Topics in Large Data Analysis and Visualization (CS677)

Soumya Dutta, Preeti Malakar

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur (IITK)

email: {soumyad, pmalakar}@cse.iitk.ac.in

Acknowledgements

- Some of the slides are adapted from the excellent course materials made available by: Prof. Klaus Mueller, Prof. Tamara Munzner, and Prof. Han-Wei Shen

Scientific Visualization (SciVis)

- **Scientific visualization:** The use of computers or techniques for comprehending data or to extract knowledge from the results of simulations, computations, or measurements
- Scientific visualization is the formal name given to the field in computer science that encompasses
  - data representation and processing algorithms
  - visual representations
  - user interface
- Relatively (new) domain of research
  - ~ 36 years
  - Formal inception in 1987 by US NSF as "**Visualization in Scientific Computing**" by McCormick et al.

Scientific Visualization (SciVis)

- Scientific data has an implied spatial layout
- With scientific visualization
  - Techniques can exploit the spatial properties of the information (**meshes, grids, vectors, tensors**, etc.)
  - Utilize the three-dimensional capabilities of today's graphics engines to help visually present their analysis

Data Representation & Scientific Data Model

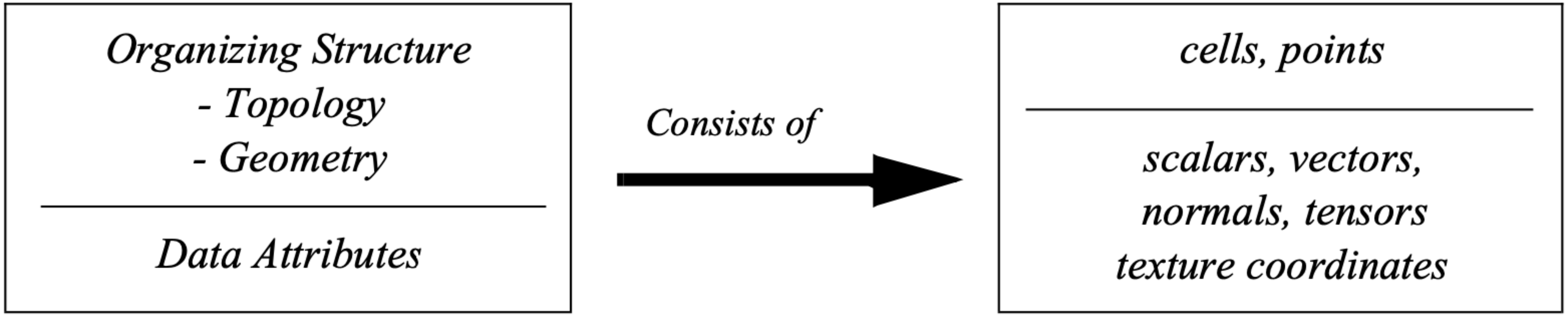
Data Representation

- Scientific data is Continuous in nature
  - Measure physical quantities that are studied by various scientific and engineering disciplines, such as physics, chemistry, mechanics, or engineering
  - Examples: pressure, temperature, velocity, density, etc.

- In practice, such data is sampled in a discrete form in computers
  - Representation
  - Manipulation
  - Analysis
  - Visualization

Scientific Dataset

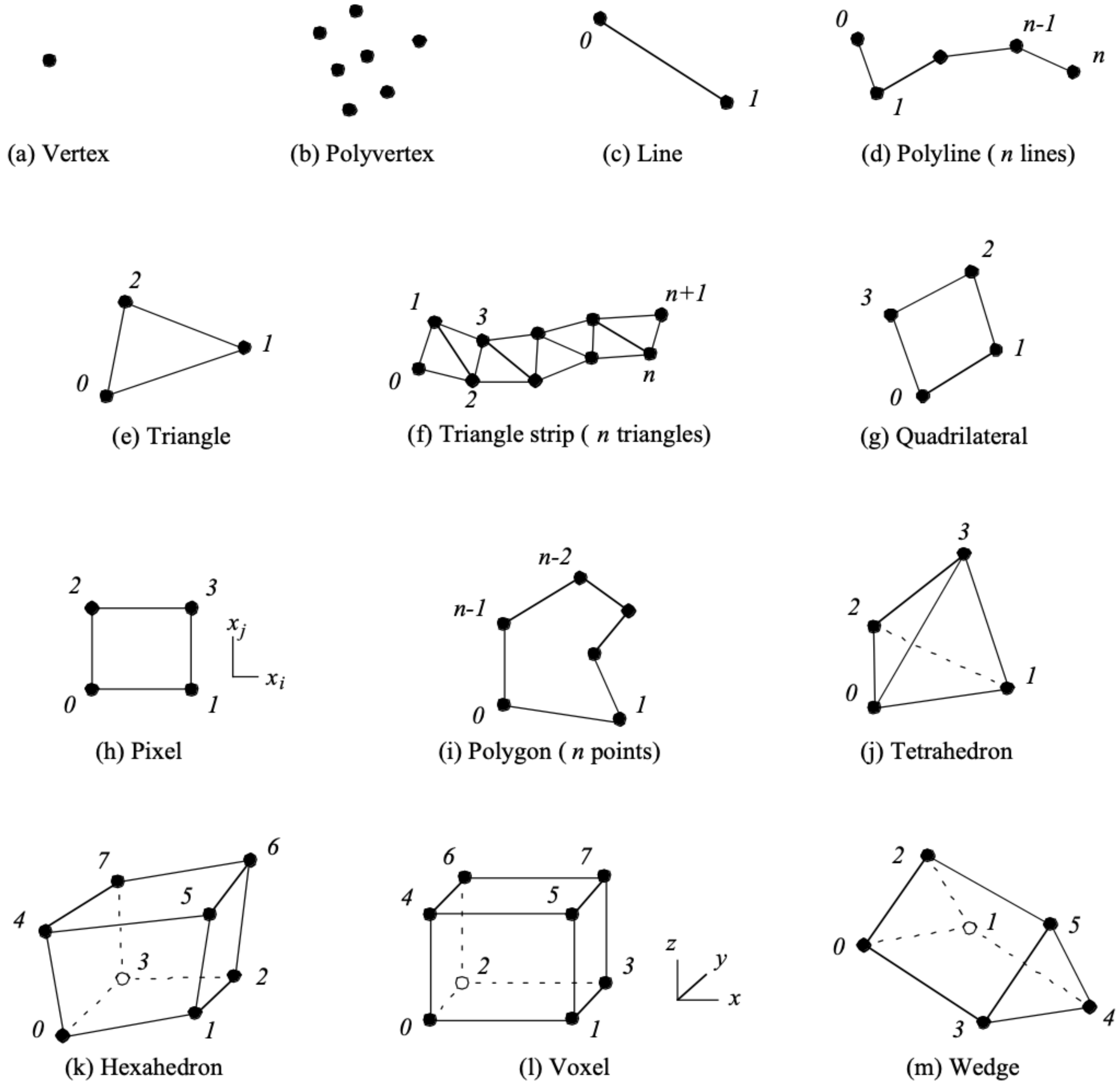
- A scientific dataset is a related collection of data
  - With an attached spatial context (e.g., a grid)
  - Captures all relevant characteristics of a data collection
- Discrete scientific dataset:



Scientific Dataset

- Typically, a discrete dataset is sampled from a continuous domain into a grid/mesh structure
  - A grid is a set of cells, which are defined by a set of sample points, referred to as the **Geometry information**
  - The connectivity between the points are referred to as the **Topology**
  - Each (grid) point can have multiple attached **Attributes**
- Attributes can be of various types
  - Scalar, Vector, Tensor etc.
- By using the grid structure and using a reconstruction technique, data values at any point can be computed which gives us a way to <u>approximately</u> reconstruct the continuous data
  - A typical reconstruction method is <u>**interpolation**</u>

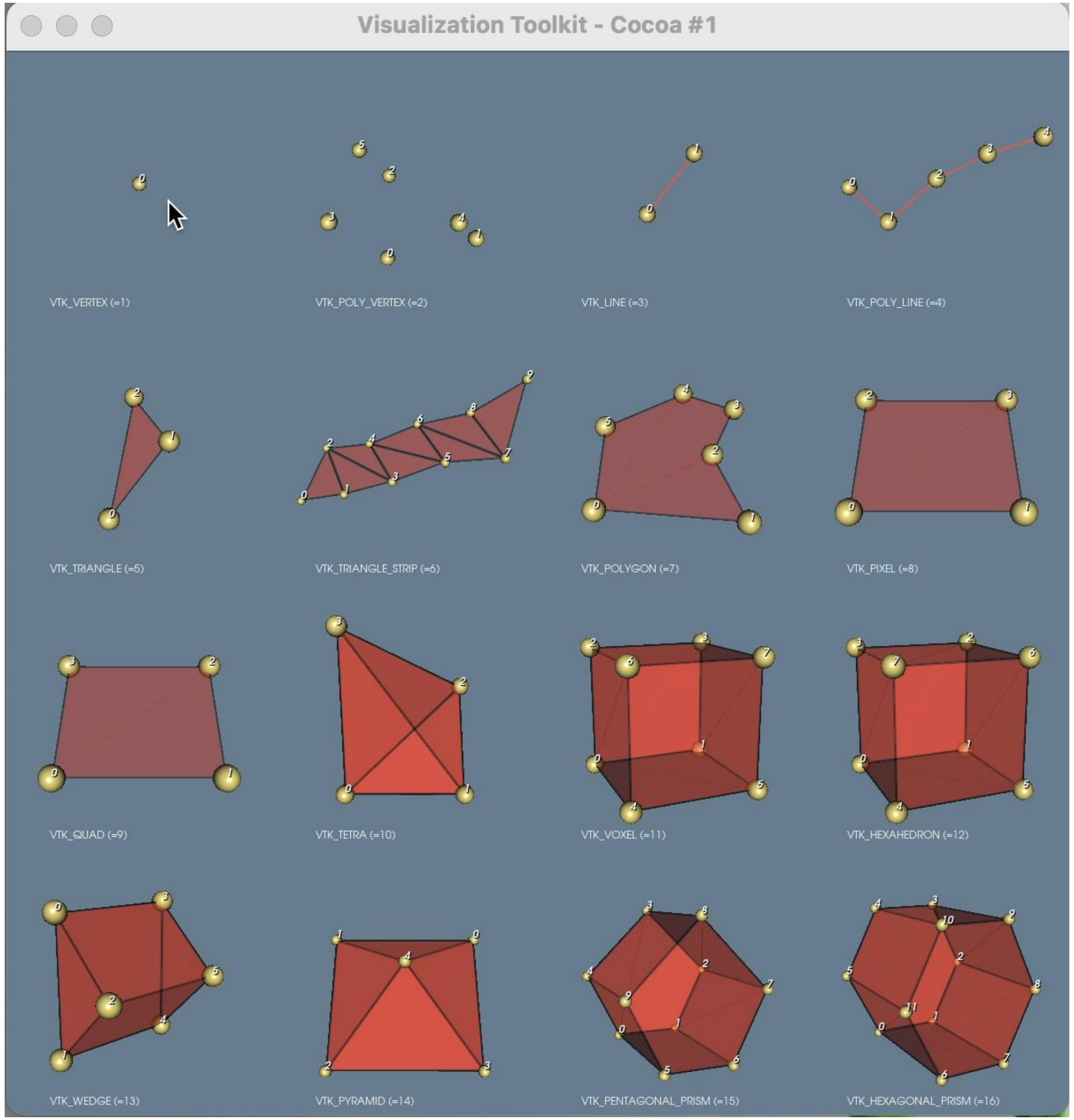
Scientific Dataset: Various Cell Types



Cell Types

- **Code can be found:** https://kitware.github.io/vtk-examples/site/Python/GeometricObjects/LinearCellDemo/

- Written using a Visualization library called **Visualization Toolkit (VTK)**
  - **https://vtk.org/**
  - You can use VTK in your assignment

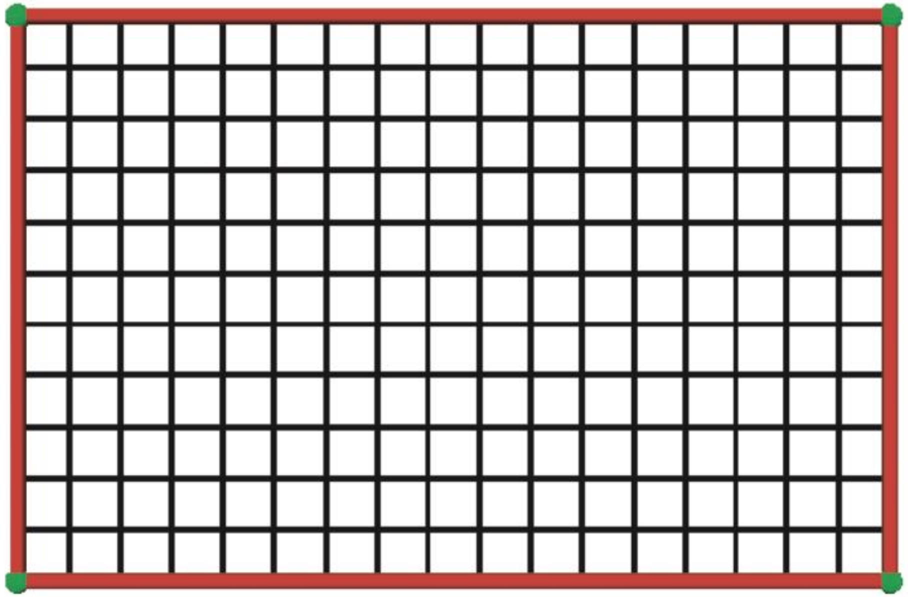


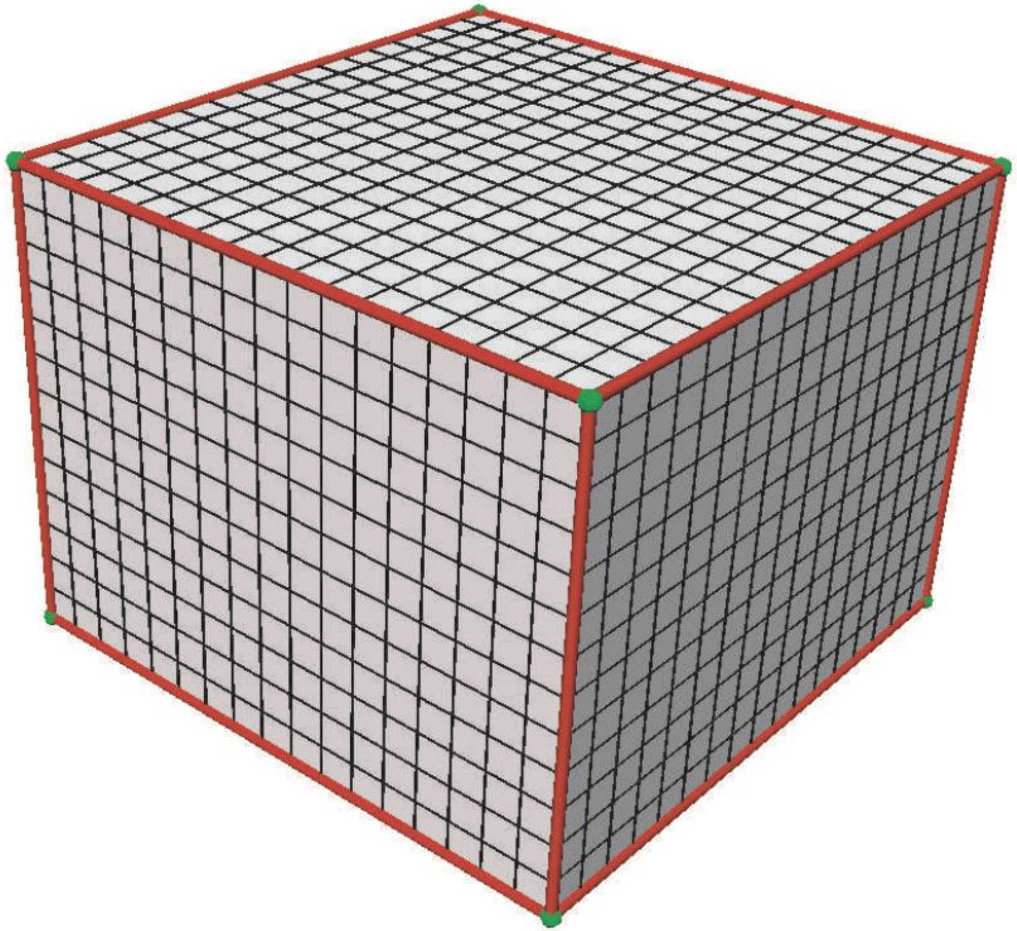
Scientific Dataset: Grid Types

- Uniform Grid
- Rectilinear Grid
- Structured Grid
- Unstructured Grid

Scientific Dataset: Uniform Grid

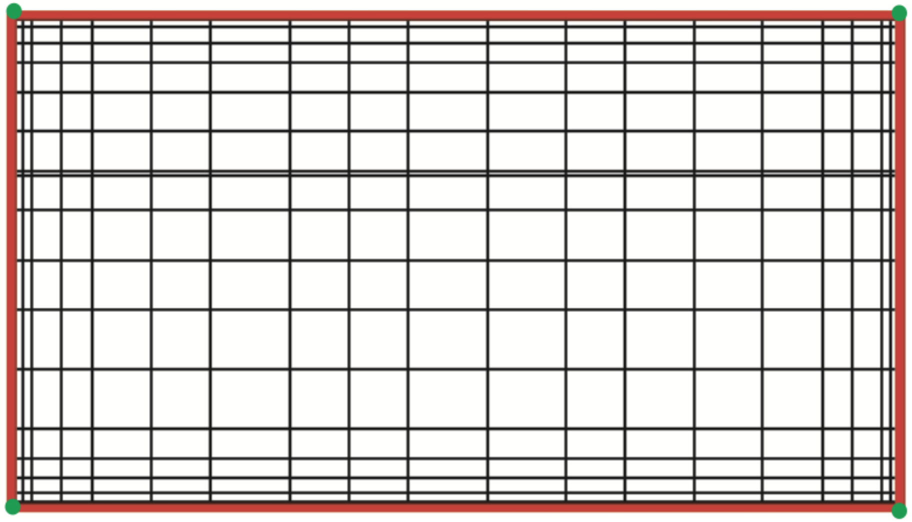
- Axis aligned box
- Sample points are equally spaced

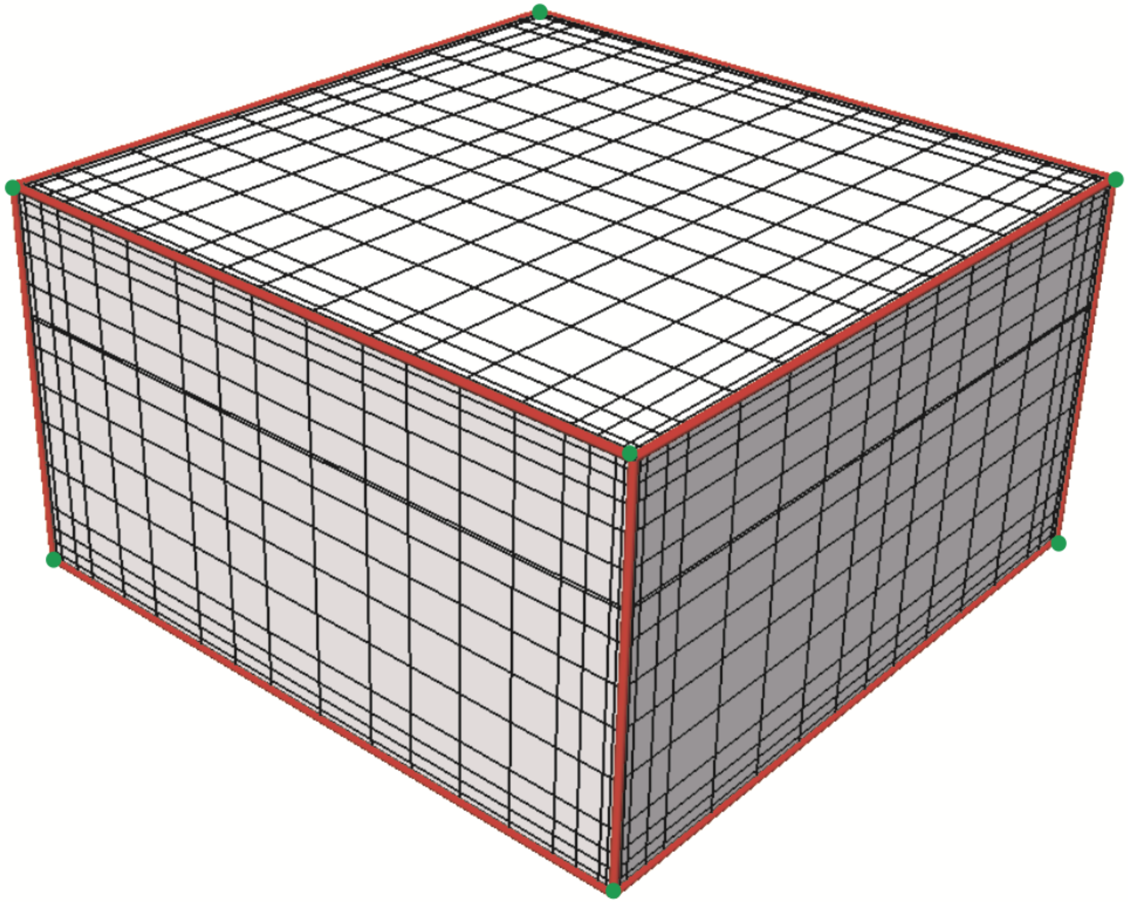




Scientific Dataset: Rectilinear Grid

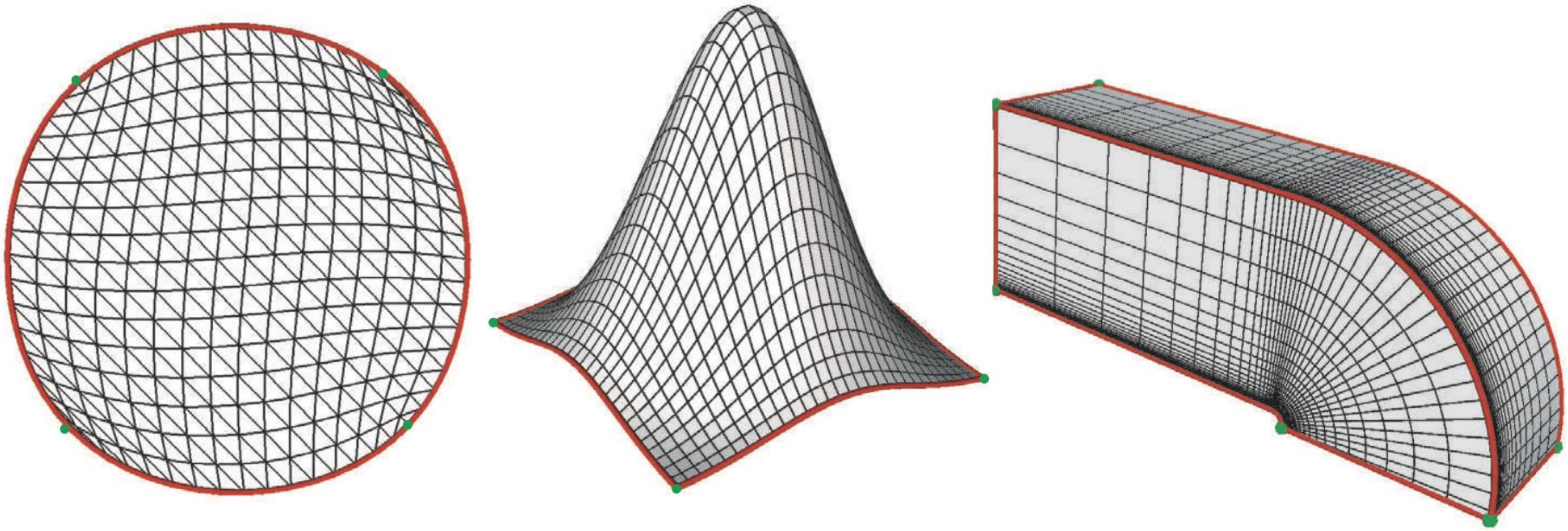
- Axis aligned box
- Sample points are nonequally spaced





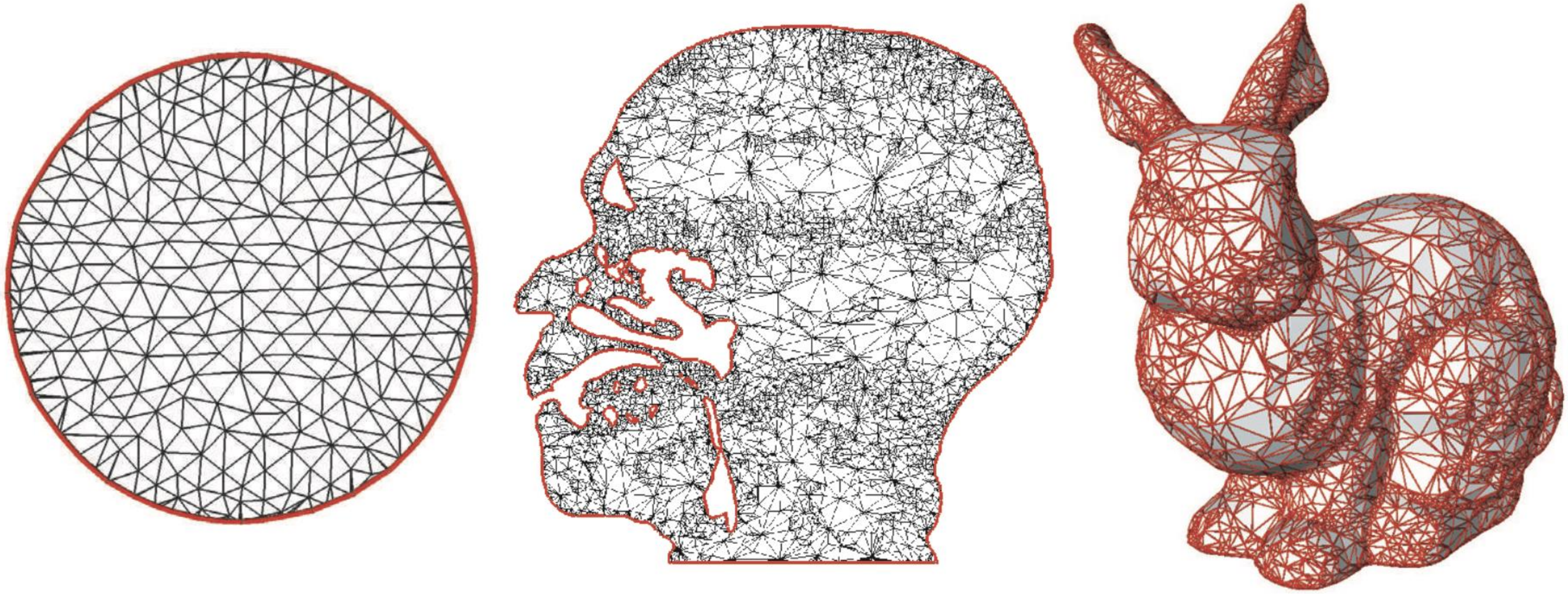
Scientific Dataset: Structured Grid

- Allow explicit placement of every sample point
- Yet preserve the matrix-like ordering
- Structured grids can be seen as a deformation of uniform grids



Scientific Dataset: Unstructured Grid

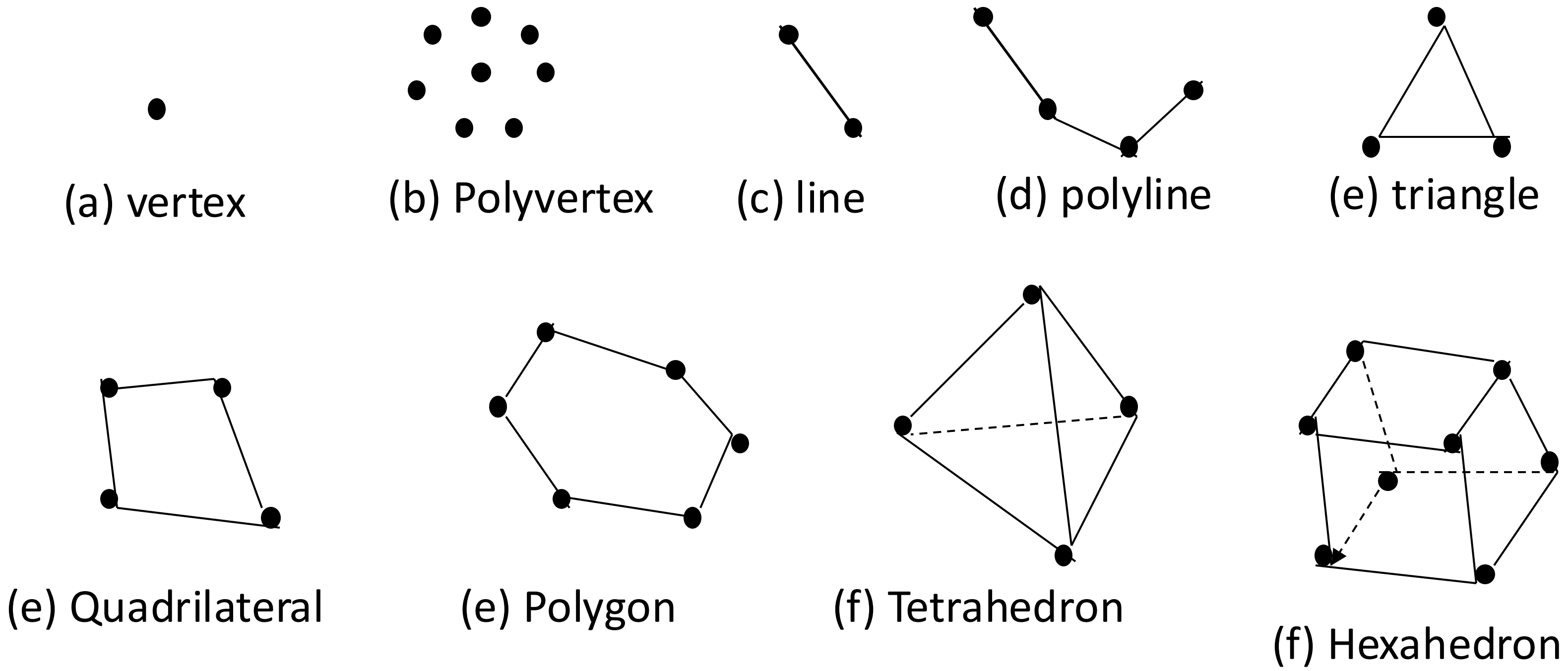
- Allow us to define both the sample points and cells explicitly
- Different cell types can be mixed
- Connectivity is explicitly specified



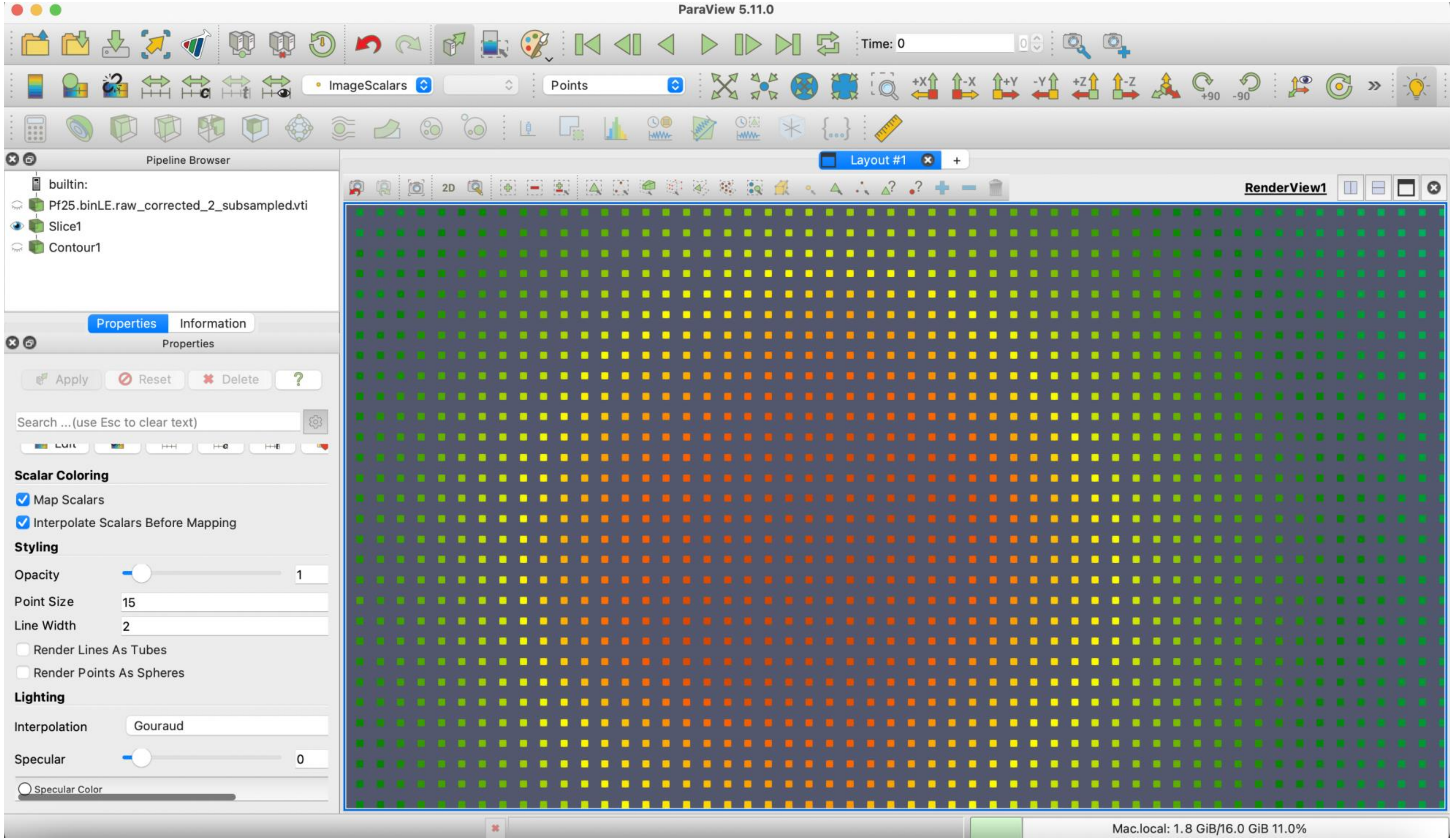
Linear Interpolation for Scientific Data

Why Interpolation?

- Most visualization algorithms have to deal with discrete data
  - Data attributes that are defined at the cell vertices

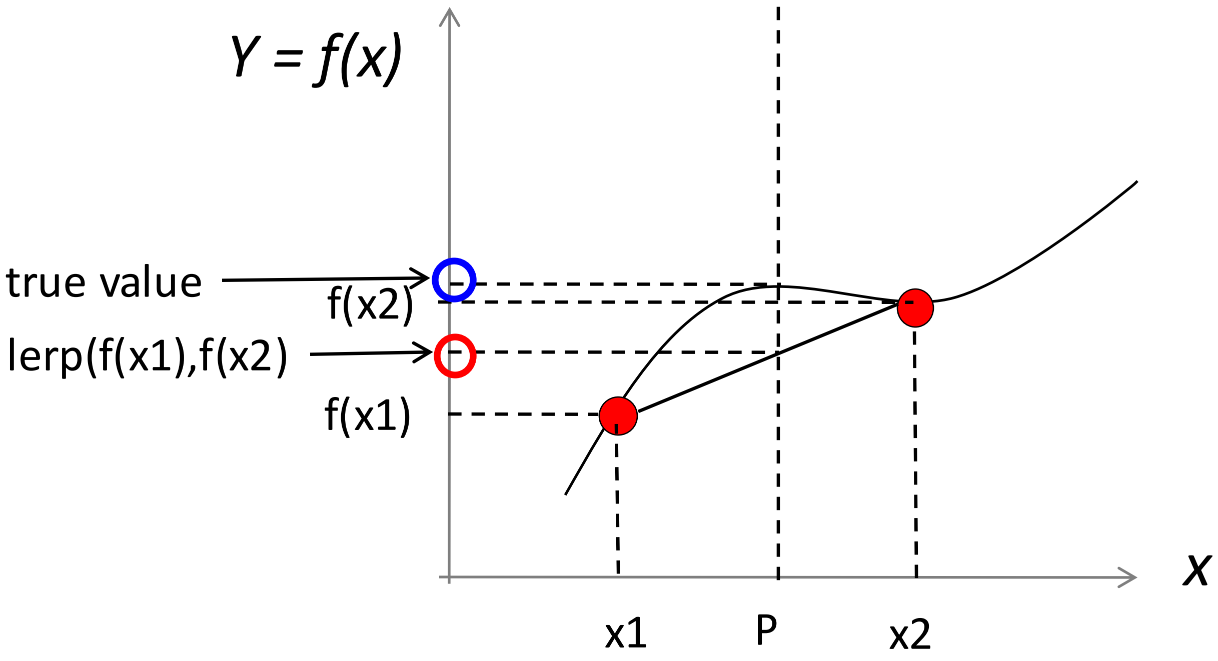


Why Interpolation? ParaView Demo



Linear Interpolation (LERP)

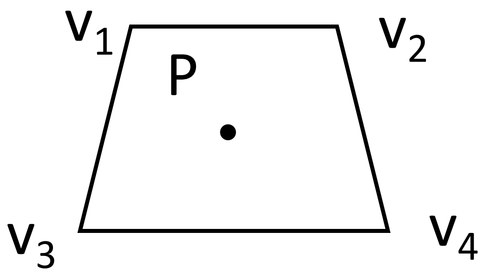
- Linear interpolation (lerp): connecting two points with a straight line in the function plot



Linear Interpolation (LERP)

- General form:

$$V_p = \Sigma \quad w_i * v_i \qquad \text{(weighted sum)}$$

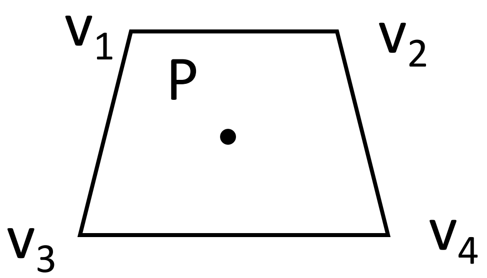


v_i : value at vertex i

$w_{i:}$ weight for v_i

Linear Interpolation (LERP)

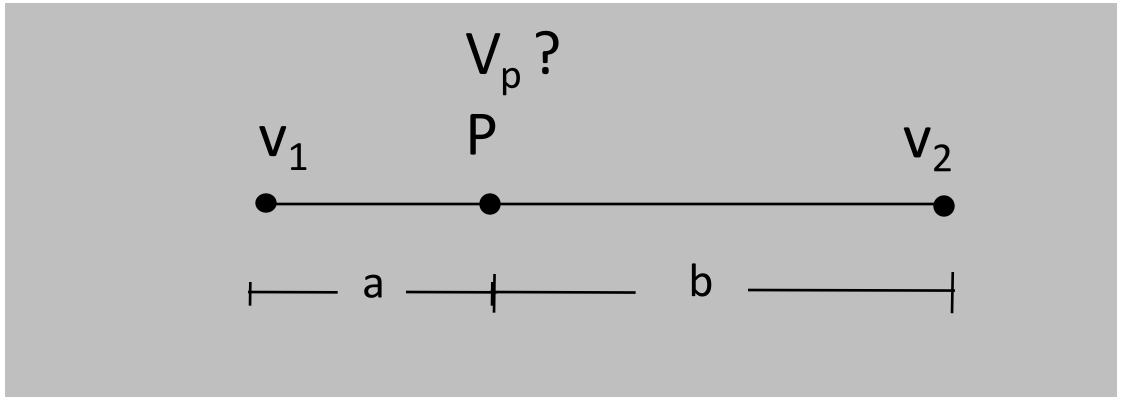
- General form: $V_p = \Sigma \; w_i * v_i$ (weighted sum)



v_i : value at vertex i
w_i: weight for v_i

- Essential information needed:
  - Cell type
  - Data value at cell corners
  - Parametric coordinates of the point in question (P)
    - Related to the position of point P in the cell

LERP in Line

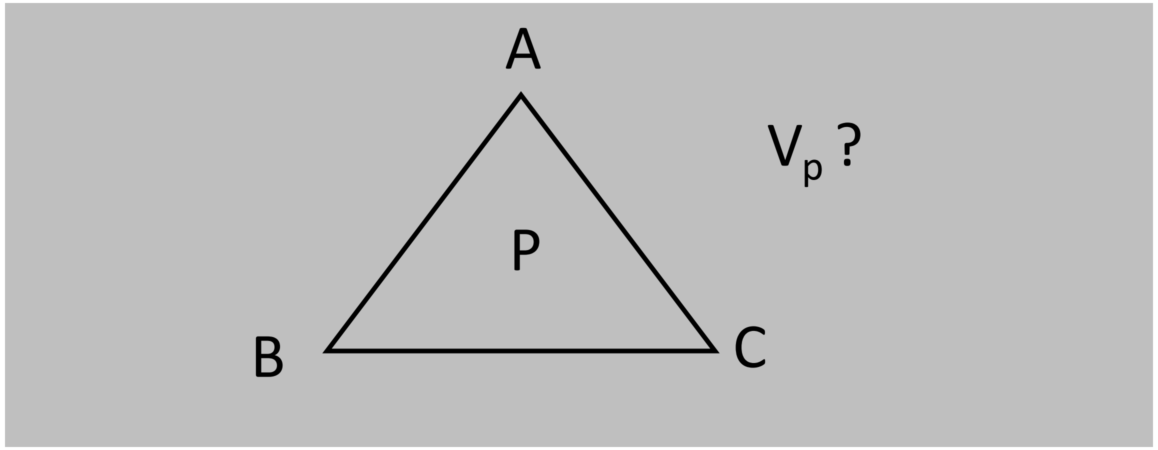


- Parametric coordinate of P: $\alpha = a / (a+b)$
- Linearly interpolated value of P:

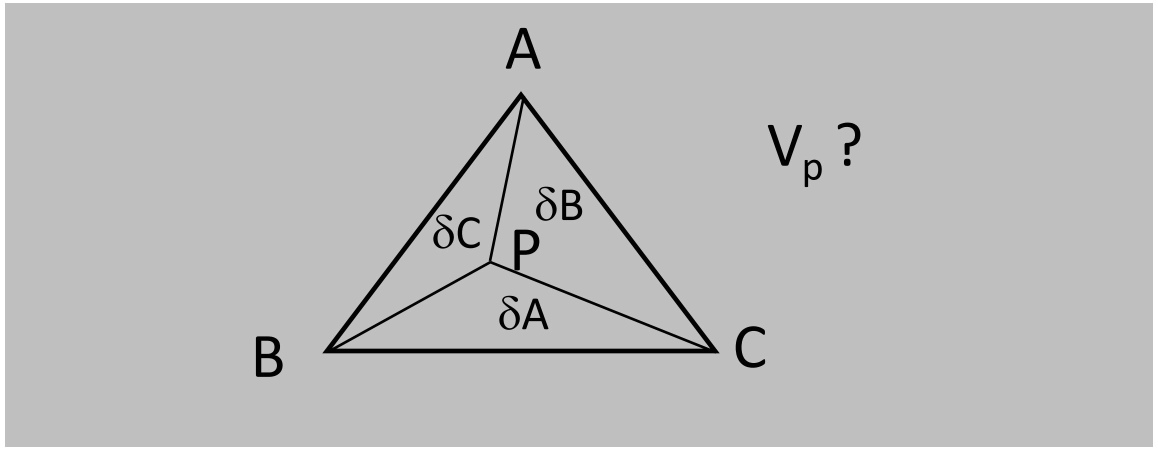
lerp(v1,v2, α)

$$\boxed{V_p = (1-\alpha) * V_1 + \alpha * V_2}$$

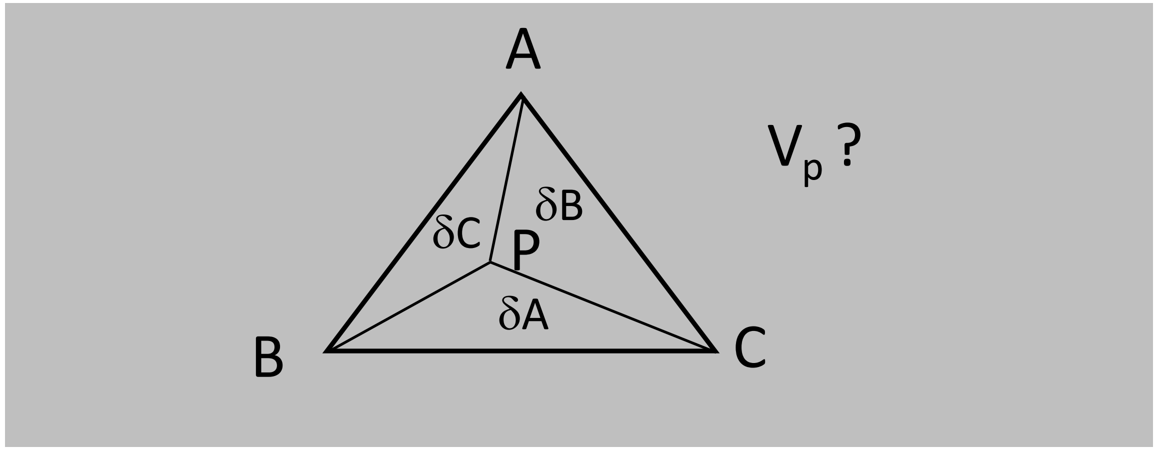
Lerp in Triangle



Lerp in Triangle



Lerp in Triangle



- Parametric coordinates of P: (α,β,γ)

$\alpha = \delta A \,/\, (\delta A + \delta B + \delta C)$

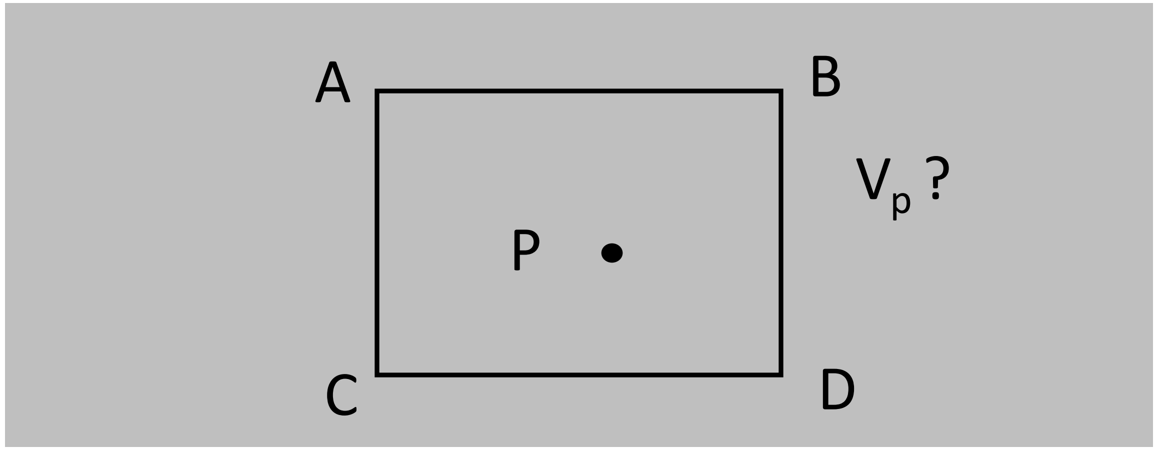
$\beta = \delta B \,/\, (\delta A + \delta B + \delta C)$

$\gamma = \delta C \,/\, (\delta A + \delta B + \delta C)$

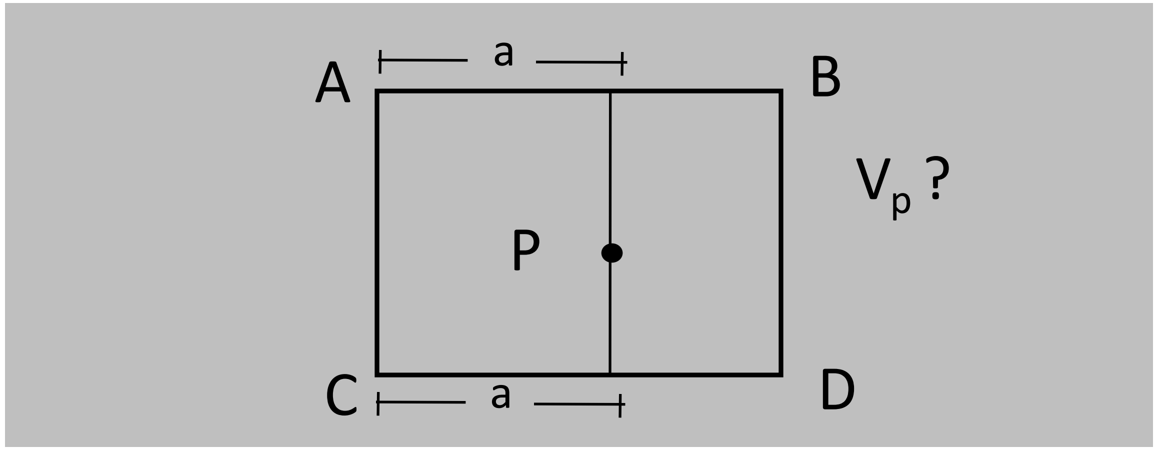
Baricentric Coordinates

- Linearly interpolated value of P: $\boxed{V_A * \alpha + V_B * \beta + V_C * \gamma}$

Lerp in Rectangle



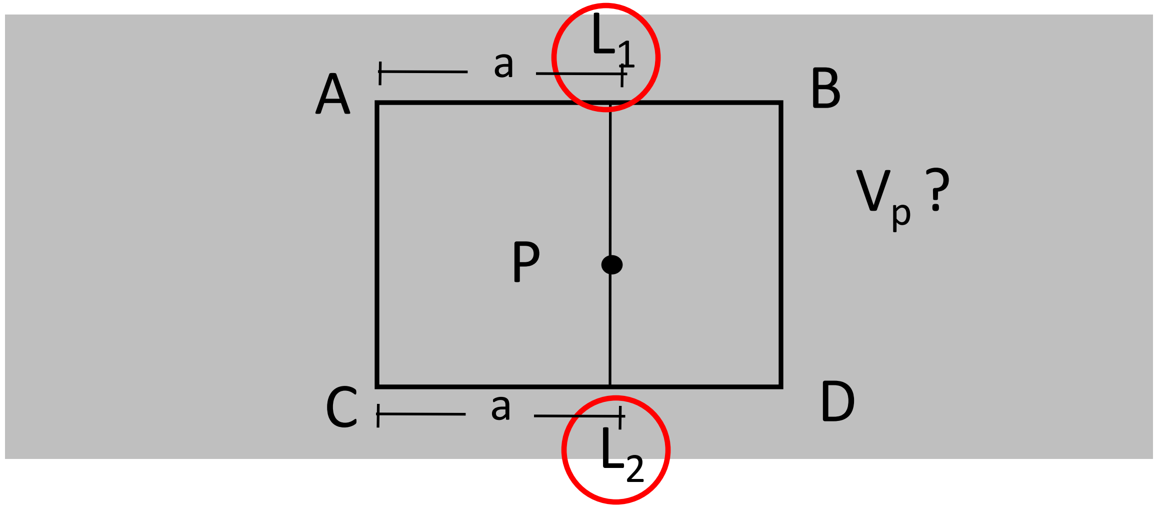
Lerp in Rectangle



- Parametric coordinates of P: (α,β)

$$\alpha = a / \text{width};$$

Lerp in Rectangle

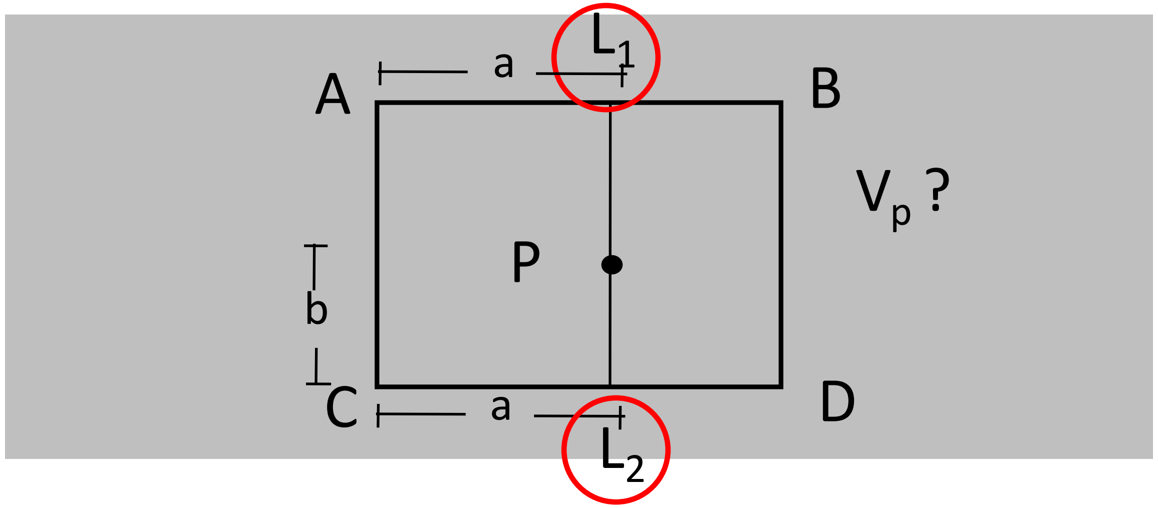


- Parametric coordinates of P: (α,β)

$$\alpha = a / \text{width};$$

- Value at L_1 = Lerp(V_A,V_B,α) ;
- Value at L_2 = Lerp(V_C,V_D,α) ;

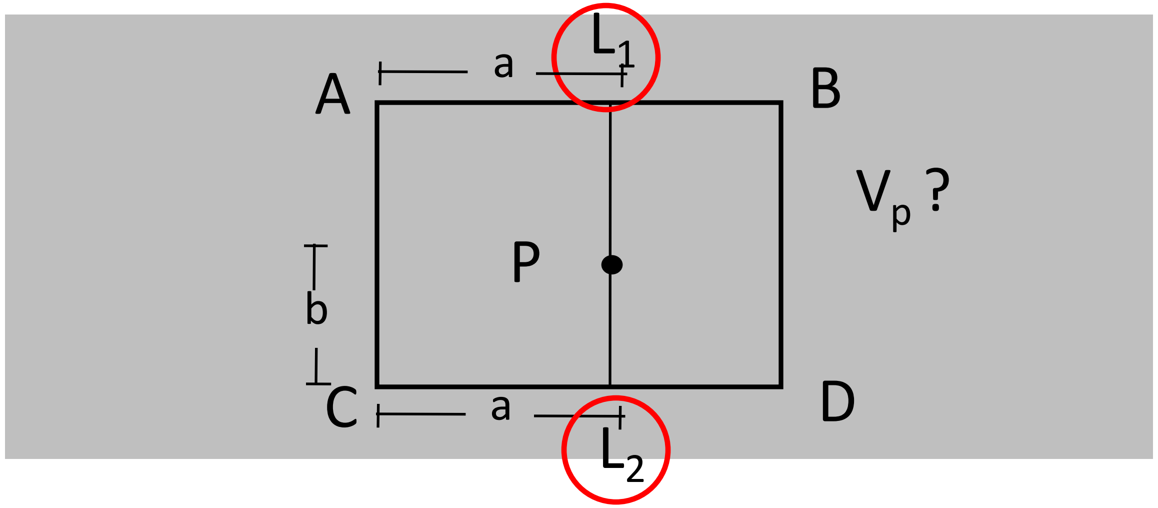
Lerp in Rectangle



- Parametric coordinates of P: (α,β)

$$\alpha = a \,/\, \text{width};$$

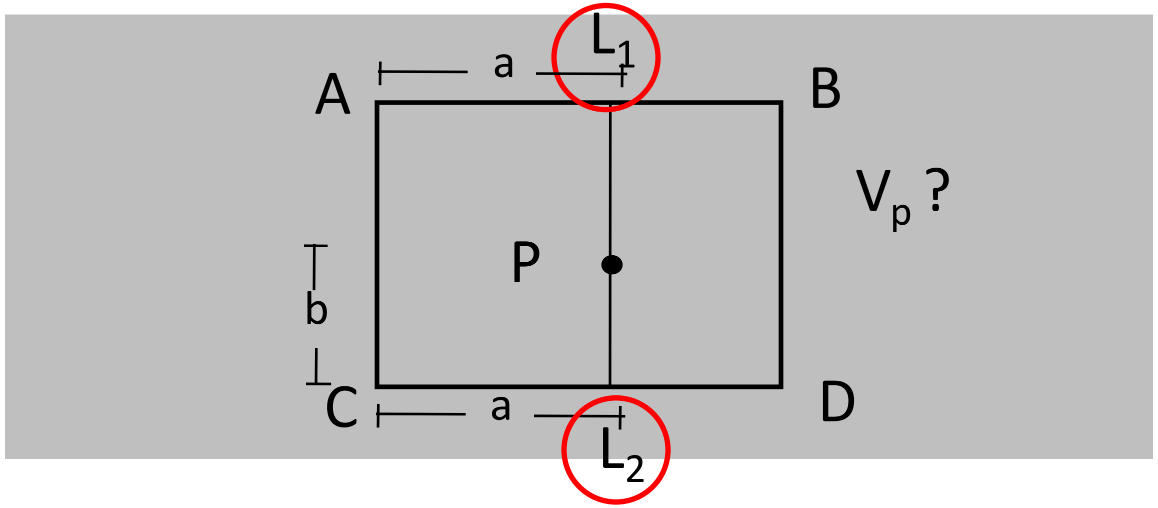
Lerp in Rectangle



- Parametric coordinates of P: (α,β)

$$\alpha = a / \text{width}; \ \beta = b / \text{height}$$

Lerp in Rectangle

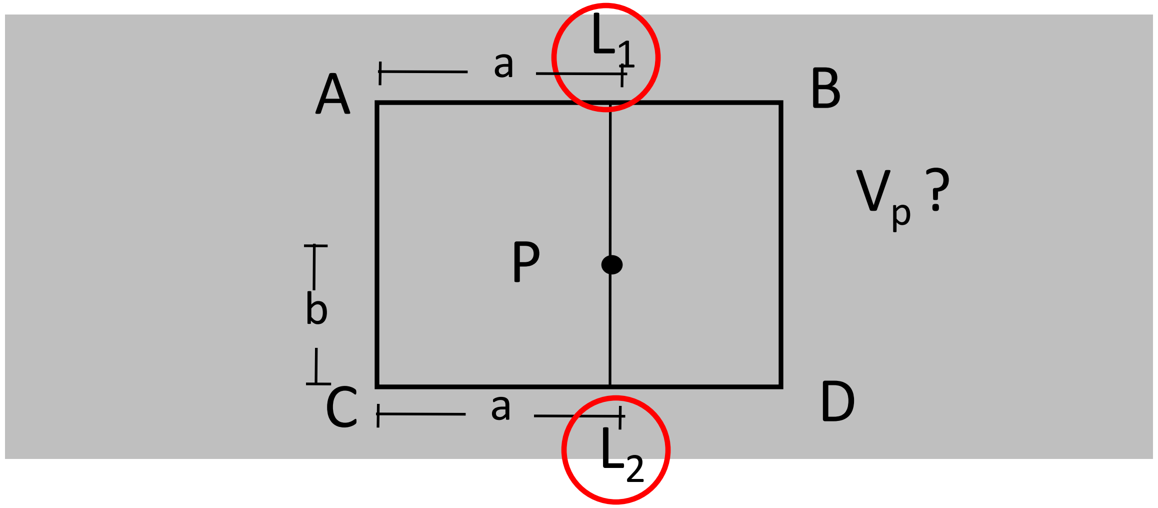


- Parametric coordinates of P: (α,β)

$$\alpha = a / \text{width}; \; \beta = b / \text{height}$$

- Linearly interpolated value of P: $\boxed{\text{Lerp}(V_{L1}, V_{L2}, \beta)}$

Lerp in Rectangle



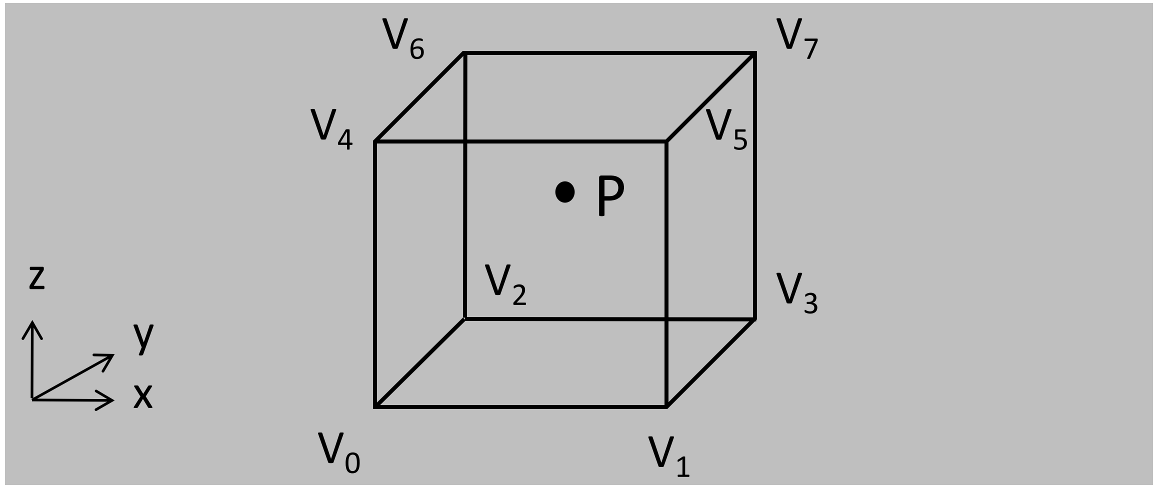
- Parametric coordinates of P: (α,β)

$$\alpha = a / \text{width}; \beta = b / \text{height}$$

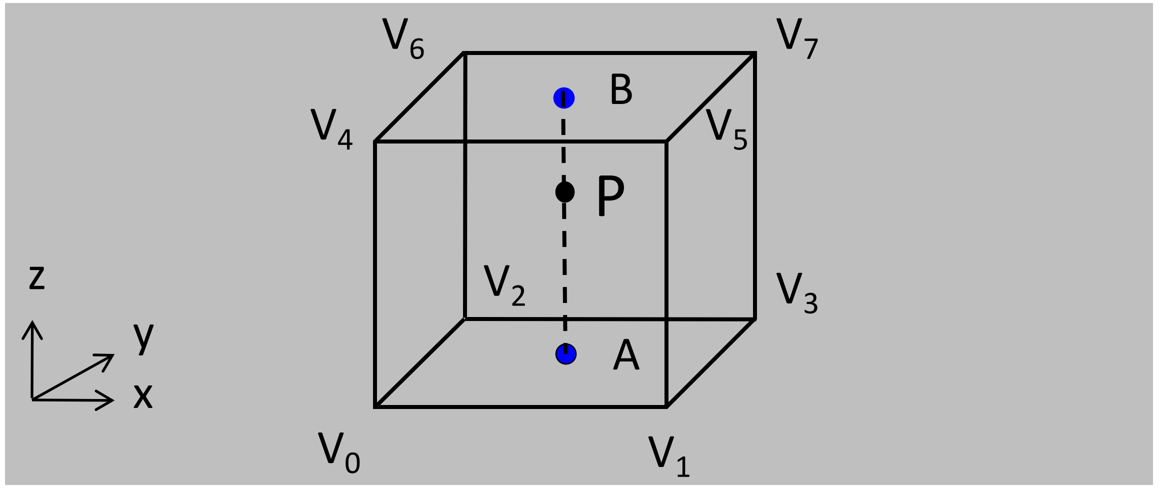
Bi-linear interpolation
Bi-Lerp(V_A,V_B, V_C,V_D)

- Linearly interpolated value of P: $\boxed{\text{Lerp}(V_{L1}, V_{L2}, \beta)}$

Lerp in Cube

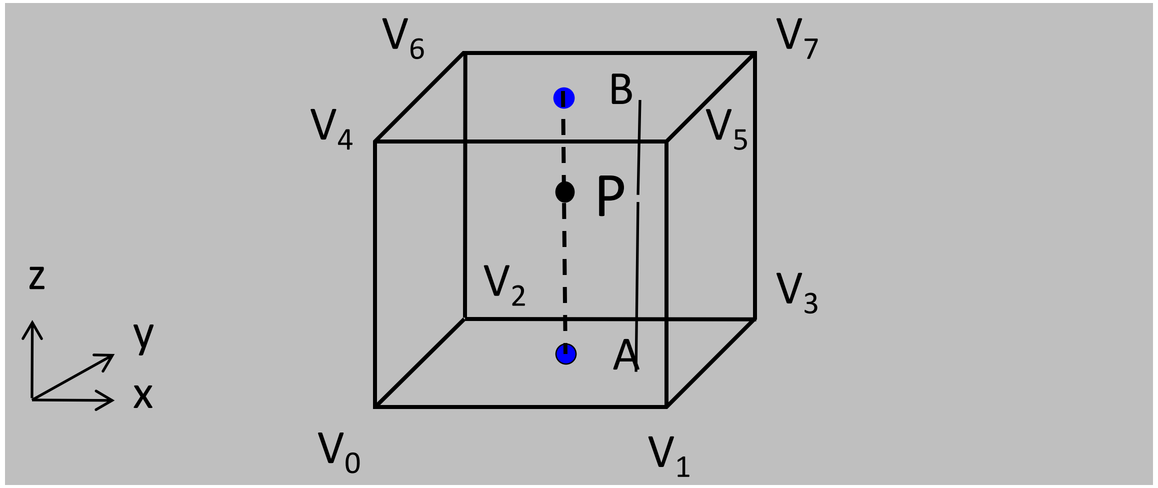


Lerp in Cube



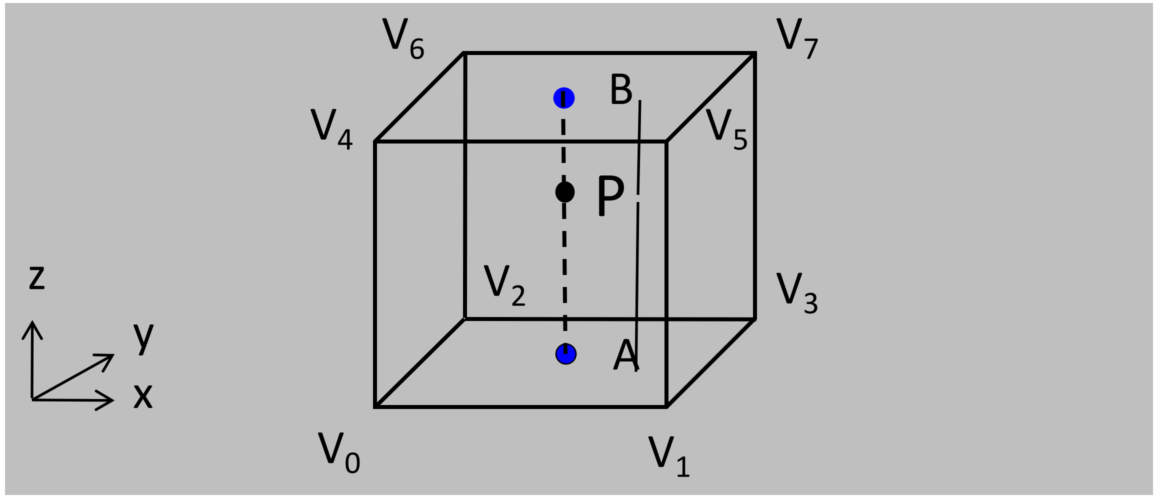
- Value at A = Bi-Lerp(V_0,V_1,V_2,V_3) ;
- Value at B = Bi-Lerp(V_4,V_5,V_6,V_7) ;

Lerp in Cube



- Value at A = Bi-Lerp(V_0,V_1,V_2,V_3) ;
- Value at B = Bi-Lerp(V_4,V_5,V_6,V_7) ;

Lerp in Cube



- Value at A = Bi-Lerp(V_0,V_1,V_2,V_3) ;
- Value at B = Bi-Lerp(V_4,V_5,V_6,V_7) ;
- Value at P = Lerp(A,B, PA/AB);

Tri-linear interpolation

Isocontour Algorithm (2D and 3D)

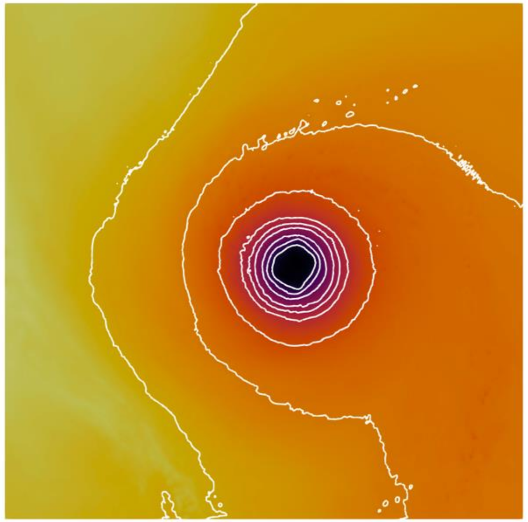
What is an Isocontour?

- 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

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

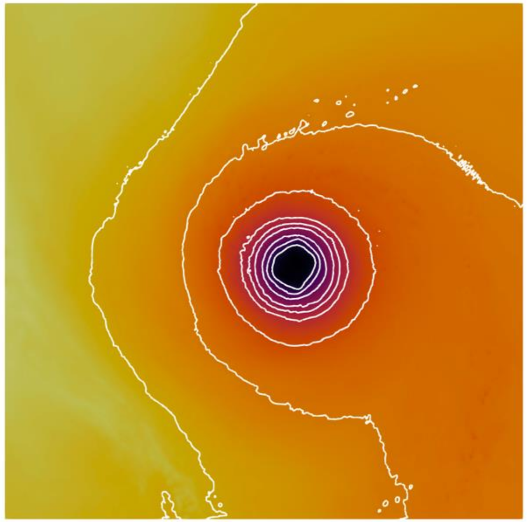


What is an Isocontour?

- 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

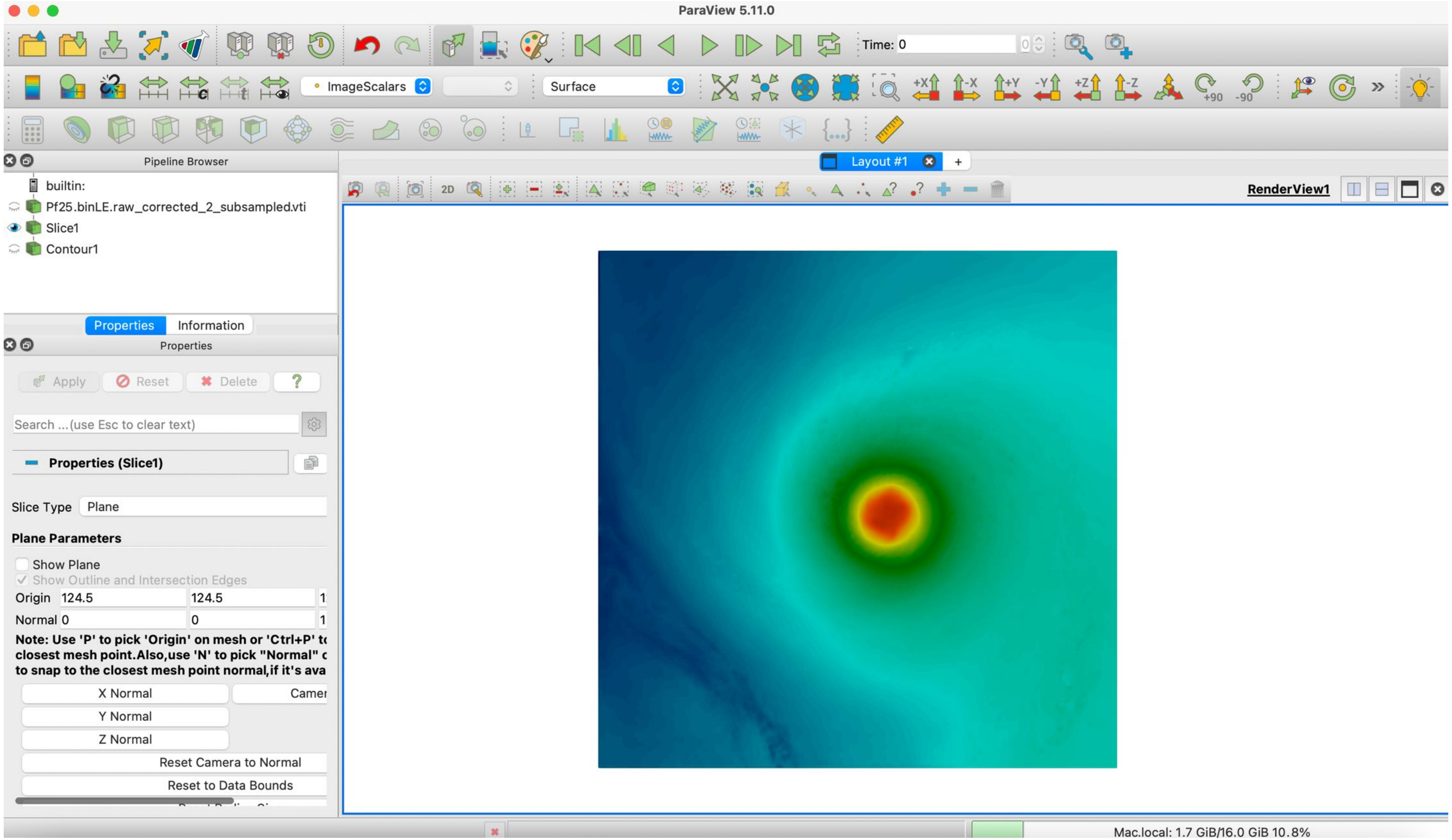
– A technique for analyzing and visualizing scalar field data or scalar functions

2D isocontour: Isoline



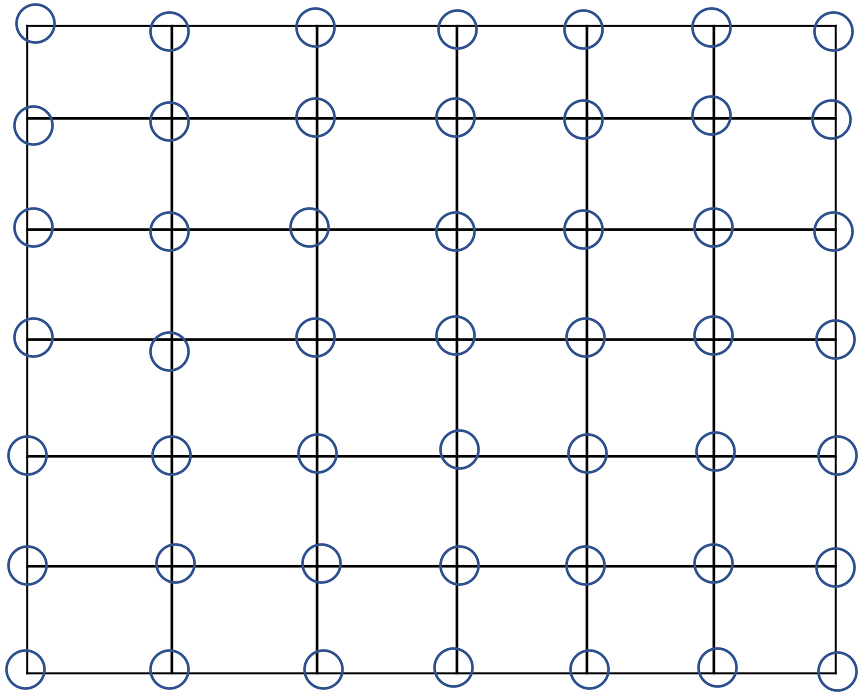
3D isocontour: Isosurface

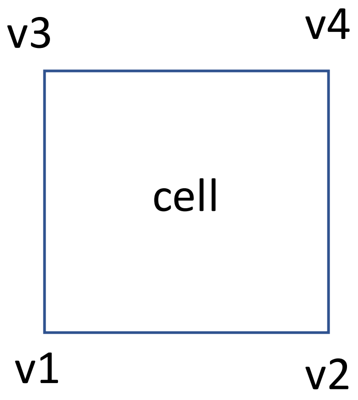
Isocontour Demo ParaView



2D Isocontour Extraction

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

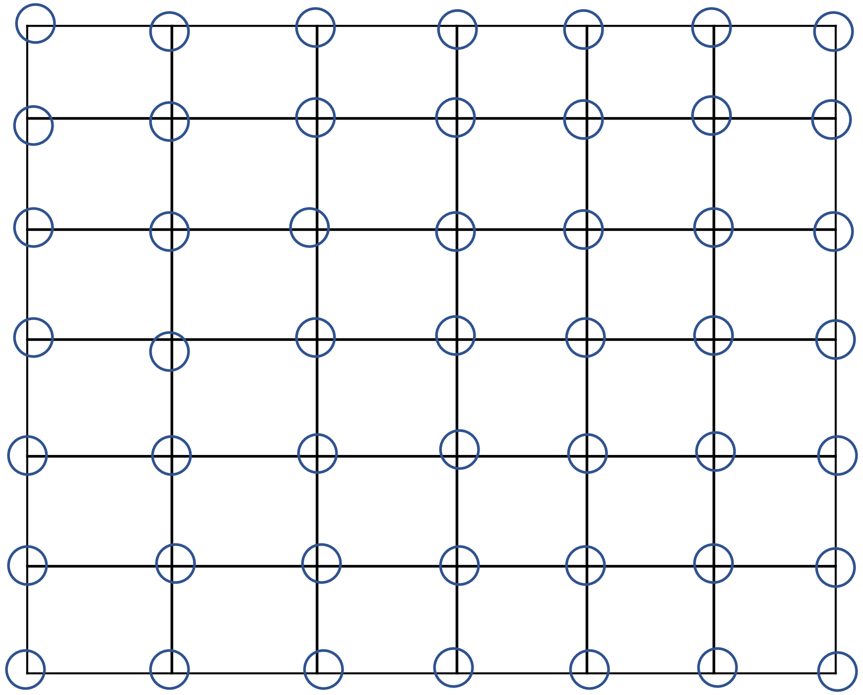


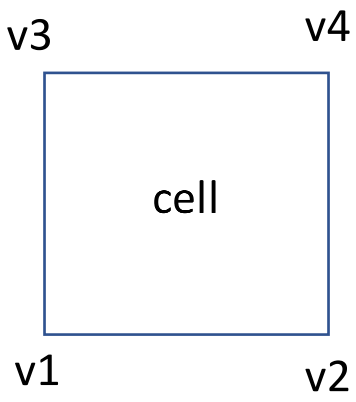


v1,v2,v3,v4 all are > C
No isoline in this cell

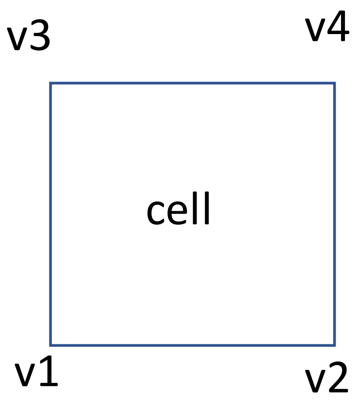
2D Isocontour Extraction

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





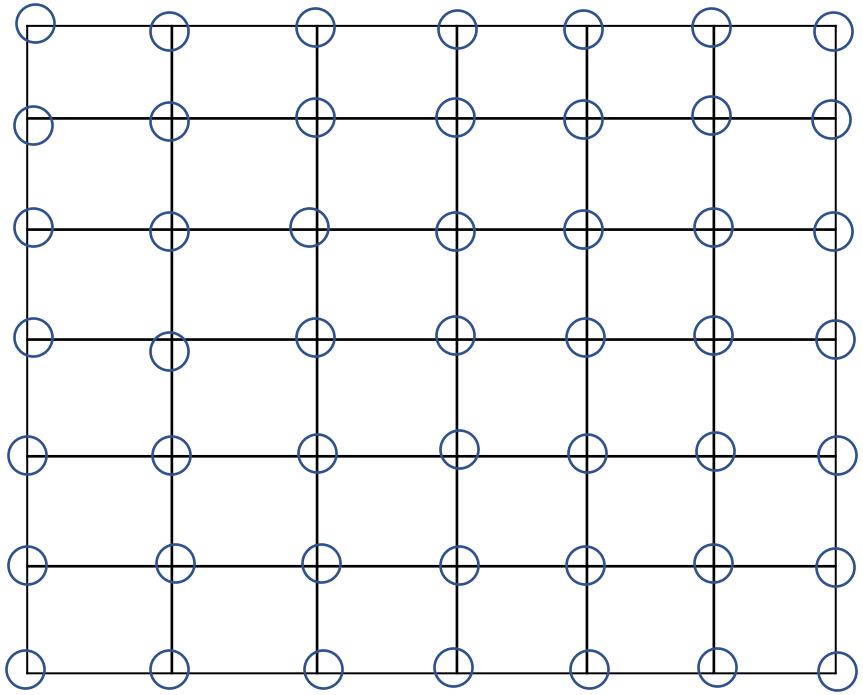
v1,v2,v3,v4 all are **> C**
No isoline in this cell

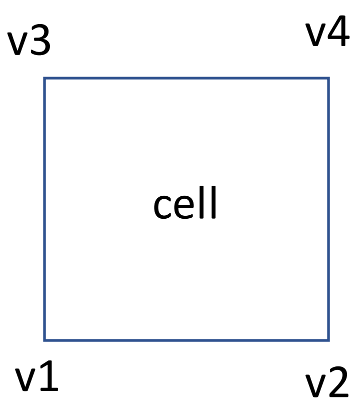


v1,v2,v3,v4 all are **< C**
No isoline in this cell

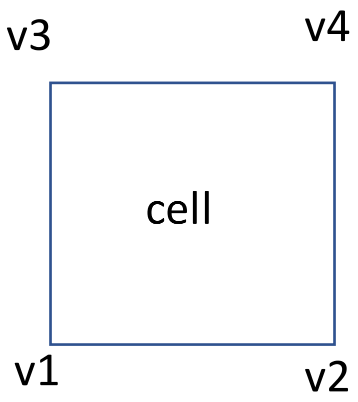
2D Isocontour Extraction

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

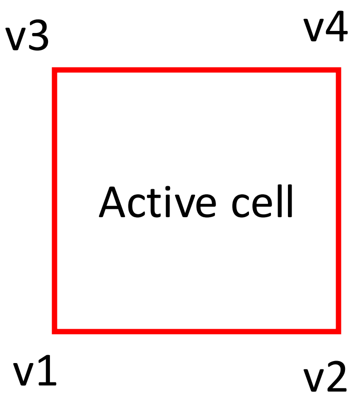




v1,v2,v3,v4 all are > C
No isoline in this cell



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



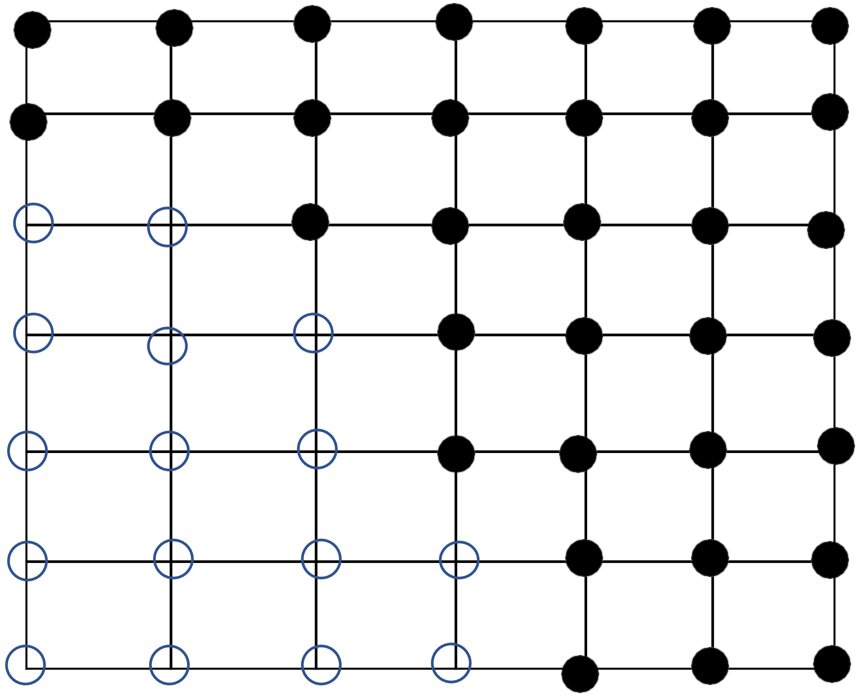
Otherwise, cell contains isocontour segment

2D Isocontour Extraction: Marching Squares

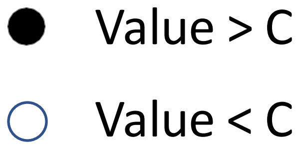
- Given a 2D scalar field, compute isocontour (isoline) for isovalue = **C**
- This is usually done in a cell-by-cell manner using **<u>Marching Squares algorithm</u>**

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 **<u>Marching Squares algorithm</u>**

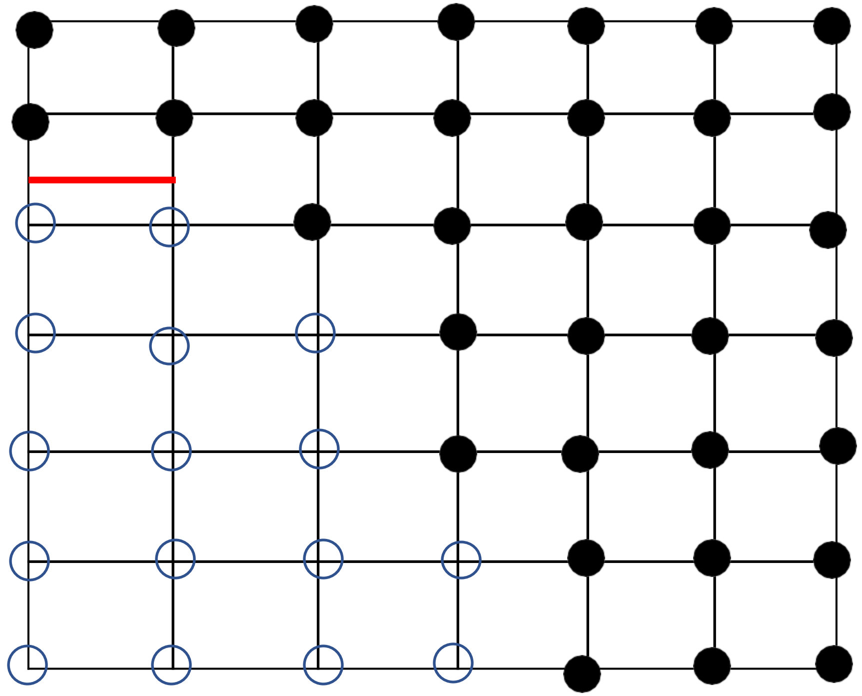


Contour value (isovalue) = C

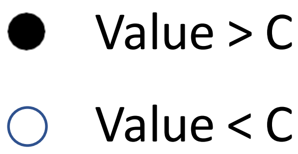


2D Isocontour Extraction: Marching Squares

- 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

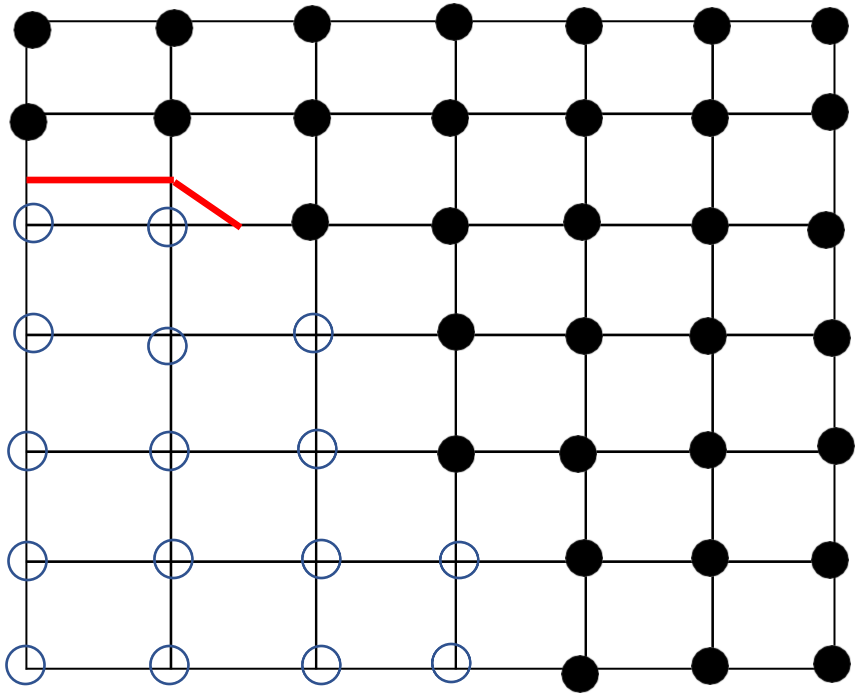


Contour value (isovalue) = C

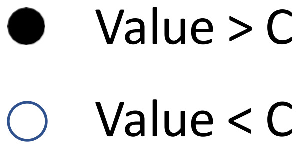


2D Isocontour Extraction: Marching Squares

- 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

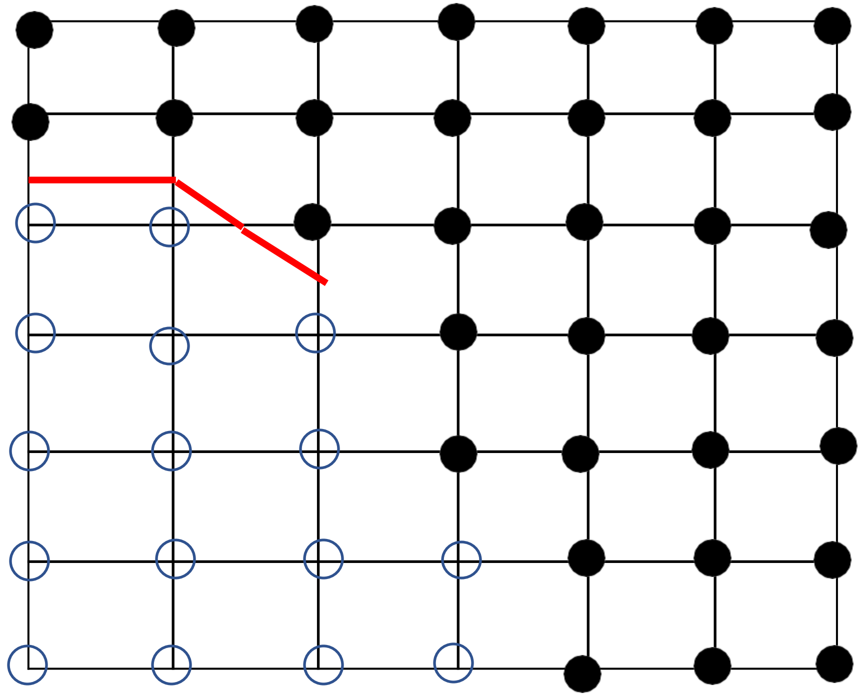


Contour value (isovalue) = C

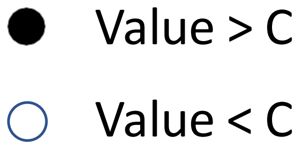


2D Isocontour Extraction: Marching Squares

- 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

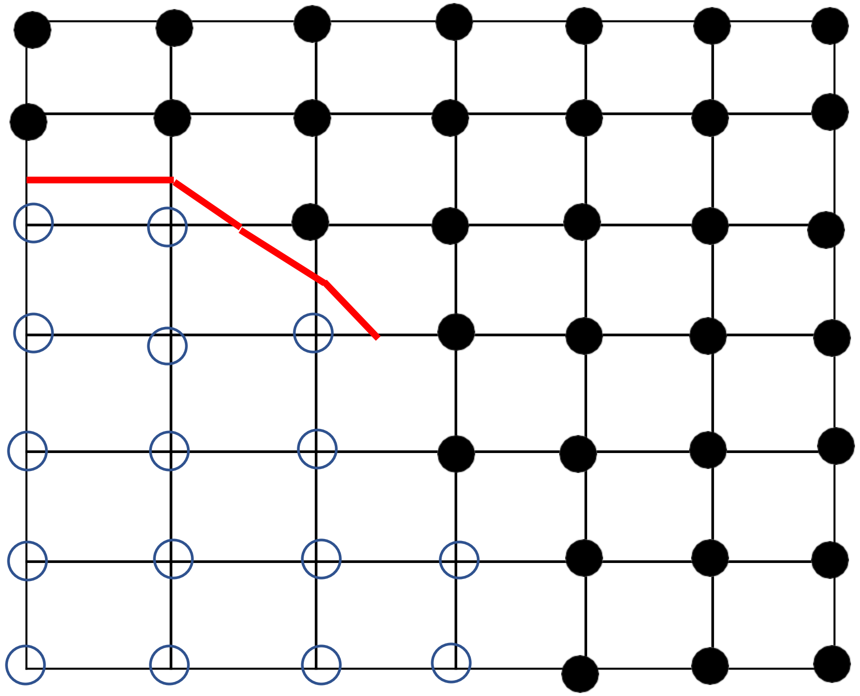


Contour value (isovalue) = C



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



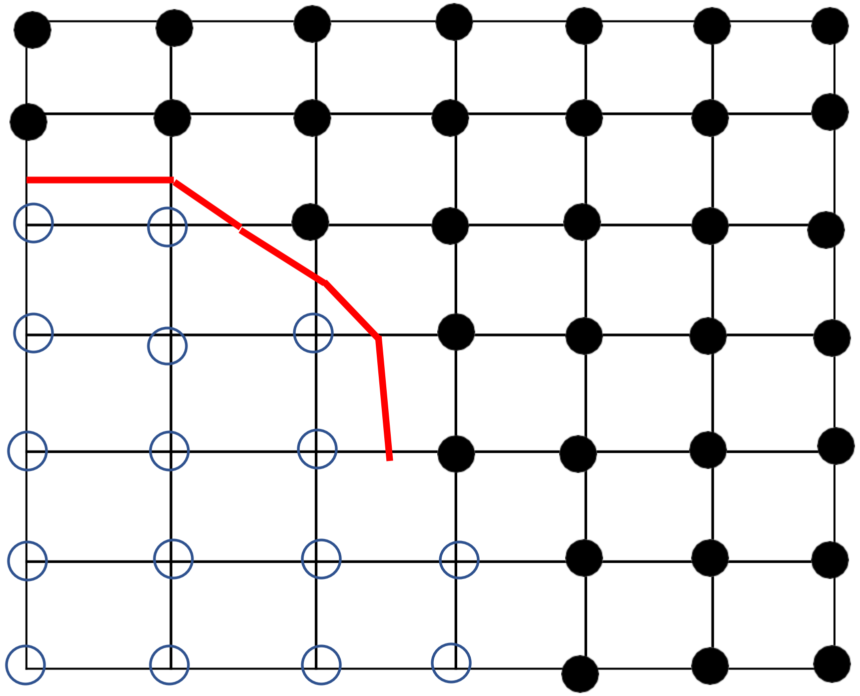
Contour value (isovalue) = C

● 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



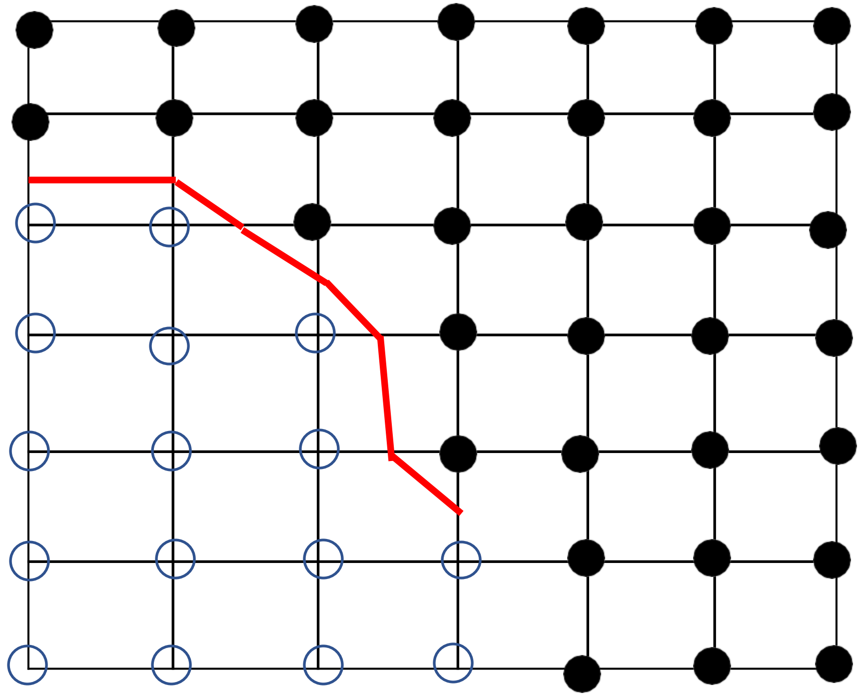
Contour value (isovalue) = C

● 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



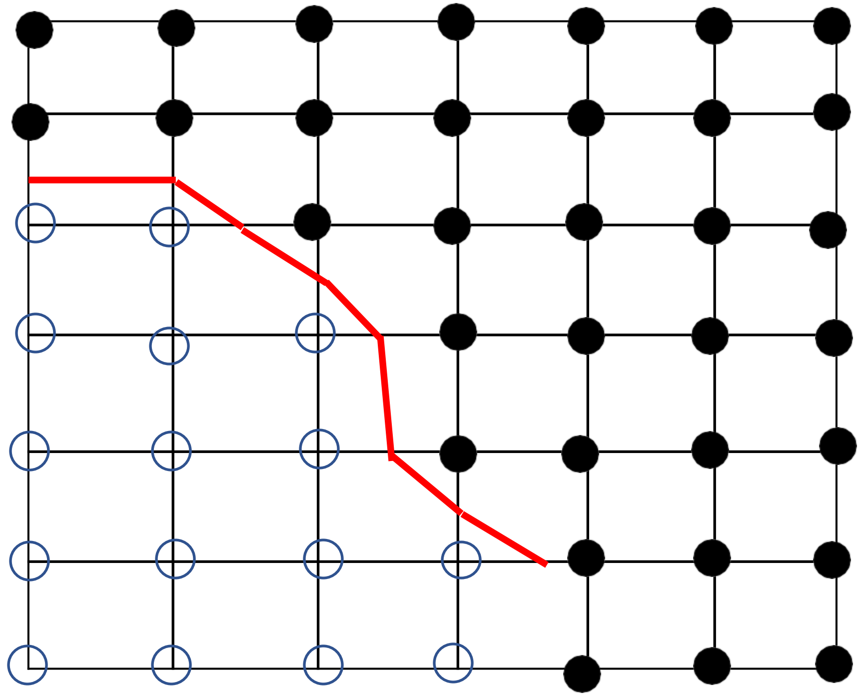
Contour value (isovalue) = C

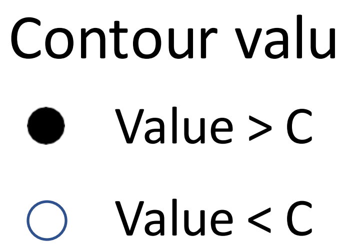
● Value > C

○ Value < C

2D Isocontour Extraction: Marching Squares

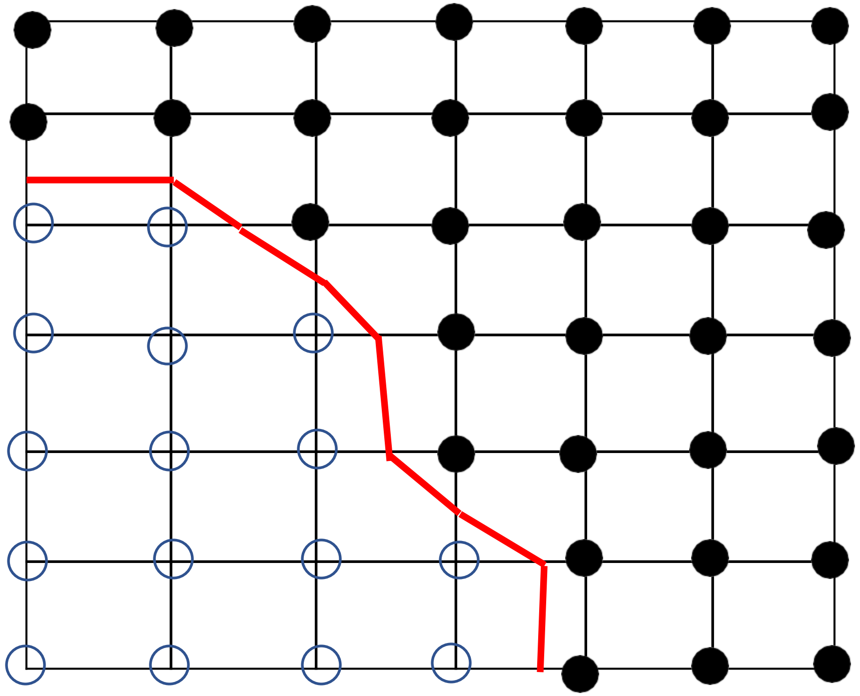
- Given a 2D scalar field, compute isocontour (isoline) for isovalue = **C**
- This is usually done in a cell-by-cell manner using Marching Squares algorithm



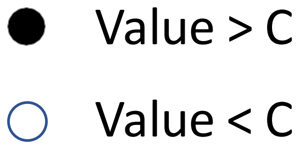


2D Isocontour Extraction: Marching Squares

- 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

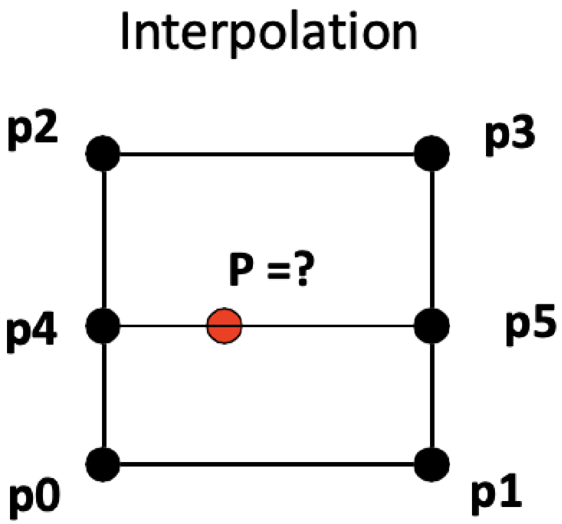


Contour value (isovalue) = C



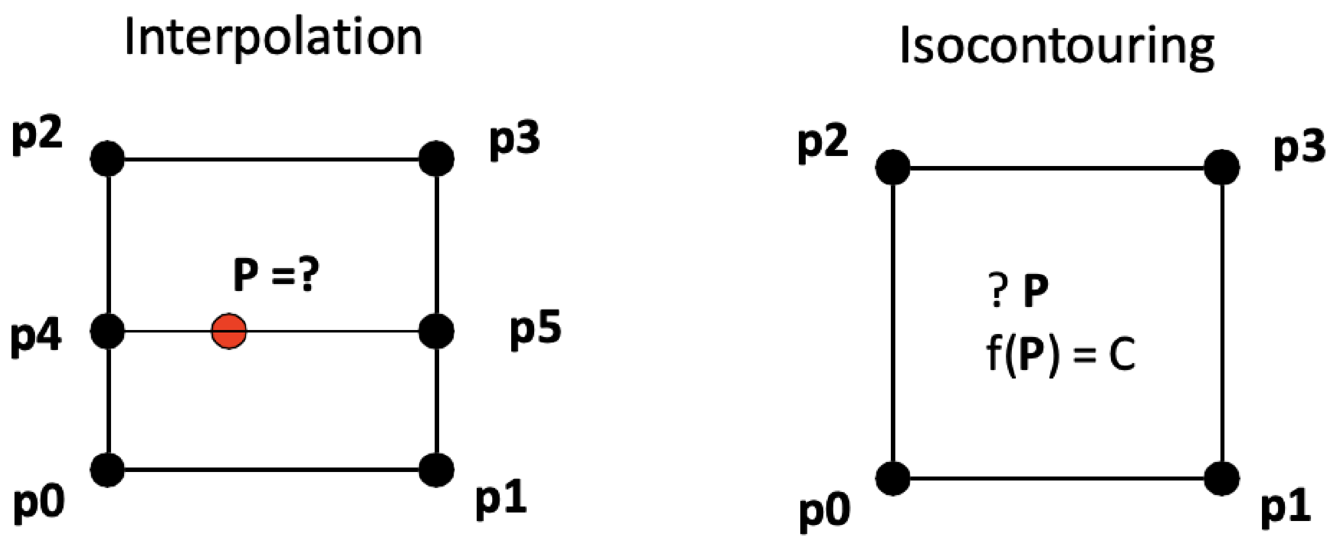
Isocontour in a 2D Cell

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



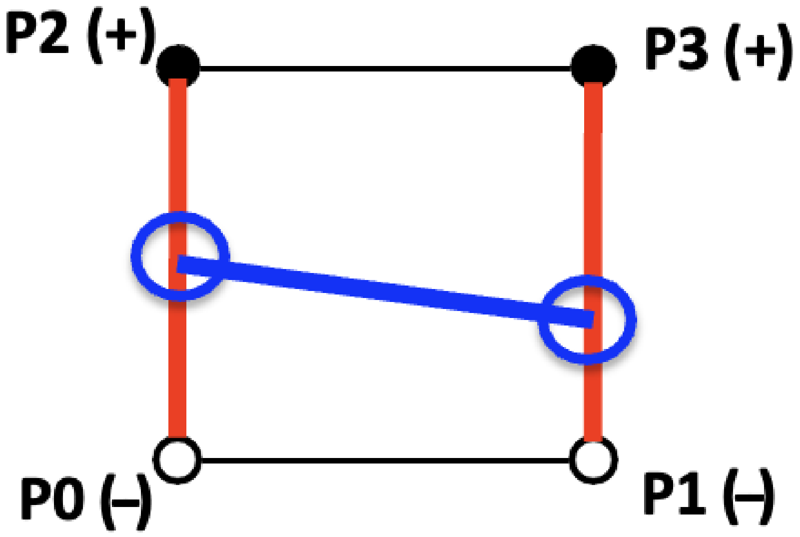
Isocontour in a 2D Cell

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



Isocontouring by Linear Interpolation

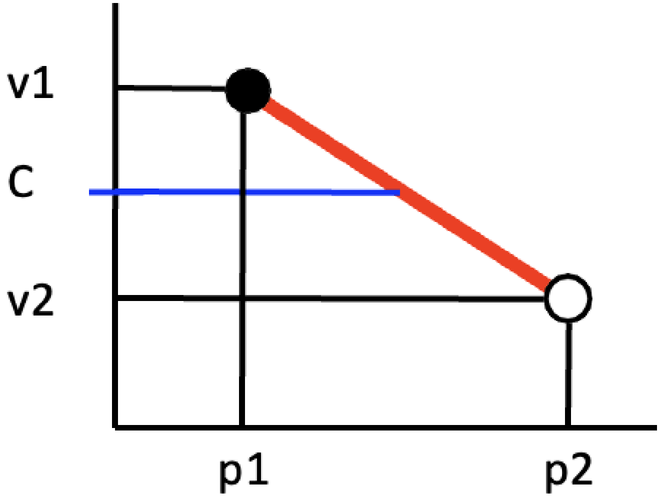
- Compute isocontour within a cell based on linear interpolation



- 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

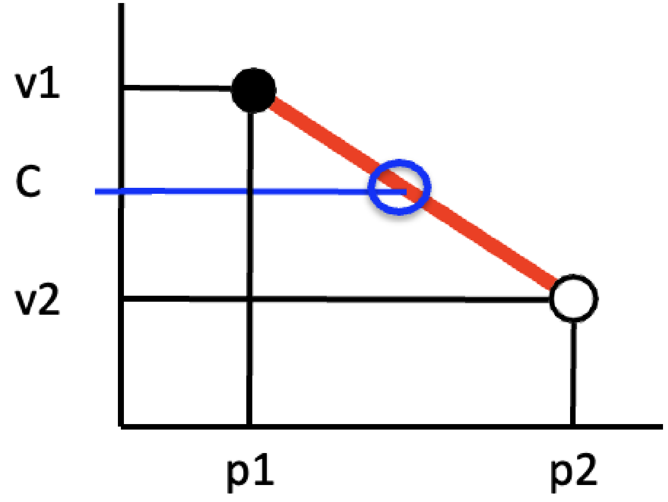


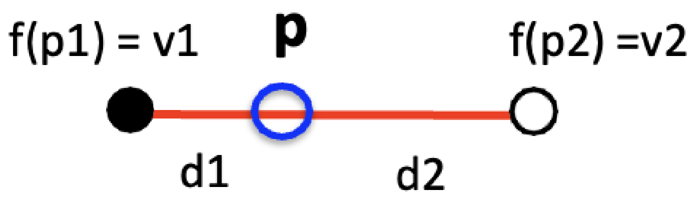
Step 2: Compute Intersection

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

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

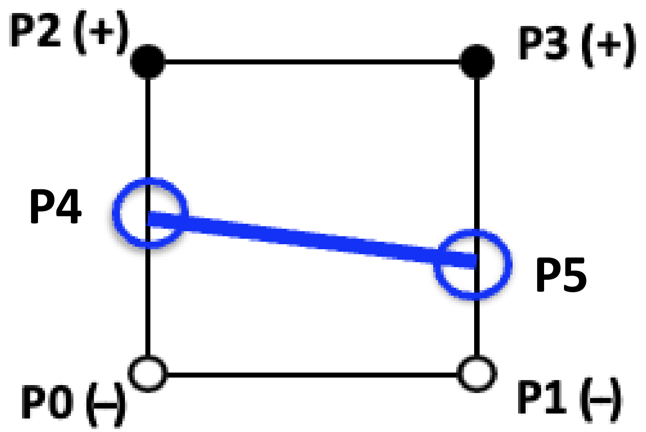
$$\mathbf{p} = (v1-C)/(v1 - v2) * (\mathbf{p}2 - \mathbf{p}1) + \mathbf{p}1$$





Step 3: Connect the Dots

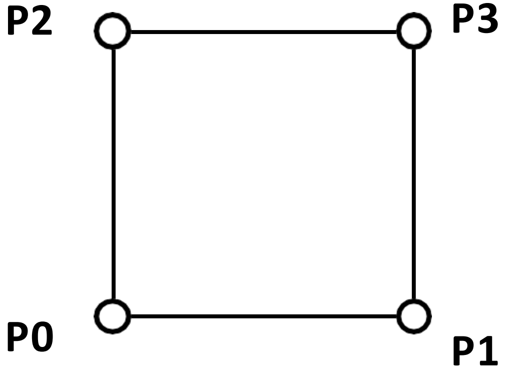
- Based on the principle of linear interpolation, all points along the line **P4P5** have values equal to C (isovalue)



Repeat Step1 – Step 3 for all cells

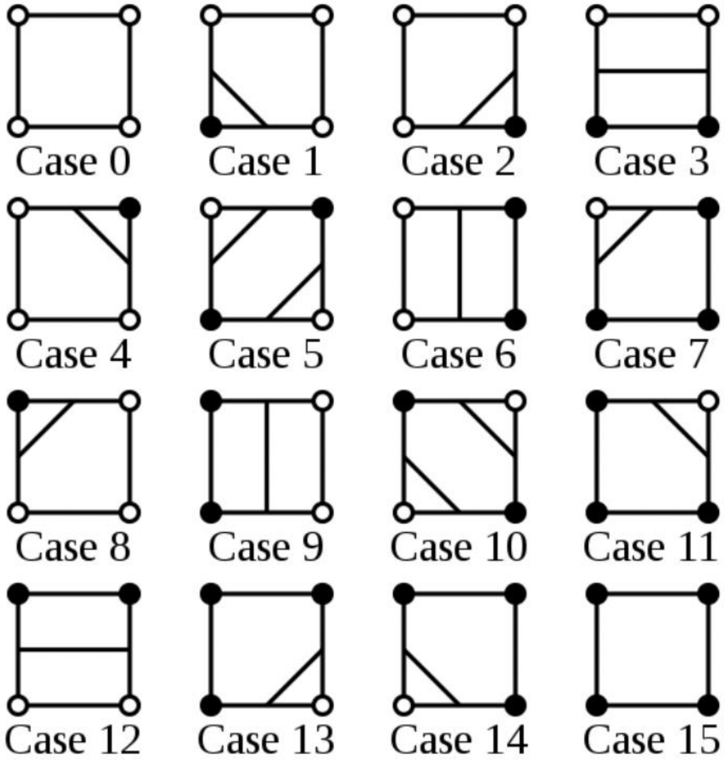
Isocontour Cases

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



P4	P3	P2	P1

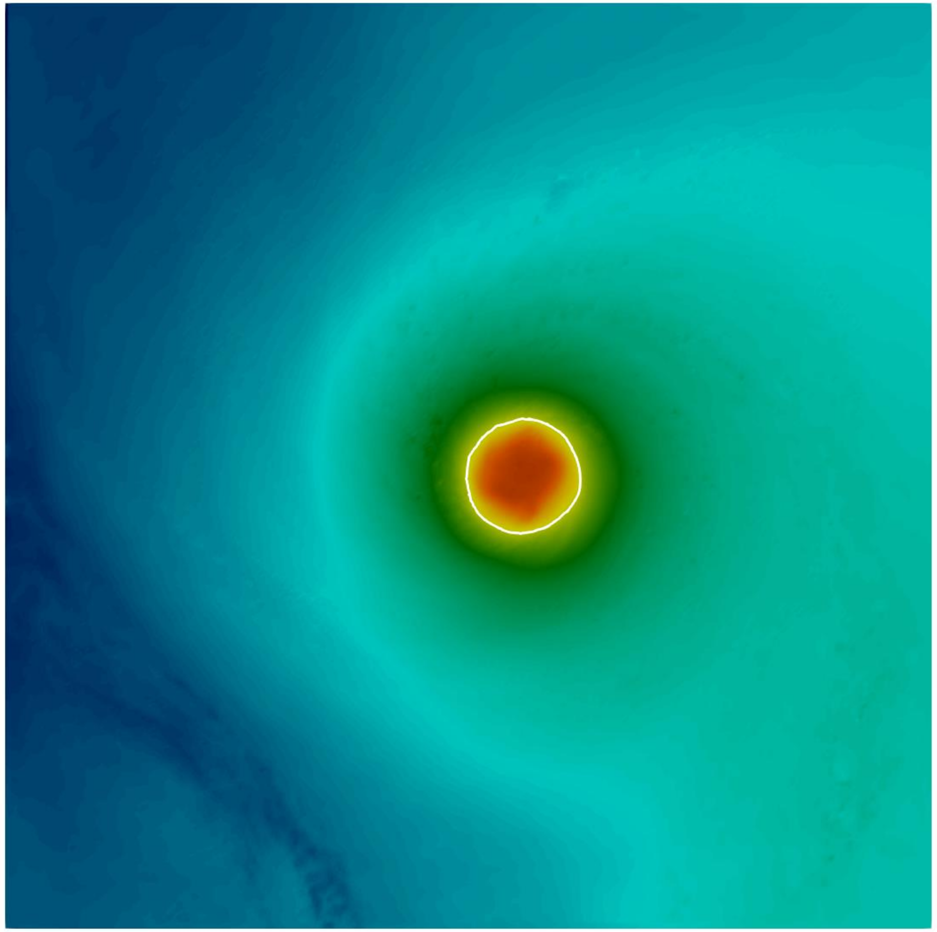
1: > C
0: < C



- 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

Putting it All Together

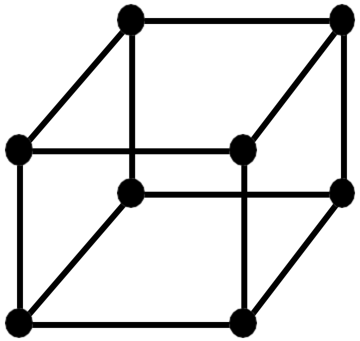
- 2D Isocontouring algorithm for square meshes:
  - Process one cell at a time
  - Compare the values at 4 vertices with the contour value C and identify intersected edges
  - Linearly interpolate along the intersected edges
  - Connect the interpolated points together



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

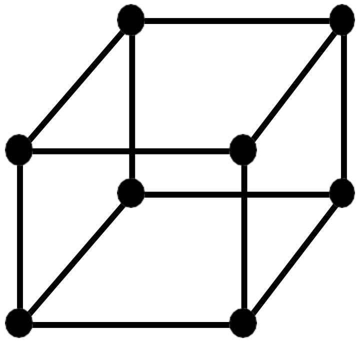
3D Isocontour: Cube/Rectangular Cells



Cube/Rectangular cell

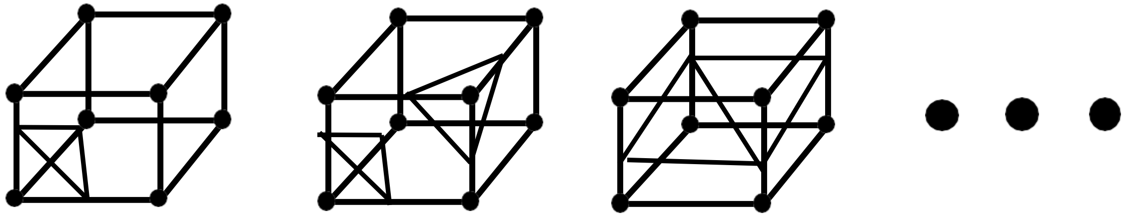
- With 8 vertices in a cell, each having a value greater or smaller than the contour value, there can be $2^8 = 256$ possible cases

3D Isocontour: Cube/Rectangular Cells



Cube/Rectangular cell

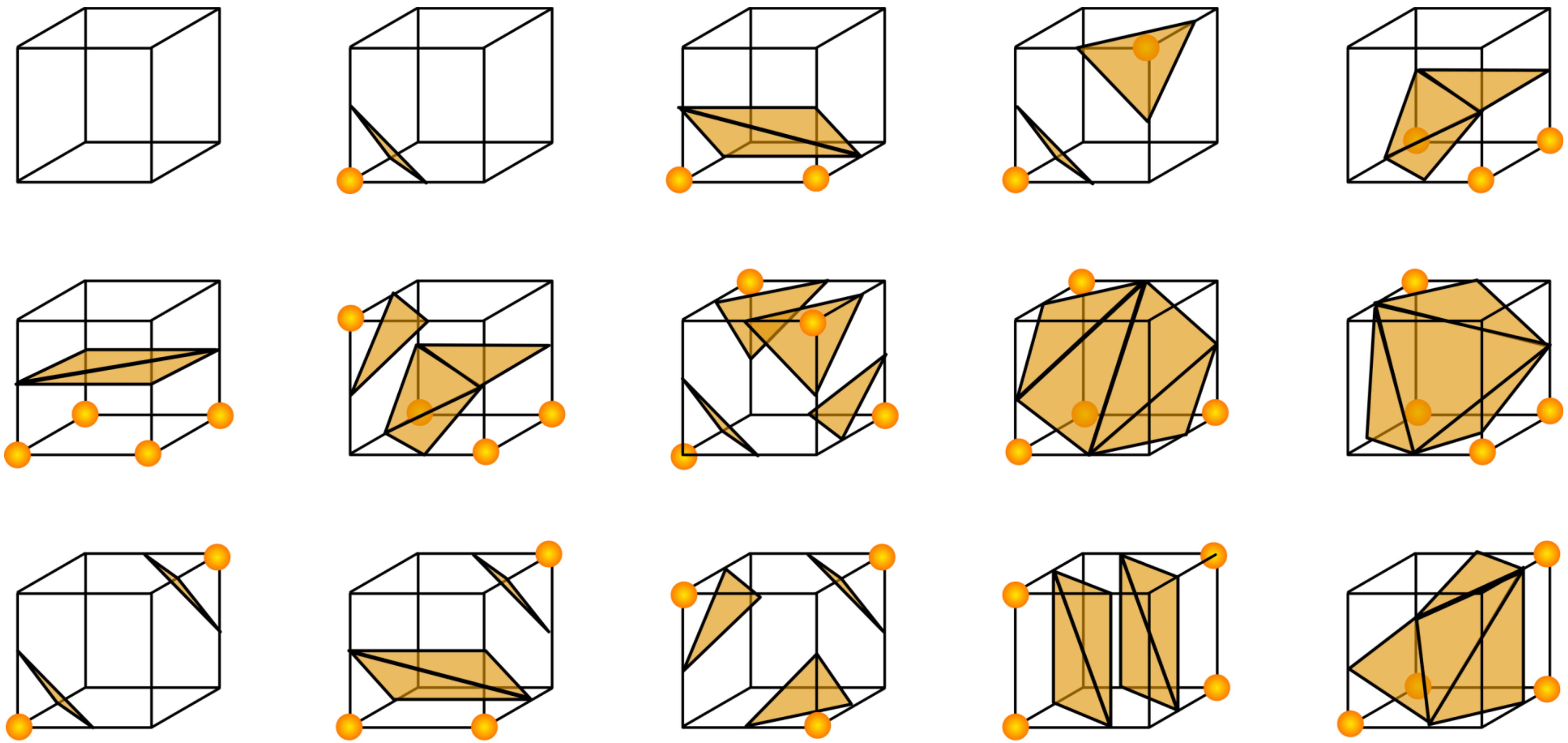
- With 8 vertices in a cell, each having a value greater or smaller than the contour value, there can be $2^8 = 256$ possible cases



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

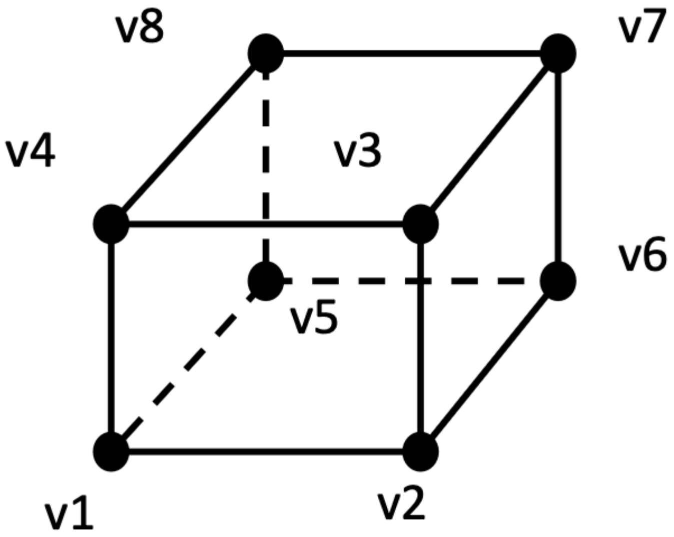
3D Isosurface Unique Cases

- 15 Topologically Unique Cases



Marching Cubes Algorithm

- Lorensen and Cline in 1987
- Mark each cell vertex 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]

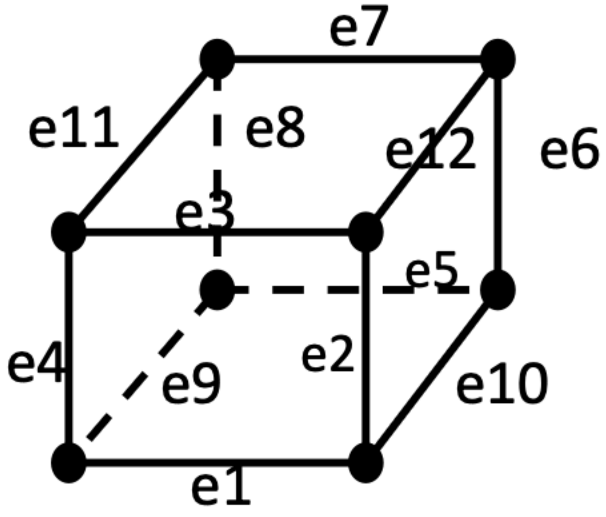


Index =

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



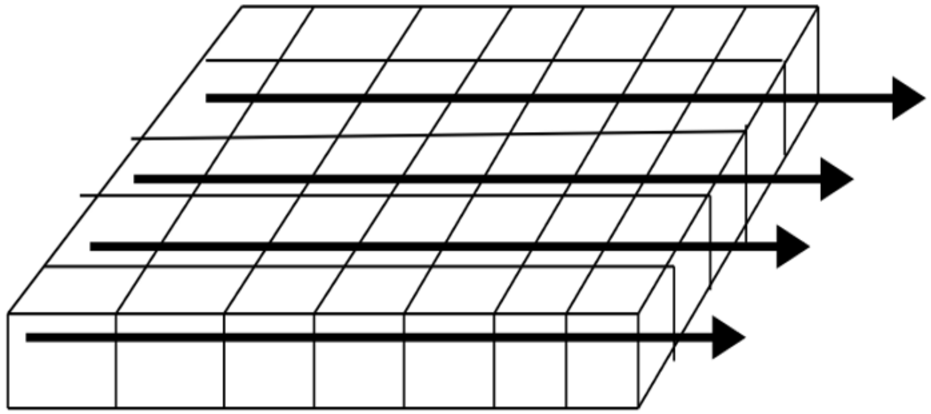
Index =

v8	v7	v6	v5	v4	v3	v2	v1

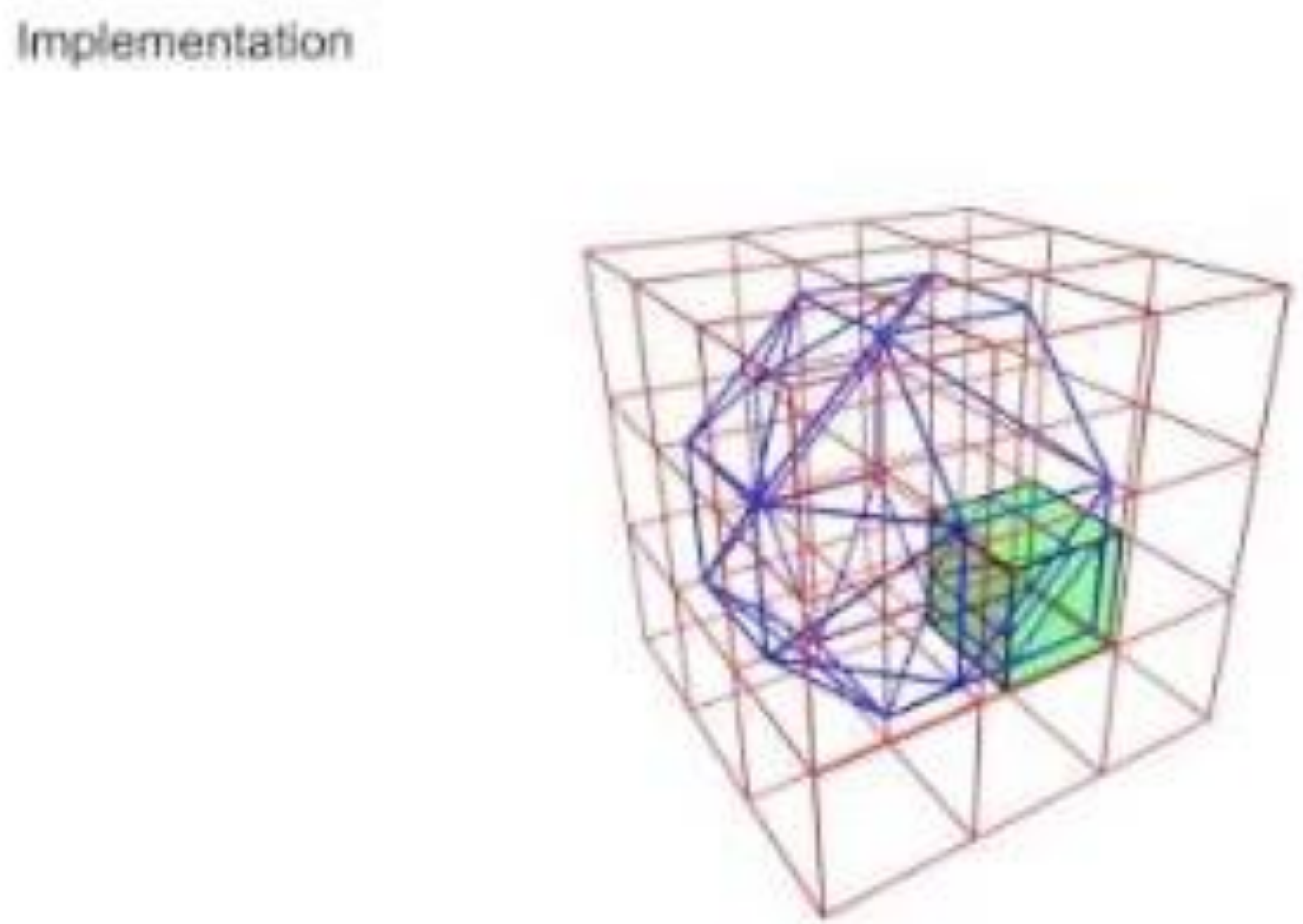
Index	intersection edges
0	e1, e3, e5
1	…
2	
3	
	• • •
14	

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



Marching Cubes Algorithm: Animation



https://www.youtube.com/watch?v=B_xk71YopsA