CSE 5544: Introduction to Data Visualization

Raghu Machiraju
machiraju.1@osu.edu

[xkcd]

Many Many Thanks to

- Aaron Knoll, University of Utah
- Alex Lex, University of Utah
- Torsten Moeller, University of Vienna

Overview

- Basic strategies
- Function plots and height fields
- Isolines
- Color coding
- Volume visualization (overview)
- Classification
- Segmentation

Basic Strategies

- 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!**

Basic Strategies

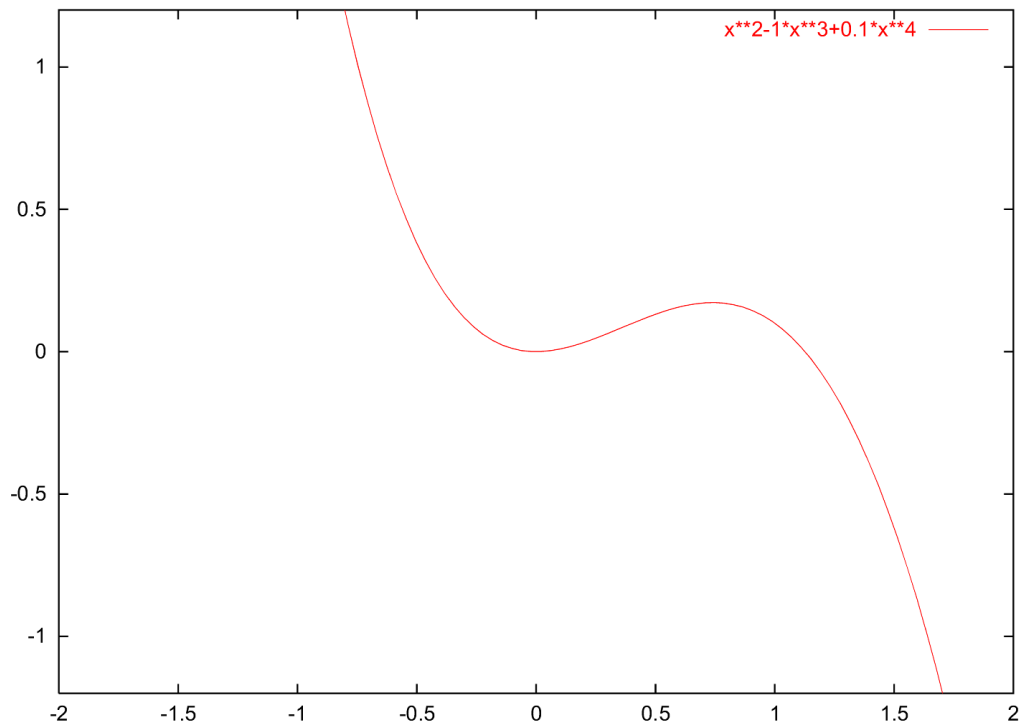
- Mapping to geometry
  - Function plots
  - Height fields
  - Isolines and isosurfaces
- Color coding
- Specific techniques for 3D data
  - Indirect volume visualization
  - Direct volume visualization
  - Slicing

Function Plots and Height Fields

- Function plot for a 1D scalar field

$$\{(s, f(s)) | s \in R\}$$

  - Points
  - 1D manifold: line
  - Error bars possible



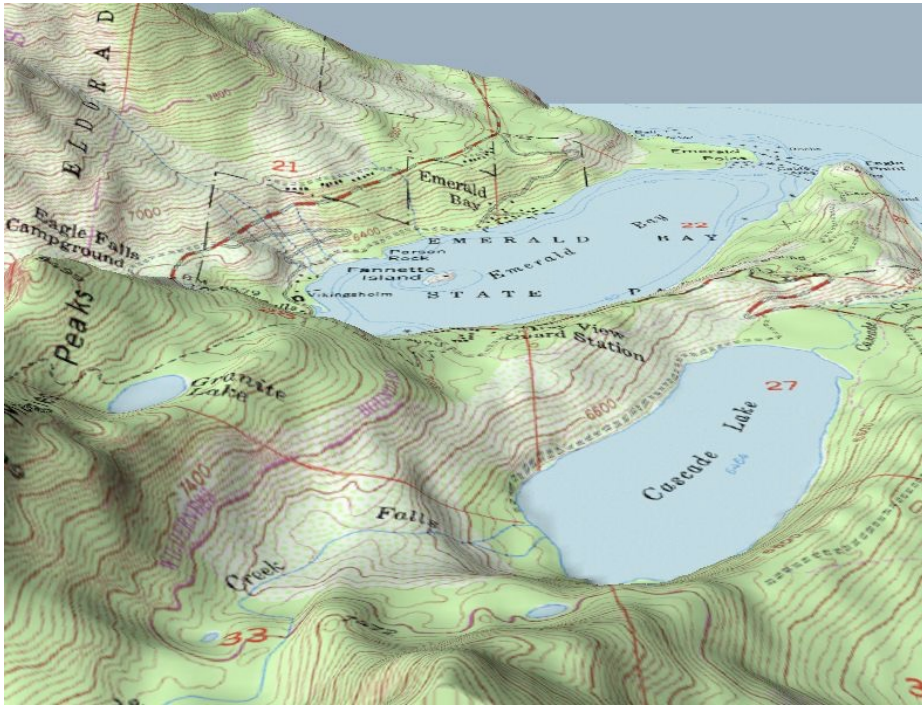
Gnuplot example

Function Plots and Height Fields

- Function plot for a 2D scalar field

$$\{(s, t, f(s, t)) | (s, t) \in R^2\}$$

  - Points
  - 2D manifold: surface
- Surface representations
  - Wireframe
  - Hidden lines
  - Shaded surface



Isolines/Isocontours

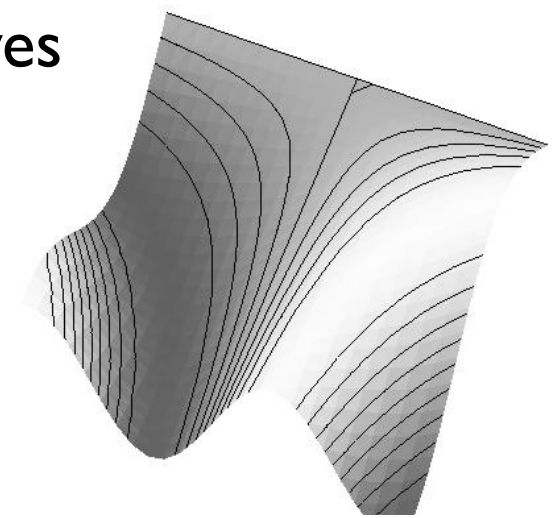
- Visualization of 2D scalar fields
- Given a scalar function and a scalar value

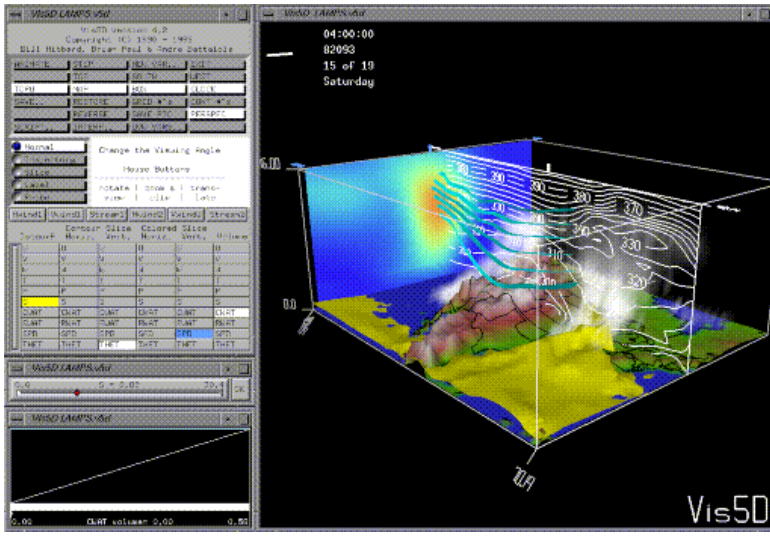
$$f : \Omega \rightarrow R$$
$$c \in R$$

- Isoline consists of points

$$\{(x, y) | f(x, y) = c\}$$

- If *f*() is differentiable and *grad*(*f*) ≠ 0, then isolines are curves
- Contour lines





Level sets

- It's easier to use some mathematical terminology to generalize contours.

- A **level set** of a function $f : R^n \to R$ is the set of points **x,**

$$L_c(f) = \{\mathbf{x} \mid f(\mathbf{x}) = c\}$$

In R^3, a level set is an isosurface. More generally, a contour.

- *c* also defines the **sublevel set,**

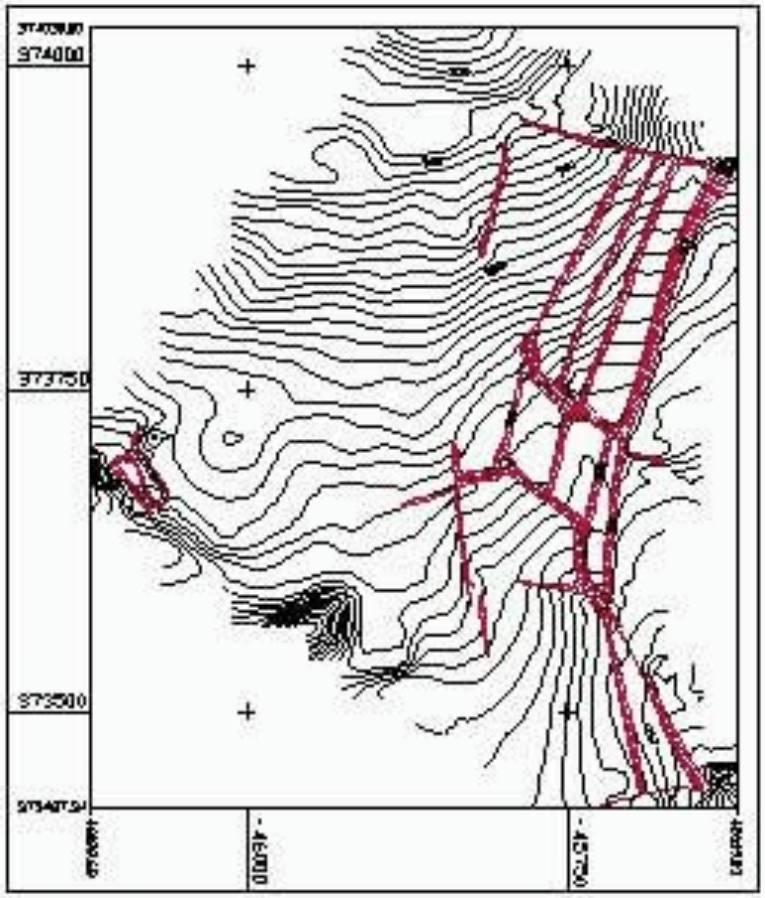
$$L_c^-(f) = \{\mathbf{x} \mid f(\mathbf{x}) \leq c\}$$

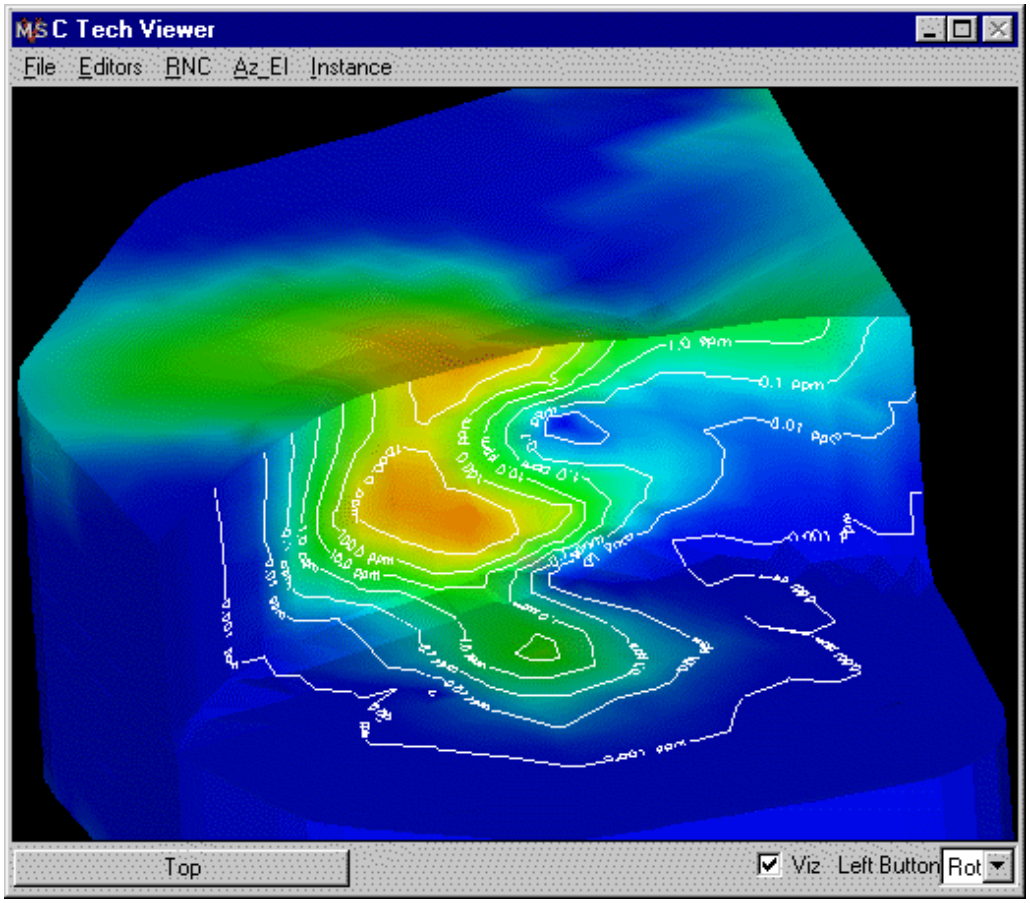
- and the **superlevel set,**

$$L_c^+(f) = \{\mathbf{x} \mid f(\mathbf{x}) \geq c\}$$

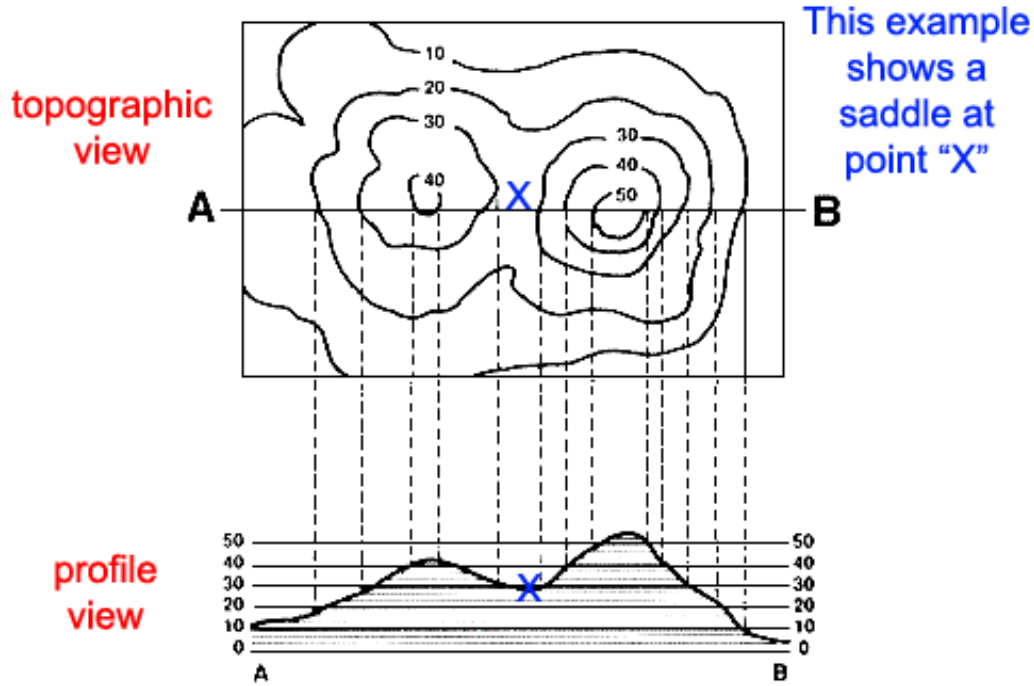
both bounded manifolds in R^n.

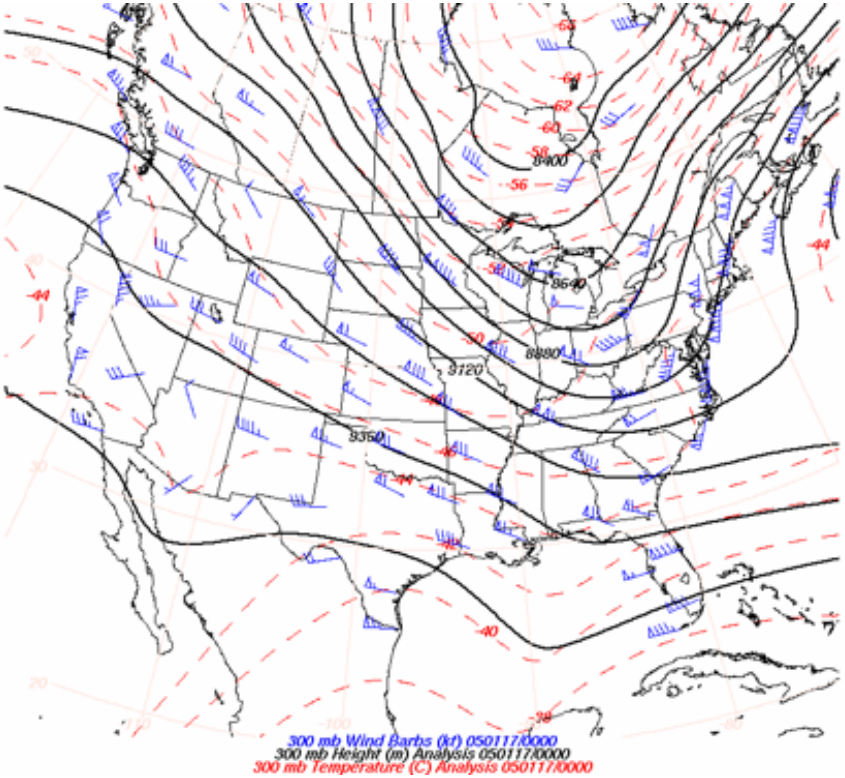
Isolines/Isocontours

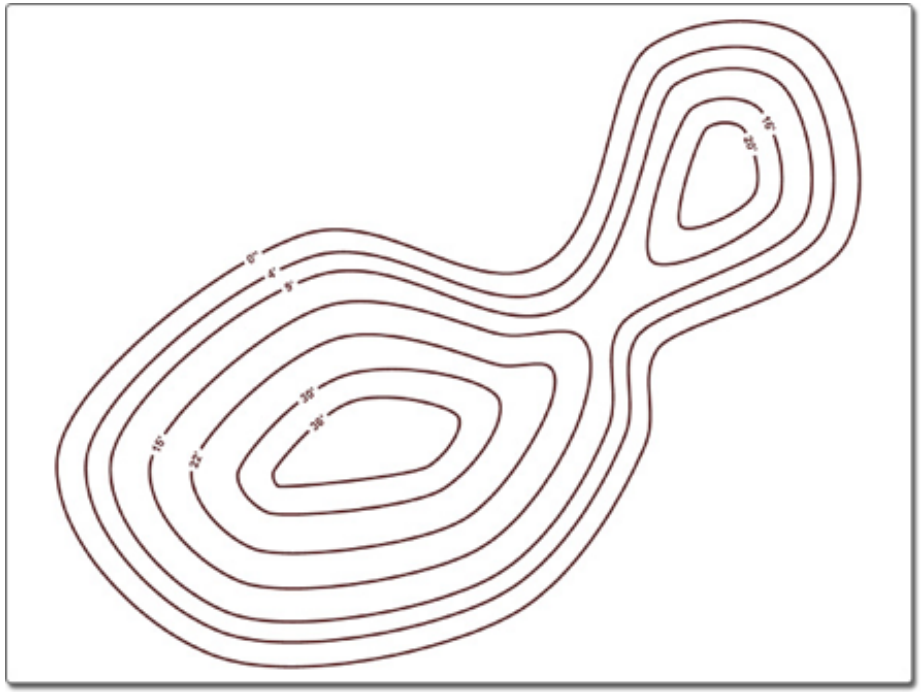


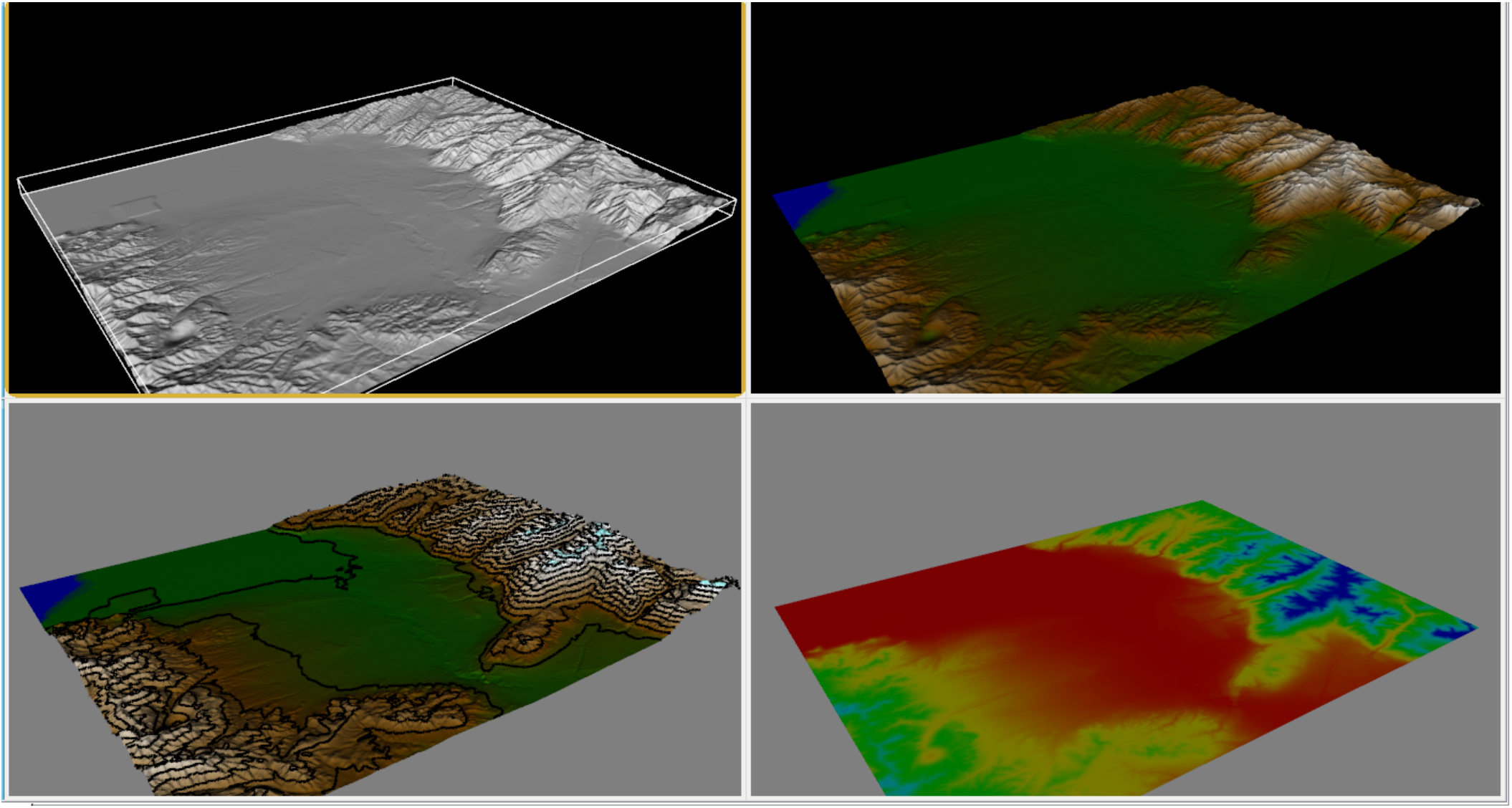


Visualizing Contours




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





Isolines

- Pixel by pixel contouring
- Straightforward approach: scanning all pixels for equivalence with isovalue
- Input
  - $f : (1,\ldots,x_{max}) \times (1,\ldots,y_{max}) \rightarrow \boldsymbol{R}$
  - Isovalues $l_1,\ldots, l_n$ and isocolors $c_1,\ldots,c_n$

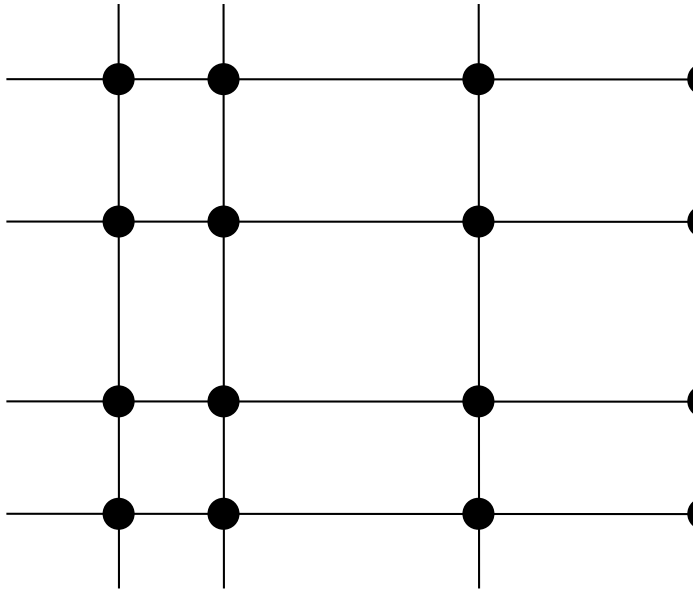
Isolines

- Simple Algorithm

```
for all (x,y) ∈ (1,...,x_max) x (1,...,y_max) do
  for all k∈ { 1,...,n } do
    if |f(x,y)-l_k| < ε then
      draw(x,y,c_k)
```

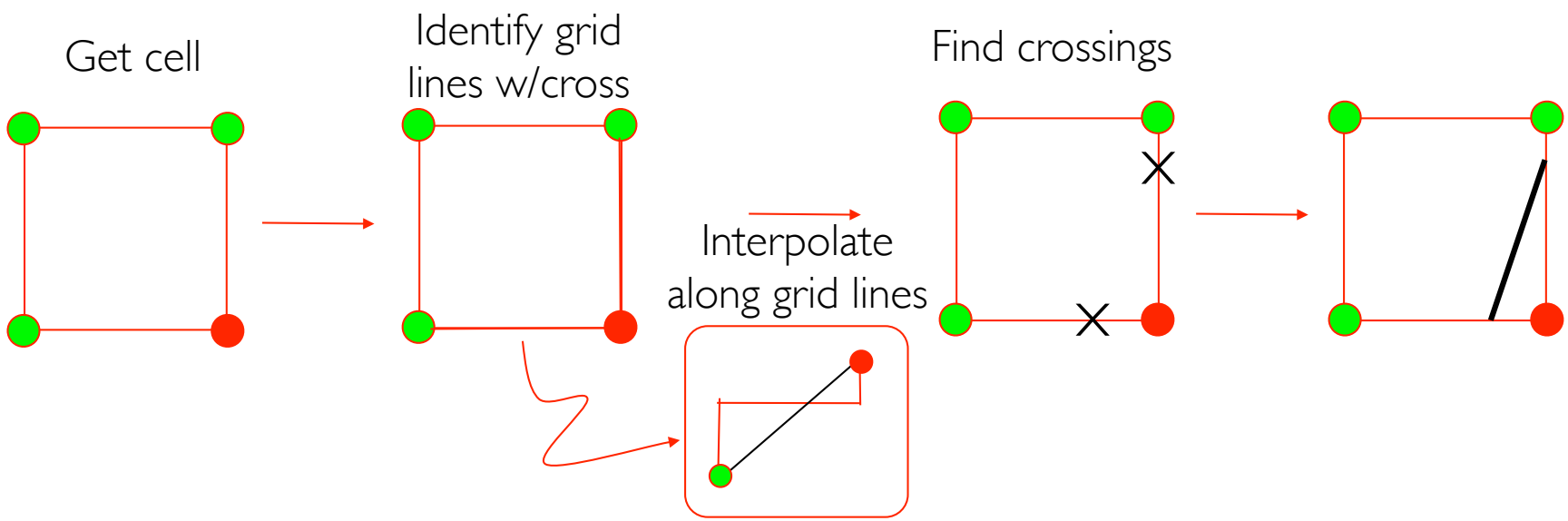
- Problem: Isoline can be missed if the gradient of *f()* is too large (despite range ε)

Marching Squares

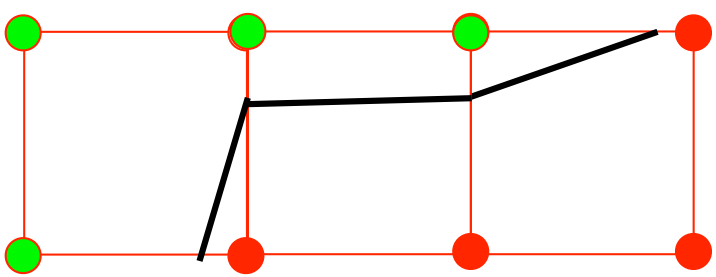


- Marching squares
  - Representation of the scalar function on a rectilinear grid
  - Scalar values are given at each vertex $f \leftrightarrow f_{ij}$
  - Take into account the interpolation within cells
  - Isolines cannot be missed
  - Consider cells independently of each other

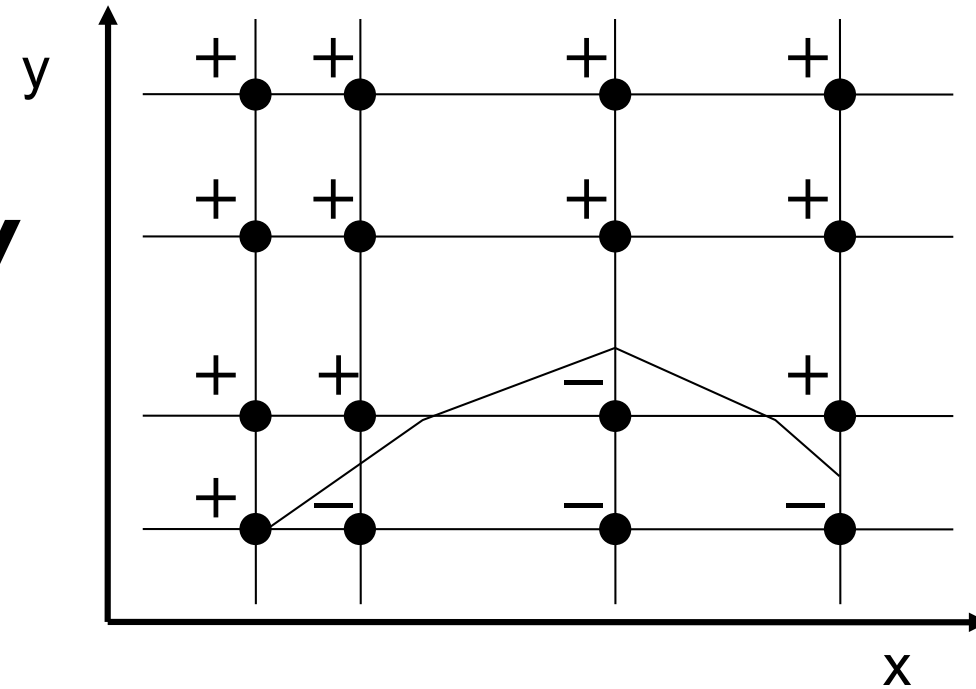
Marching Squares



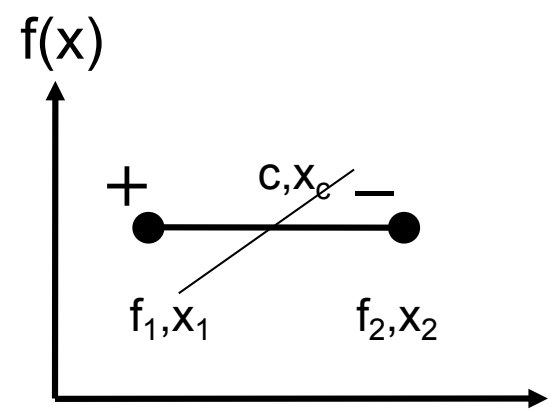
Primitives naturally chain together



Formally

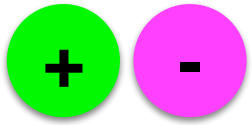


- Which cells will be intersected ?
  - Initially mark all vertices by + or – , depending on the conditions $f_{ij} \geq c$, $f_{ij} < c$
- No isoline passes through cells (=rectangles) which have the same sign at all four vertices
  - So we only have to determine the edges with different signs
  - And find intersection by linear interpolation

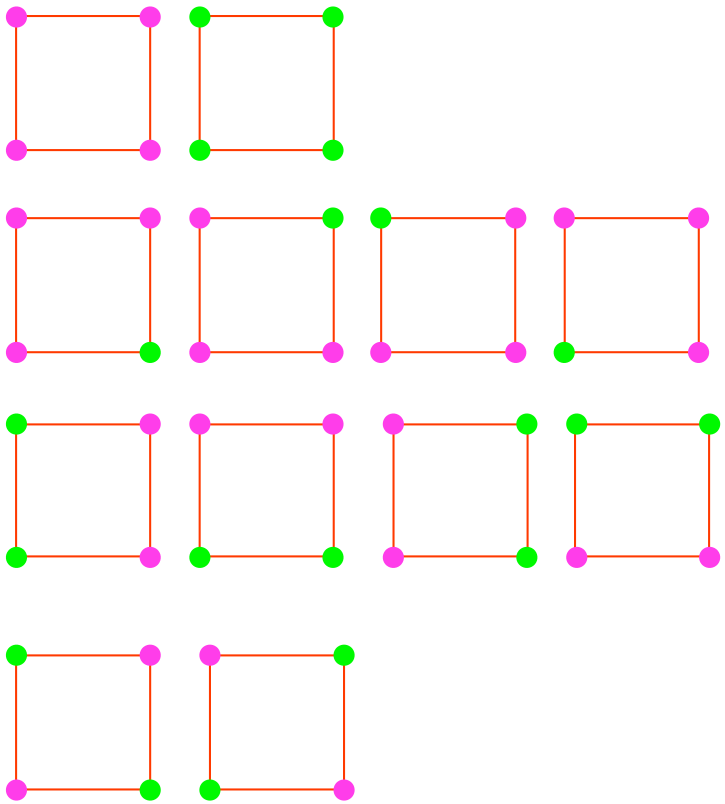


$$x_c = [(f_2 - c)x_1 + (c - f_1)x_2] / (f_2 - f_1)$$

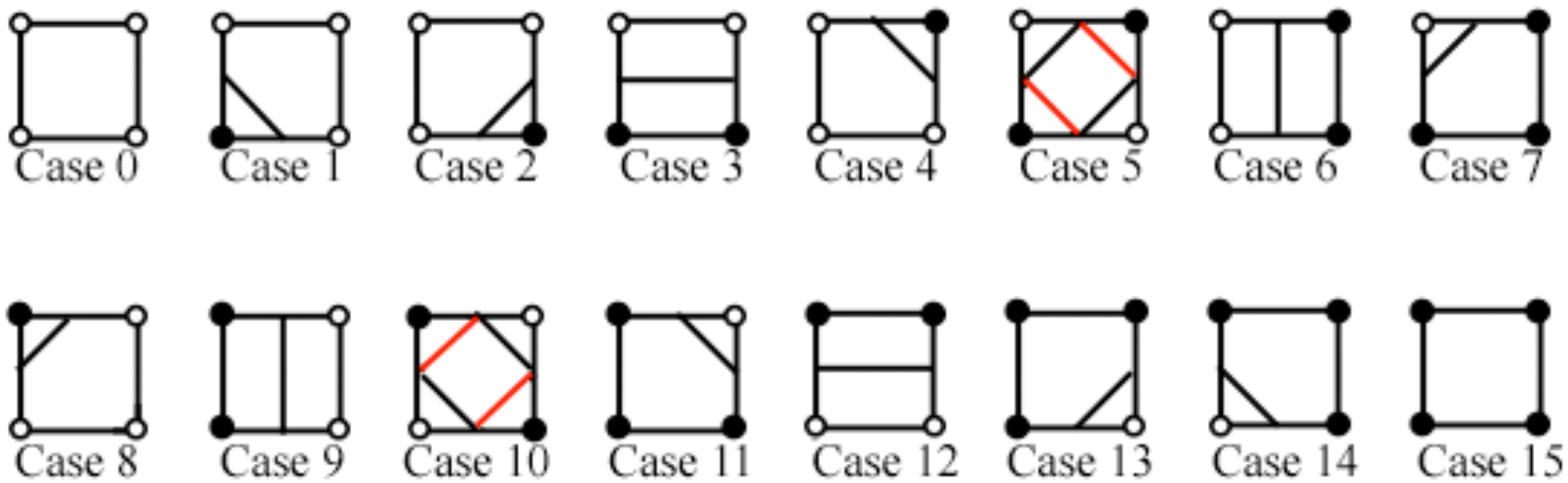
Cases



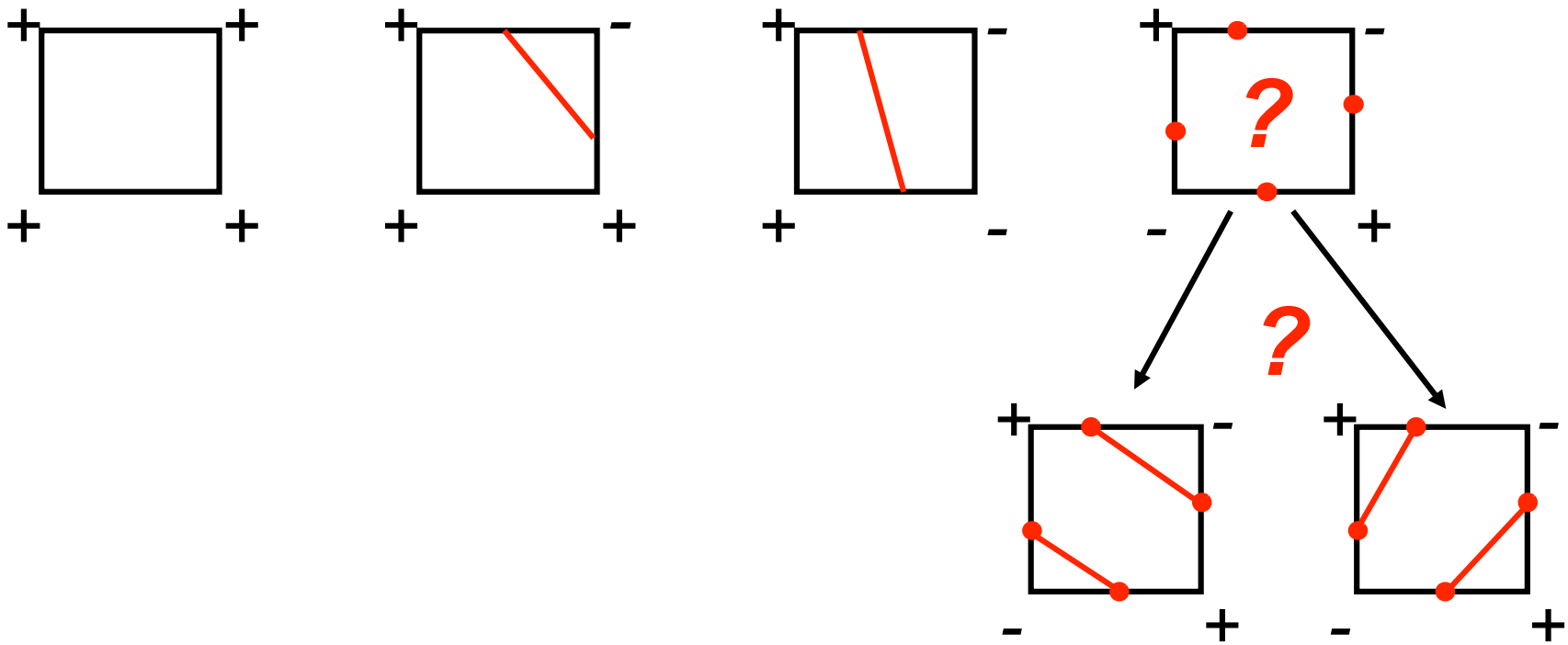
Case	Polarity	Rotation	Total
No Crossings	x2		2
Singlet	x2	x4	8
Double adjacent	x2	x2 (4)	4
Double Opposite	x2	x1 (2)	2
			$16 = 2^4$

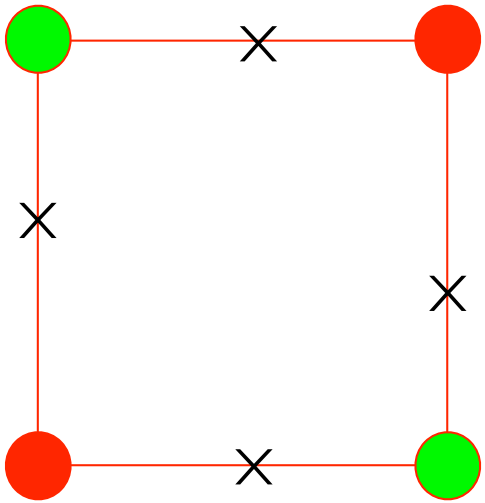


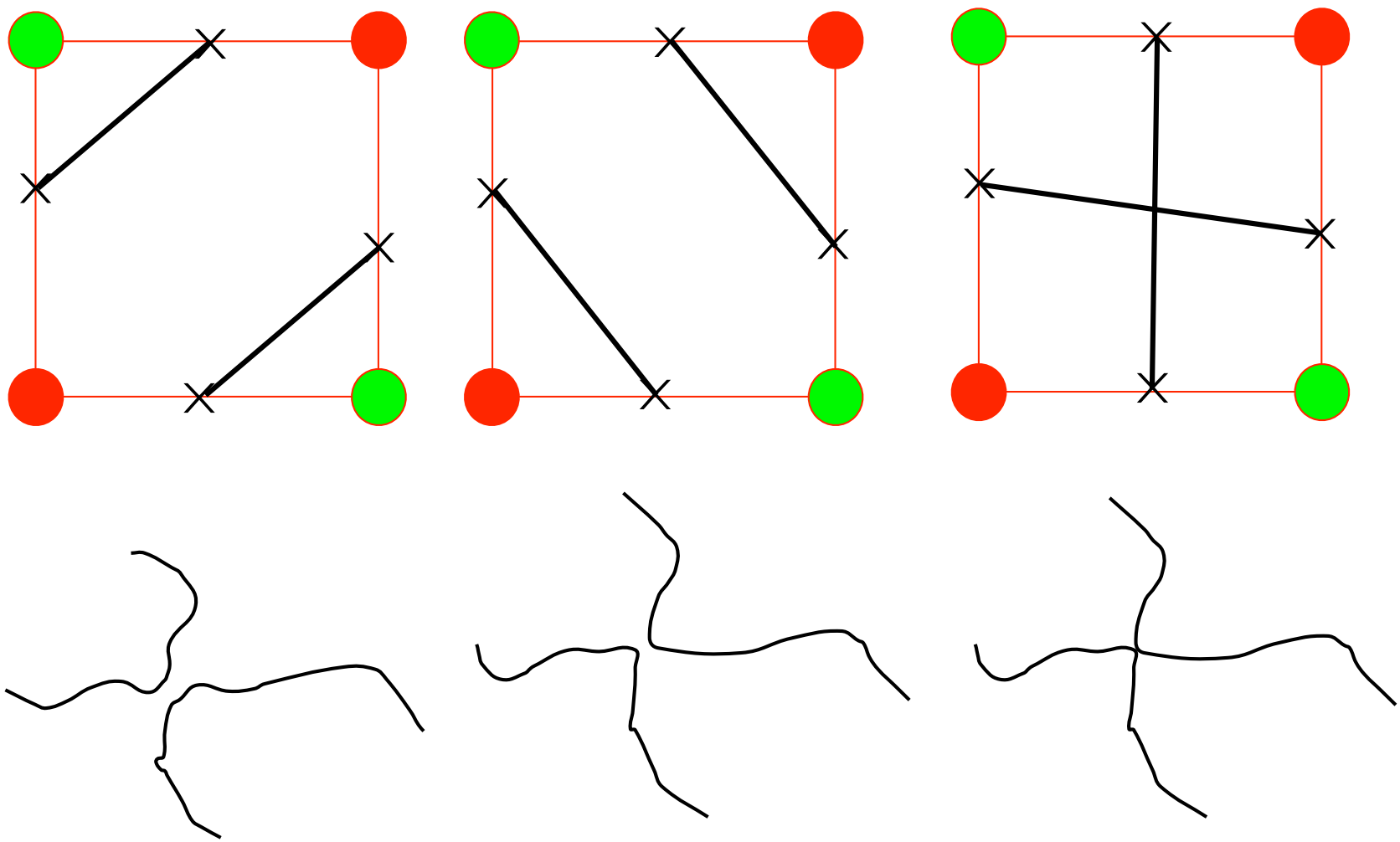
(x2 for polarity)


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

Is All Good ?





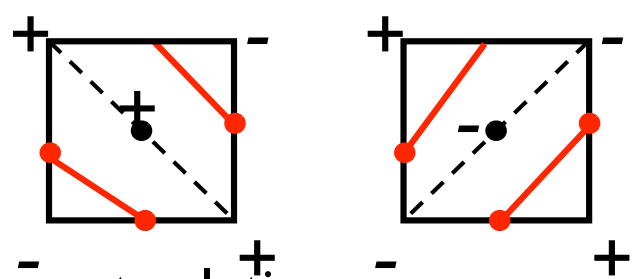


Ambiguity Resolution

- We can distinguish the cases by a decider
- Mid point decider
  - Interpolate the function value in the center

$$f_{\text{center}} = \frac{1}{4}(f_{i,j} + f_{i+1,j} + f_{i,j+1} + f_{i+1,j+1})$$

  - If $f_{center} < c$ we chose the right case, otherwise we chose the left case



  - Not always correct solution

Better Solution

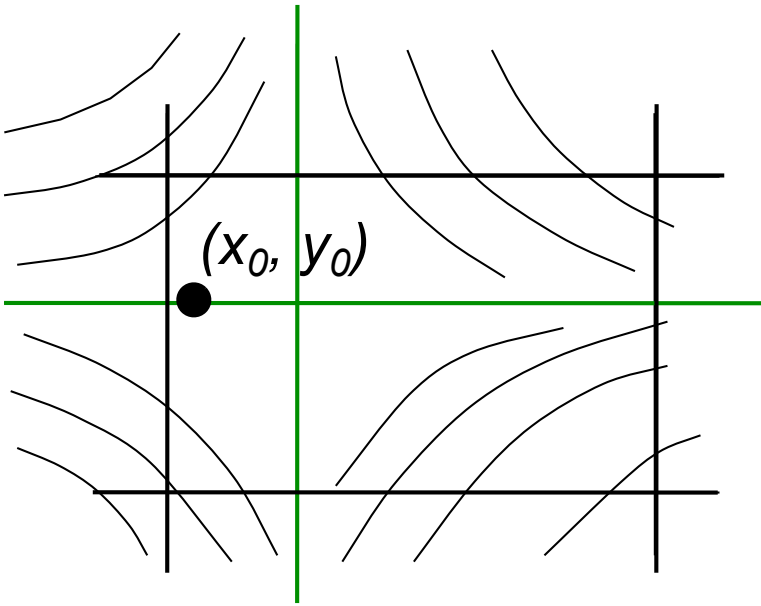
- Interpolate the function bilinearly

$$f(x,y) = f_{i,j}(1-x)(1-y) + f_{i+1,j}x(1-y) + f_{i,j+1}(1-x)y + f_{i+1j+1}xy$$

- Transform *f()* to

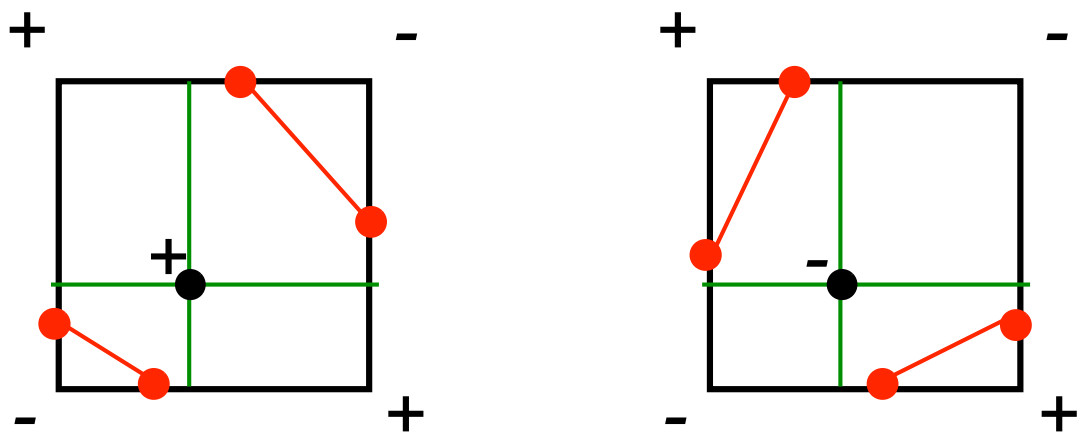
$$f(x,y) = \eta(x - x_0)(y - y_0) + \gamma$$

- γ is the function value at intersection point of the asymptotes



The Choice

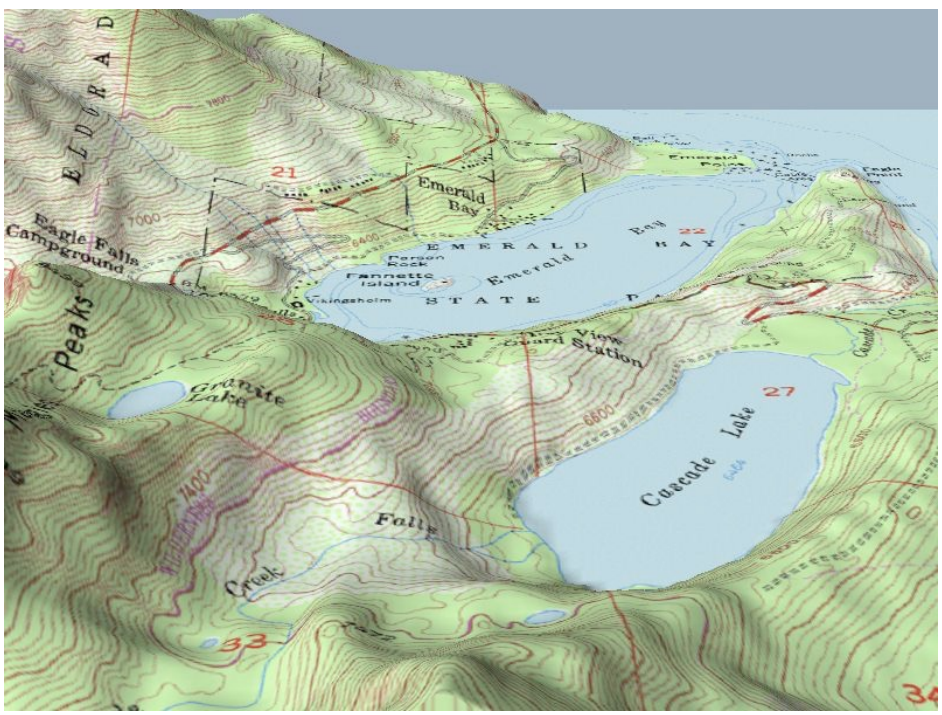
If $\gamma \leq c$ we chose the right case, otherwise we chose the left one

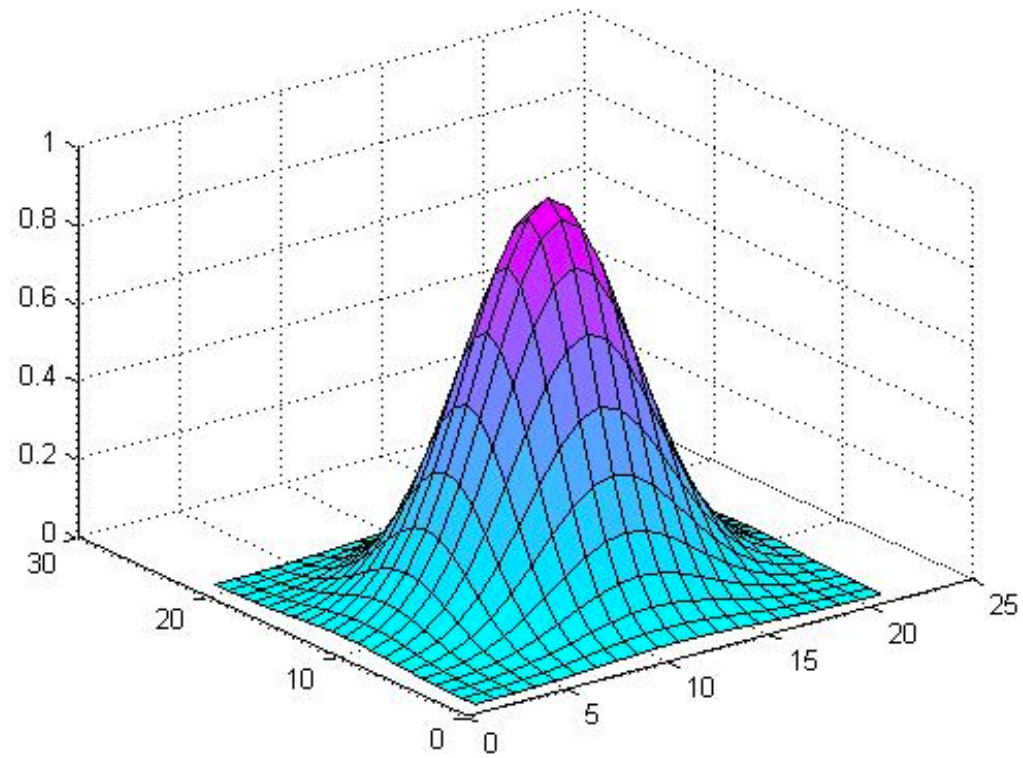


Functions again

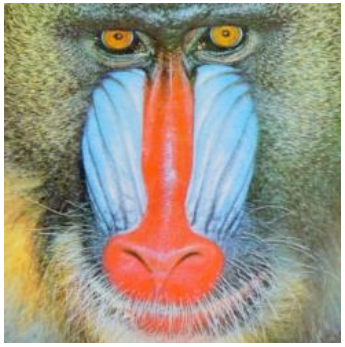
Function plot for a 2D scalar field

$$\{(s, t, f(s,t)) | (s,t) \in R^2\}$$





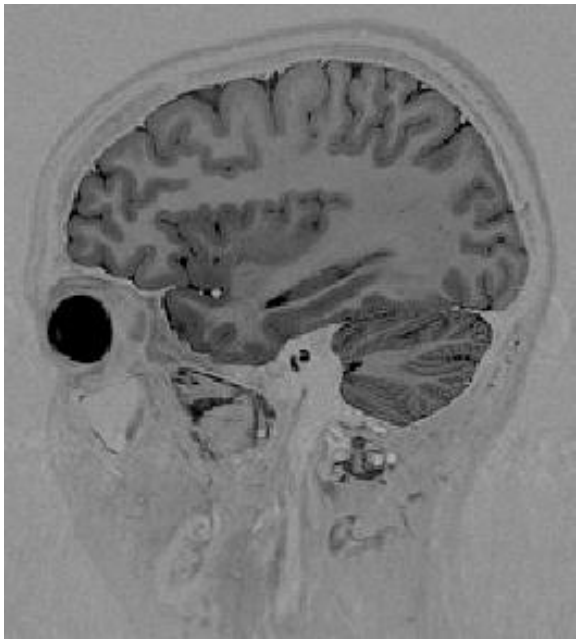
Color Coding



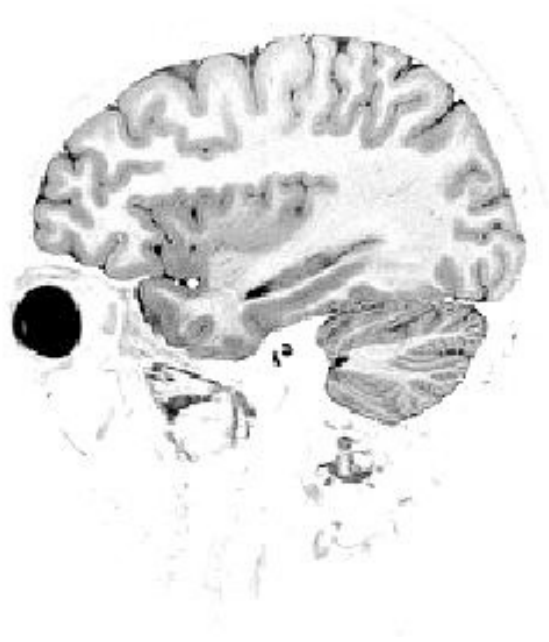
- Easy to apply to 1D and 2D scalar fields
  - Map color to each pixel on 1D or 2D image

Color Coding

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



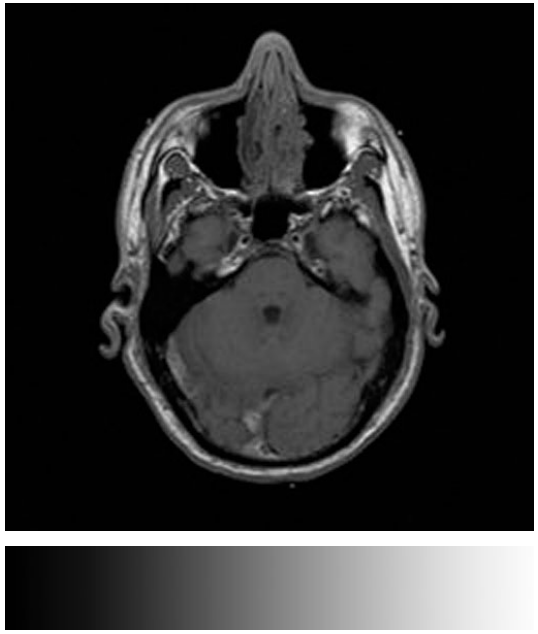
Original



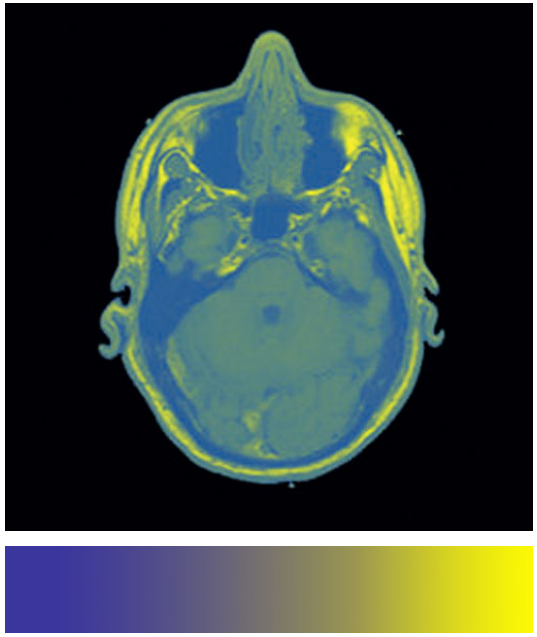
Brain

Tissue

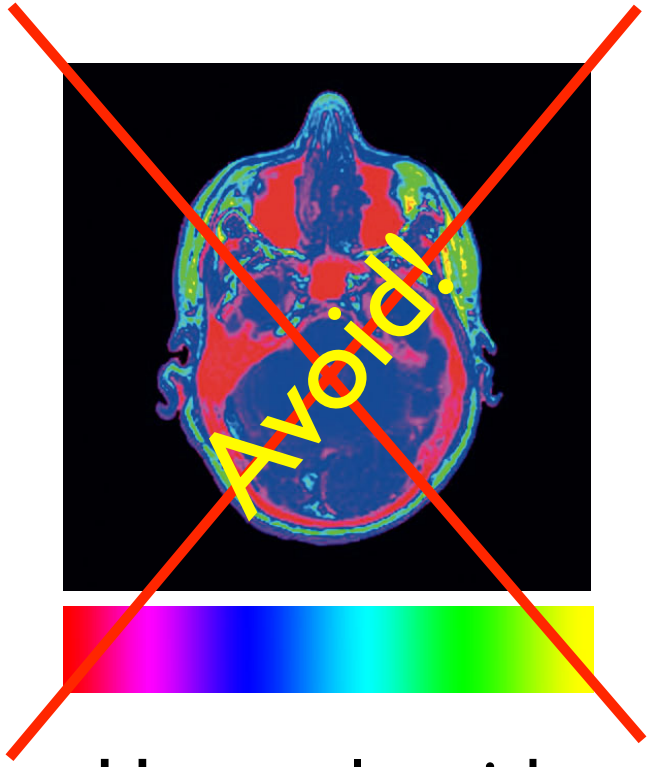
Density Map



Lightness scale



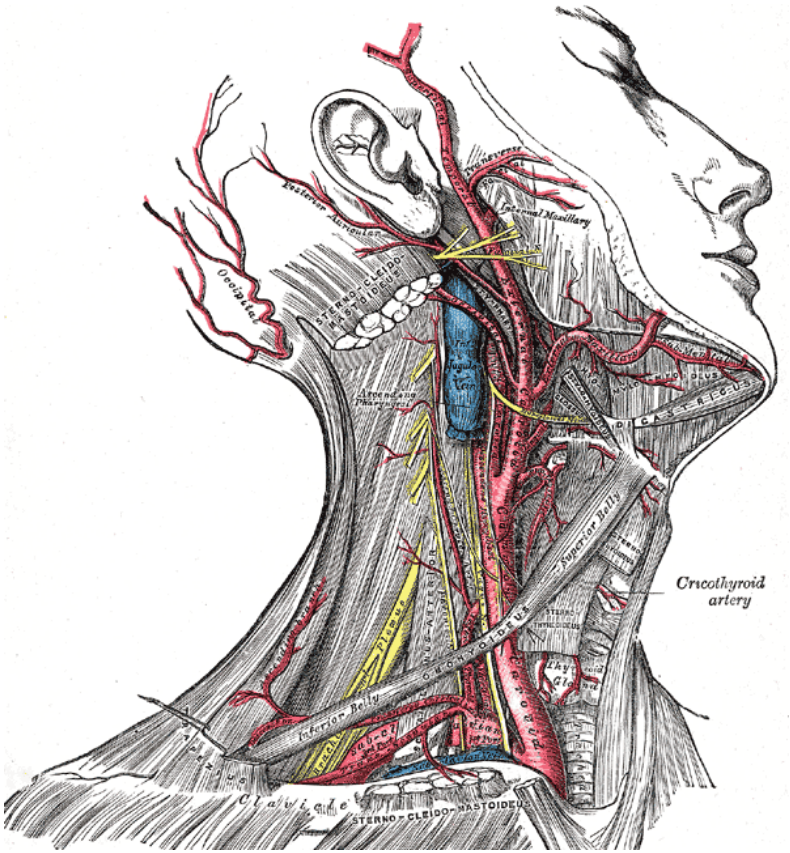
Lightness scale with hue variation



Hue scale with lightness variation

Based on slide from Stone

Illustrative Color



Gray's Anatomy of the Human Body

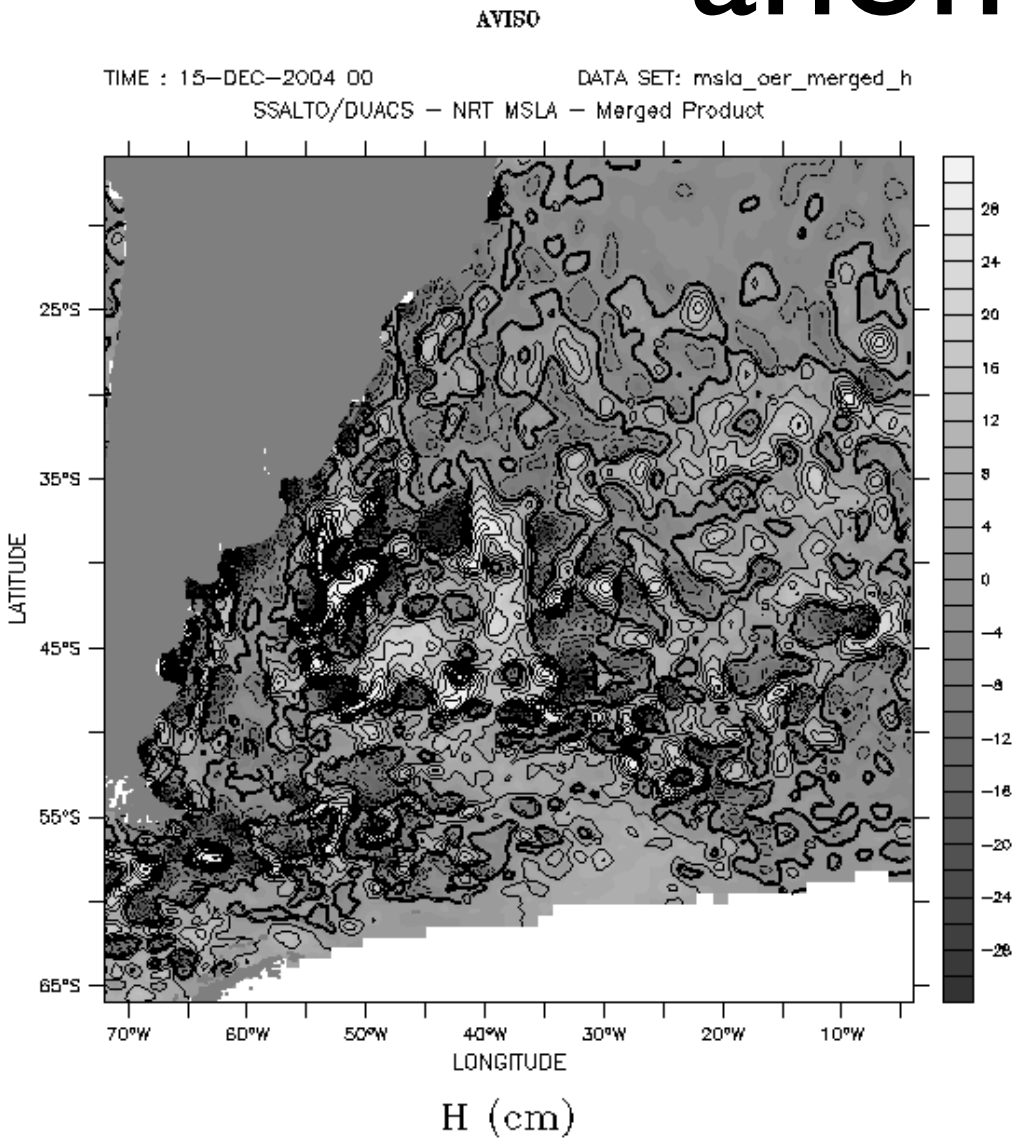
www.bartleby.com/107/illus520.html

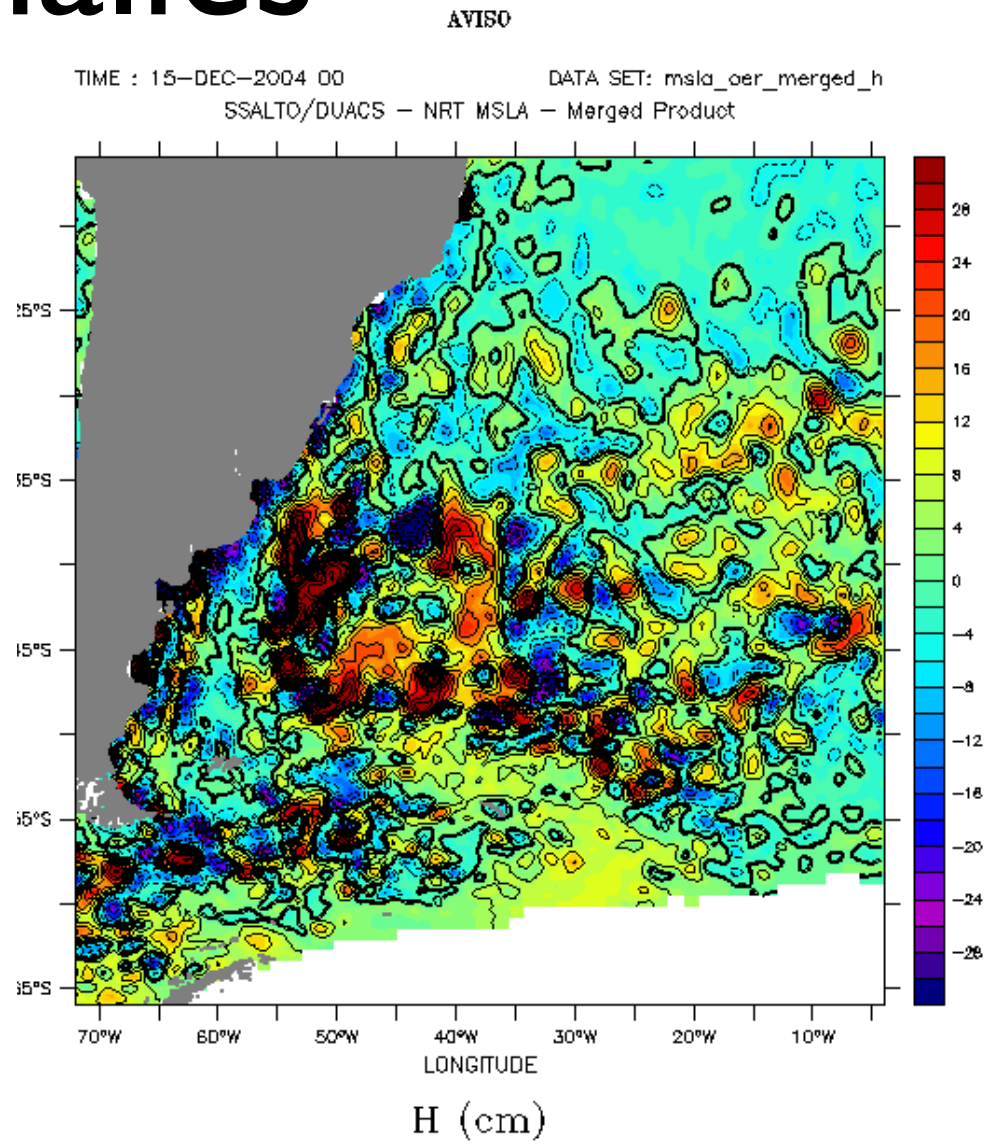
Map of Point Reyes

www.nps.gov

Based on slide from Stone

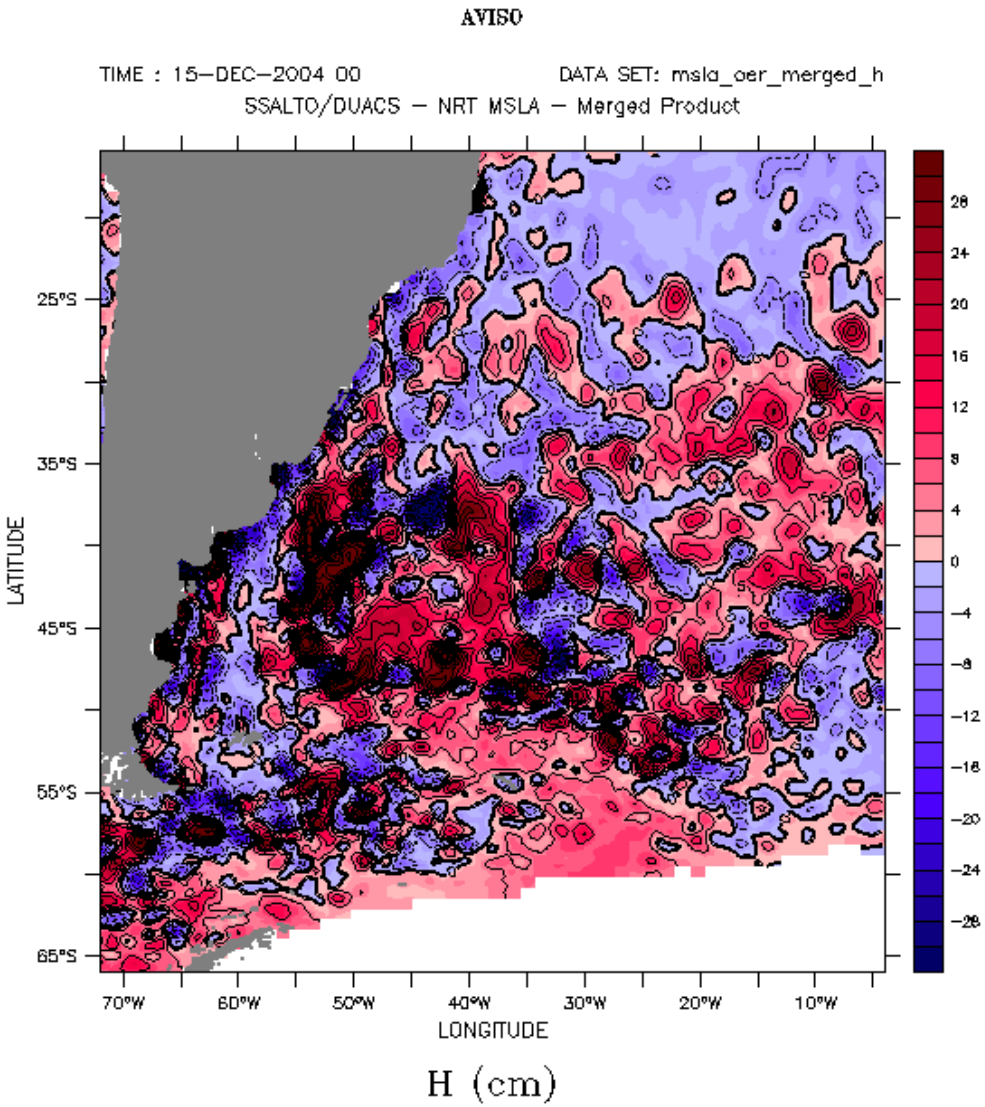
Task: Find high & low anomalies

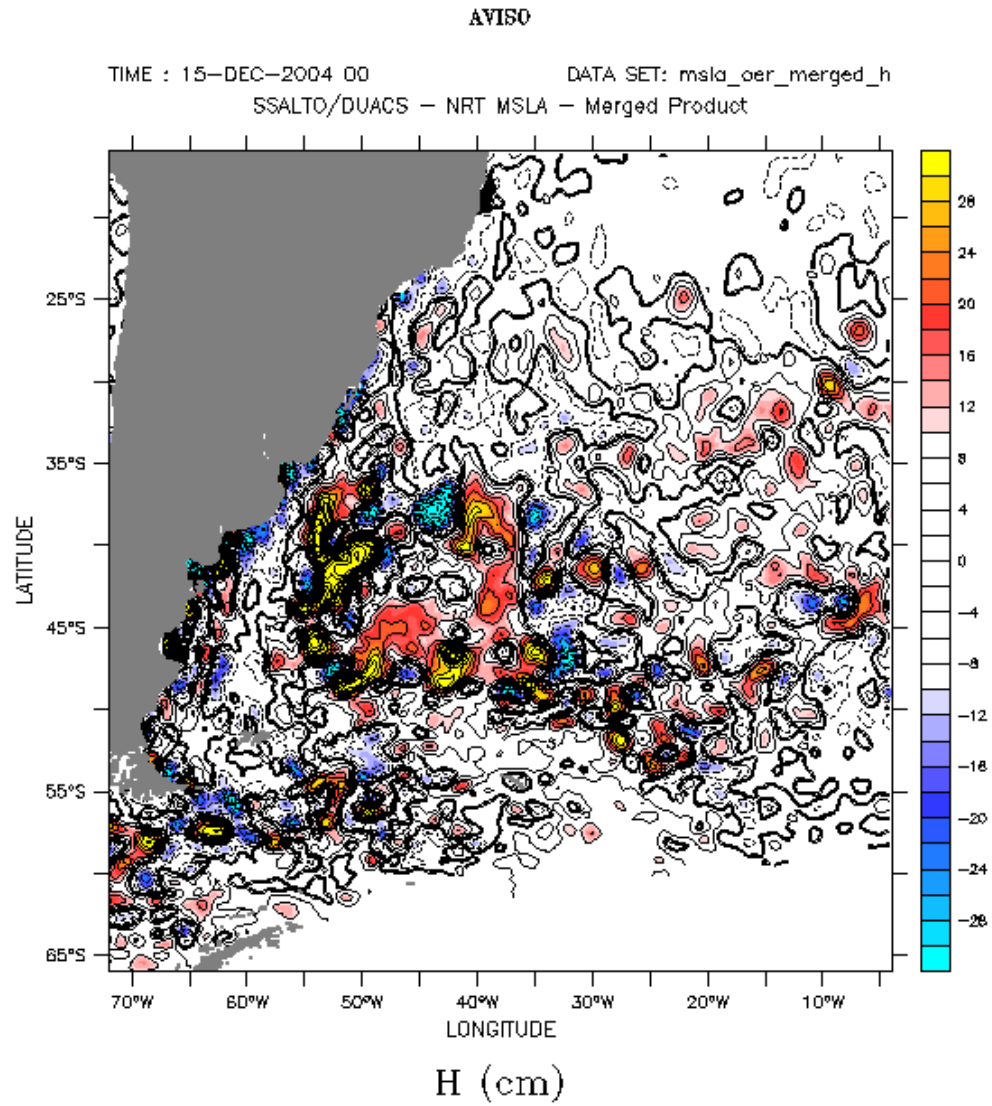




http://www.jason.oceanobs.com/html/donnees/las/2005-03_uk.html

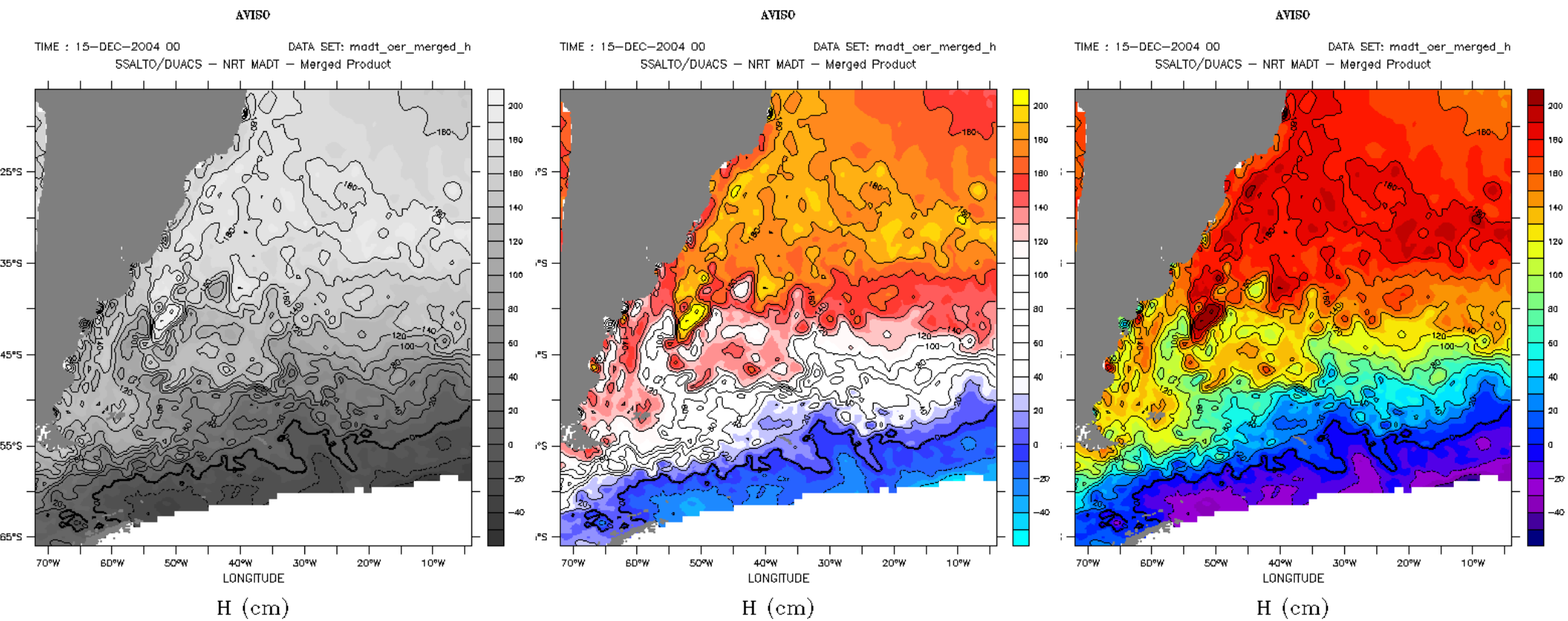
Task: Find high & low anomalies





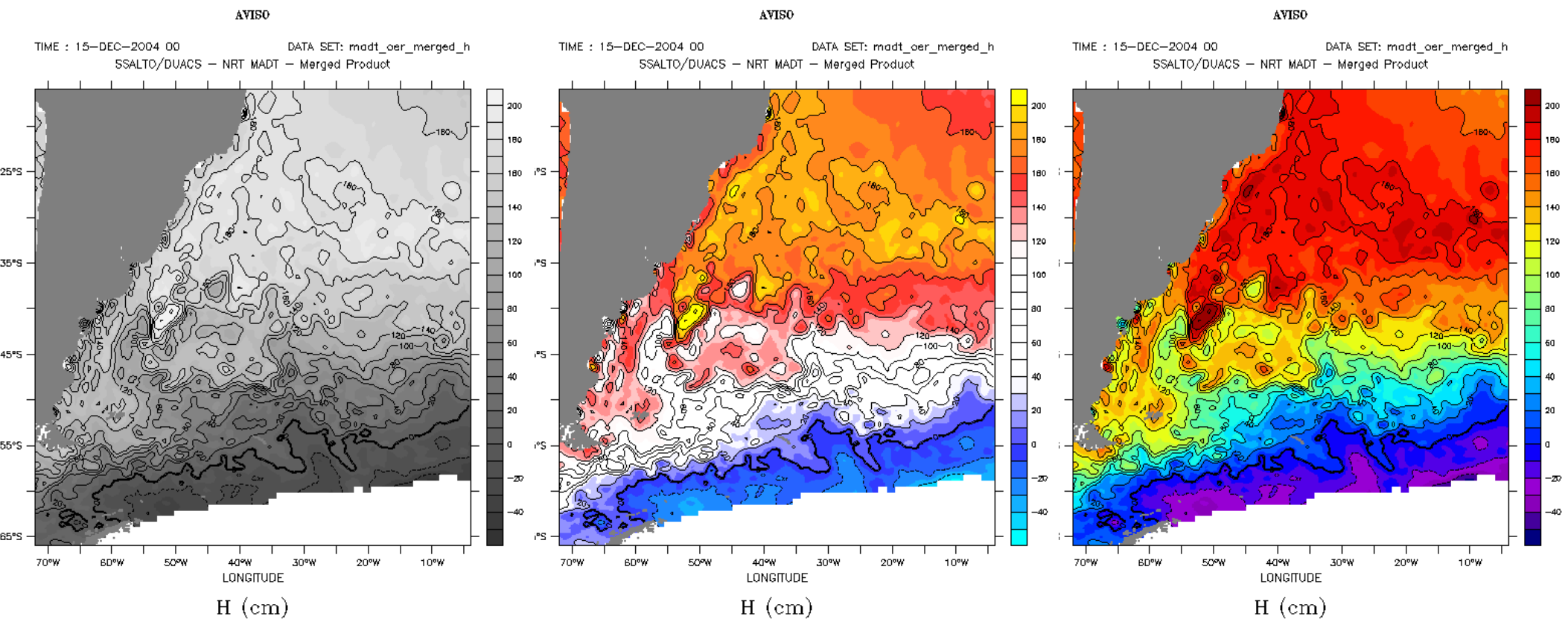
http://www.jason.oceanops.com/html/donnees/las/2005-03_uk.html

Task: Understand relative height



http://www.jason.oceanobs.com/html/donnees/las/2005-03_uk.html

Task: Find height 120

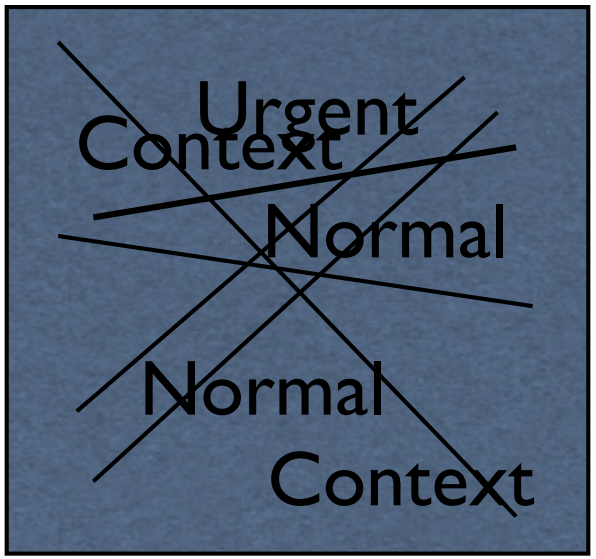


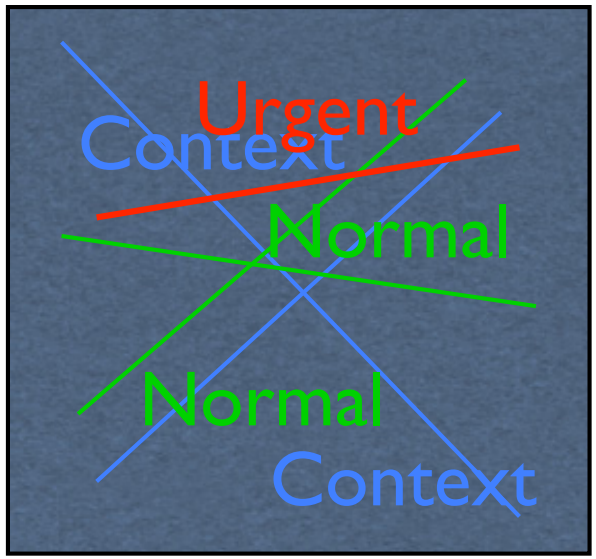
http://www.jason.oceanobs.com/html/donnees/las/2005-03_uk.html

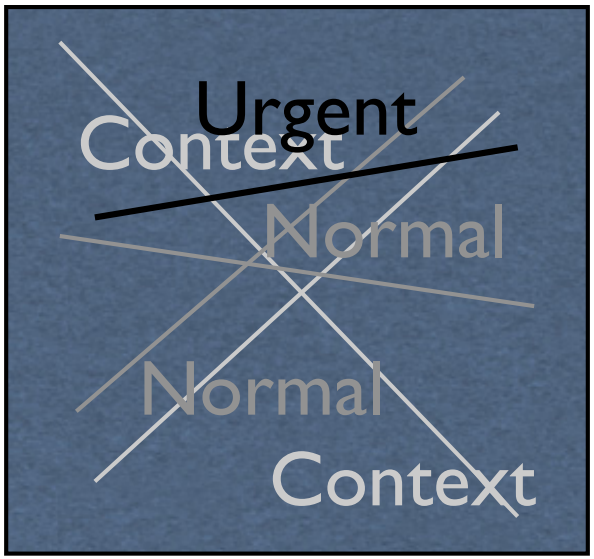
DANGER!

- Inappropriate use of colour can be disasterous to the application

Controls Attention, Clutter



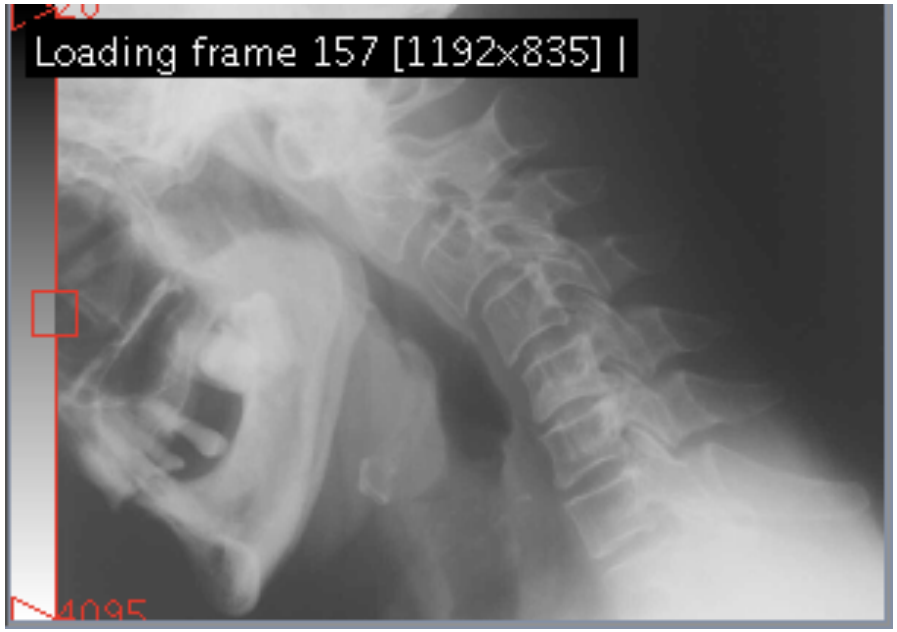


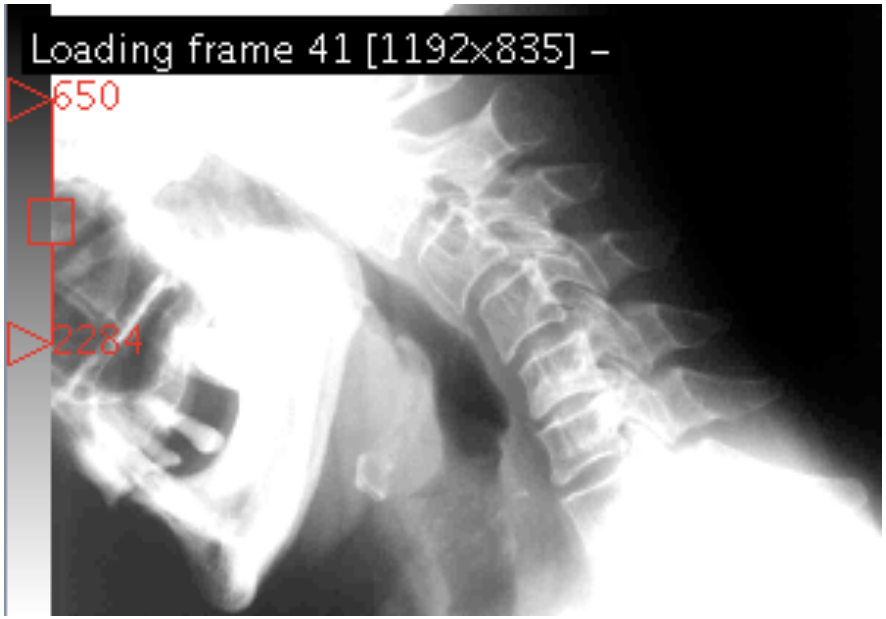


Courtesy of Maureen Stone

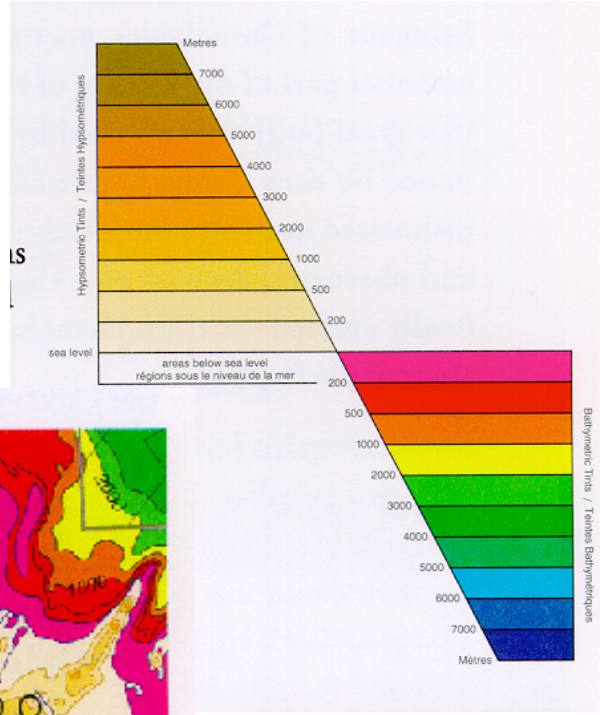
Window & Level

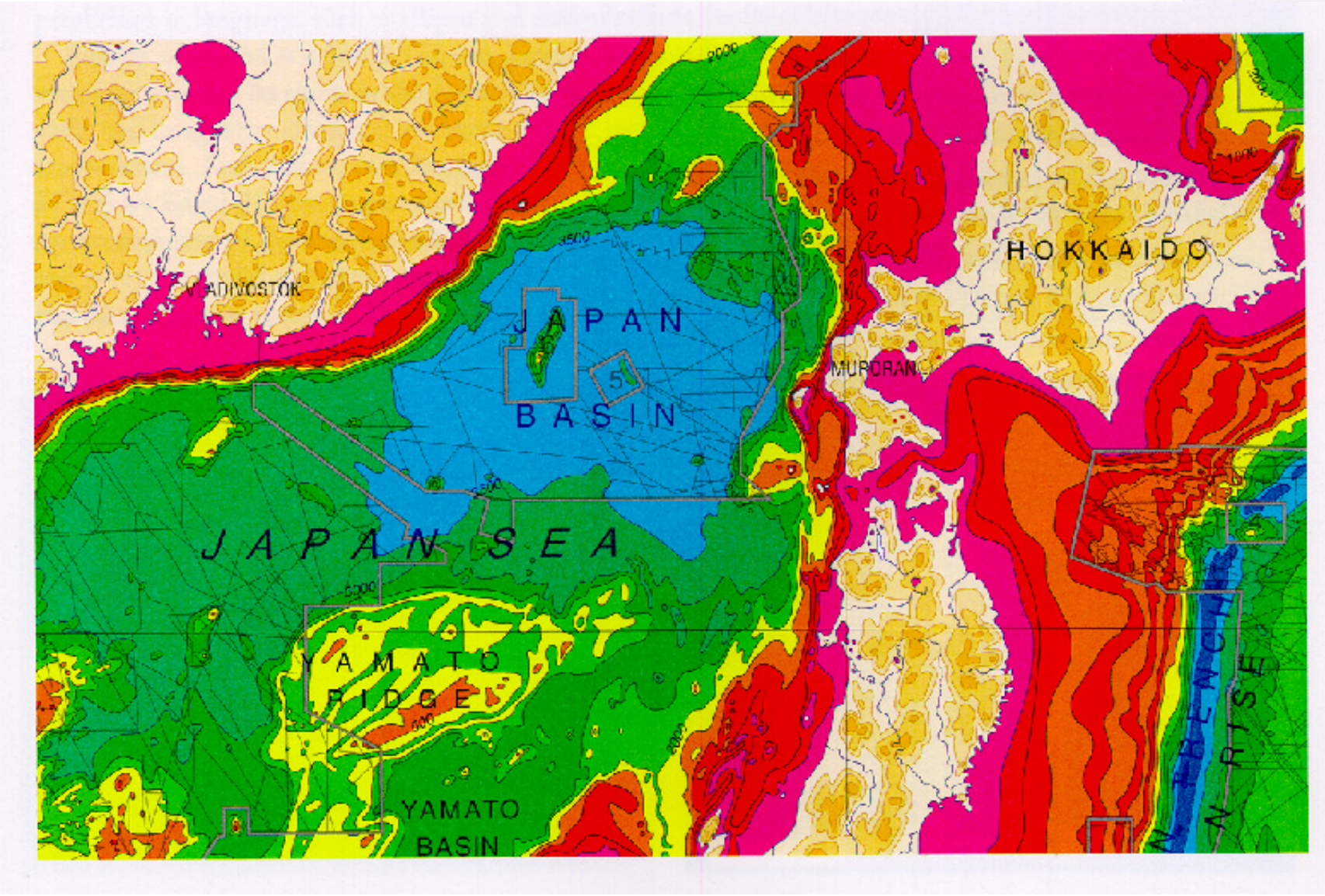
Adjusts contrast in an image



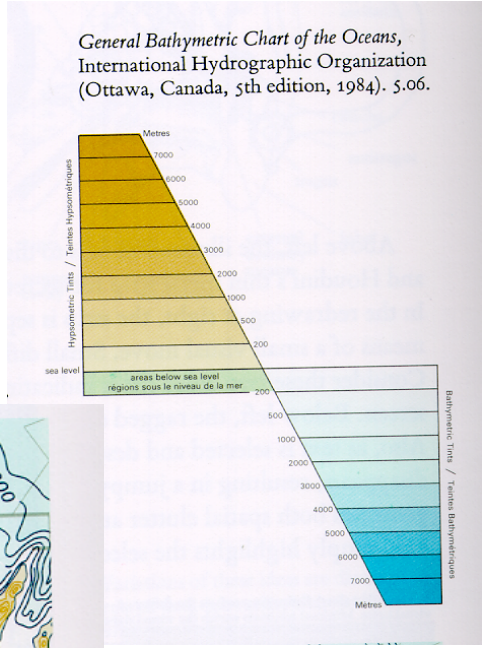


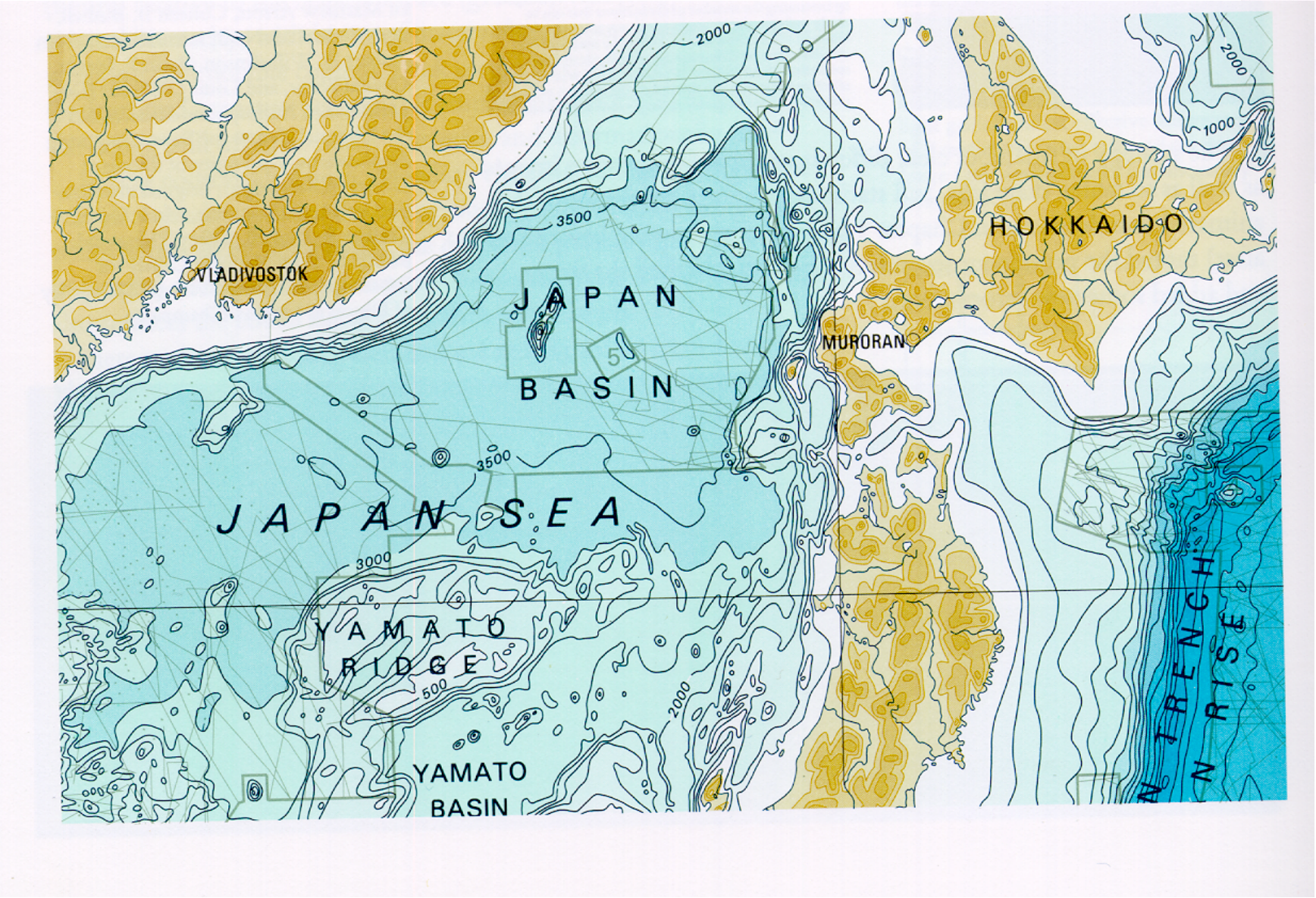
Rainbow ...


Metres
7000
6000
5000
4000
3000
2000
1000
500
200
sea level
areas below sea level
régions sous le niveau de la mer
Hypsometric Tints / Teintes Hypsométriques
200
500
1000
2000
3000
4000
5000
6000
7000
Mètres
Bathymetric Tints / Teintes Bathymétriques

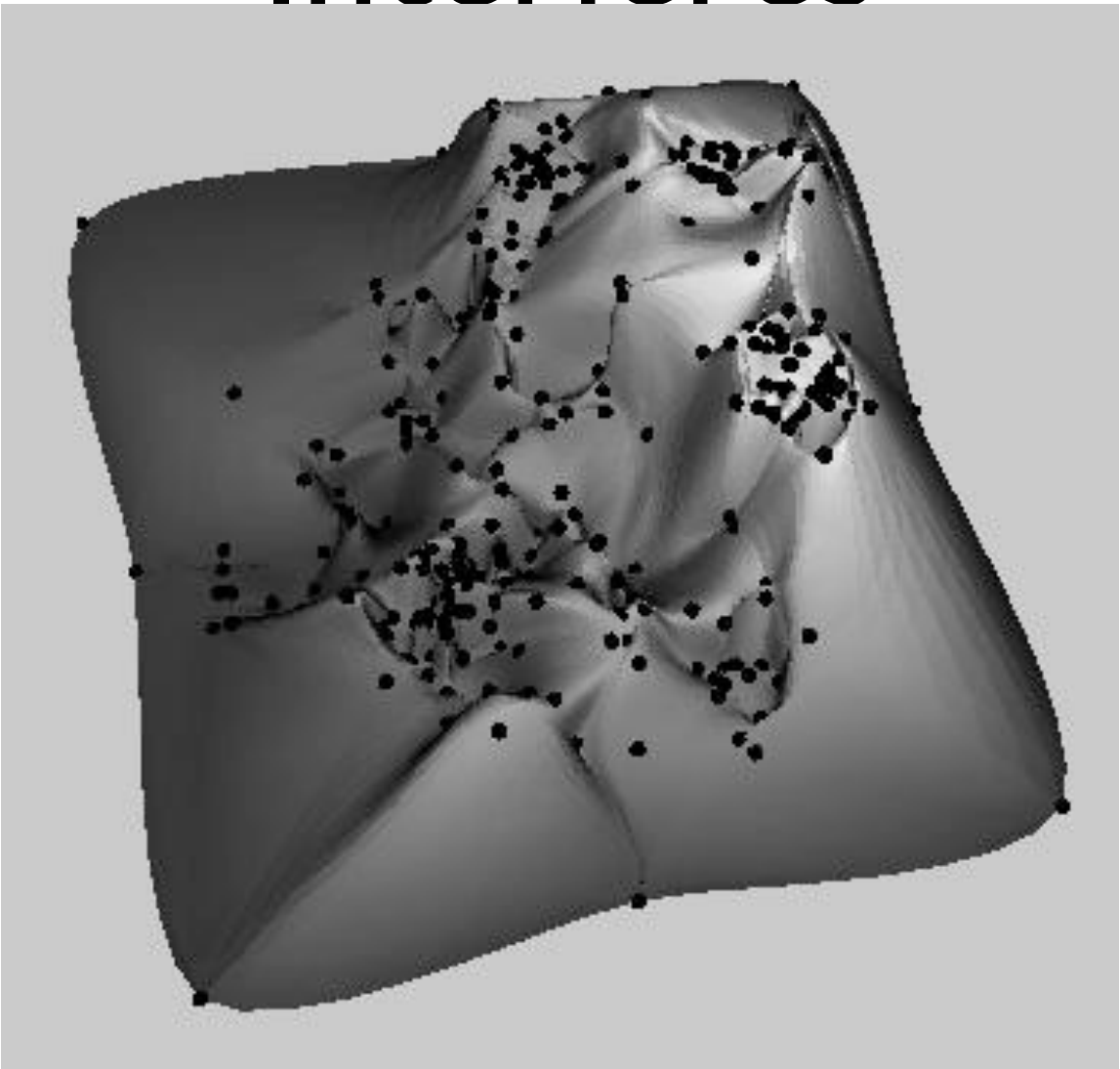

VLADIVOSTOK
JAPAN
BASIN
HOKKAIDO
MURORAN
JAPAN SEA
YAMATO
RIDGE
YAMATO
BASIN

No Rainbow ...

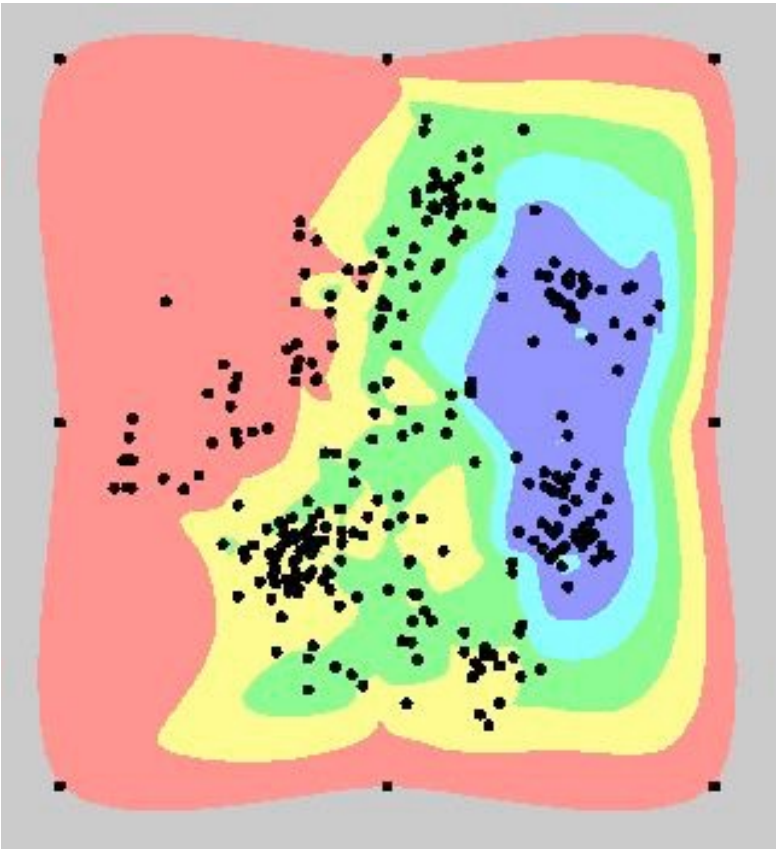

General Bathymetric Chart of the Oceans,
International Hydrographic Organization
(Ottawa, Canada, 5th edition, 1984). 5.06.
Metres
7000
6000
5000
4000
3000
2000
1000
500
200
sea level
areas below sea level
régions sous le niveau de la mer
Hypsometric Tints / Teintes Hypsométriques
Bathymetric Tints / Teintes Bathymétriques

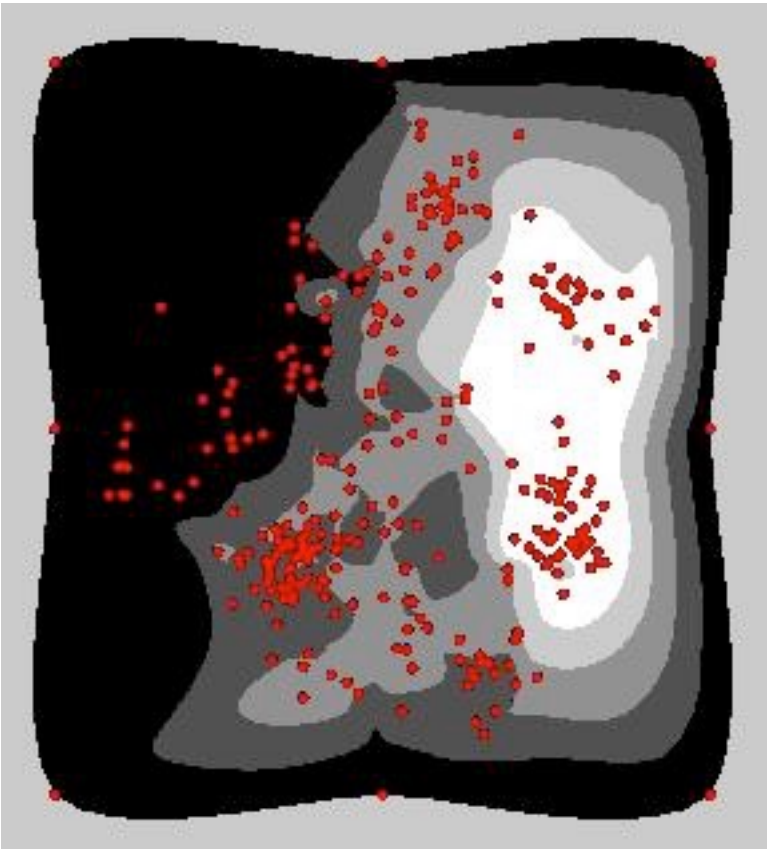

VLADIVOSTOK
JAPAN
BASIN
MURORAN
HOKKAIDO
JAPAN SEA
YAMATO
RIDGE
YAMATO
BASIN
TRENCH
RISE
2000
3500
1000
3000
500

3D Graphic Shading Interferes



Hue vs. Grayscale

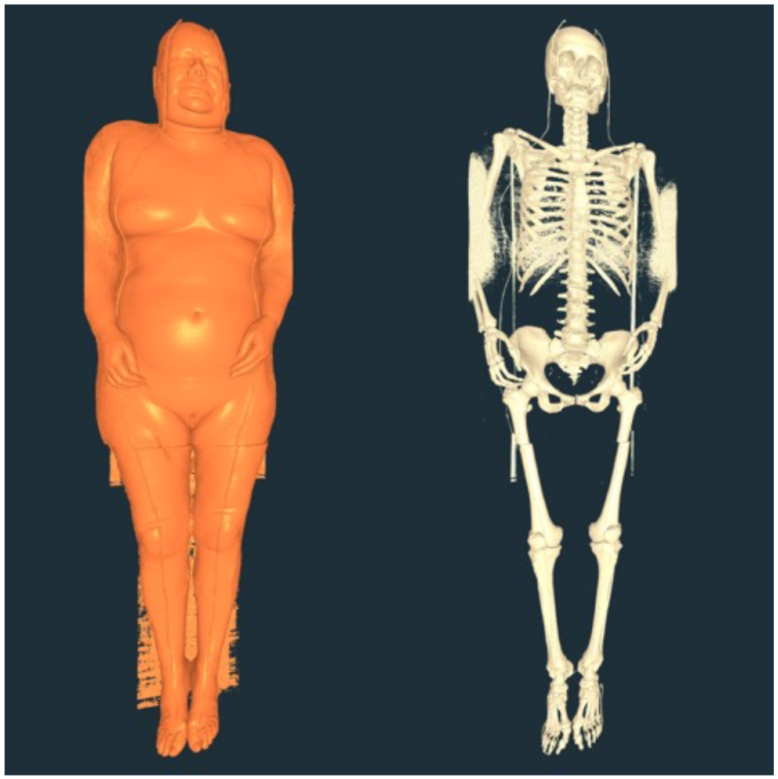


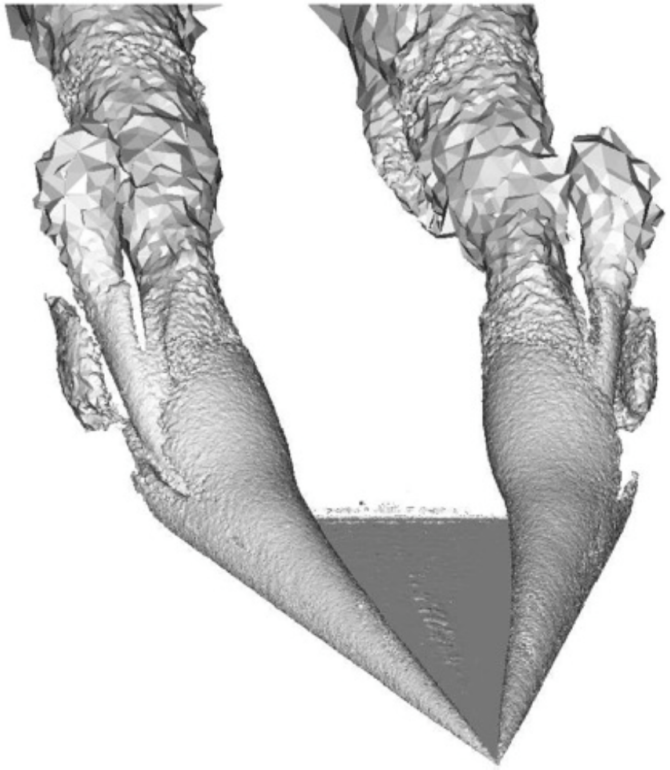


Isosurfaces
3D Contouring

Isosurfaces

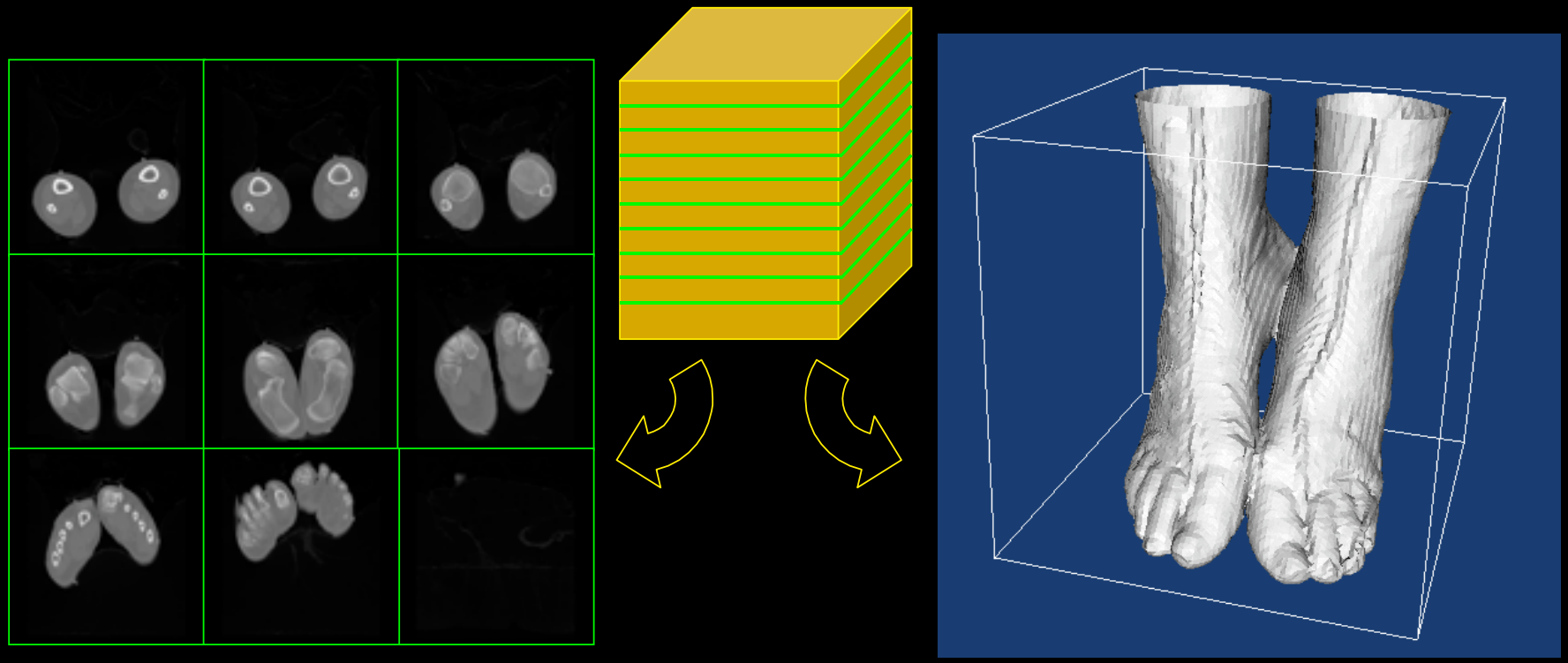
- An isosurface is a contour of a scalar field in 3D.
- An **isosurface** at a value *v* of a scalar field (volume) F(x,y,z) is the set of surfaces where F(x,y,z) = *v*.





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



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

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.

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

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: 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

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.

10,887 citations on Google Scholar

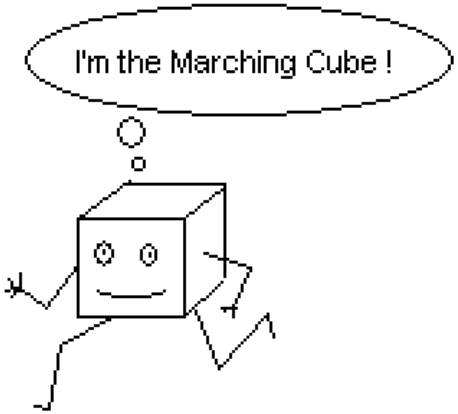
able medical tool. Images of these surfaces, constructed from multiple 2D slices of computed tomography (CT), magnetic resonance (MR), and single-photon emission computed tomography (SPECT), help physicians to understand the complex anatomy present in the slices. Interpretation of 2D

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

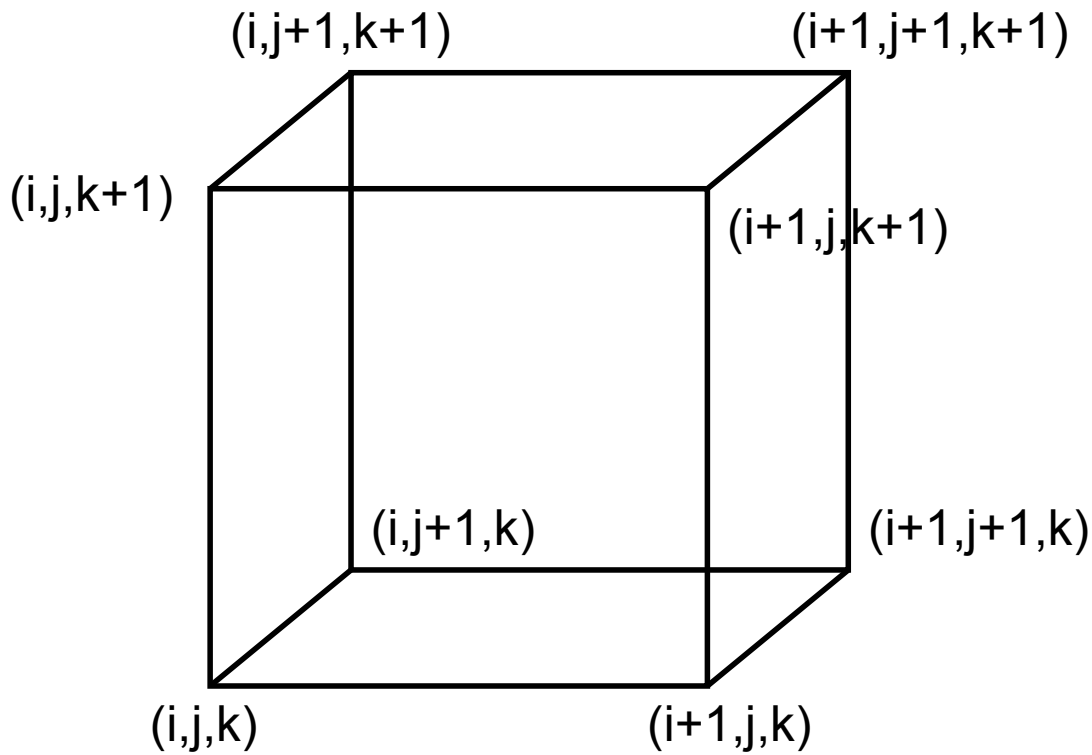
1. *Data acquisition.*
   This first step, performed by the medical imaging hardware, samples some property in a patient and pro-

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

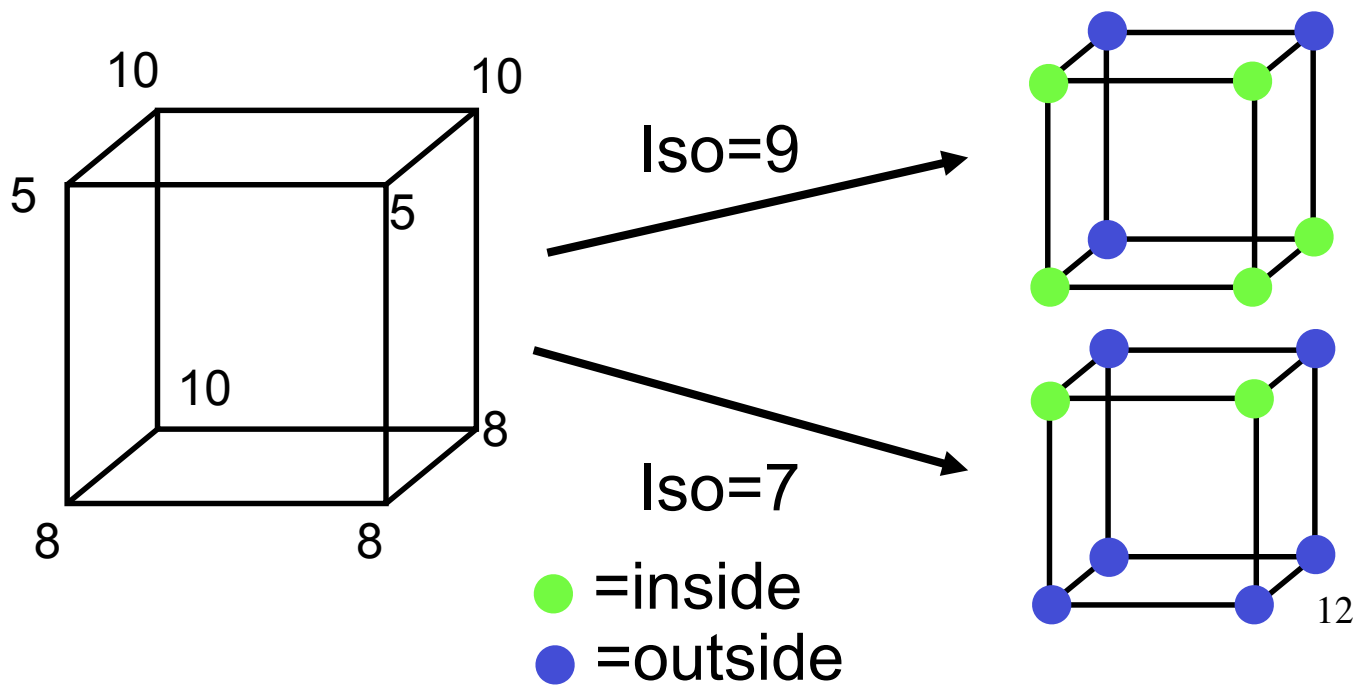
1. Consider a cell
2. Classify each vertex as inside or outside
3. Build an index
4. Get edge list from table[index]
5. Interpolate the edge location
6. Compute gradients
7. Consider ambiguous cases
8. Go to next cell



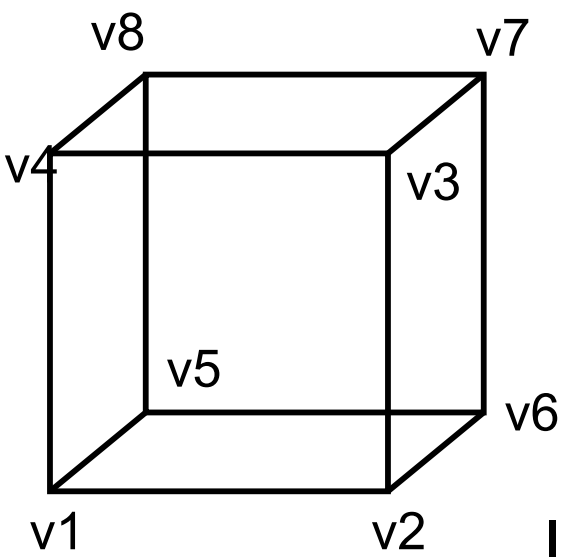
- Step 1: Consider a cell defined by eight data values



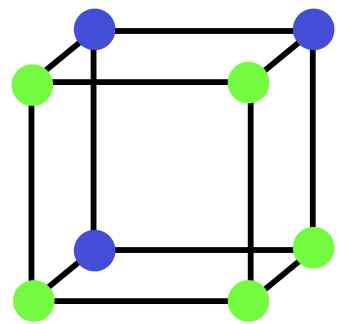
- Step 2: Classify each voxel according to whether it lies
  - Outside the surface (value > isosurface value)
  - Inside the surface (value <= isosurface value)



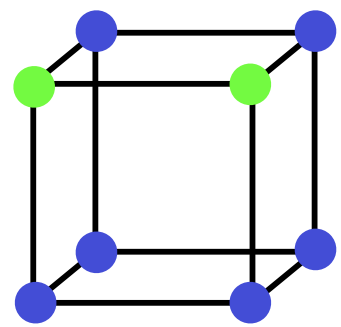
- Step 3: Use the binary labeling of each voxel to create an index



inside =1
outside=0



11110100

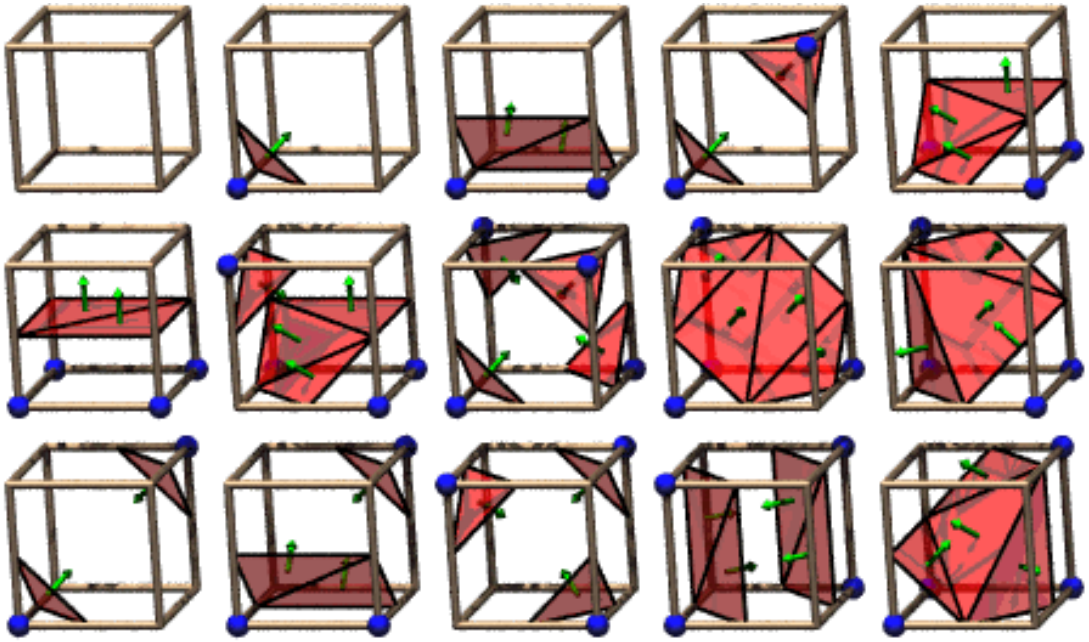


00110000

Index:

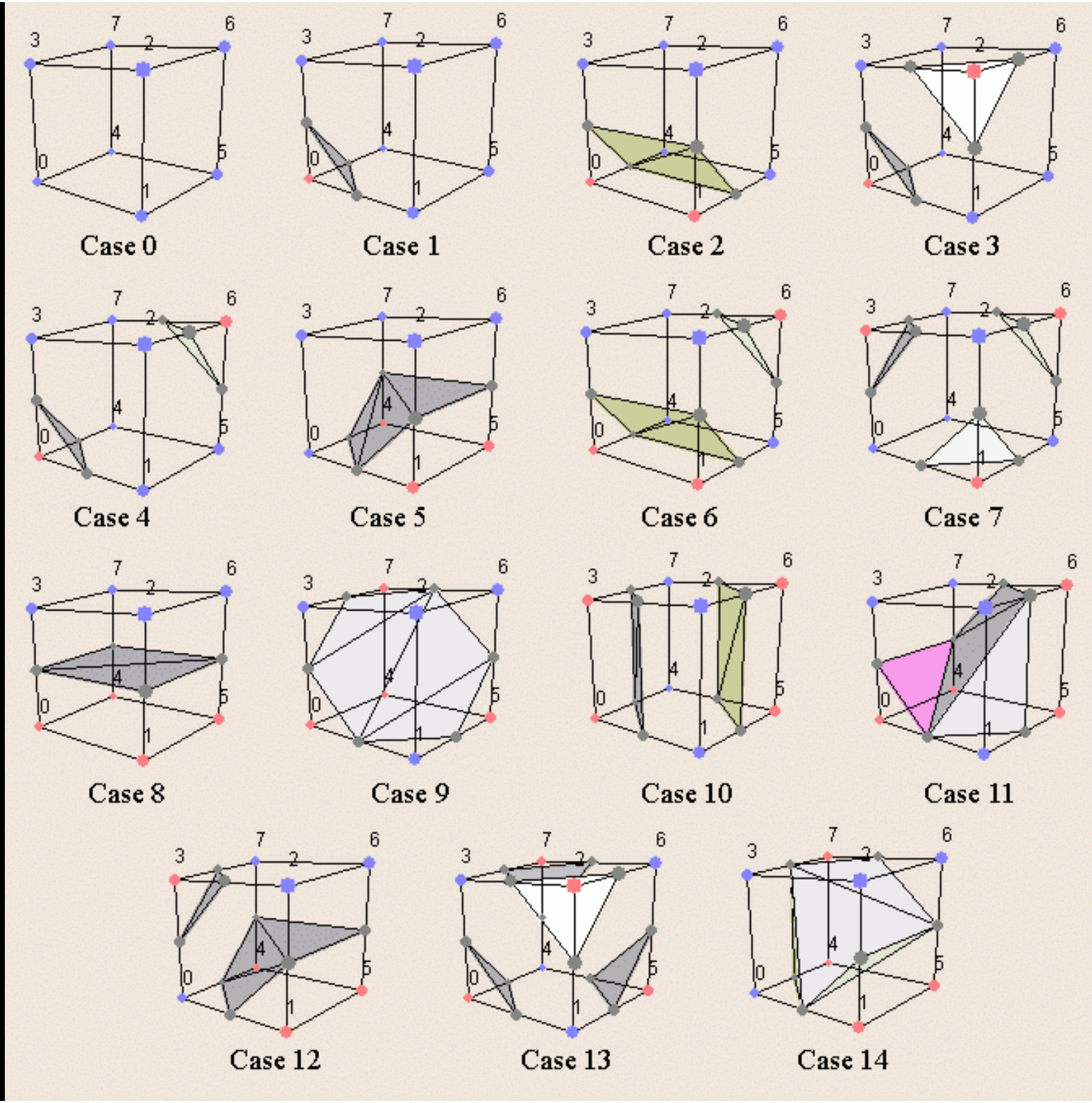
v1	v2	v3	v4	v5	v6	v7	v8

- 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



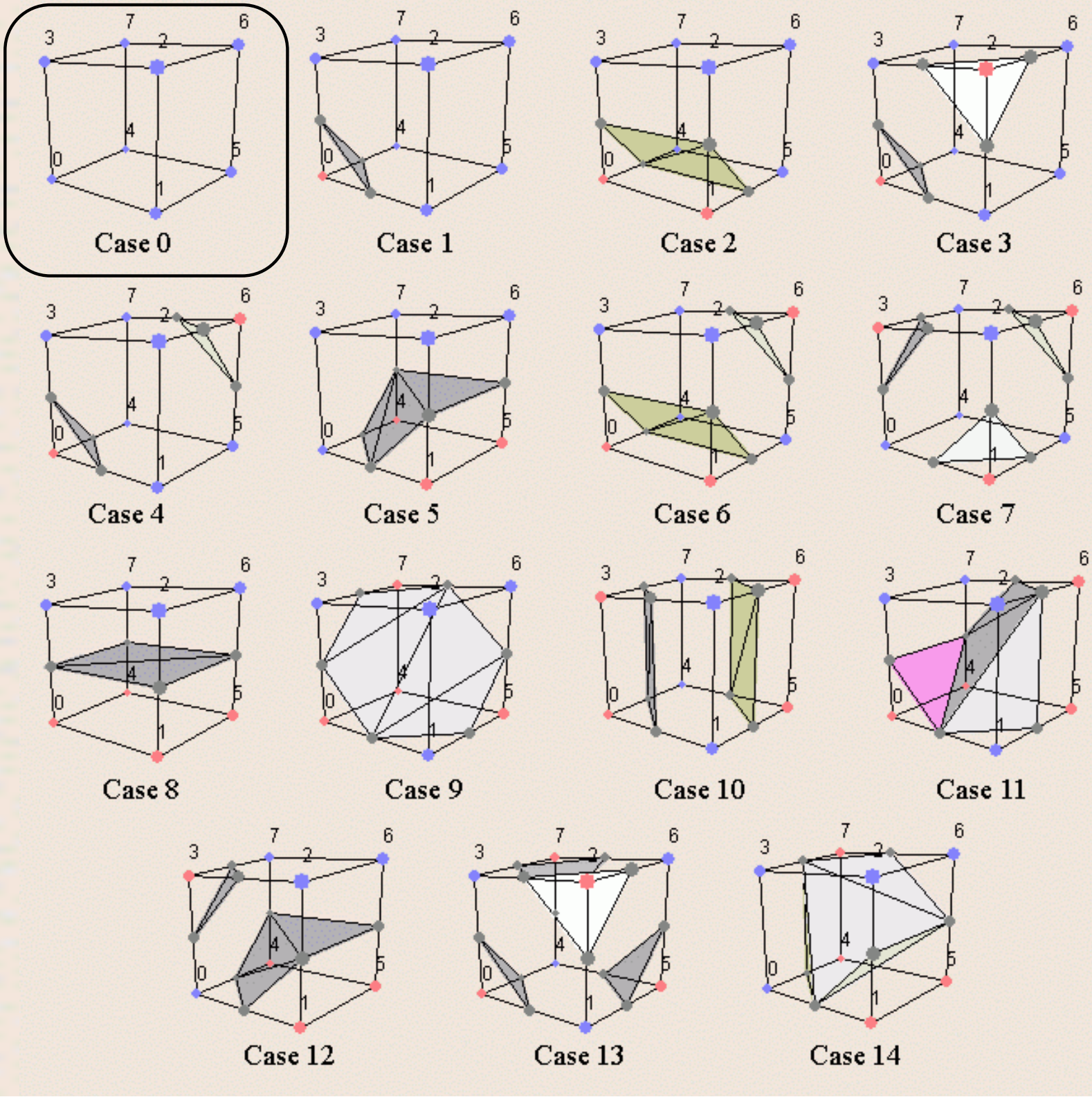
The 15 Cube Combinations

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

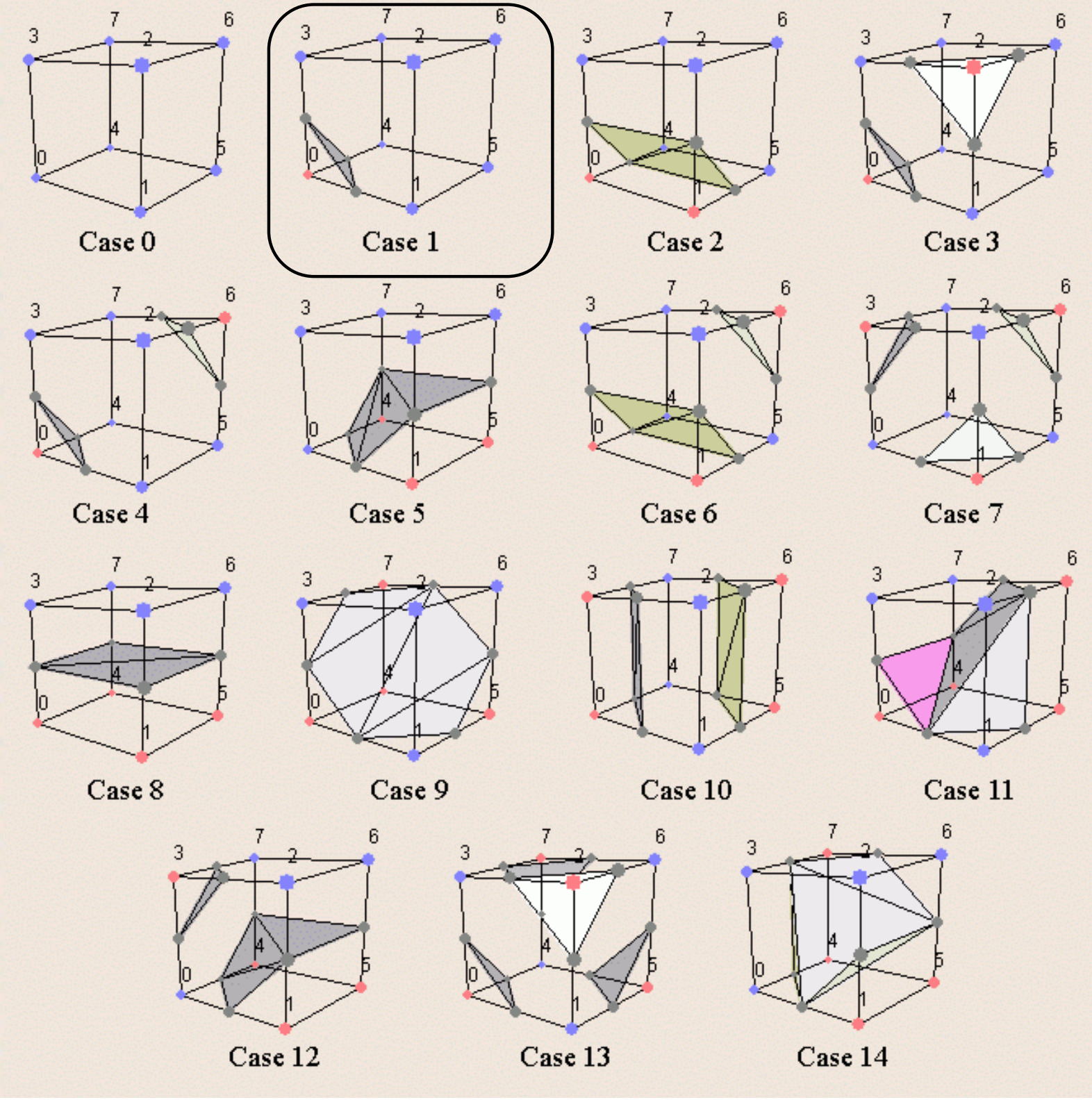
- 8 Above
- 0 Below

1 case



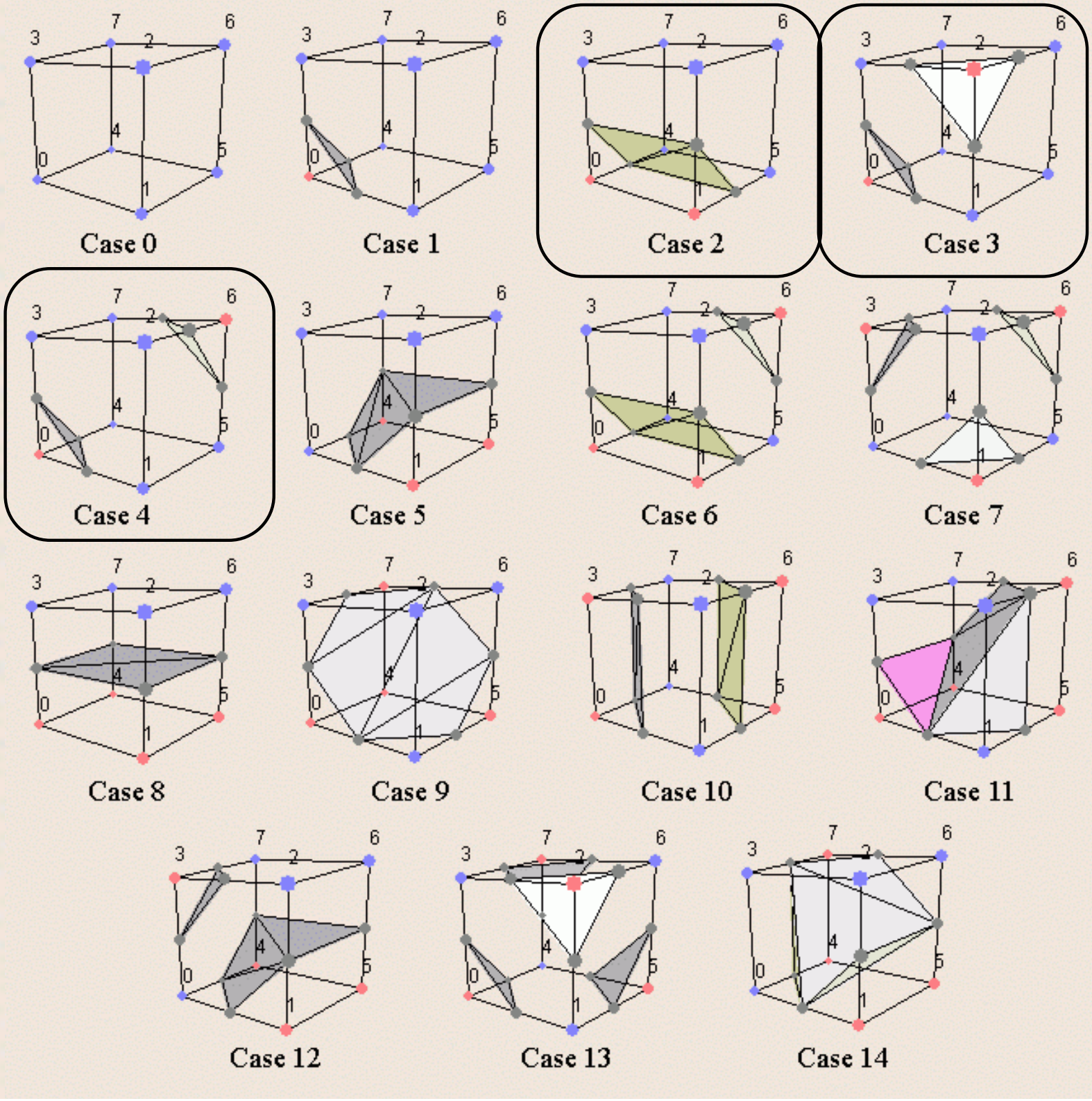
- 7 Above
- 1 Below

1 case



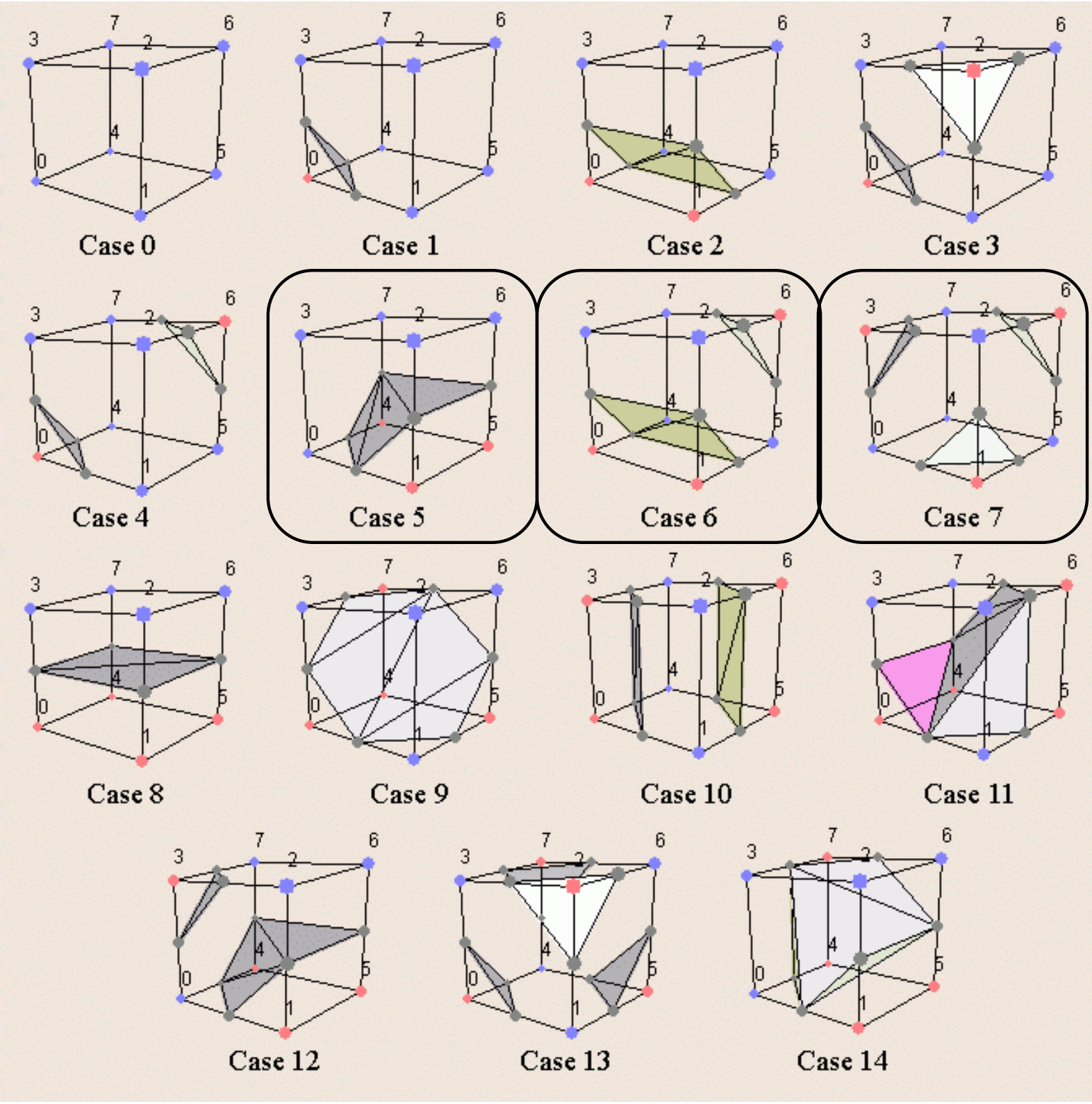
- 6 Above
- 2 Below

3 cases



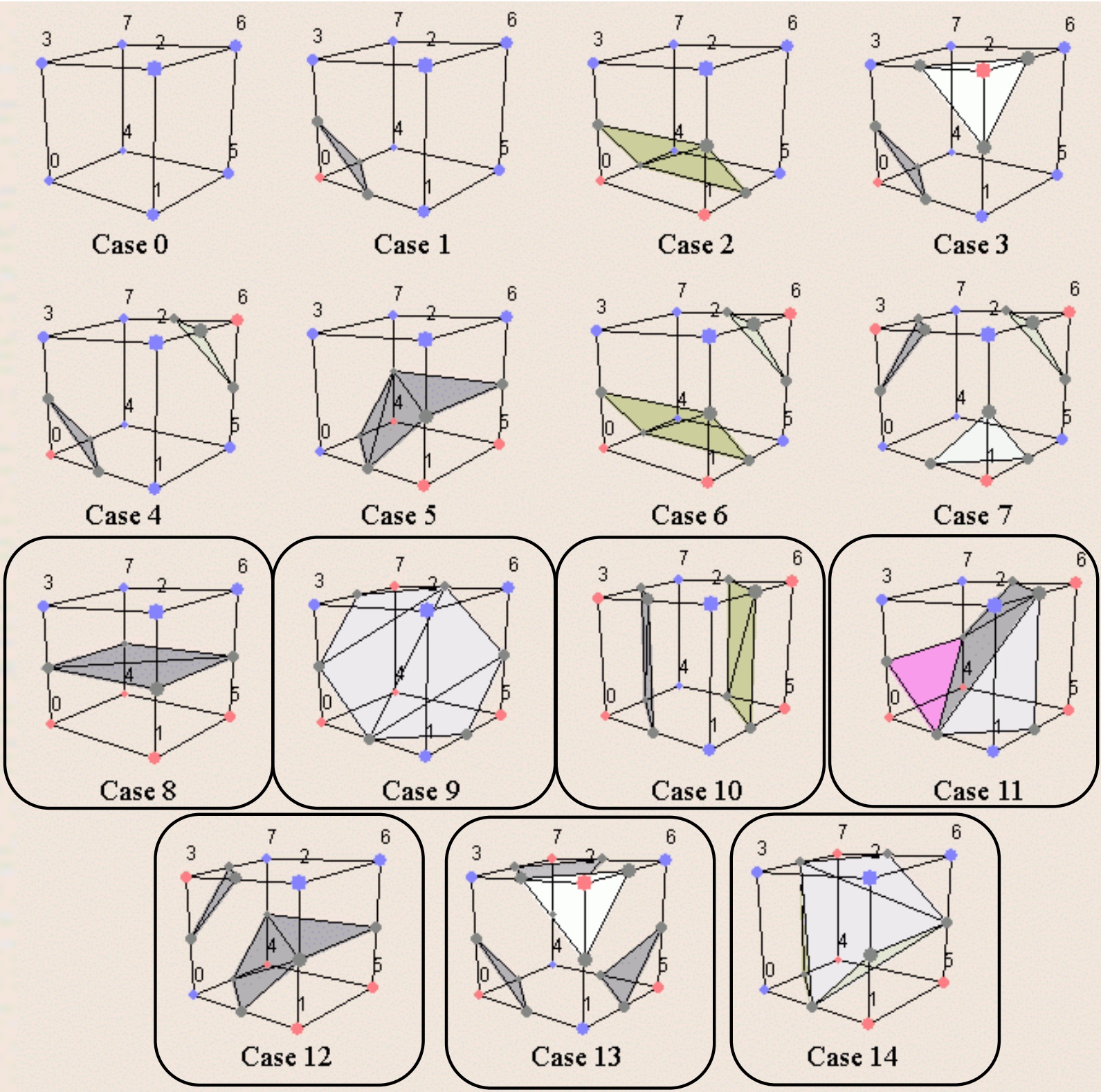
- 5 Above
- 3 Below

3 cases



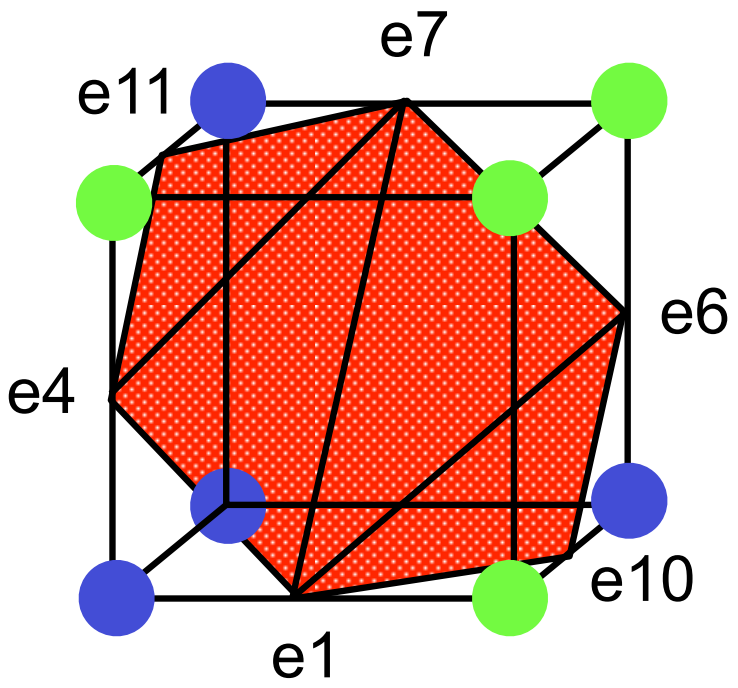
- 4 Above
- 4 Below

7 cases

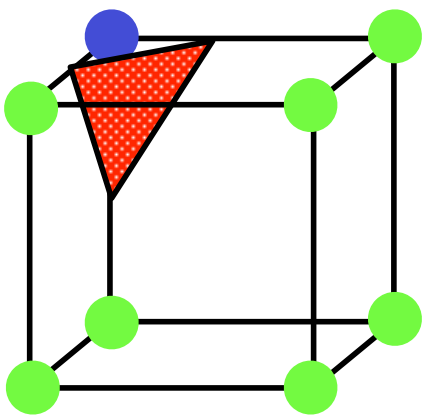


- Step 4 *cont.*: Get edge list from table
  - Example for

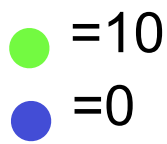
Index = 10110001
triangle 1 = e4,e7,e11
triangle 2 = e1, e7, e4
triangle 3 = e1, e6, e7
triangle 4 = e1, e10, e6

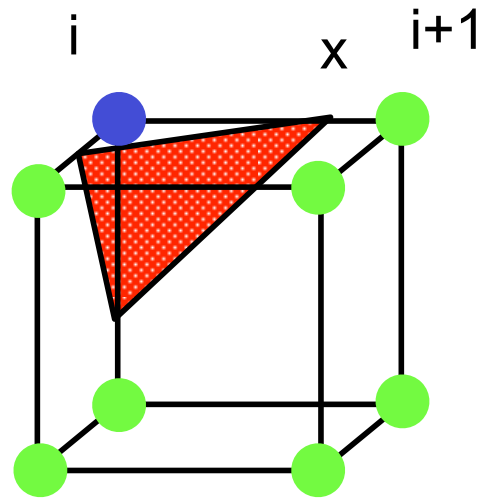


- Step 5: For each triangle edge, find the vertex location along the edge using linear interpolation of the voxel values



T=5

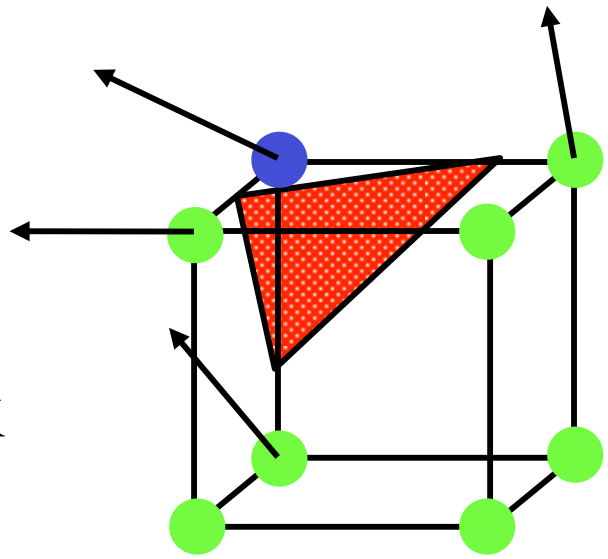




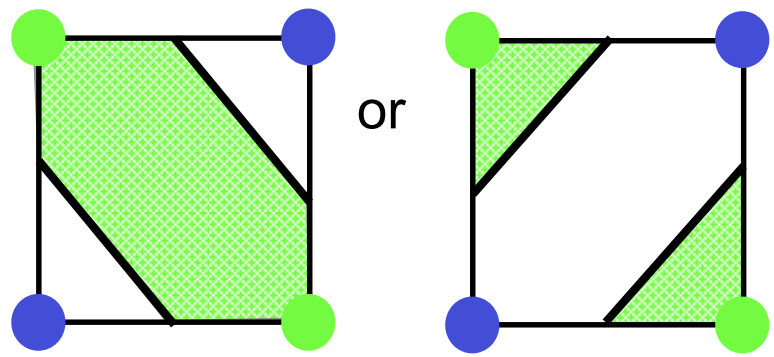
T=8

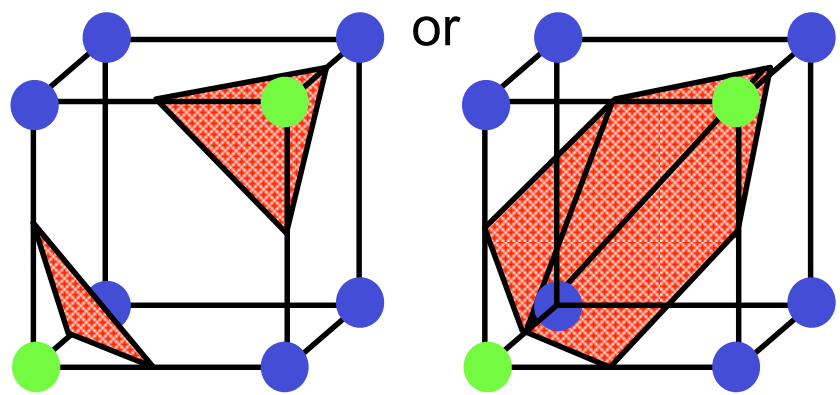
$$x = i + \left(\frac{T - v[i]}{v[i+1] - v[i]} \right)$$

- 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!)





Hint: there is no “right”, just “consistent”.

The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes

Gregory M. Nielson Bernd Hamann

Computer Science
Arizona State University
Tempe, AZ 85287-5406

Abstract

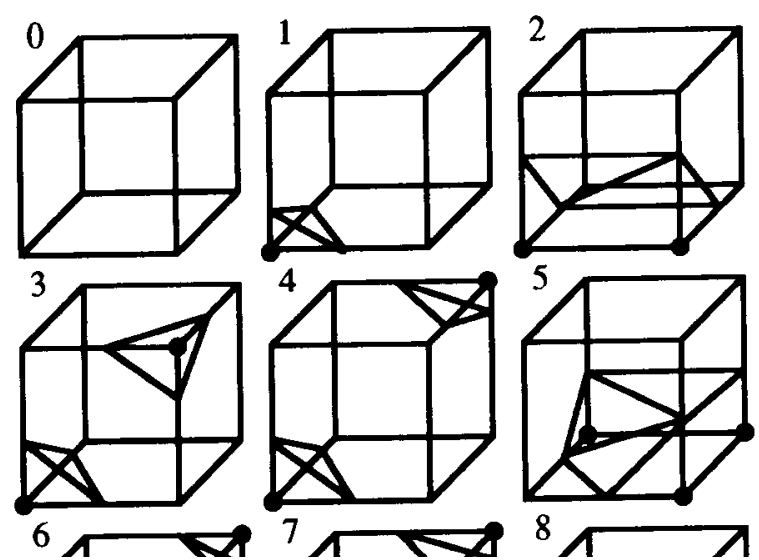
A method for computing isovalue or contour surfaces of a trivariate function is discussed. The input data are values of the trivariate function, F_{ijk}, at the cuberille grid points (x_i, y_j, z_k) and the output is a collection of triangles representing the surface consisting of all points where $F(x,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



The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes

Gregory M. Nielson Bernd Hamann

Computer Science
Arizona State University
Tempe, AZ 85287-5406

Abstract

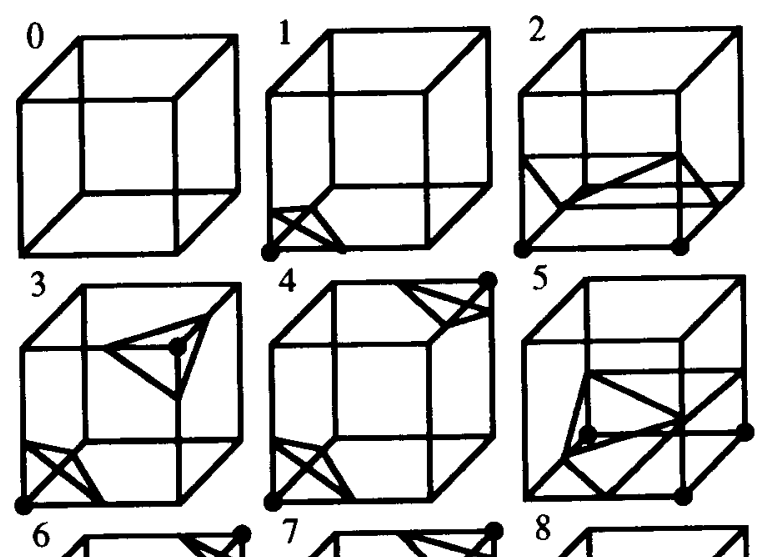
A method for computing isovalue or contour surfaces of a trivariate function is discussed. The input data are values of the trivariate function, F_{ijk}, at the cuberille grid points (x_i, y_j, z_k) and the output is a collection of triangles representing the surface consisting of all points where $F(x,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

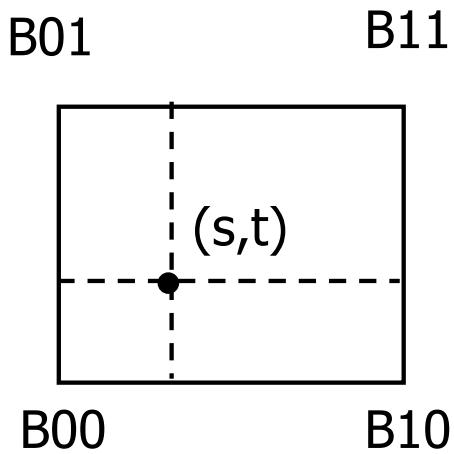
$$S_\alpha = \{ (x, y , z) : F(x, y, z) = \alpha\}.$$

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.



Asymptotic Decider (1)

- Based on bilinear interpolation over faces



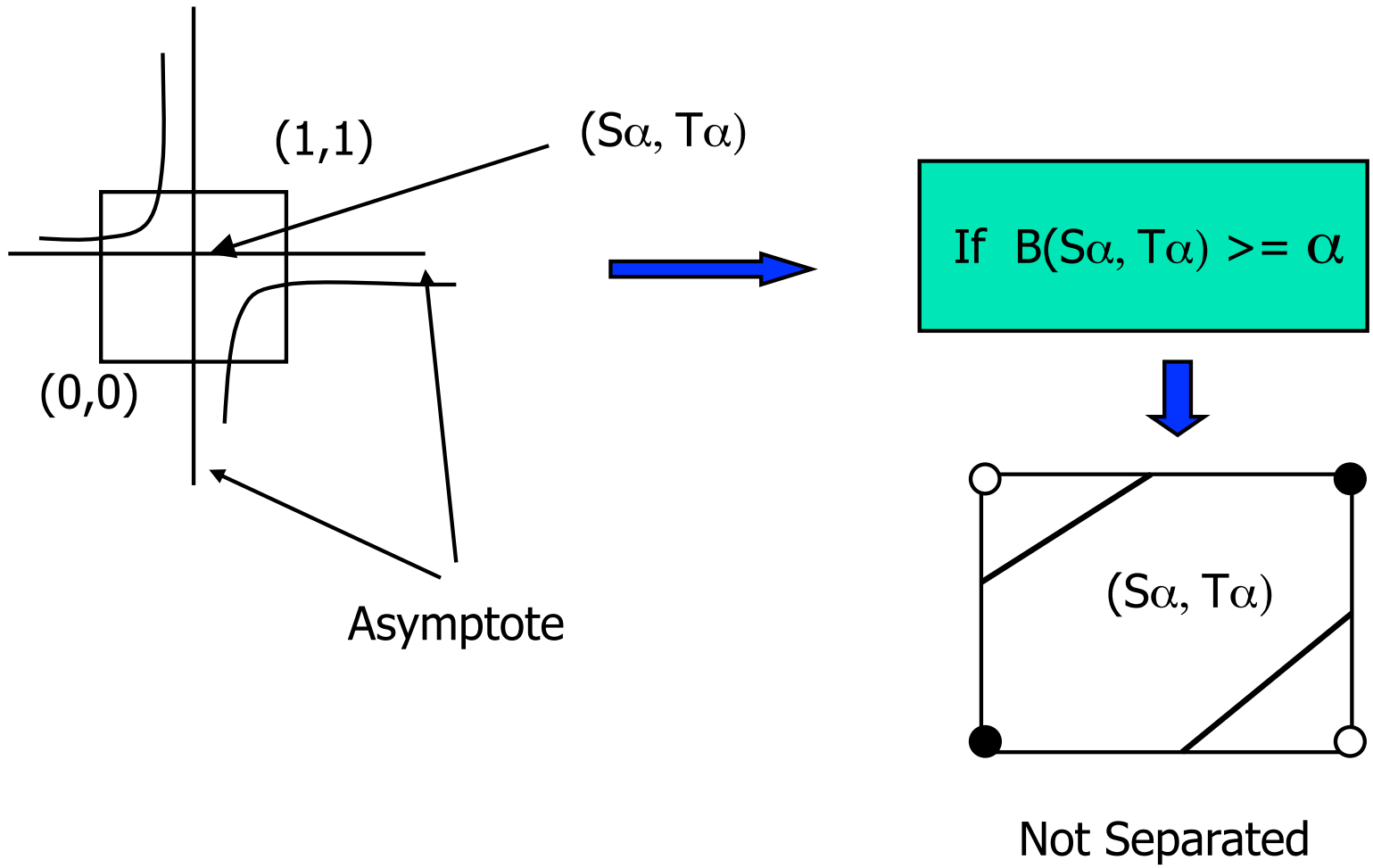
$$B(s,t) = (1-s,\ s) \begin{vmatrix} B00 & B01 \\ B10 & B11 \end{vmatrix} \begin{vmatrix} 1-t \\ t \end{vmatrix}$$

$$= B00(1-s)(1-t) + B10(s)(1-t) + B01(1-s)(t) + B11(s)(t)$$

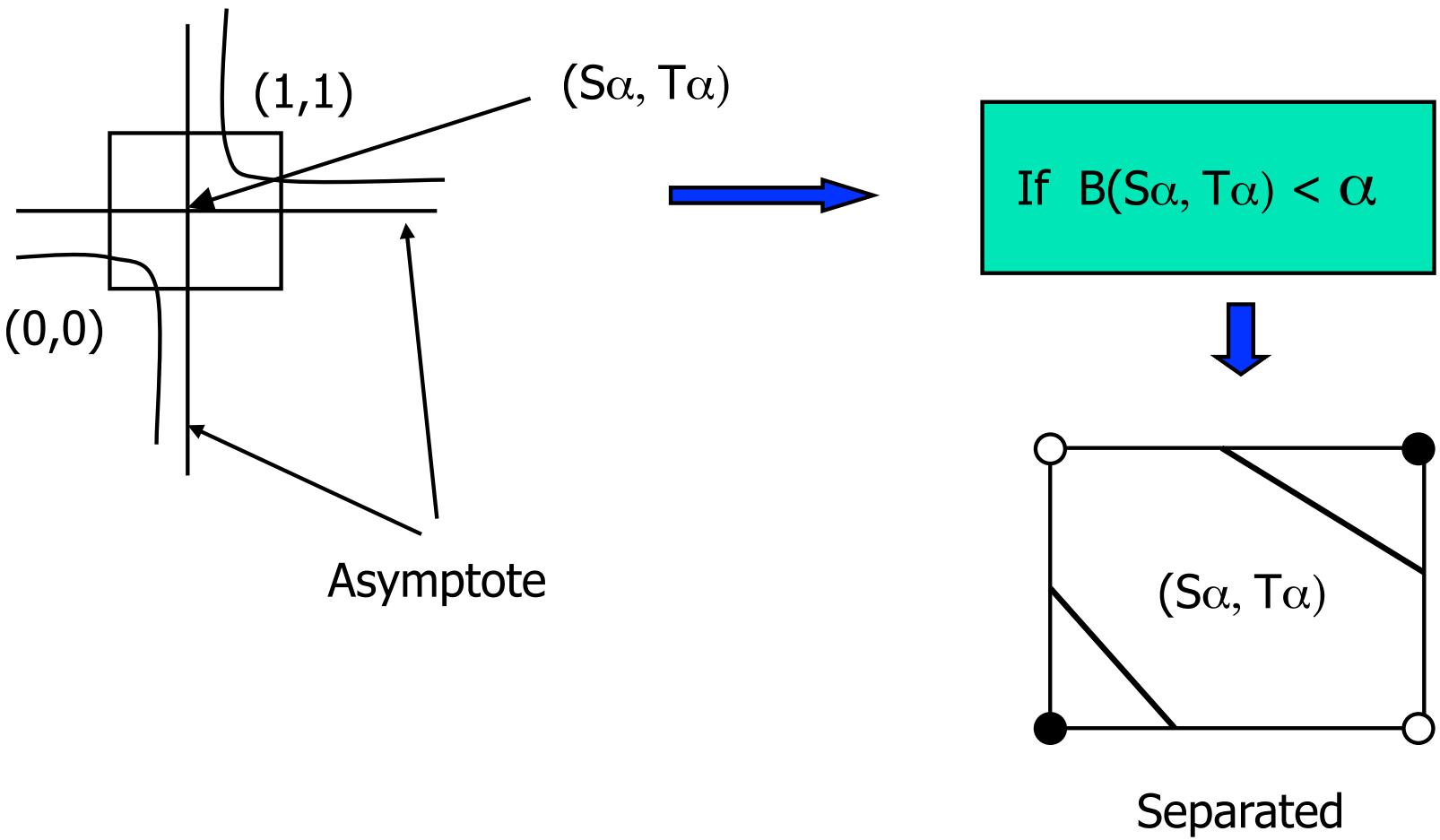
The contour curves of B:

$\{(s,t) \mid B(s,t) = \alpha\}$ are hyperbolas

Asymptotic Decider (2)

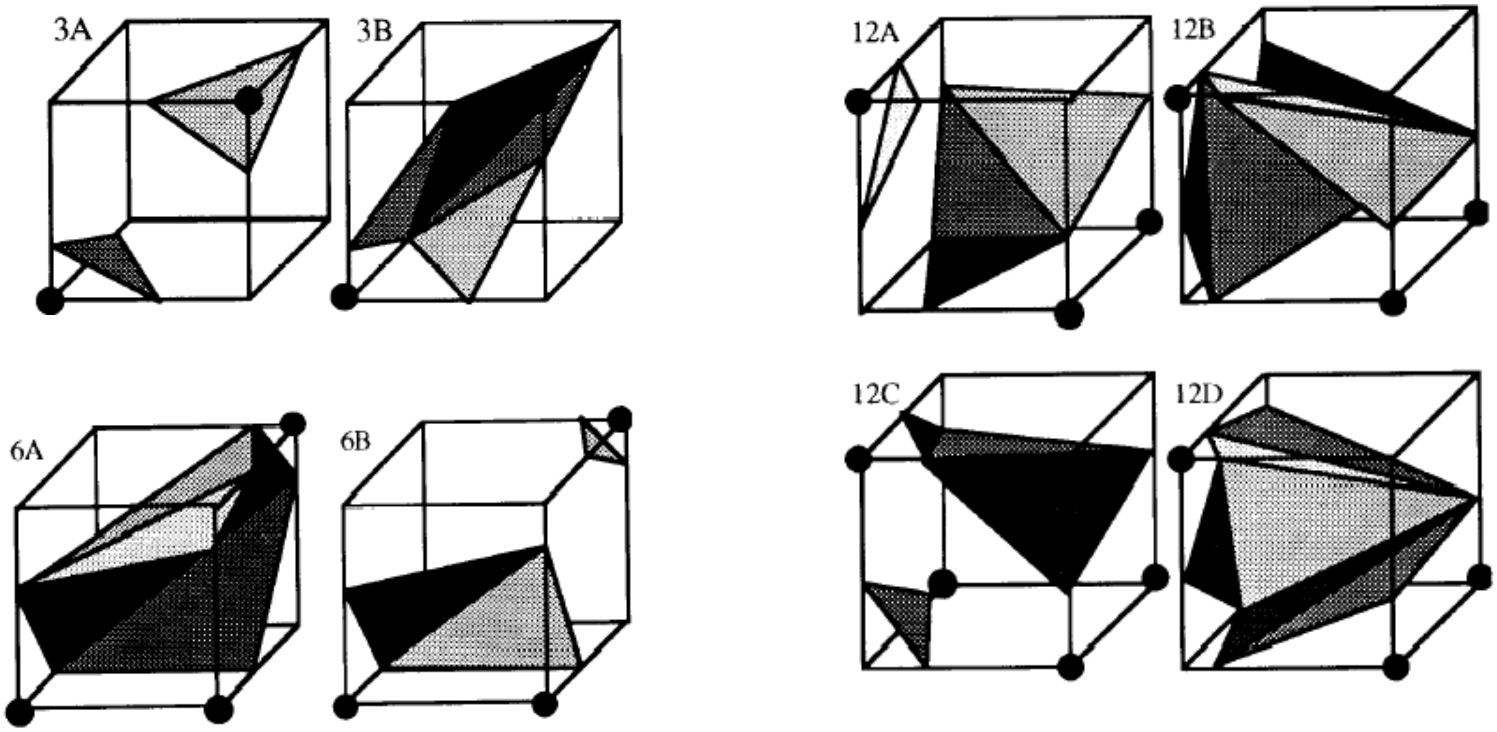


Asymptotic Decider (3)

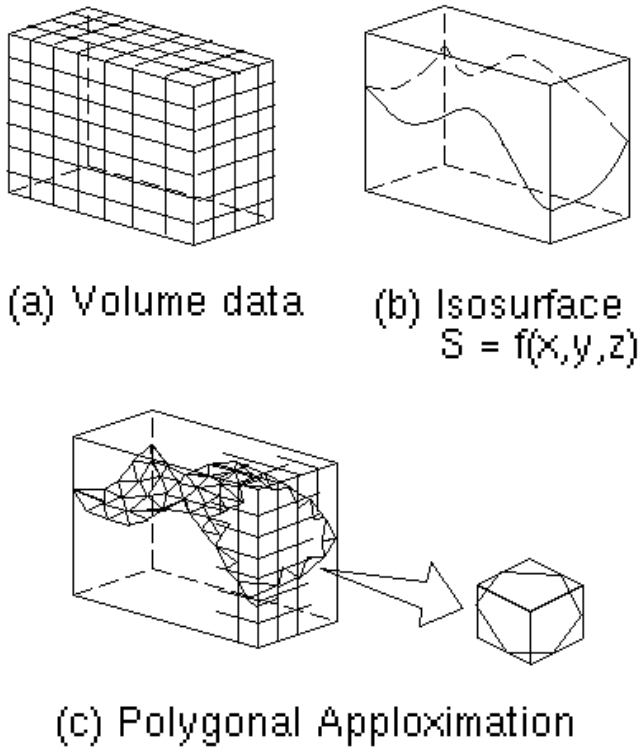


Asymptotic Decider (5)

- case 3, 6, 12, 10, 7, 13
  - (These are the cases with at least one ambiguious faces)

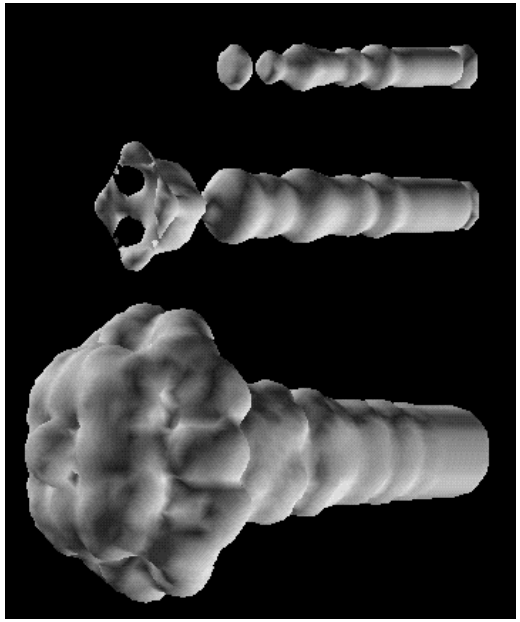


- Summary
  - 256 Cases
  - Reduce to 15 cases by symmetry
  - Ambiguity in cases
    3, 6, 7, 10, 12, 13
  - Causes holes if arbitrary choices are made
- Up to 5 triangles per cube
- Several isosurfaces
  - Run MC several times
  - Semi-transparency requires spatial sorting

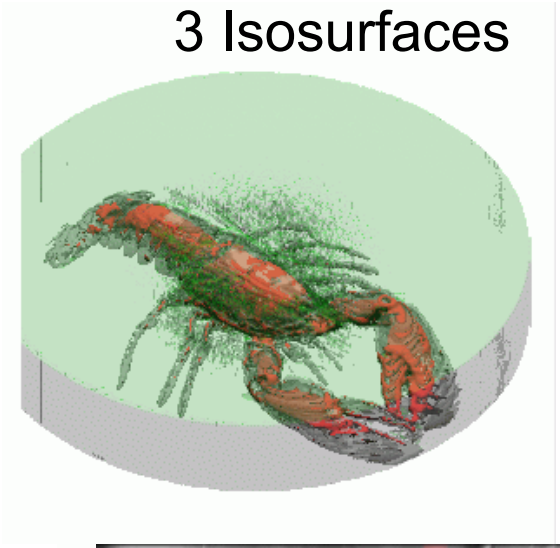


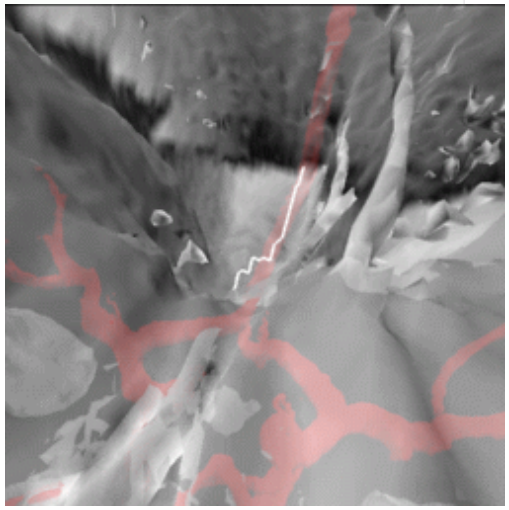
- Examples

1 Isosurface



3 Isosurfaces

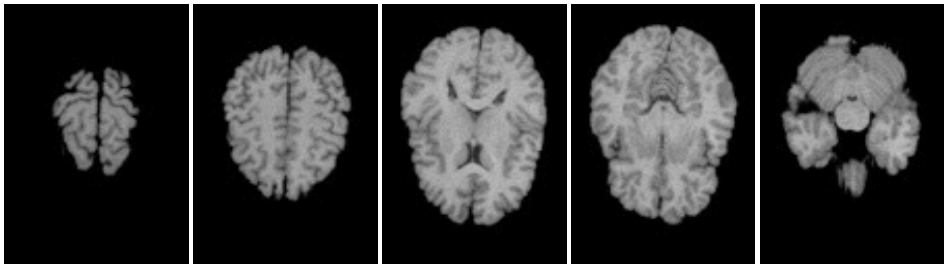


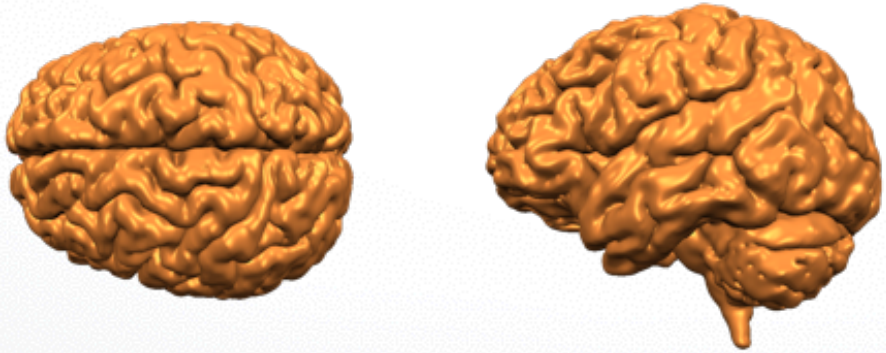


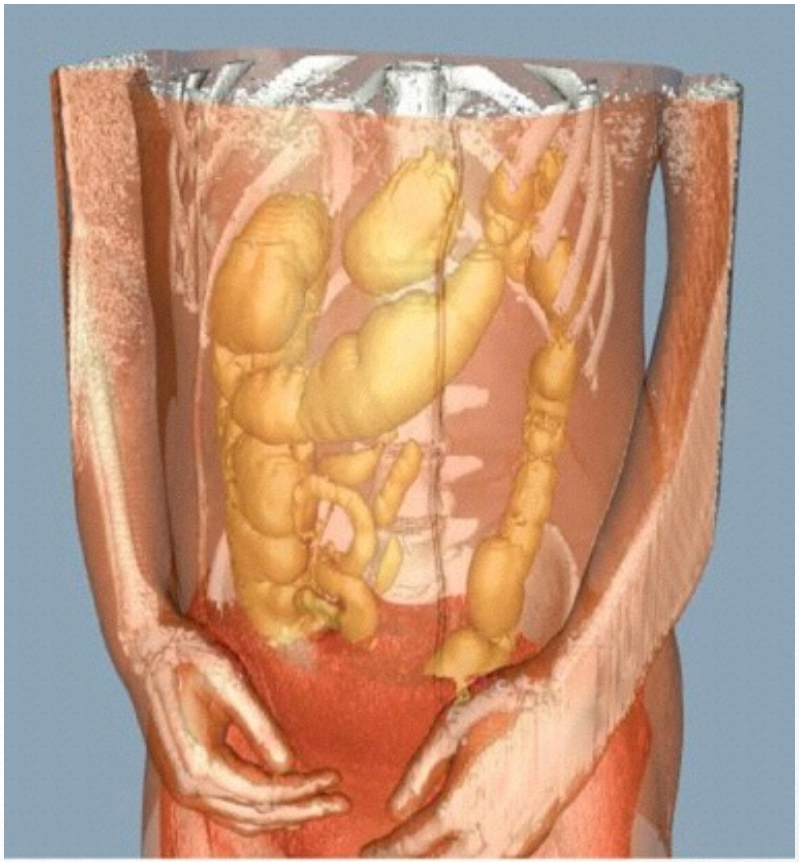
2 Isosurfaces

Marching Cubes

Algorithm for isosurface extraction from medical scans (CT, MRI)



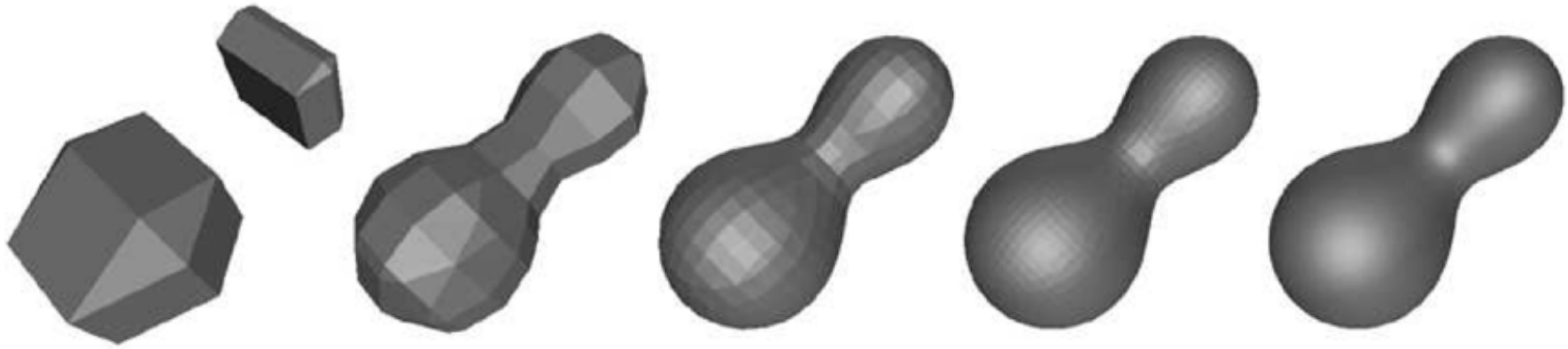




Slides: Hao Li, USC

Marching Cubes

Effect of grid size



Grid size=10
70 Facets

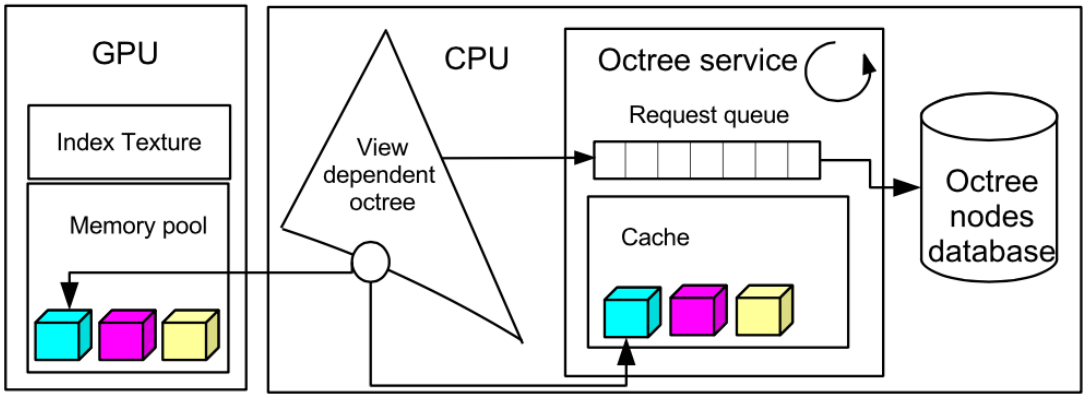
Grid size=5
220 Facets

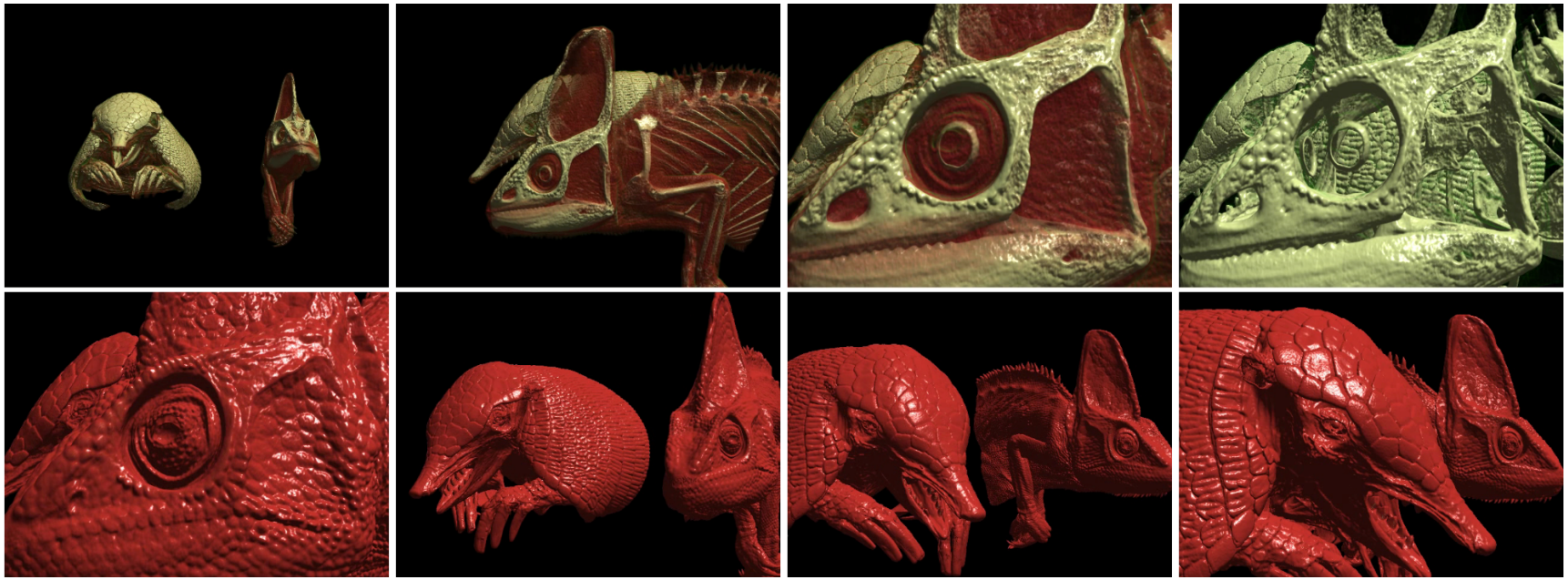
Grid size=2
1700 Facets

Grid size=1
6800 Facets

Grid size=0.5
27000 Facets

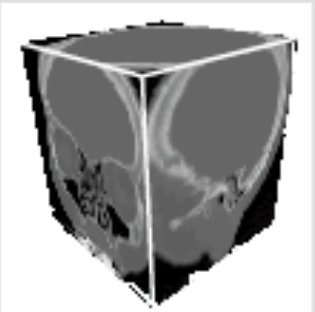
Fast GPU isosurface ray casting

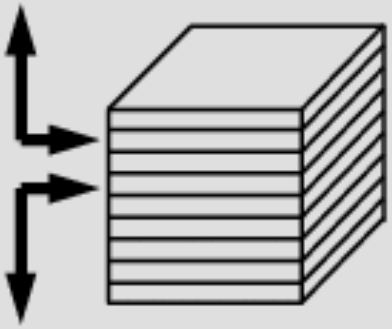


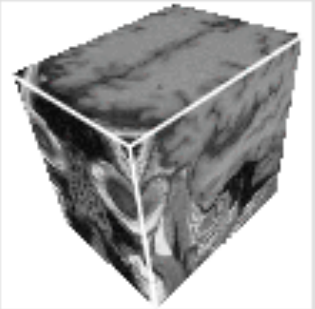


E Gobbetti et al. A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. The Visual Computer, 2008

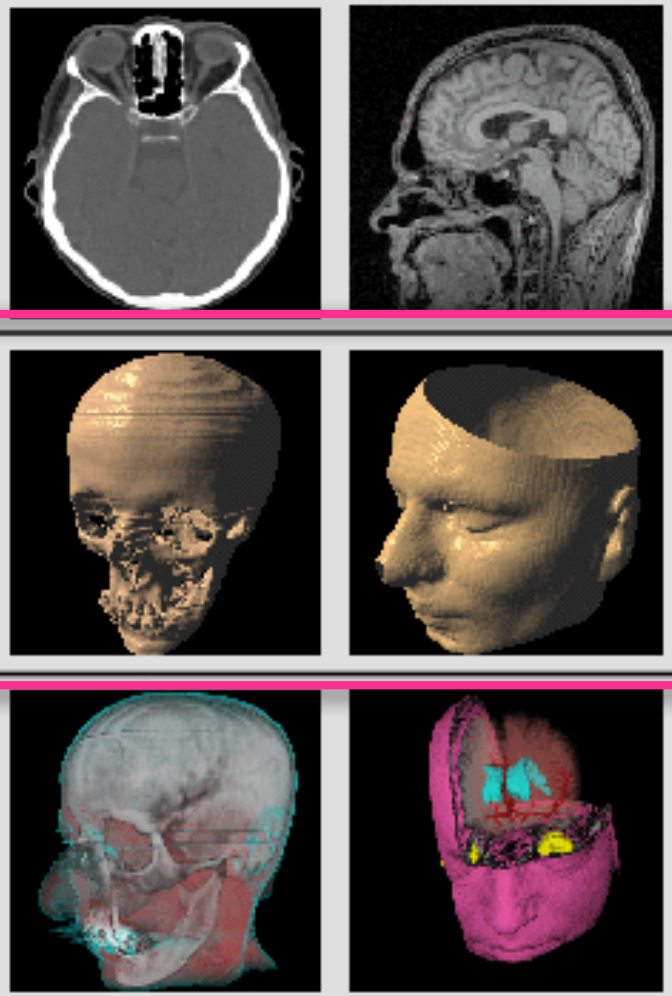
Surfaces







- 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)



Explicit vs Implicit

- In graphics, we often differentiate between *explicit* and *implicit geometry*.
- For our purposes in scientific visualization:
  - *Explicit geometry* is defined directly by vertices.
    - i.e. a triangle mesh
  - *Implicit geometry* is defined by an isovalue of an implicit function (specifically, the scalar field)
    - i.e., an isosurface of volume data
  - *Parametric geometry*: explicit geometry in R^n interpolated via parametric equations in R^{n-1}
    - I.e. a heightfield of uniform vertices, interpolated via B-spline patches
    - Depending on parameterization, can be implicit (converted into a scalar field) or explicit (requires geometric subdivision). To learn more, take Elaine Cohen's CAGD class.
- **Indirect visualization** usually involves turning implicit geometry into explicit geometry to be rasterized.

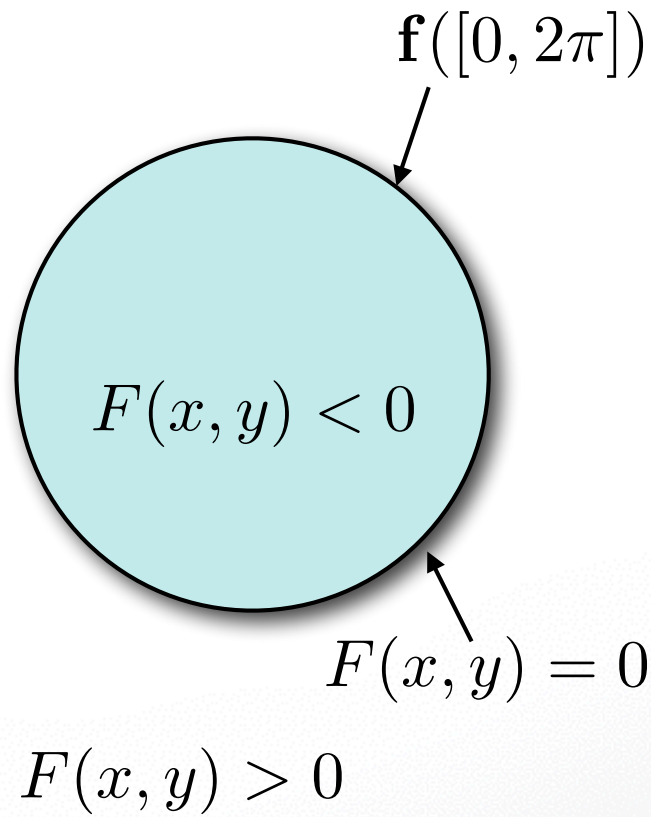
Explicit vs. Implicit

Explicit: $\mathbf{f}(x) = (r\cos(x), r\sin(x))^T$

- Range of parameterization function

Implicit: $F(x, y) = \sqrt{x^2 + y^2} - r$

- Kernel of implicit function



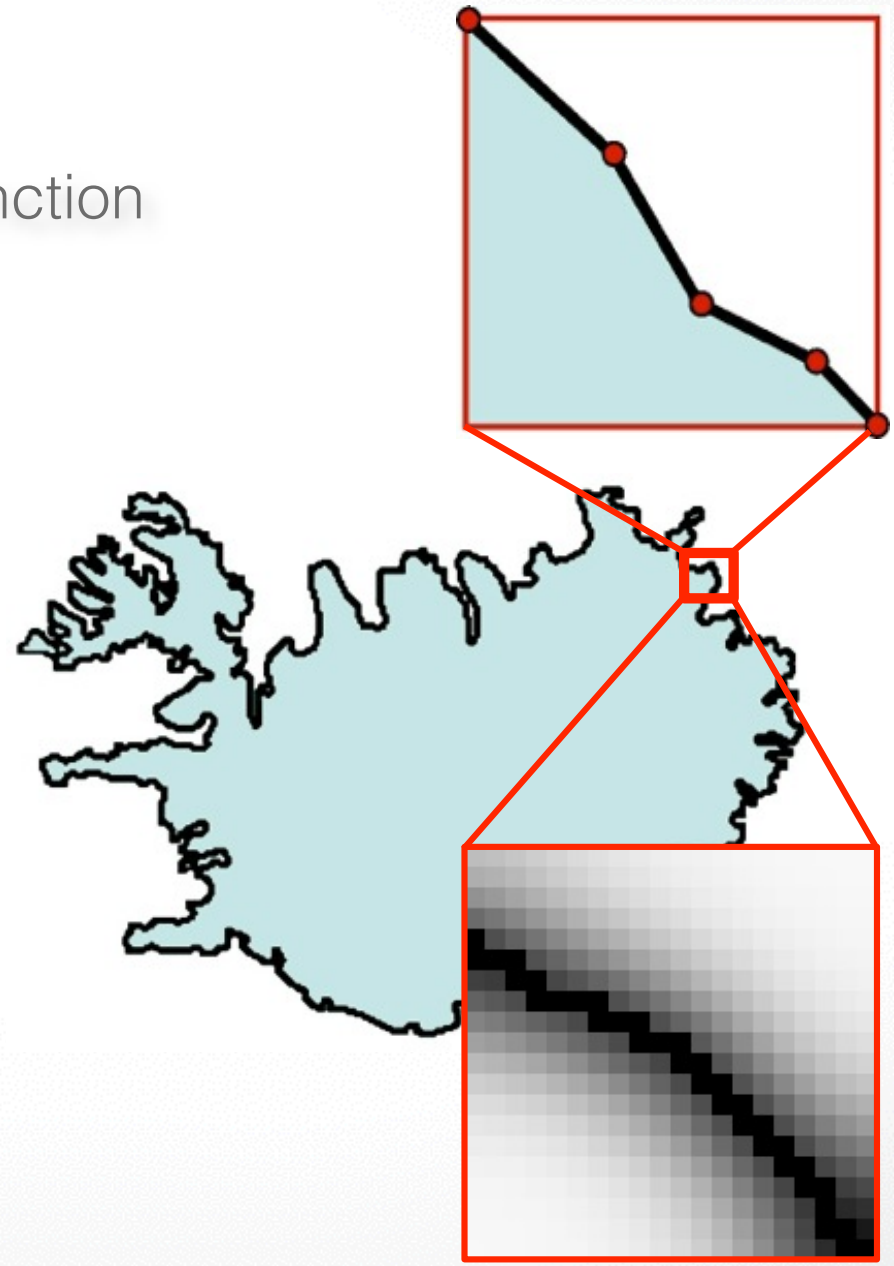
Explicit vs. Implicit

Explicit: $\mathbf{f}(x) = \ ?$

- Range of parameterization function
- Piecewise approximation

Implicit: $F(x, y) = \ ?$

- Kernel of implicit function
- Piecewise approximation



Slides: Hao Li, USC

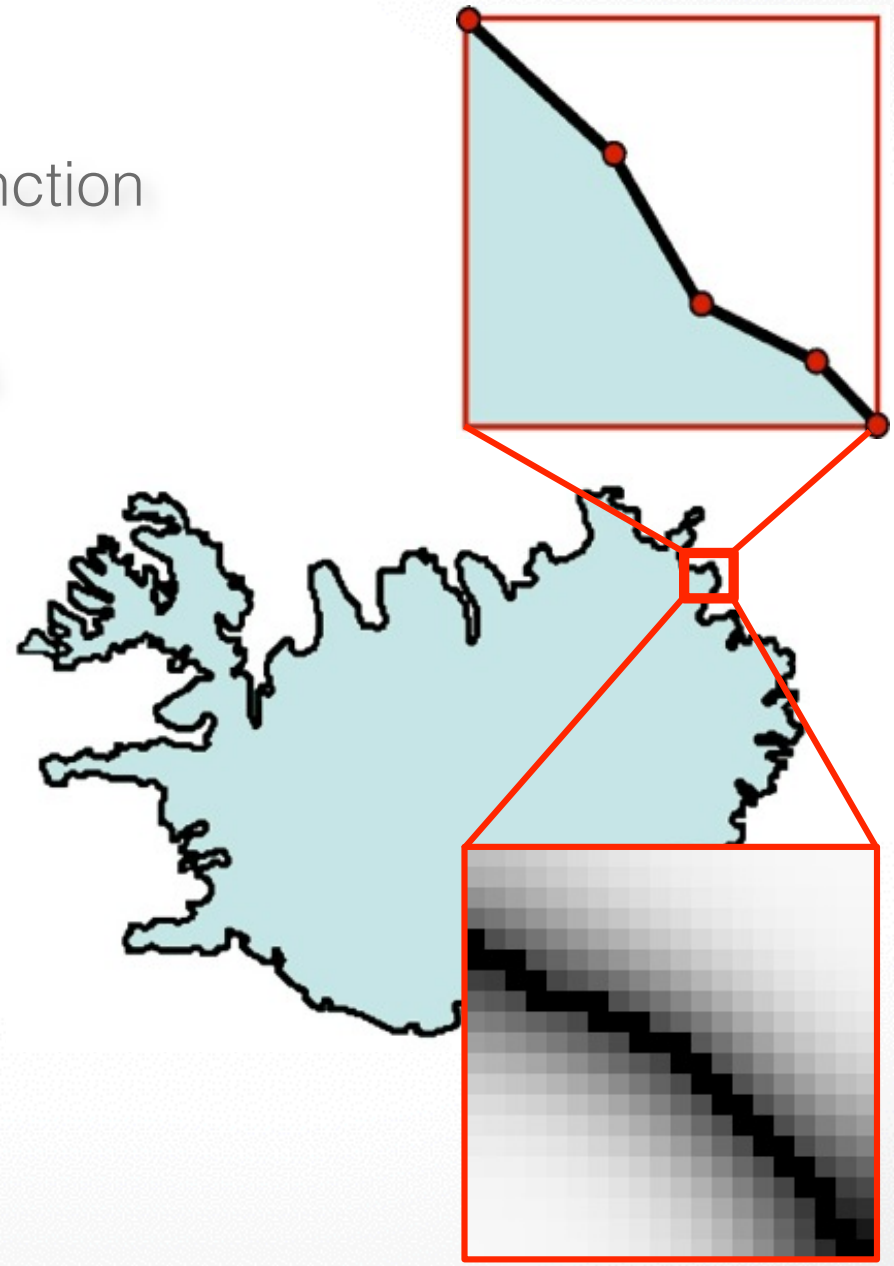
Explicit vs. Implicit

Explicit:

- Range of parameterization function
- Piecewise approximation
- Splines, triangle mesh, points
- Easy enumeration
- Easy geometry modification

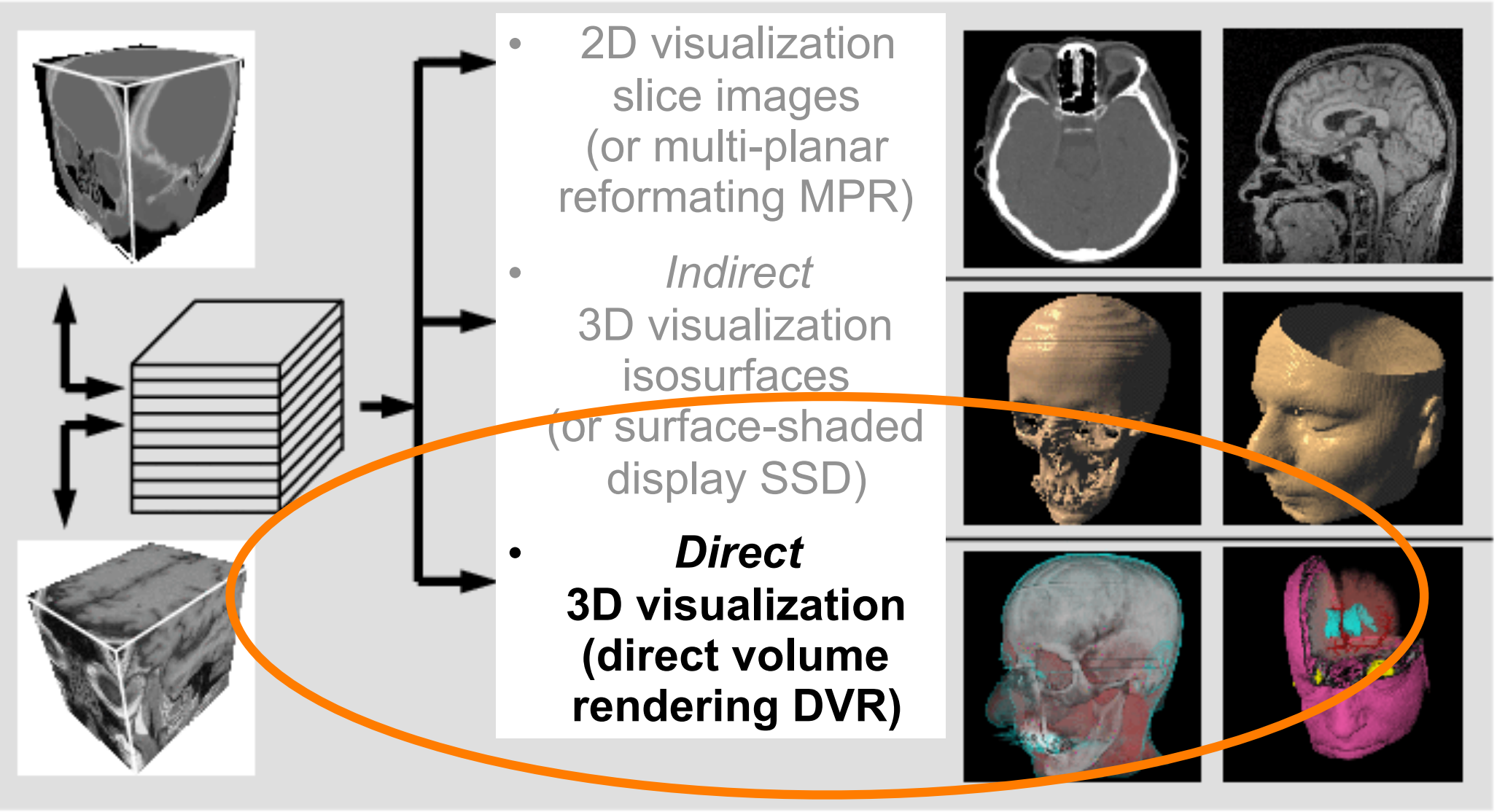
Implicit:

- Kernel of implicit function
- Piecewise approximation
- Scalar-valued 3D grid
- Easy in/out test
- Easy topology modification



Slides: Hao Li, USC

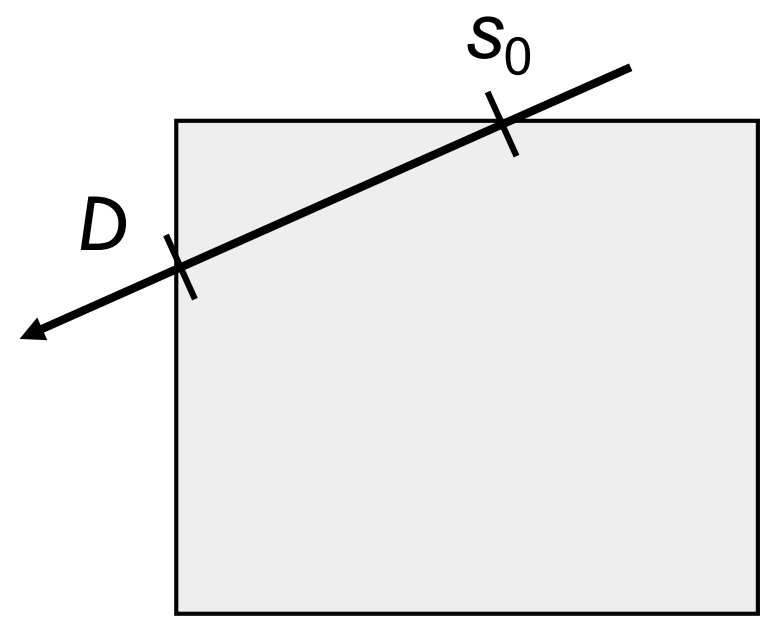
Direct Volume Rendering



Overview

- Directly get a 3D representation of the volume data
  - The data is considered to represent a semi-transparent light-emitting medium
    - Also gaseous phenomena can be simulat
  - Approaches are based on the laws of physic (emission, absorption, scattering)
  - The volume data is used as a whole (look inside, see all interior structures)

Overview



- Optical model
  - Emission and absorption of light
  - Participating media
- Volume rendering equation

$$\frac{dI(s)}{ds} = g(s) - \tau(s)I(s)$$

- Volume integral

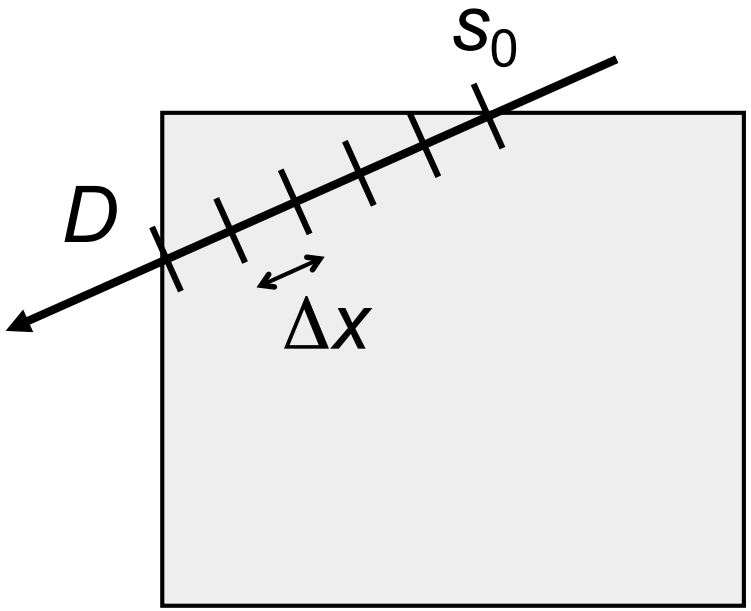
$$I(D) = I(s_0)T(s_0) + \int_{s_0}^{D} g(s)T(s)\,ds$$

- Transparency

$$T(s) = \exp\left(-\int_{s}^{D} \tau(t)\,dt \right)$$

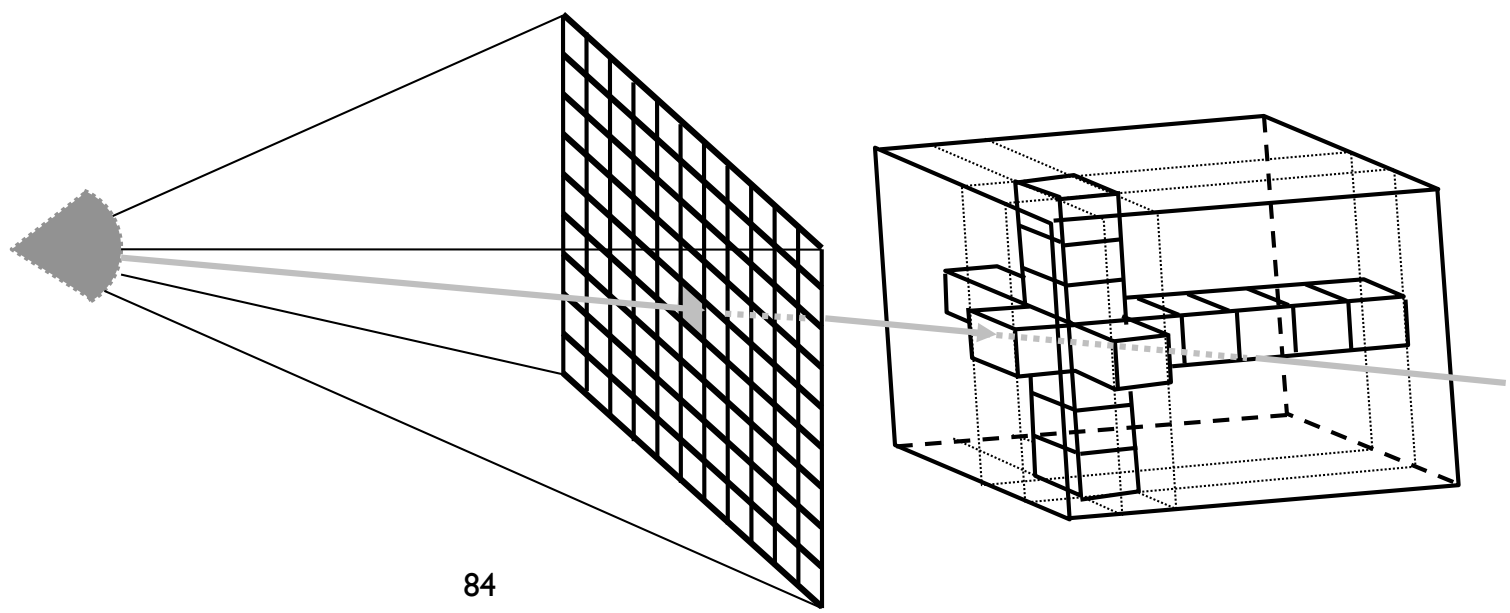
Overview

- Numerical approach to compute the volume rendering integral
  - Riemann sum
  - Sampling of light rays



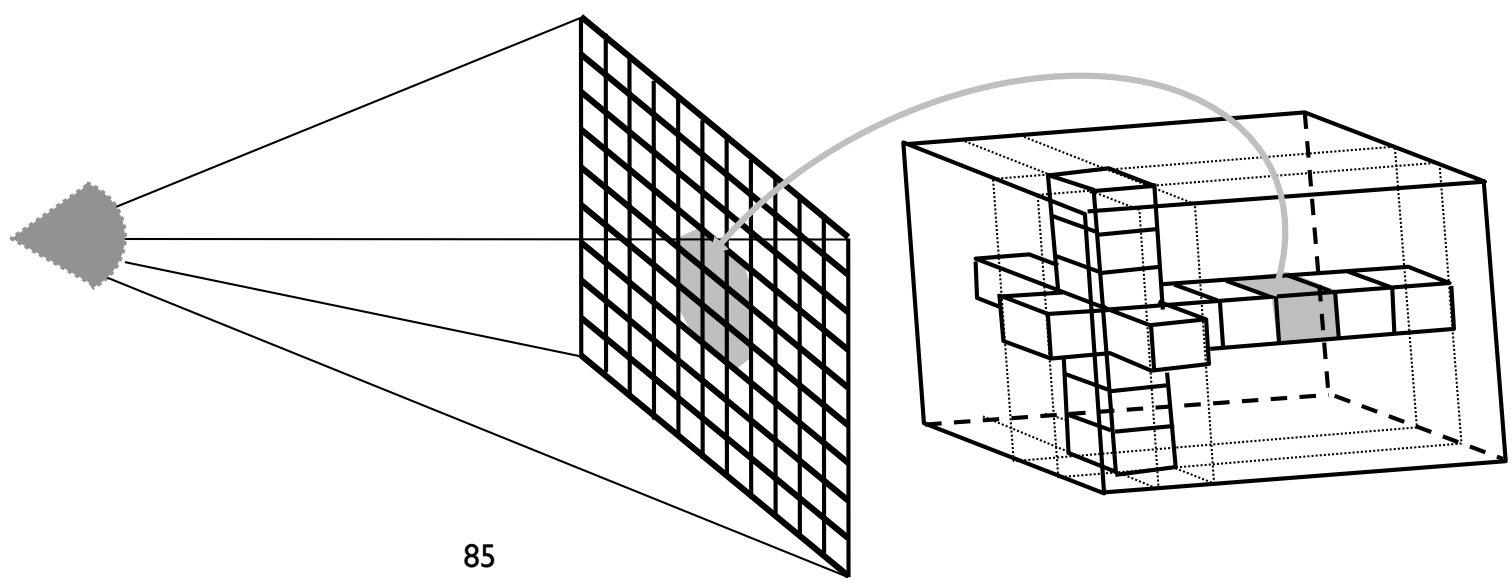
Overview

- Backward methods
  - Image space, image order algorithms
  - Performed pixel-by-pixel
  - Example: Ray casting

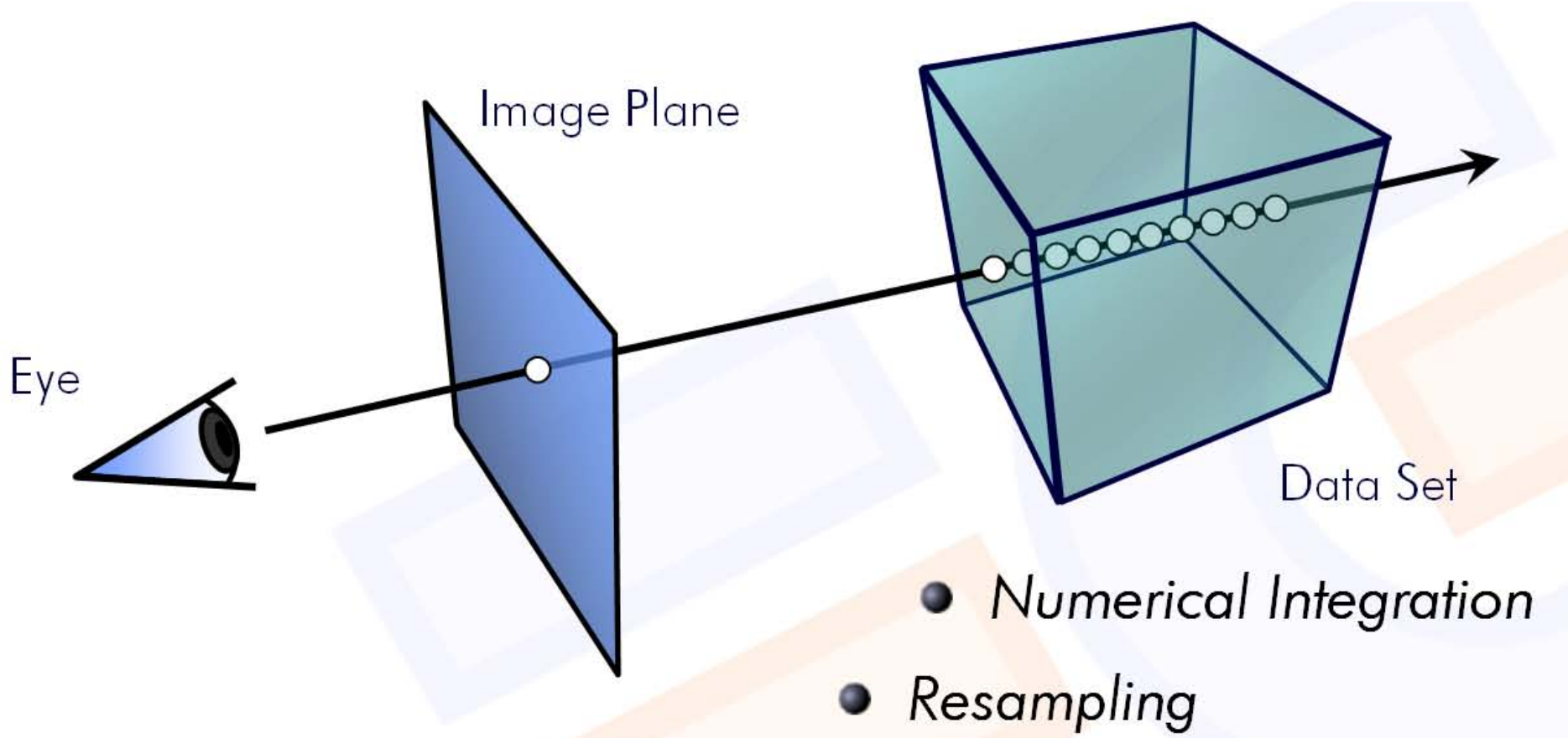


Overview

- Forward methods
  - Object-space, object-order algorithm
  - Cell projection
  - Performed voxel by voxel
  - Examples: Slicing, shear-warp, splatting



Volume ray casting



- *Numerical Integration*
- *Resampling*

Equation -Light Transport

$$I(s_k) = I(s_{k-1})e^{-\tau(s_{k-1},s_k)} + \int_{s_{k-1}}^{s_k} q(s)e^{-\tau(s,s_k)}\,ds$$

- Discretization of volume rendering equation
  - Discrete steps s_k
  - Often equidistant

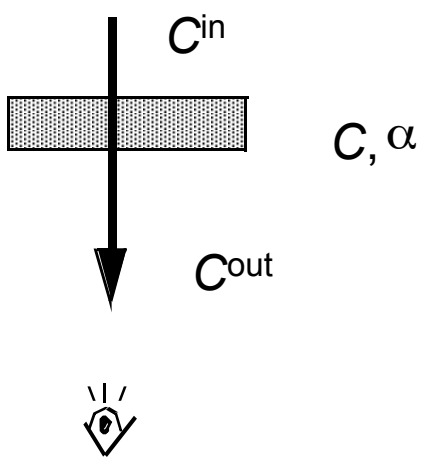
Compositing

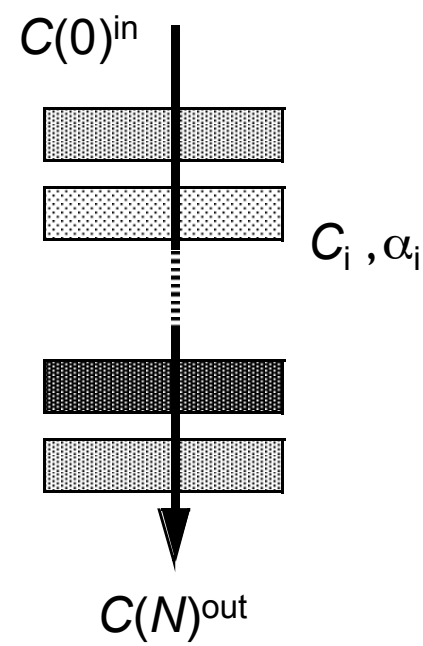
- Compositing = iterative computation of discretized volume integral
- Traversal strategies
  - Front-to-back
  - Back-to-front
- Back-to-front compositing
  - Directly derived from discretized integral $I(s_n) = I(s_{n-1}) \times (1 - \alpha_n) + b_n$
  - Just different notation: $C^{\text{out}} = C^{\text{in}} \times (1 - \alpha) + C$

Compositing

- Back-to-front compositing *(cont.)*
  - Over operator [Porter & Duff 1984]
  - Used, e.g., in texture-based volume rendering
- Compositing equation:

$$C^{out} = (1 - \alpha)\ C^{in} + C \quad C(i)^{in} = C(i-1)^{out}$$

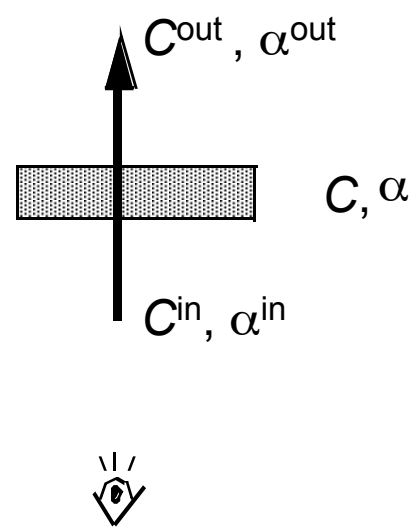




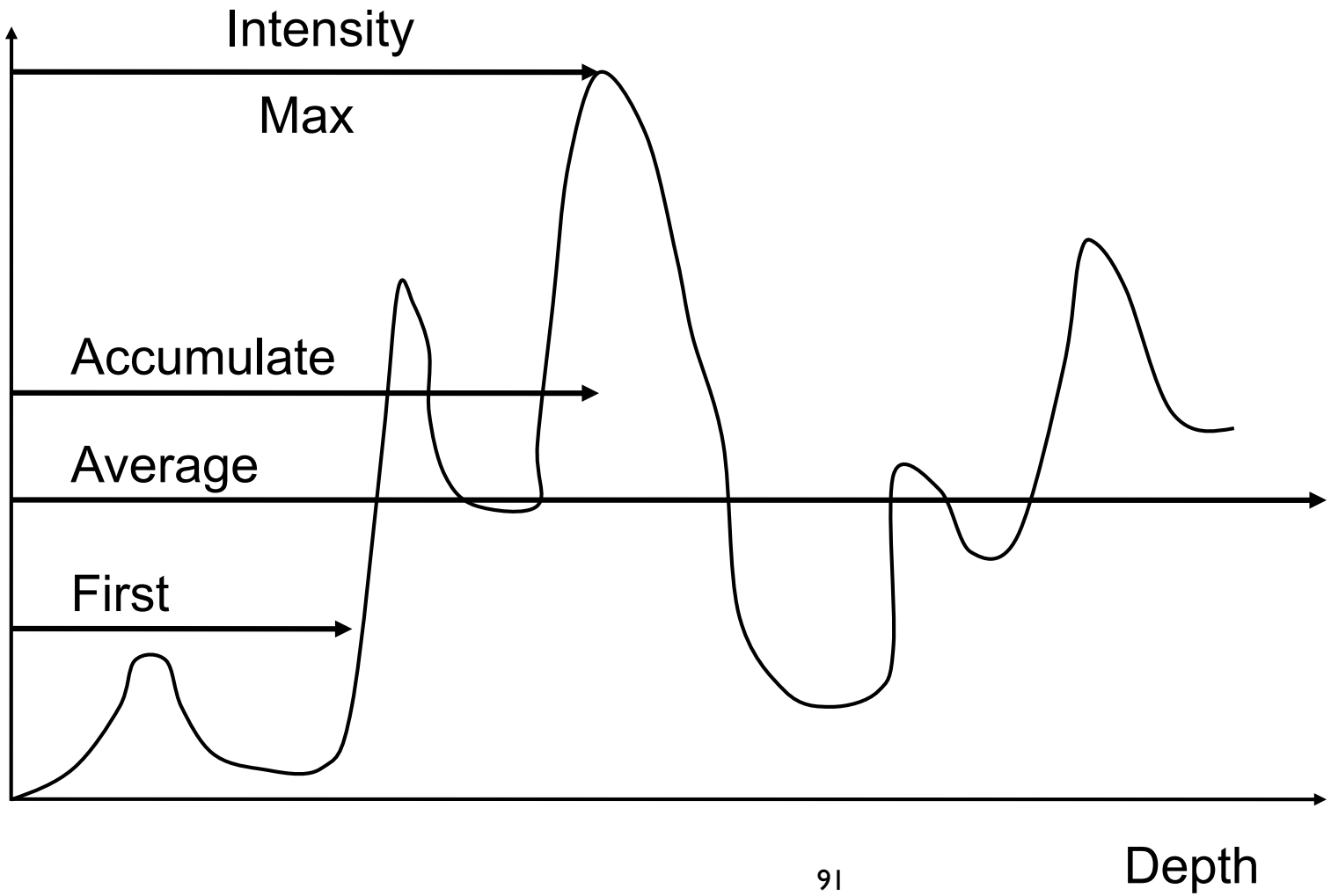
Compositing

- Front-to-back compositing *(cont.)*
  - Needs to maintain α^{in}
  - Most often used in ray casting
- Compositing equation:

$$C^{out} = C^{in} + (1 - \alpha^{in})\, C$$
$$\alpha^{out} = \alpha^{in} + (1 - \alpha^{in})\, \alpha$$

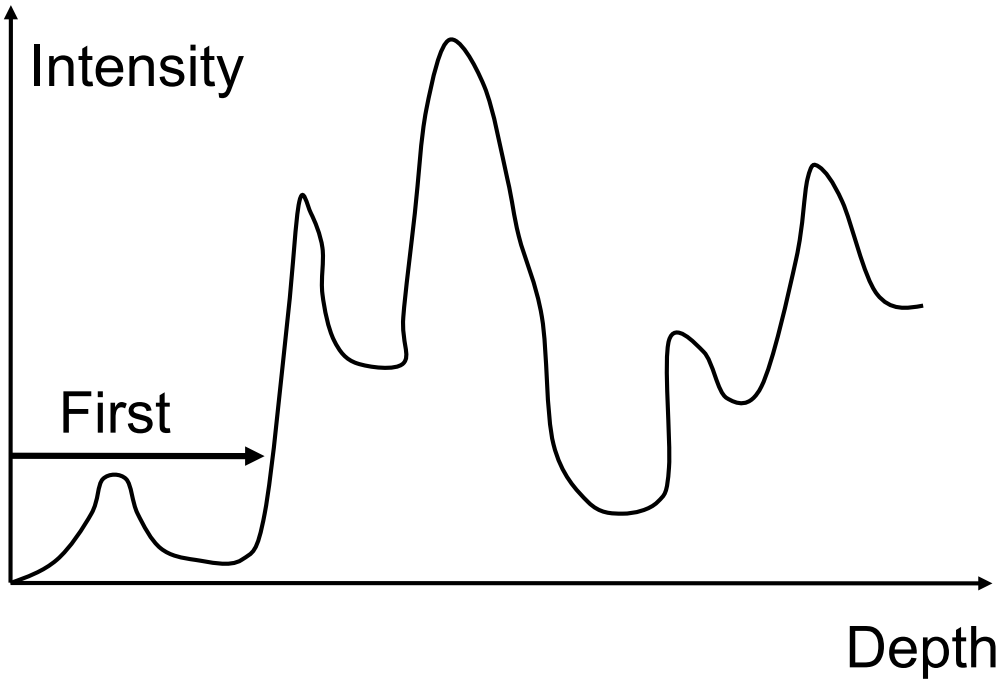


Compositing



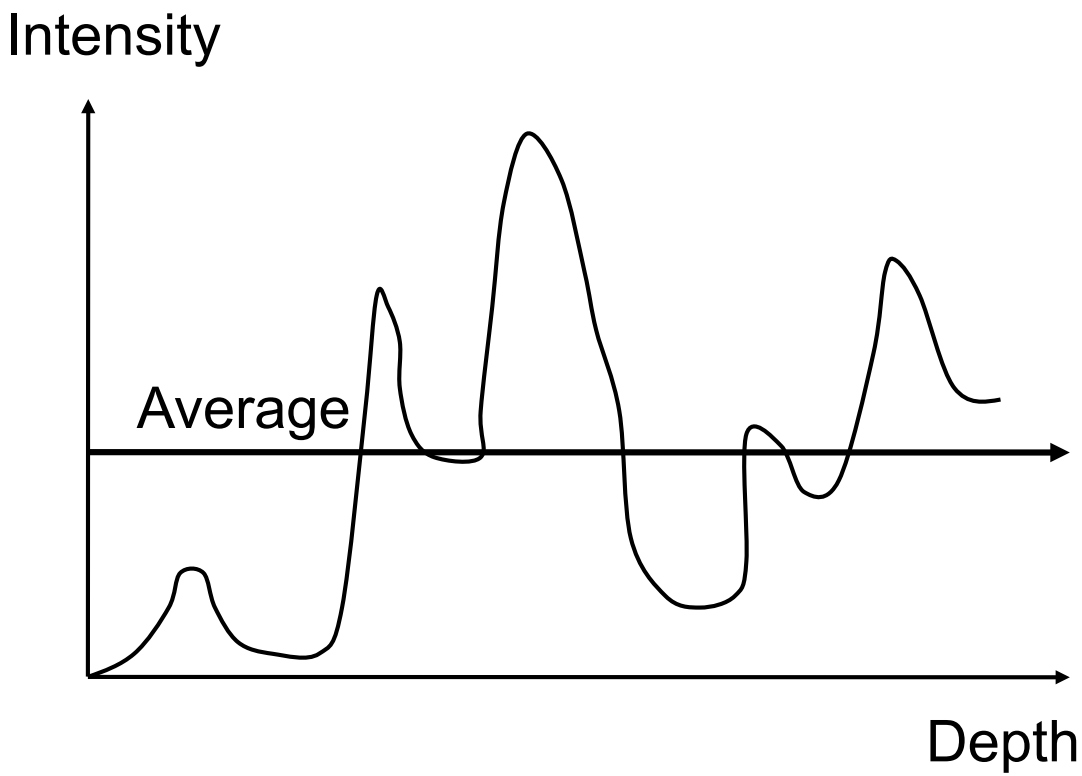
Compositing

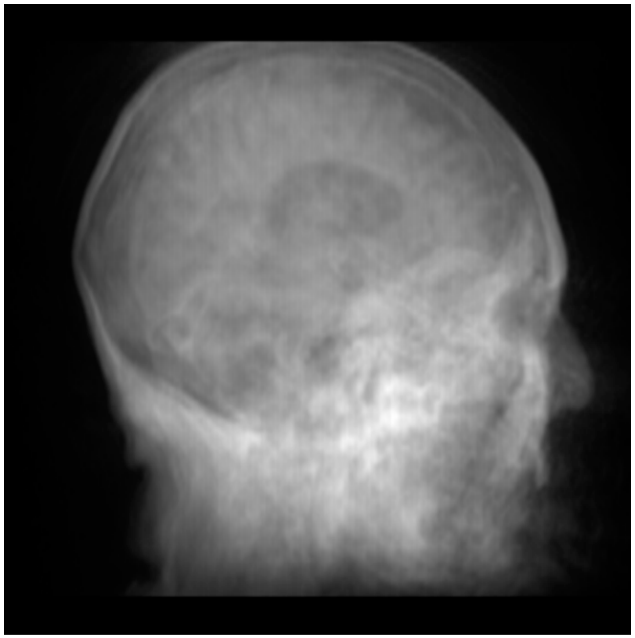
- Compositing: First
- Extracts isosurfaces



Compositing

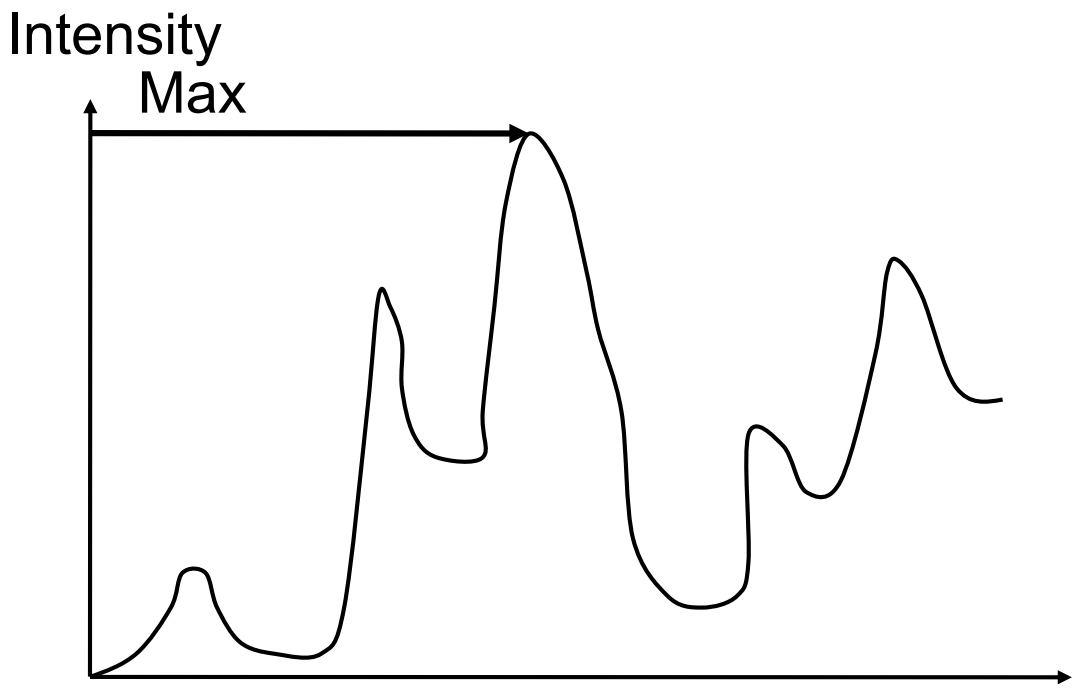
- Compositing: Average
- Produces basically an X-ray picture

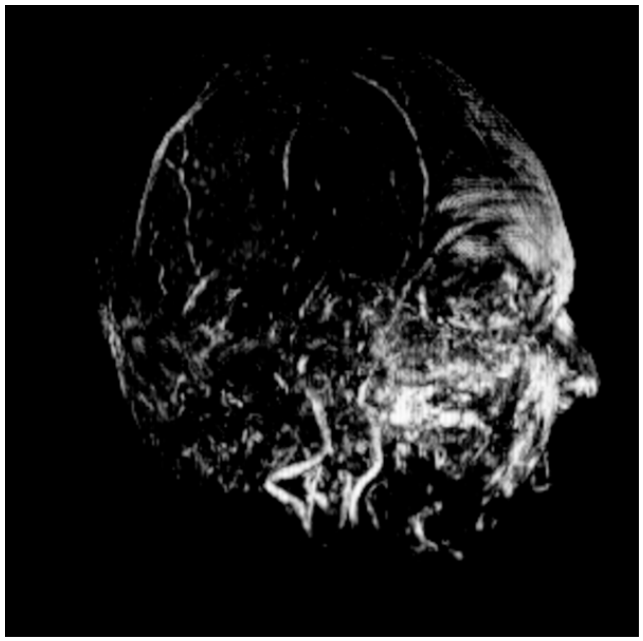




Compositing

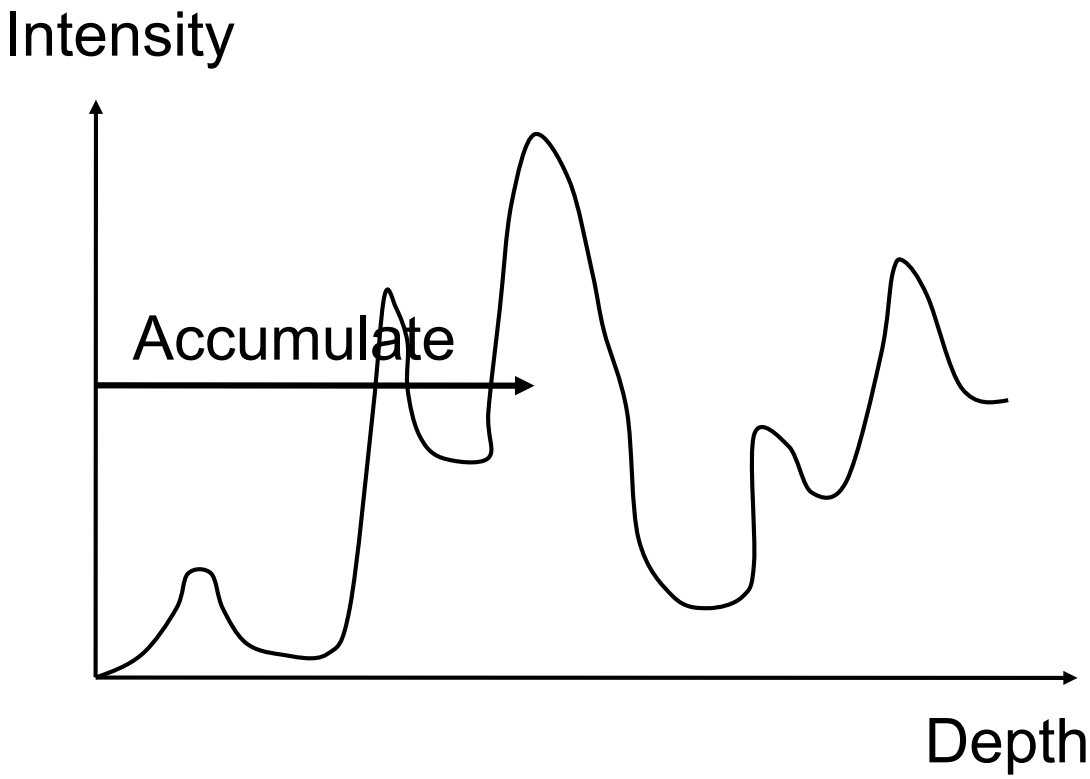
- Maximum Intensity Projection (MIP)
- Often used for MR or CT angiograms
- Good to extract vessel structures

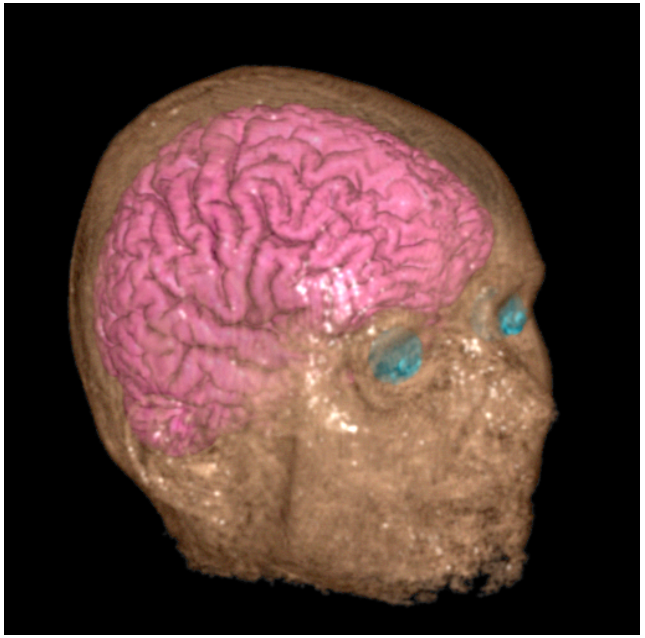




Compositing

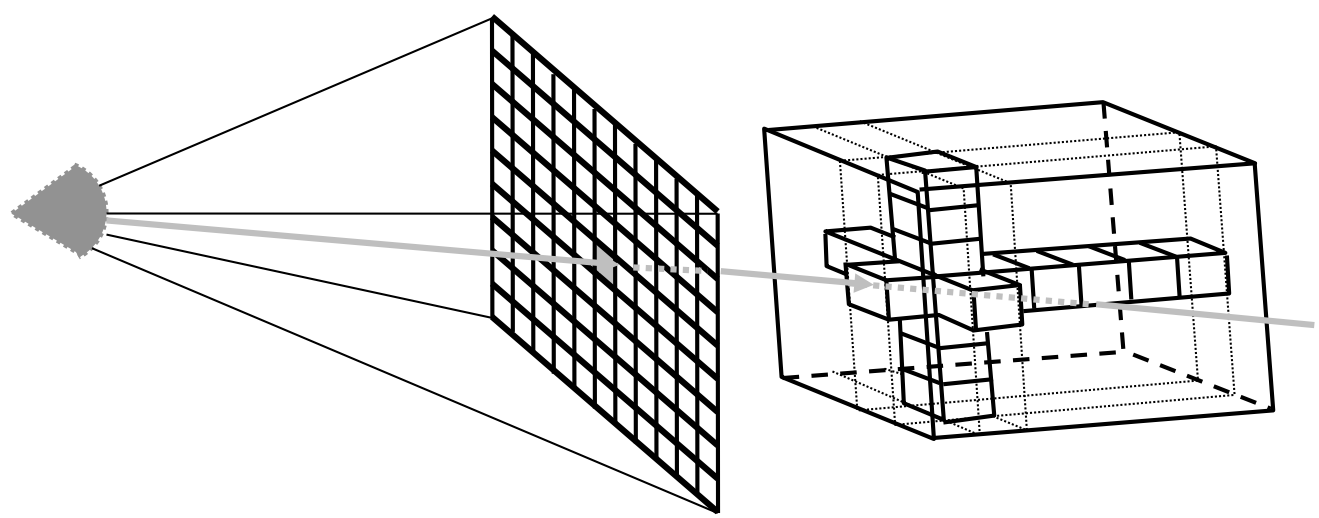
- Compositing: Accumulate
- Emission-absorption model
- Make transparent layers visible (see classif.)





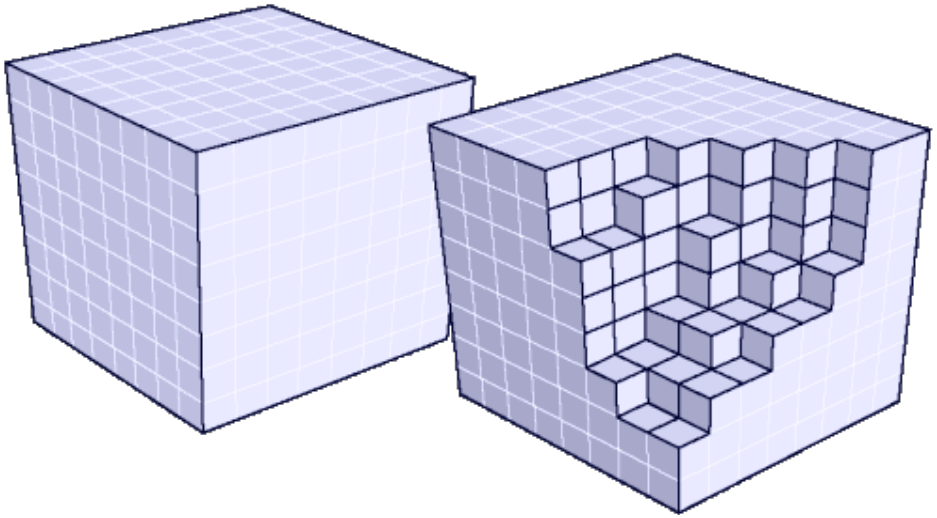
Ray Casting

- Similar to ray tracing in surface-based computer graphics
- In volume rendering we only deal with primary rays; hence: ray casting
- Natural image-order technique
- As opposed to surface graphics - how do we calculate the ray/surface intersection?



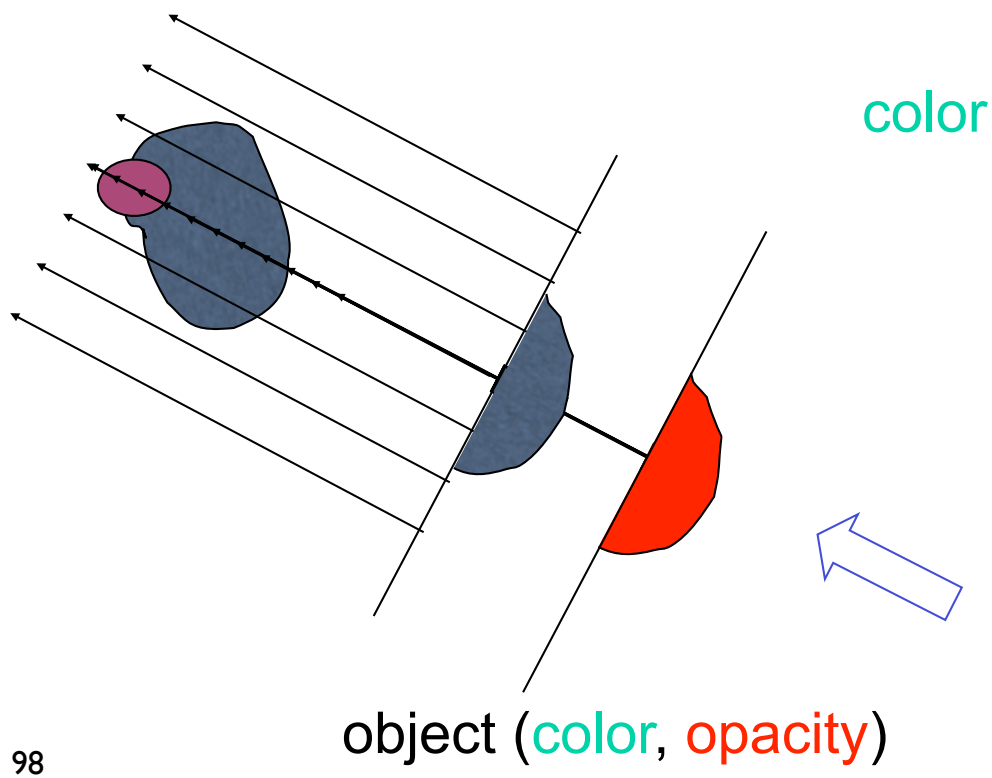
Ray Casting

- Since we have no surfaces - need to carefully step through volume
- A ray is cast into the volume, sampling the volume at certain intervals
- Sampling intervals are usually equidistant, but don't have to be (e.g. importance sampling)
- At each sampling location, a sample is interpolated / reconstructed from the voxel grid
- Popular filters are: nearest neighbor (box), trilinear, or more sophisticated (Gaussian, cubic spline)
- First: Ray casting in uniform grids
  - Implicit topology
  - Simple interpolation schemes

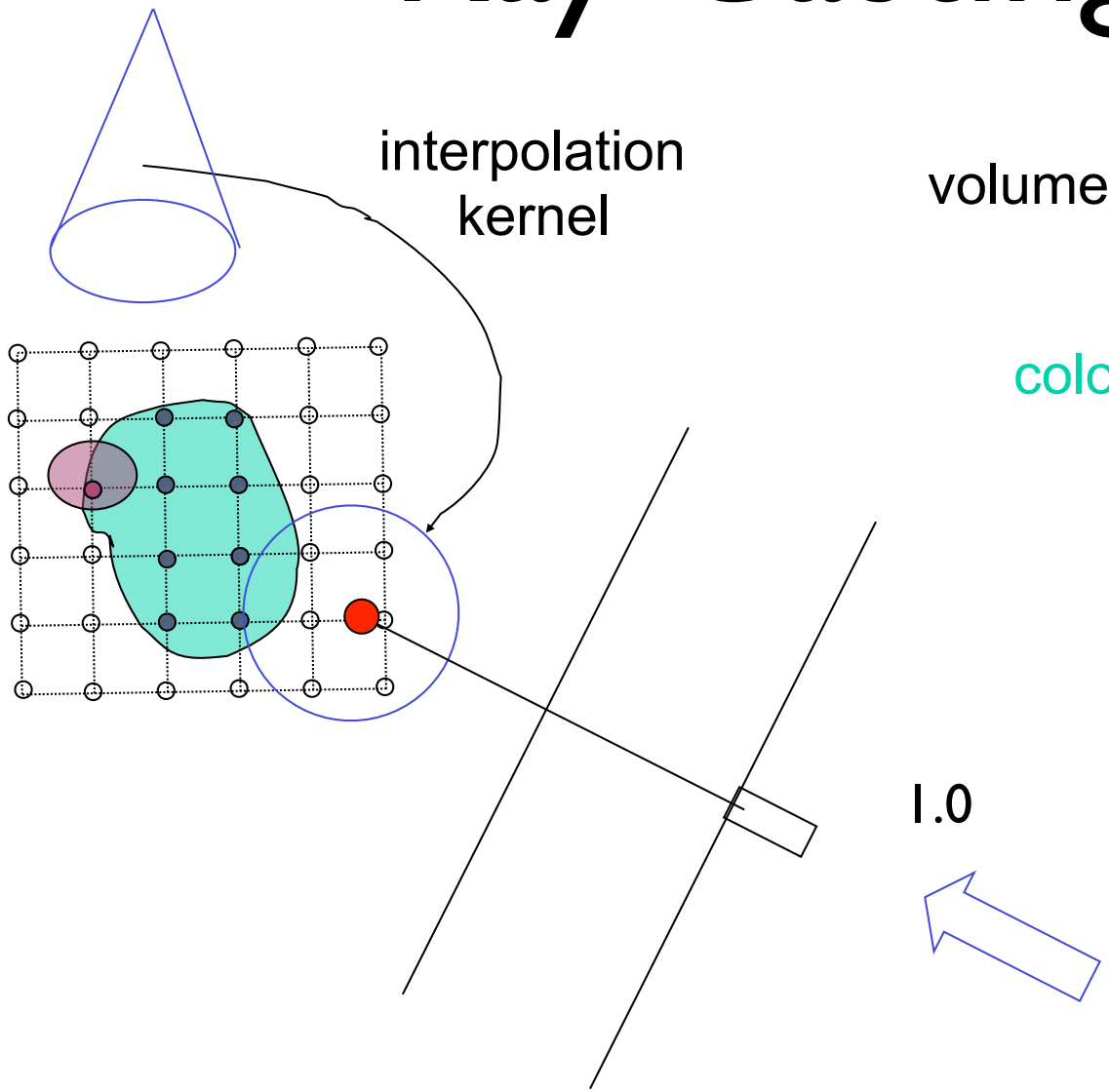


Ray Casting

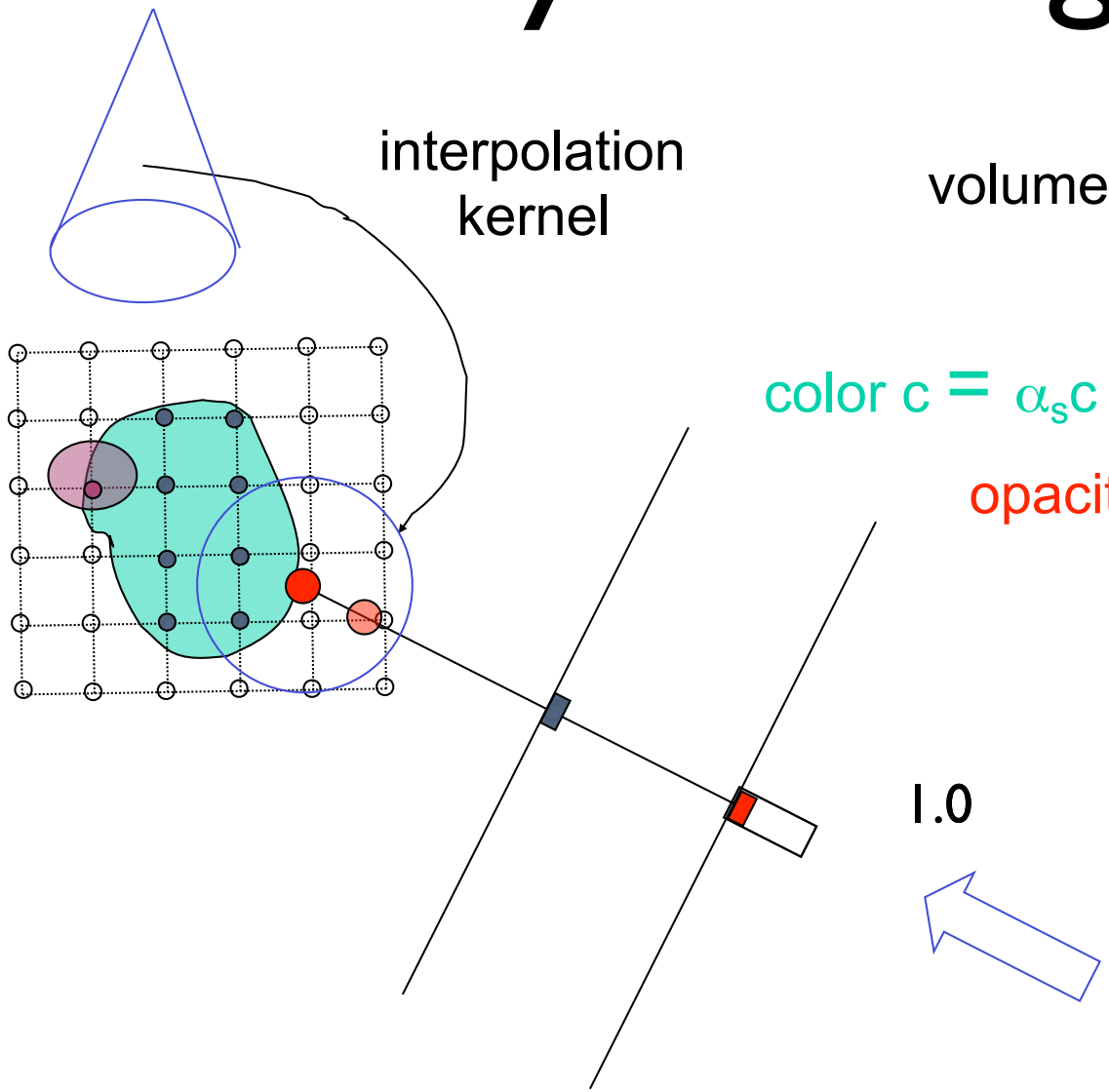
- Volumetric ray integration:
  - Tracing of rays
  - Accumulation of color and opacity along ray: compositing



Ray Casting


interpolation
kernel
volumetric compositing
color
opacity
1.0
object (color, opacity)

Ray Casting



volumetric compositing

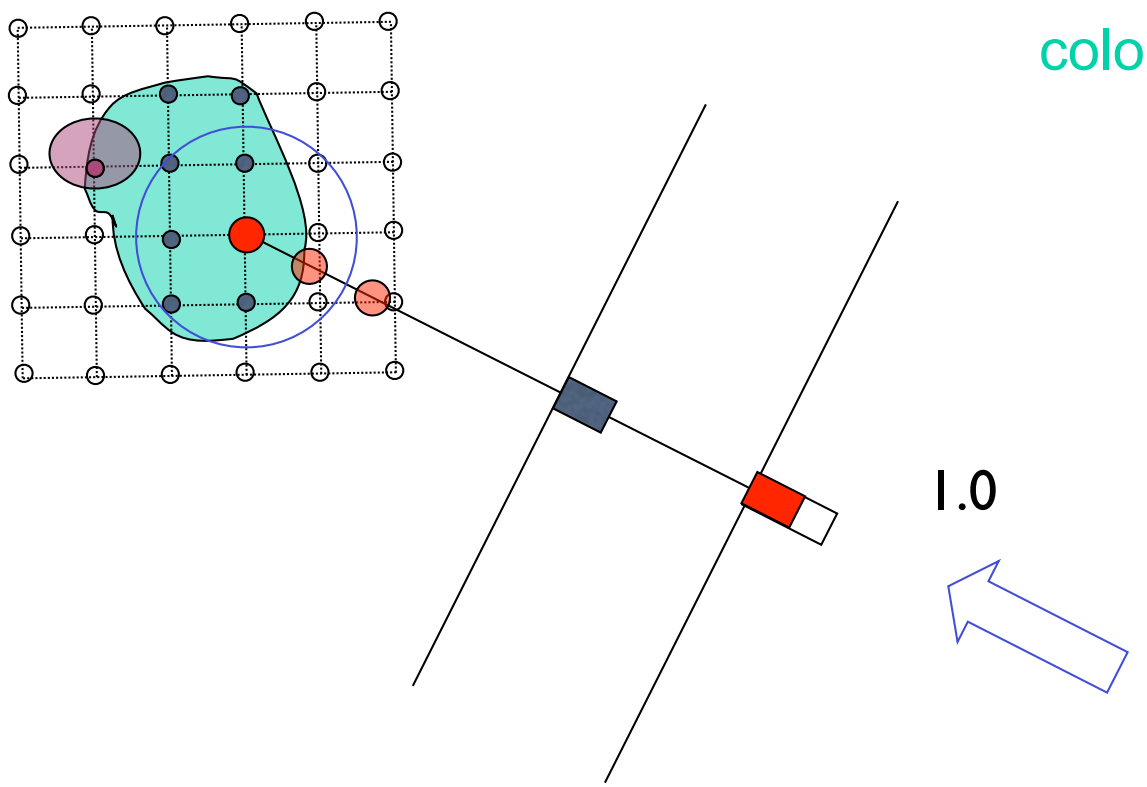
color $c = \alpha_s c_s (1 - \alpha) + c$

opacity $\alpha = \alpha_s (1 - \alpha) + \alpha$

object (color, opacity)

Ray Casting

volumetric compositing



color

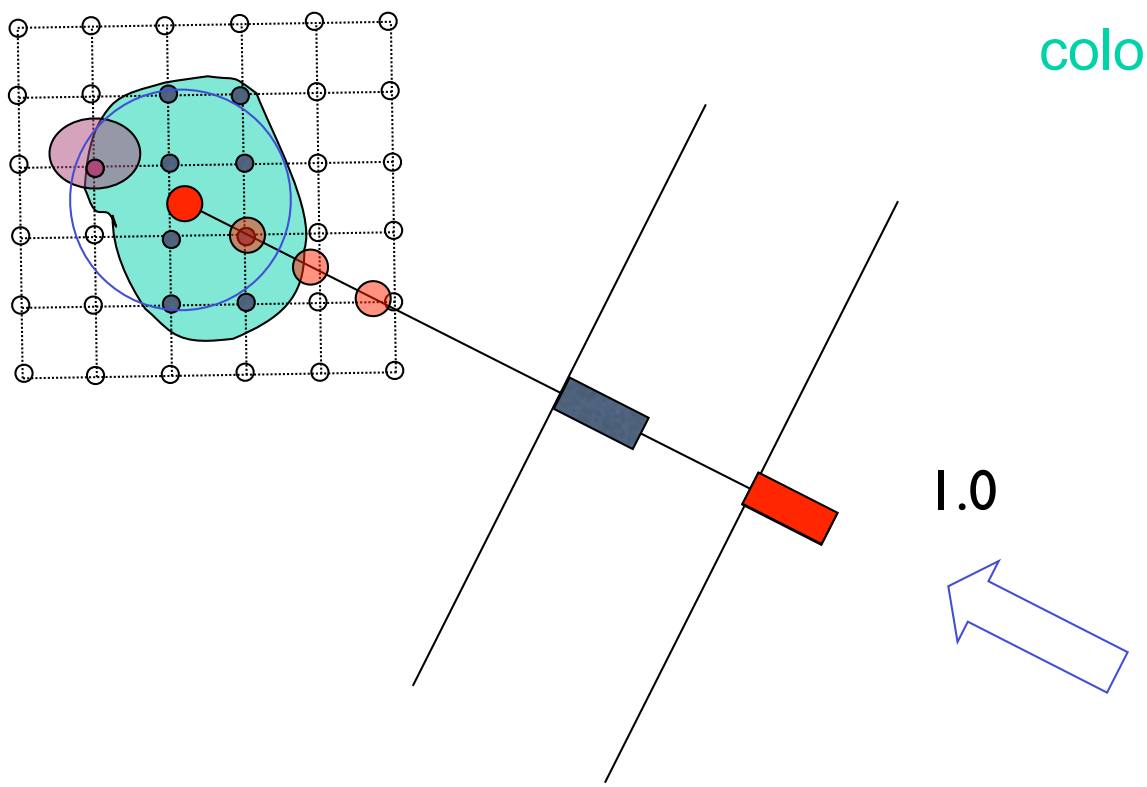
opacity

1.0

object (color, opacity)

Ray Casting

volumetric compositing



color

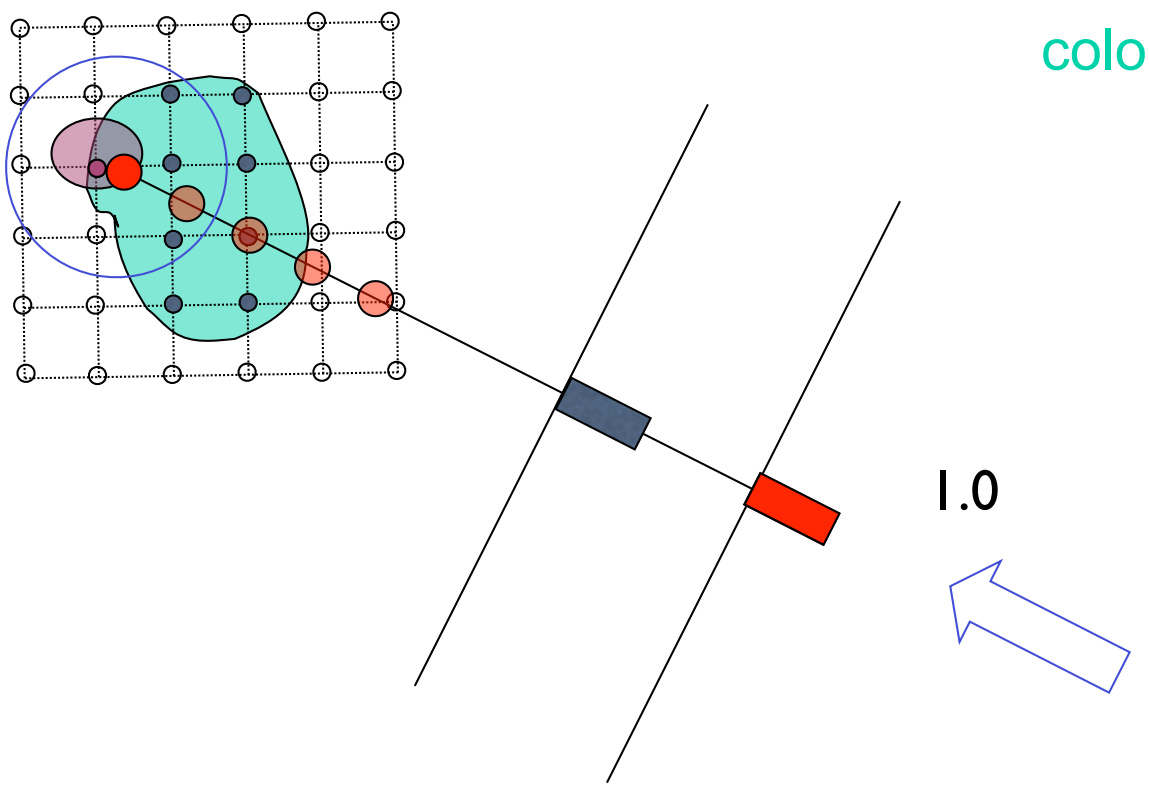
opacity

1.0

object (color, opacity)

Ray Casting

volumetric compositing



color

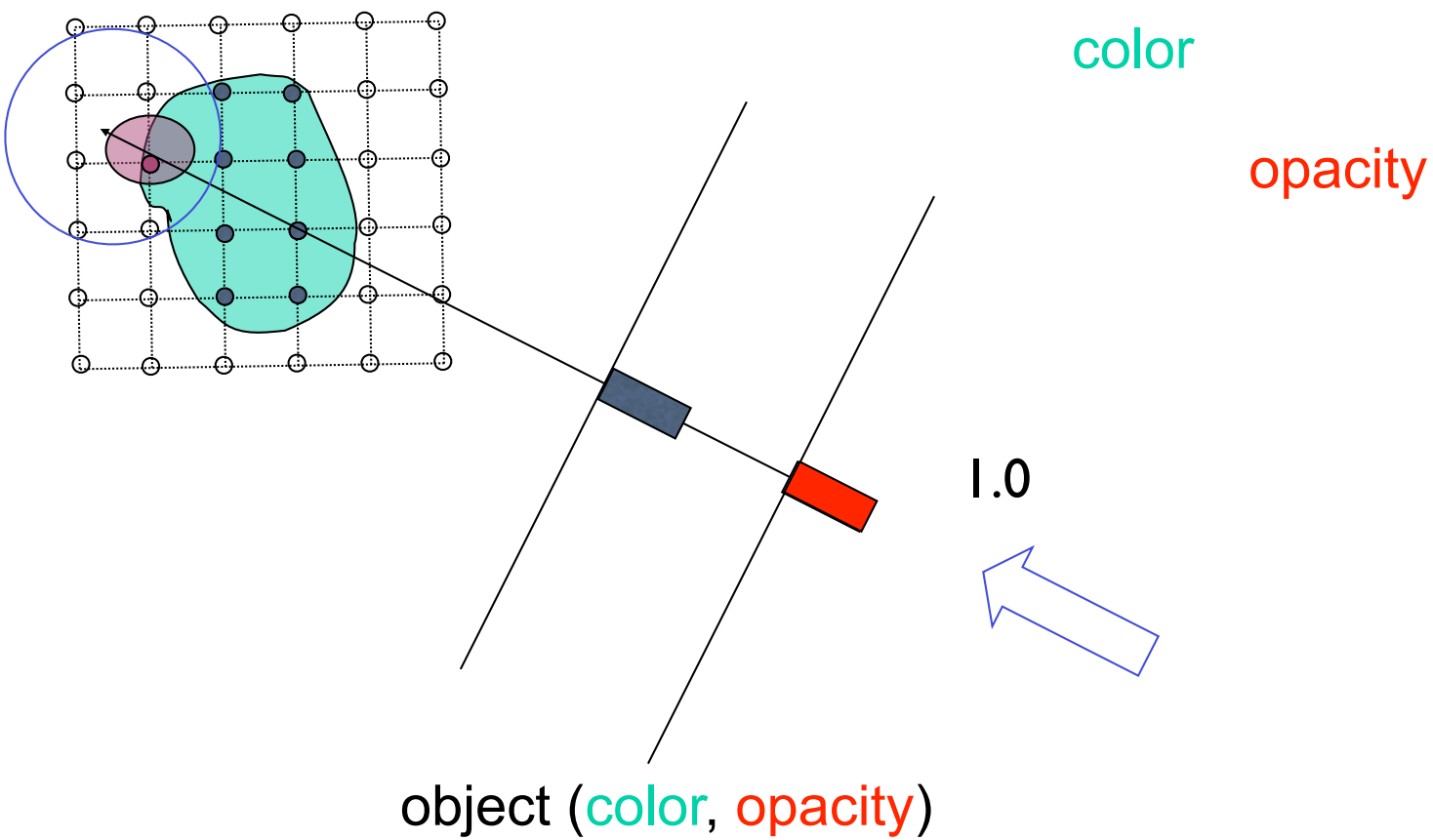
opacity

1.0

object (color, opacity)

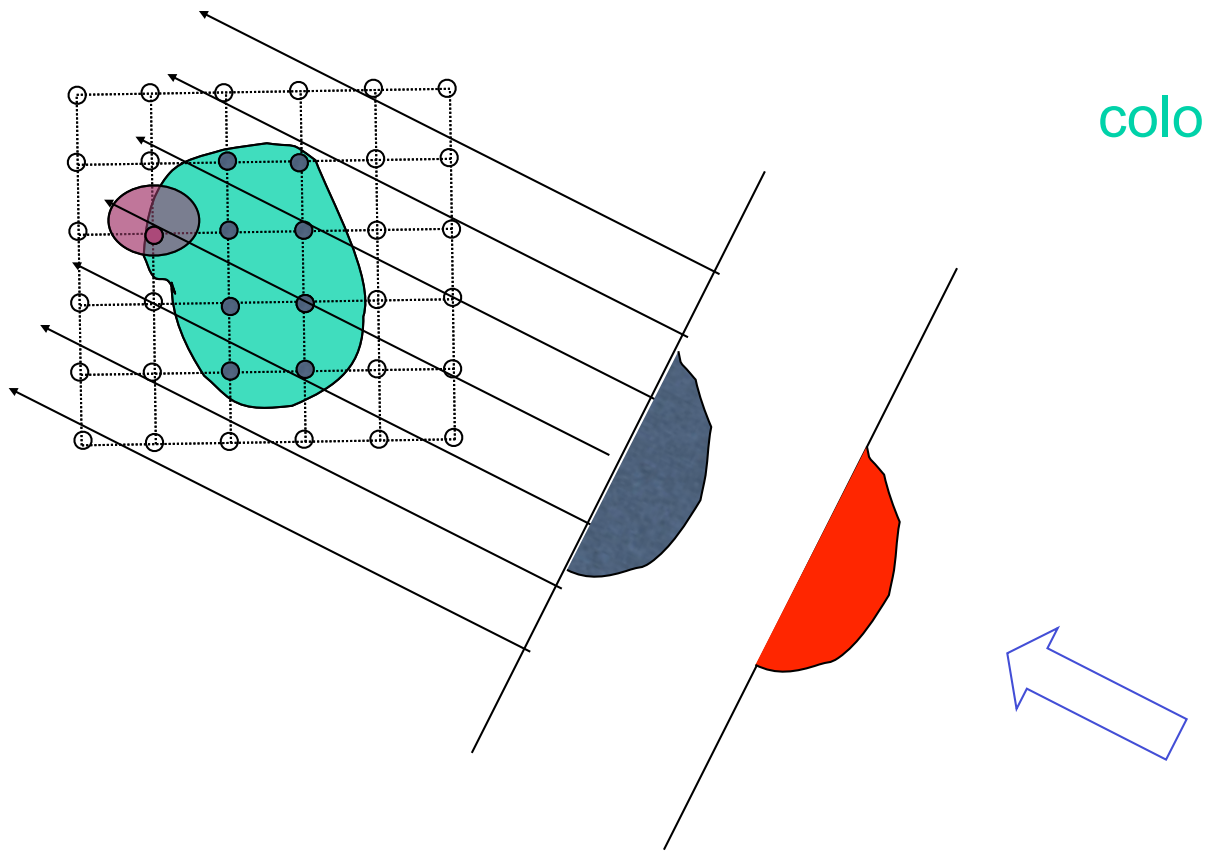
Ray Casting

volumetric compositing



Ray Casting

volumetric compositing



color

opacity

object (color, opacity)

Volume rendering recap

- *Emission-absorption* model: Emission **C** (RGB), Absorption **A** (scalar)
- Continuous volume rendering equation, e.g. Sabella 1988, Kniss 2003

$$I(a,b) = \int_a^b C\rho(v(u))\, e^{-\int_a^u \tau\rho(v(t))dt} du$$

Sabella, P. A rendering algorithm for visualizing 3D scalar fields. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, vol. 22, 51–58.
Kniss, J, Premoze, S, Ikits, M, Lefohn, A, Hansen, C, Praun, E. Gaussian transfer functions for Multi-field Volume Visualization. Proc. IEEE Vis 2003, 497—504.

- *p(v(u))* means classify(interpolate(position)), i.e. a **post-classified transfer function**

- Discrete (Riemann sum) volume rendering

$$\tilde{C} = \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i \prod_{j=0}^{i-1} (1 - A_j)$$

- Blending operation (front-to-back)

$$\begin{aligned} C'_i &= C'_{i+1} + (1 - A'_{i+1}) C_i \\ A'_i &= A'_{i+1} + (1 - A'_{i+1}) A_i \end{aligned}$$

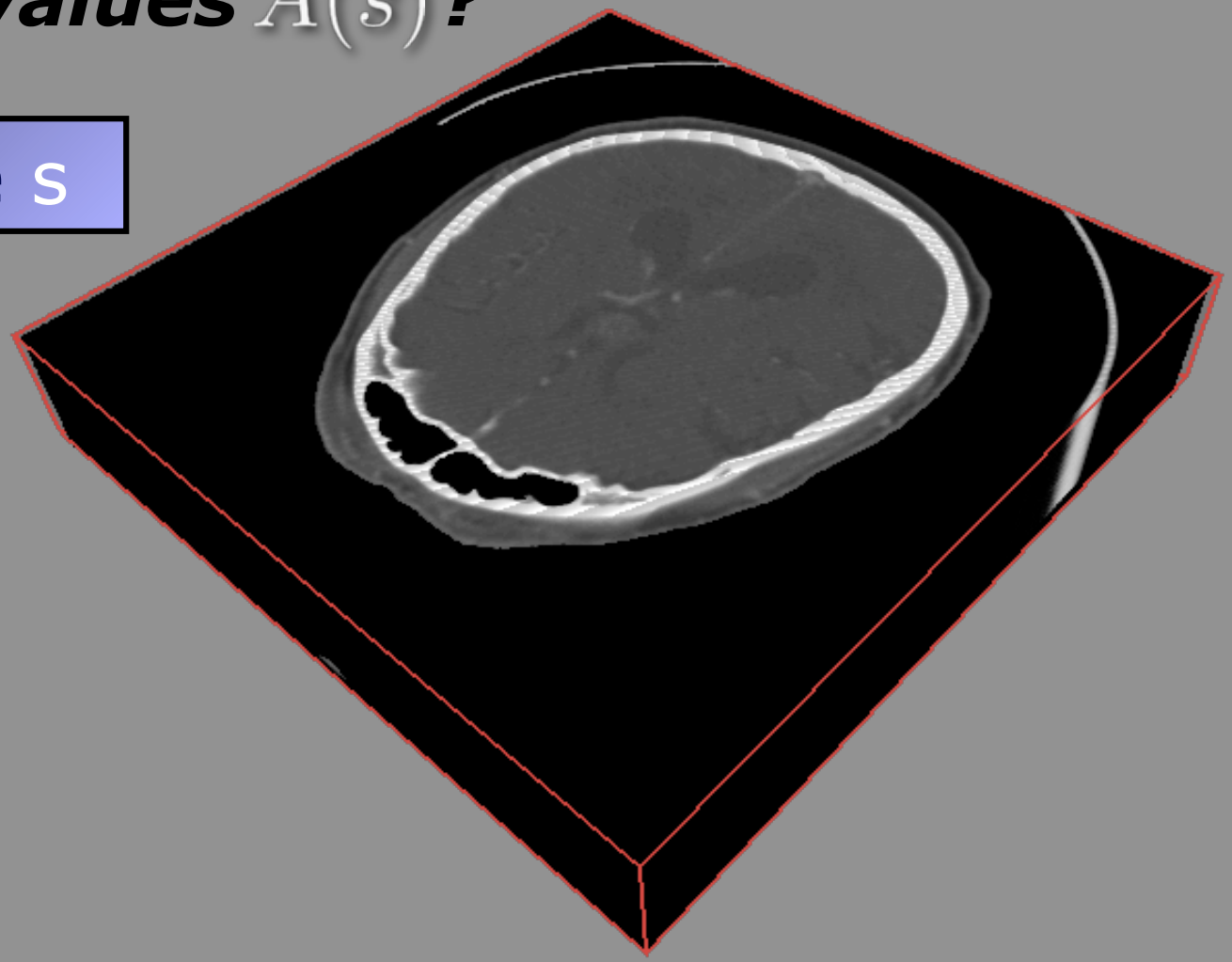
Ray Casting

- How is color and opacity at each integration step determined?
- Opacity and (emissive) color in each cell according to classification
- Additional color due to external lighting:
  according to volumetric shading (e.g. Blinn-Phong)
- No shadowing, no secondary effects
- Implementations
  - Traditional CPU implementation
  - straightforward, very efficient GPU implemenations
    - Fragment shader loops (Shader Model 3 GPUs)

Classification

How do I obtain the emission values $q(s)$ (C_i) ***and Absorption values*** $A(s)$***?***

scalar value s



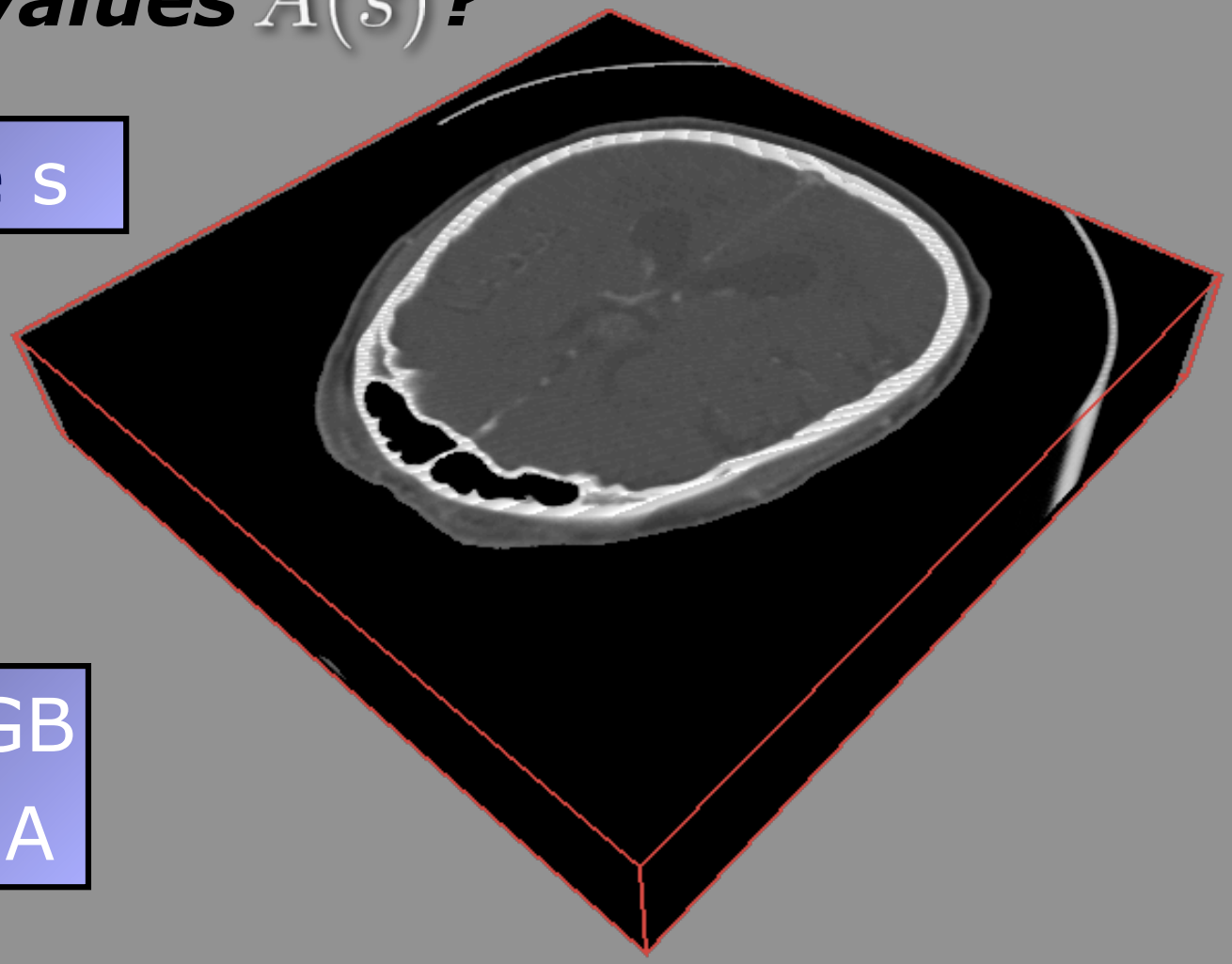
Classification

How do I obtain the emission values $q(s)$ (C_i) ***and Absorption values*** $A(s)$***?***

scalar value s

T(s)

emission RGB
absorption A



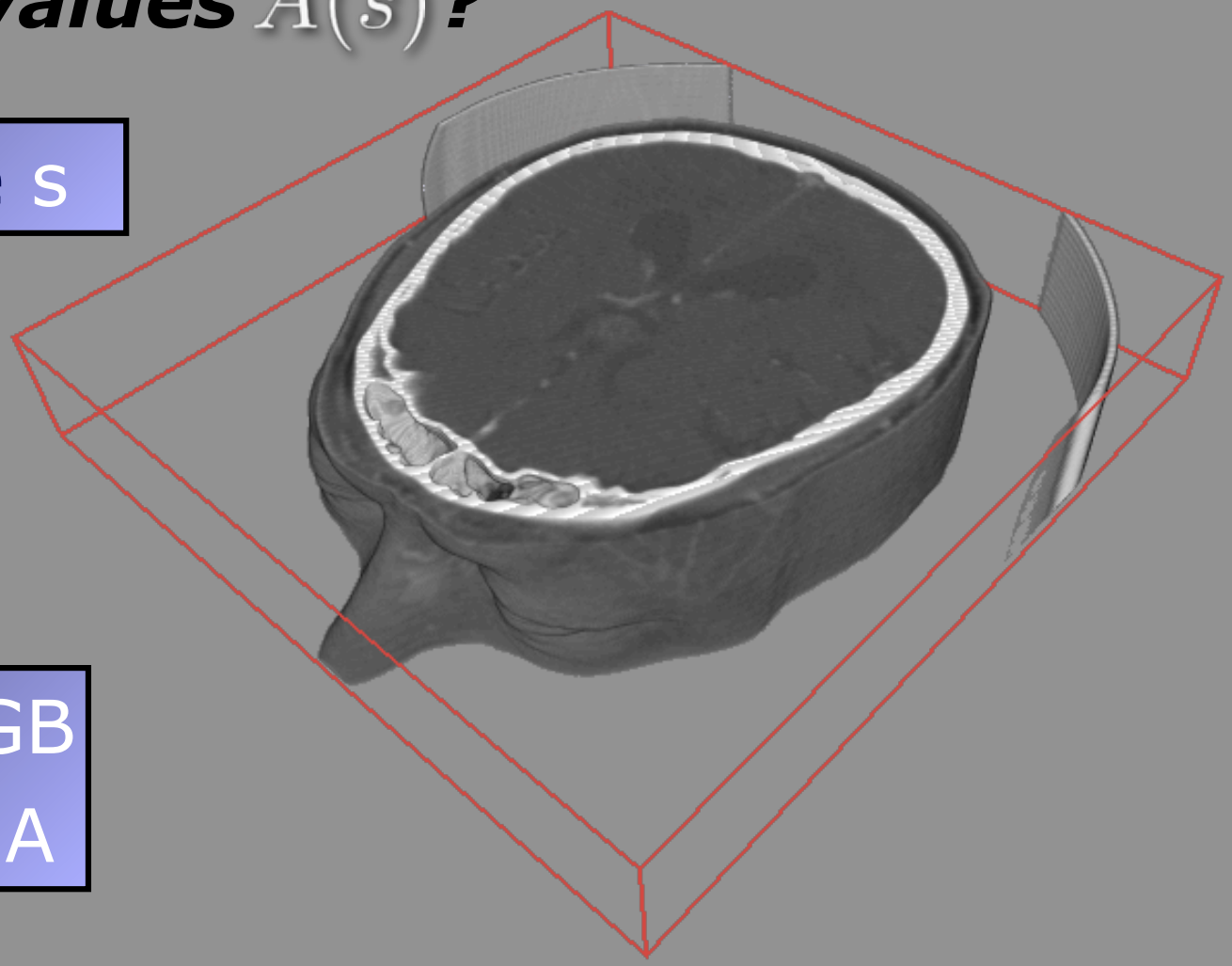
Classification

How do I obtain the emission values $q(s)$ (C_i) ***and Absorption values*** $A(s)$***?***

scalar value s

T(s)

emission RGB
absorption A



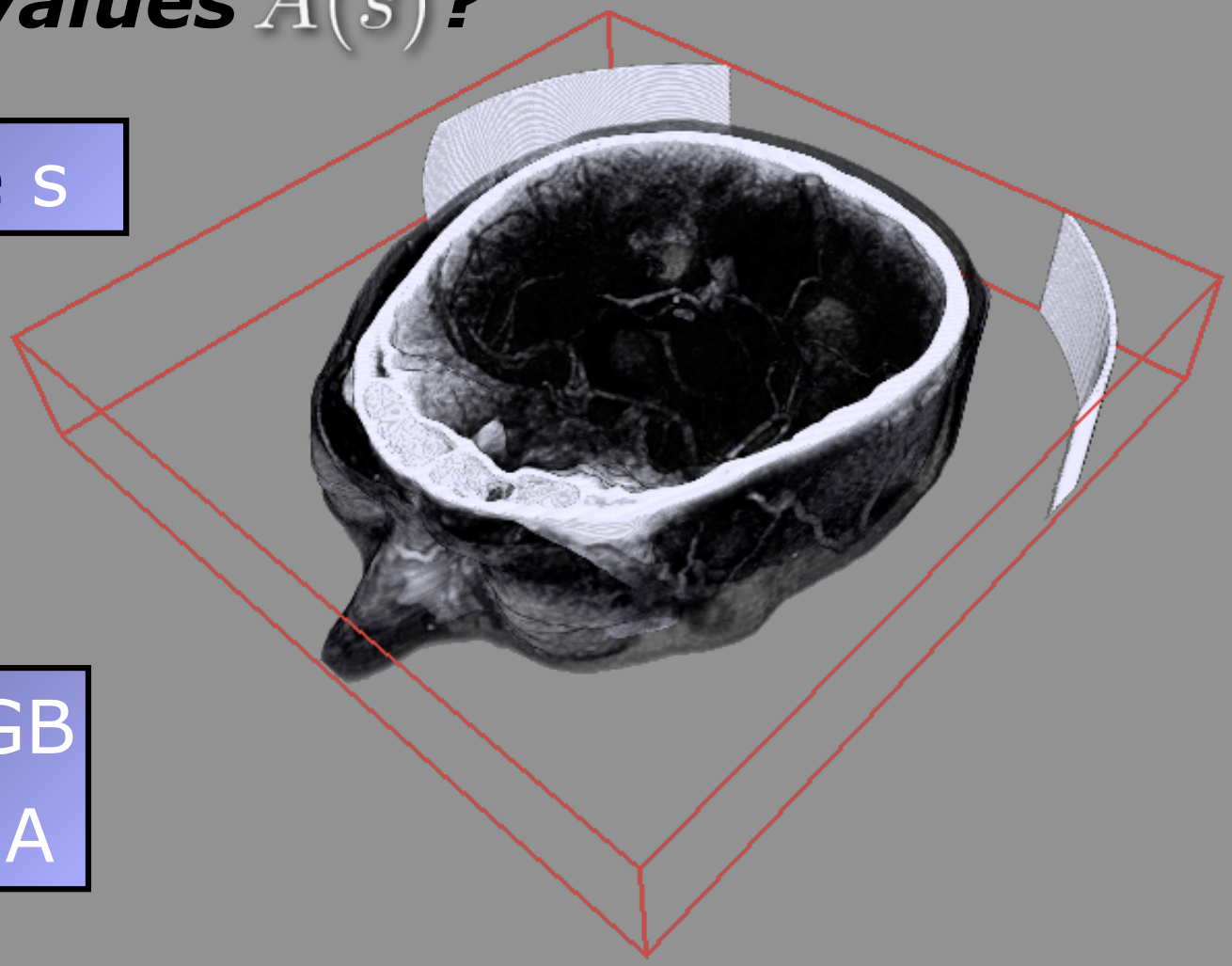
Classification

How do I obtain the emission values $q(s)$ (Ci) ***and Absorption values*** $A(s)$***?***

scalar value s

T(s)

emission RGB
absorption A



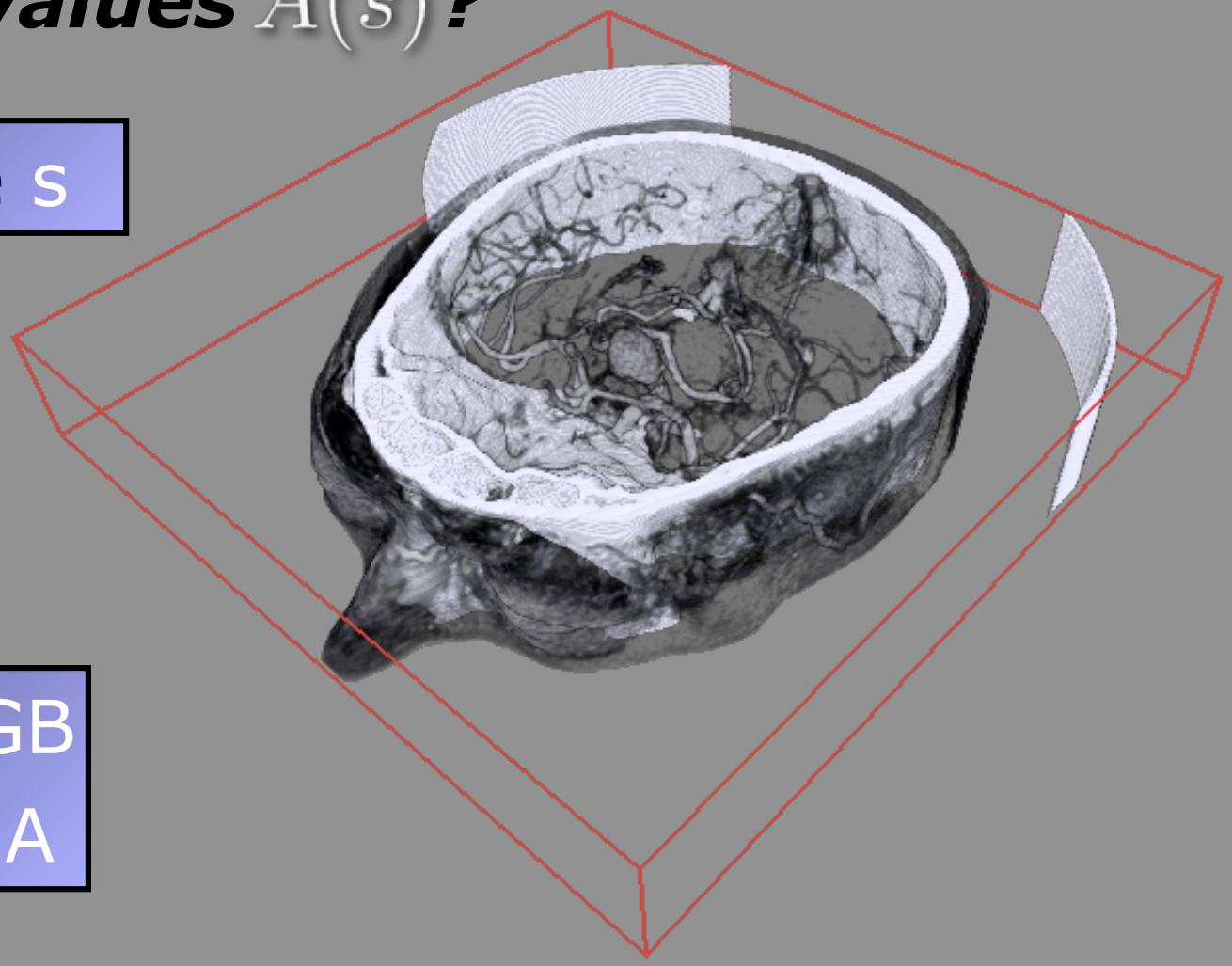
Classification

How do I obtain the emission values $q(s)$ (C_i) ***and Absorption values*** $A(s)$***?***

scalar value s

T(s)

emission RGB
absorption A



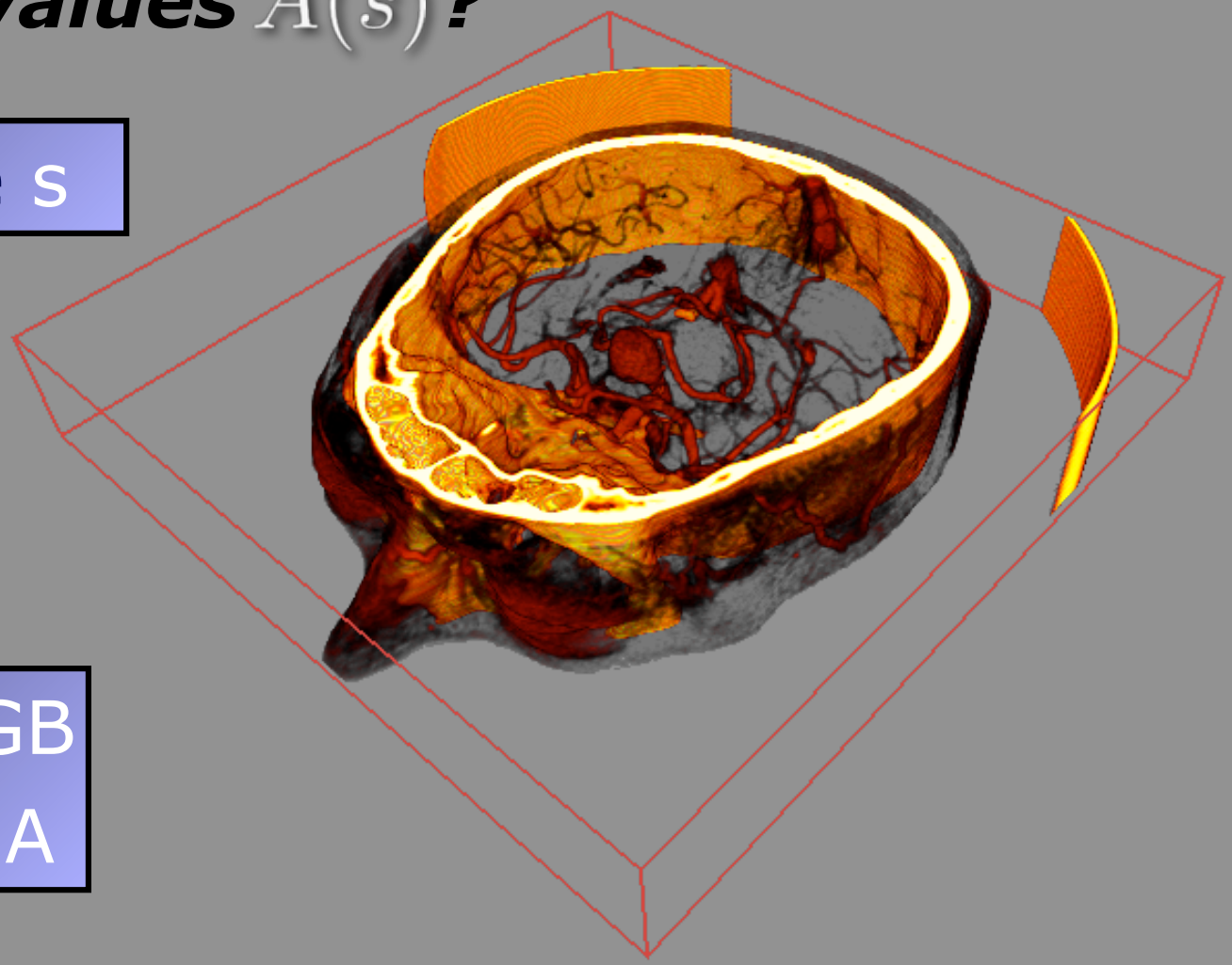
Classification

How do I obtain the emission values $q(s)$ (C_i) ***and Absorption values*** $A(s)$***?***

scalar value s

T(s)

emission RGB
absorption A



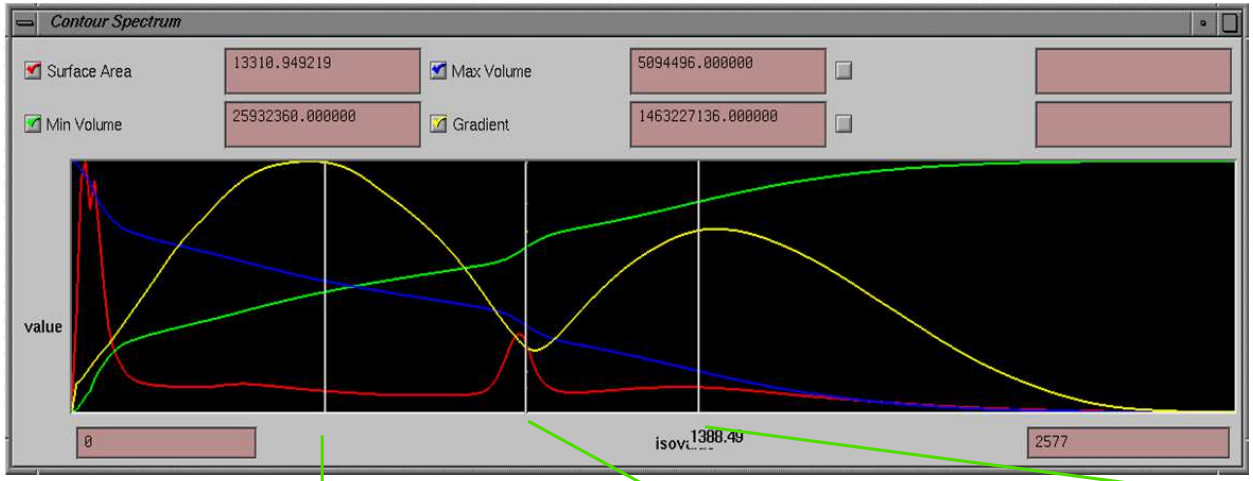
1. Trial and Error

1. Manually edit graph of transfer function
2. Enforces learning by experience
3. Get better with practice
4. Can make terrific images

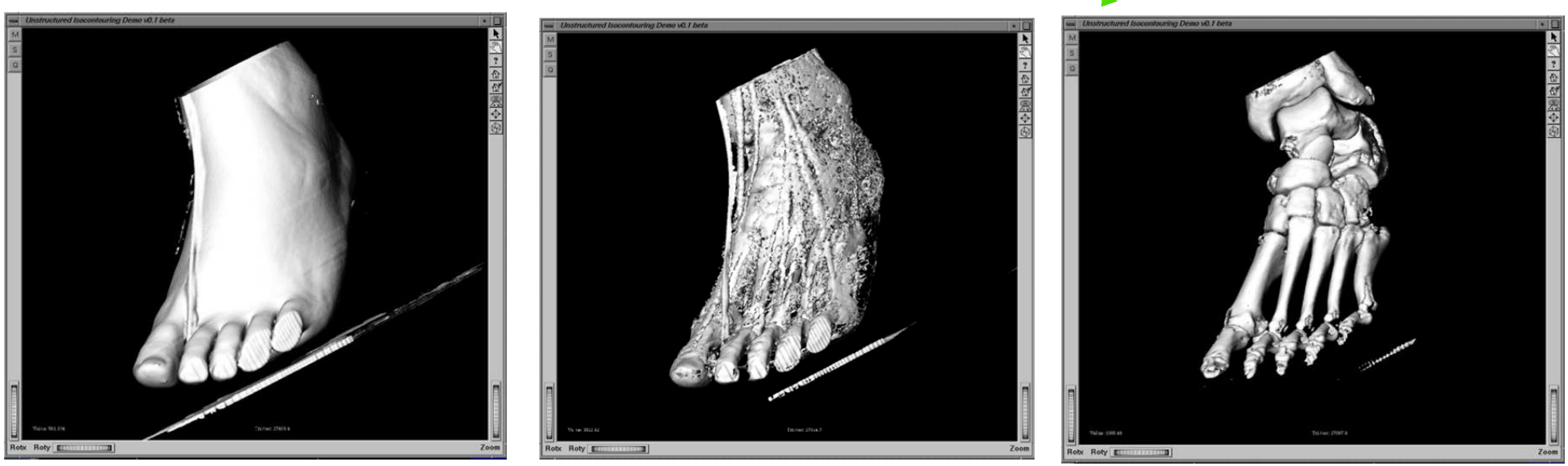
William Schroeder, Lisa Sobierajski Avila, and Ken Martin; Transfer Function Bake-off Vis '00

The Contour Spectrum

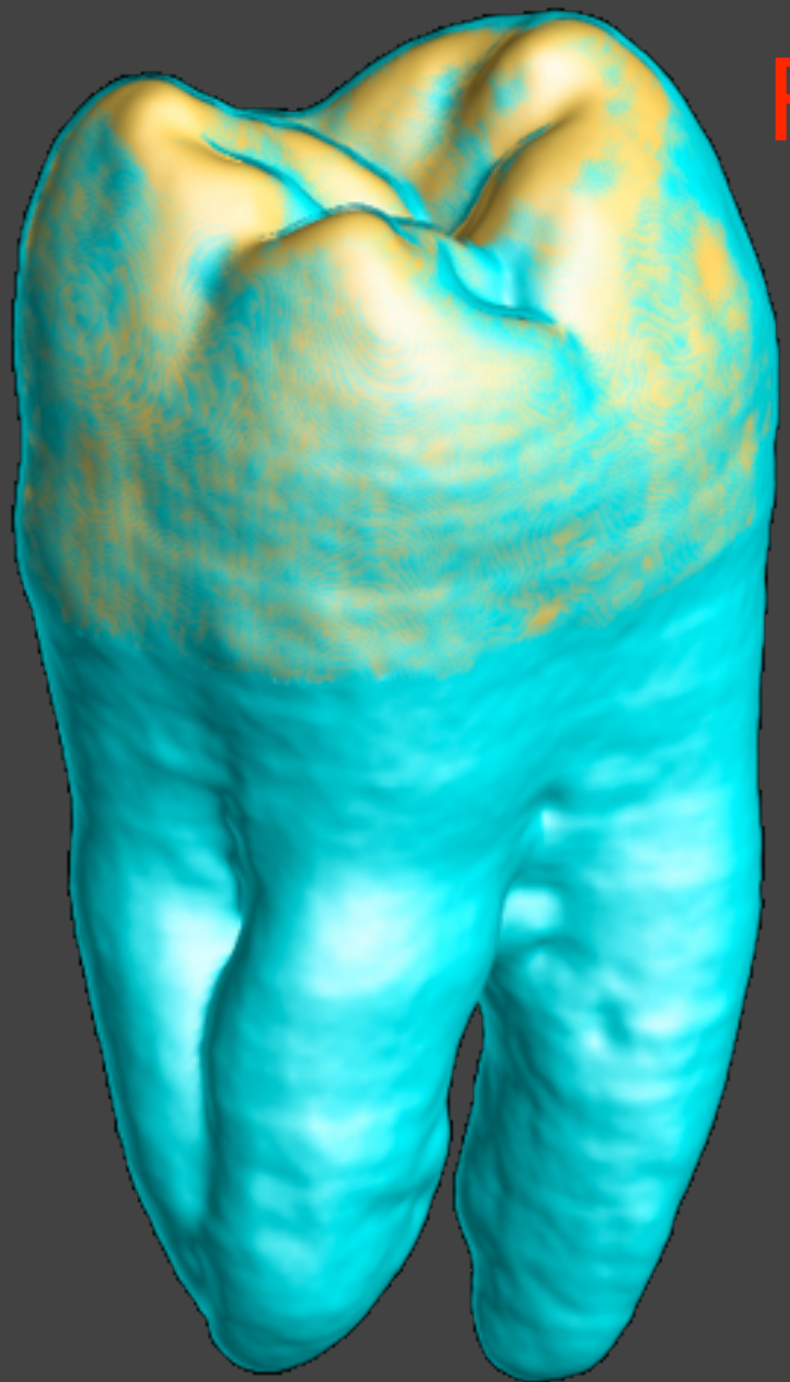
(colored lines correspond to different isosurface metrics)



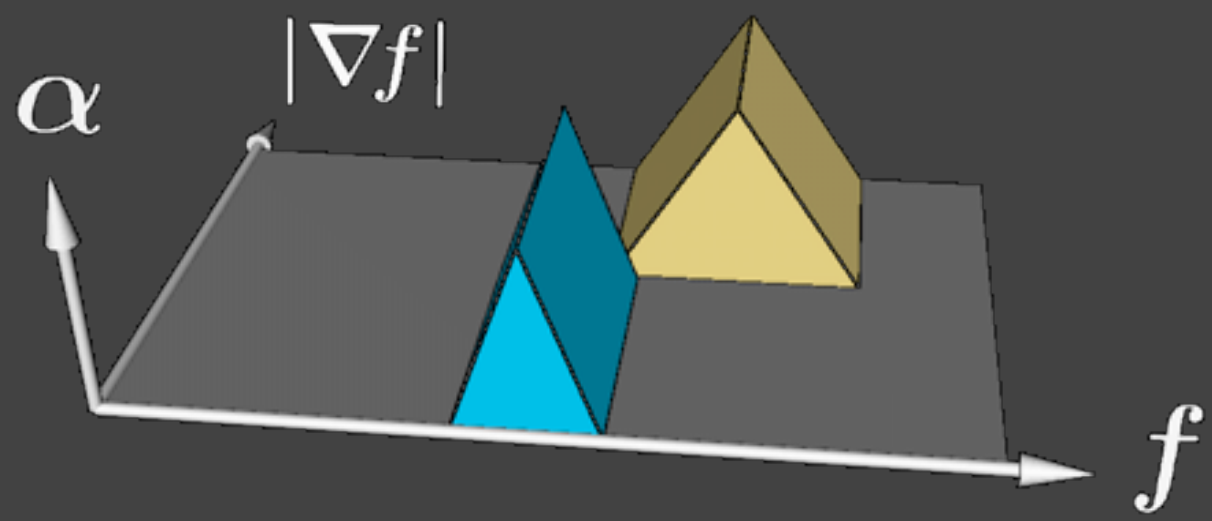
The contour spectrum allows the development of an adaptive ability to separate *interesting* isovalues from the others.



2D Transfer Function



$$\left.\begin{array}{l} \mathrm{RGB}(f, |\nabla f|) \\ \alpha(f, |\nabla f|) \end{array}\right\} \text{Modify…}$$

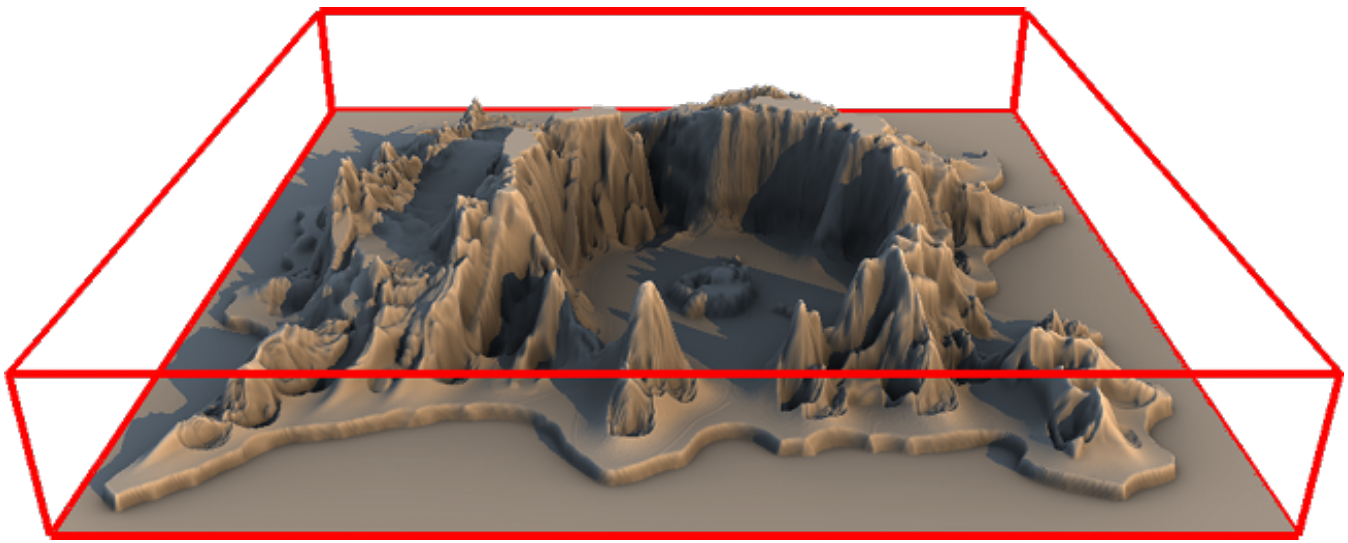


Trying to reintroduce dentin / background boundary …

Heightfields

Heightfields

- F(x,y) = h
- At its simplest, just a raster image (2D texture)
- Need some way to reconstruct the mesh in between
  - Explicit geometry (interpolating mesh)
  - Implicit geometry (ray tracing parametric patches)

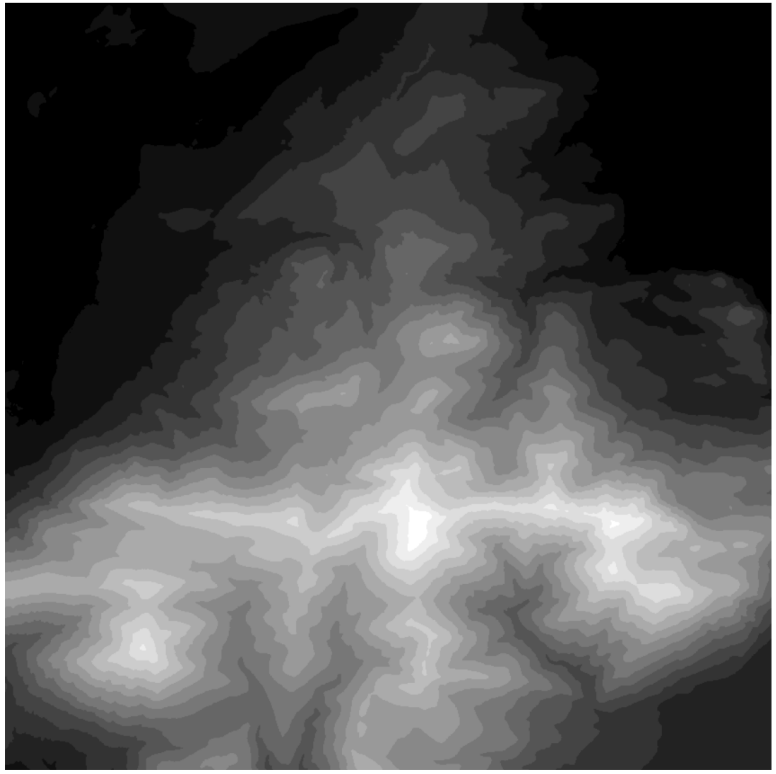


Terrain visualization

- DEM acquired by resampling LiDAR point data onto a grid
- Often accompanied by color
  - F(x,y) = {h,r,g,b}

Orthophoto

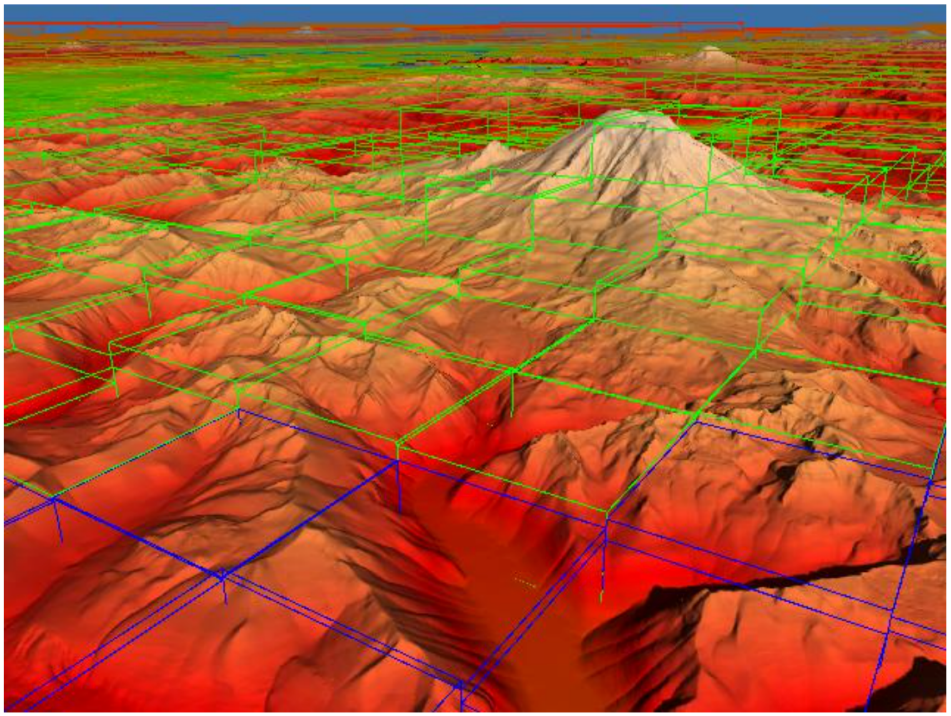
Digital Elevation Model

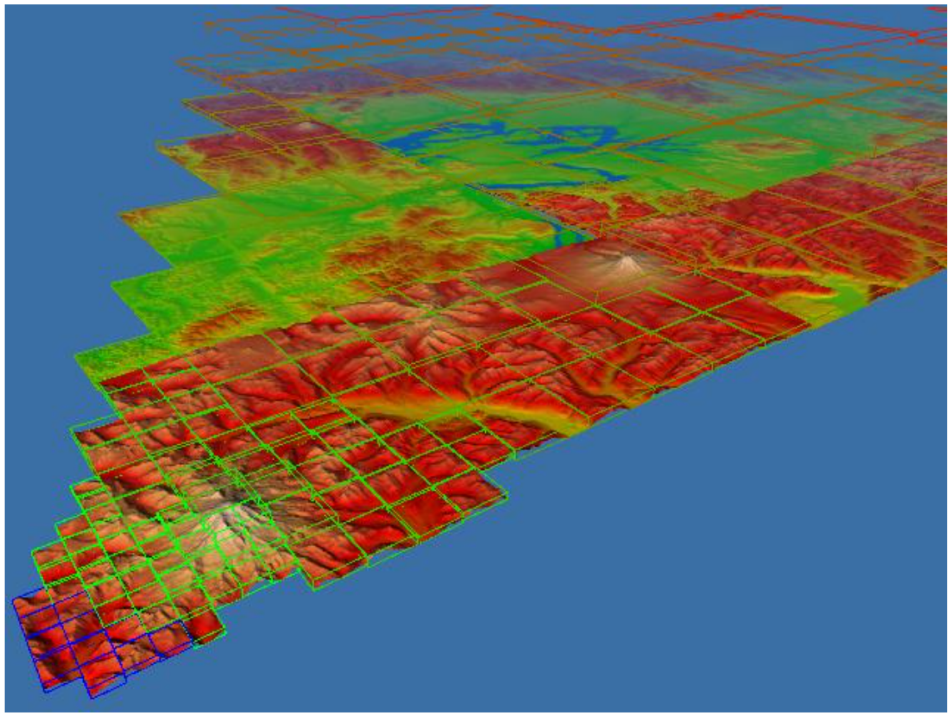


Explicit Terrain Rendering

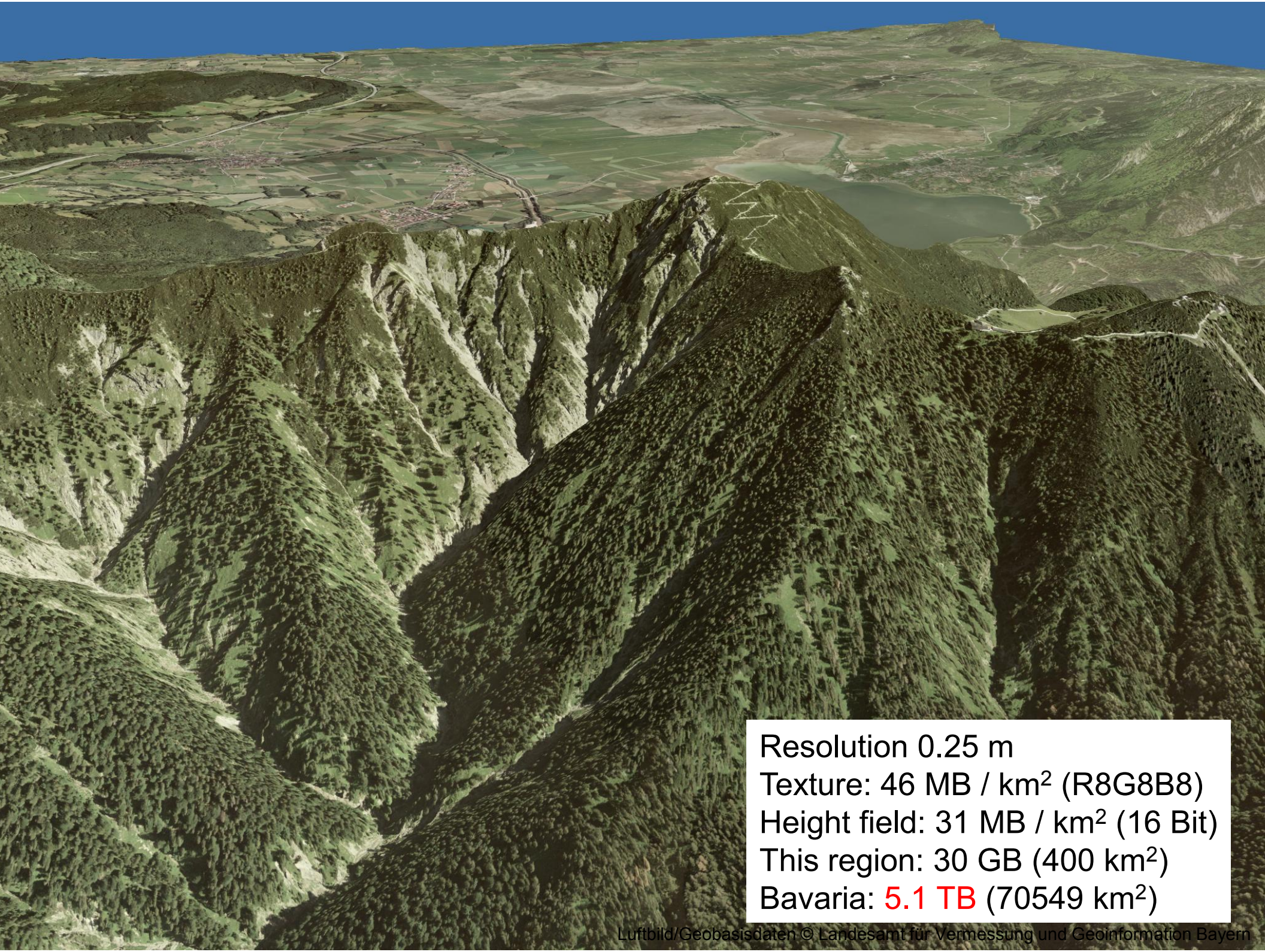
- Geometry compression (split quadtree)
  Texture compression (Built-in S3TC compression in DirectX)

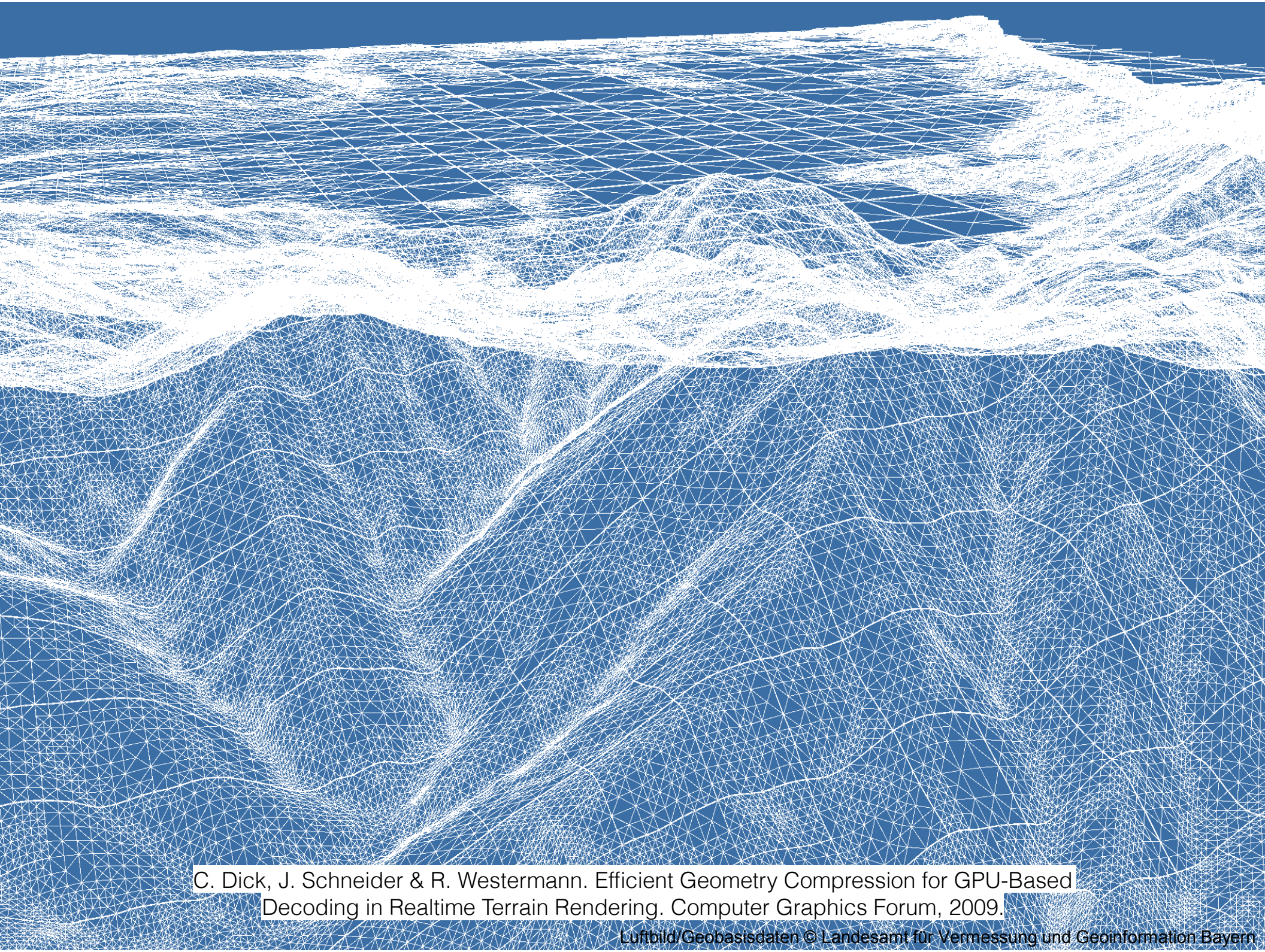
- Out-of-core rendering of a 5.1 TB terrain dataset, .25m LiDAR
  135+ fps at 1080p on a 880 GTX in 2007!





C. Dick, J. Schneider & R. Westermann. Efficient Geometry Compression for GPU-Based Decoding in Realtime Terrain Rendering. Computer Graphics Forum, 2009.

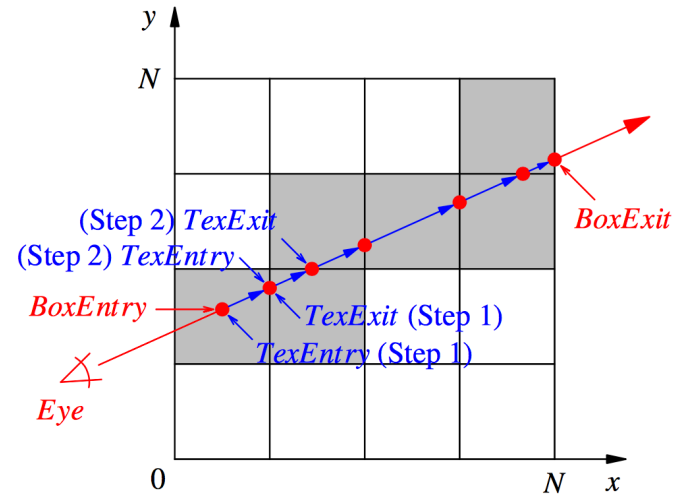

Resolution 0.25 m
Texture: 46 MB / km² (R8G8B8)
Height field: 31 MB / km² (16 Bit)
This region: 30 GB (400 km²)
Bavaria: 5.1 TB (70549 km²)
Luftbild/Geobasisdaten © Landesamt für Vermessung und Geoinformation Bayern



C. Dick, J. Schneider & R. Westermann. Efficient Geometry Compression for GPU-Based Decoding in Realtime Terrain Rendering. Computer Graphics Forum, 2009.

Implicit Terrain Rendering

- Use ray casting to intersect bilinear patches directly.
- Same quadtree LOD as before, but without the diagonal splits
- Lower memory footprint
  (i.e., you can fit more high-resolution tiles in core)
- Significantly faster for high-resolution data (.25 m Vorarlberg); slower for smoother low-resolution data (1 m Utah)



C. Dick, J. Krueger & R. Westermann. GPU Ray-Casting for Scalable Terrain Rendering.
Eurographics 2009 Area Papers

More Terrain Rendering

- Terrain visualization for whole planets in a Planetarium

  R. Kooima, J Leigh, A Johnson, D Roberts, M SubbaRao, T DeFan
  Planetary-Scale Terrain Composition. IEEE Visualization 2009.

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

- LA Times Data Visualization: Mars Gale Crater in Three.js

  http://graphics.latimes.com/mars-gale-crater-how-we-did-it/