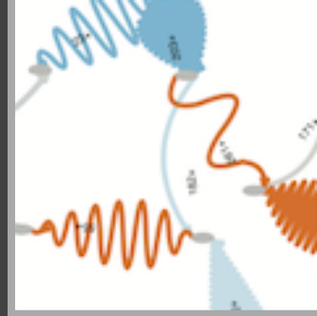
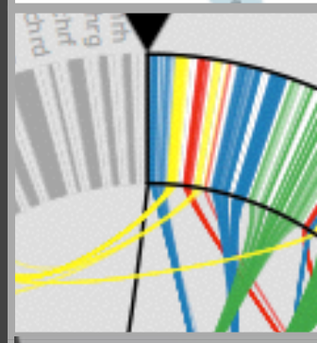
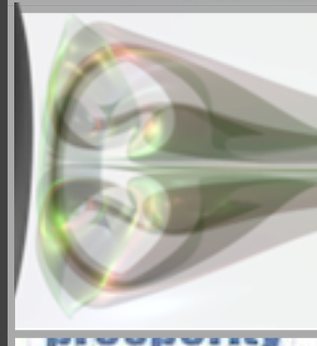
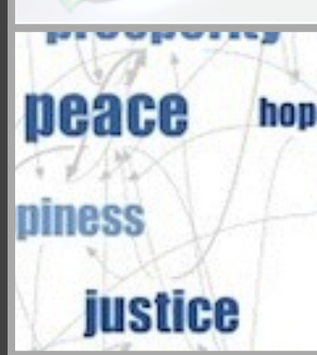
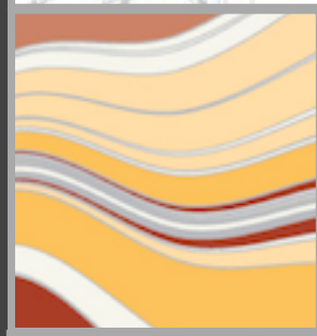
cs6630 | November 3 2014

ISOSURFACES

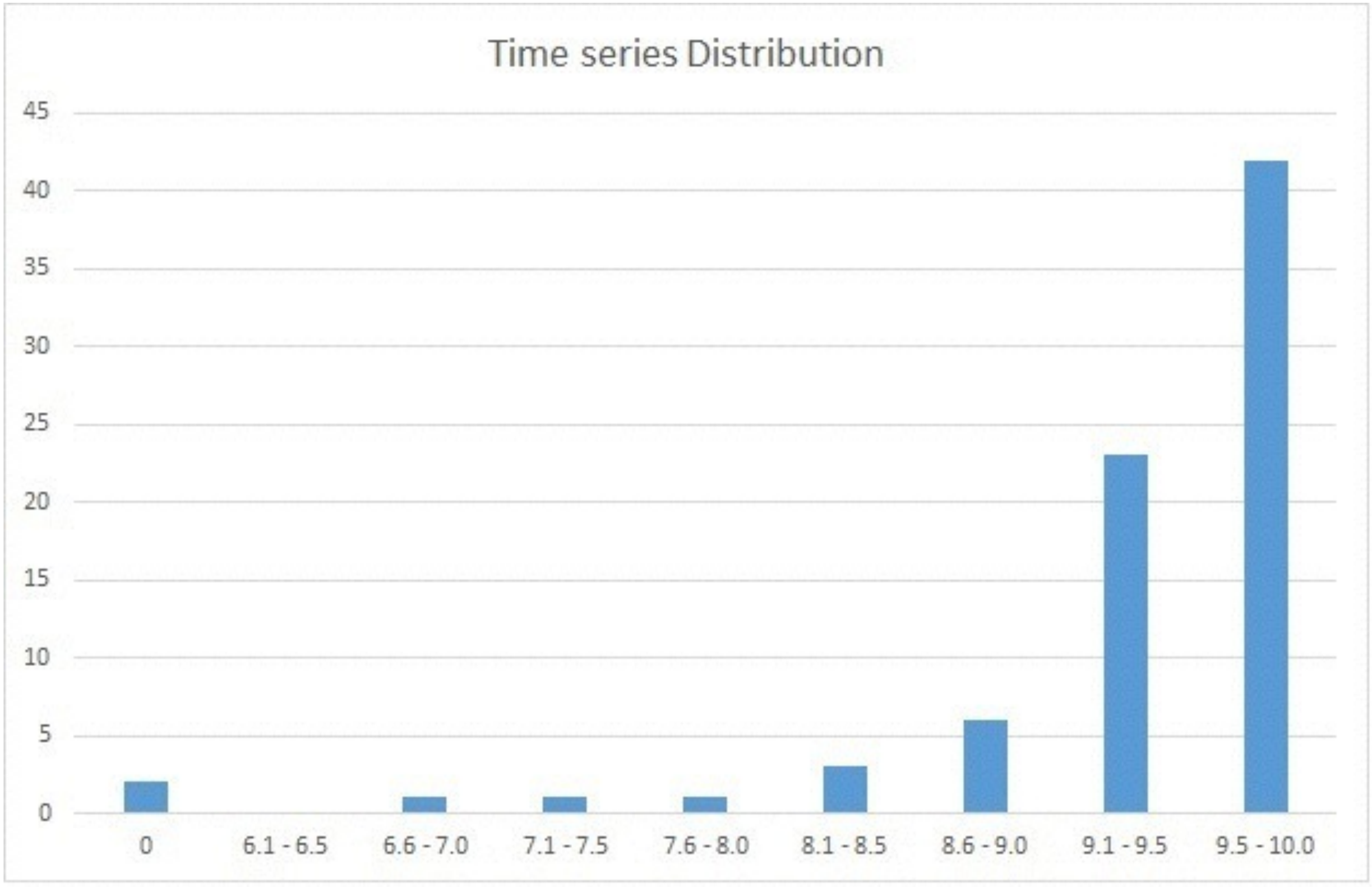
Miriah Meyer
University of Utah




administrivia . . .

- grades out for time series assignment
- scalar data assignment up; new due date
- what to do about last assignment?


Time series Distribution
45
40
35
30
25
20
15
10
5
0
0
6.1 - 6.5
6.6 - 7.0
7.1 - 7.5
7.6 - 8.0
8.1 - 8.5
8.6 - 9.0
9.1 - 9.5
9.5 - 10.0

last time . . .

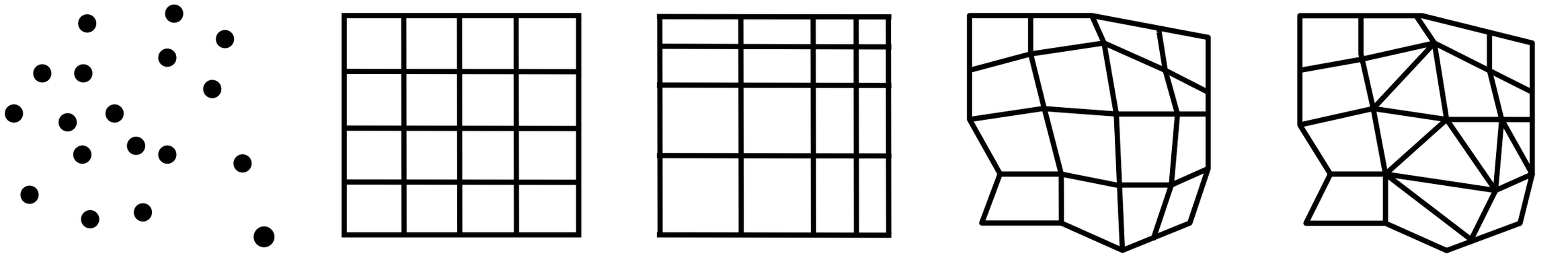
- Visualization of 1D, 2D, or 3D scalar fields
  - 1D scalar field: $\Omega \in R \rightarrow R$
  - 2D scalar field: $\Omega \in R^2 \rightarrow R$
  - 3D scalar field: $\Omega \in R^3 \rightarrow R$

    $\rightarrow$ **Volume visualization!**

Grids (Meshes)

- Meshes combine positional information (geometry) with topological information (connectivity).

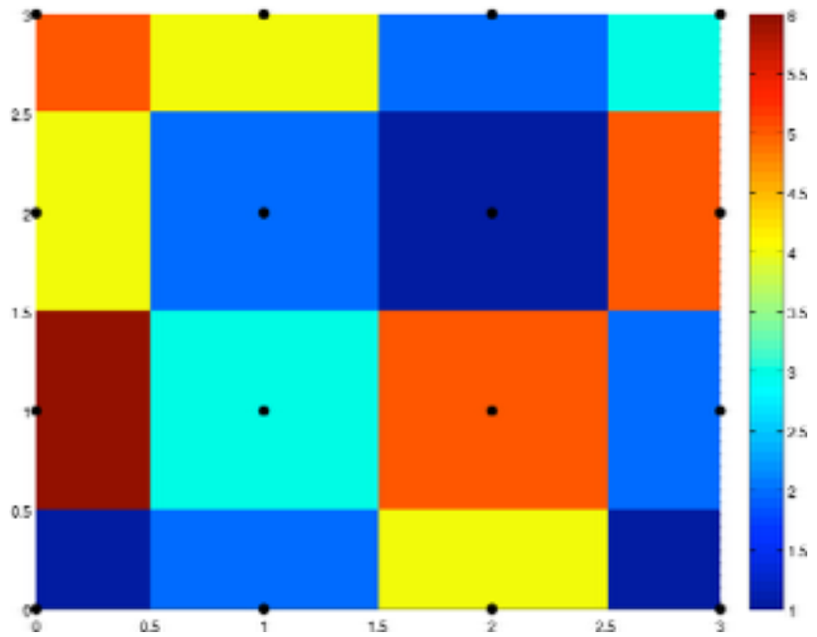
- Mesh type can differ substantial depending in the way mesh cells are formed.



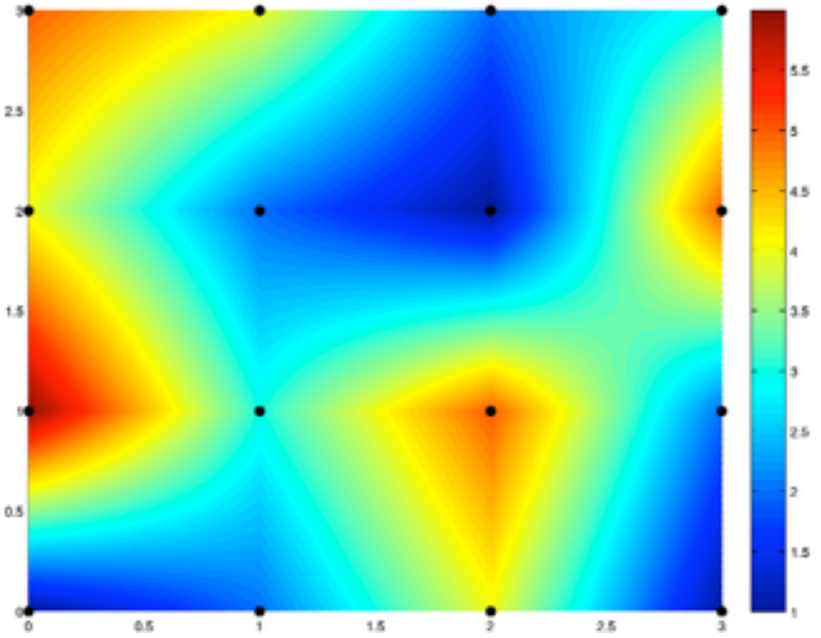
scattered uniform rectilinear structured unstructured

From Weiskopf, Machiraju, Möller

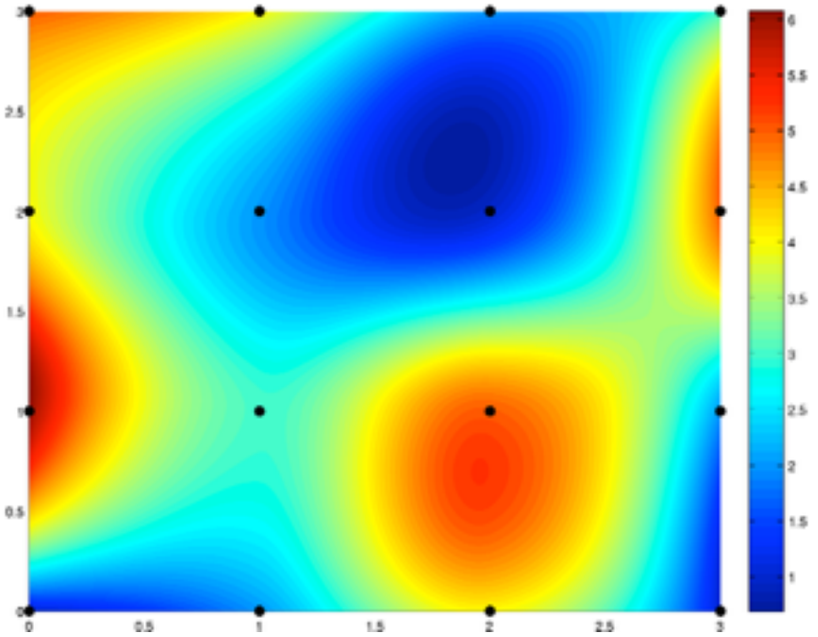
Interpolation



Nearest Neighbor



Bilinear



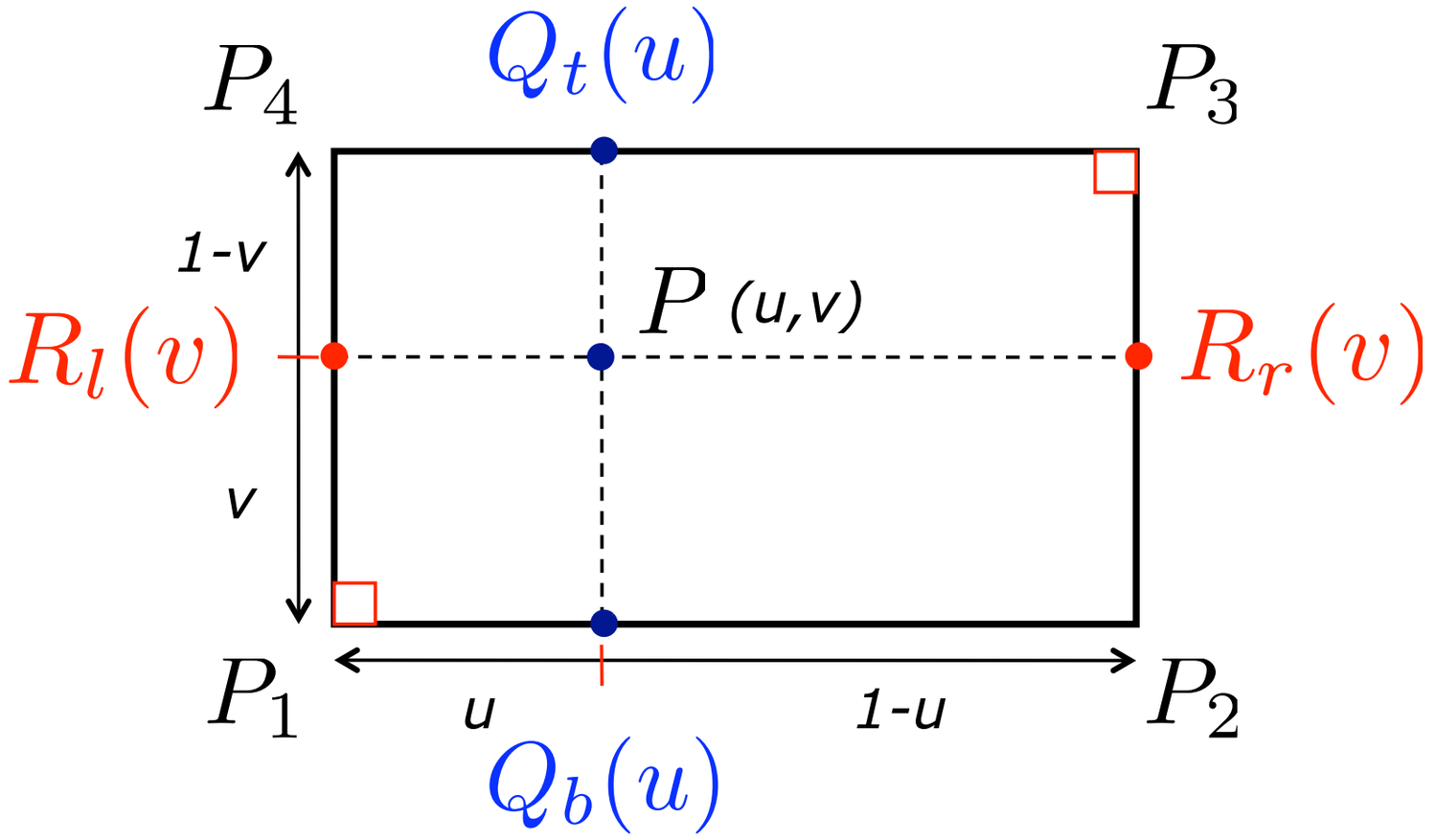
Bicubic

http://en.wikipedia.org/wiki/Bicubic_interpolation

Bilinear Interpolation

- In rectangle

$$P = (1-v)Q_b(u) + vQ_t(u)$$
$$= (1-u)R_l(v) + uR_r(v)$$



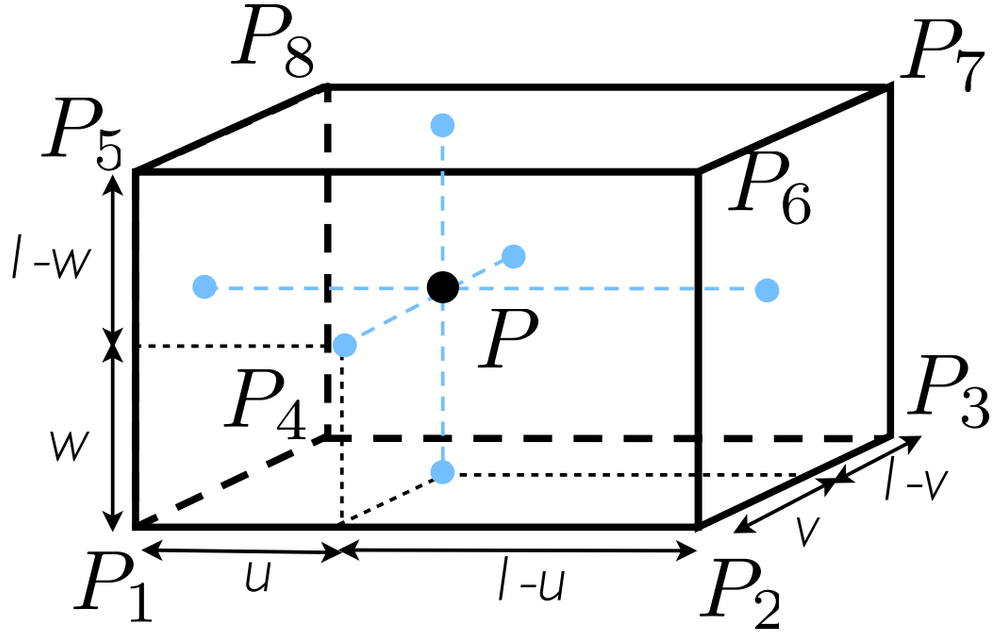
Trilinear Interpolation

- In a cuboid (axis parallel)
  - general formula

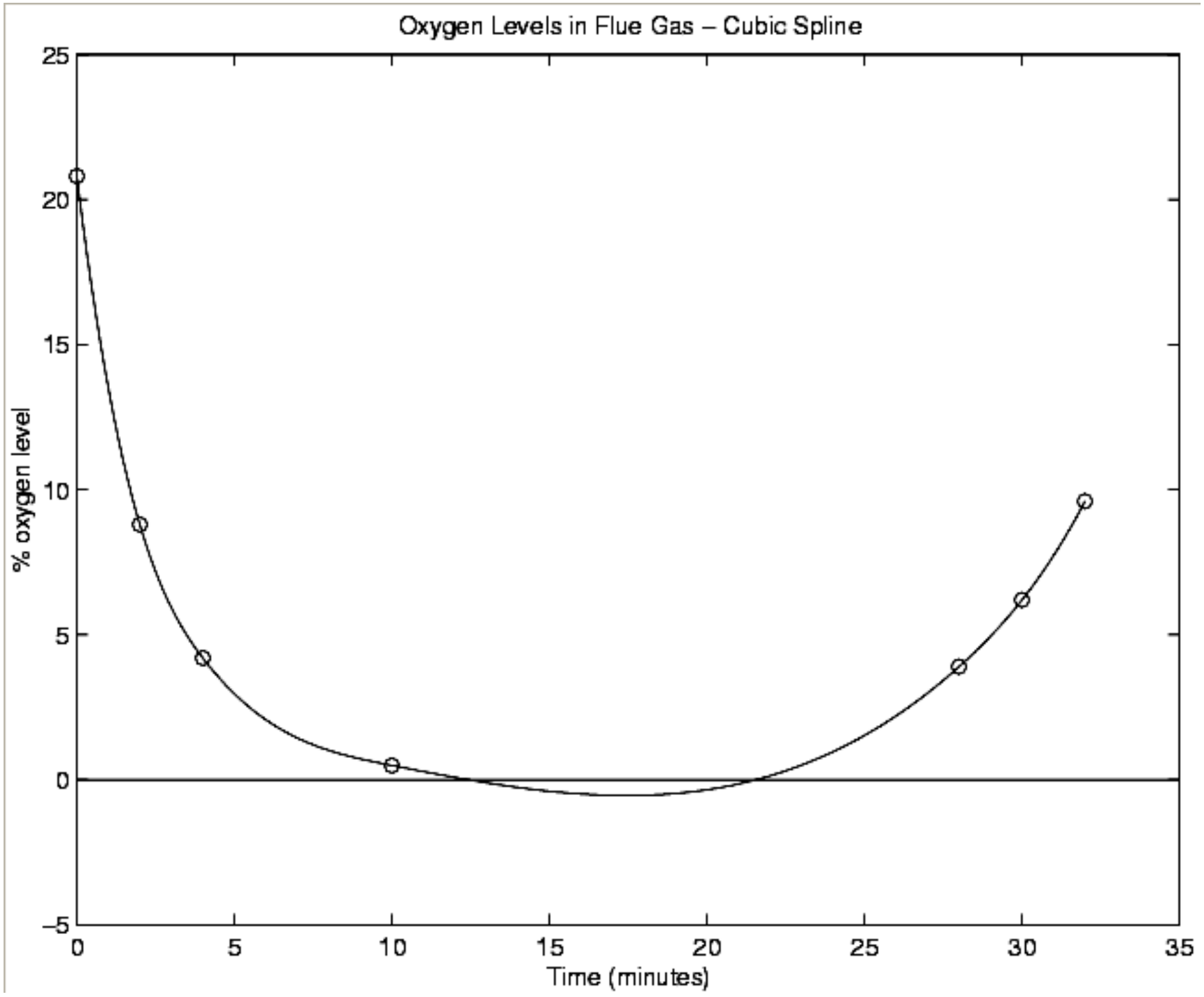
$$\phi(x, y, z) = axyz + bxy + cxz + dyz + ex + fy + gz + h$$

  - with local coordinates

$$\begin{aligned} P = P_1 \\ &+u(P_2 - P_1) \\ &+v(P_4 - P_1) \\ &+w(P_5 - P_1) \\ &+uv(P_1 - P_2 + P_3 - P_4) \\ &+uw(P_1 - P_2 + P_6 - P_5) \\ &+vw(P_1 - P_4 + P_8 - P_5) \\ &+uvw(P_1 - P_2 + P_3 - P_4 + P_5 - P_6 + P_7 - P_8) \end{aligned}$$

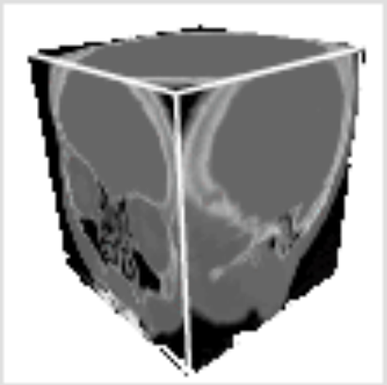


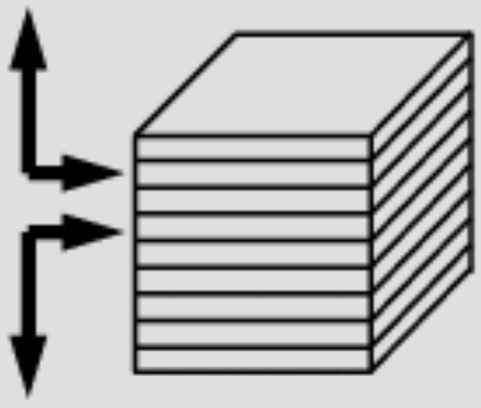
What is “Correct” Interpolation?

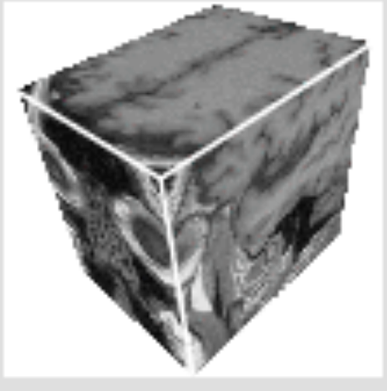


today . . .

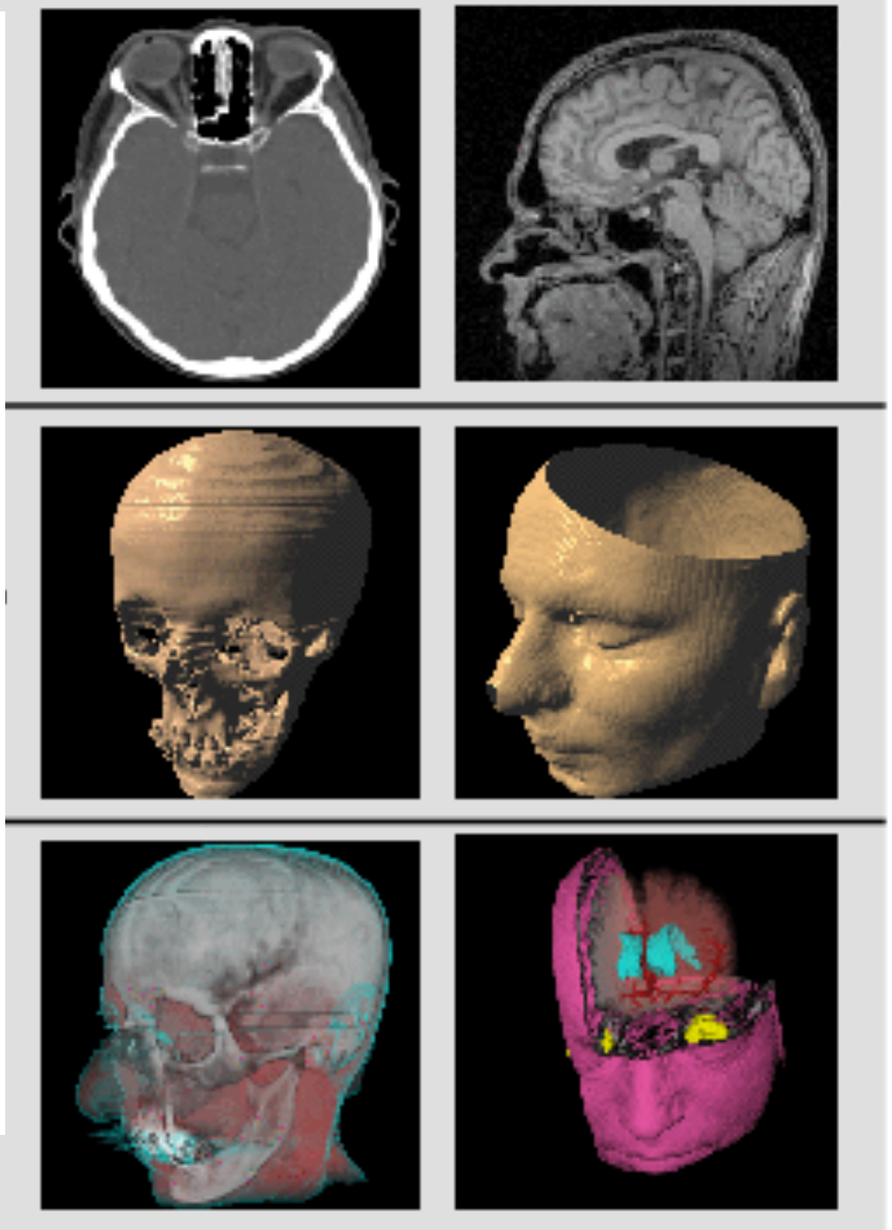
- overview of visualization for 2D scalar fields
- isocurves with marching squares
- isosurfacing with marching cubes

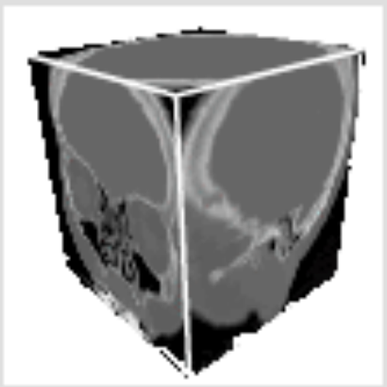


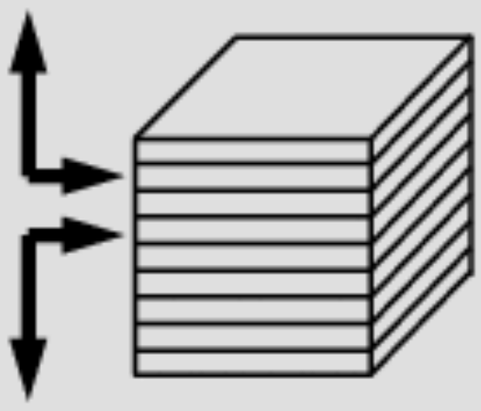


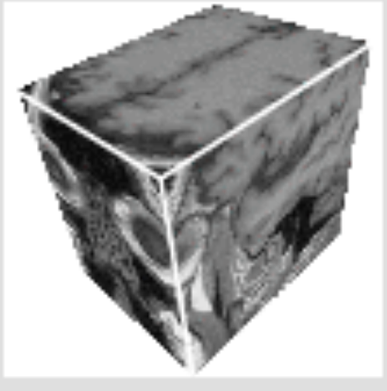


- 2D visualization
  slice images
  (or multi-planar
  reformating MPR)
- *Indirect*
  3D visualization
  isosurfaces
  (or surface-shaded
  display SSD)
- *Direct*
  3D visualization
  (direct volume
  rendering DVR)

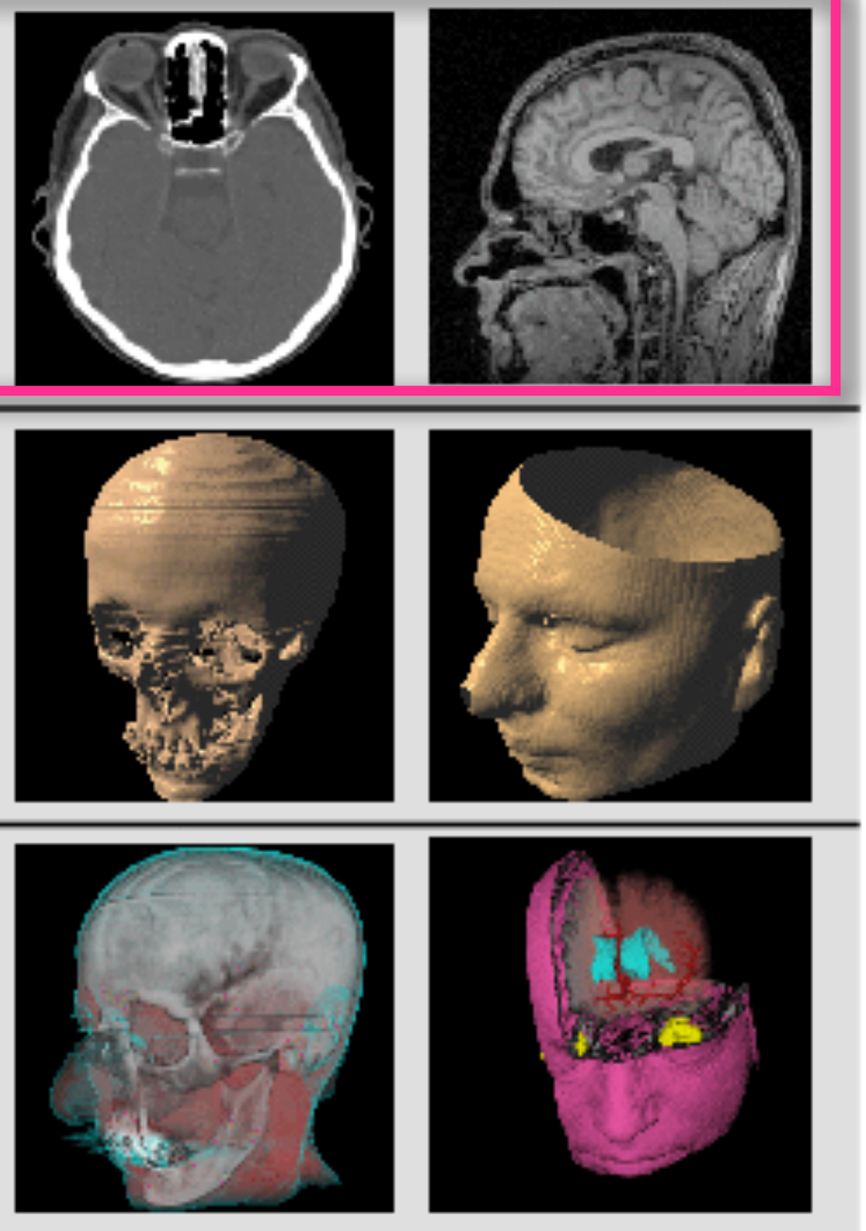








- 2D visualization
  slice images
  (or multi-planar
  reformating MPR)
- *Indirect*
  3D visualization
  isosurfaces
  (or surface-shaded
  display SSD)
- *Direct*
  3D visualization
  (direct volume
  rendering DVR)



Techniques for 2D Scalar Field Vis

- Geometry-based:
  - Height fields, surface plots
  - Contours
- Color-based
  - Transfer Function / LUT Selection

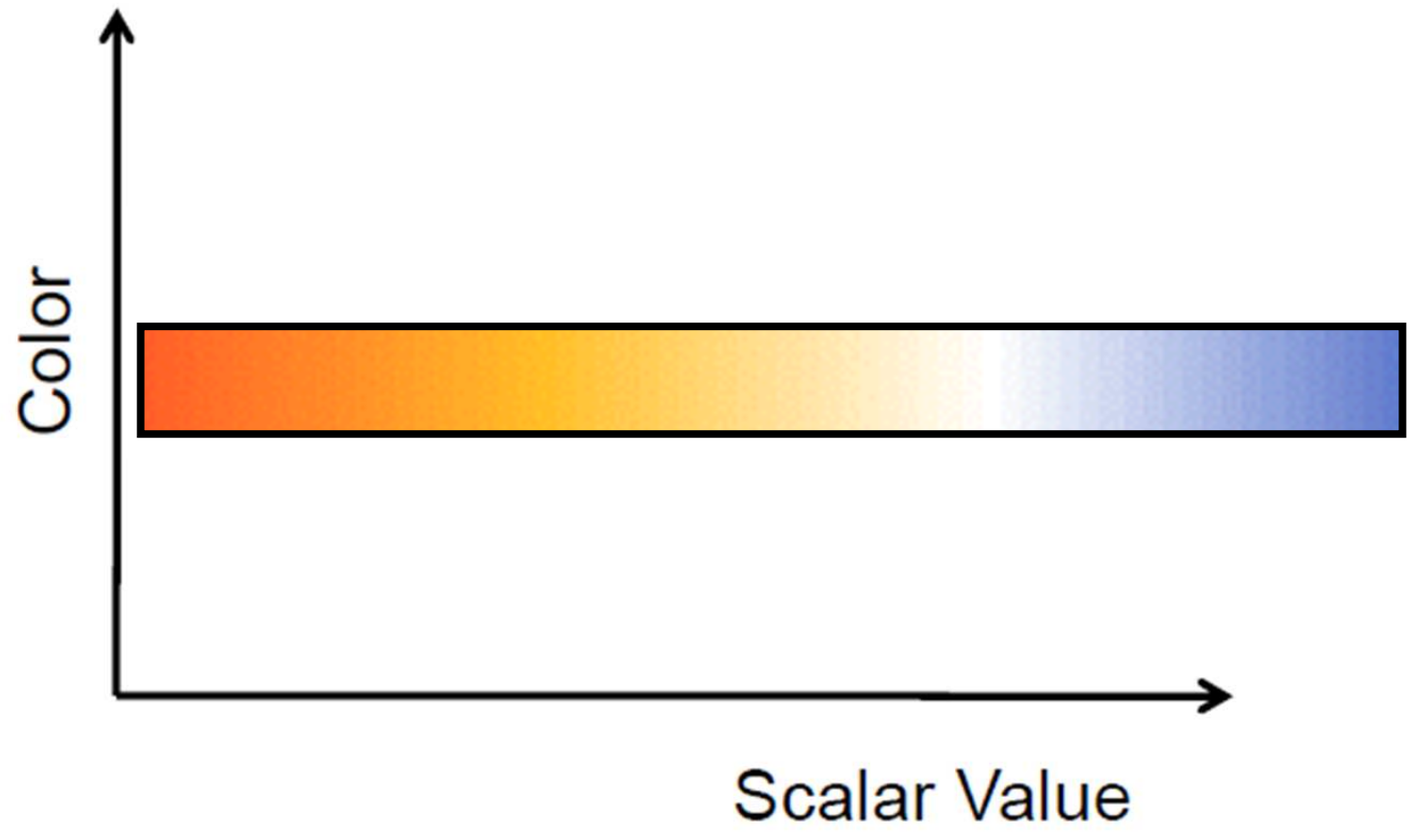
Color Mapping

- Display scalar value through a color map, transfer function, color scale, or lookup table (LUT)

- Map interval on the real line to a path through the color space.

$$f : R \rightarrow \{\text{RGB}, \text{HSV}\}$$

In Visualization, we Use the Concept of a ***Transfer Function*** to set Color as a Function of Scalar Value

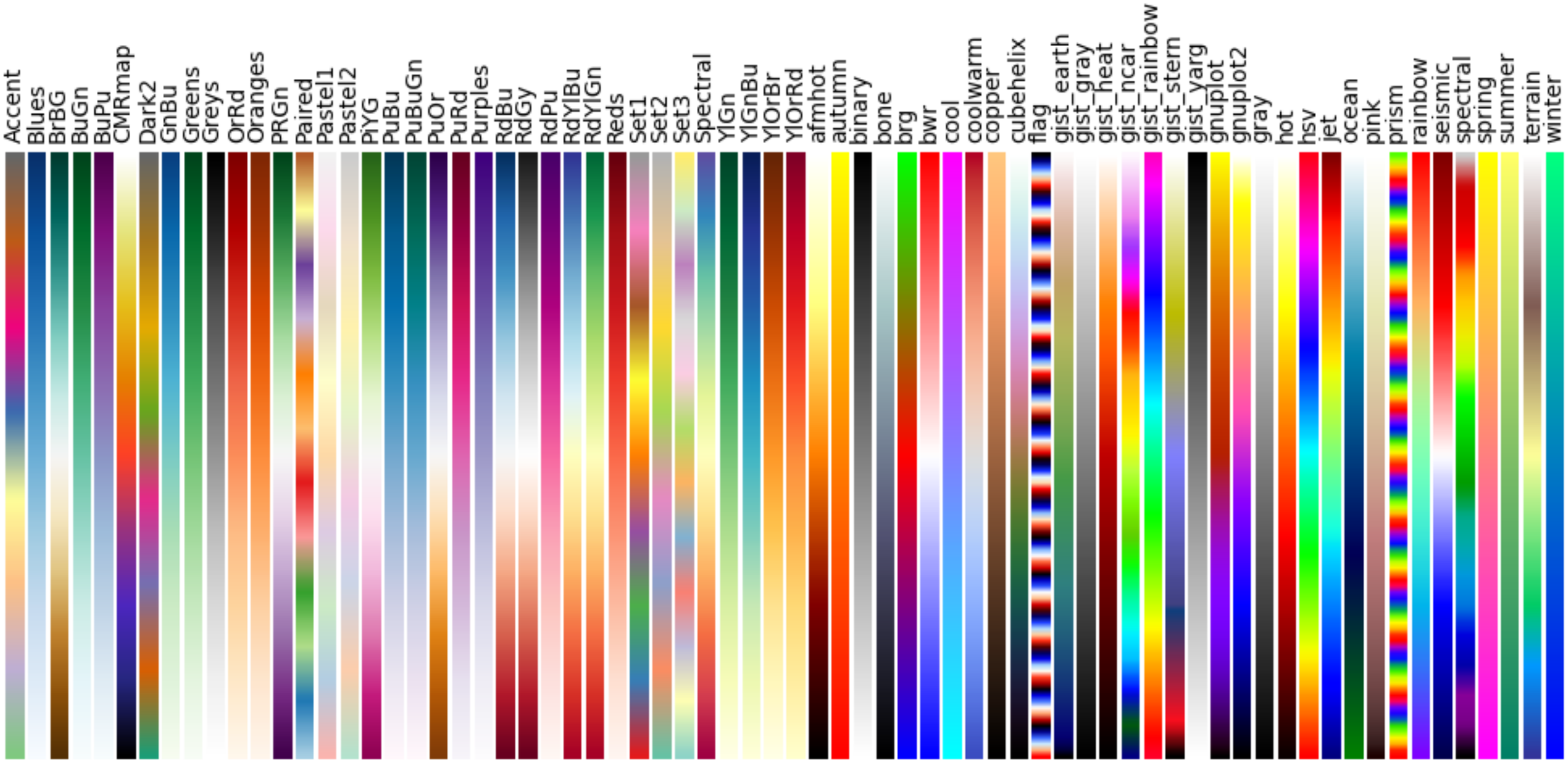


Scalar values ->[0,1] -> Colors

Use the Right Transfer Function Color Scale to Represent a Range of Scalar Values

- Gray scale
- Intensity Interpolation
- Saturation interpolation
- Two-color interpolation
- Rainbow scale
- Heated object interpolation
- Blue-White-Red

A Gallery of Color Scales



- Example
  - Special color table to visualize the brain tissue
  - Special color table to visualize the bone structure

Original

Brain

Tissue

More Examples

Penny Rheingans (1999). Task-based Color Scale Design. Proceedings of Applied Image and Pattern Recognition '99, SPIE, pp. 35-43.

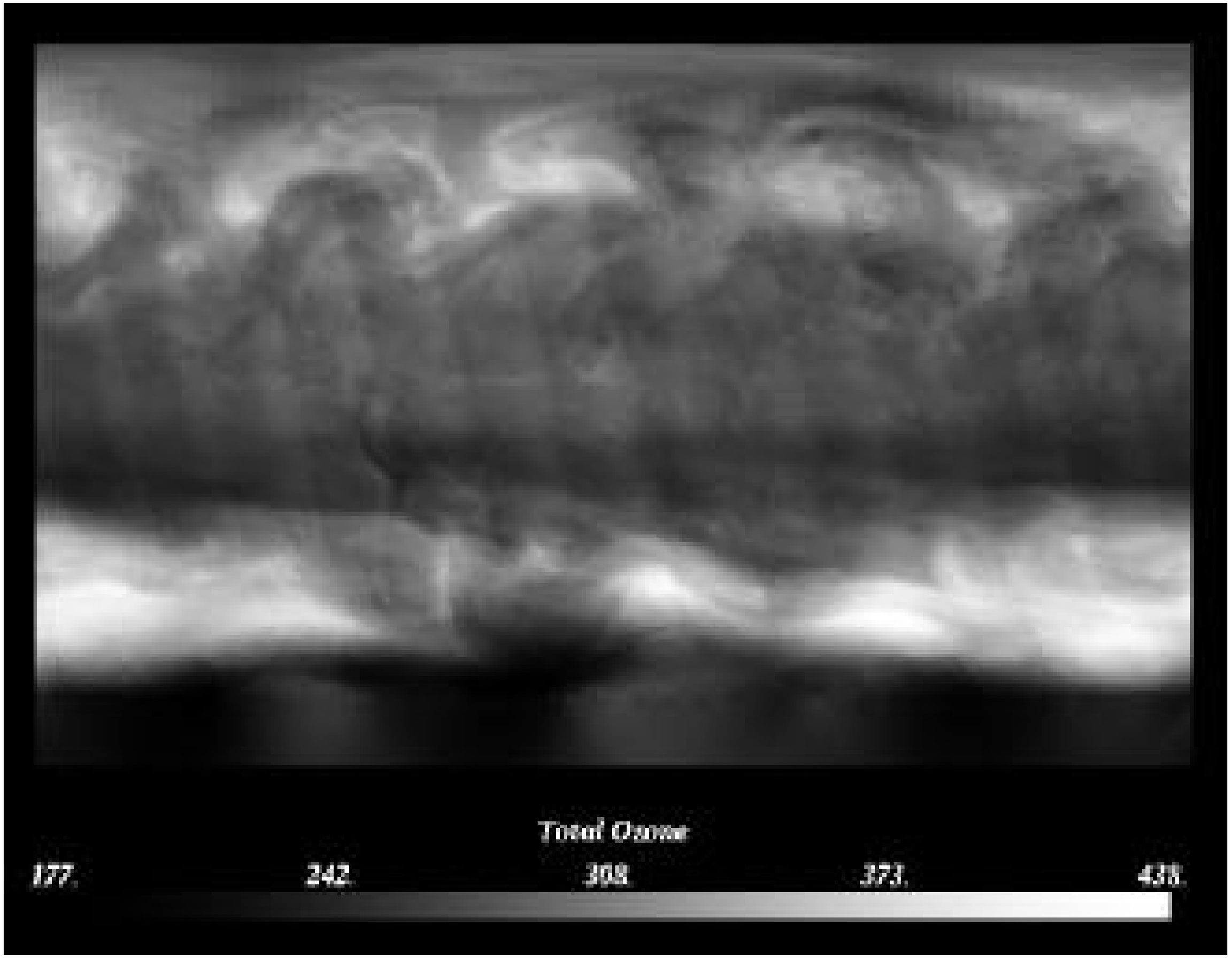


Figure 1. Grey scale.

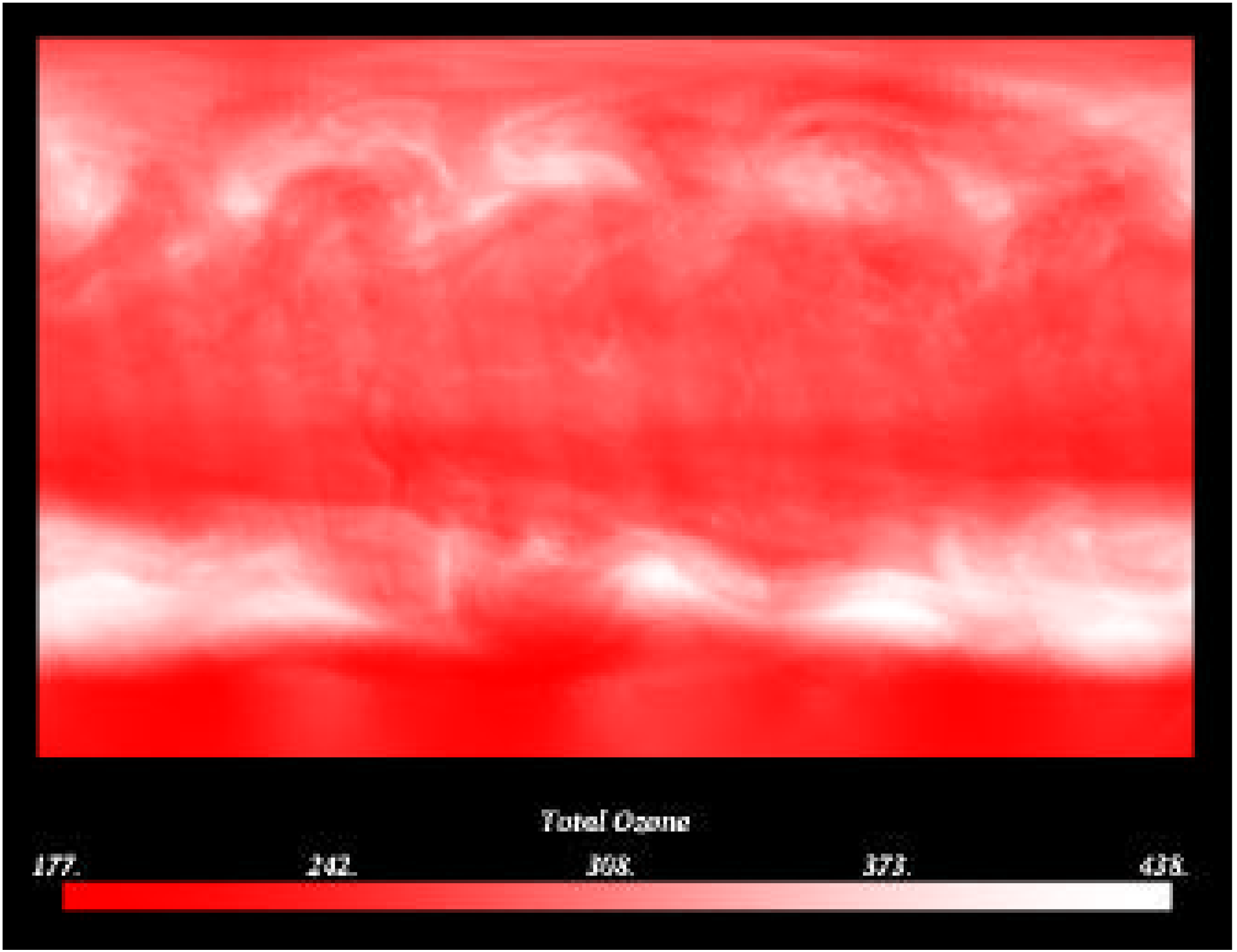


Figure 2. Saturation scale.

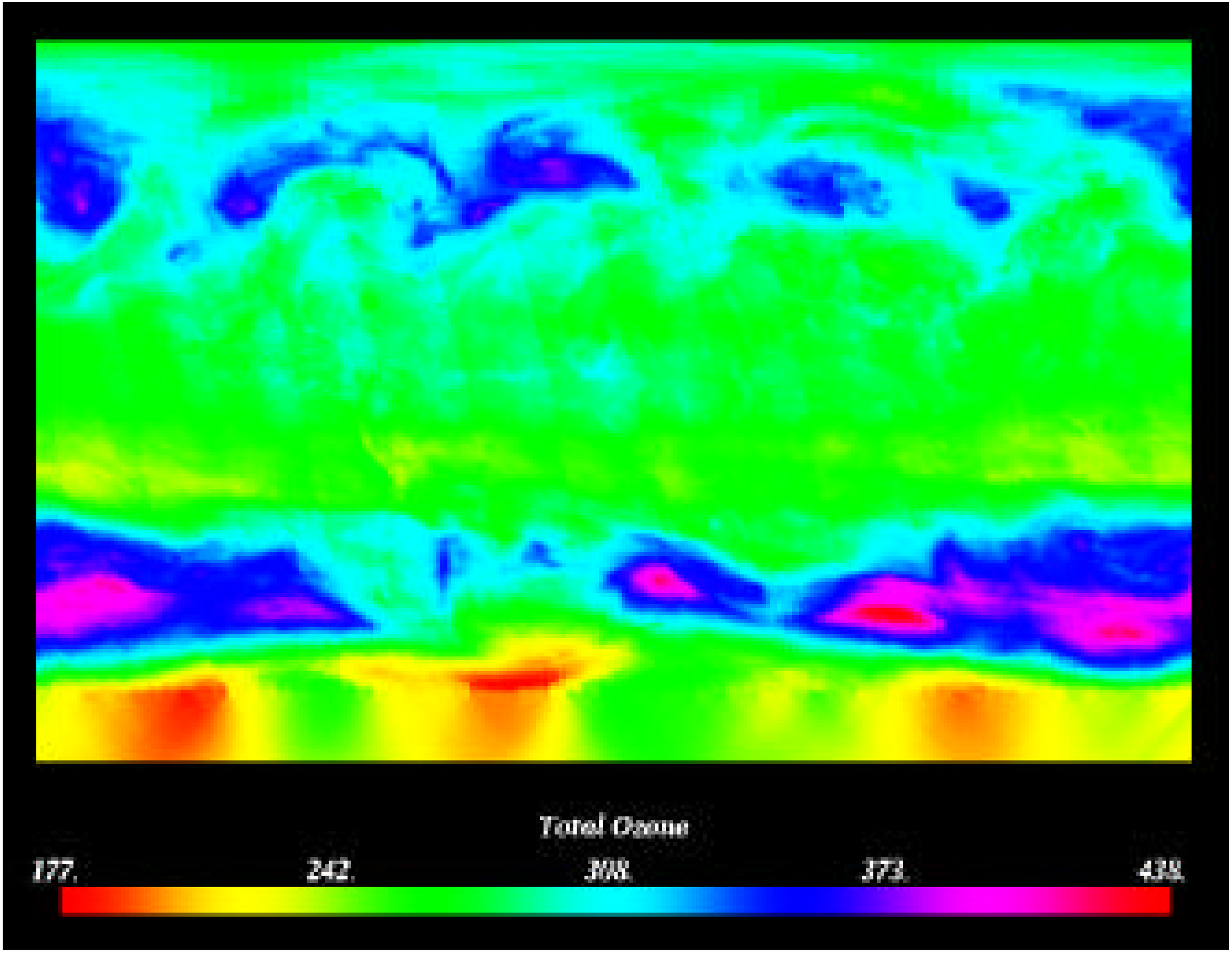


Figure 3. Spectrum scale.

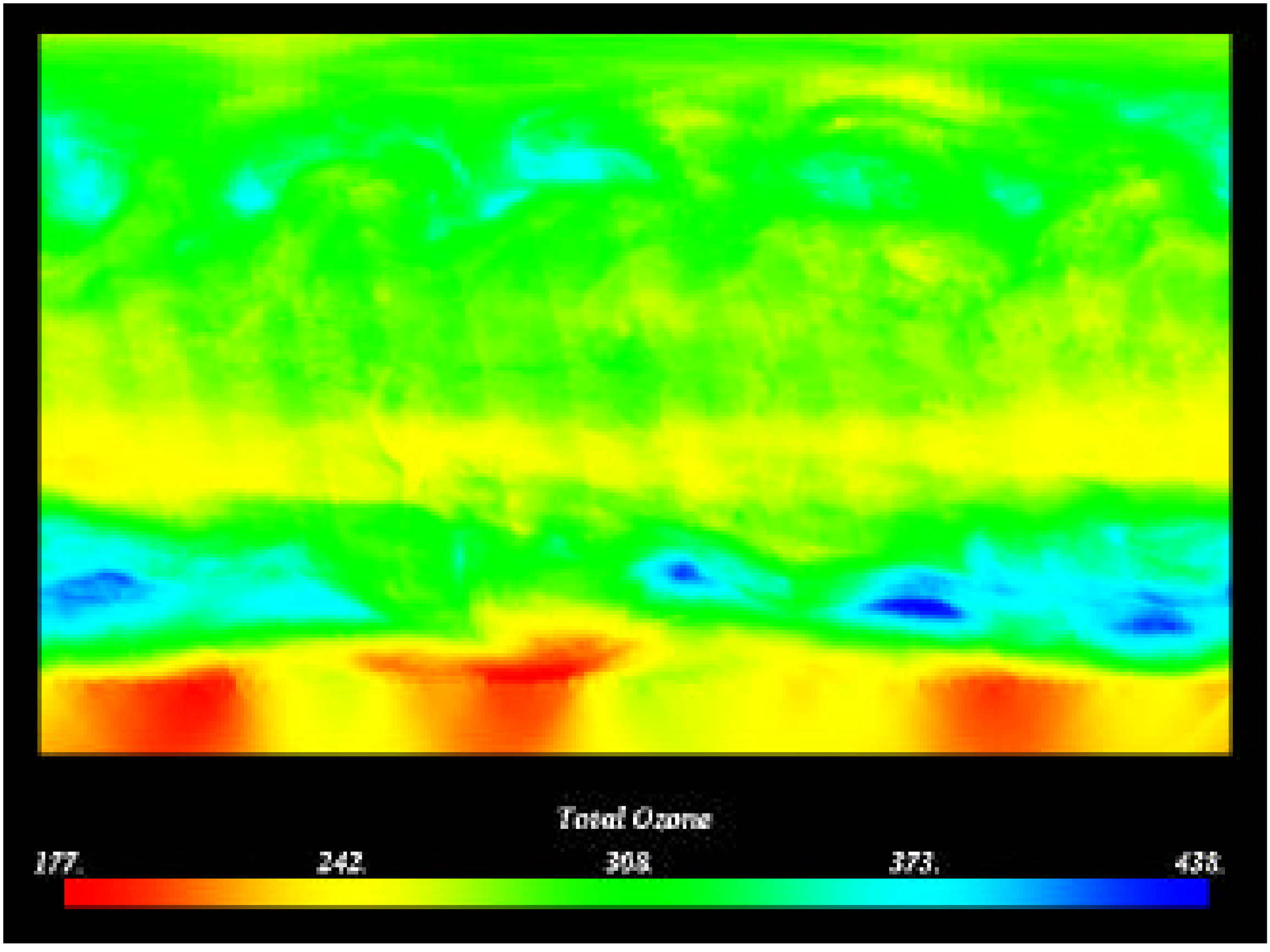


Figure 4. Limited spectrum scale.

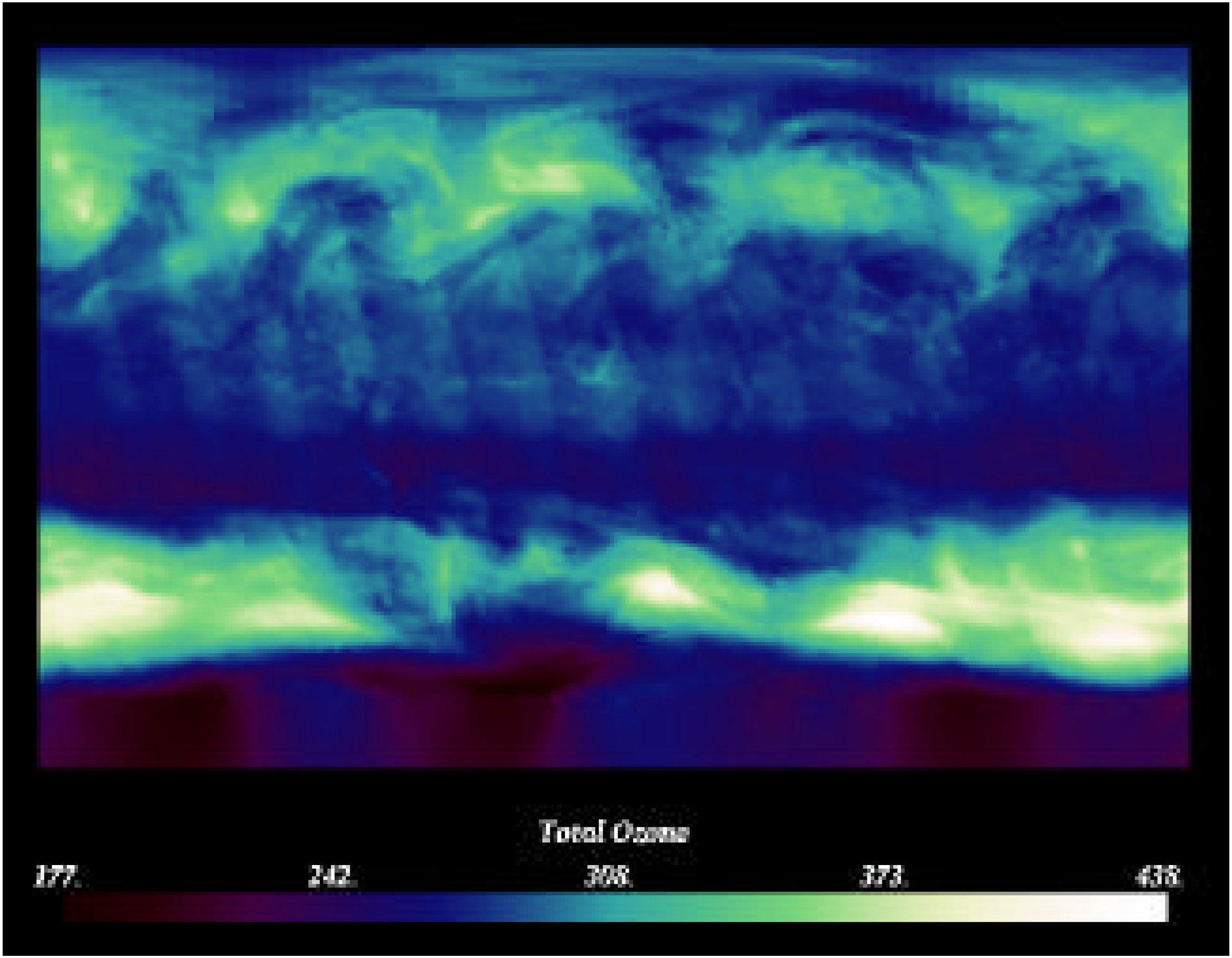


Figure 5. Redundant hue/lightness scale.

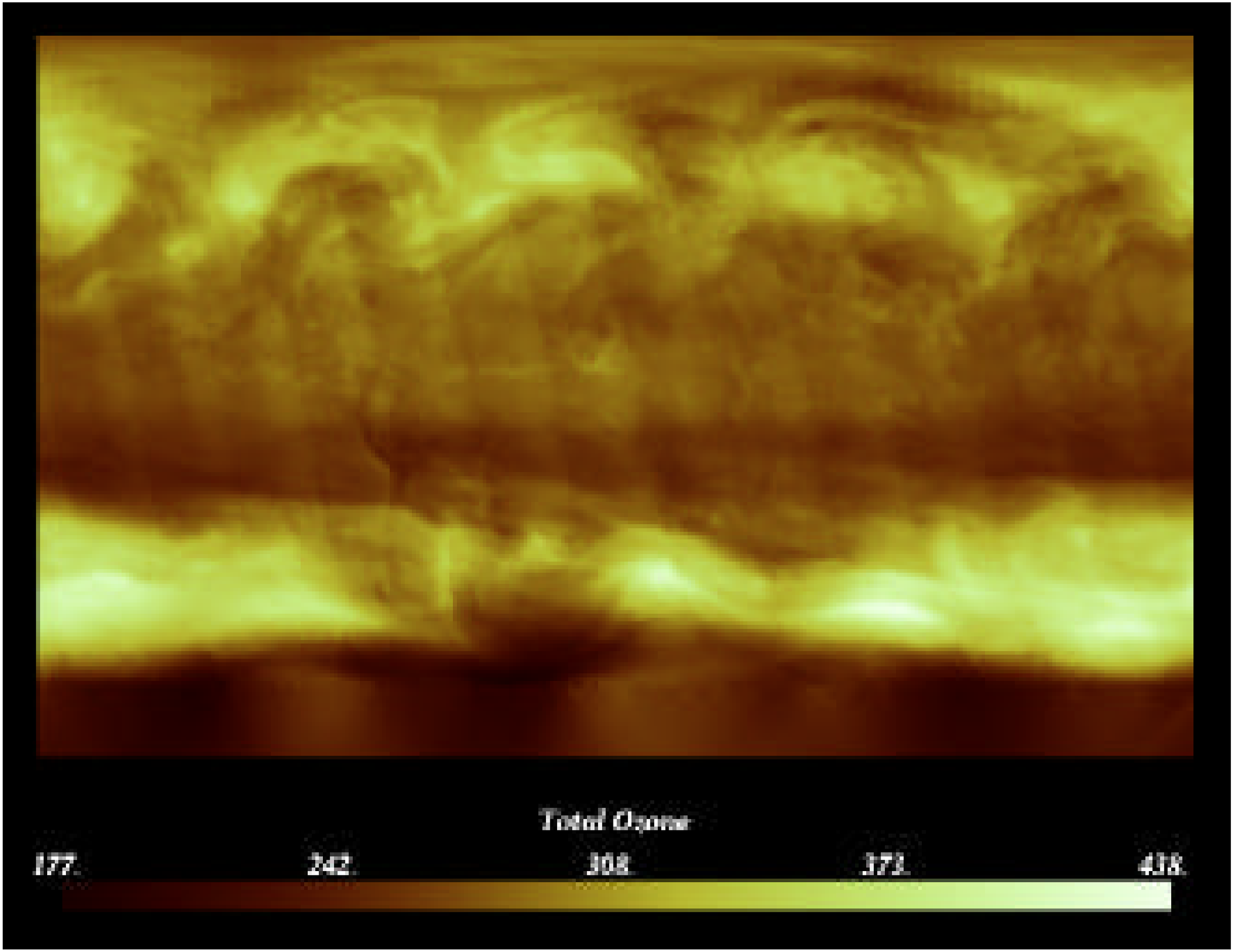
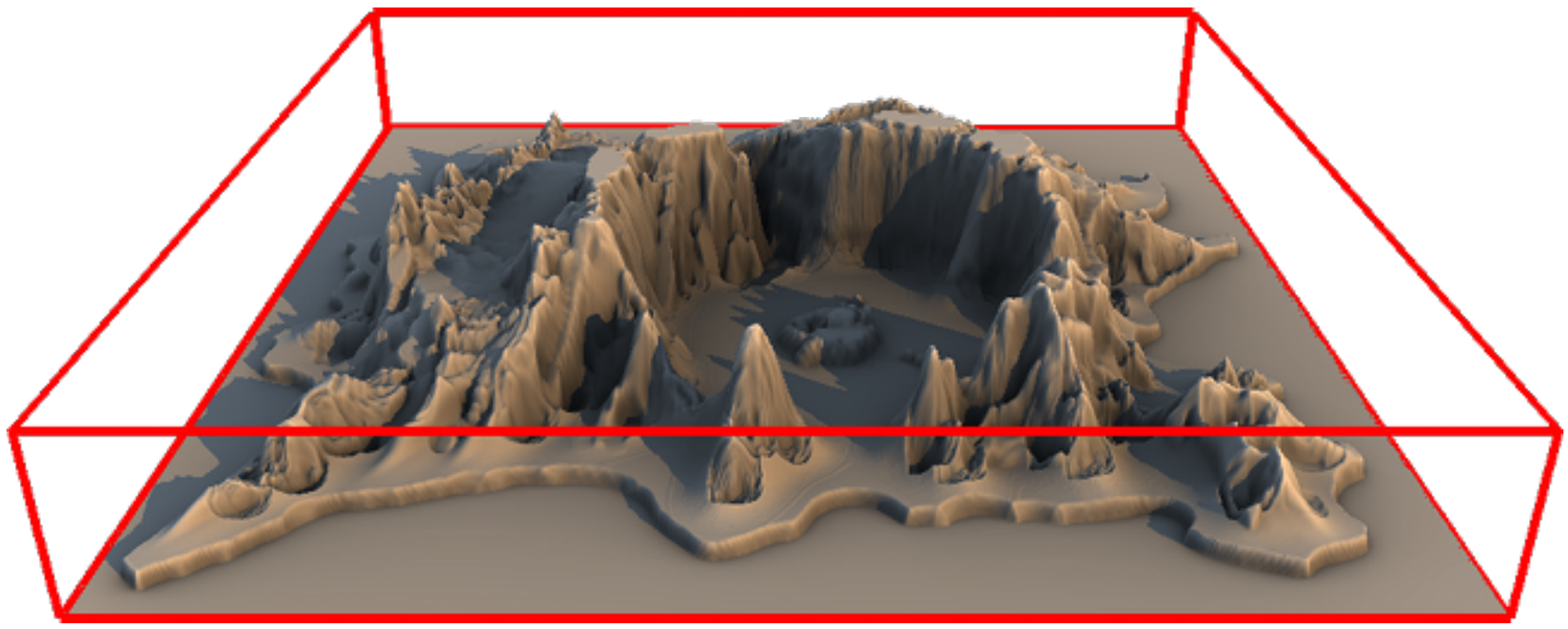
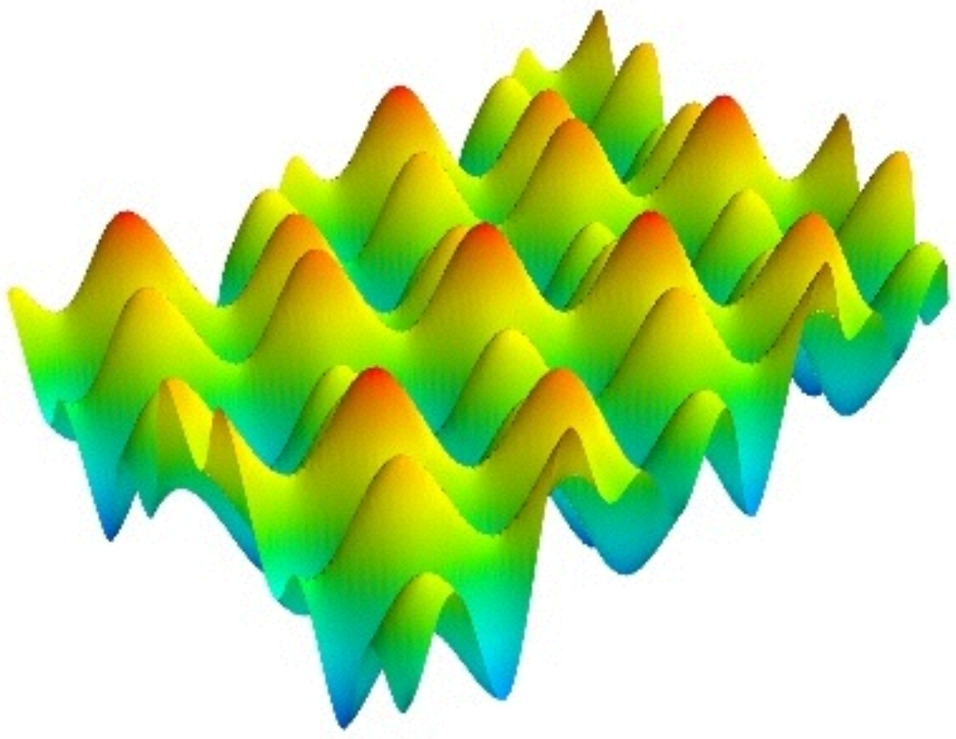


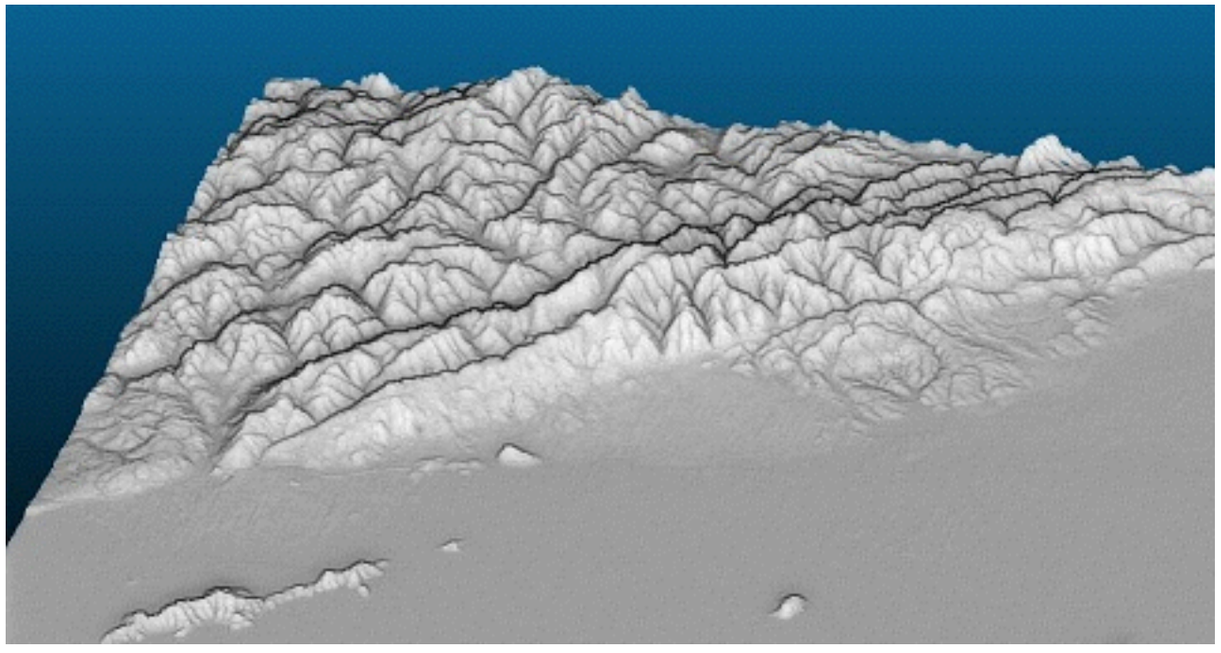
Figure 6. Heated-object scale.

Height Fields

- We use height in 1D plots, let's use it in 2D plots
- Direct intuition of the topography
- Let the geometry convey the data

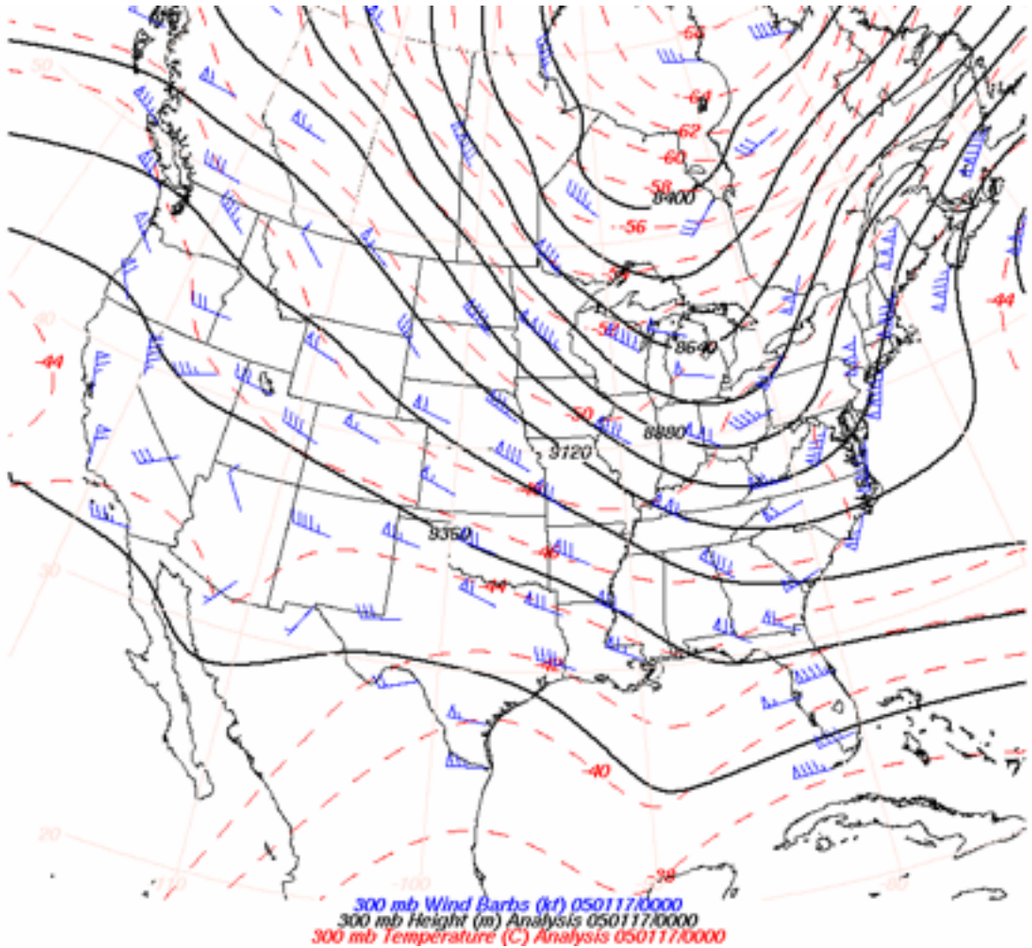


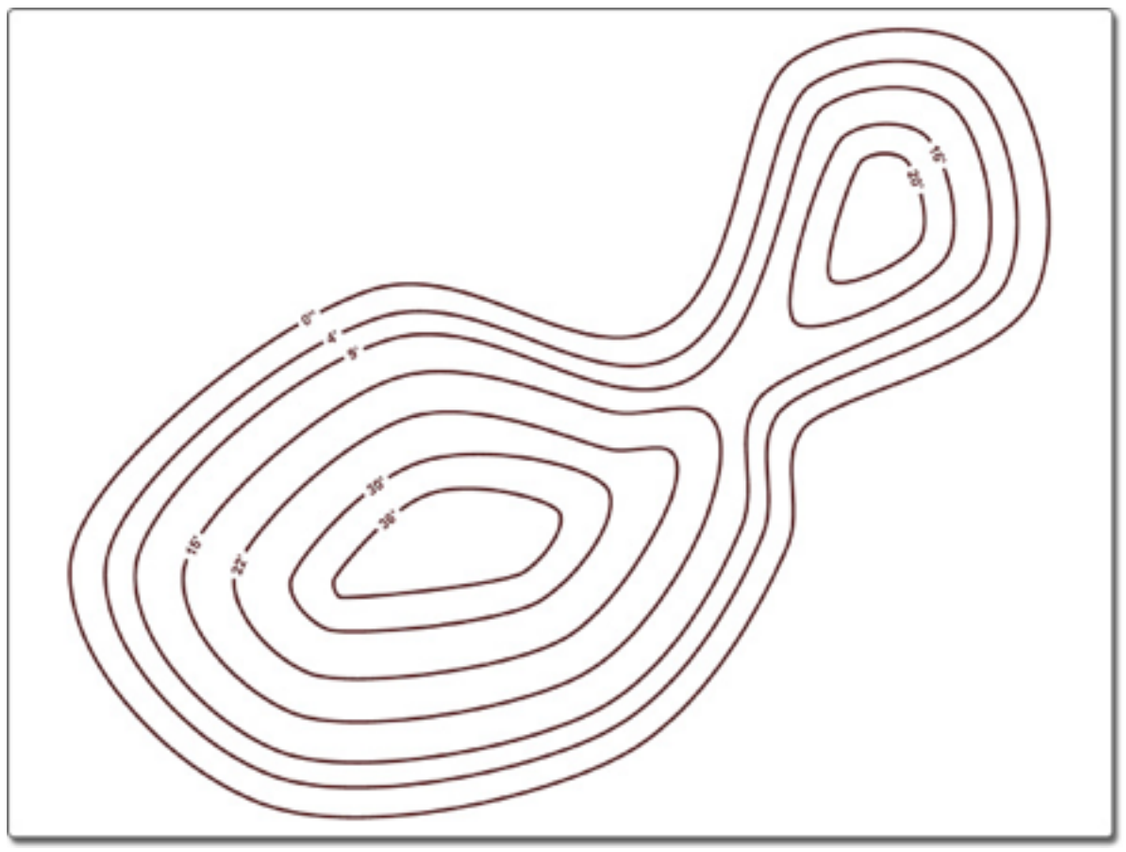




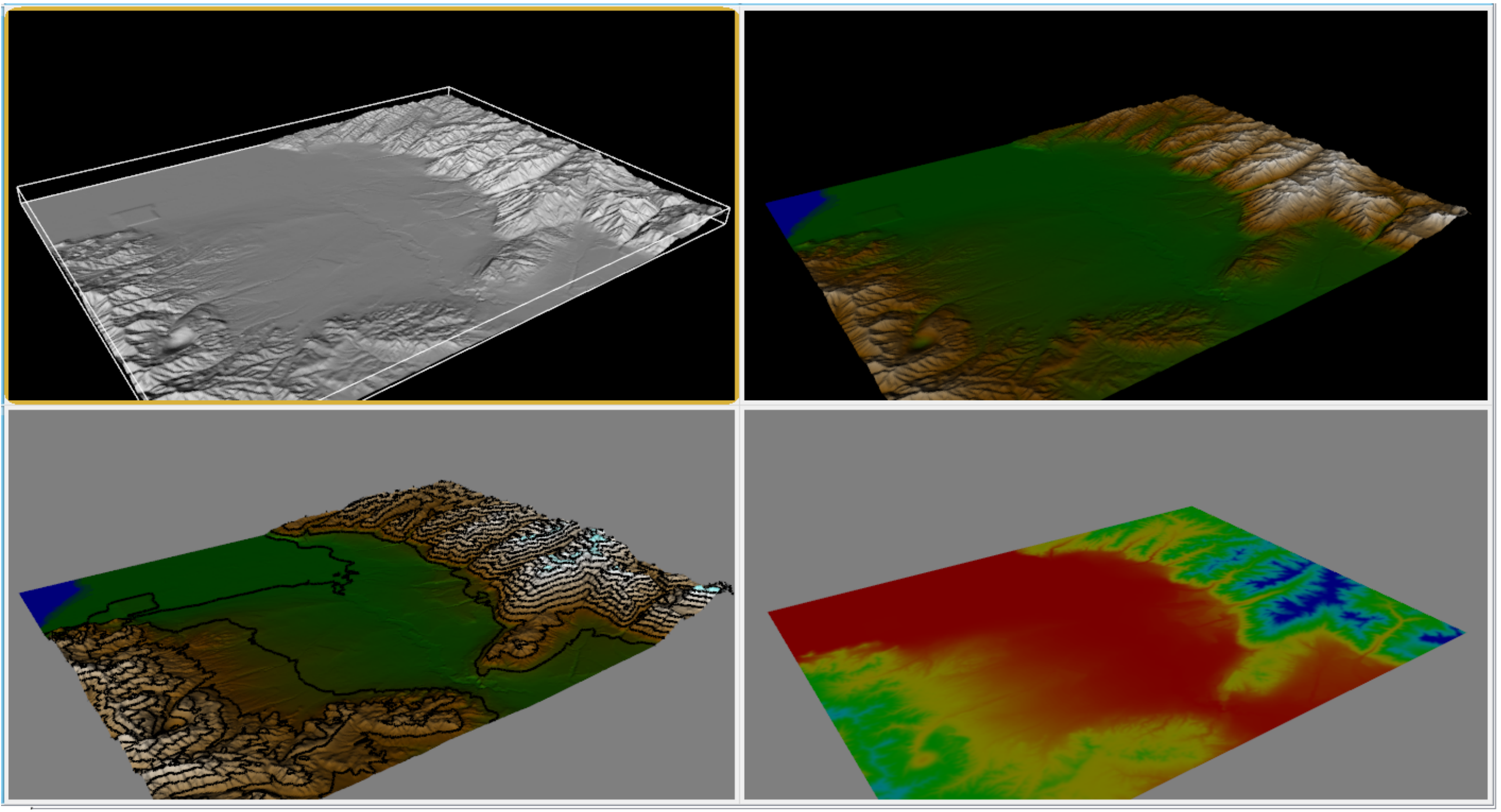
Contour Lines

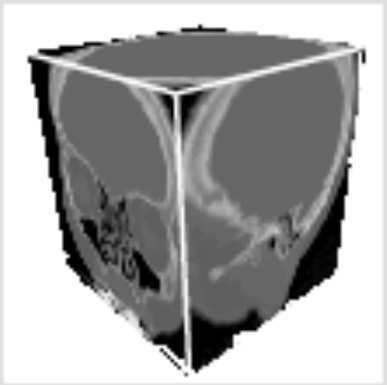
- Draw lines of constant value.
- These bound regions of contiguous hues
    - Loops or lines through end of the dataset
- Usually best to use multiple contours
    - Why?

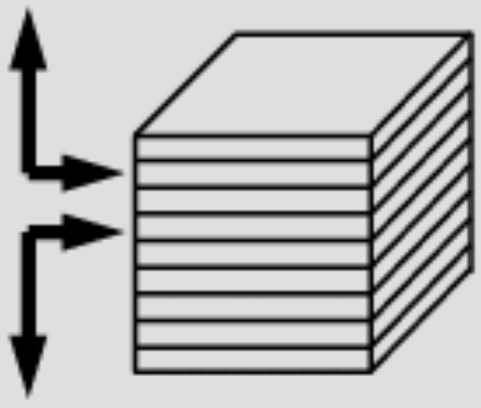

300 mb Wind Barbs (kt) 050117/0000
300 mb Height (m) Analysis 050117/0000
300 mb Temperature (C) Analysis 050117/0000

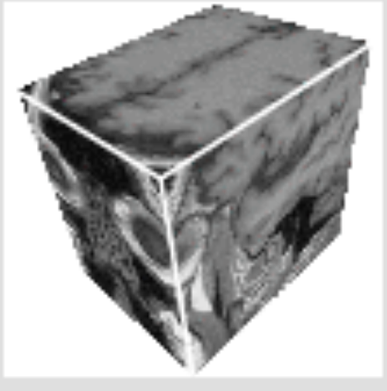


Compare

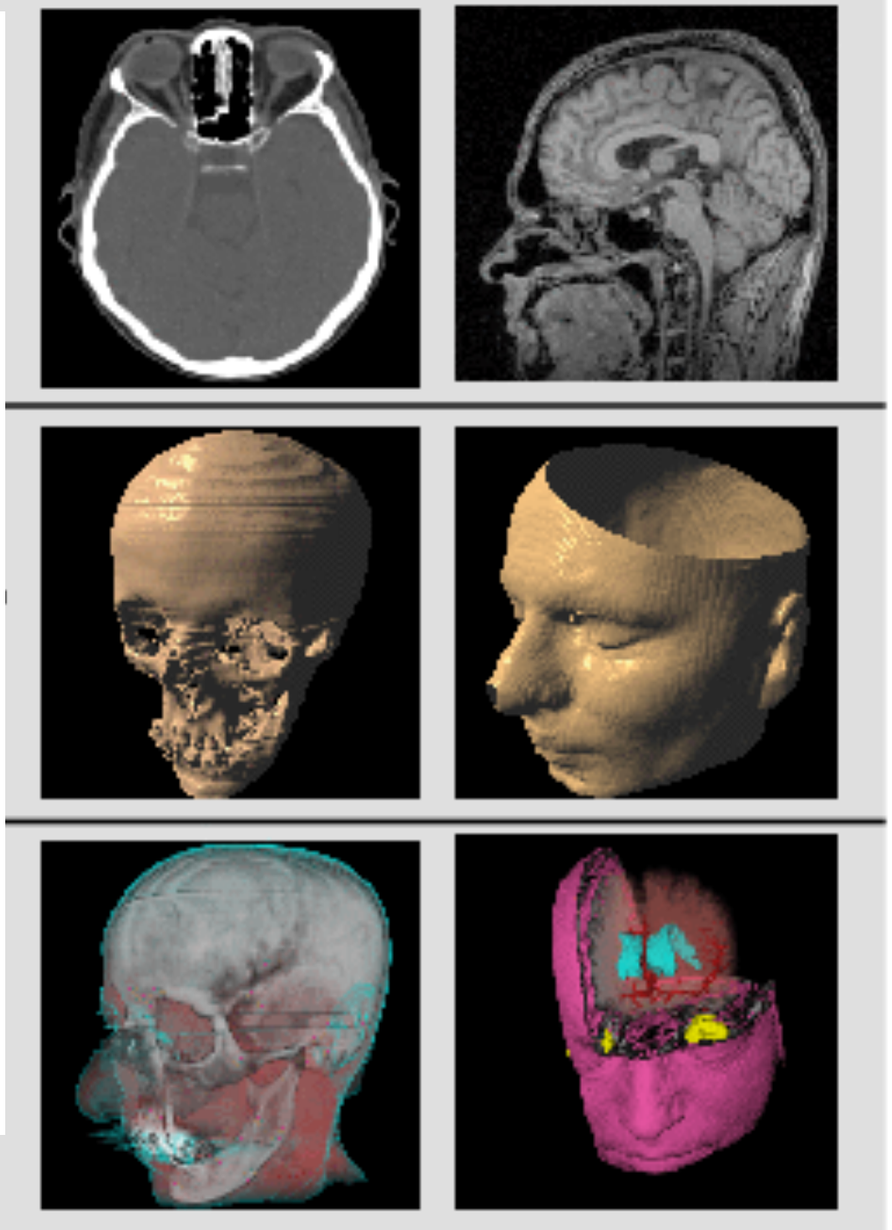


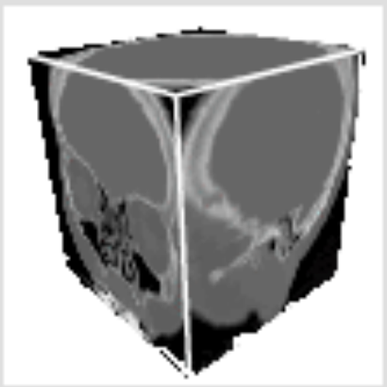


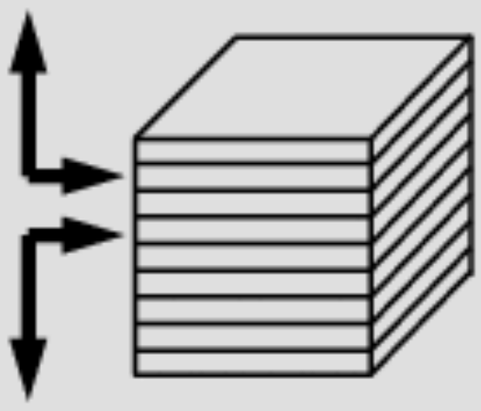


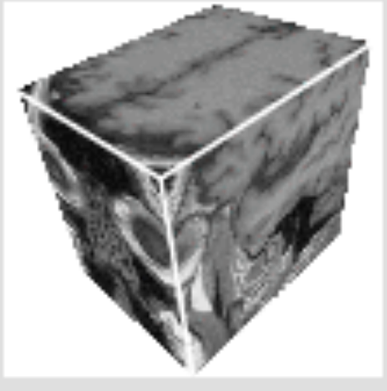


- 2D visualization
  slice images
  (or multi-planar
  reformating MPR)
- *Indirect*
  3D visualization
  isosurfaces
  (or surface-shaded
  display SSD)
- *Direct*
  3D visualization
  (direct volume
  rendering DVR)

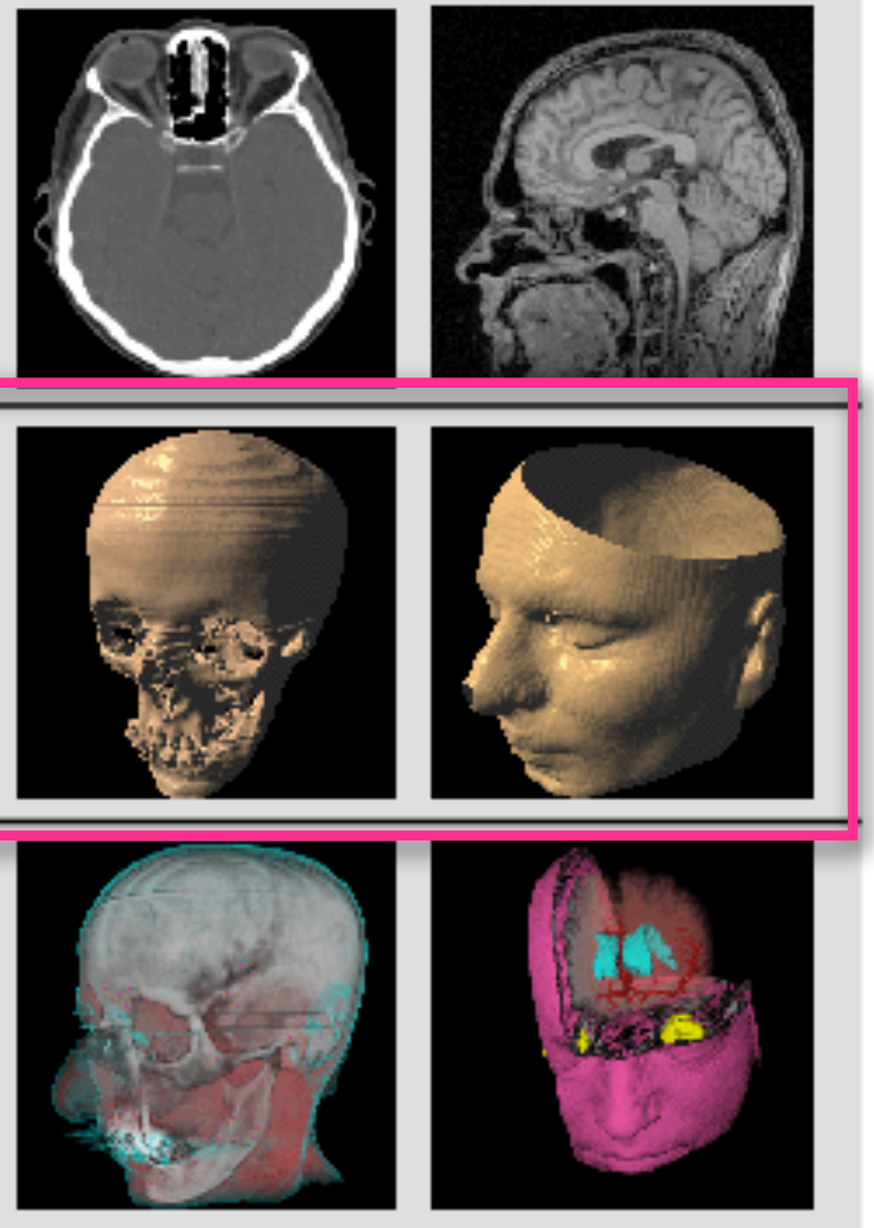




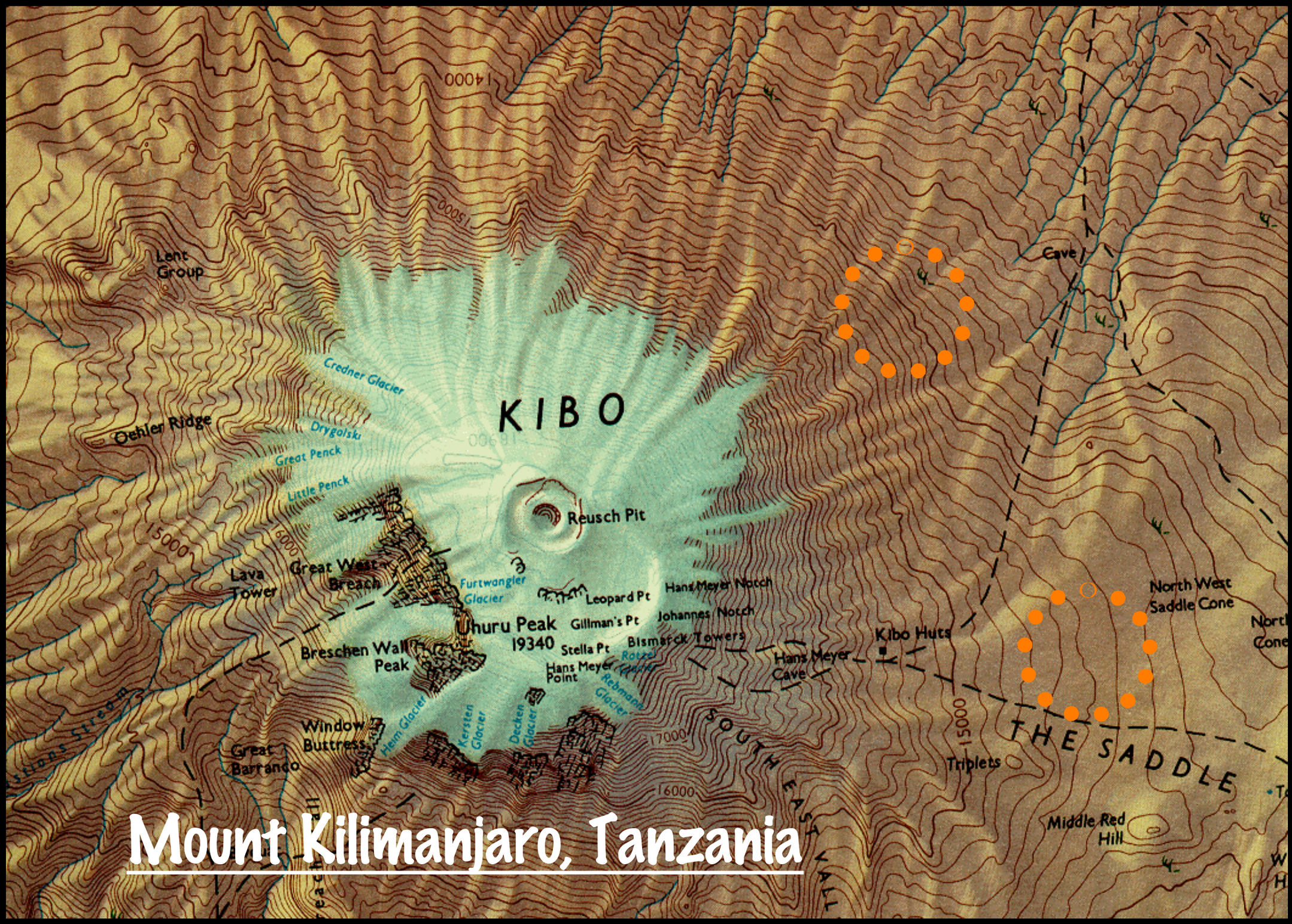




- 2D visualization
  slice images
  (or multi-planar
  reformating MPR)
- *Indirect*
  3D visualization
  isosurfaces
  (or surface-shaded
  display SSD)
- *Direct*
  3D visualization
  (direct volume
  rendering DVR)

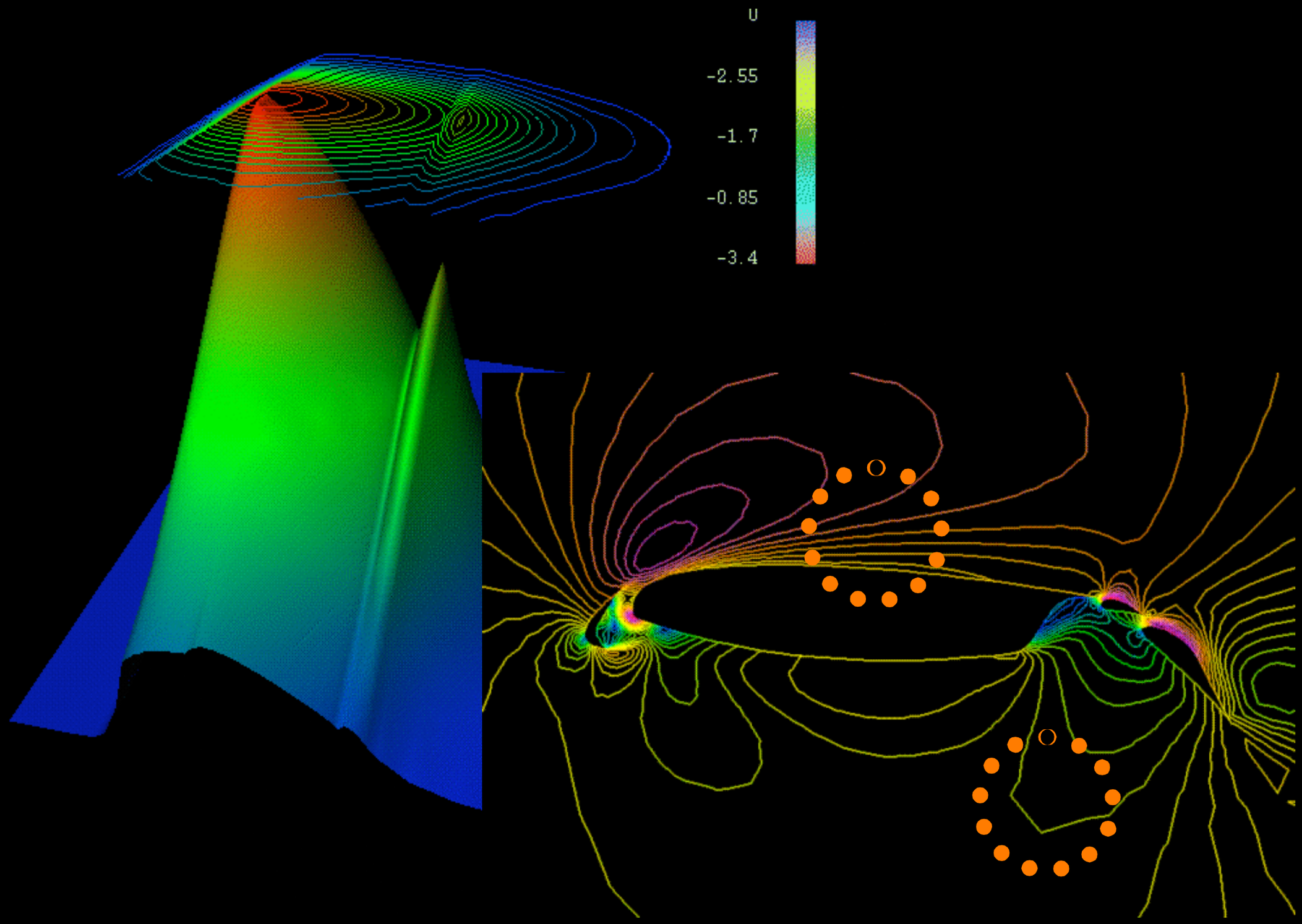


http://www.lib.berkeley.edu/EART/digital/topo.html

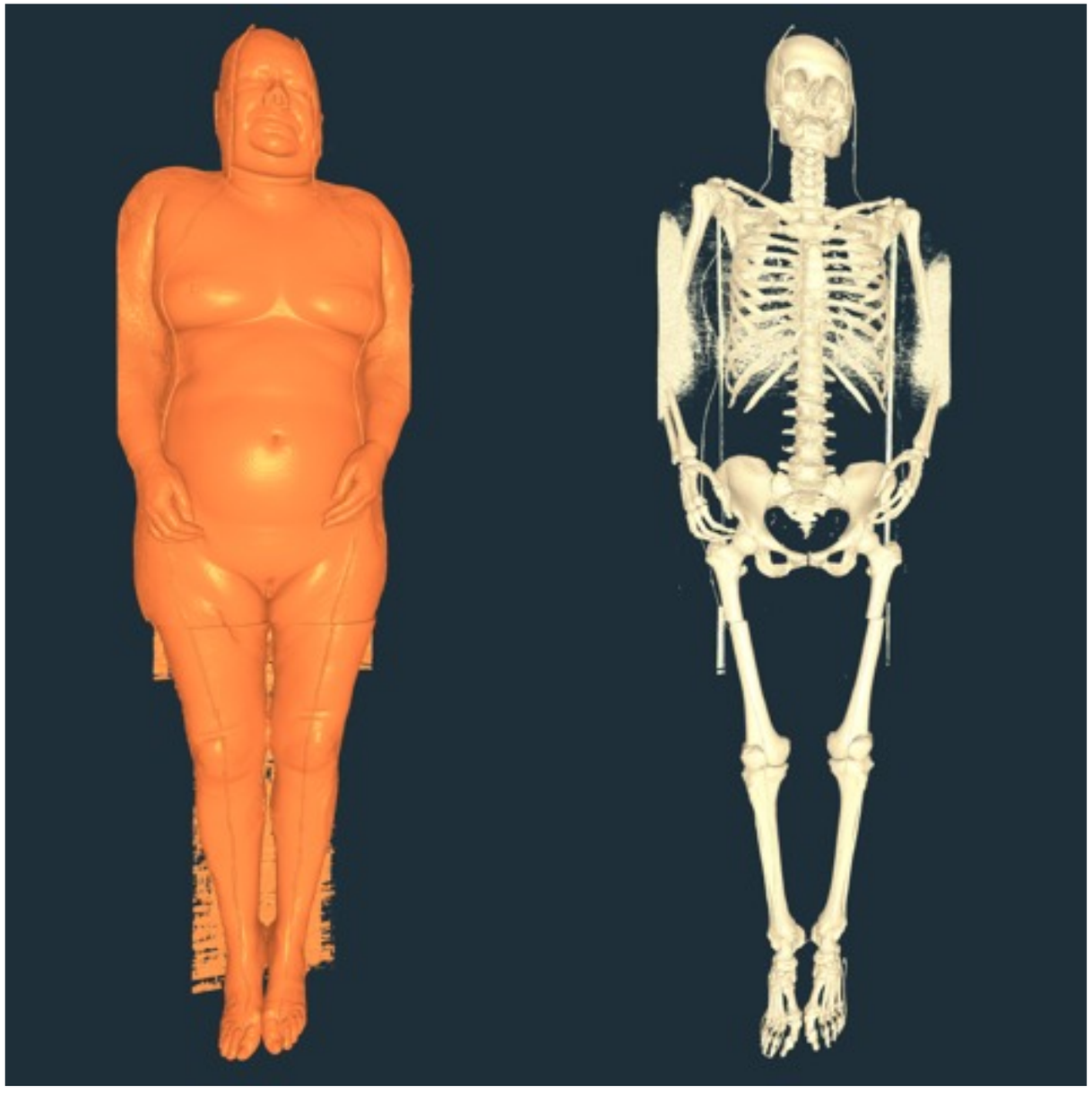


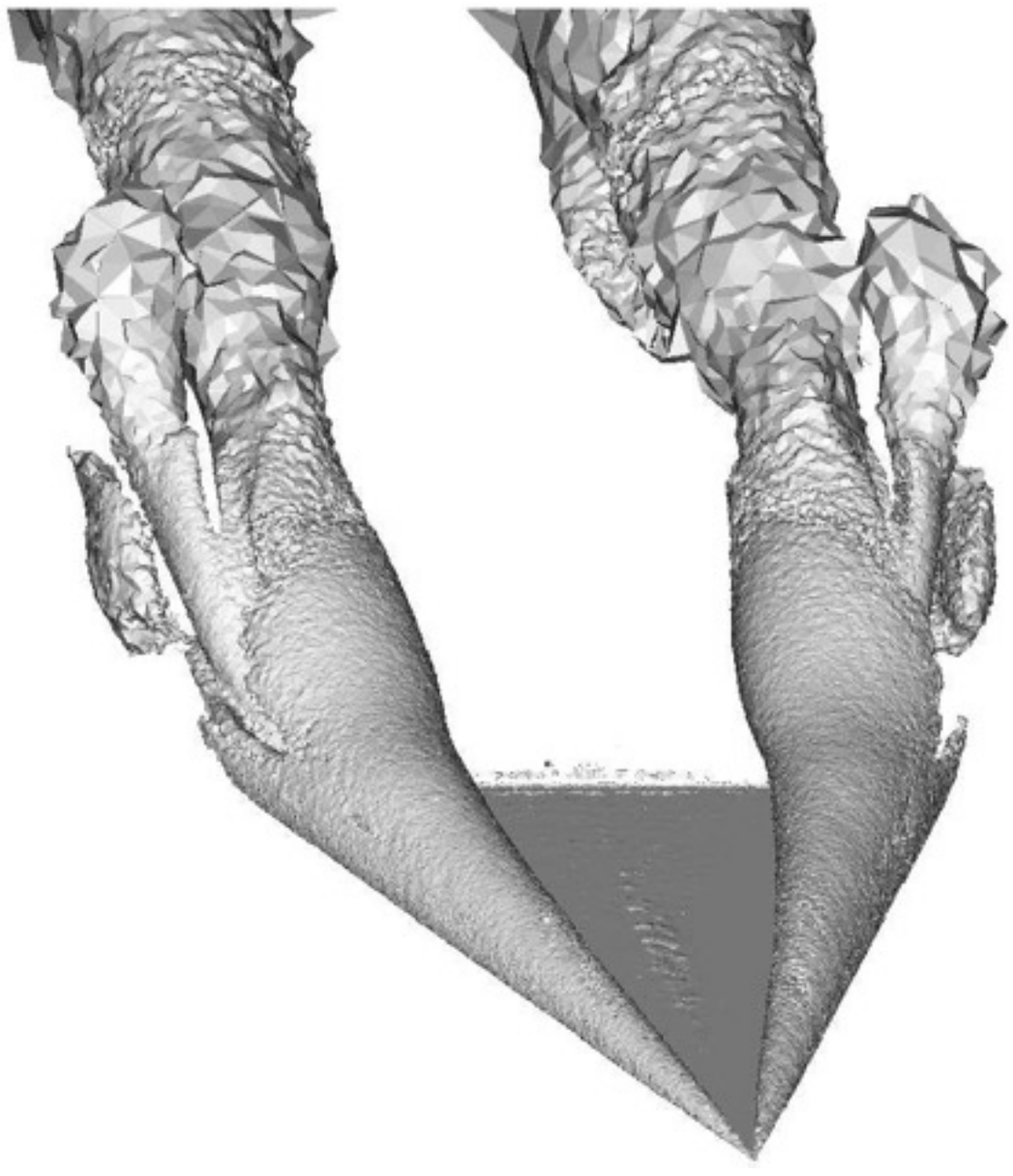
Mount Kilimanjaro, Tanzania

Other examples

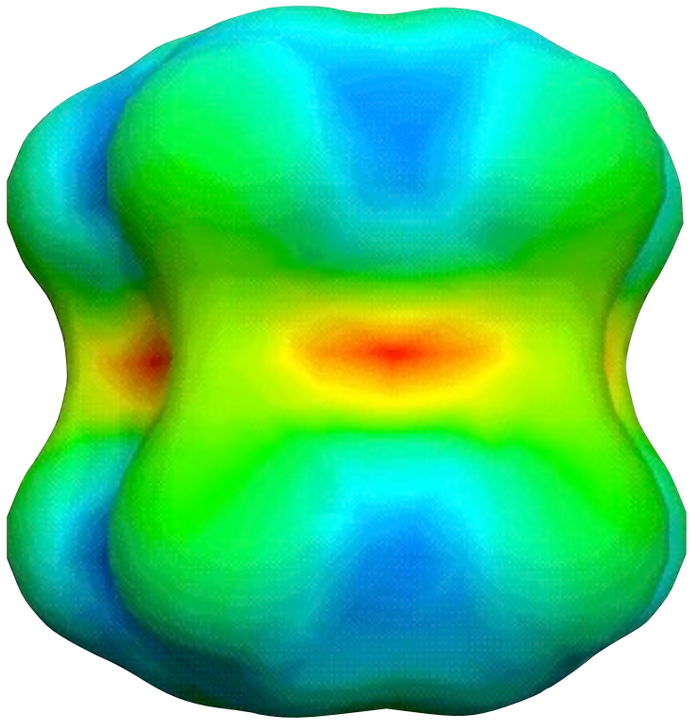


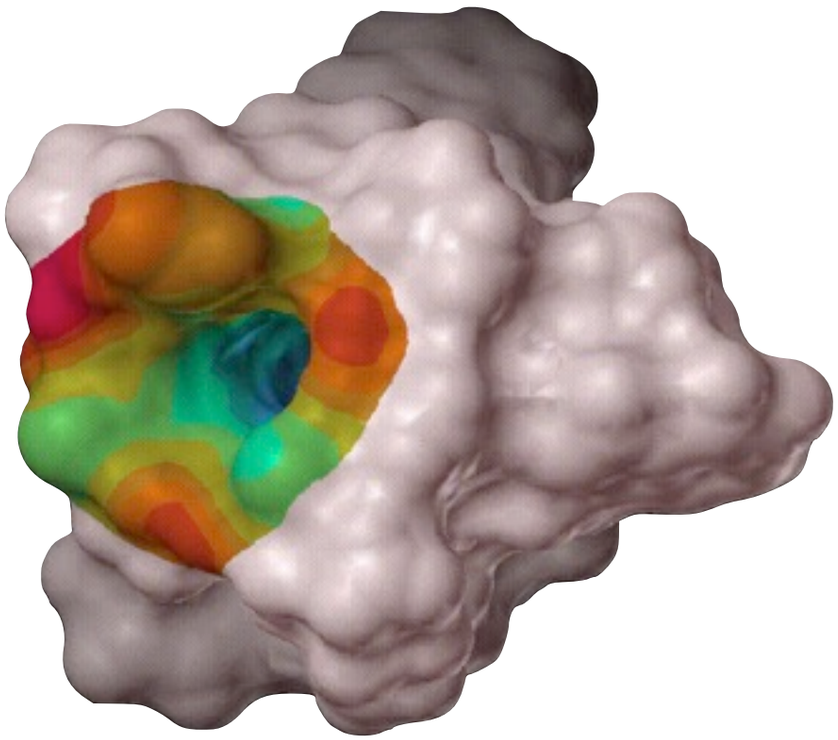
Other Examples

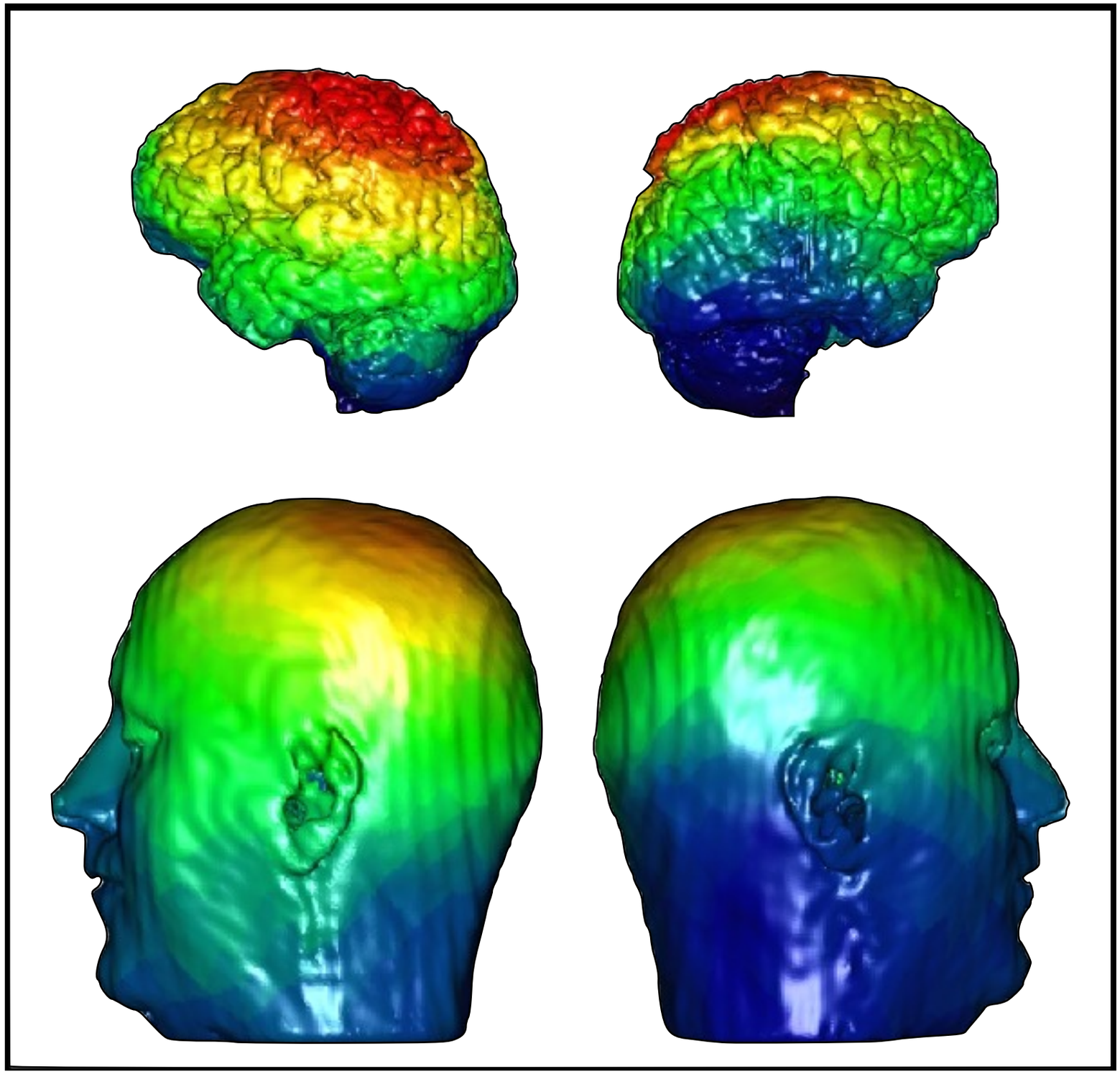




Colored Isosurfaces

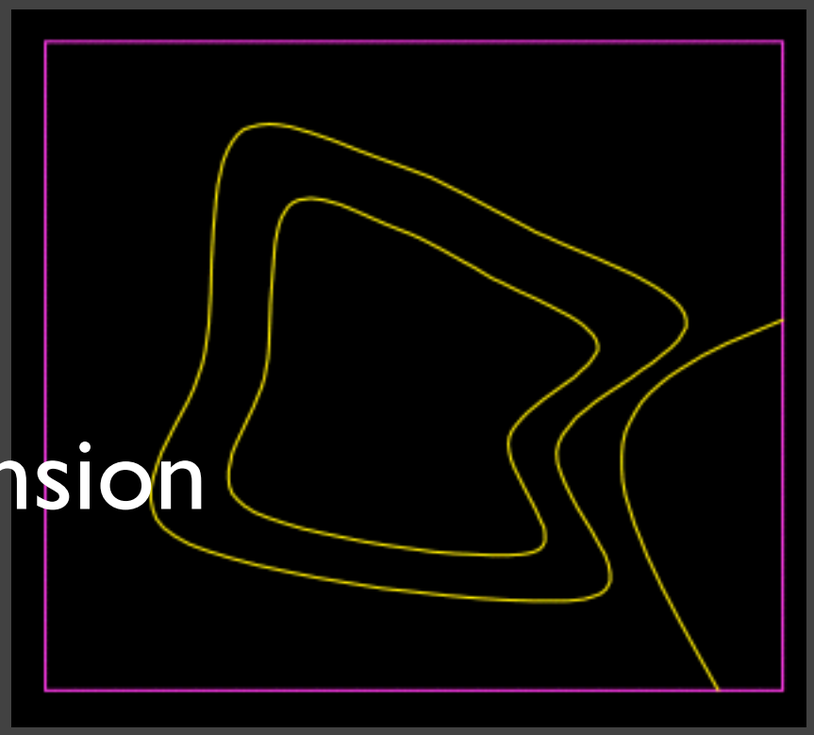






ISOCONTOURS in 2D

properties



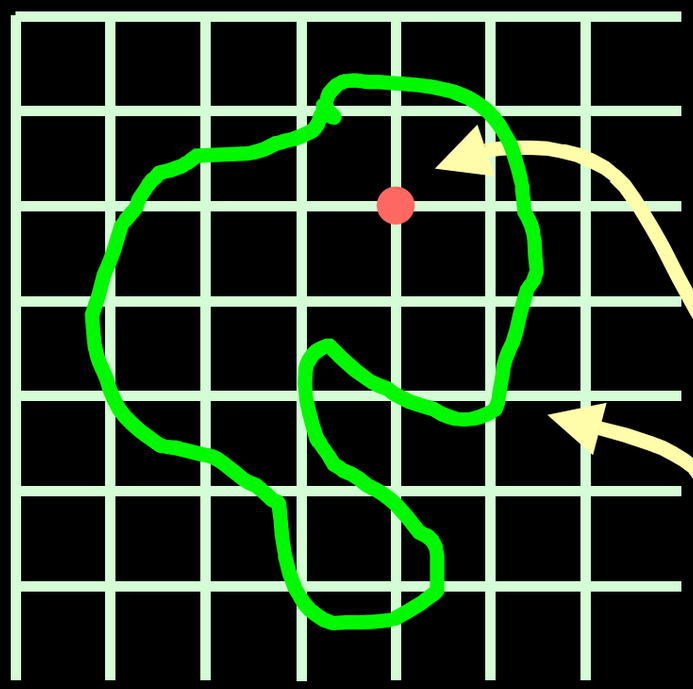
- concepts generalize to any dimension

- closed, except at boundaries

- **nested isocontours that don’t cross**
  - *can consider the zero-set case (generalizes)*
  - $f(x,y) = v \longrightarrow f(x,y) - v = 0$

- **normals given by gradient vector of f()**

Where are the data values?



Data value f defined on grid points only

$f(x,y,z)$

S_v

But we want a continuous, closed surface

Two solutions:

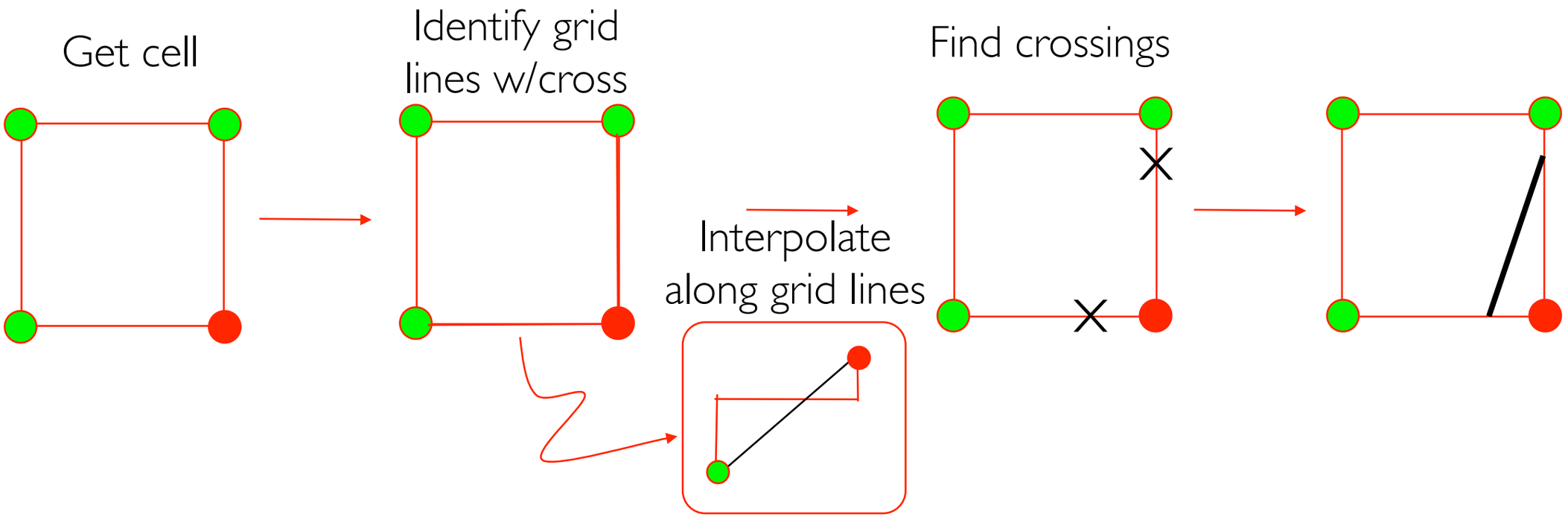
- Interpolate to get the "right" answer
  - Subsampling or raycasting
  - Dividing Cubes
- Approximate to get a "good" answer
  - Geometric primitives
  - Go cell by cell

Approach to Contouring in 2D

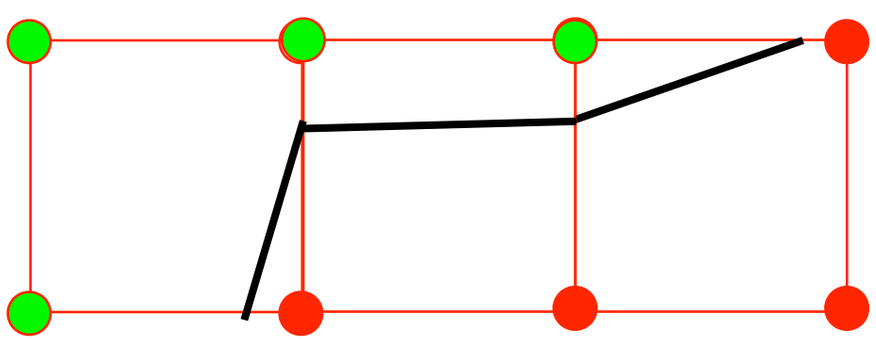
- **Idea**: Assign geometric primitives to individual cells
    - Will use line segments
- **Method**: Consider the “sign” of the values at vertices relative to if they are above or below the isovalue
    - Intersections MUST occur on **edges with sign change**
- Determine exact position of intersection by interpolate along grid edges

Approach to Contouring in 2D

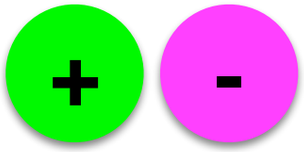
- Contour must cross every grid line connecting two grid points of opposite sign

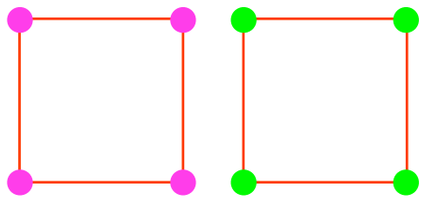
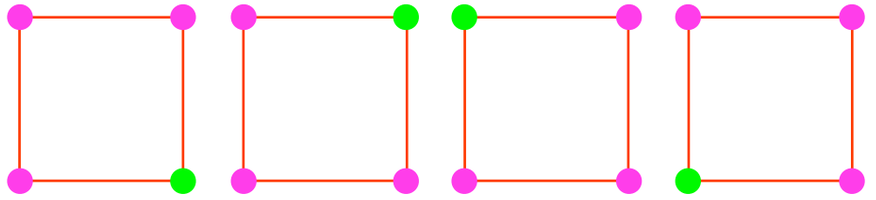
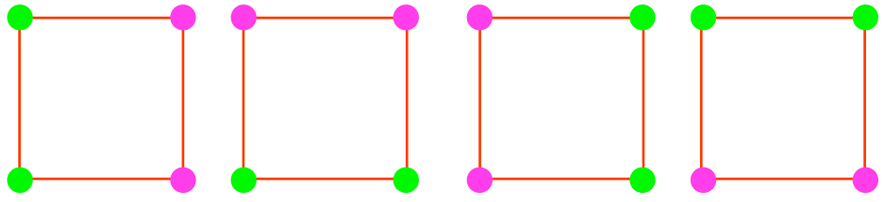
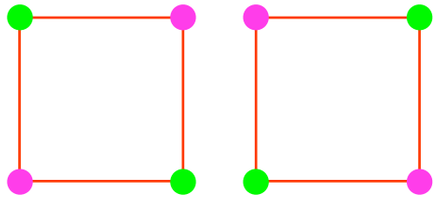


Primitives naturally chain together



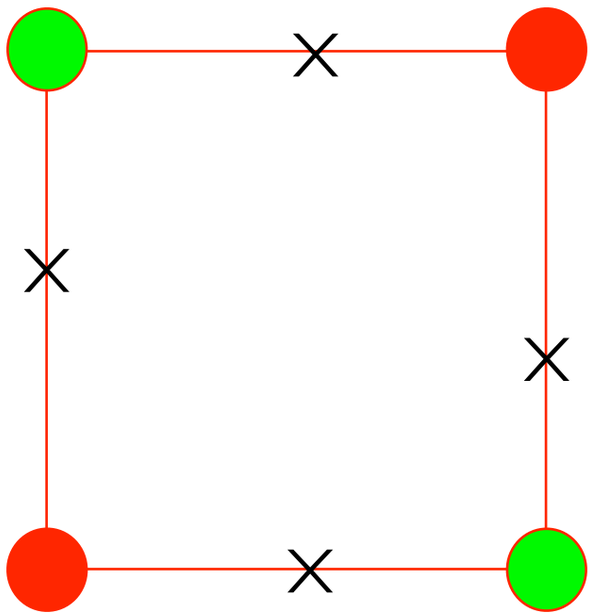
Cases



Case	Polarity	Rotation	Total	
No Crossings	x2		2	
Singlet	x2	x4	8	 (x2 for polarity)
Double adjacent	x2	x2 (4)	4	
Double Opposite	x2	x1 (2)	2	
			$16 = 2^4$	

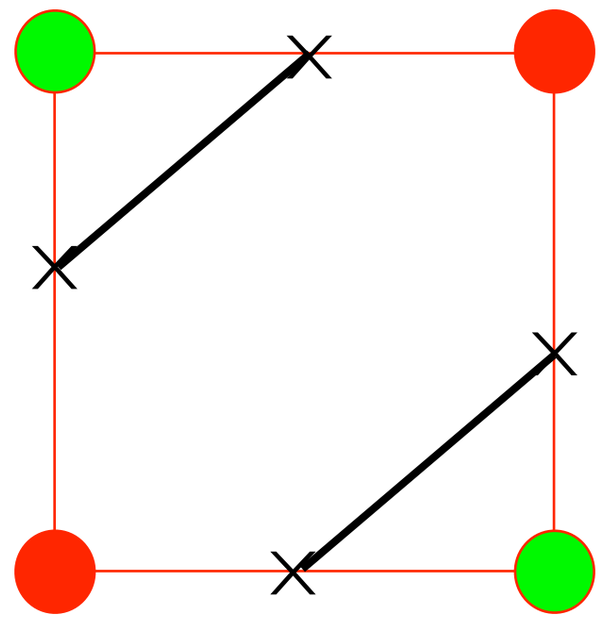
Ambiguities

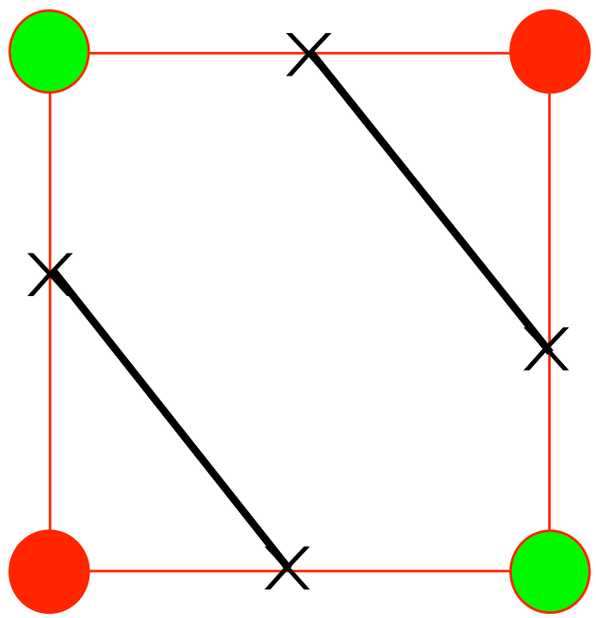
- How to form lines?

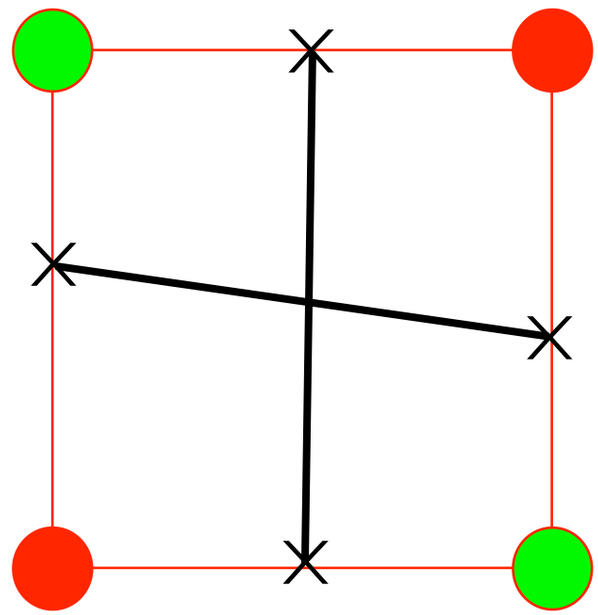


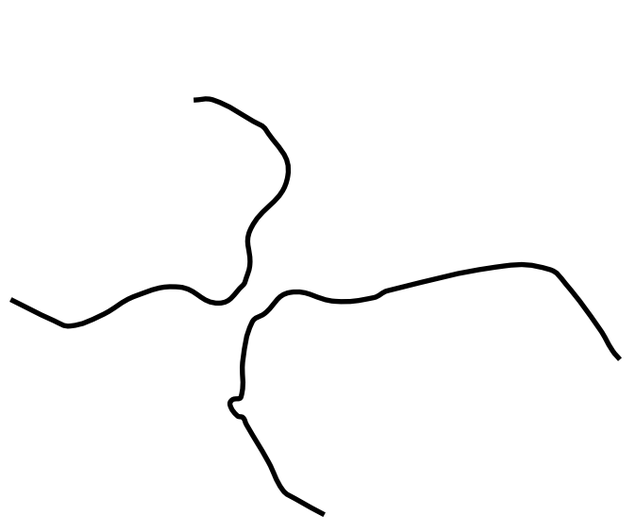
Ambiguities

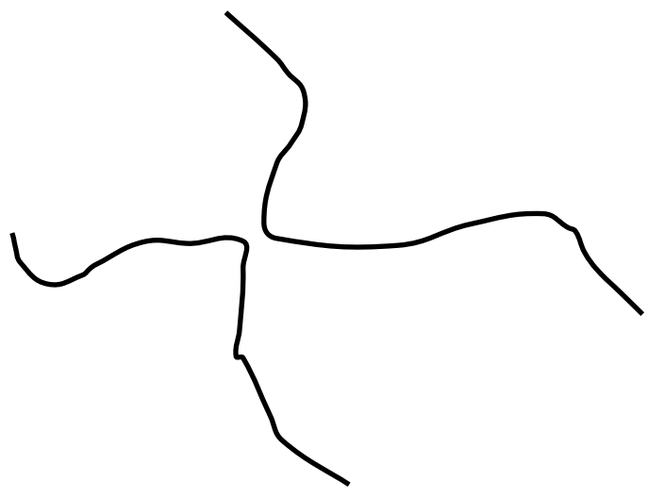
- Right or Wrong?

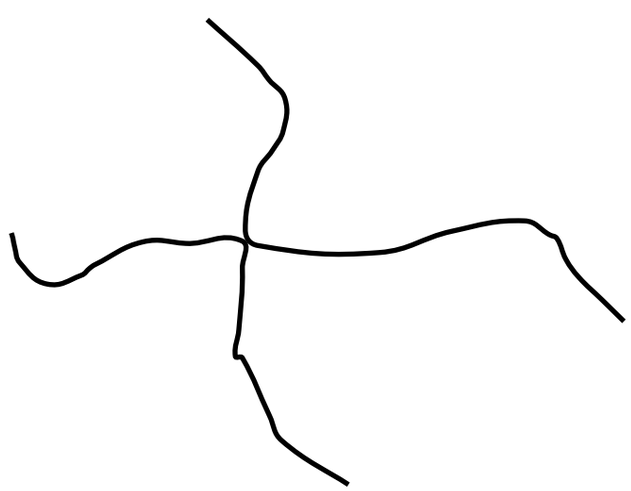












The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes

Gregory M. Nielson Bernd Hamann

Computer Science
Arizona State University
Tempe, AZ 85287-5406

Abstract

A method for computing isovalue or contour surfaces of a trivariate function is discussed. The input data are values of the trivariate function, F_{ijk}, at the cuberille grid points (x_i, y_j, z_k) and the output is a collection of triangles representing the surface consisting of all points where $F(x, y, z)$ is a constant value. The method described here is a modification that is intended to correct a problem with a previous method.

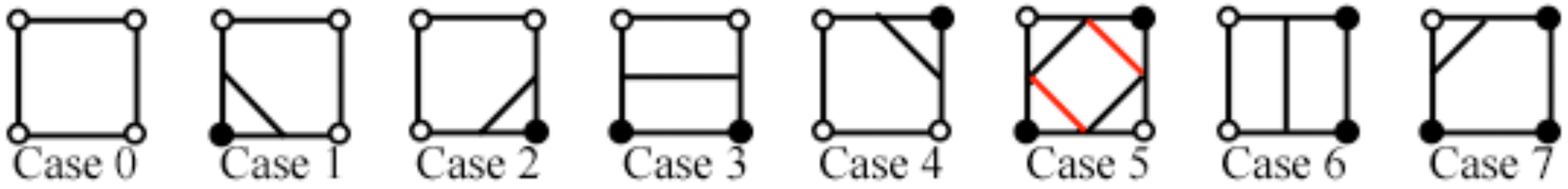
1.0 Introduction

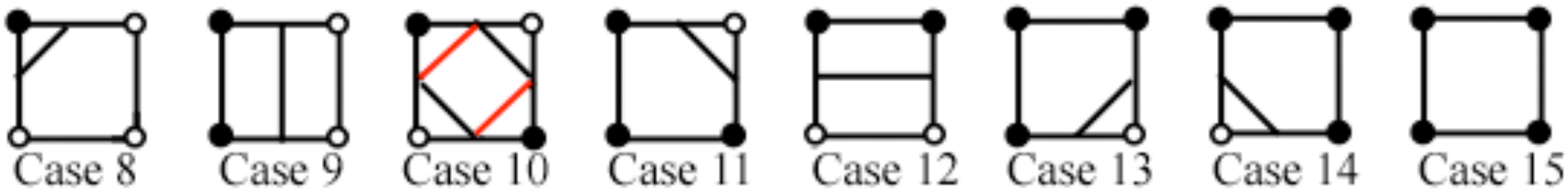
The purpose of this paper is to describe a method for computing contour or isovalue surfaces of a trivariate function F(x, y, z). It is assumed that the function is continuous and that samples over a cuberille grid (see Figure 1) are available. These values are denoted by $F_{ijk} = F(x_i, y_j, z_k)$; i = 1, ... , Nx, j = 1, ... , Ny, k = 1, ... , Nz. The problem is to compute the isovalue or contour surface

$$S_\alpha = \{ (x, y , z) : F(x, y, z) = \alpha\}.$$

marked indicates $F_{ijk} > \alpha$. While there are $2^8 = 256$ possible configurations, there are only 15 shown in Figure 2. This is because some configurations are equivalent with respect to certain operations. First off, the number can be reduced to 128 by assuming two configurations are equivalent if marked grid points and unmarked grid points are switched. This means that we only have to consider cases where there are four or fewer marked grid points. Further reduction to the 15 cases shown is possible by equivalence due to rotations.

A Case Table Can Be Used To Implement The Algorithm





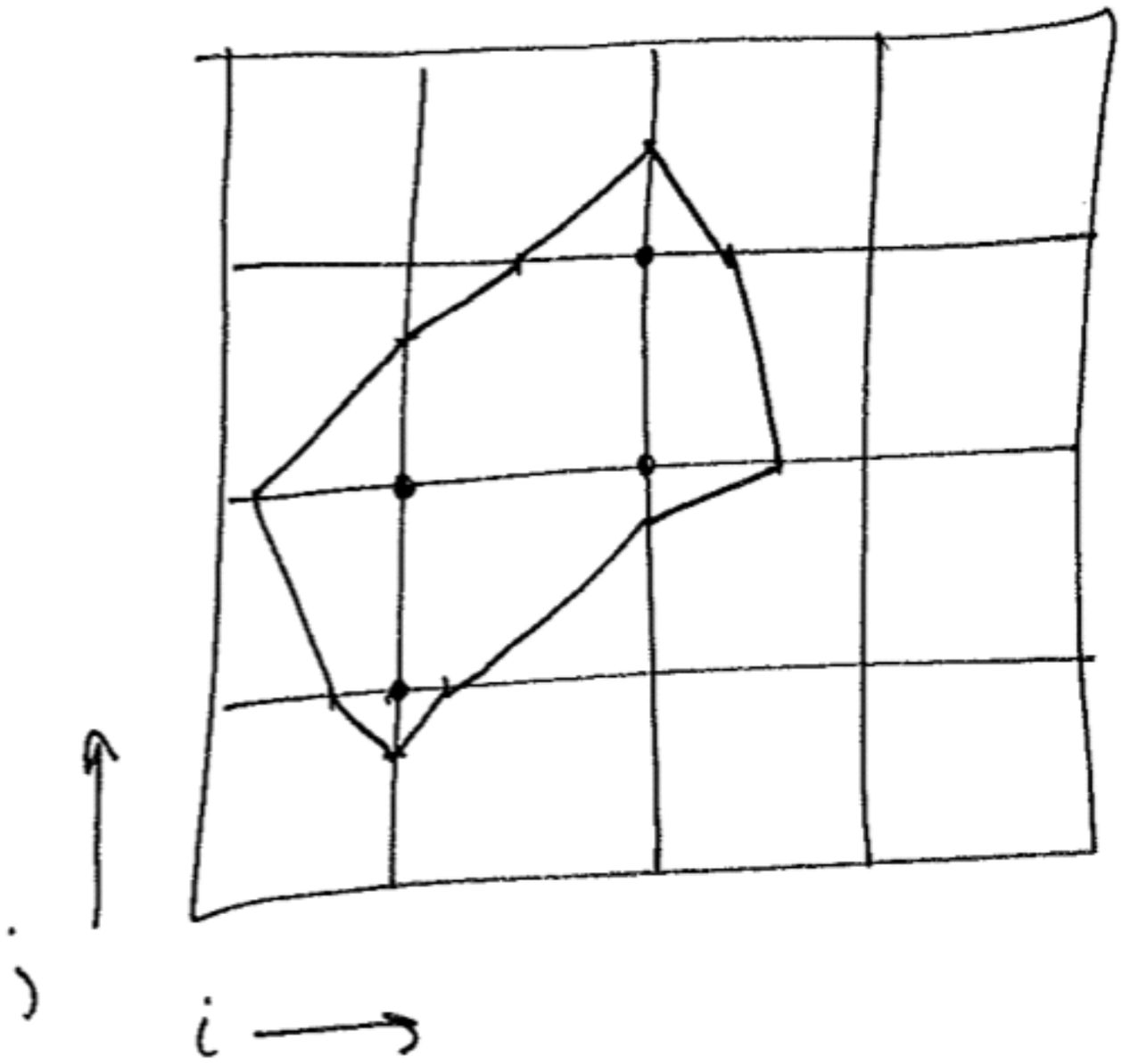
01323150033863202611012 31

5x5 grid

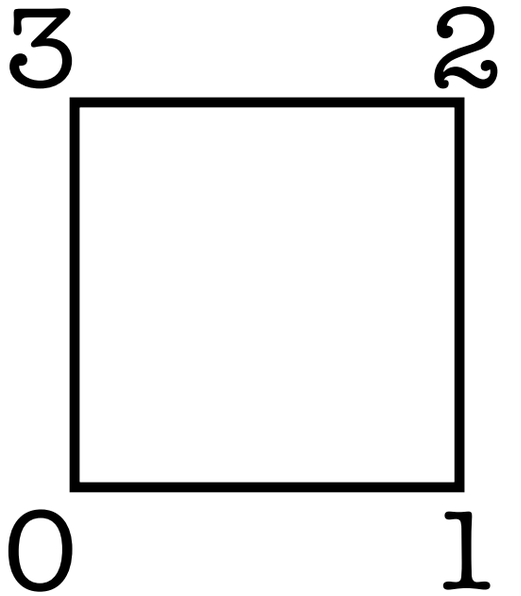
what is the isocontour
for isovalue = 4?

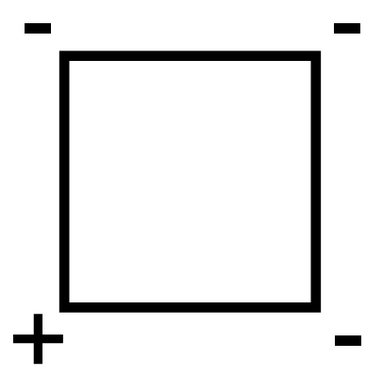
01323150033863202611012 31

5x5 grid

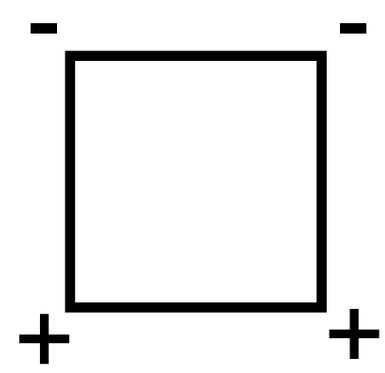


what is the isocontour
for isovalue = 4?

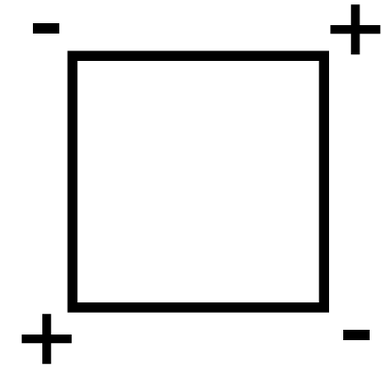




1000

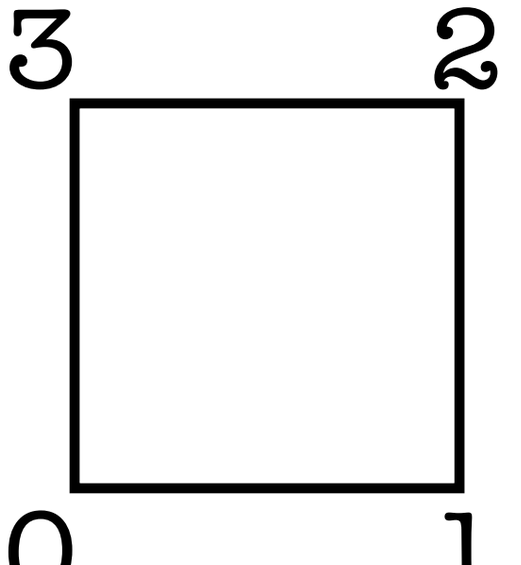


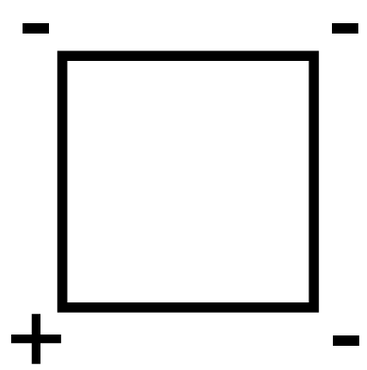
1100



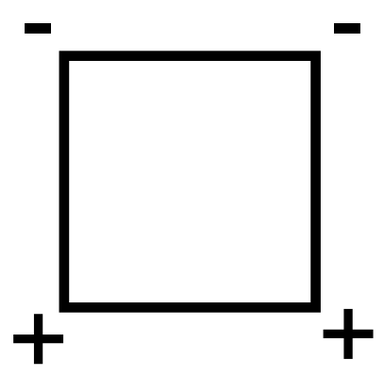
1010

$\overline{0}$ $\overline{1}$ $\overline{2}$ $\overline{3}$

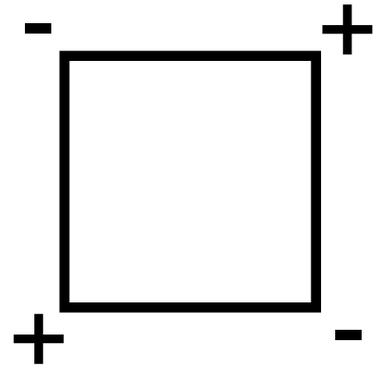




1000

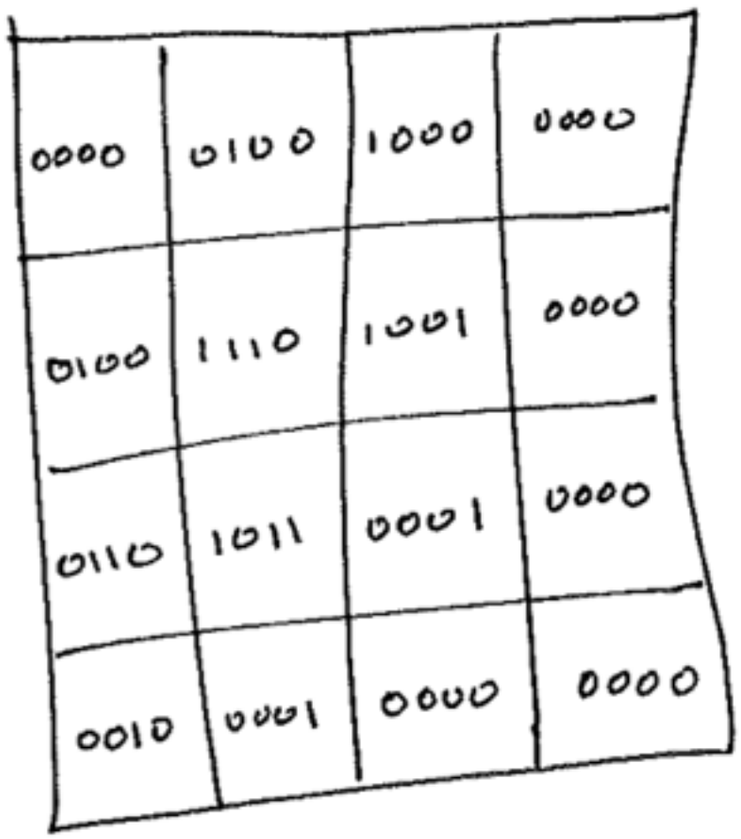


1100

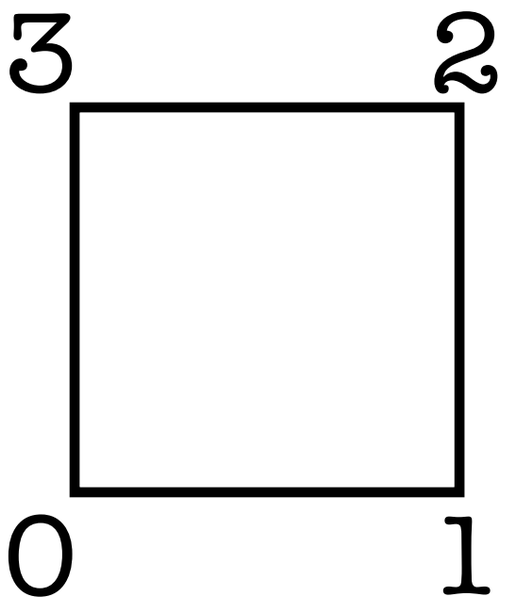


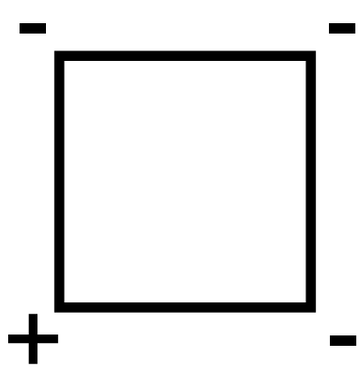
1010

$\overline{0}\ \overline{1}\ \overline{2}\ \overline{3}$

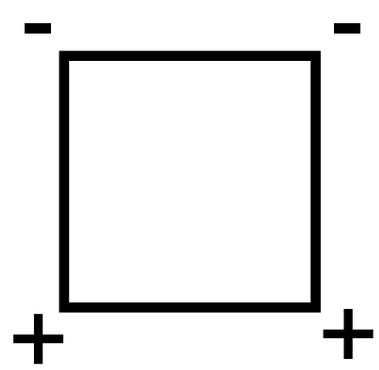


Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

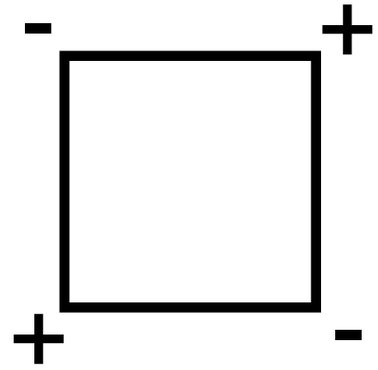




1000

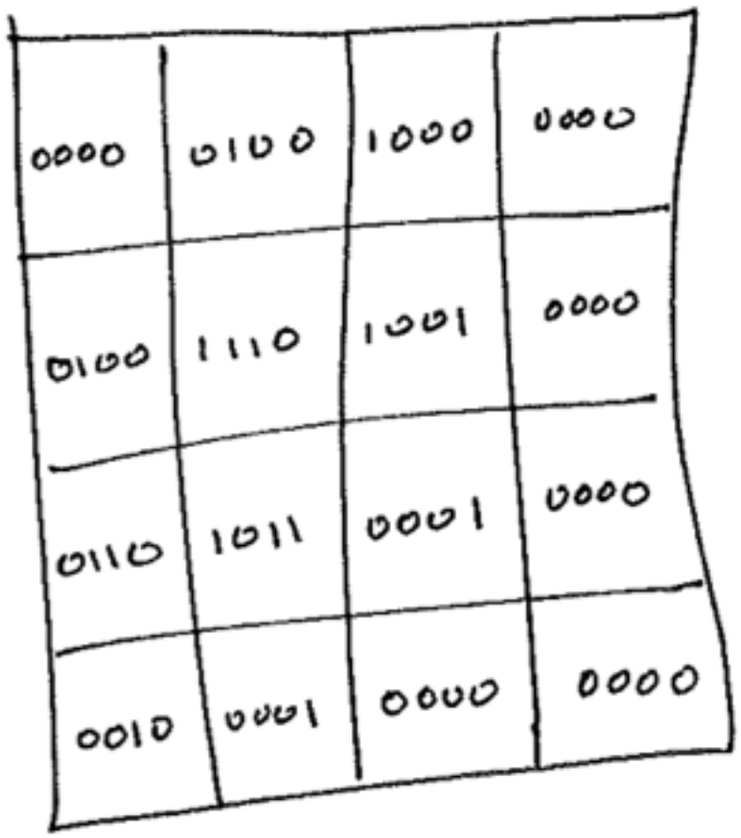


1100

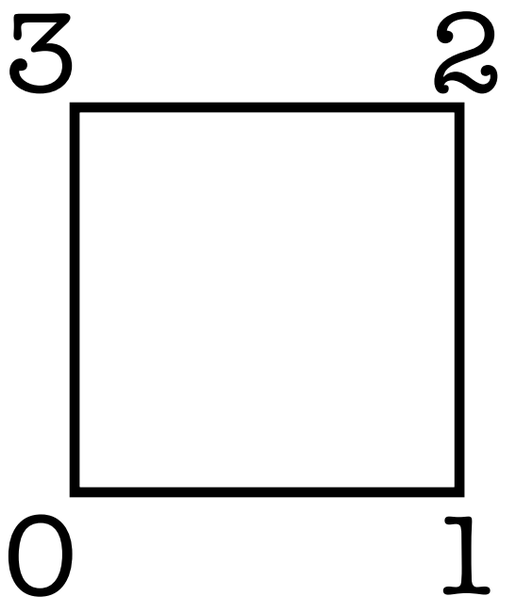


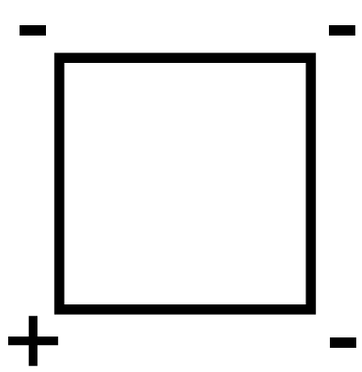
1010

$\overline{0}\ \overline{1}\ \overline{2}\ \overline{3}$

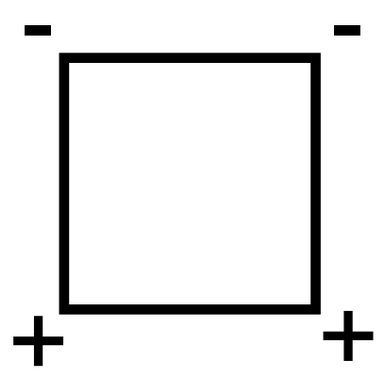


Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

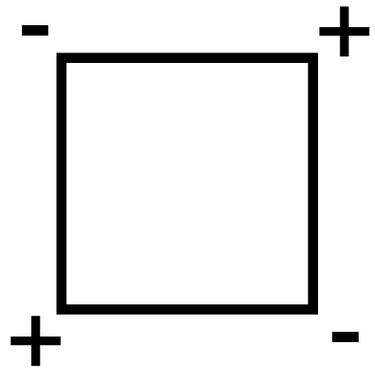




1000

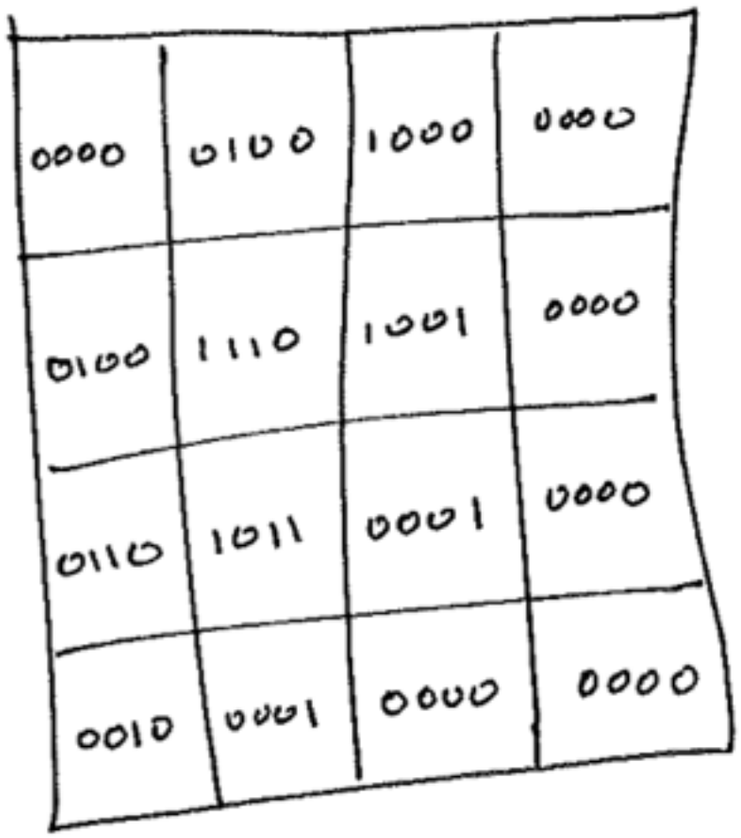


1100

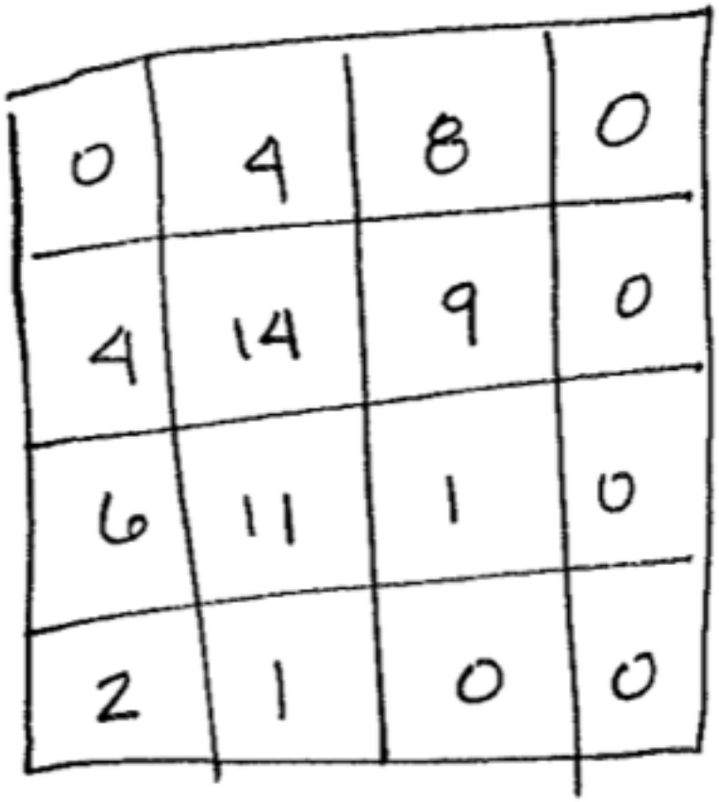


1010

$\overline{0}\ \overline{1}\ \overline{2}\ \overline{3}$

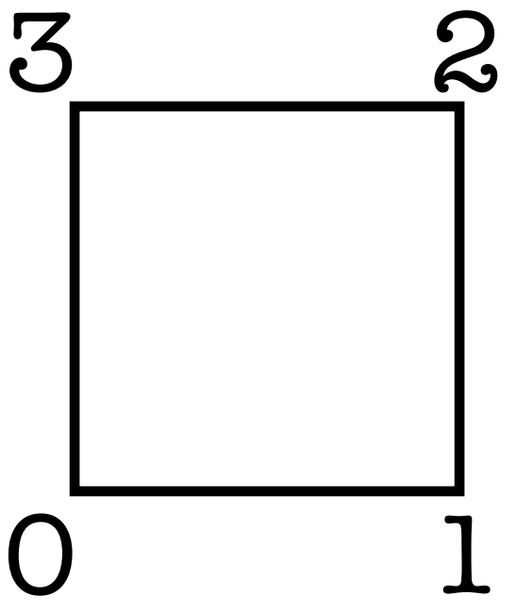


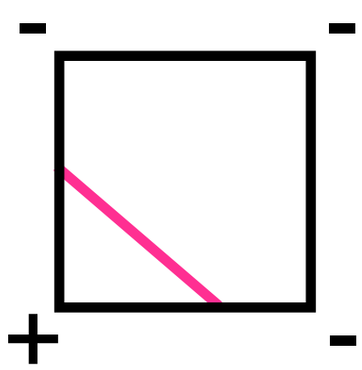
0000	0100	1000	0000
0100	1110	1001	0000
0110	1011	0001	0000
0010	0001	0000	0000



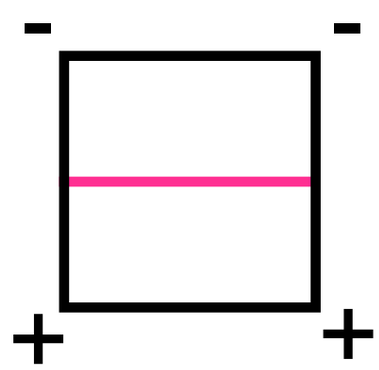
0	4	8	0
4	14	9	0
6	11	1	0
2	1	0	0

Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

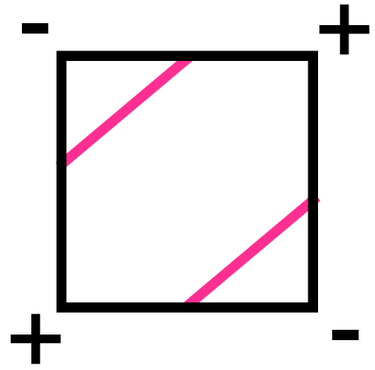




1000

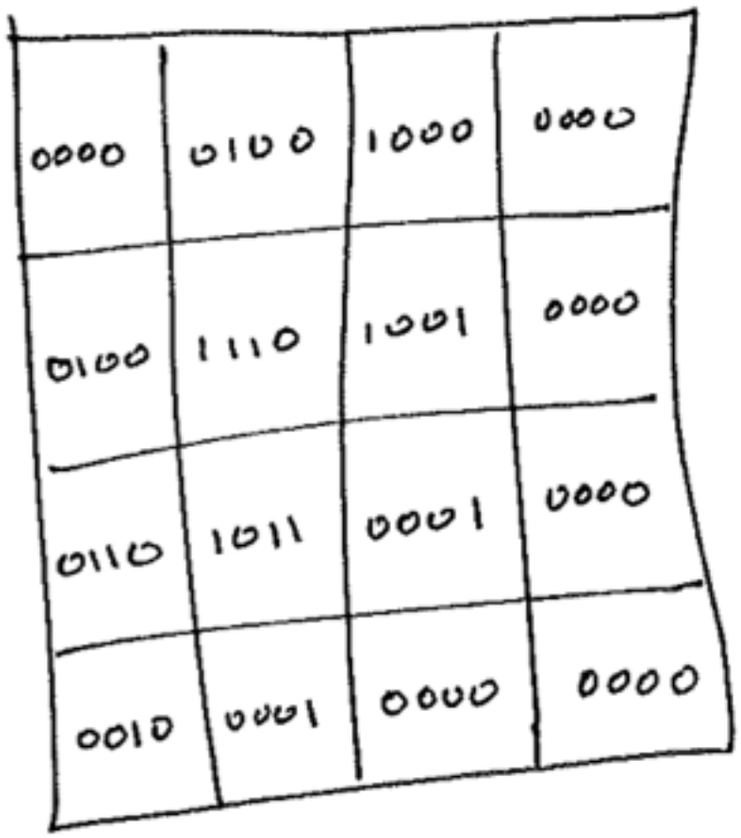


1100

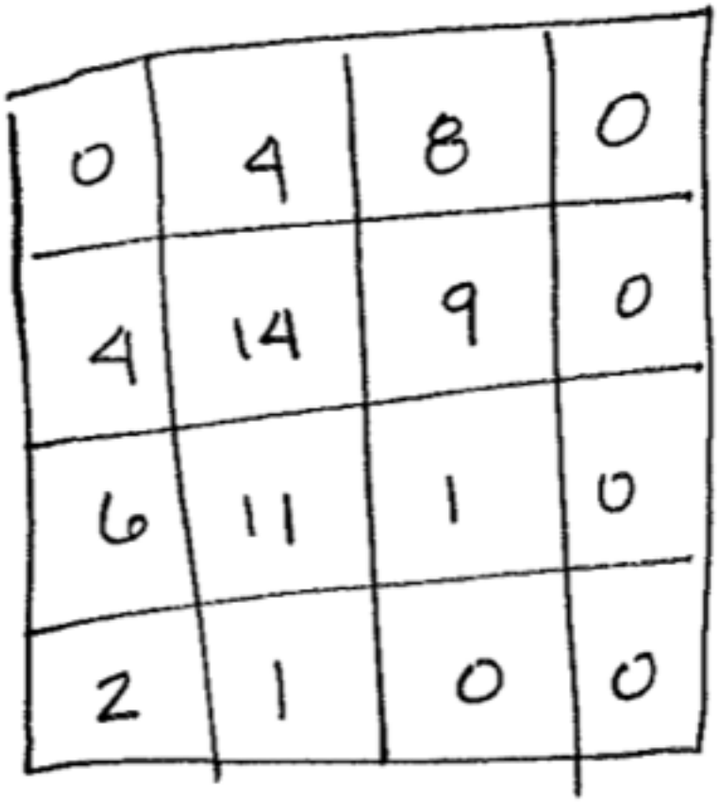


1010

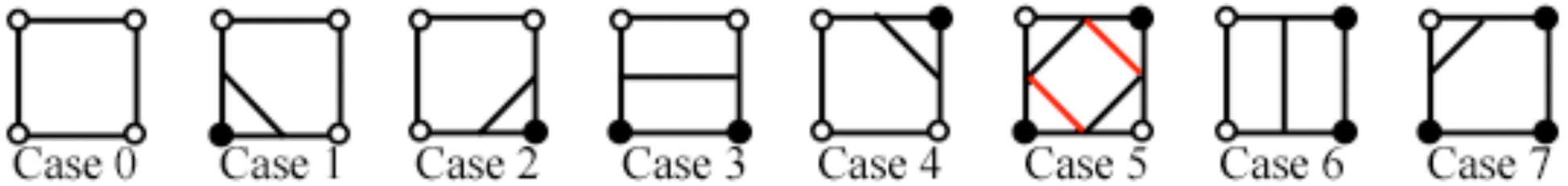
$\overline{0}\ \overline{1}\ \overline{2}\ \overline{3}$

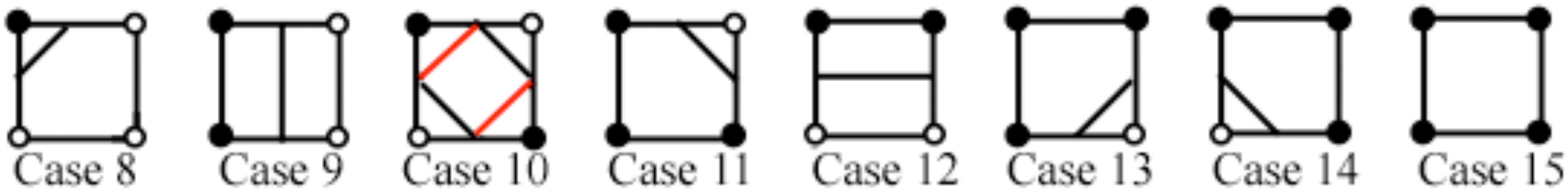


0000	0100	1000	0000
0100	1110	1001	0000
0110	1011	0001	0000
0010	0001	0000	0000



0	4	8	0
4	14	9	0
6	11	1	0
2	1	0	0

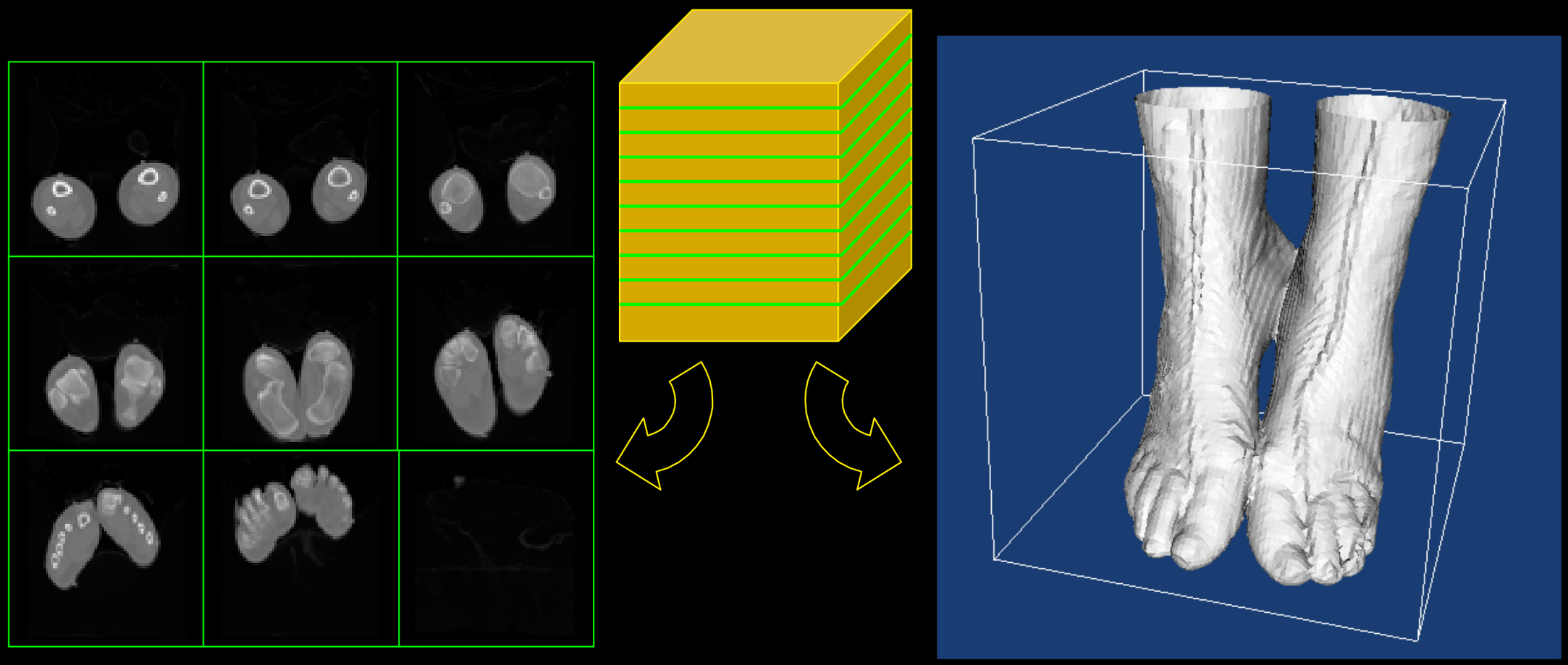

Case 0
Case 1
Case 2
Case 3
Case 4
Case 5
Case 6
Case 7


Case 8
Case 9
Case 10
Case 11
Case 12
Case 13
Case 14
Case 15

ISOSURFACES in 3D

Isosurfacing

- You’re given a big 3D block of numbers
- Make a picture
- Slicing shows data, but not its 3D shape
- Isosurfacing is one of the simplest ways



A little math

- Dataset: $v = f(x,y,z)$
- $f: R^3 \mapsto R$
- Want to find $S_v = \{(x,y,z) \mid f(x,y,z) = v\}$
- All the locations where the value of f is v

- S_v : <u>isosurface of f at v</u>
  - In 2D: isocontours (some path)
  - In 3D: isosurface

- Why is this useful?

MARCHING CUBES: A HIGH RESOLUTION 3D SURFACE CONSTRUCTION ALGORITHM

William E. Lorensen
Harvey E. Cline

General Electric Company
Corporate Research and Development
Schenectady, New York 12301

Abstract

We present a new algorithm, called *marching cubes*, that creates triangle models of constant density surfaces from 3D medical data. Using a divide-and-conquer approach to generate inter-slice connectivity, we create a case table that defines triangle topology. The algorithm processes the 3D medical data in scan-line order and calculates triangle vertices using linear interpolation. We find the gradient of the original data, normalize it, and use it as a basis for shading the models. The detail in images produced from the generated surface models is the result of maintaining the inter-slice connectivity, surface data, and gradient information present in the original 3D data. Results from computed tomography (CT), magnetic resonance (MR), and single-photon emission computed tomography (SPECT) illustrate the quality and functionality of *marching cubes*. We also discuss improvements that decrease processing time and add solid modeling capabilities.

CR Categories: 3.3, 3.5

Additional Keywords: computer graphics, medical imaging, surface reconstruction

able medical tool. Images of these surfaces, constructed from multiple 2D slices of computed tomography (CT), magnetic resonance (MR), and single-photon emission computed tomography (SPECT), help physicians to understand the complex anatomy present in the slices. Interpretation of 2D

acetabular fractures [6], craniofacial abnormalities [17,18], and intracranial structure [13] illustrate 3D's potential for the study of complex bone structures. Applications in radiation therapy [27,11] and surgical planning [4,5,31] show interactive 3D techniques combined with 3D surface images. Cardiac applications include artery visualization [2,16] and non-graphic modeling applications to calculate surface area and volume [21].

Existing 3D algorithms lack detail and sometimes introduce artifacts. We present a new, high-resolution 3D surface construction algorithm that produces models with unprecedented detail. This new algorithm, called *marching cubes*, creates a polygonal representation of constant density surfaces from a 3D array of data. The resulting model can be displayed with conventional graphics-rendering algorithms implemented in software or hardware.

After describing the information flow for 3D medical applications, we describe related work and discuss the drawbacks of that work. Then we describe the algorithm as well as efficiency and functional enhancements, followed by case studies using three different medical imaging techniques to illustrate the new algorithm's capabilities.

ure 1). Although one can combine the last three steps into one algorithm, we logically decompose the process as follows:

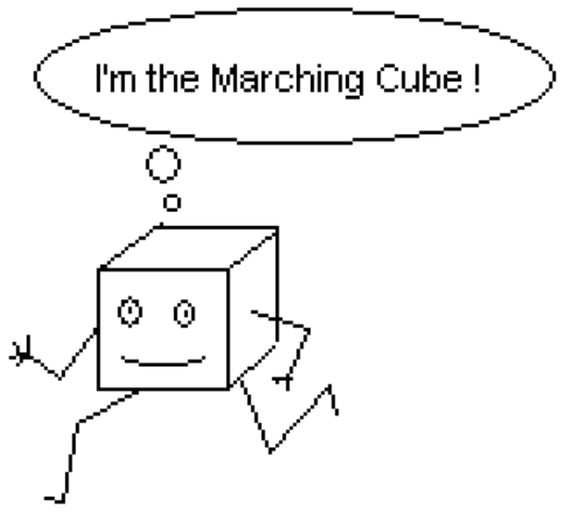
1. *Data acquisition.*
   This first step, performed by the medical imaging hardware, samples some property in a patient and pro-

Marching Cubes

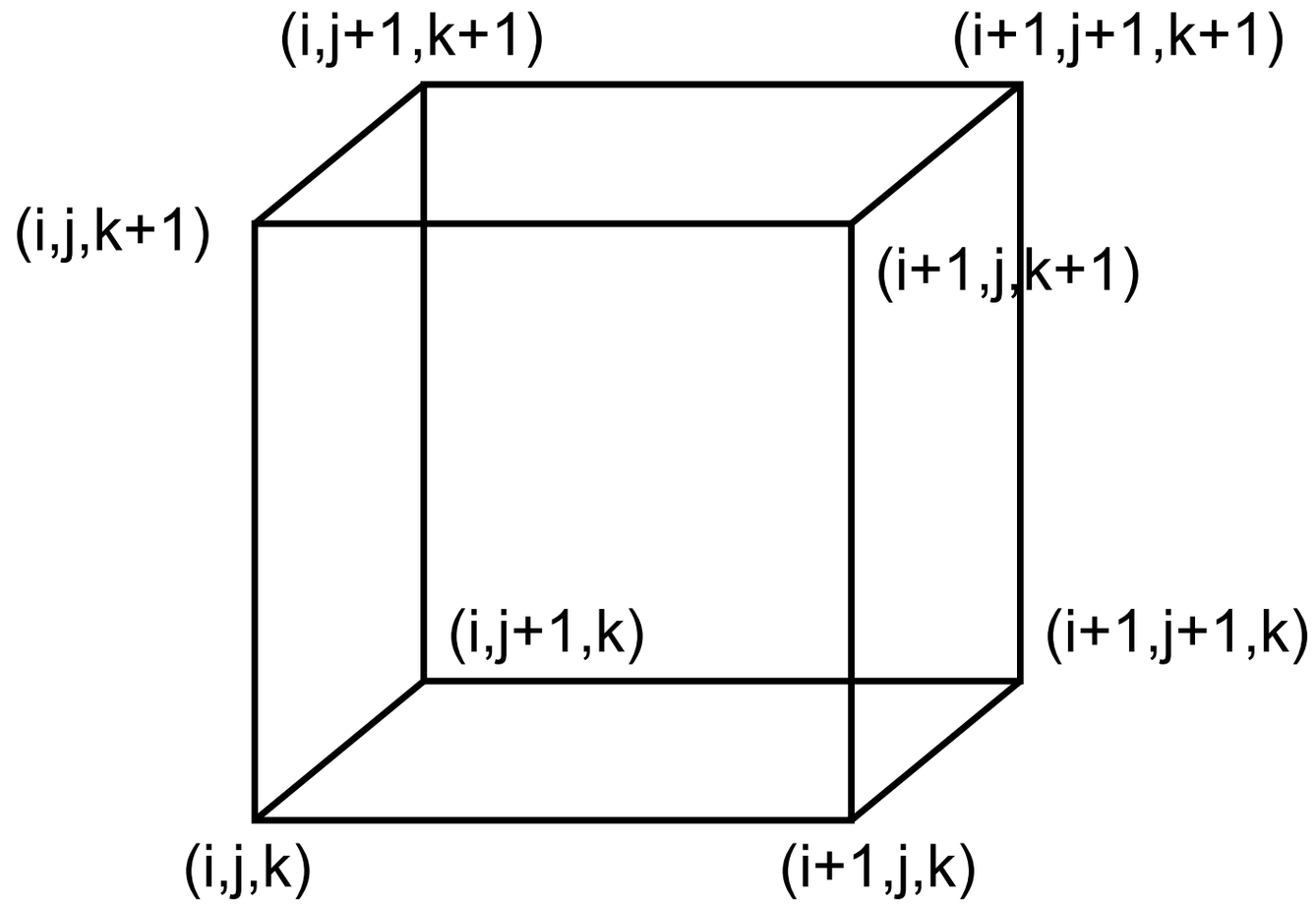
- "The" isosurface algorithm
- Lorensen + Cline ('87), Wyvill *et al.* ('86)
- Approximate, Efficient
- Involves many pre-computed tables
- Easy to understand, mostly easy to implement
- The foundation of how most people do isosurfacing

- The core MC algorithm
  - Cell consists of 4(8) pixel (voxel) values: (i+[01], j+[01], k+[01])

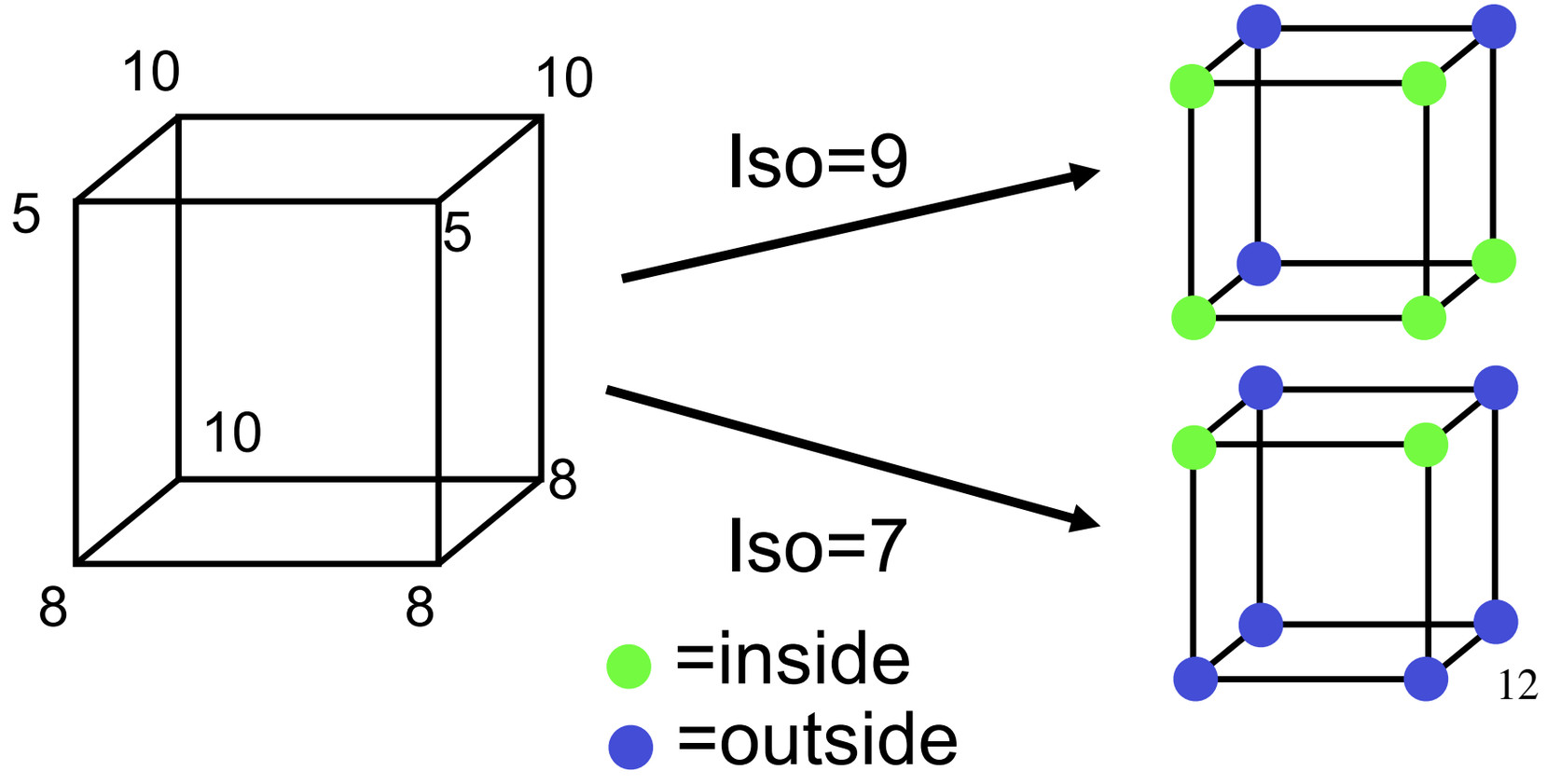
1. Consider a cell
2. Classify each vertex as inside or outside
3. Build an index
4. Get edge list from table[index]
5. Interpolate the edge location
6. Compute gradients
7. Consider ambiguous cases
8. Go to next cell



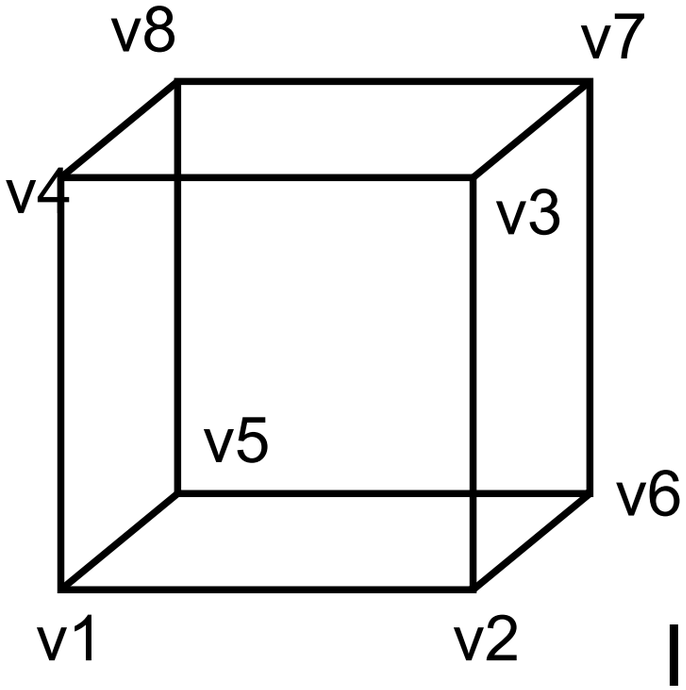
- Step 1: Consider a cell defined by eight data values



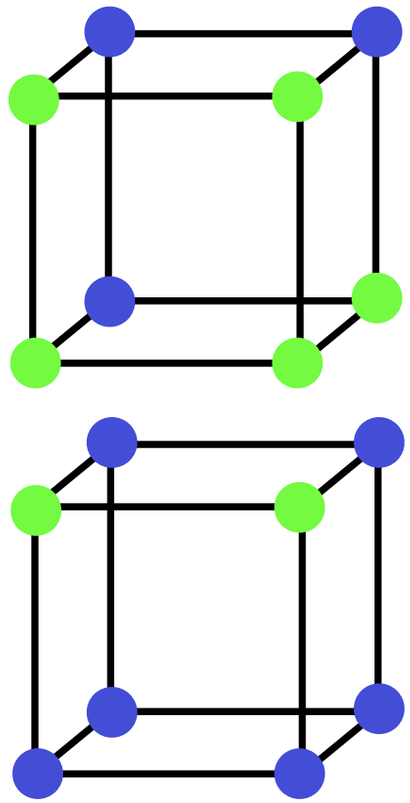
- Step 2: Classify each voxel according to whether it lies
  - Outside the surface (value > isosurface value)
  - Inside the surface (value <= isosurface value)



- Step 3: Use the binary labeling of each voxel to create an index



inside =1
outside=0



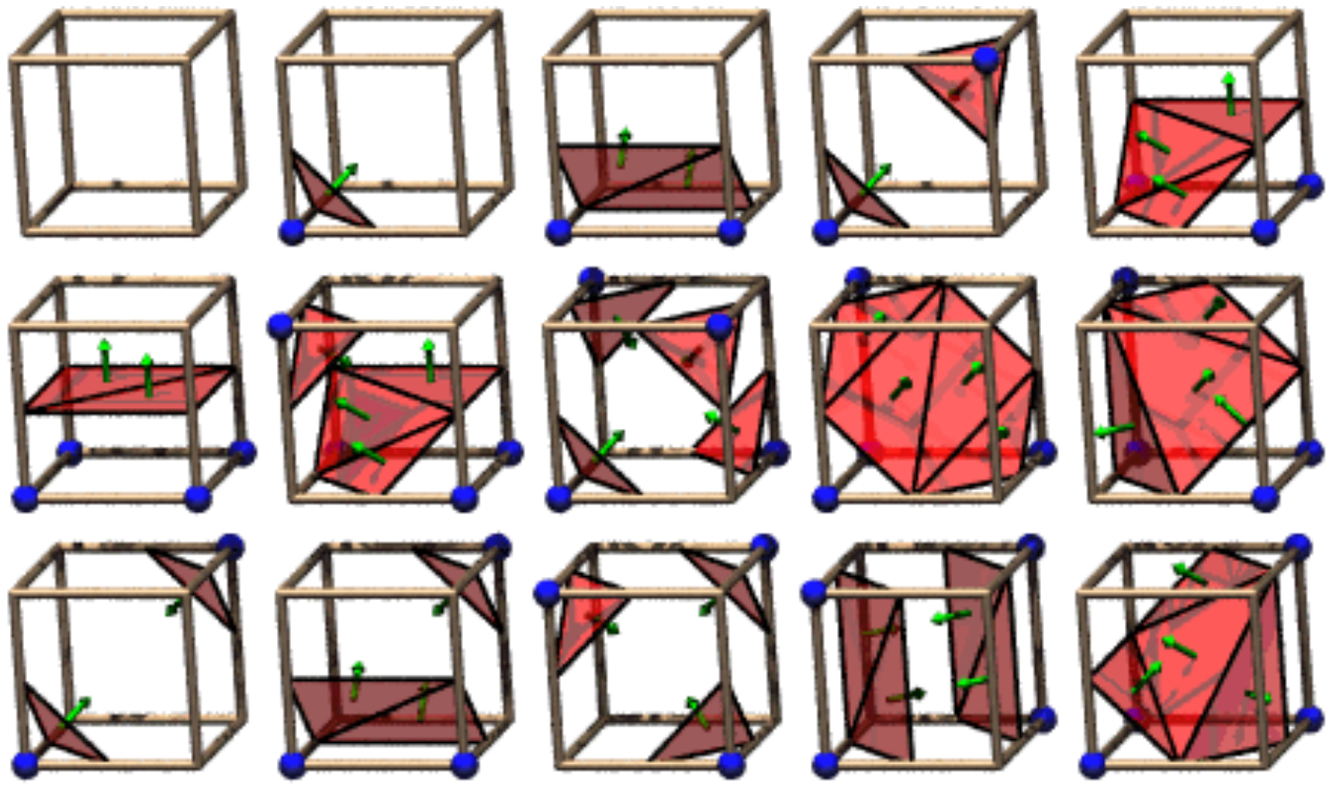
11110100

00110000

Index:

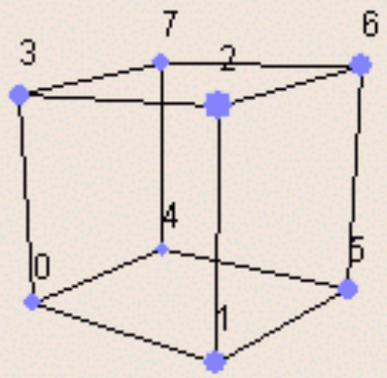
v1	v2	v3	v4	v5	v6	v7	v8

- Step 4: For a given index, access an array storing a list of edges
  - All 256 cases can be derived from 1+14=15 base cases due to symmetries

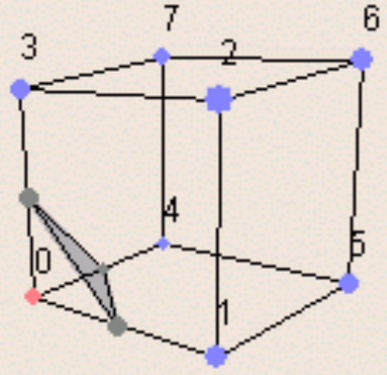


The 15 Cube Combinations

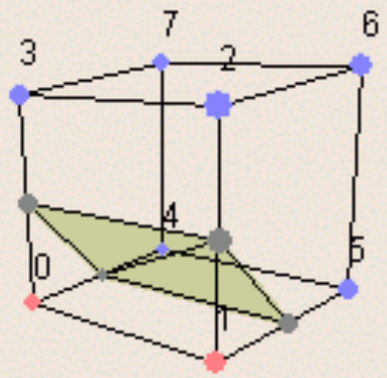
Case Table



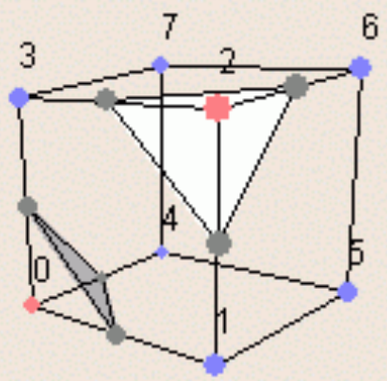
Case 0



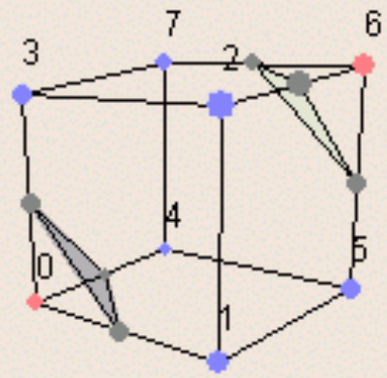
Case 1



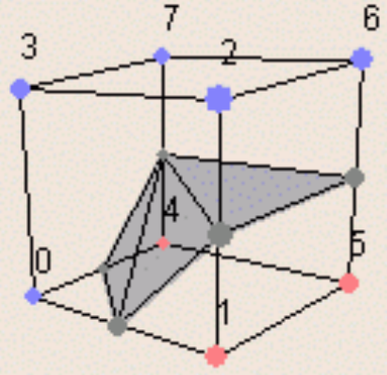
Case 2



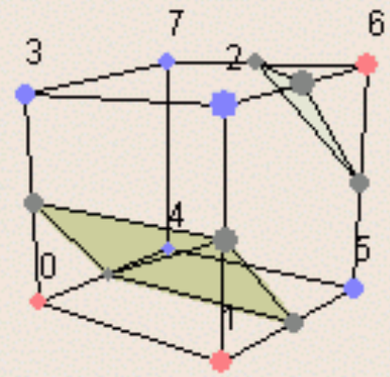
Case 3



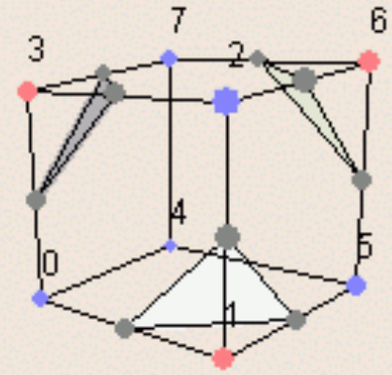
Case 4



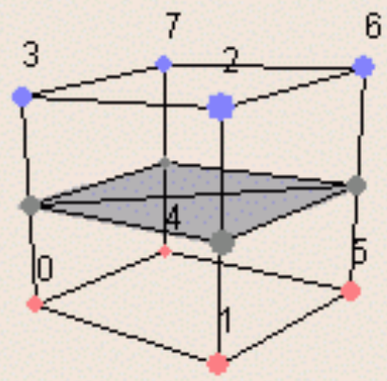
Case 5



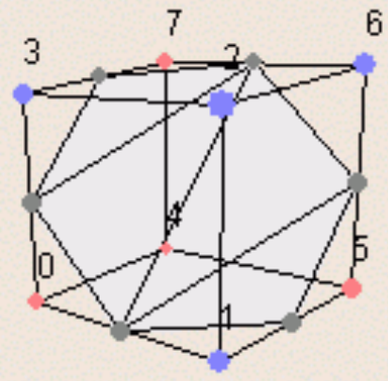
Case 6



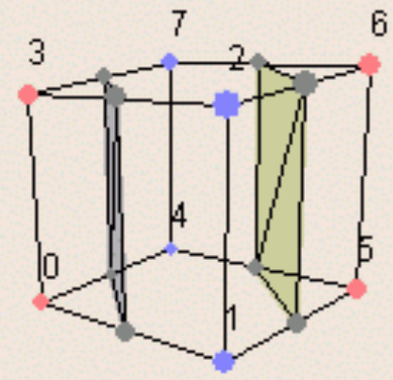
Case 7



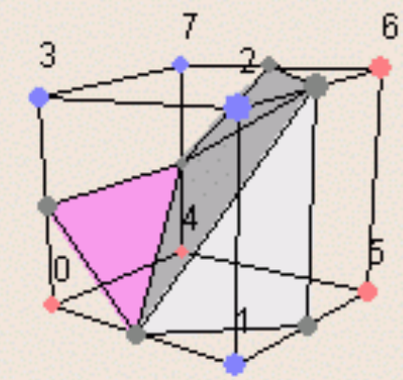
Case 8



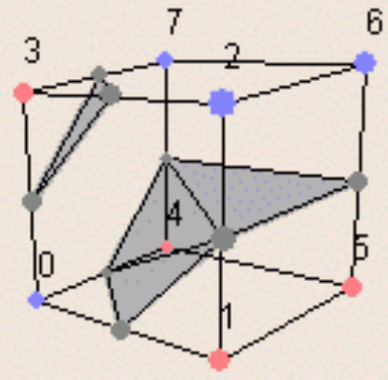
Case 9



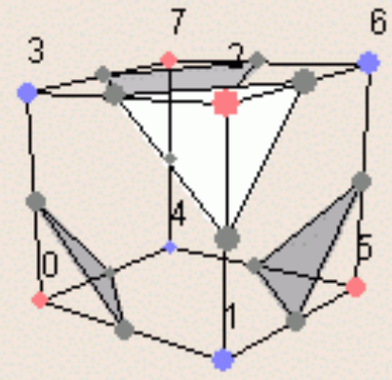
Case 10



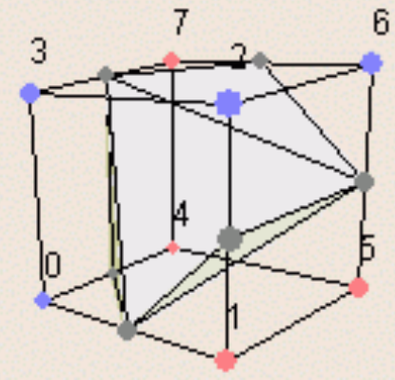
Case 11



Case 12



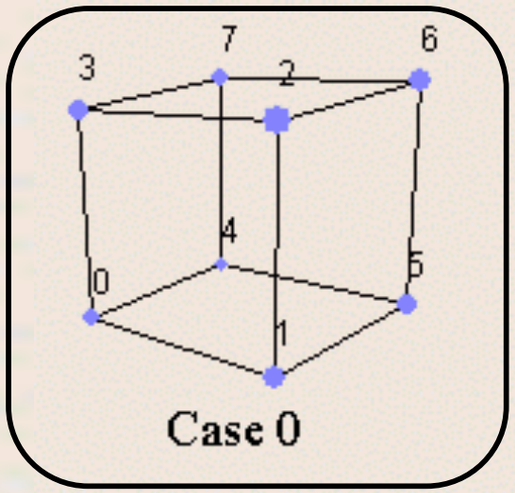
Case 13



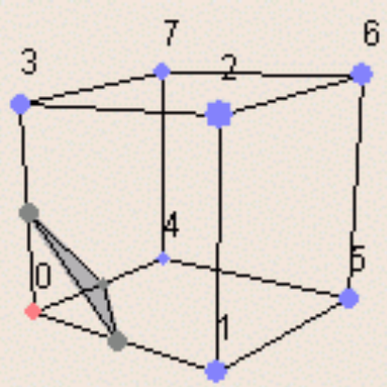
Case 14

- 8 Above
- 0 Below

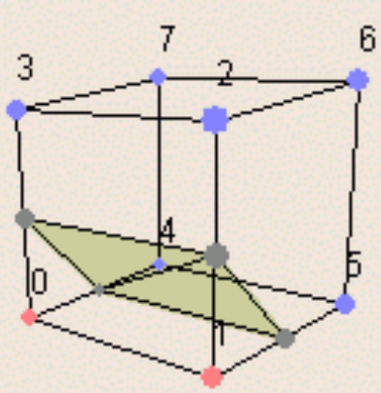
1 case



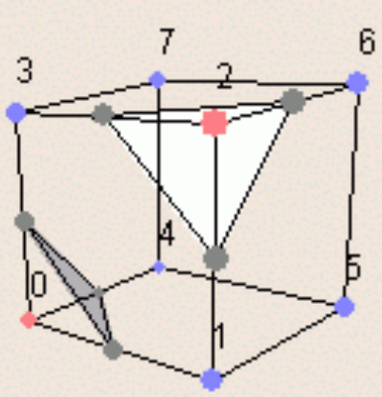
Case 0



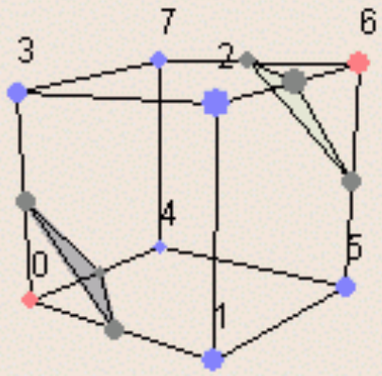
Case 1



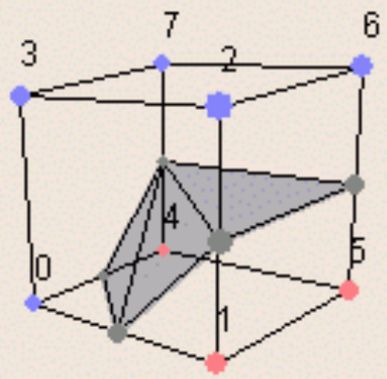
Case 2



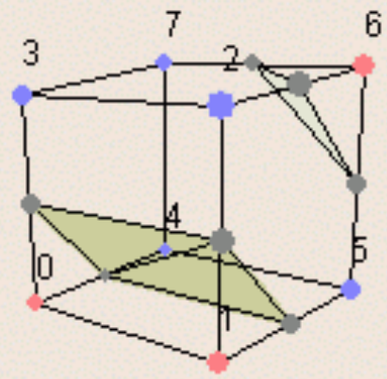
Case 3



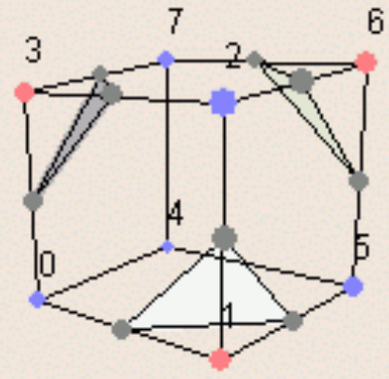
Case 4



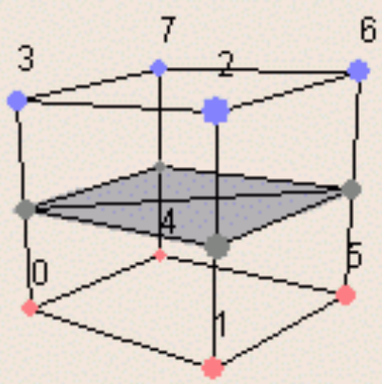
Case 5



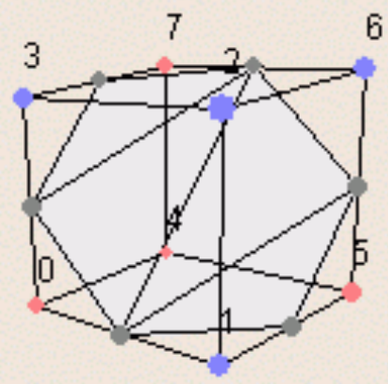
Case 6



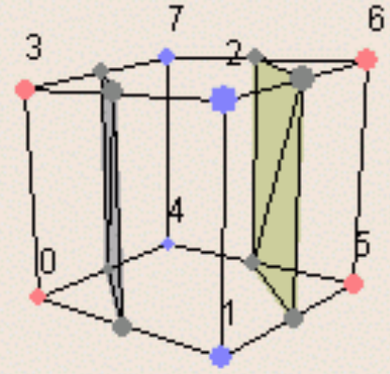
Case 7



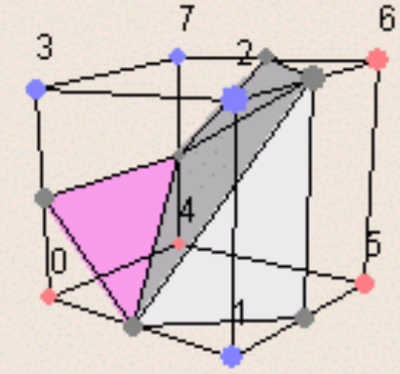
Case 8



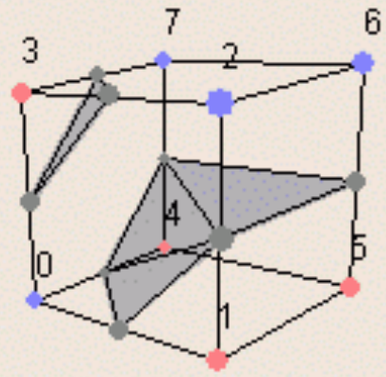
Case 9



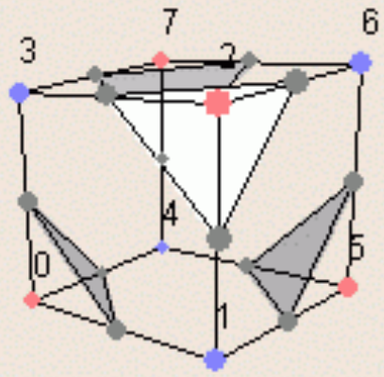
Case 10



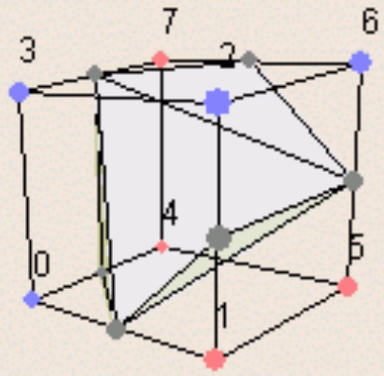
Case 11



Case 12



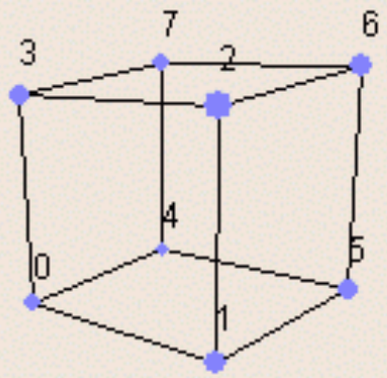
Case 13



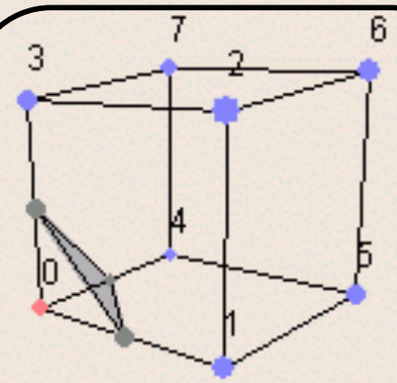
Case 14

- 7 Above
- 1 Below

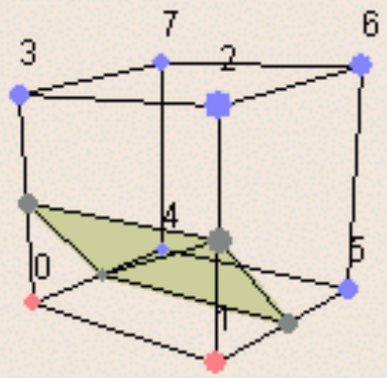
1 case



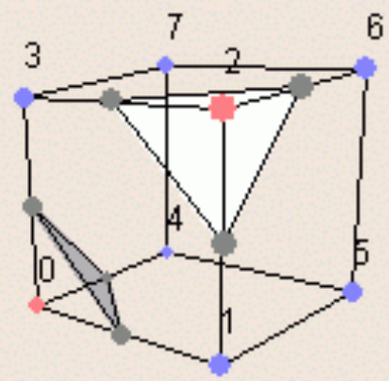
Case 0



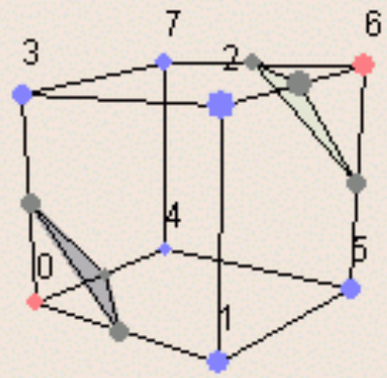
Case 1



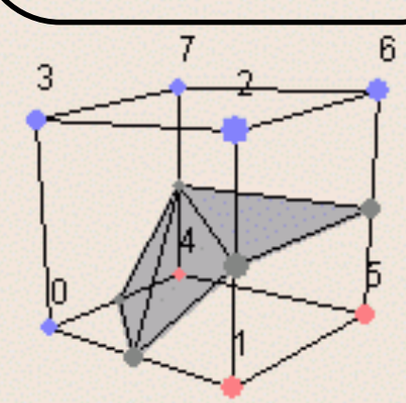
Case 2



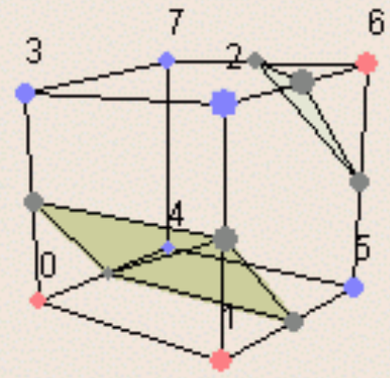
Case 3



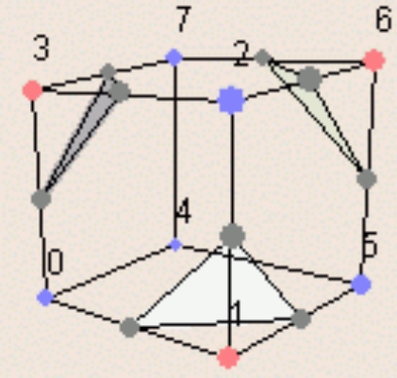
Case 4



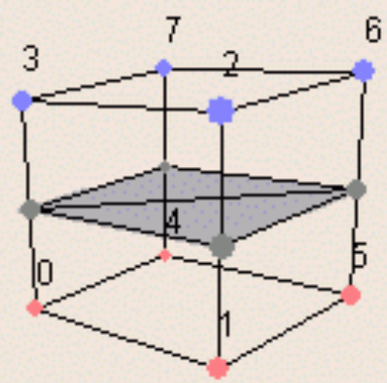
Case 5



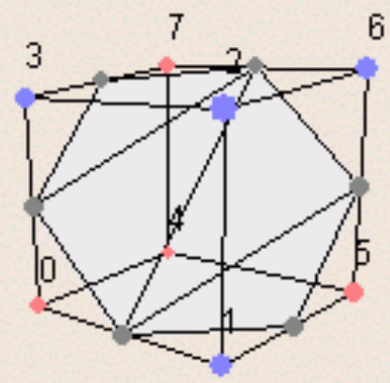
Case 6



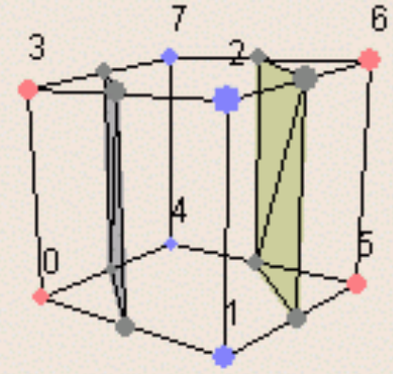
Case 7



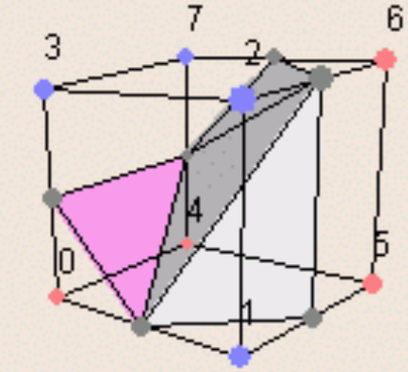
Case 8



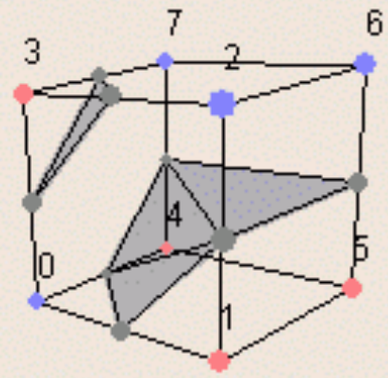
Case 9



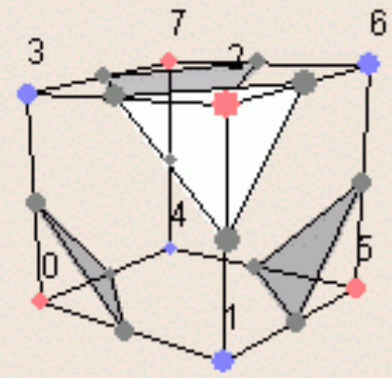
Case 10



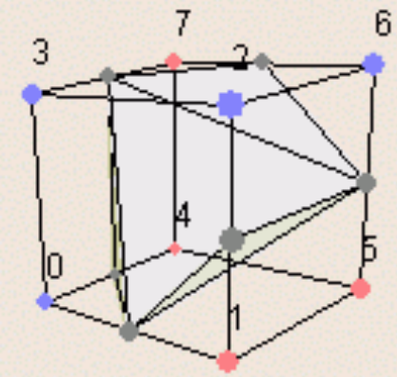
Case 11



Case 12



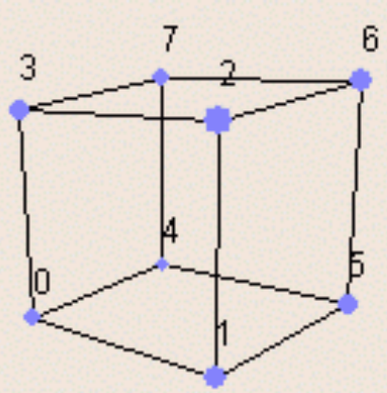
Case 13



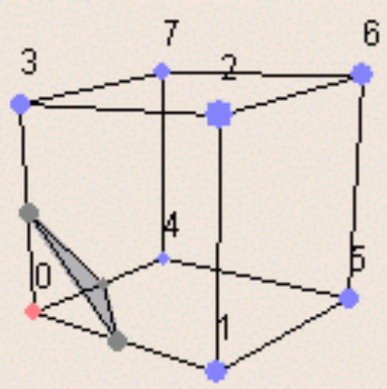
Case 14

- 6 Above
- 2 Below

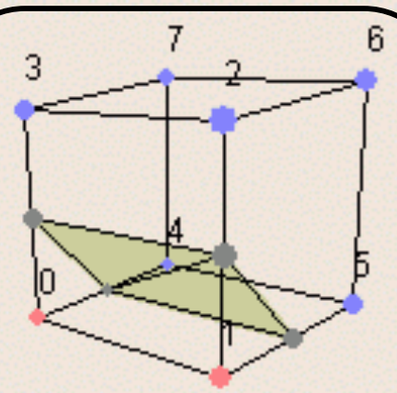
3 cases



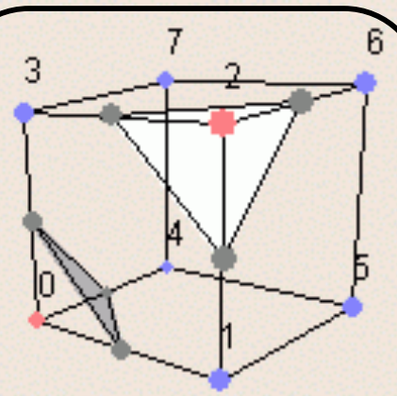
Case 0



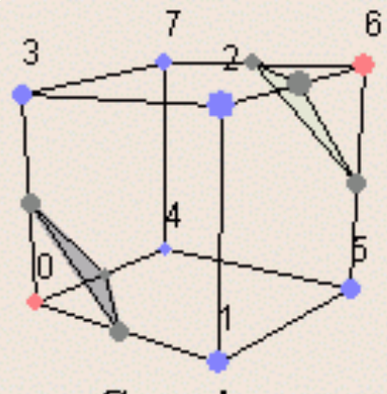
Case 1



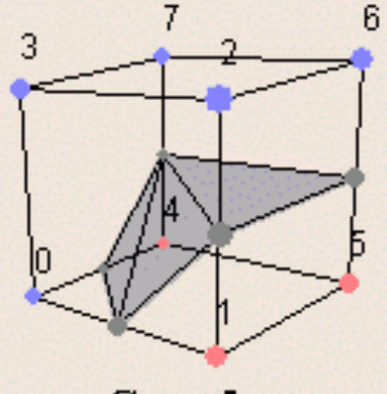
Case 2



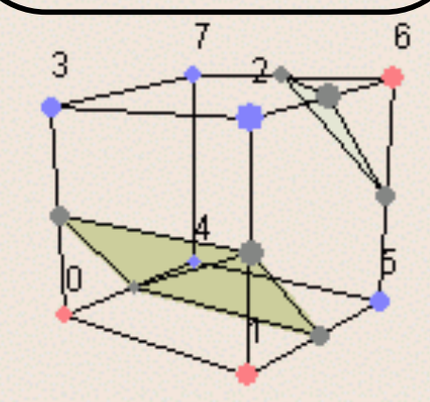
Case 3



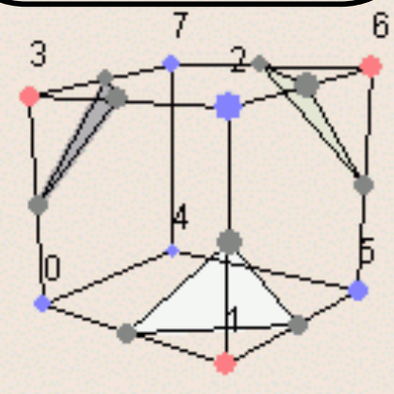
Case 4



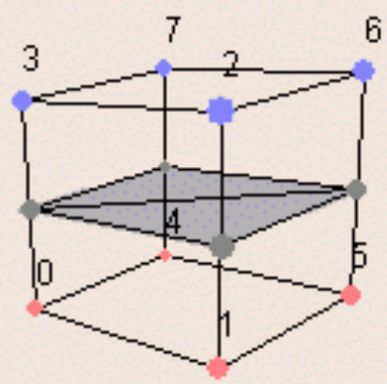
Case 5



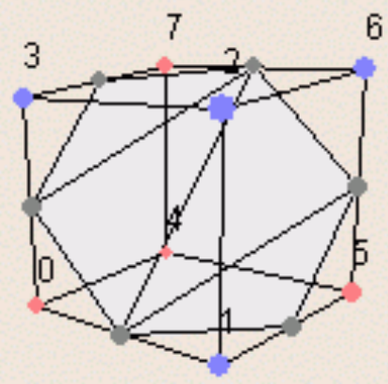
Case 6



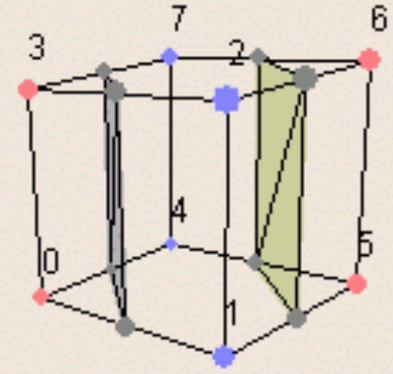
Case 7



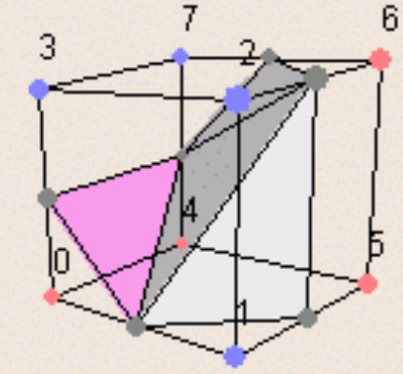
Case 8



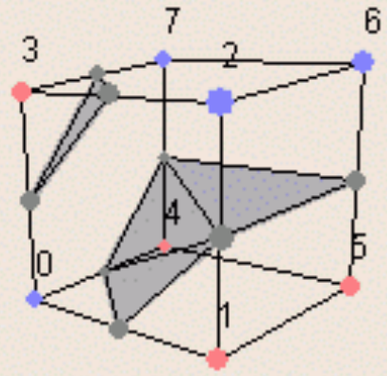
Case 9



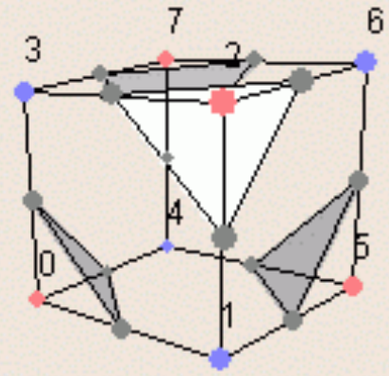
Case 10



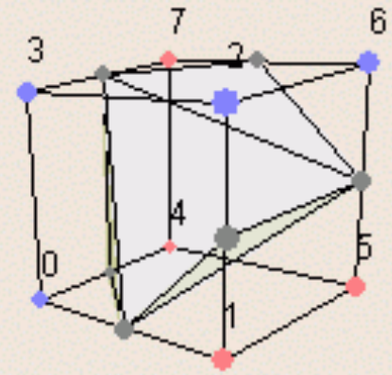
Case 11



Case 12



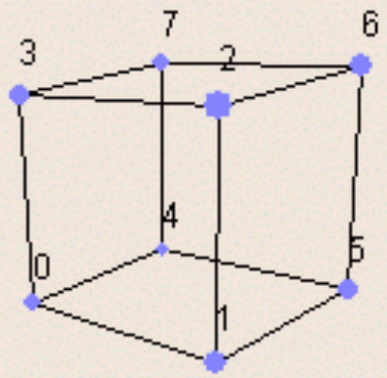
Case 13



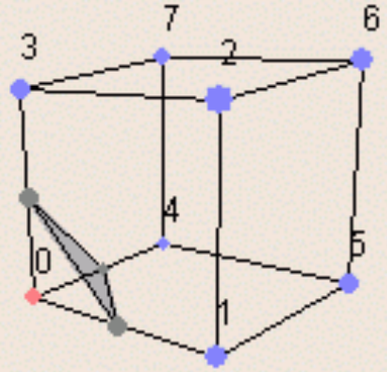
Case 14

- 5 Above
- 3 Below

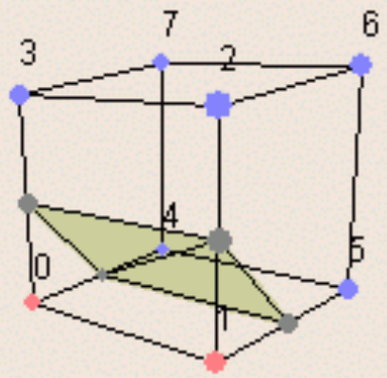
3 cases



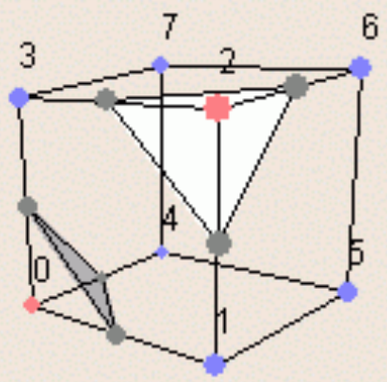
Case 0



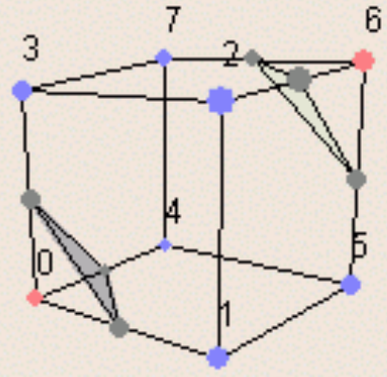
Case 1



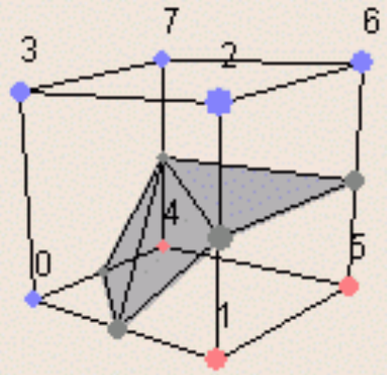
Case 2



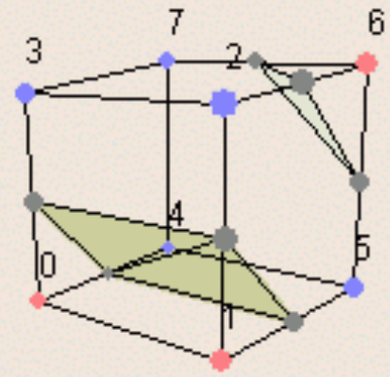
Case 3



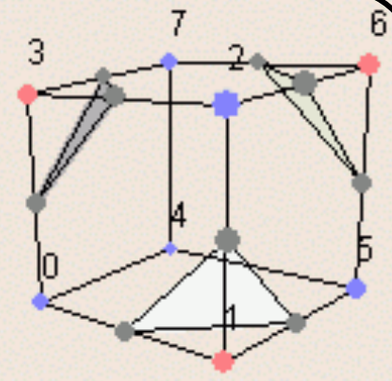
Case 4



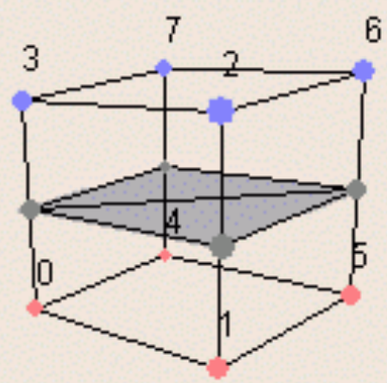
Case 5



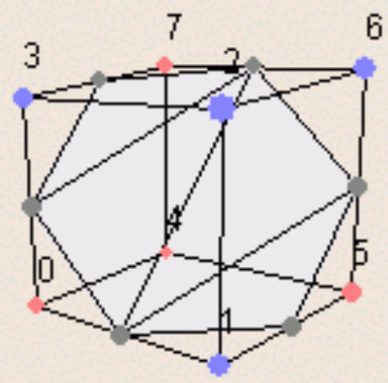
Case 6



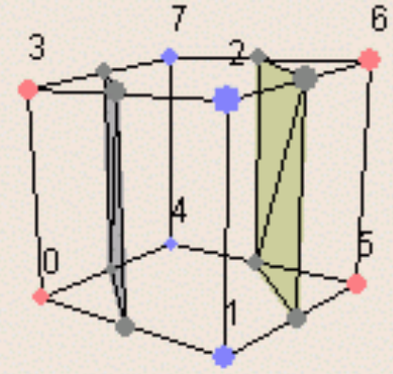
Case 7



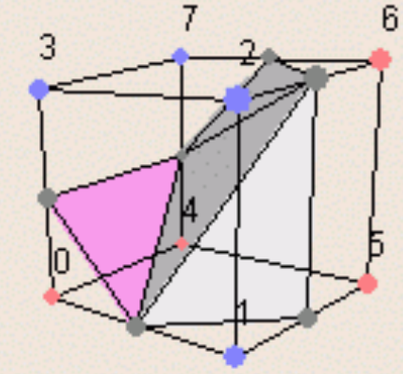
Case 8



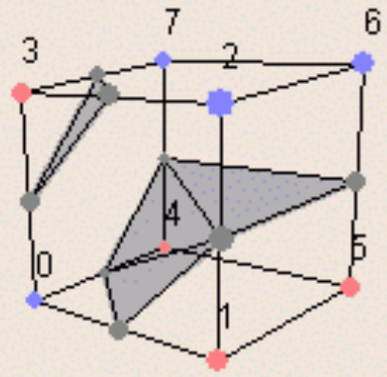
Case 9



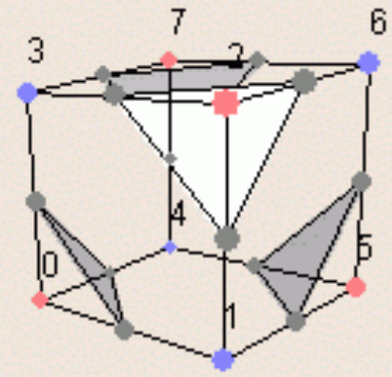
Case 10



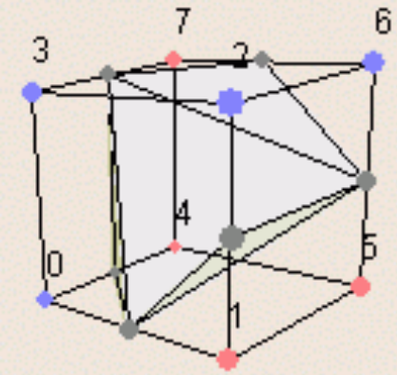
Case 11



Case 12



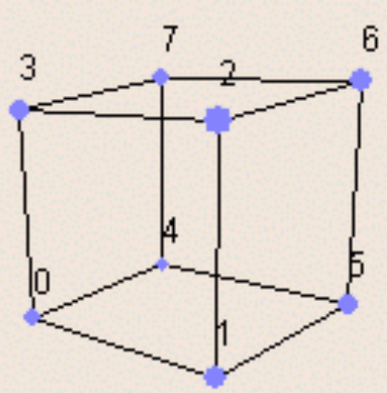
Case 13



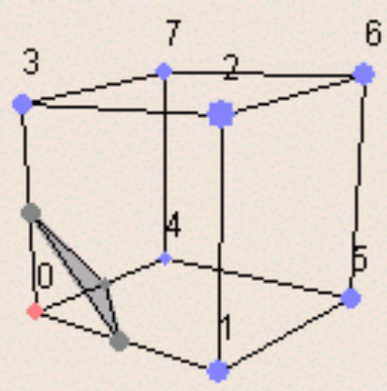
Case 14

- 4 Above
- 4 Below

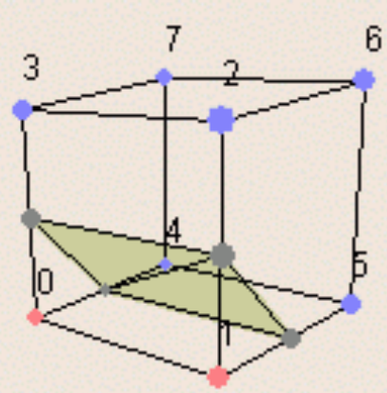
7 cases



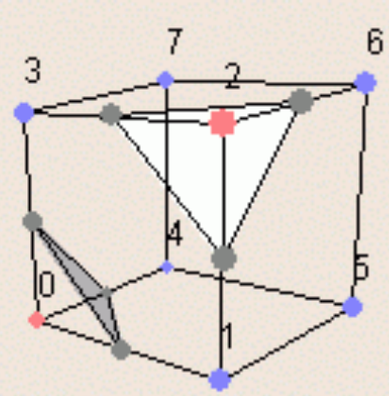
Case 0



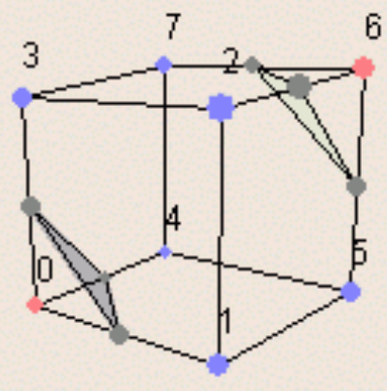
Case 1



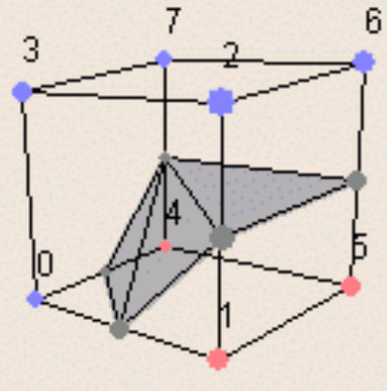
Case 2



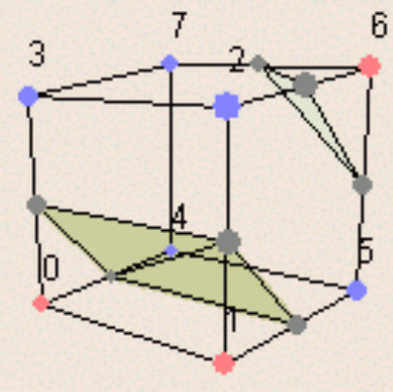
Case 3



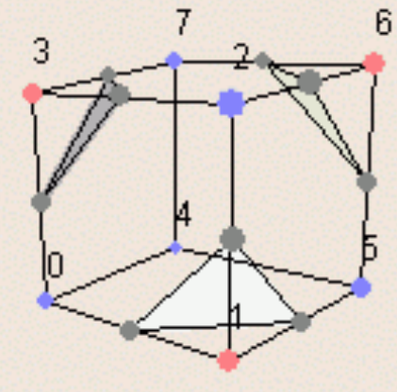
Case 4



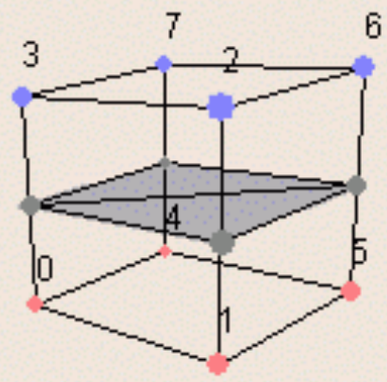
Case 5



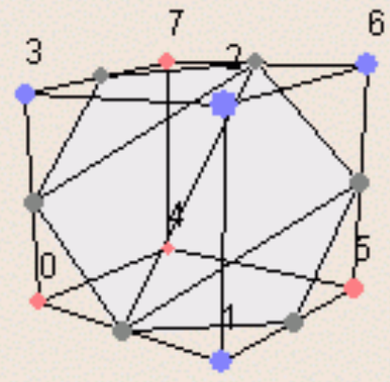
Case 6



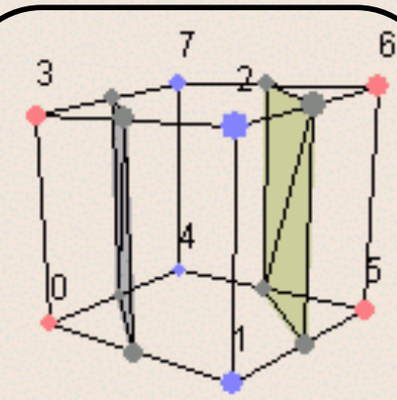
Case 7



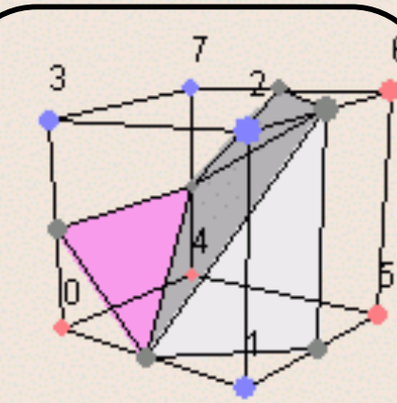
Case 8



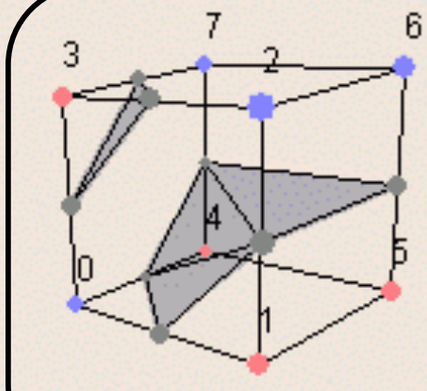
Case 9



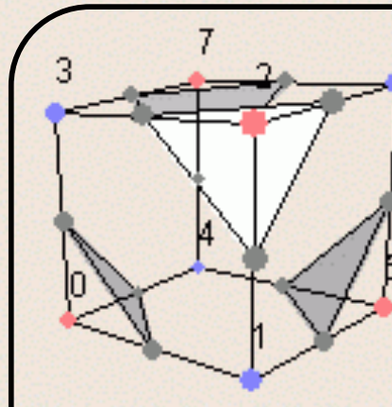
Case 10



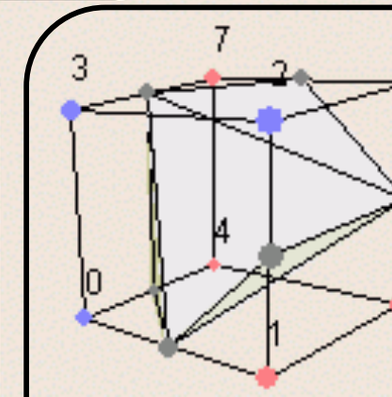
Case 11



Case 12



Case 13



Case 14

- Step 4 *cont.*: Get edge list from table
  - Example for

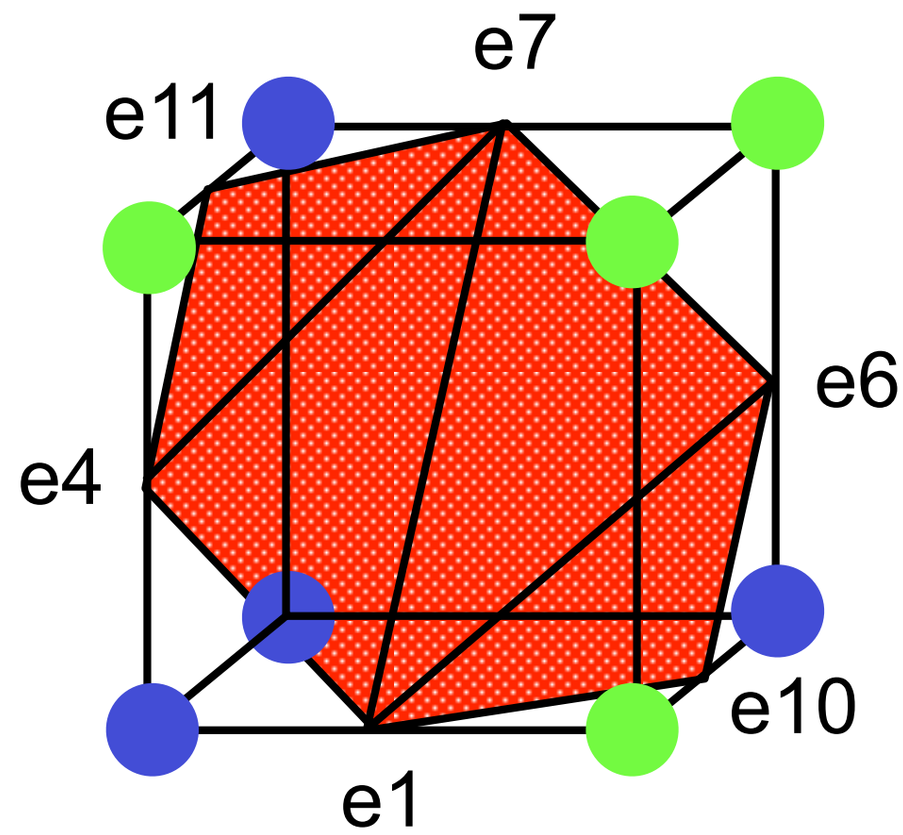
Index = 10110001

triangle 1 = e4,e7,e11

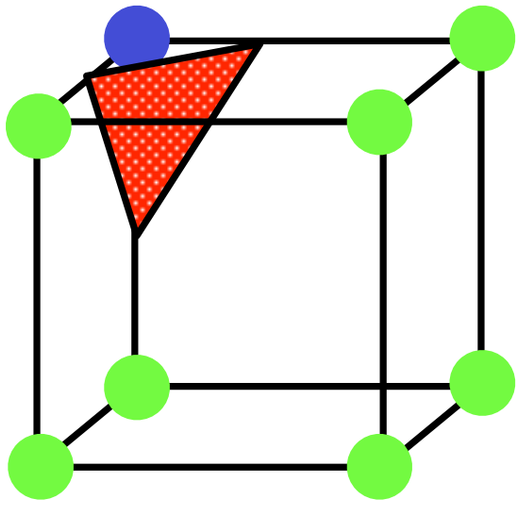
triangle 2 = e1, e7, e4

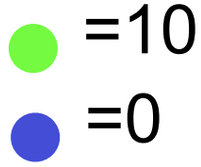
triangle 3 = e1, e6, e7

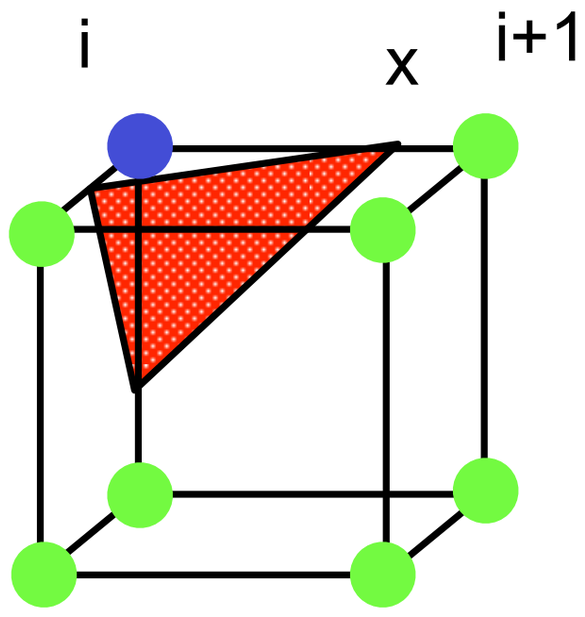
triangle 4 = e1, e10, e6



- Step 5: For each triangle edge, find the vertex location along the edge using linear interpolation of the voxel values





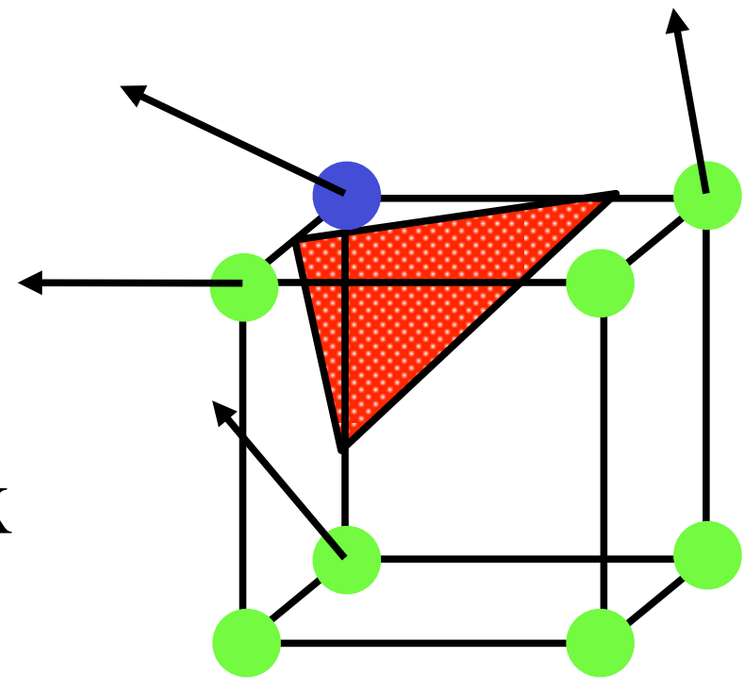


T=5

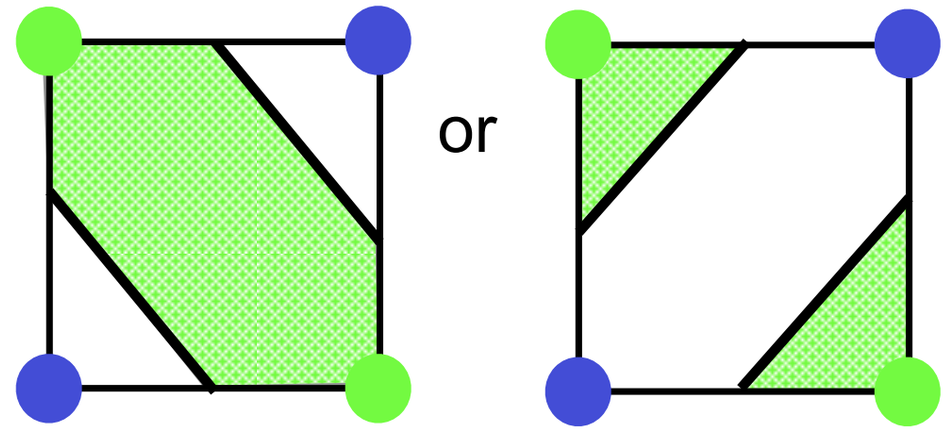
$$x = i + \left(\frac{T - v[i]}{v[i+1] - v[i]} \right)$$

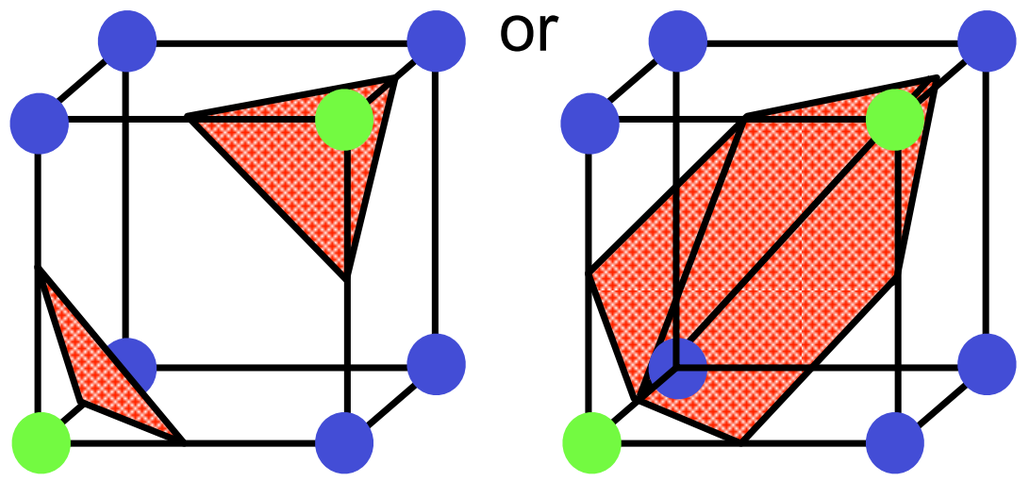
T=8

- Step 6: Calculate the normal at each cube vertex (central differences)
  - $G_x = V_{x+1,y,z} - V_{x-1,y,z}$
    $G_y = V_{x,y+1,z} - V_{x,y-1,z}$
    $G_z = V_{x,y,z+1} - V_{x,y,z-1}$
  - Use linear interpolation to compute the polygon vertex normal (of the isosurface)

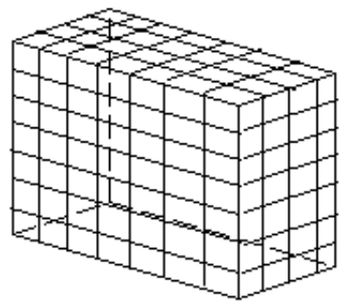


- Step 7: Consider ambiguous cases
  - Ambiguous cases: 3, 6, 7, 10, 12, 13
  - Adjacent vertices: different states
  - Diagonal vertices: same state
  - Resolution: choose one case (the right one!)

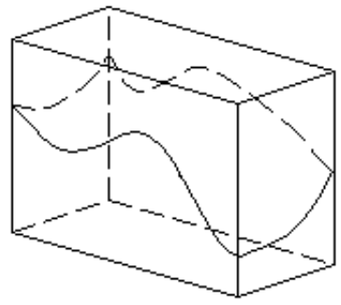




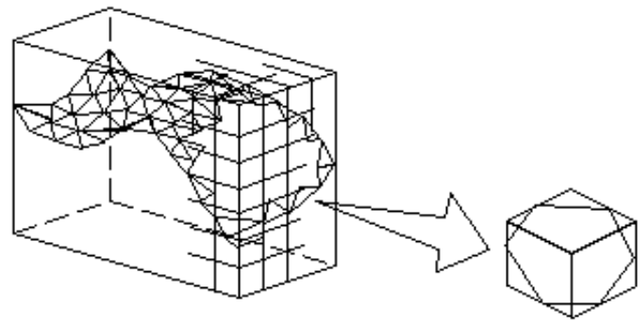
- Summary
  - 256 Cases
  - Reduce to 15 cases by symmetry
  - Ambiguity in cases
    3, 6, 7, 10, 12, 13
  - Causes holes if arbitrary choices are made



(a) Volume data



(b) Isosurface
S = f(x,y,z)

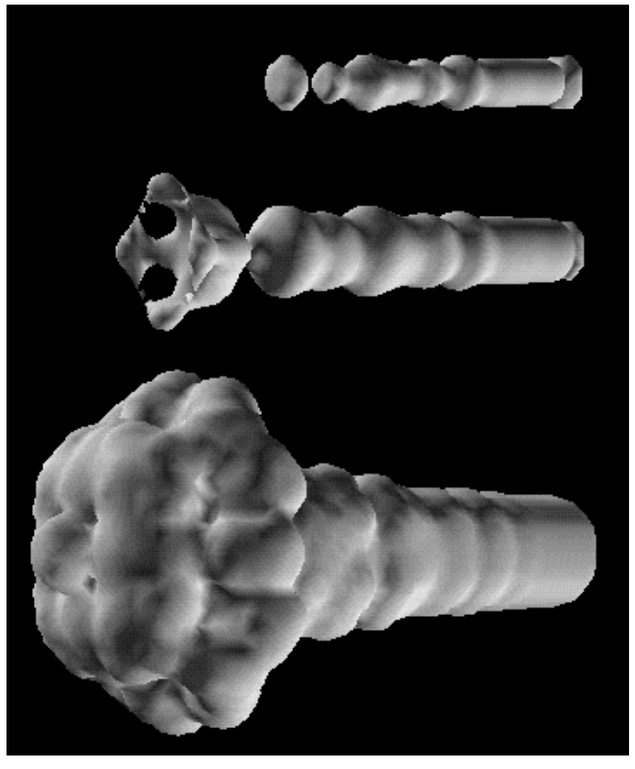


(c) Polygonal Apploximation

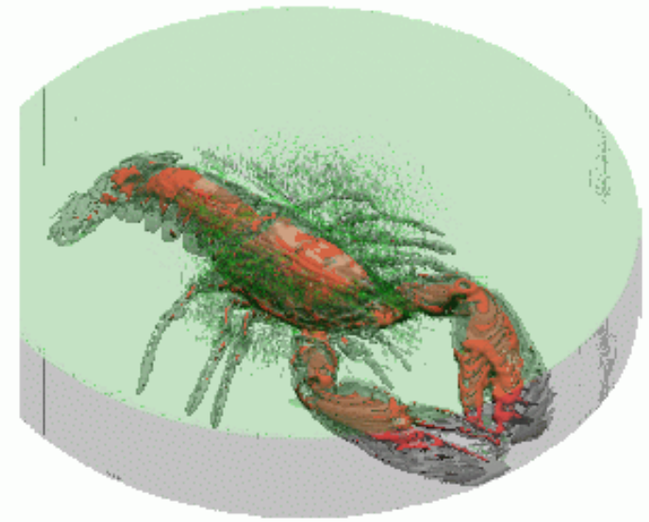
- Up to 5 triangles per cube
- Several isosurfaces
  - Run MC several times
  - Semi-transparency requires spatial sorting

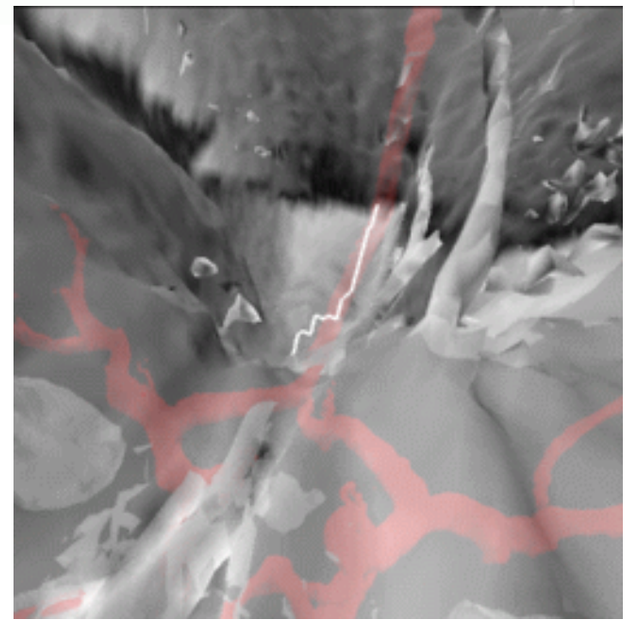
- Examples

1 Isosurface



3 Isosurfaces





2 Isosurfaces

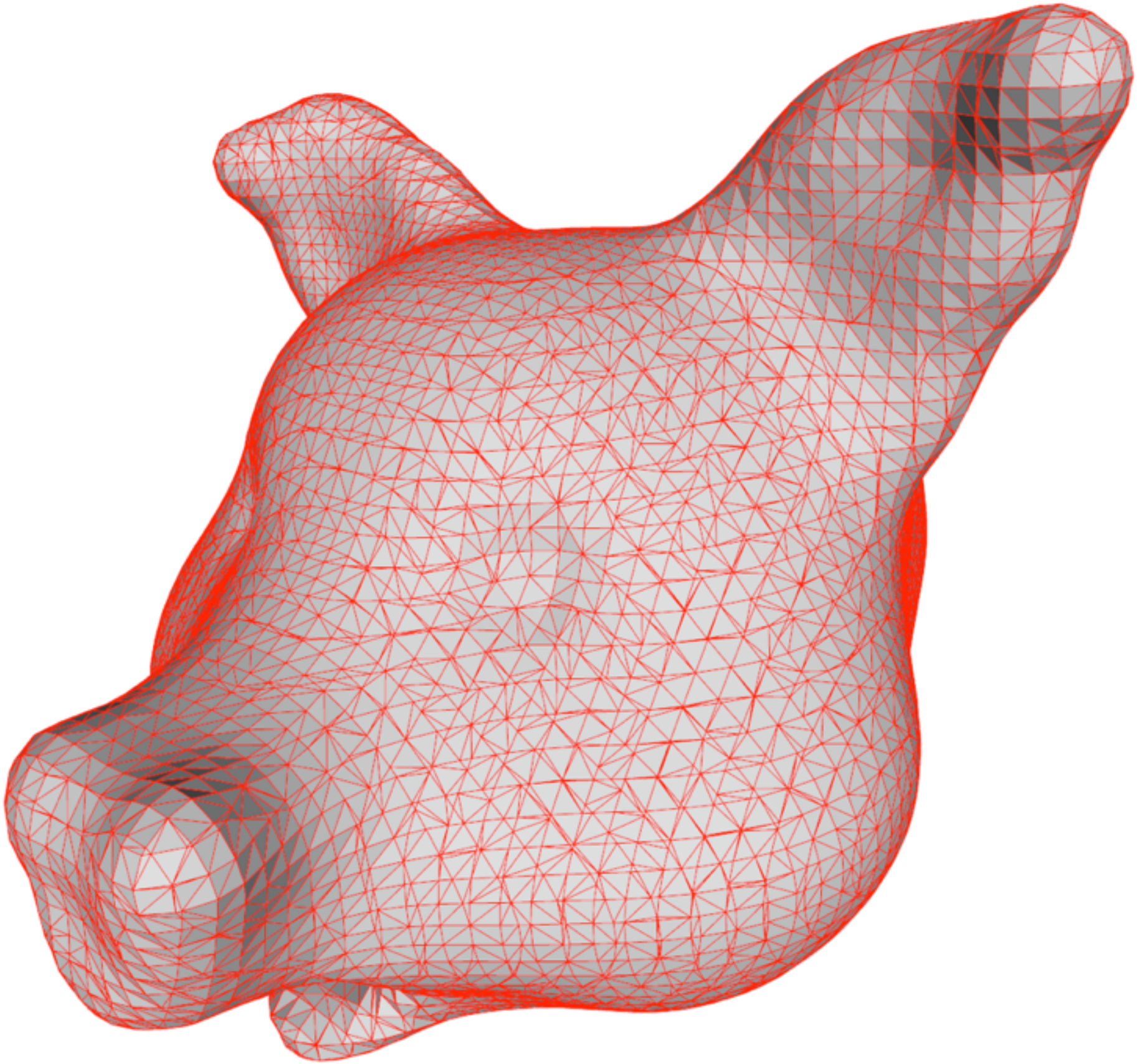
challenges

- ambiguities

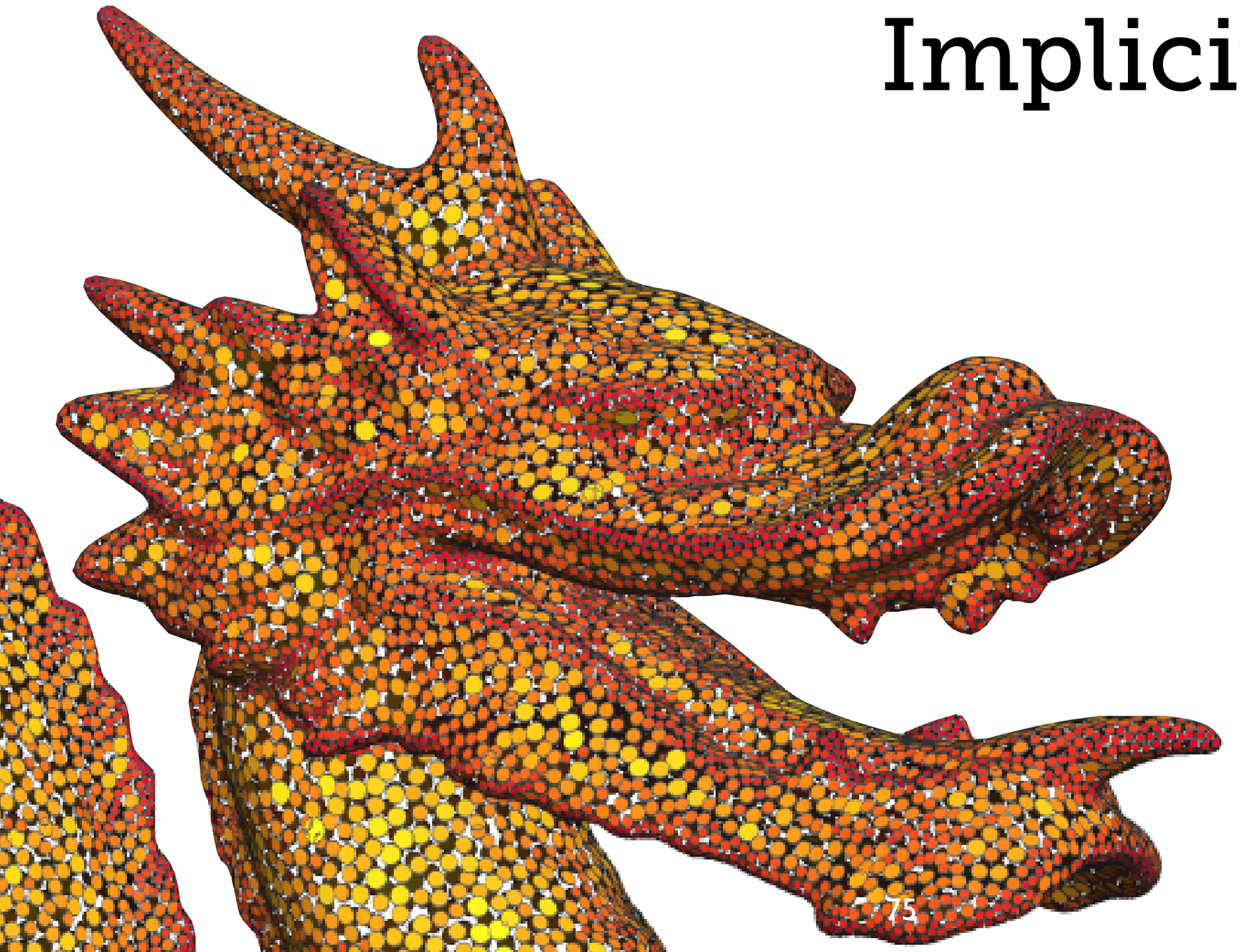
- looking at every voxel

- what is a good isovalue?

- poorly shaped, nonadaptive triangles



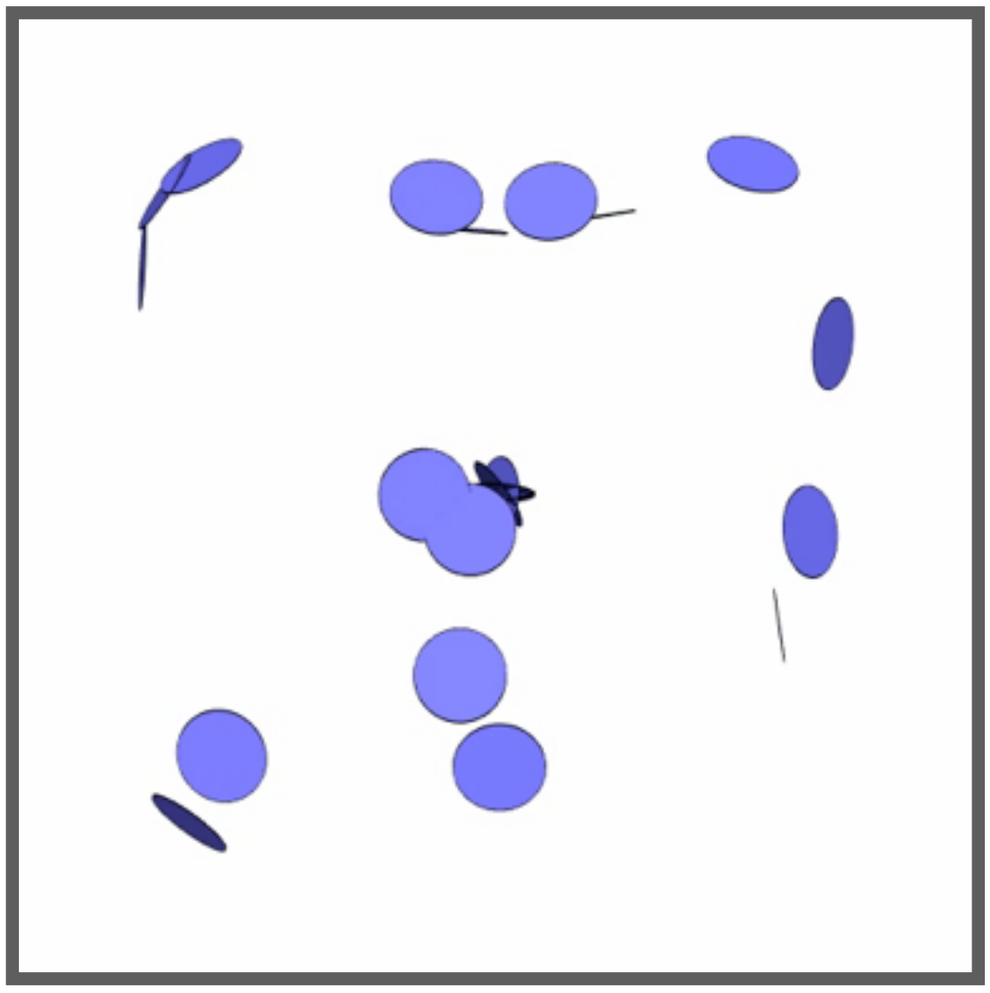
Dynamic Particles for Adaptive Sampling of Implicit Surfaces



dynamic particle system

Robust Particle Systems for Curvature Dependent Sampling of Implicit Surfaces

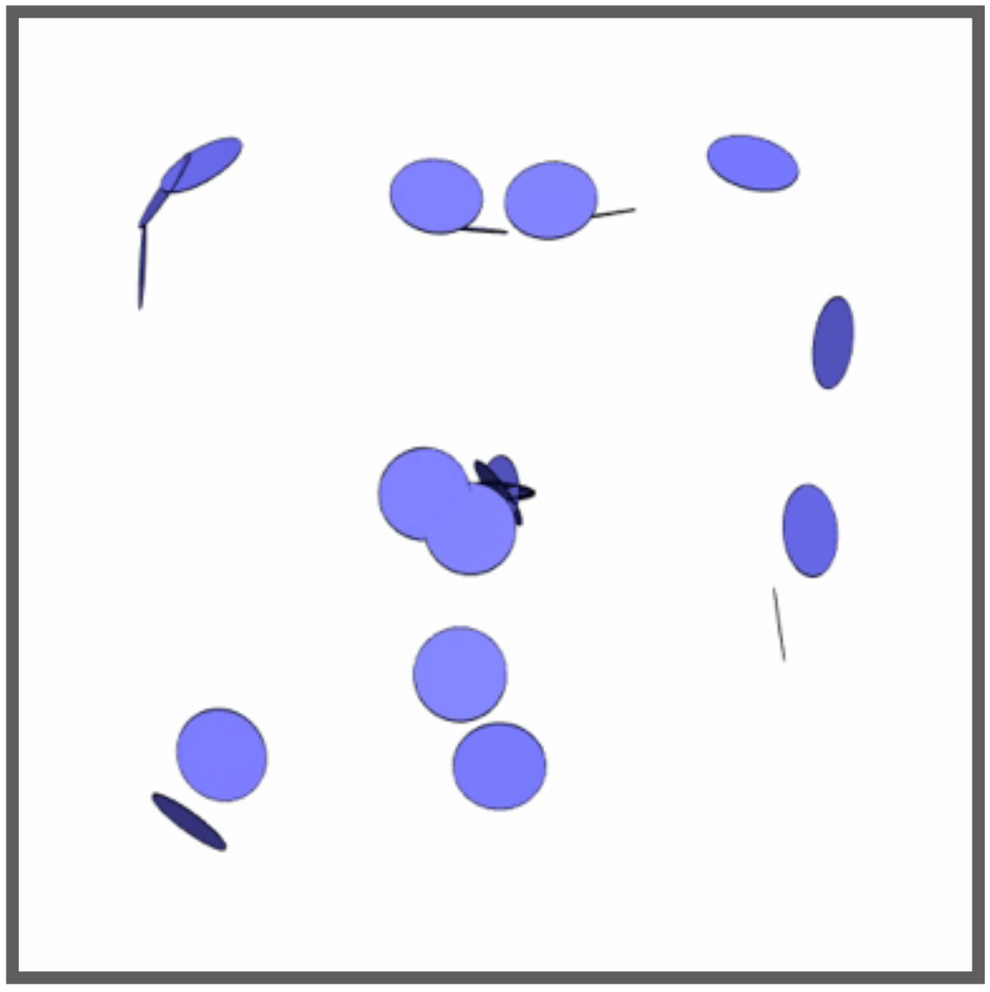
M. Meyer, P. Georgel, R. Whitaker, SMI 2005.

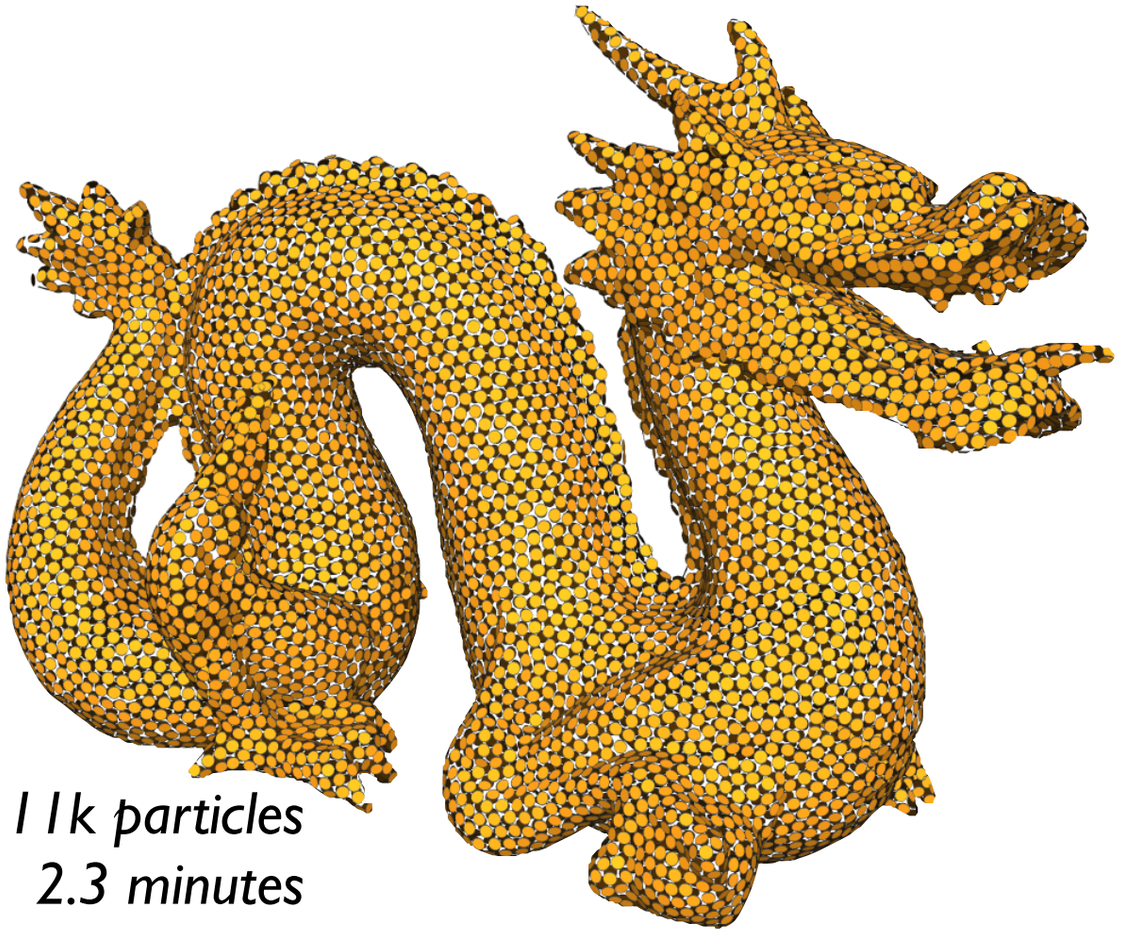


dynamic particle system

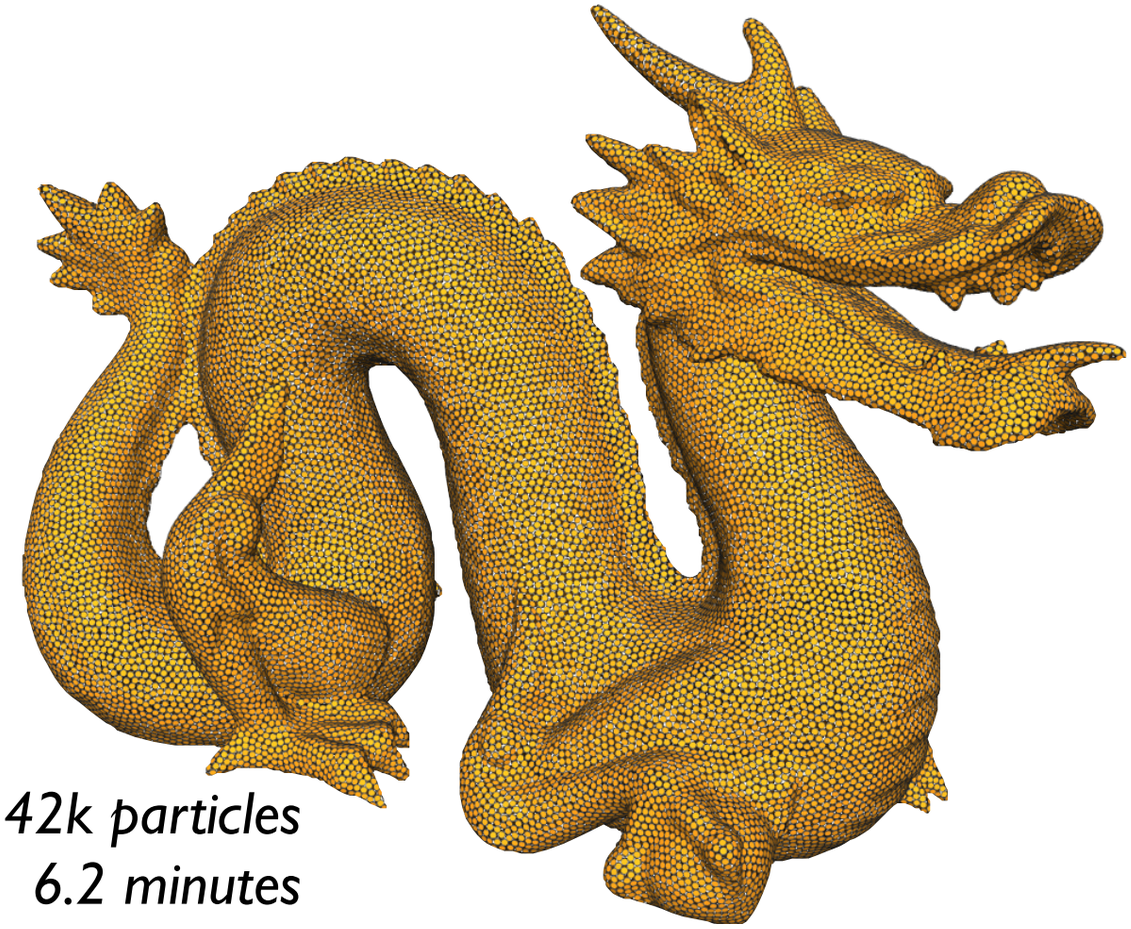
Robust Particle Systems for Curvature Dependent Sampling of Implicit Surfaces

M. Meyer, P. Georgel, R. Whitaker, SMI 2005.

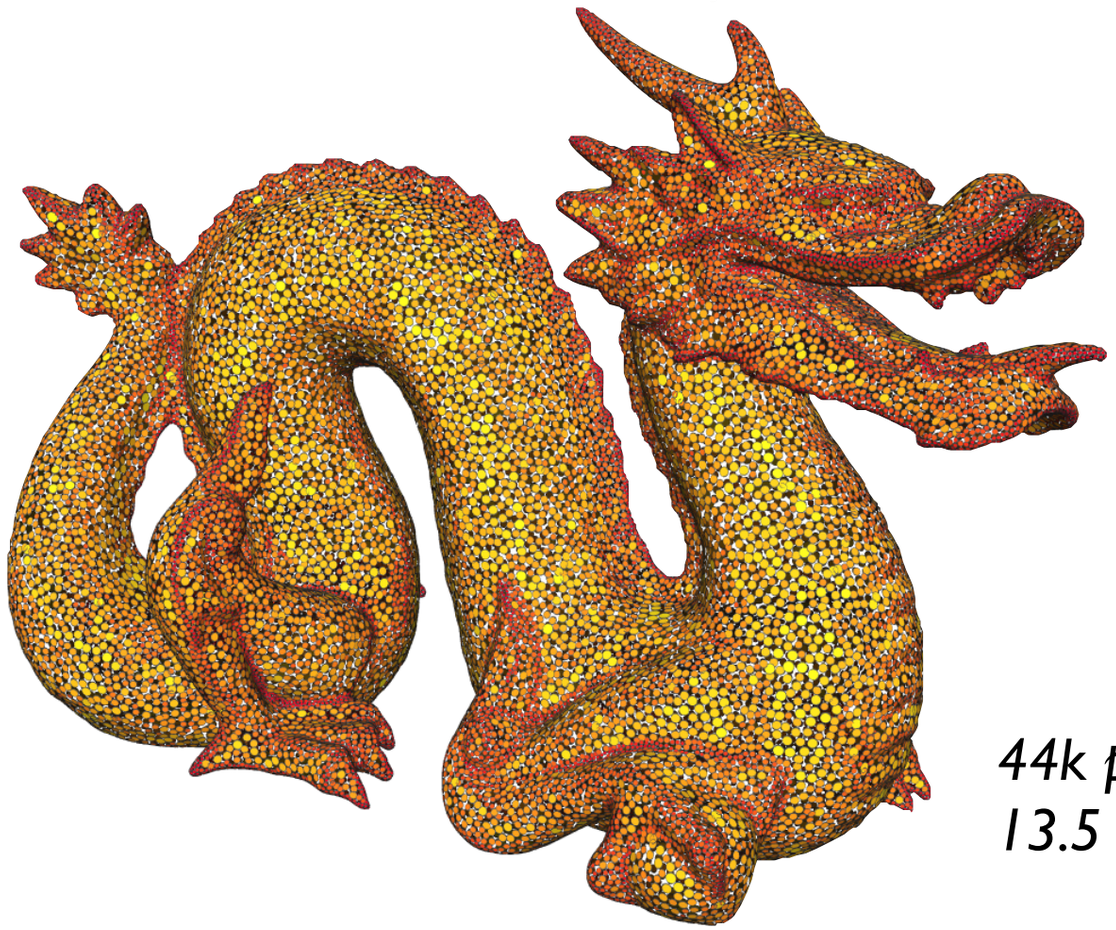




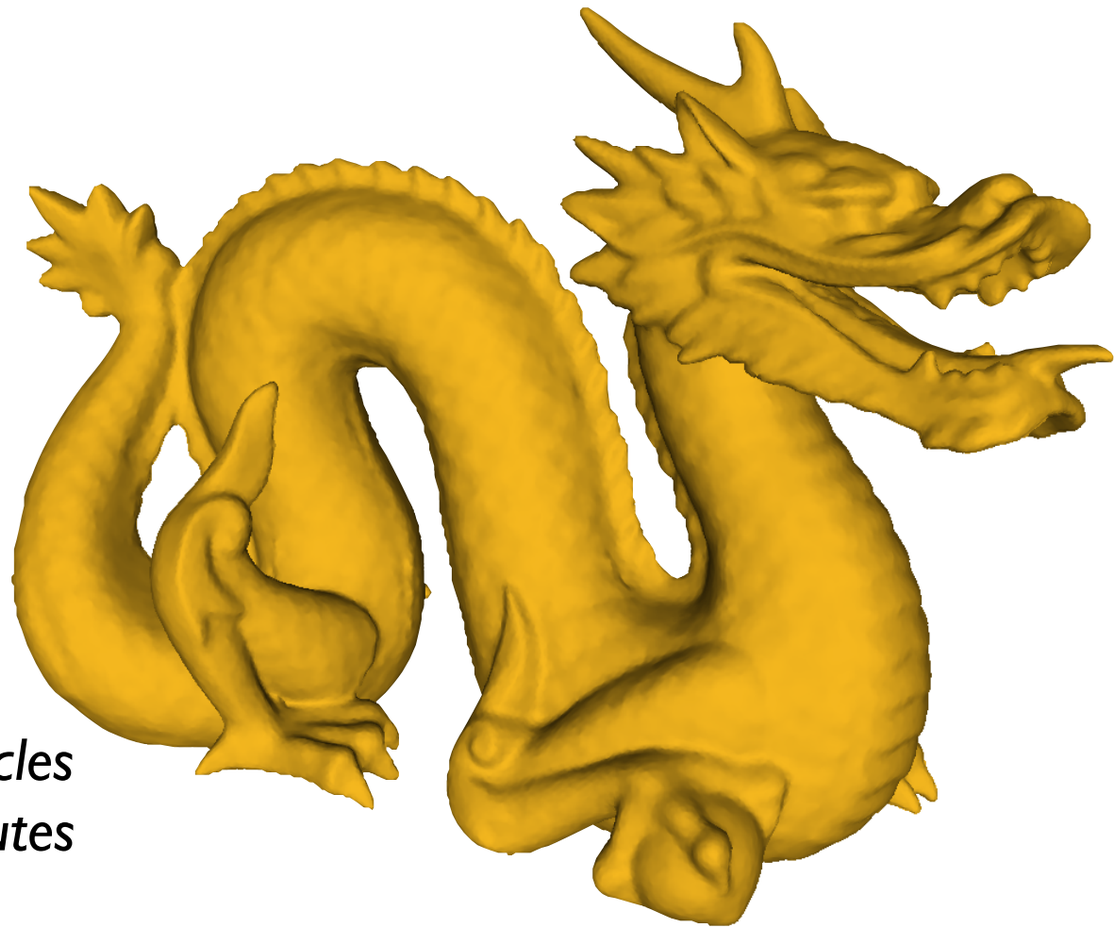
11k particles
2.3 minutes

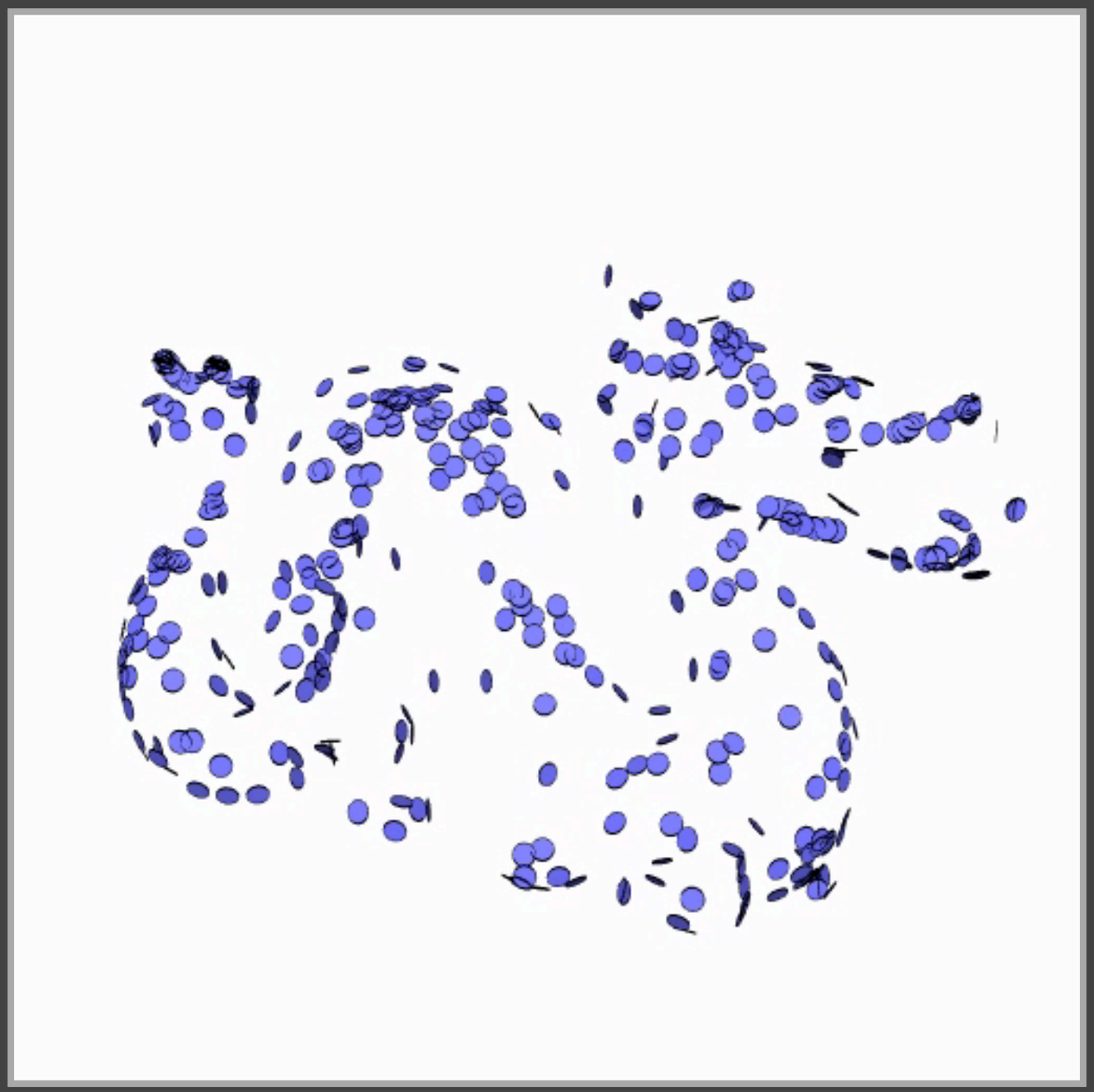


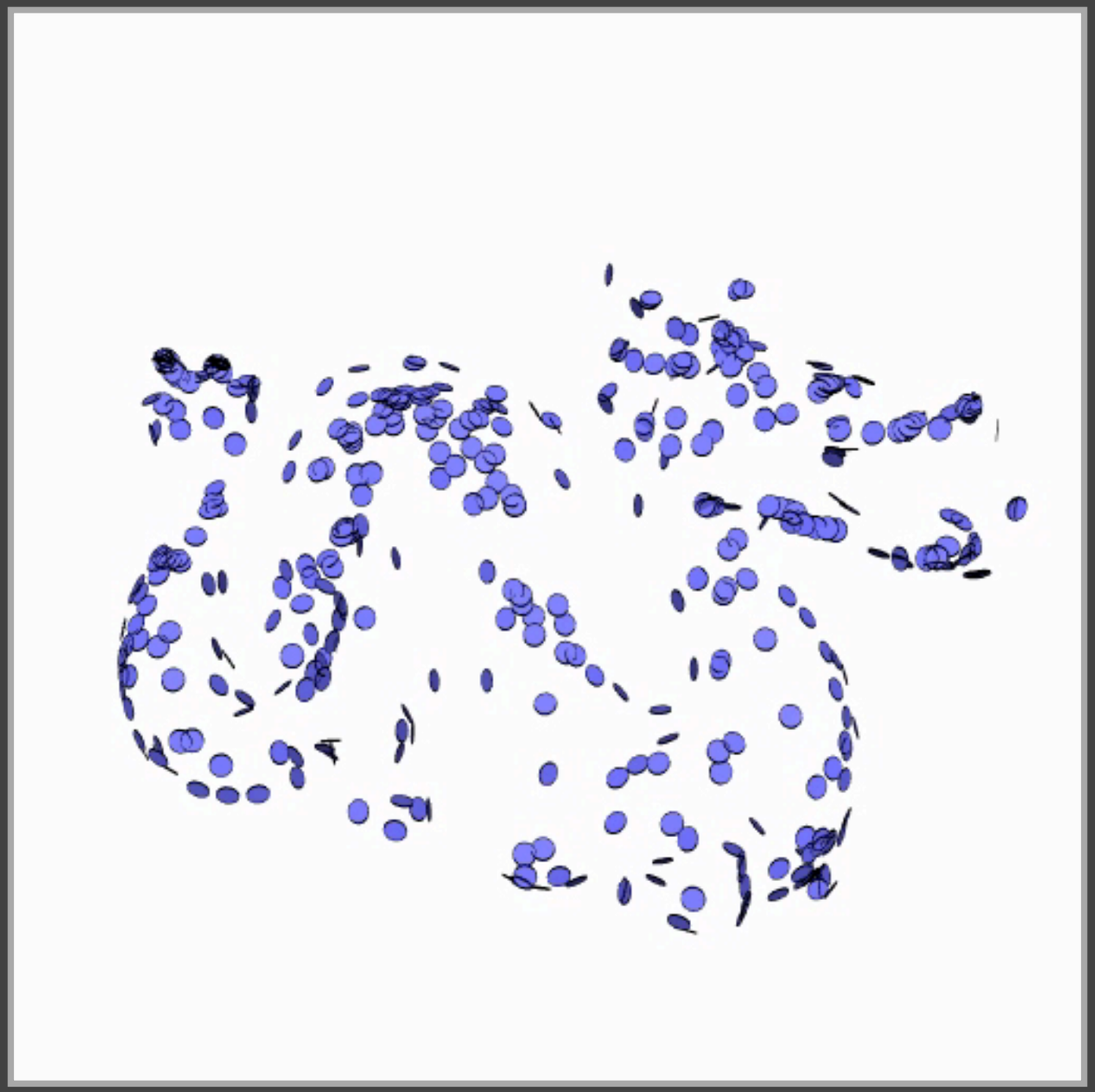
42k particles
6.2 minutes



44k particles
13.5 minutes

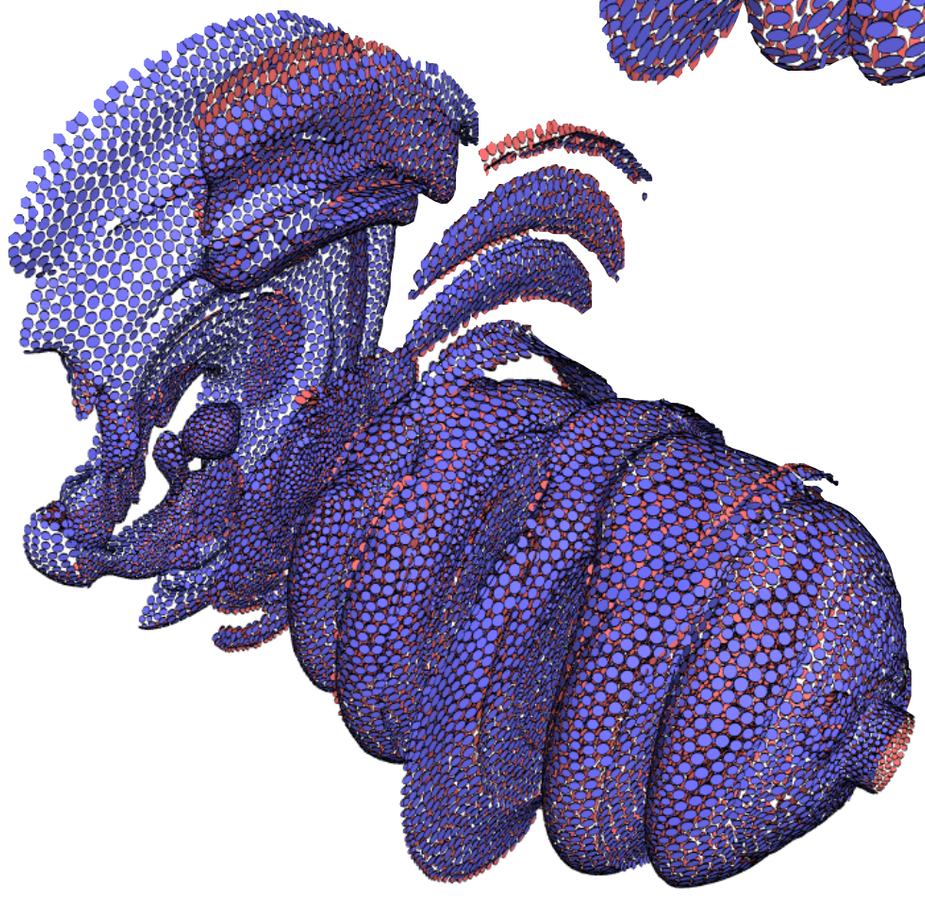


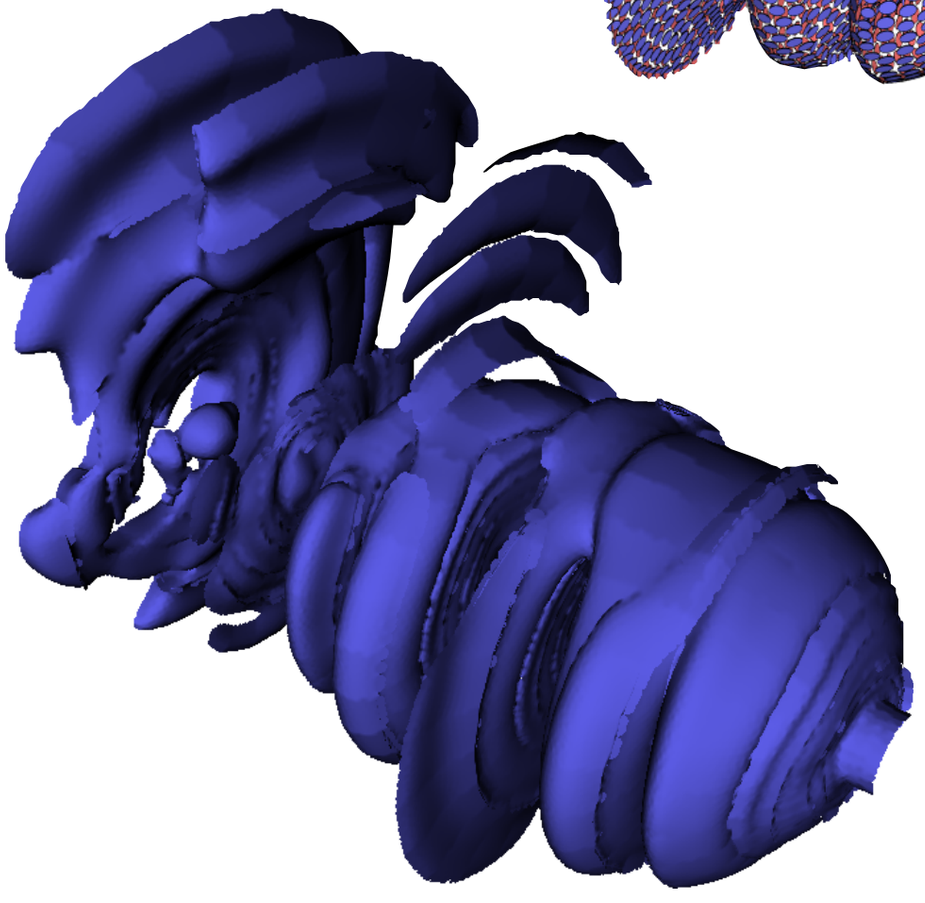




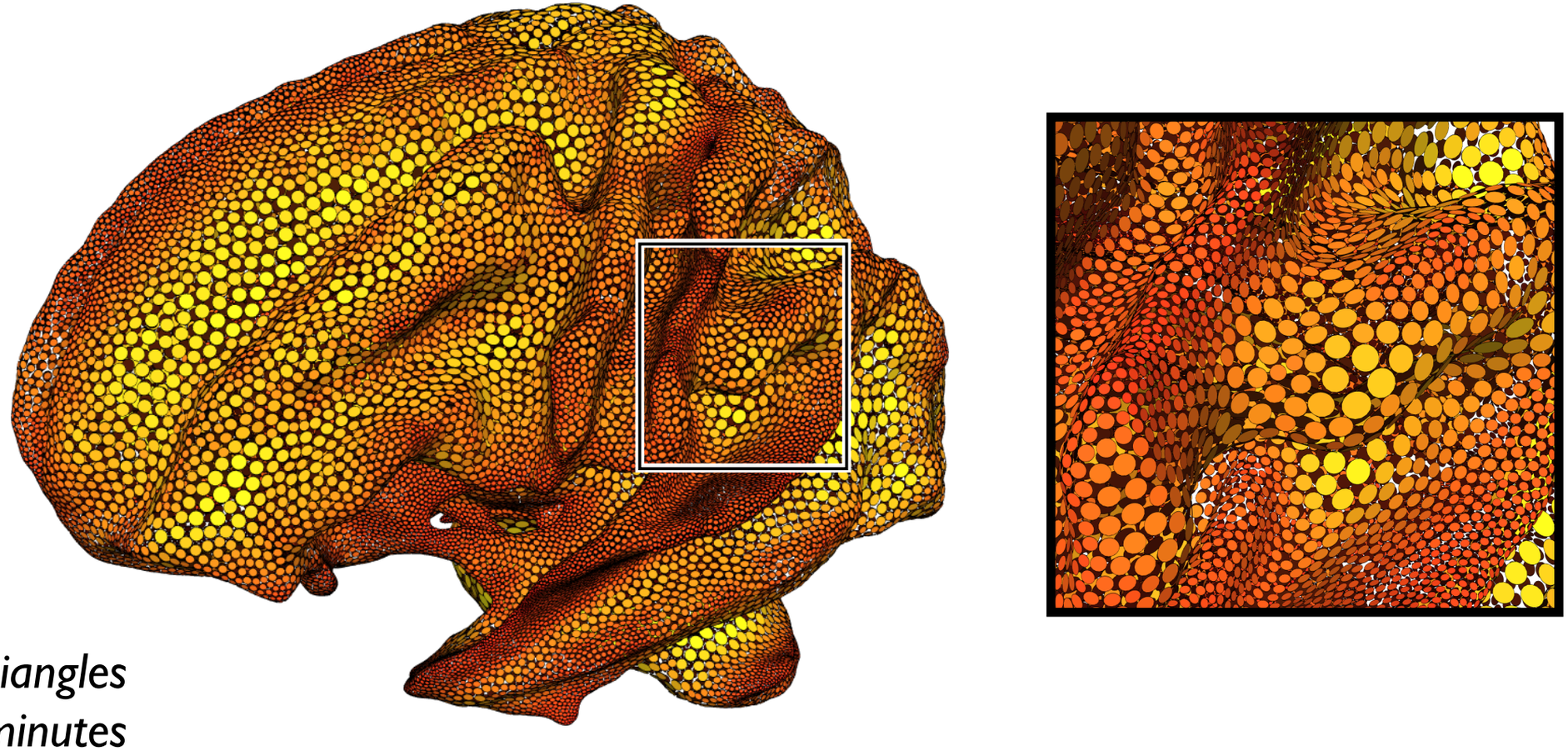
5k particles
0.5 minutes

13k particles
3.4 minutes

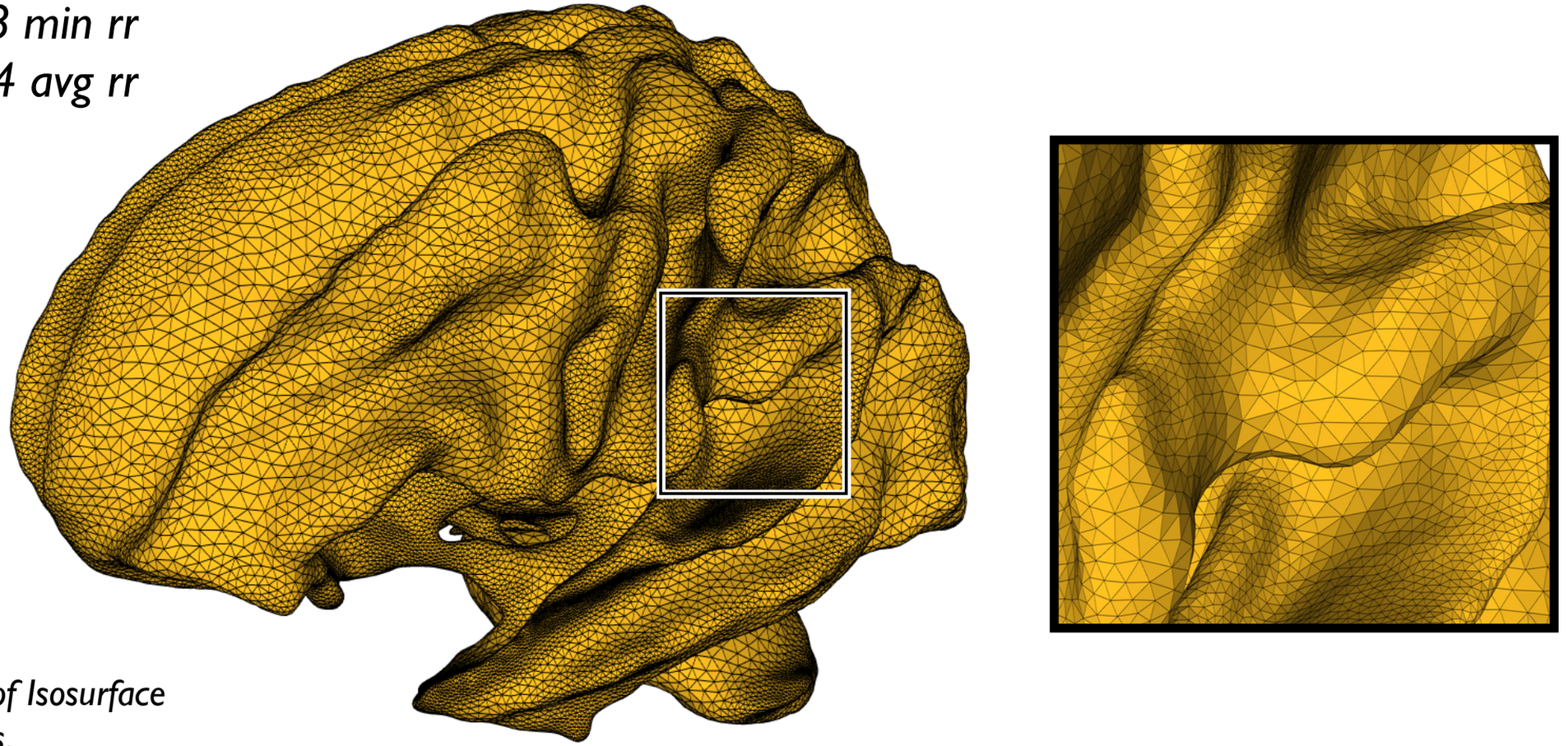




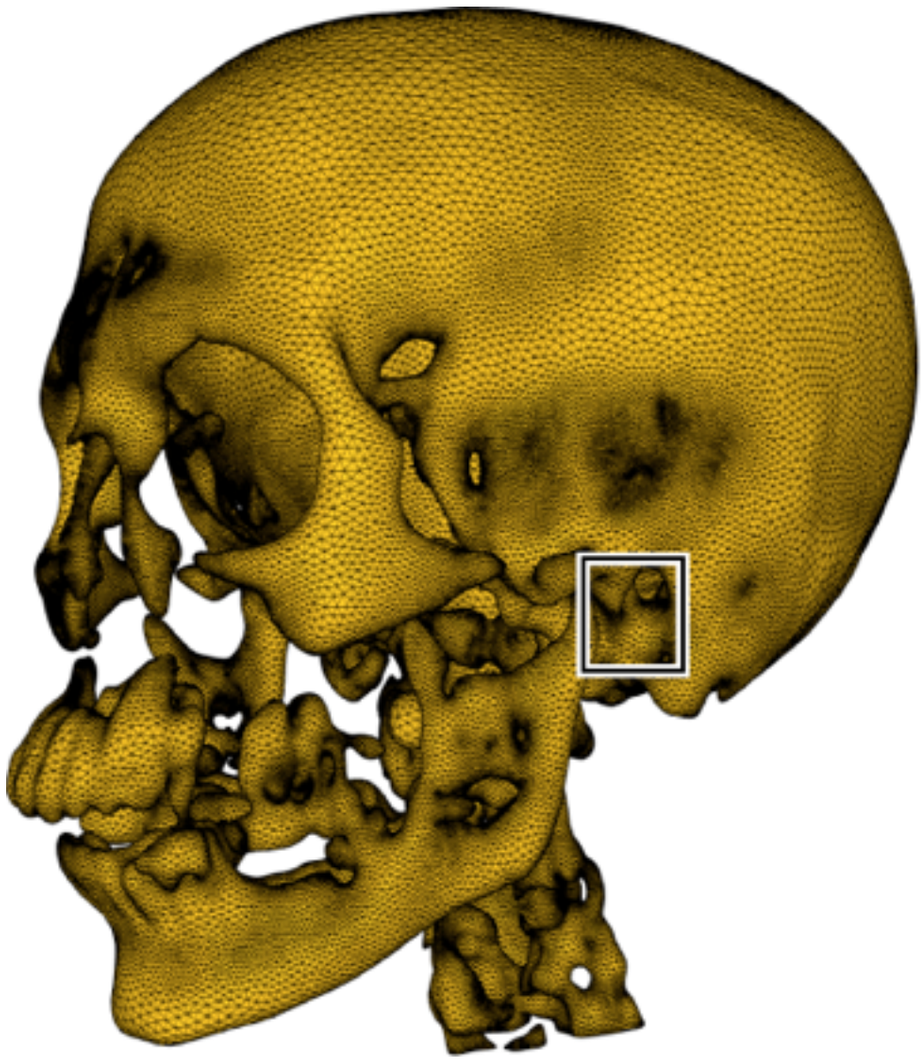
Particle Systems for Efficient and Accurate High-Order Finite Element Visualization
M. Meyer et al., TVCG 2006.

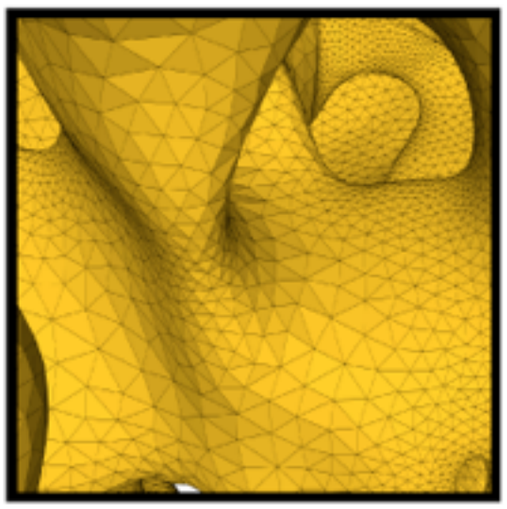


182k triangles
41 minutes
0.18 min rr
0.94 avg rr

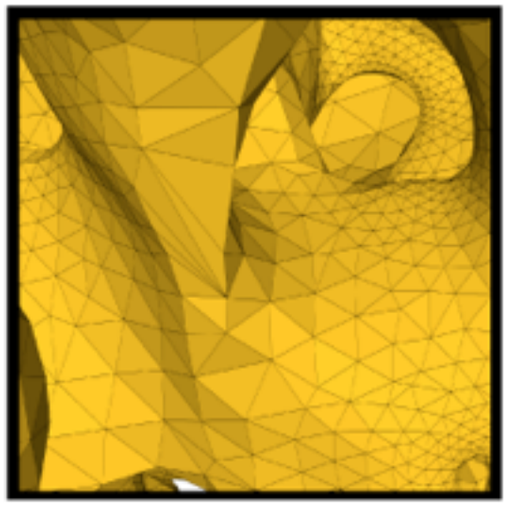


Topology, Accuracy, and Quality of Isosurface Meshes Using Dynamic Particles.
M. Meyer et al., Vis 2007.

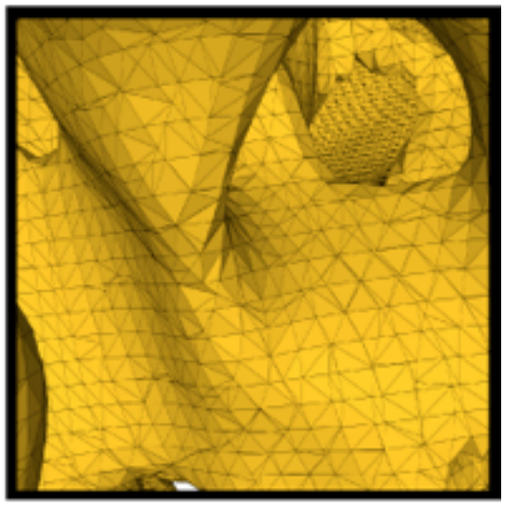




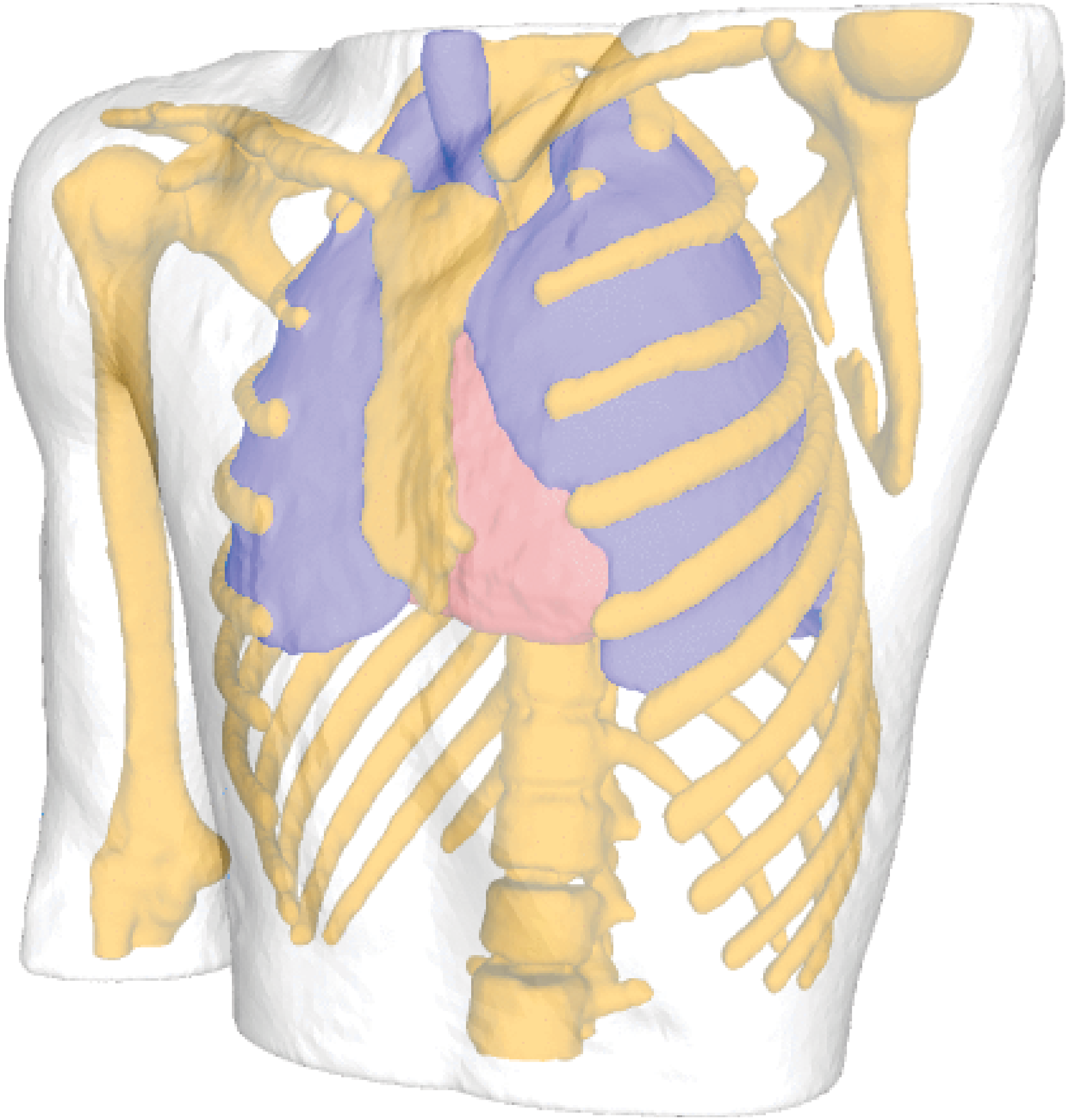
particle system

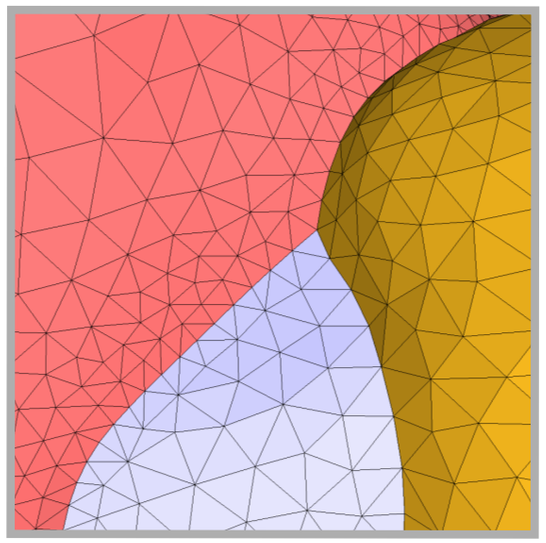


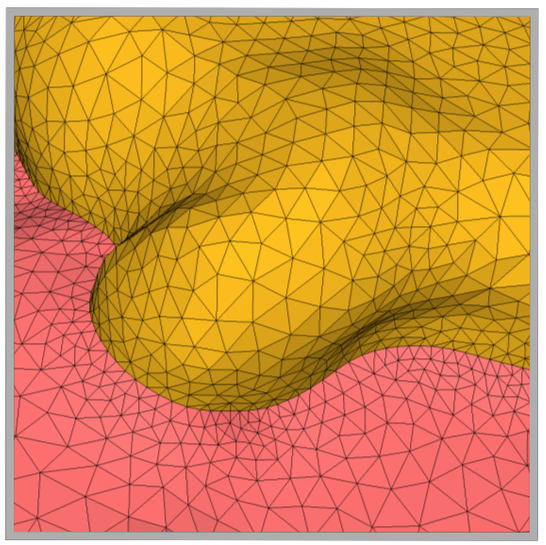
advancing front

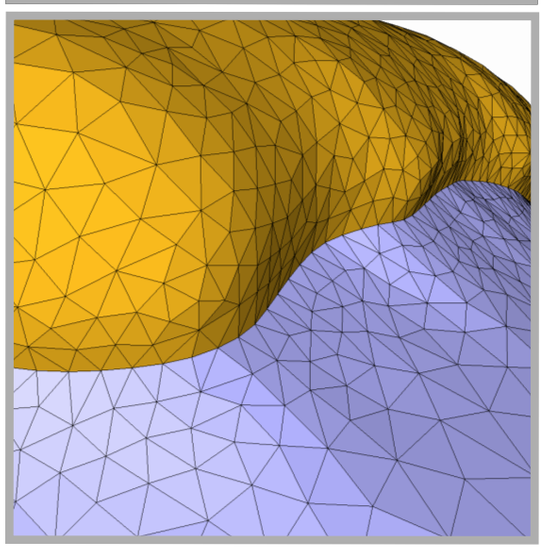


marching cubes

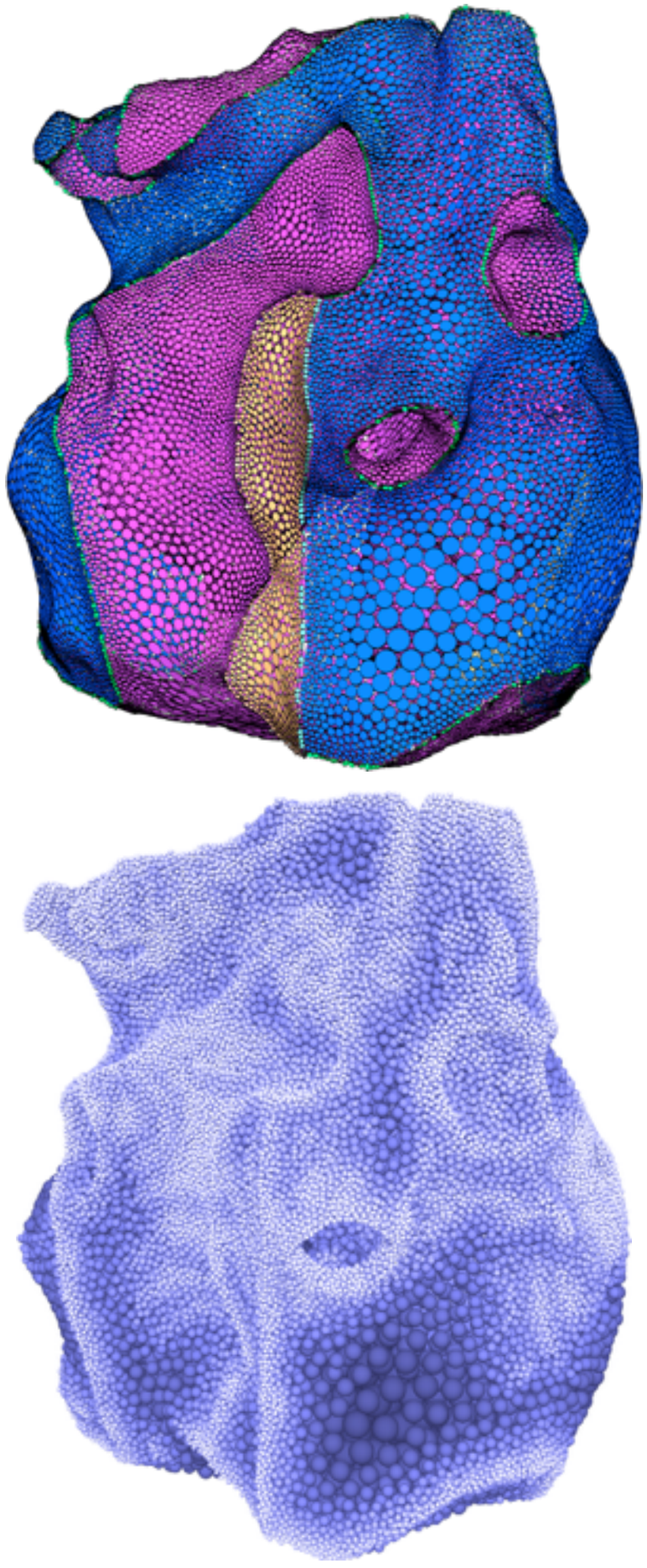


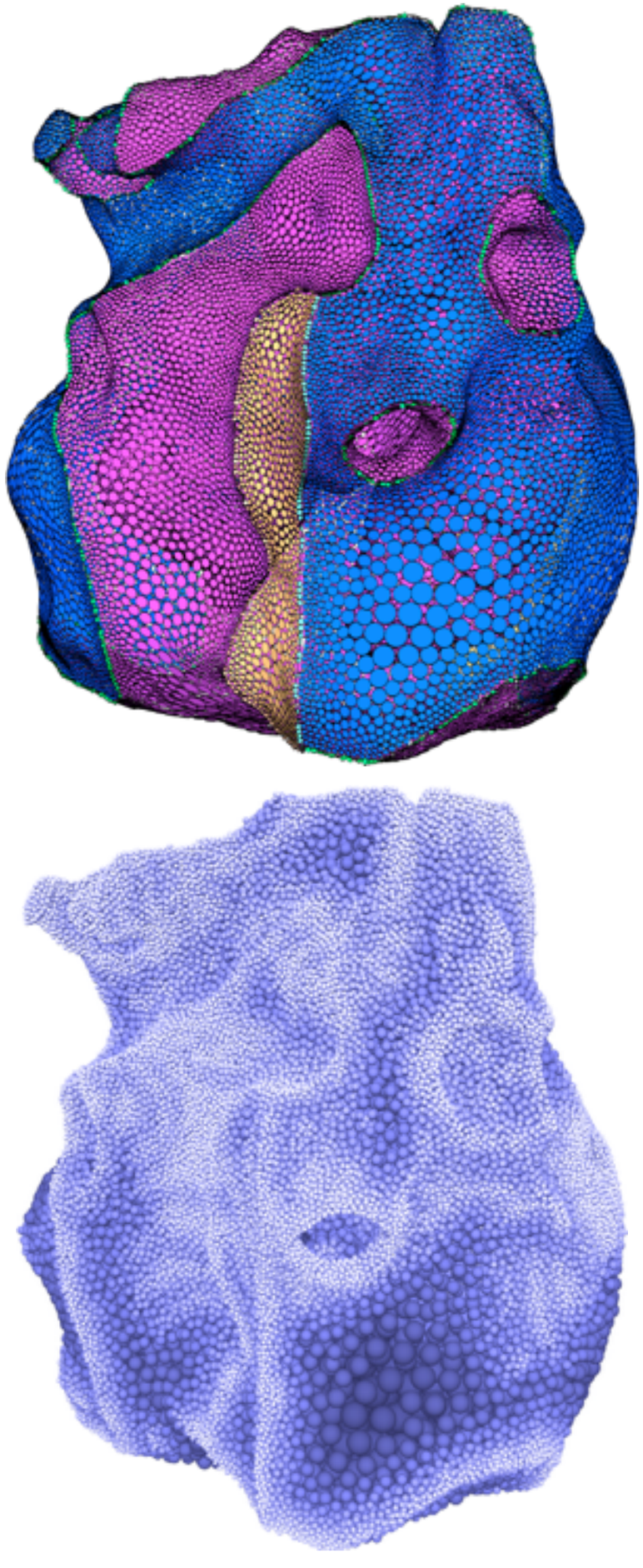


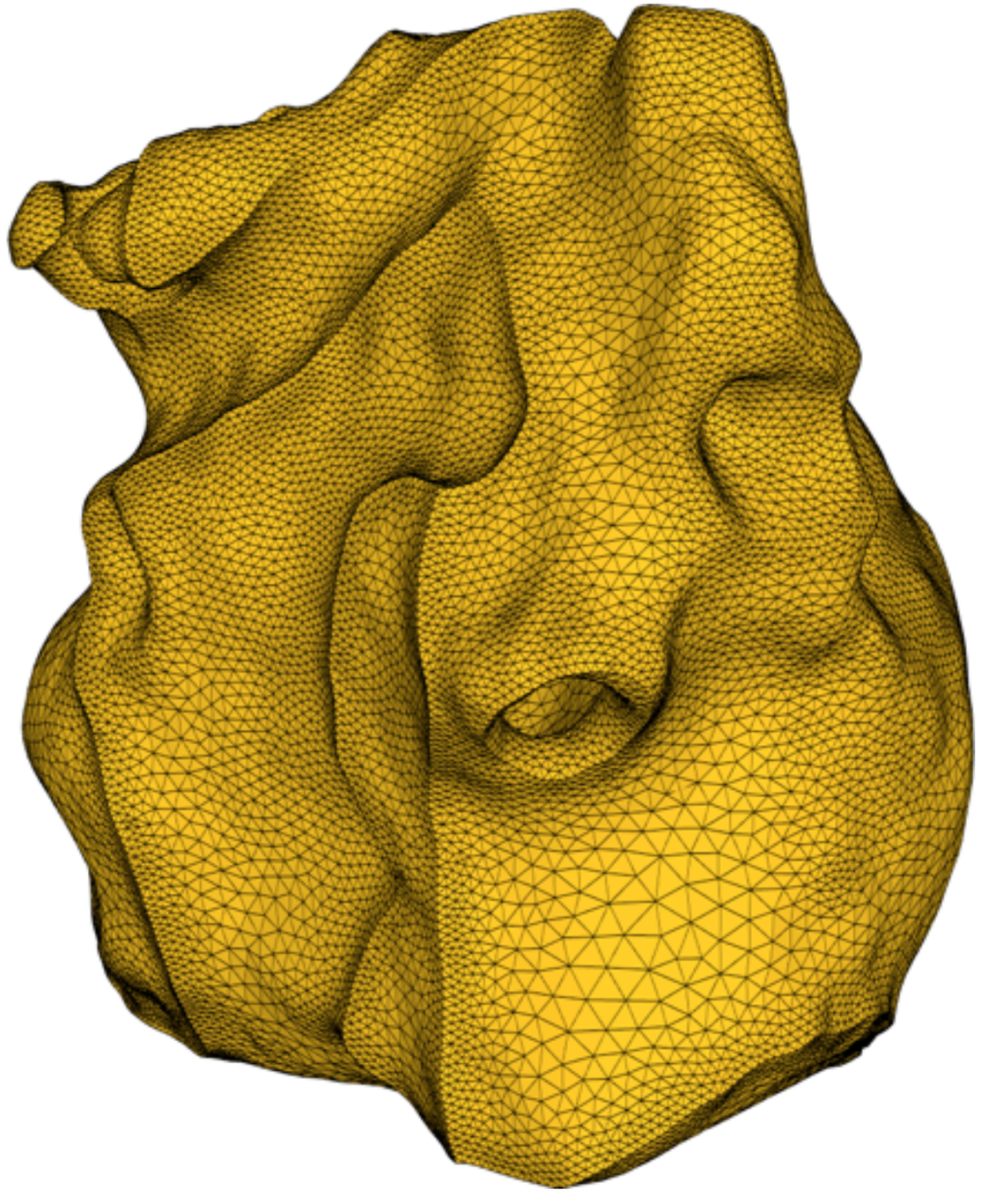


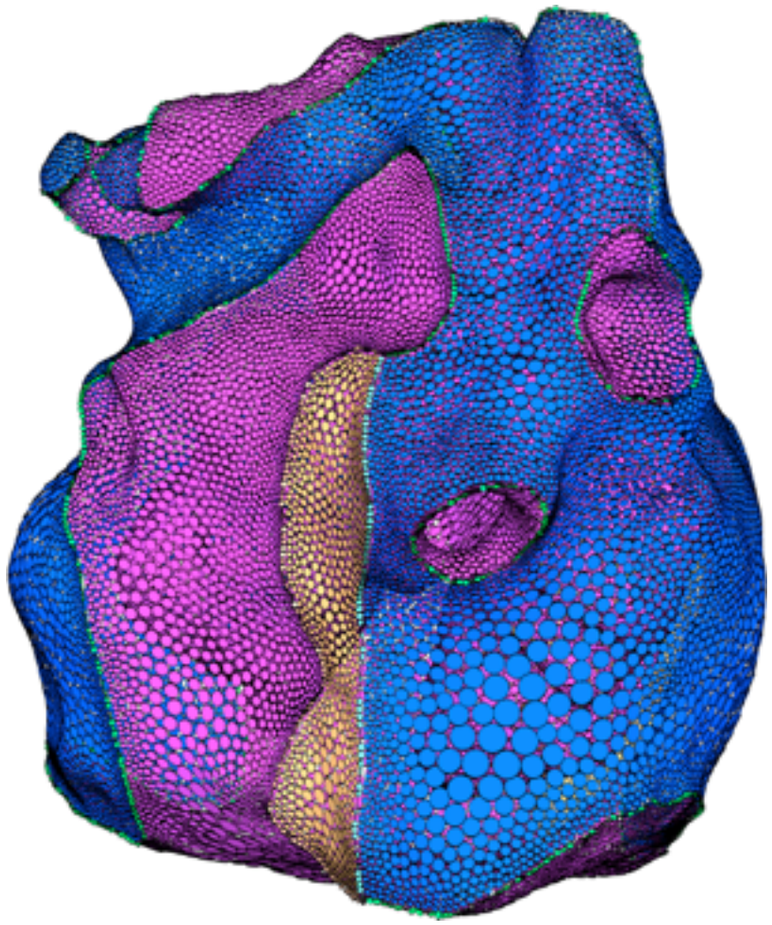


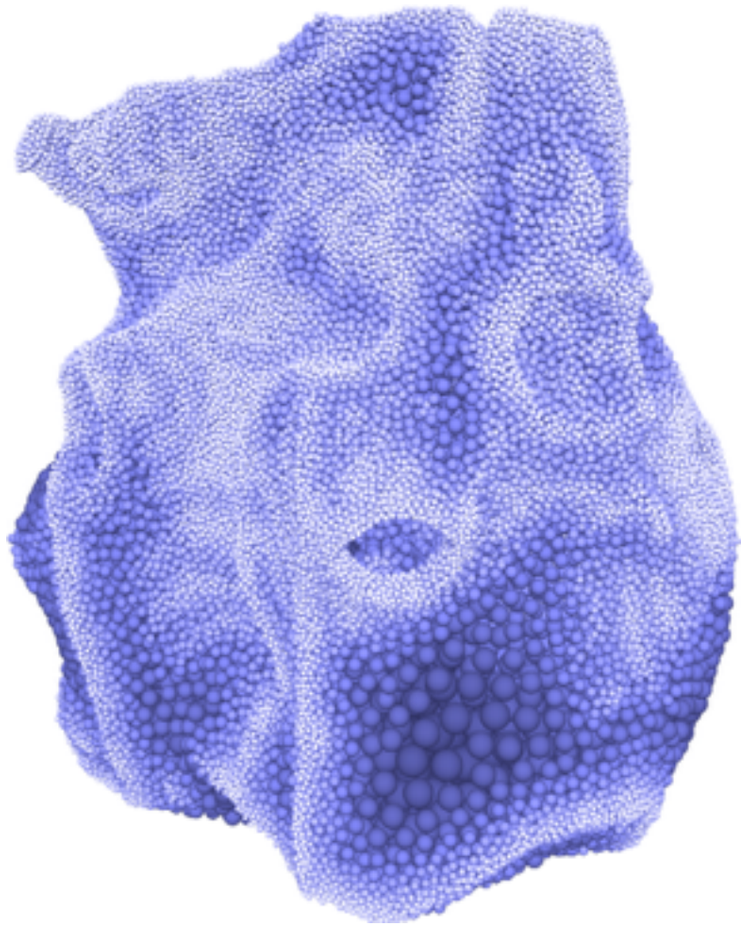
Particle-based Sampling and Meshing of Multimaterial Volumes.
M. Meyer et al., Vis 2008.

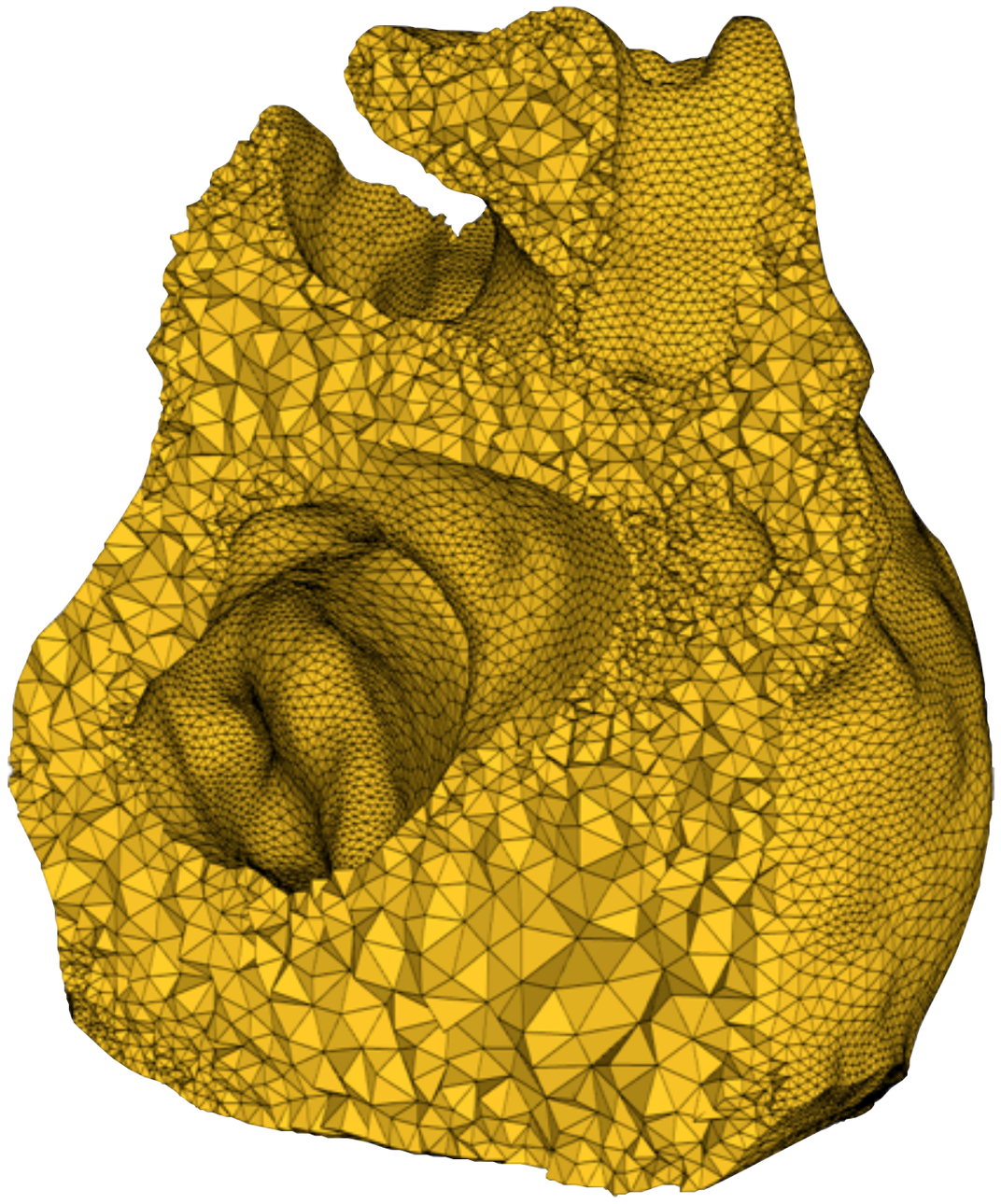


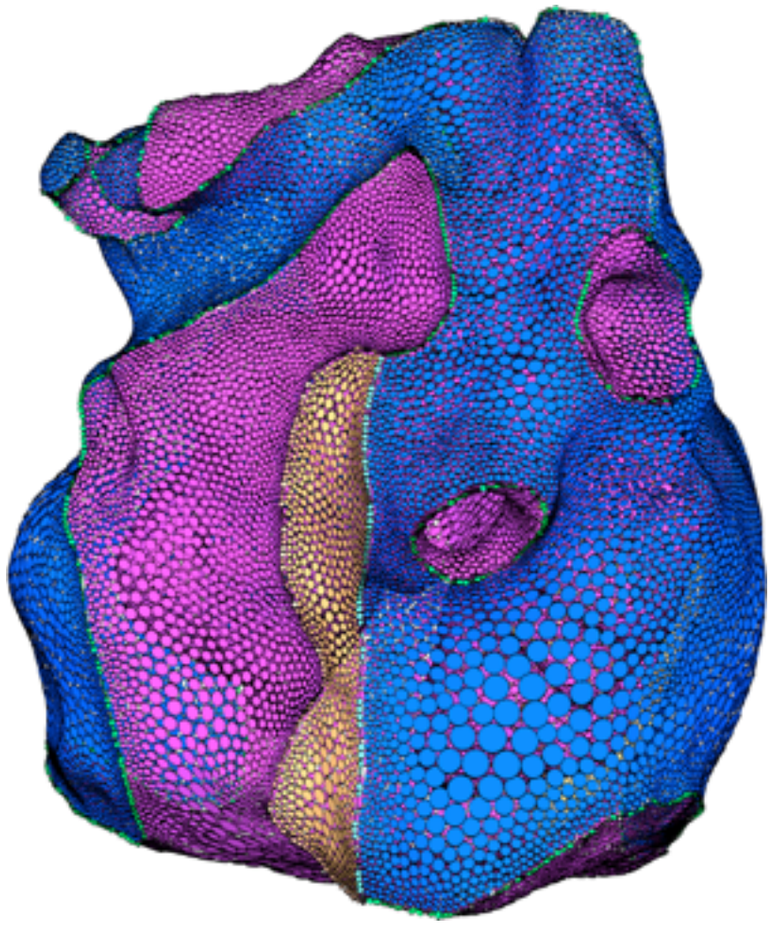


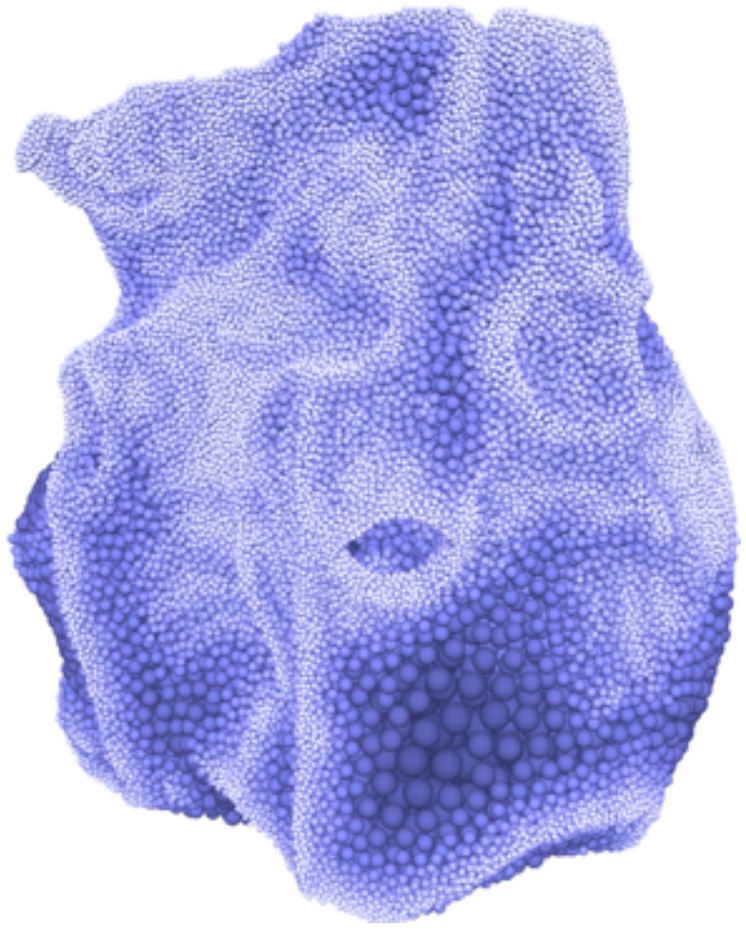


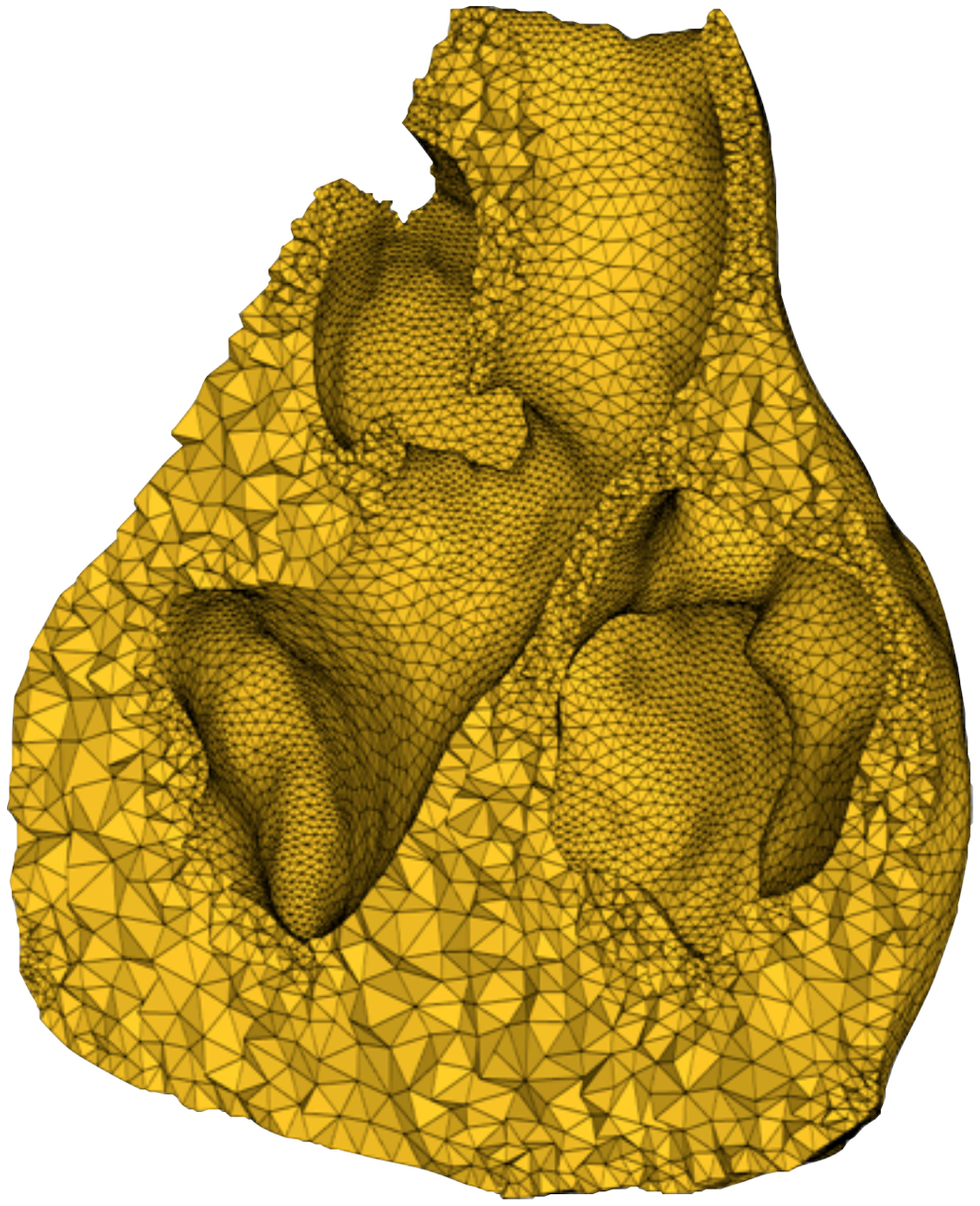


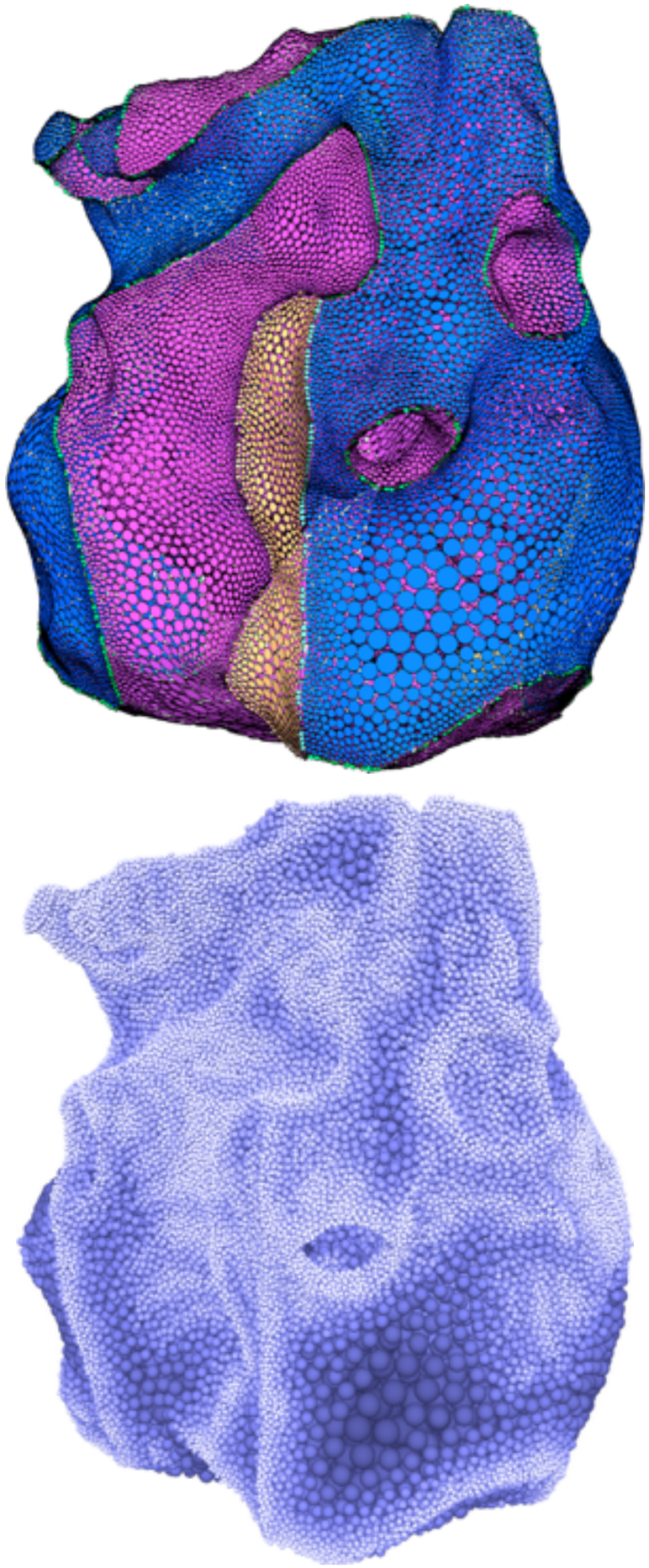


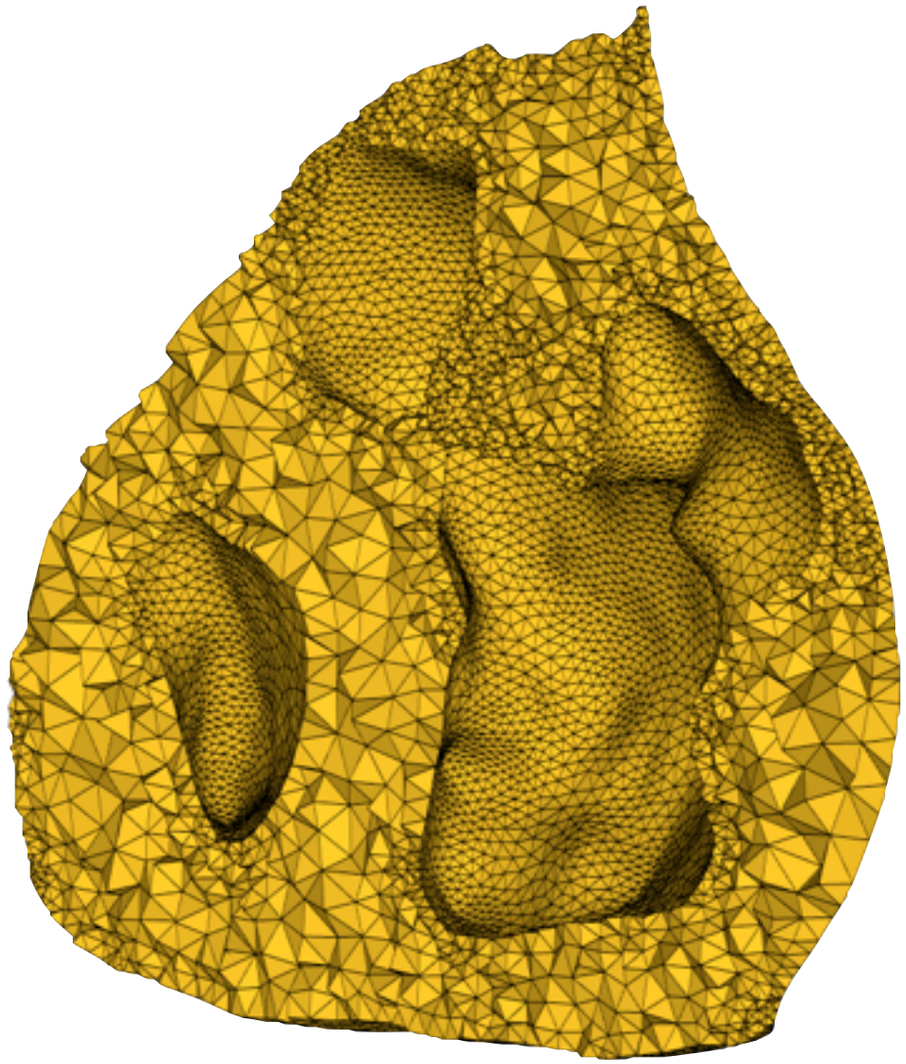












L18: 3D Graphics

REQUIRED READING

16

Background: Computer Graphics

This chapter is designed to be a tutorial on computer graphics techniques, which are the core building blocks for scientific visualization.

First we give an introduction to color models because they are key to a robust visualization. Next is a section describing key elements of the graphics pipeline. This section describes the transformations needed to convert a 3D triangle mesh to 2D pixels on the screen, resulting in the desired orientation and giving a 3D appearance. To give the mesh more realism, we apply an illumination model. Three techniques are described: local illumination, which can be computed in real time; global illumination, which generally is not computed in real time but results in more realistic rendering than local methods; and nonphotorealistic rendering (illumination), which simulates methods such as cartoon or hand-drawn renderings. Texture mapping is introduced next as a method to create complexity in an image without the expense of geometric complexity. Under-