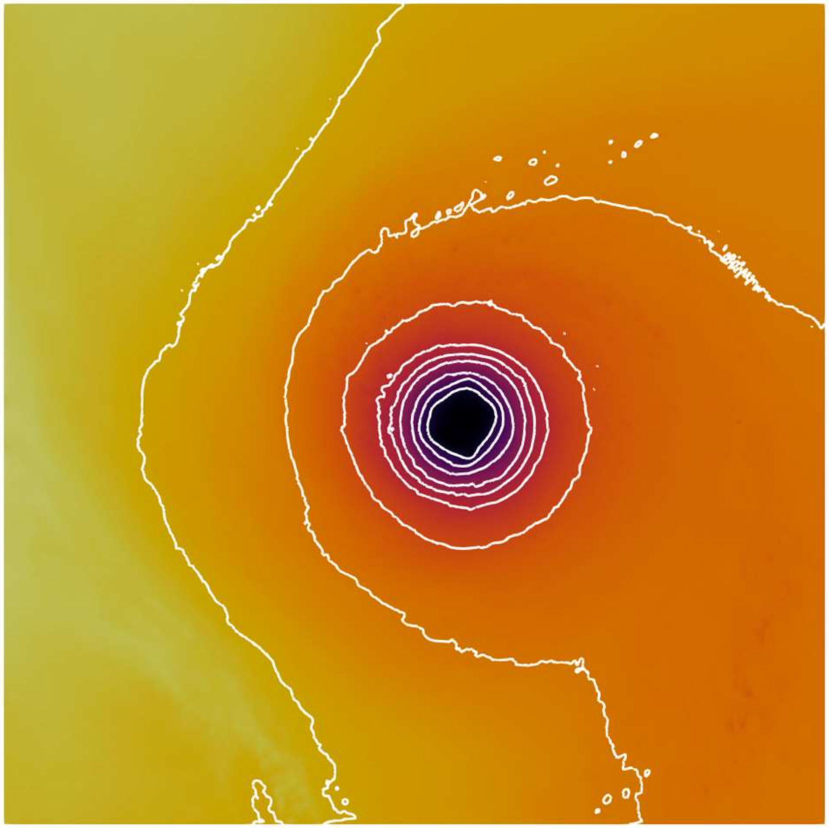
(3) Intersect with two opposite cases



(4) Two contours pass through the cell

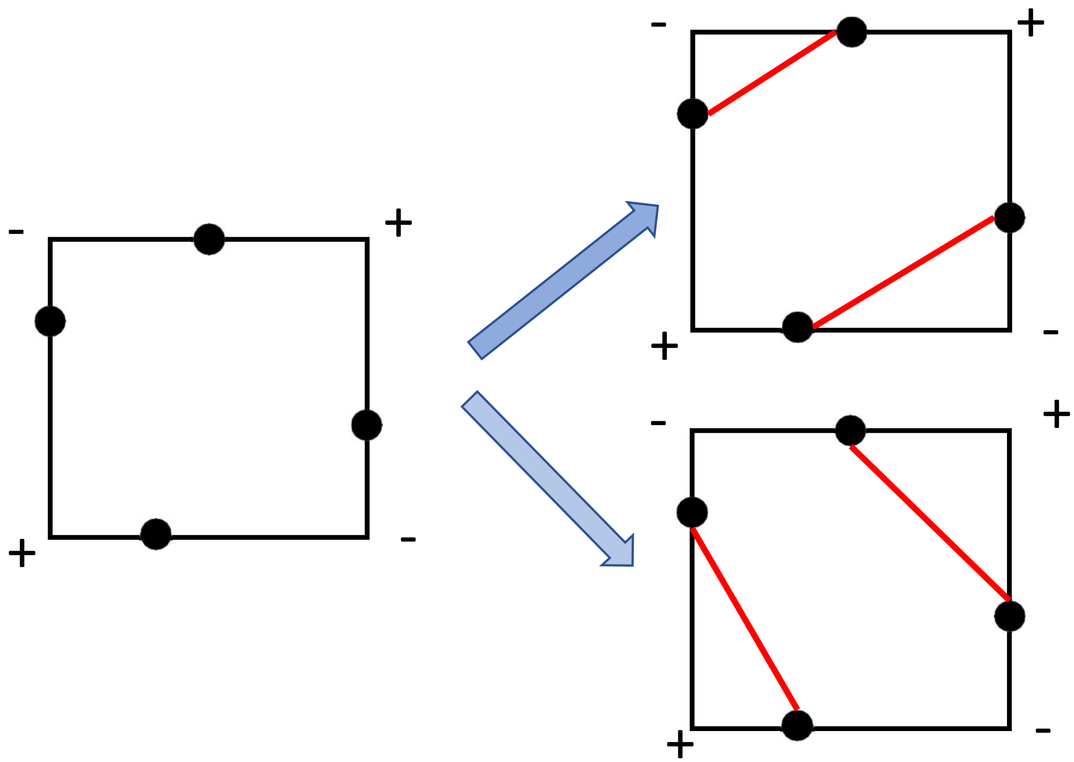
Putting it All Together

- 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

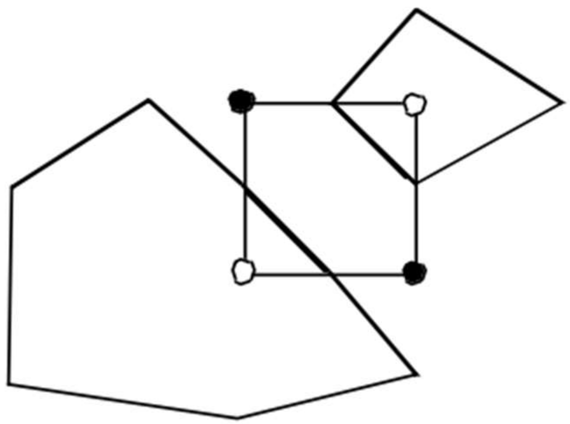


How to connect?

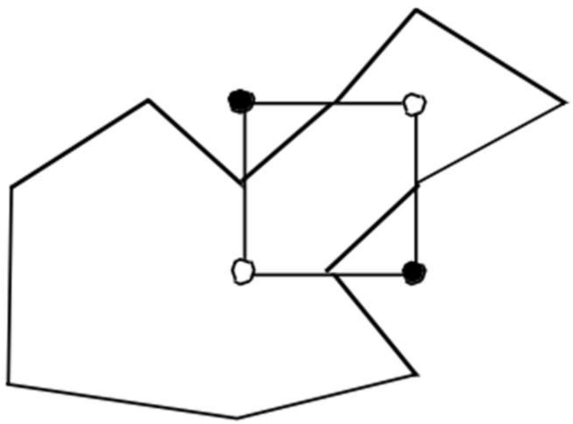
Both configurations are possible!

Dealing with Ambiguous Cases

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



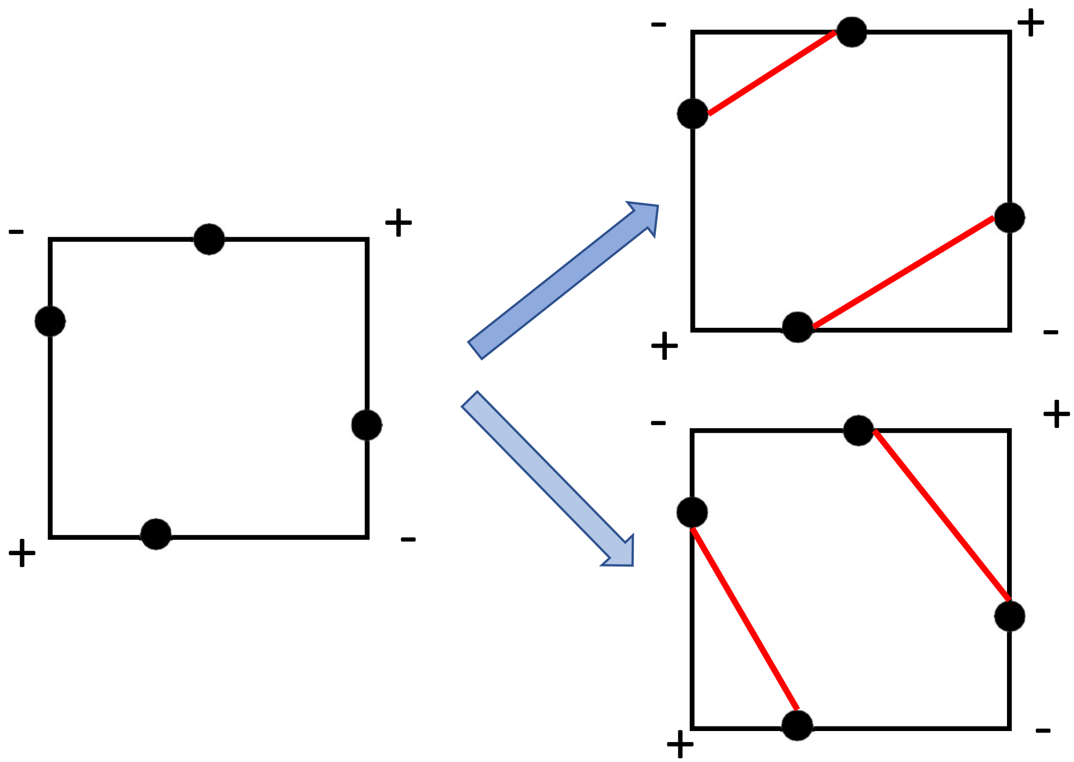
Broken Contour



Connected Contour

Dealing with Ambiguous Cases

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



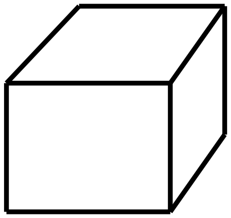
How to connect?
Both configurations are possible!

- One way to resolve: Use Asymptotic decider

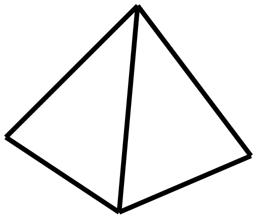
The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes by Nielson and Hamman, IEEE VIS'91

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



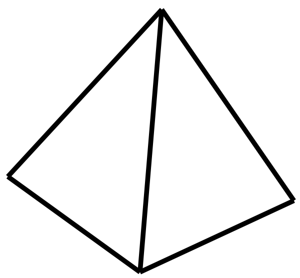
Cube/Rectangular cell



Tetrahedron cell

Tetrahedral Cell

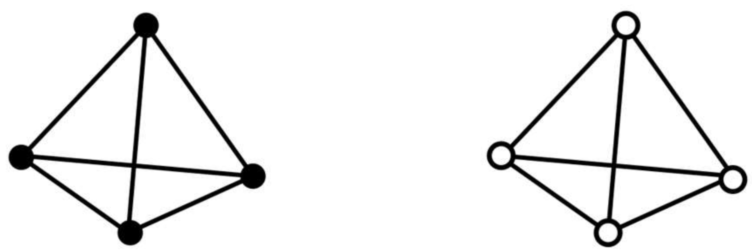
- Active cells: min value < C < max value
- Mark cell vertices that are greater than C with "+" and smaller than C with "-"
- Each cell has 4 vertices
  - Each vertex can have value greater or less than C
  - Hence, 2x2x2x2 = 16 possible combinations
  - Only <u>three unique topological cases</u>



Tetrahedron cell

Tetrahedral Cell: Case 1

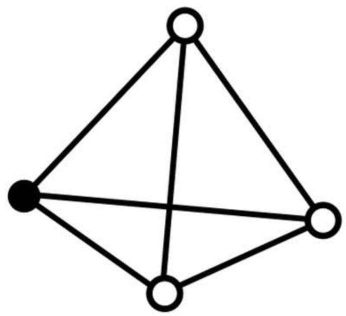
- Case 1: No intersection (all vertices are either outside or inside)
- Values at all cell vertices are either larger or smaller than the isovalue C
  - If we assume that cell values greater than the contour value C as ‘outside’ and smaller as ‘inside’, then all cell vertices are either completely inside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



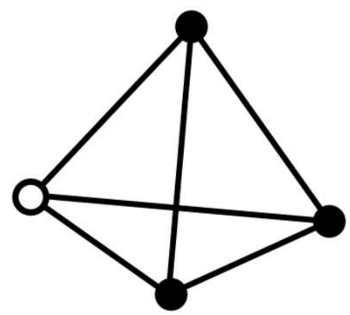
All Vertices Outside   All Vertices Inside

Tetrahedral Cell: Case 2

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



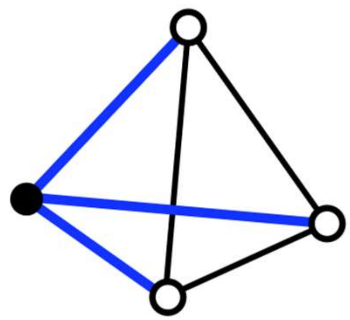
One Outside



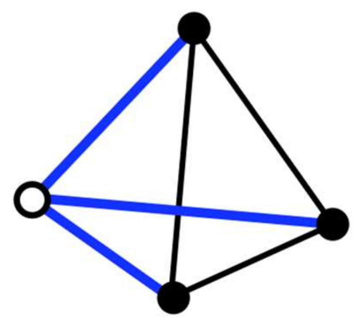
One Inside

Tetrahedral Cell: Case 2

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



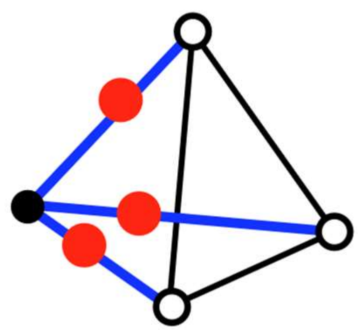
One Outside



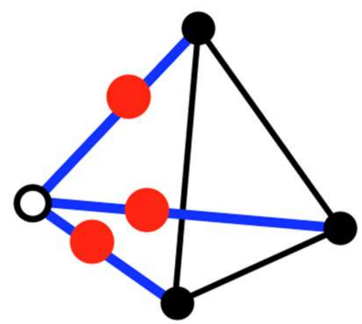
One Inside

Tetrahedral Cell: Case 2

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



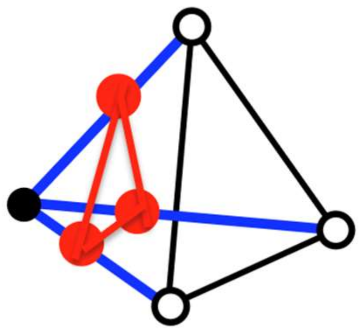
One Outside



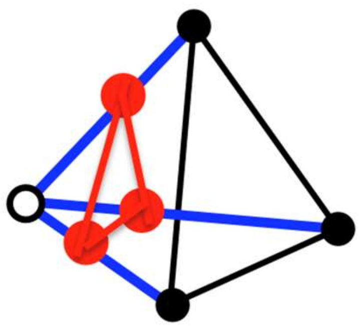
One Inside

Tetrahedral Cell: Case 2

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



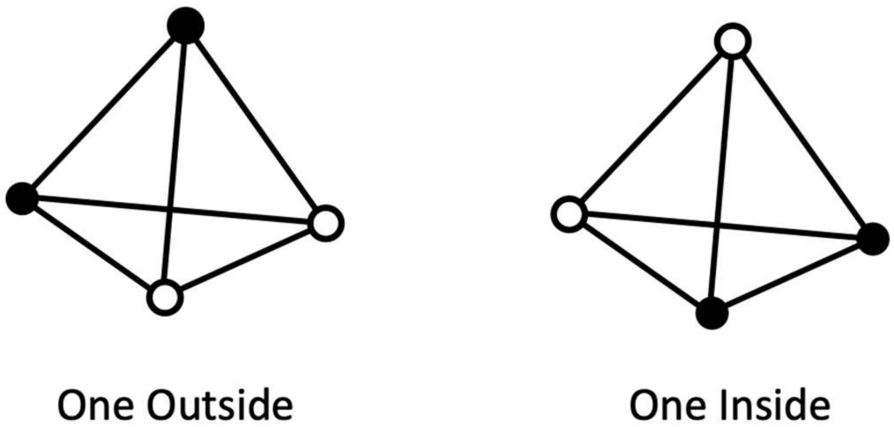
One Outside



One Inside

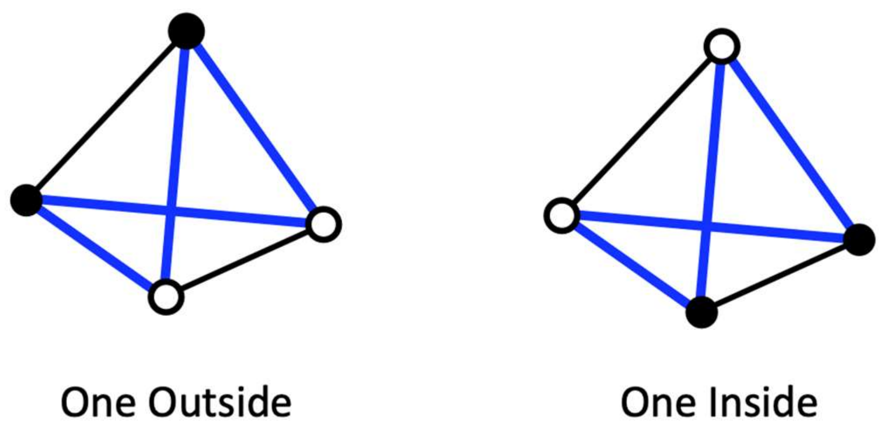
Tetrahedral Cell: Case 3

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



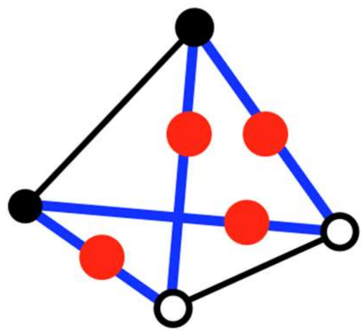
Tetrahedral Cell: Case 3

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

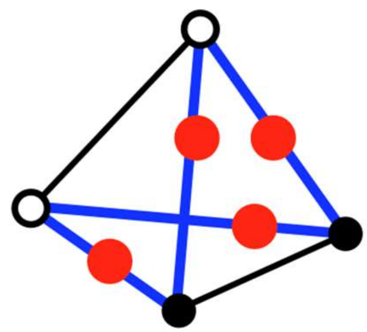


Tetrahedral Cell: Case 3

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



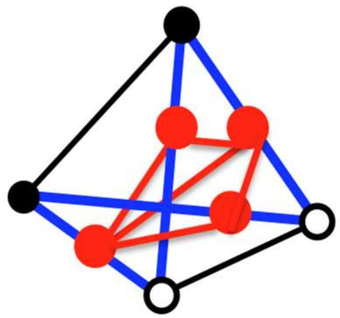
One Outside



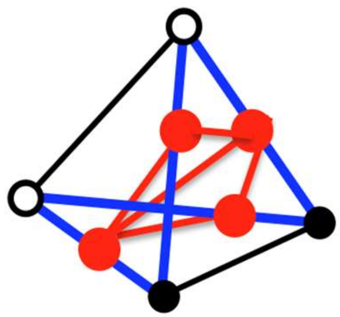
One Inside

Tetrahedral Cell: Case 3

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

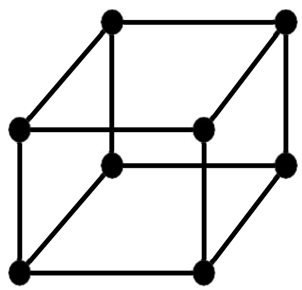


One Outside



One Inside

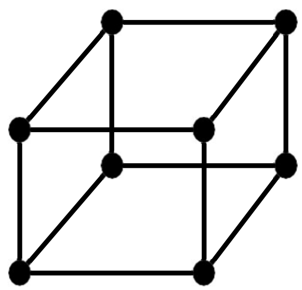
3D Isocontour: Cube/Rectangular Cells



Cube/Rectangular cell

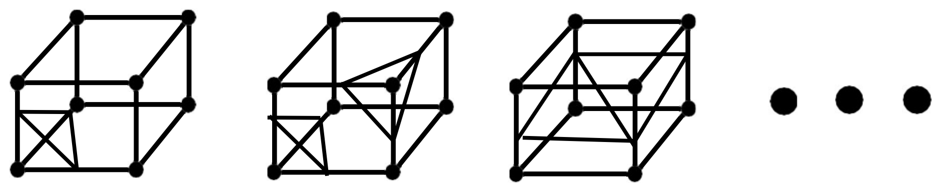
- With 8 vertices in a cell, each having a value greater or smaller than the contour value, there can be $2^8 = 256$ possible cases

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

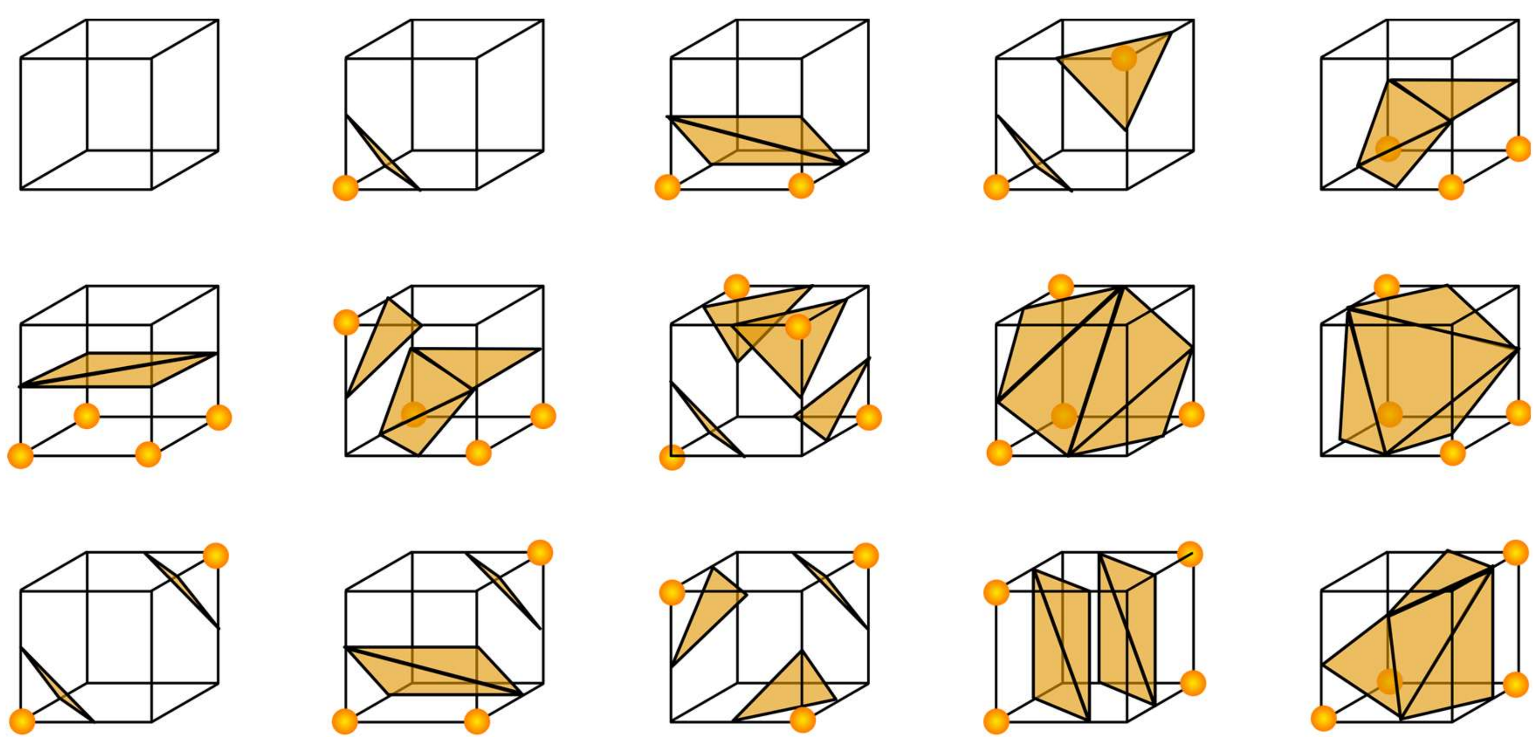
Case Reduction

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

Value Symmetry

3D Isosurface Unique Cases

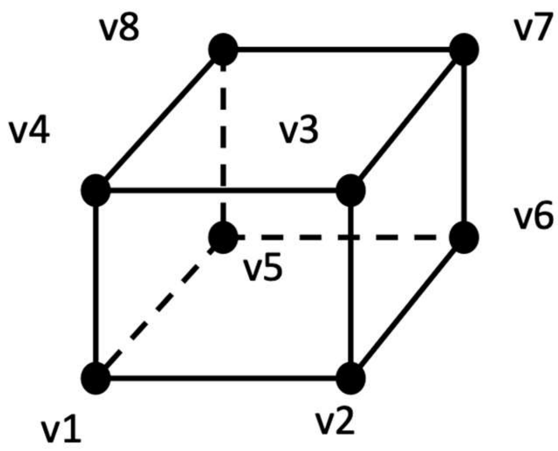
- 15 Topologically Unique Cases



Marching Cubes Algorithm

IITK CS360A

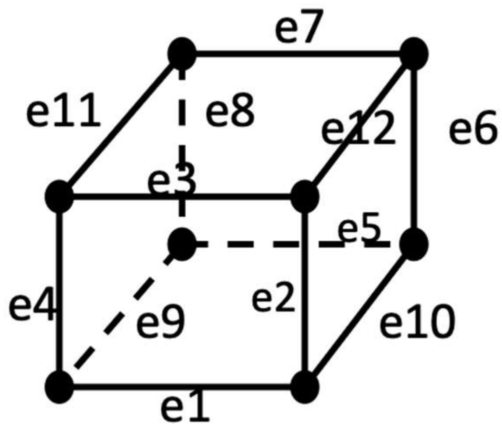
- Proposed by Lorensen and Cline in 1987
- Mark each cell with a bit
  - V_i is 1 if value > C (C=isovalue)
  - V_i is 0 if value < C
- Each cell has an index mapped to a value ranged [0,255]



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

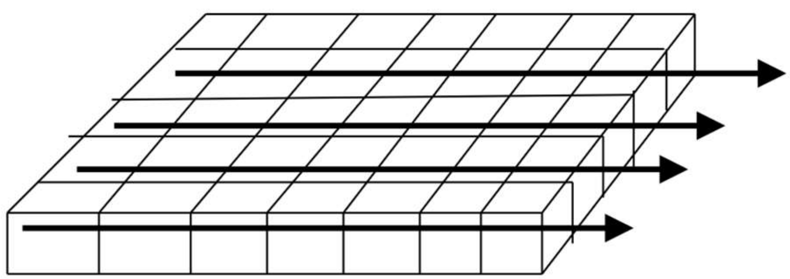


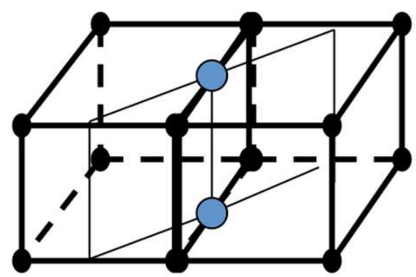
Index =	v8	v7	v6	v5	v4	v3	v2	v1

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

Marching Cubes Algorithm

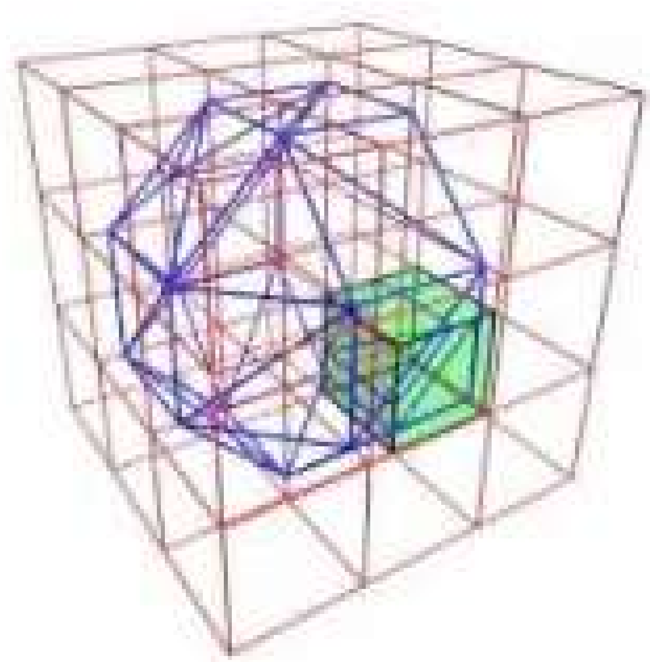
- Perform linear interpolation to compute the intersection points at the edges
- Connect the points to form surface patches
- Sequentially scan through the cells – row by row, layer by layer
- Re-use the intersection points for neighboring cells





Marching Cubes Algorithm: Animation

Implementation



References

- References:
  - Multidimensional Transfer Functions for Interactive Volume Rendering, TVCG 2002
  - Visibility-Driven Transfer Functions, IEEE PacificVis
  - State of the Art in Transfer Functions for Direct Volume Rendering, Ljung et al., EuroVis 2016
  - 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.
  - Resolving the Ambiguity in Marching Cubes, by Nielson and Hamman, IEEE VIS’91.