

Scientific Visualization: Surfaces

CS 6630, Fall 2015 — Alex Lex
Aaron Knoll, guest lecturer

Recap from last sci-vis lecture

- 3D graphics
 - rasterization vs ray tracing
 - rendering pipelines, indirect and direct visualization
- Volume rendering
 - Emission-absorption, classification, blending
- Transfer functions
 - 1D vs 2D transfer functions
 - (start from slide 117)

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.

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

- Discrete (Riemann sum) volume rendering

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

- Blending operation (front-to-back)

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

Today

- Wrap up transfer functions
- Surfaces
 - Explicit vs implicit
 - Terrain visualization
 - Contours
- Isosurfaces
 - Marching Cubes and variants
 - Particle-based extraction
 - Splatting
 - Ray casting/tracing

Wrap up transfer functions

High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading

Klaus Engel, Martin Kraus, Thomas Ertl *

Visualization and Interactive Systems Group, University of Stuttgart, Germany

Abstract

We introduce a novel texture-based volume rendering approach that achieves the image quality of the best post-shading approaches with far less slices. It is suitable for new flexible consumer graphics hardware and provides high image quality even for low-resolution volume data and non-linear transfer functions with high frequencies, without the performance overhead caused by rendering additional interpolated slices. This is especially useful for volumetric effects in computer games and professional scientific volume visualization, which heavily depend on memory bandwidth and rasterization power.

We present an implementation of the algorithm on current programmable consumer graphics hardware using multi-textures with advanced texture fetch and pixel shading operations. We implemented direct volume rendering, volume shading, arbitrary number of isosurfaces, and mixed mode rendering. The performance does neither depend on the number of isosurfaces nor the definition of the transfer functions, and is therefore suited for interactive high-quality volume graphics.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation, I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling, I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

Keywords: direct volume rendering, volume graphics, volume shading, volume visualization, multi-textures, rasterization, PC graphics hardware, flexible graphics hardware

1 Introduction

In spite of recent progress in texture-based volume rendering algorithms, volumetric effects and visualizations have not reached the mass market. One of the reasons is the requirement for ex-

while the technical details of an implementation on current programmable consumer graphics hardware are described in Section 5. In particular, we discuss the use of advanced texture fetch and pixel shading operations recently proposed by graphics hardware vendors [4]. These features are exploited in order to achieve direct volume rendering, multiple smoothly shaded isosurfaces, and volume shading. Preliminary results on a *GeForce3* graphics hardware are presented in Section 6. Finally, Section 7 sums up the paper.

2 Related Work

High accuracy in direct volume rendering is usually achieved by very high sampling rates resulting in heavy performance losses. However, for cell-projective techniques Max, Williams, and Stein have proposed elaborated optical models and efficient, highly accurate projective methods in [8, 14]. The latter were further improved by Röttger, Kraus, and Ertl in [12]. Although these techniques were initially limited to cell projection, we were able to generalize them in order to apply these ideas to texture-based rendering approaches.

The basic idea of using object-aligned textured slices to substitute trilinear by bilinear interpolation was presented by Lacroute and Levoy [6], although the original implementation did not use texturing hardware. For the PC platform, Brady et al. [2] have presented a technique for interactive volume navigation based on 2D texture mapping.

The most important texture-based approach was introduced by Cabral [3], who exploited the 3D texture mapping capabilities of high-end graphics workstations. Westermann and Ertl [13] have significantly expanded this approach by introducing a fast direct multi-pass algorithm to display shaded isosurfaces. Based on their implementation, Meißner et al. [9] have provided a method to enable diffuse illumination for semi-transparent volume rendering. However, in this case multiple passes through the rasterization hardware led to a significant loss in rendering performance. Dachille et al. [5] have proposed an approach that employs 3D texture hardware interpolation together with software shading and classification.

Preintegration

- Blend using the pre-summed (pre-integrated) transfer function between front and back samples
- Higher quality with fewer samples.

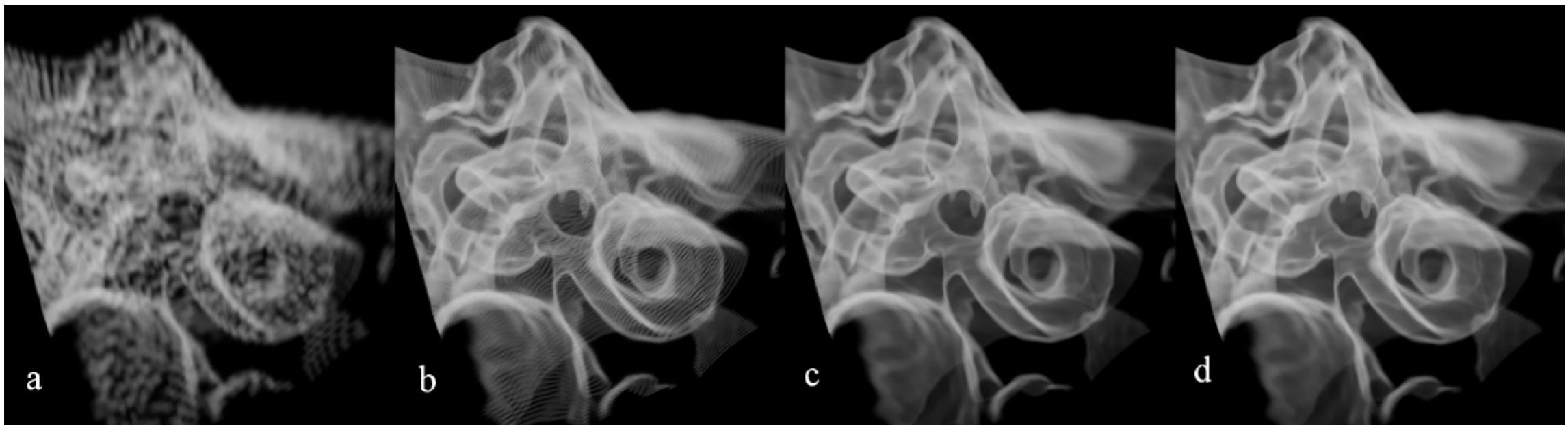
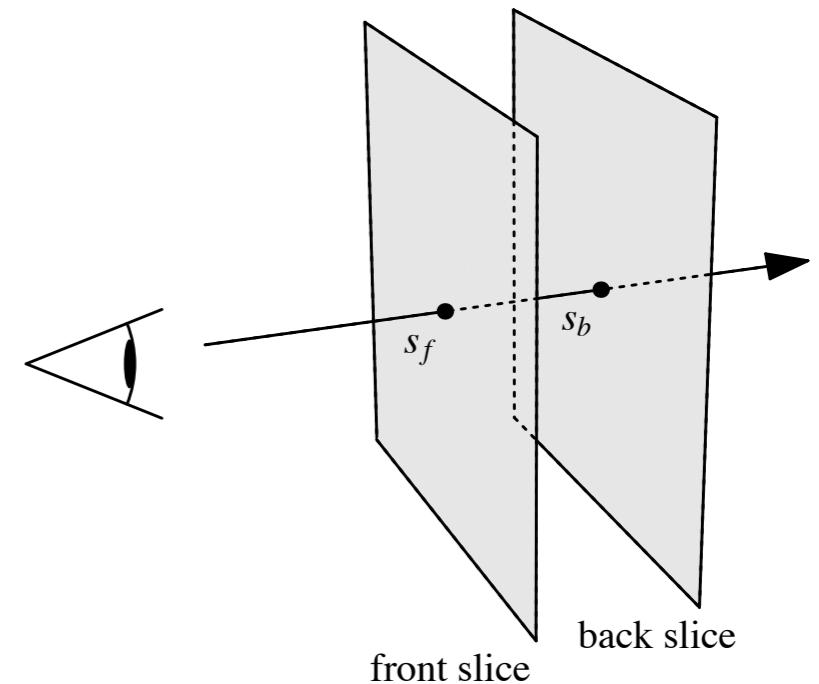
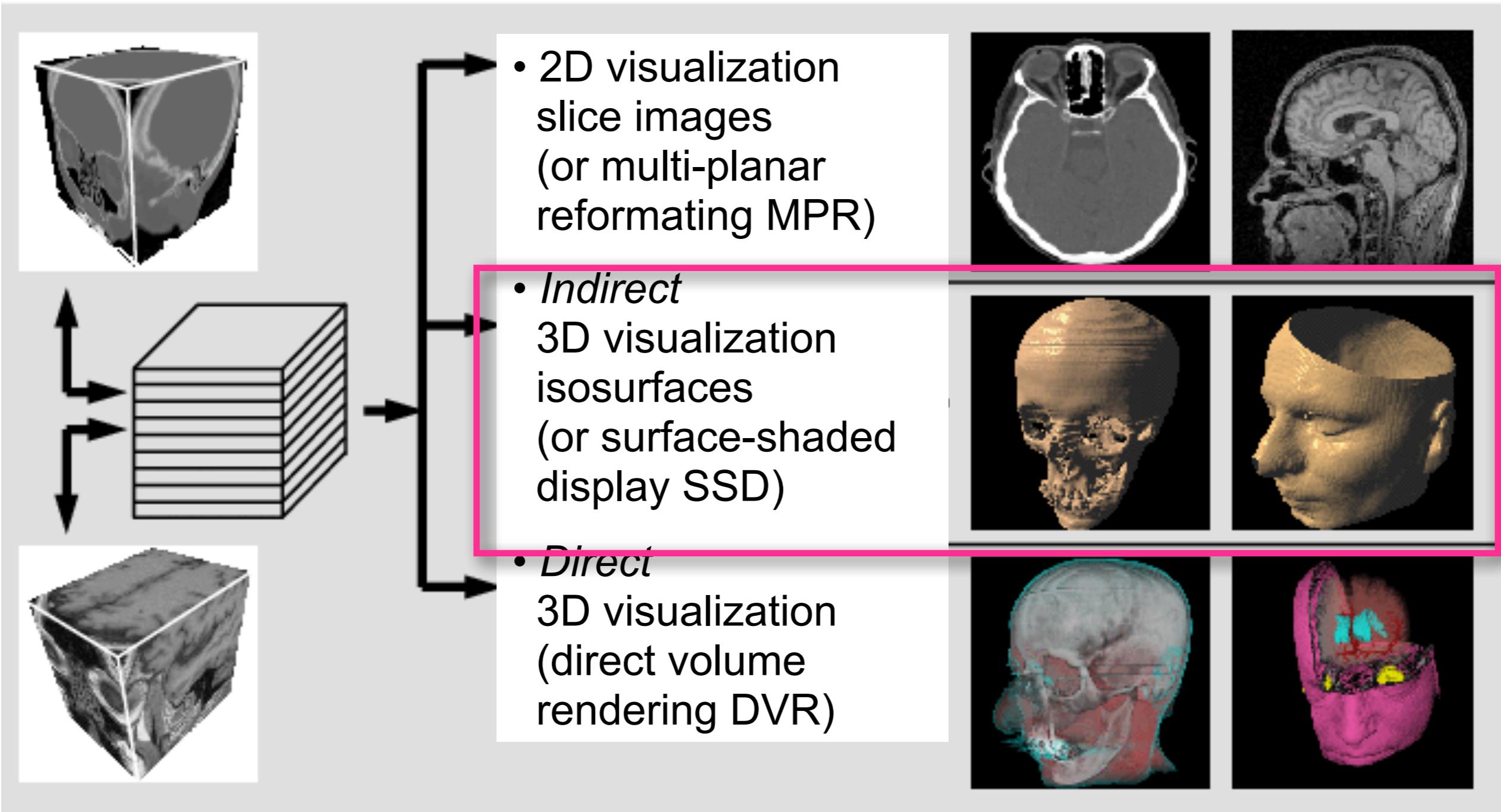


Figure 9: Images showing a comparison of a) pre-shaded, b) post-shaded without additional slices, c) post-shaded with additional slices and d) pre-integrated volume visualization of tiny structures of the inner ear ($128 \times 128 \times 30$) with 128 slices.

Surfaces



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.

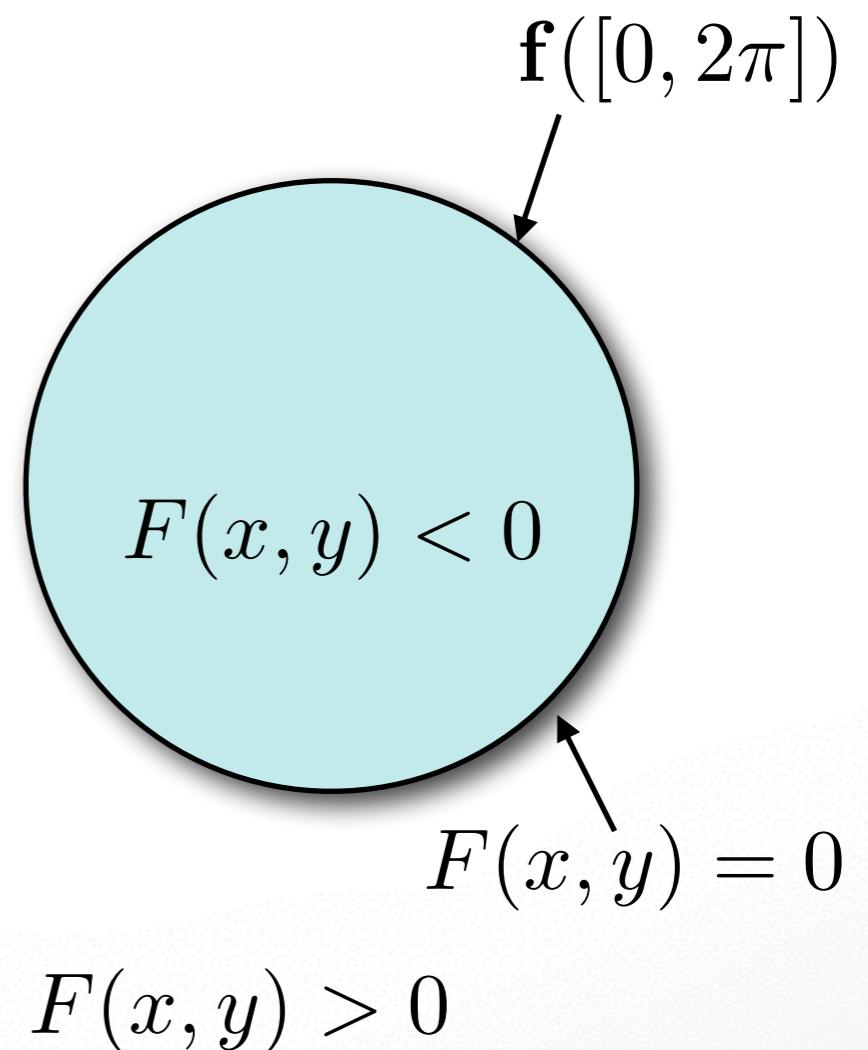
Explicit vs. Implicit

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

- Range of parameterization function

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

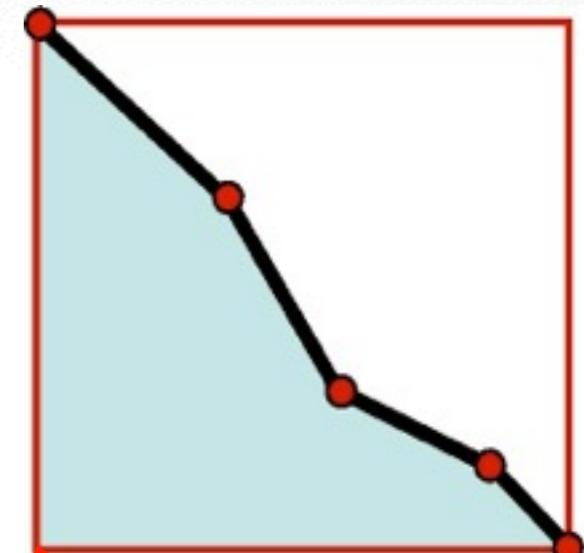
- Kernel of implicit function



Explicit vs. Implicit

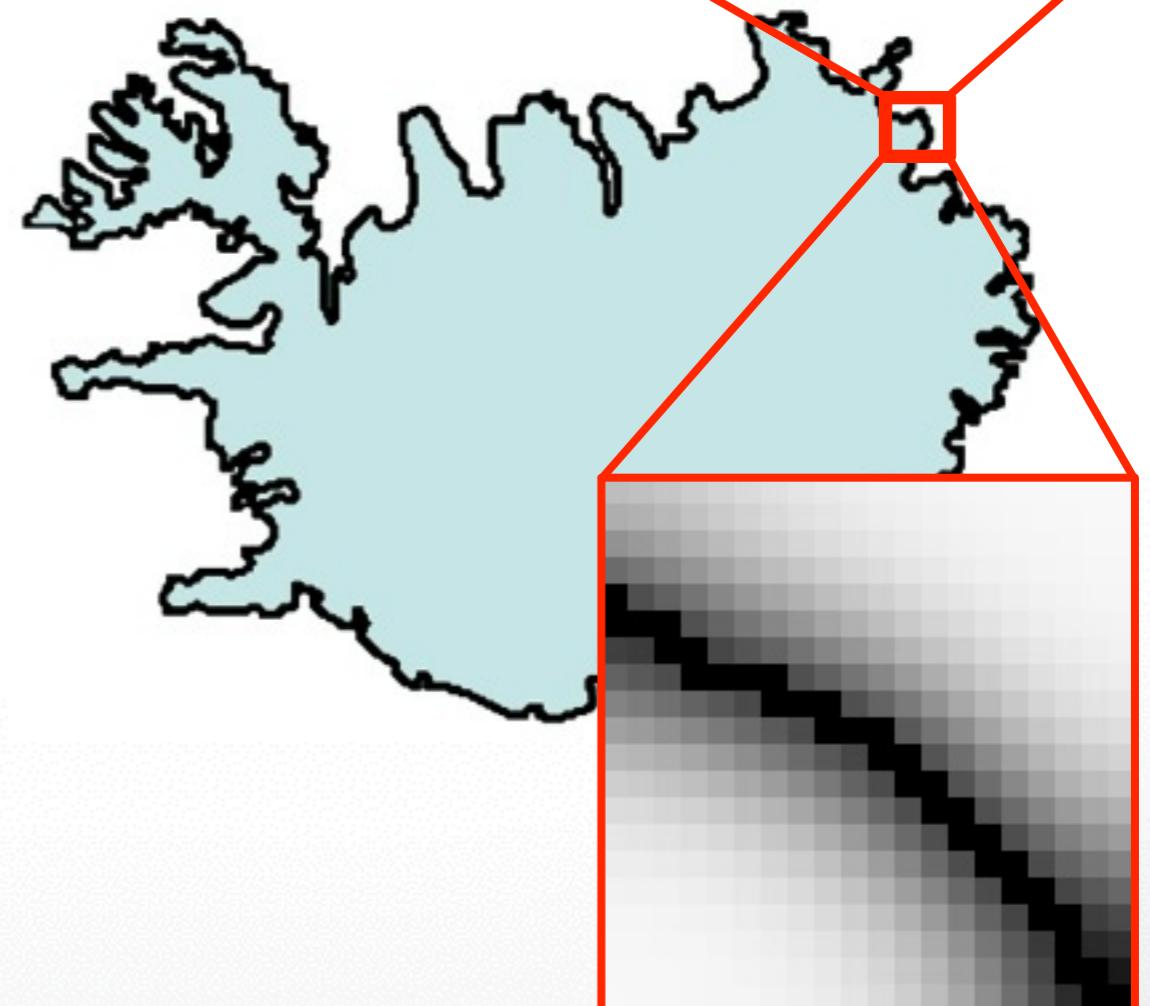
Explicit: $f(x) = ?$

- Range of parameterization function
- Piecewise approximation



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

- Kernel of implicit function
- Piecewise approximation



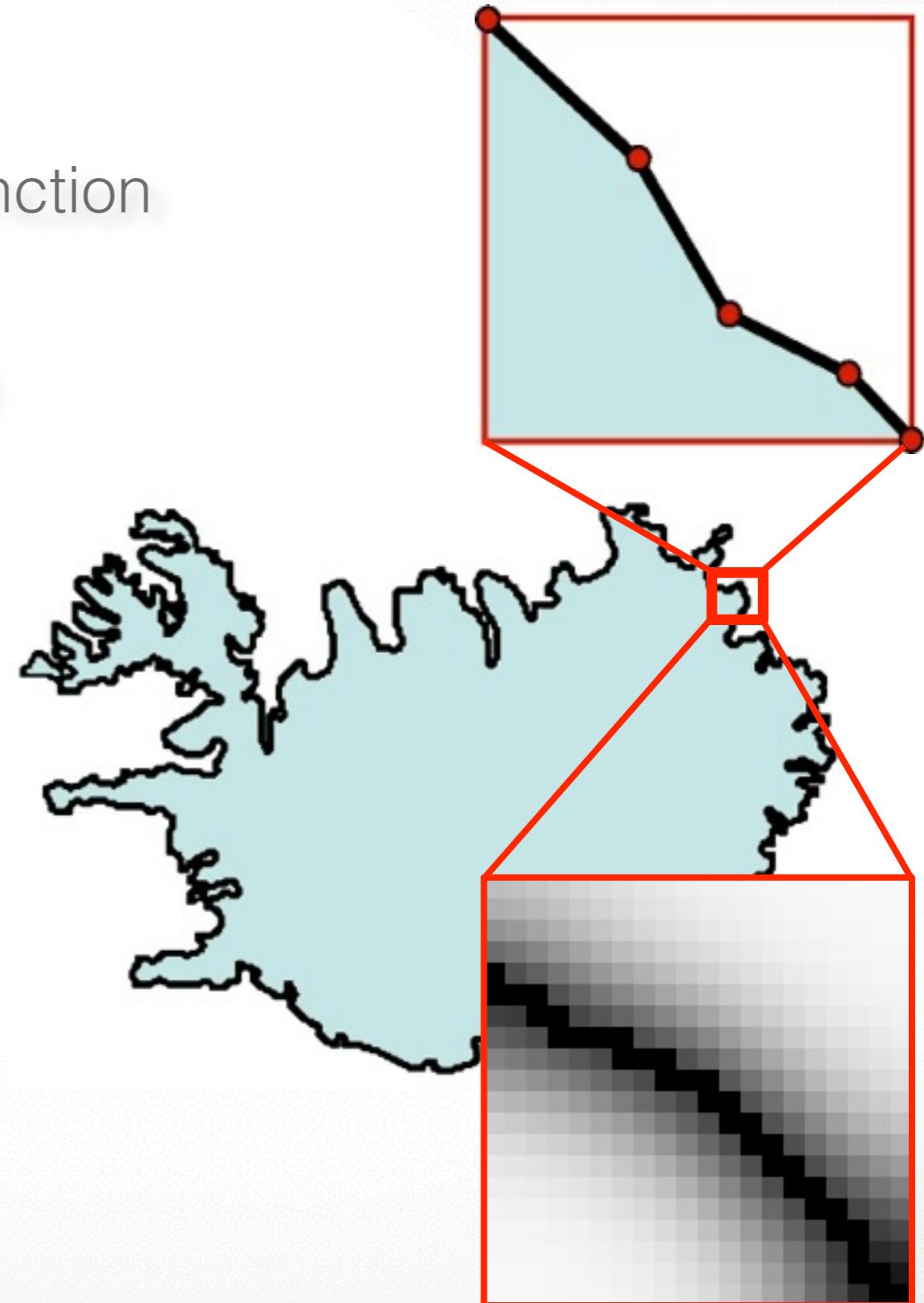
Explicit vs. Implicit

Explicit:

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

Implicit:

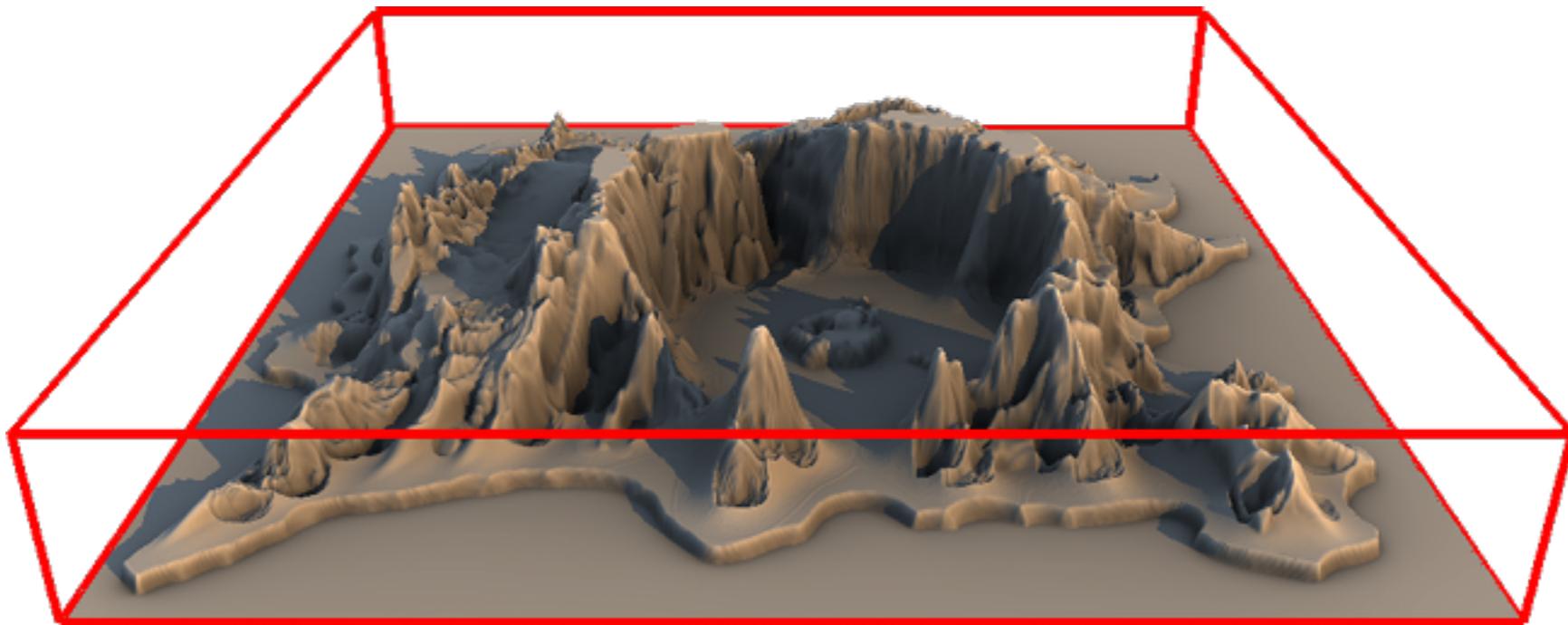
- Kernel of implicit function
- Piecewise approximation
- Scalar-valued 3D grid
- Easy in/out test
- Easy topology modification



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)



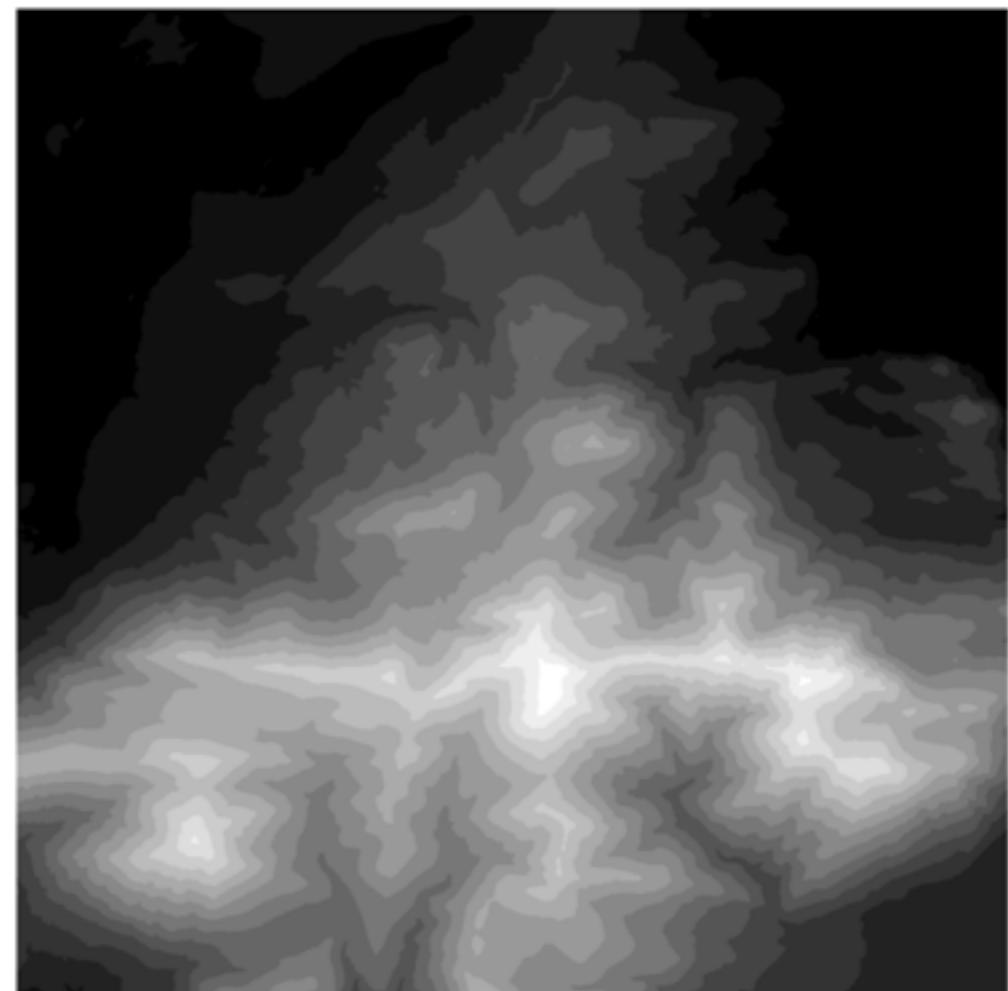
Terrain visualization

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

Orthophoto

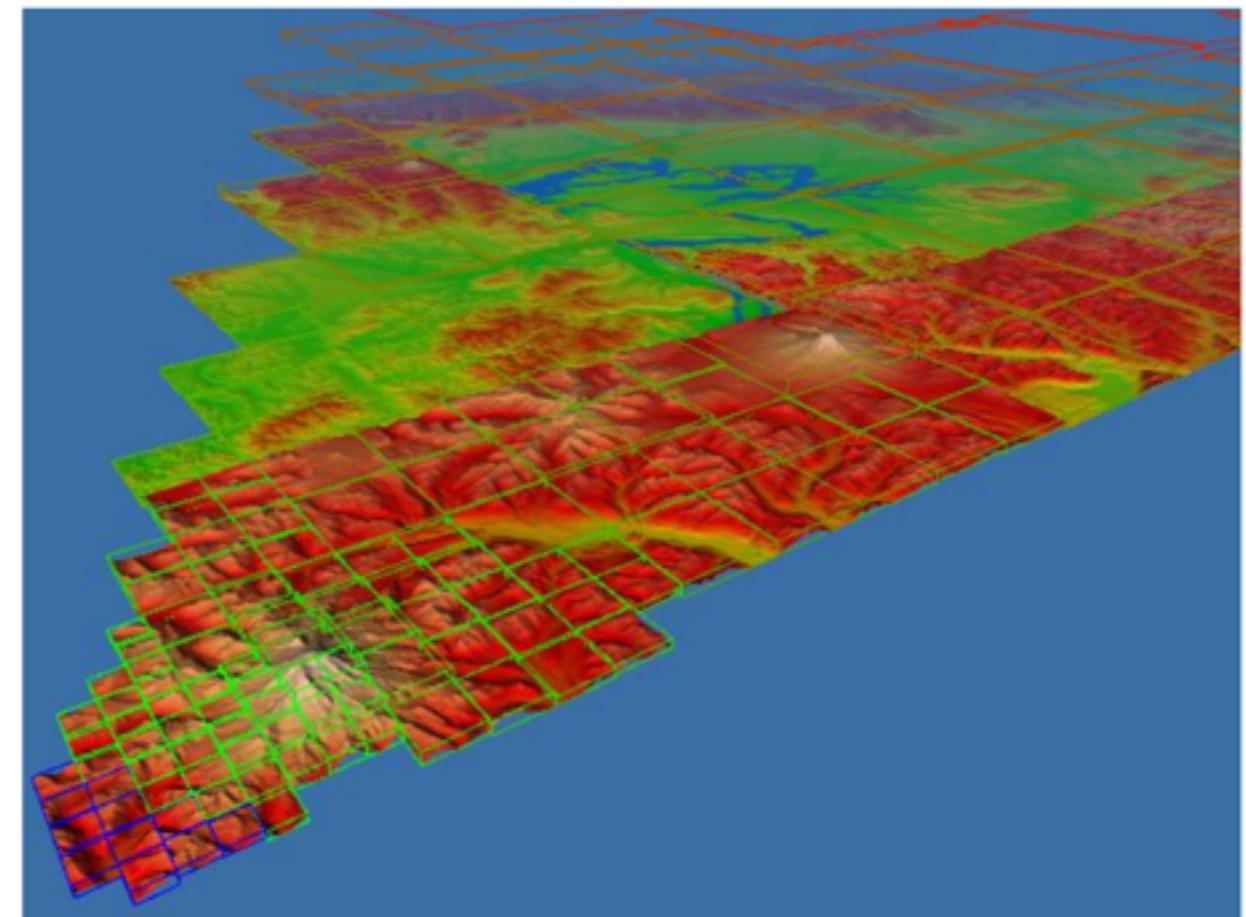
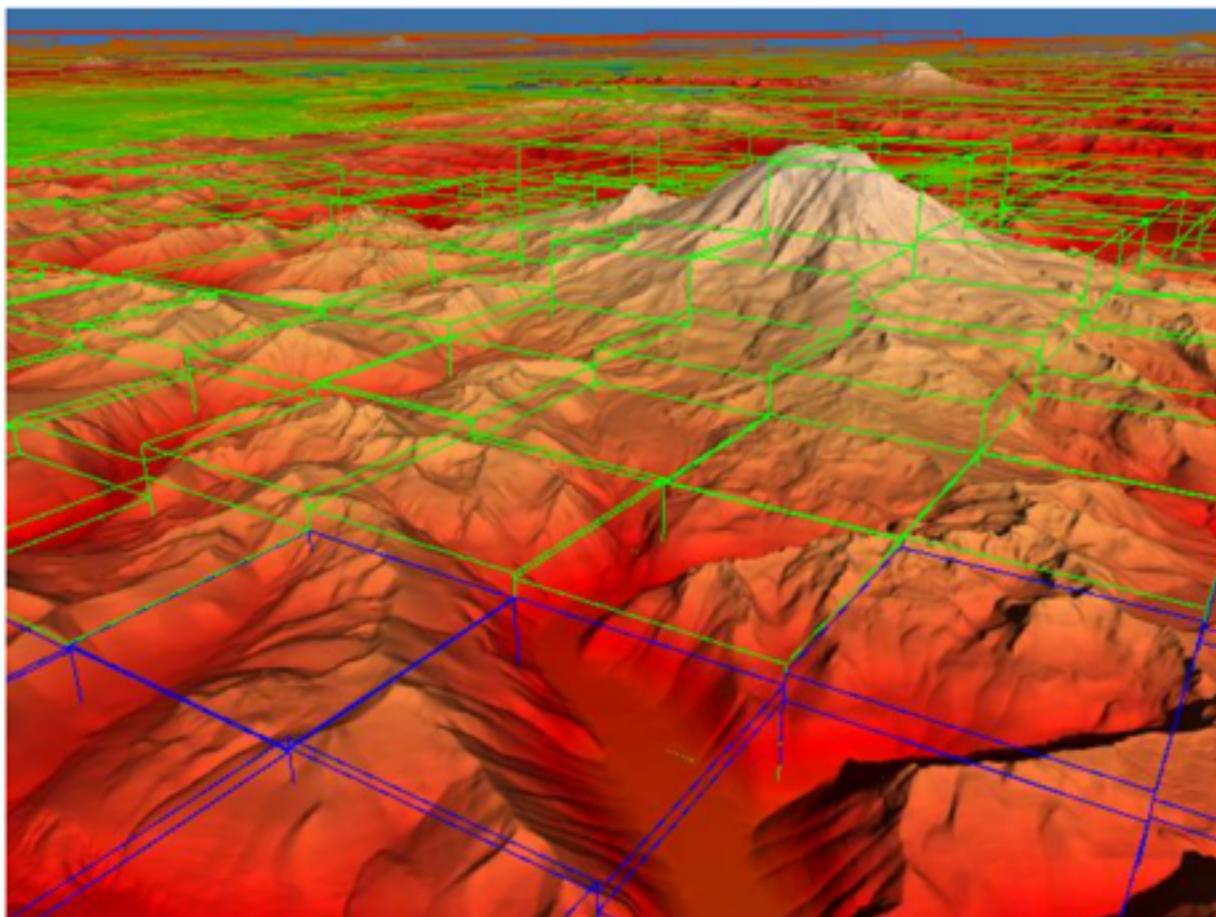


Digital Elevation Model

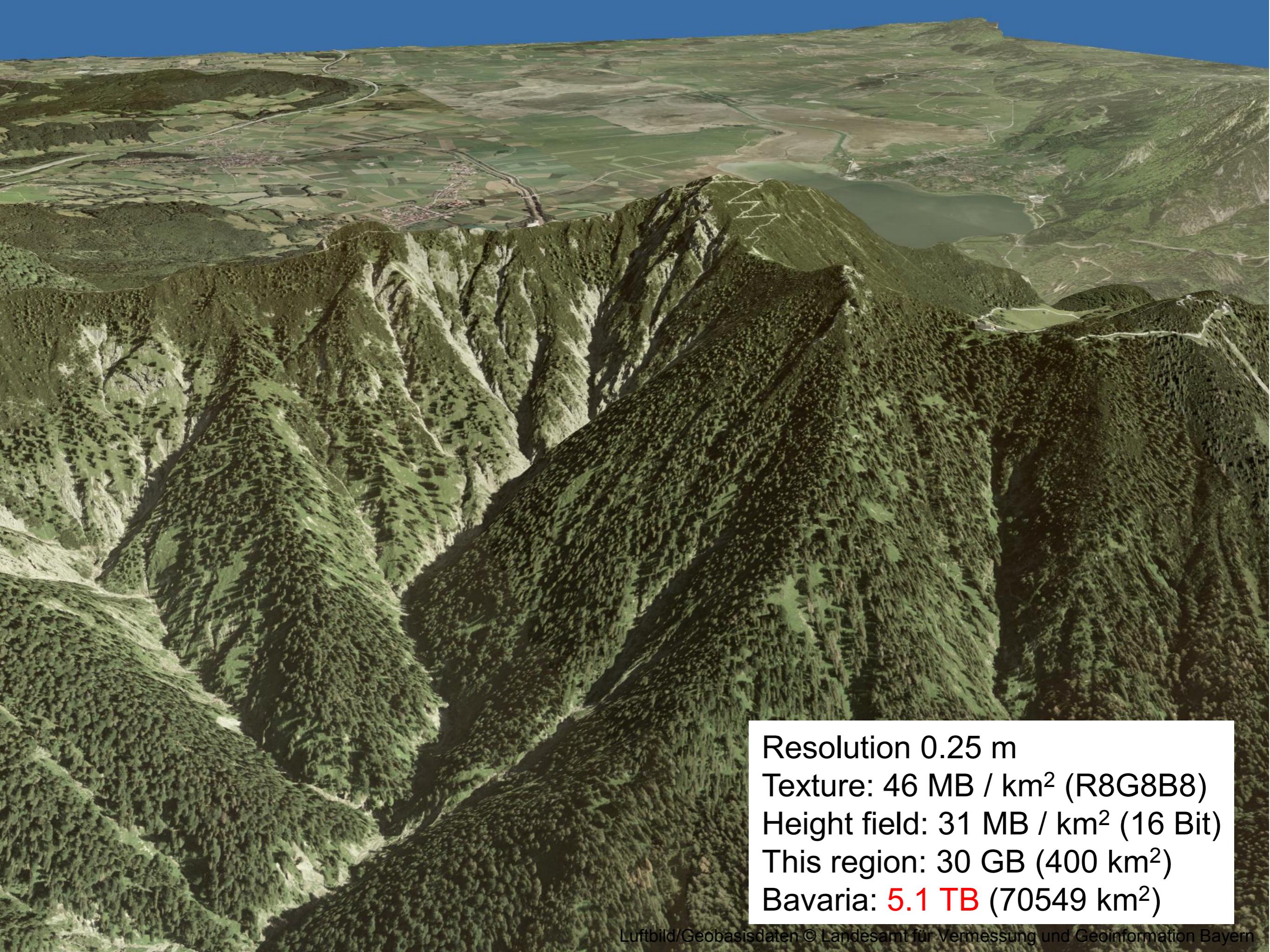


Explicit Terrain Rendering

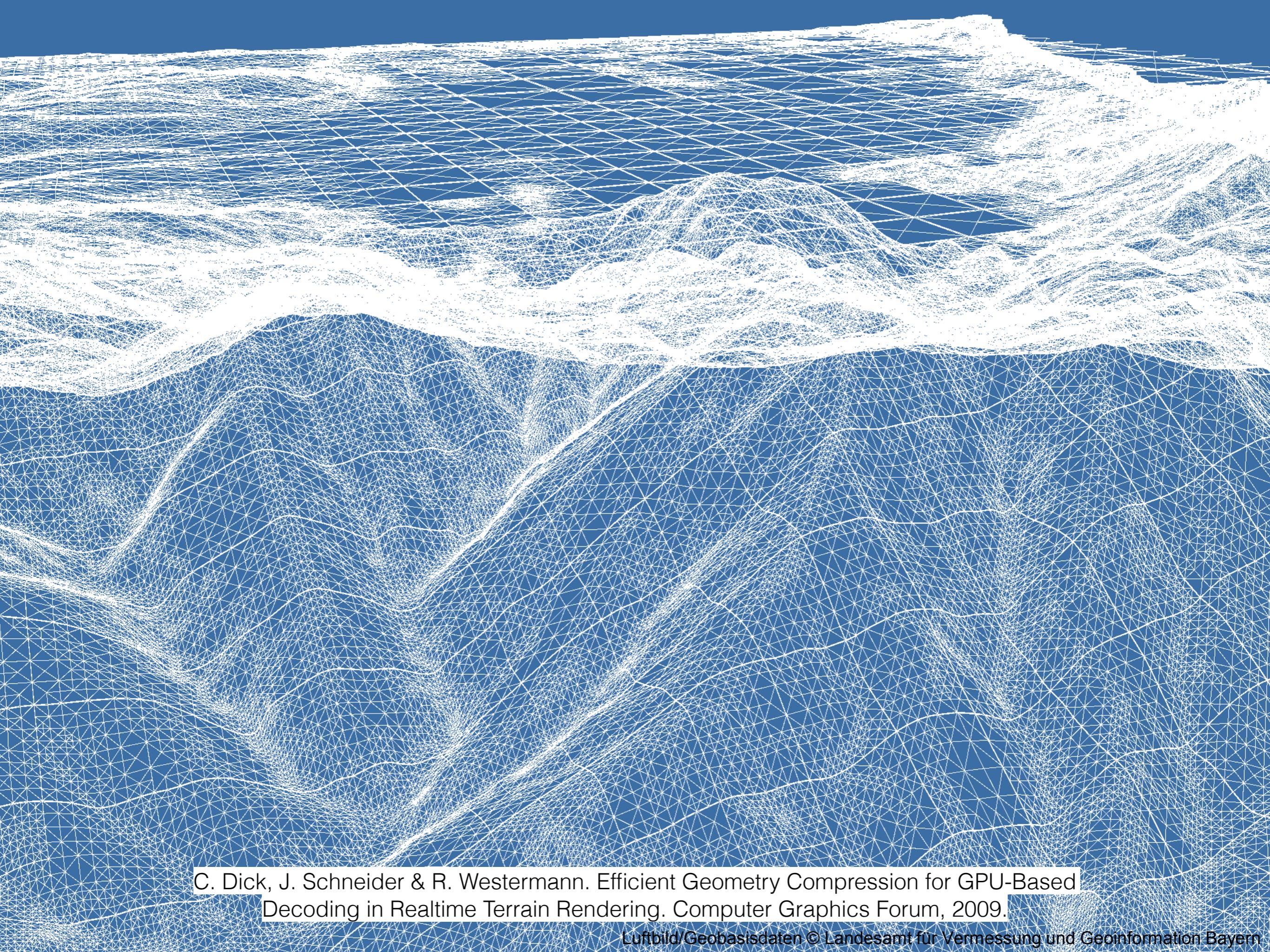
- Geometry compression (split quadtree)
Texture compression (Built-in S3TC compression in DirectX)
- Out-of-core rendering of a 5.1 TB terrain dataset, .25m LiDAR
135+ fps at 1080p on a 880 GTX in 2007!



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



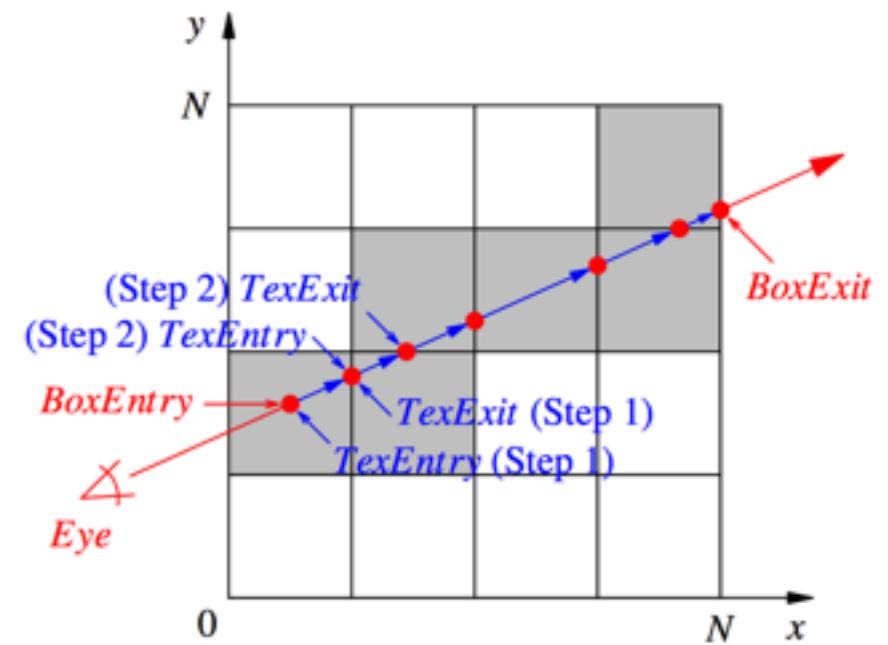
Resolution 0.25 m
Texture: 46 MB / km² (R8G8B8)
Height field: 31 MB / km² (16 Bit)
This region: 30 GB (400 km²)
Bavaria: 5.1 TB (70549 km²)



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)



More Terrain Rendering

- Terrain visualization for whole planets in a Planetarium

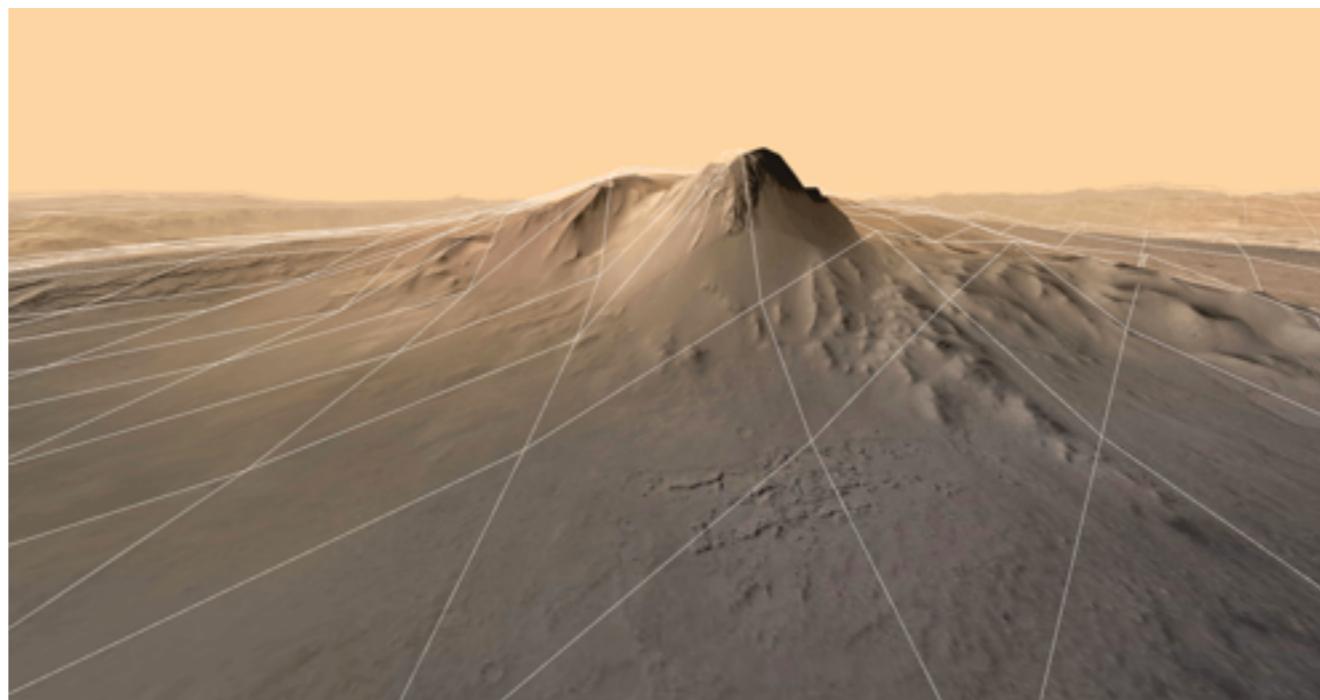
R. Kooima, J Leigh, A Johnson, D Roberts, M SubbaRao, T DeFanti. Planetary-Scale Terrain Composition. IEEE Visualization 2009.

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



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

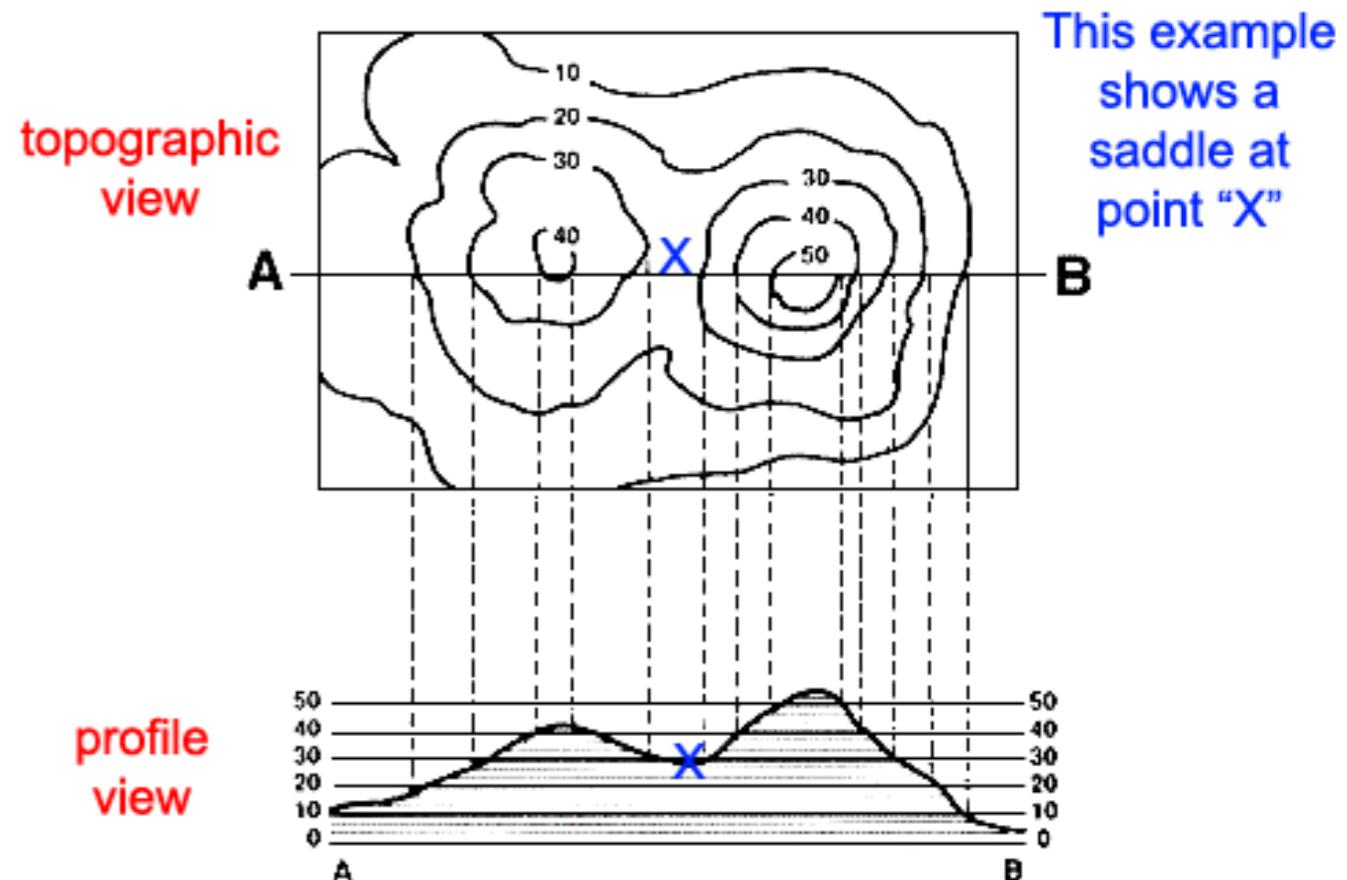
<http://graphics.latimes.com/mars-gale-crater-how-we-did-it/>

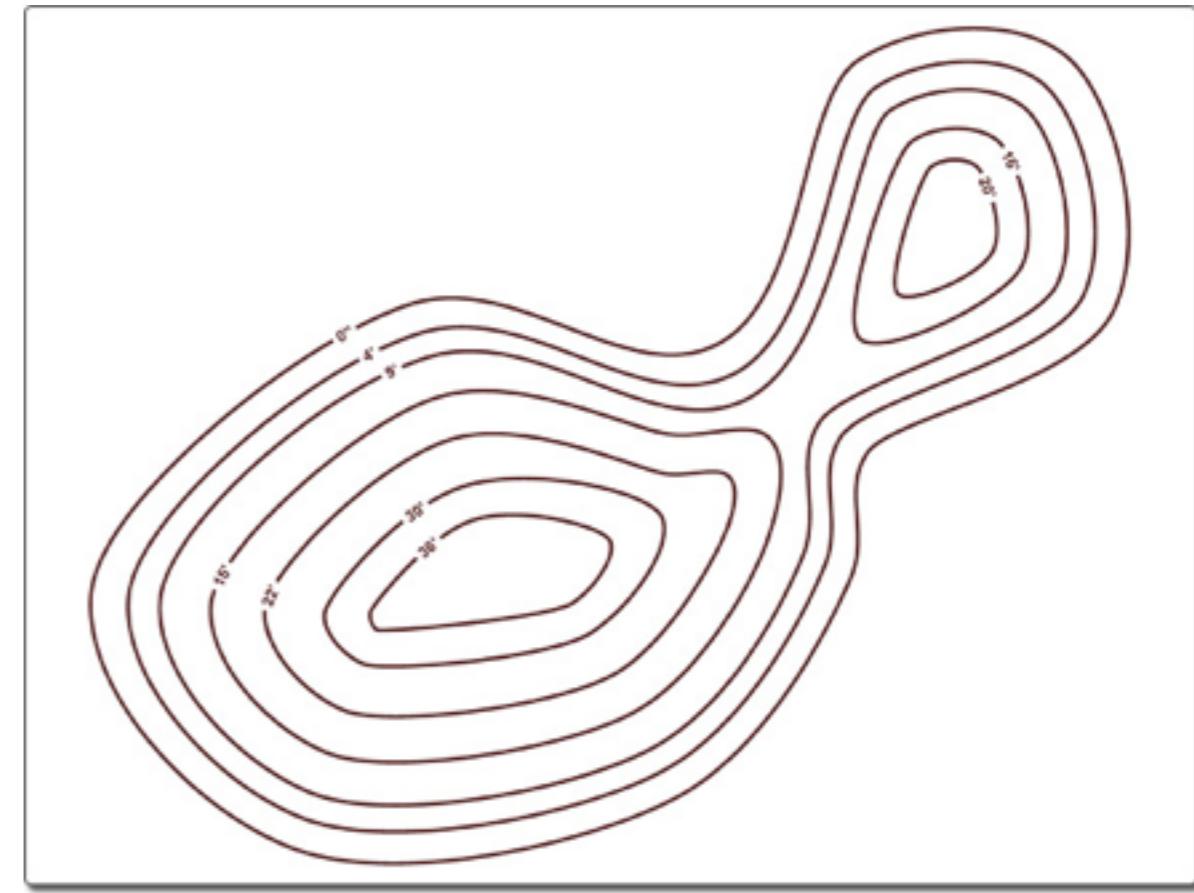
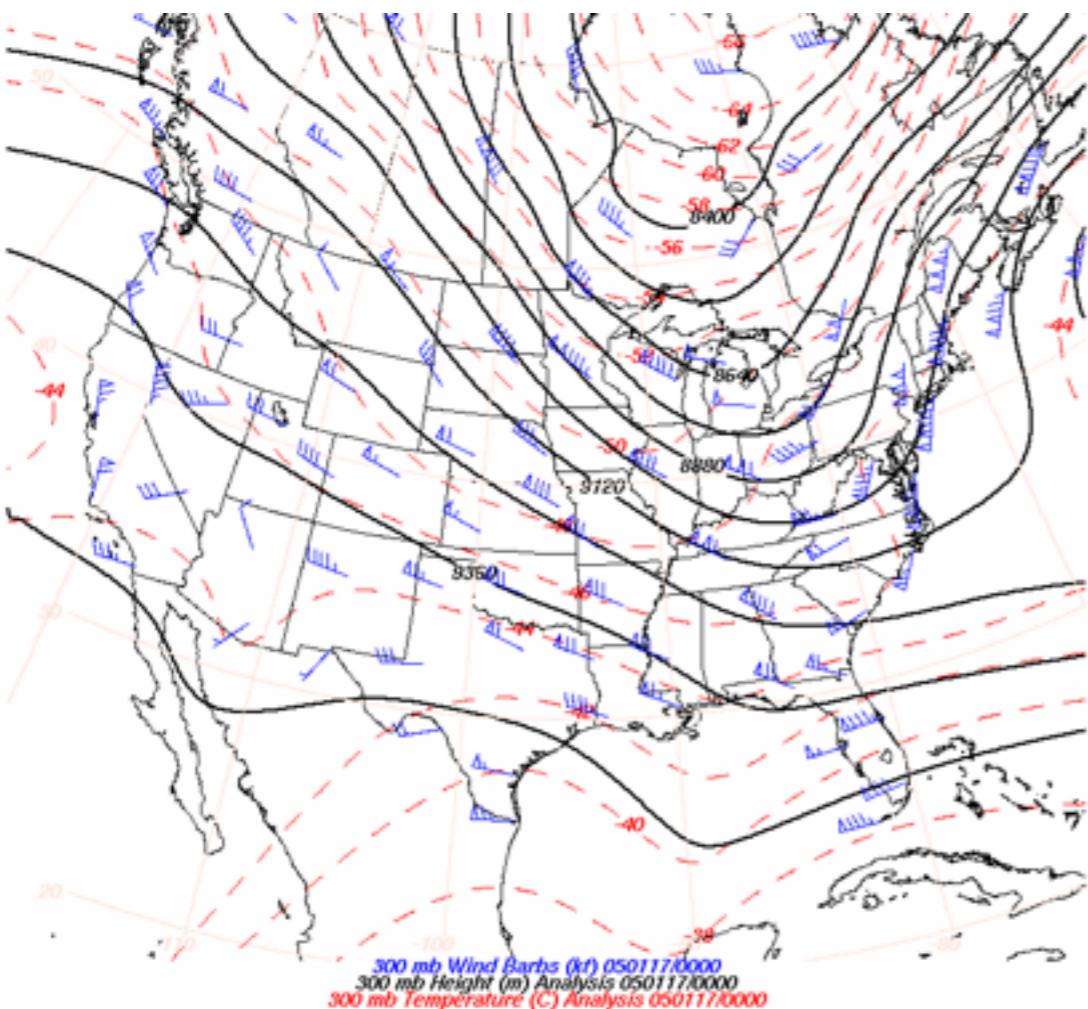


Contours

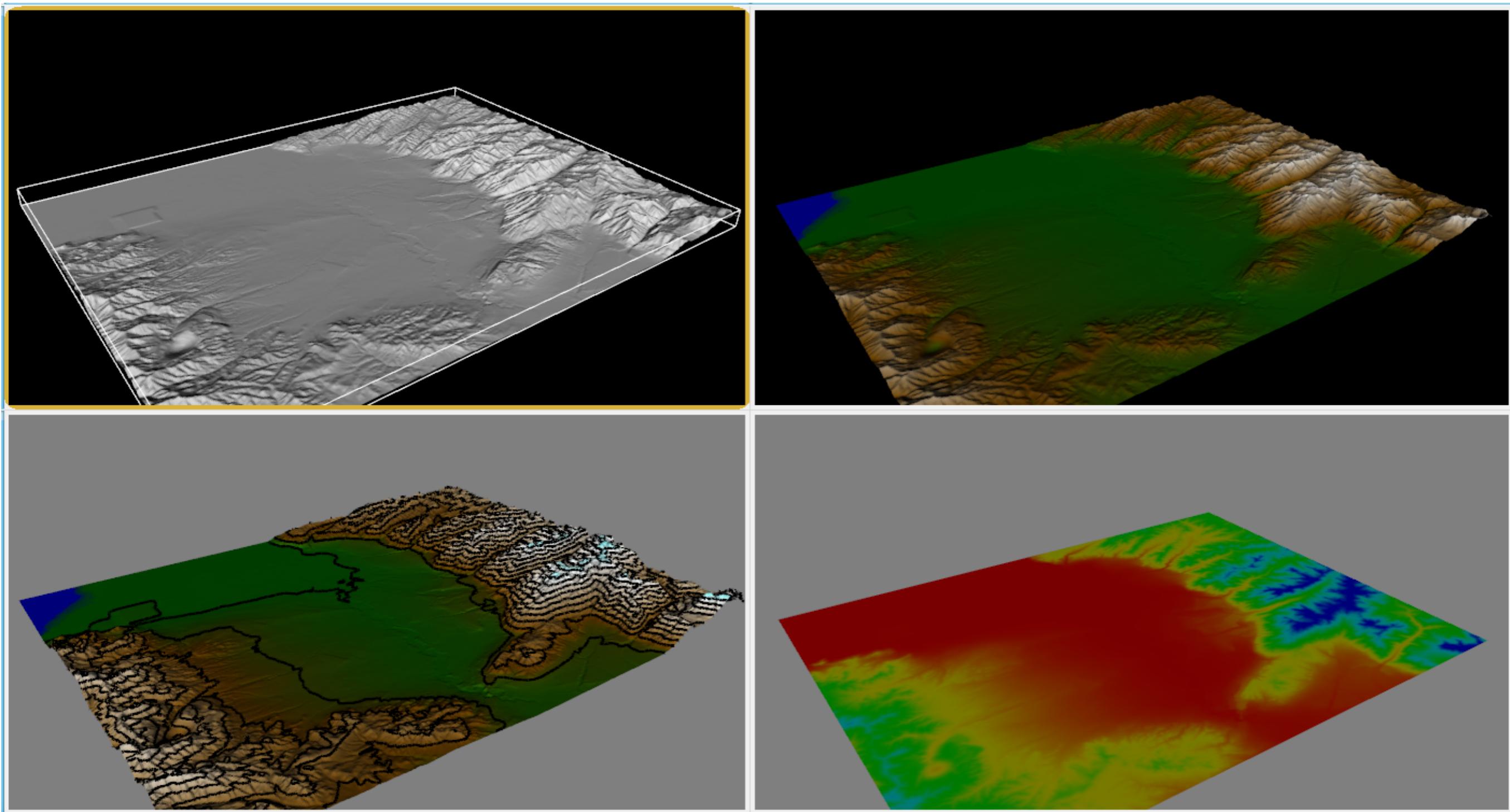
Contours

- In 2D, a **contour** at a value v of a scalar field $F(x,y)$ is the set of curves where $F(x,y) = v$.
- Design choices:
 - *Plan view vs profile view*
 - Line width, dashes, dots, labels.
- Why is it best to use multiple contours?



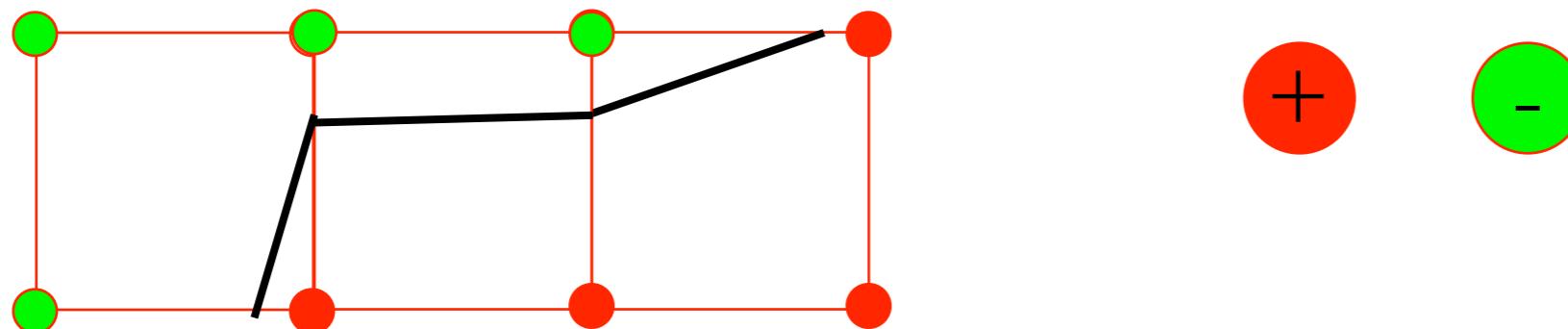
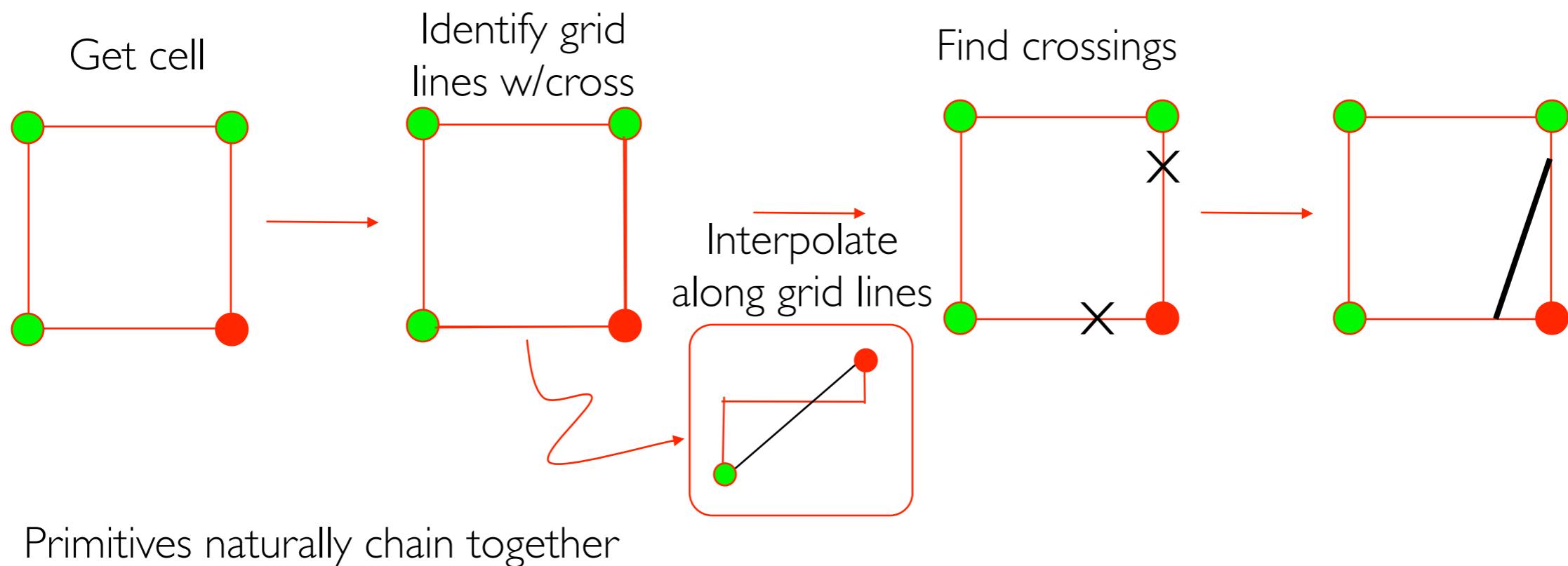


Compare

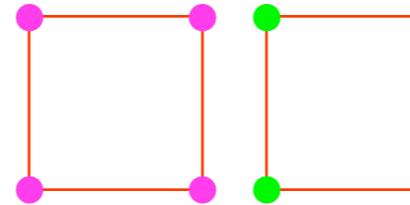
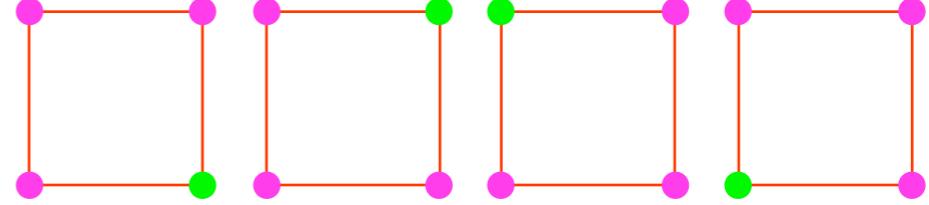
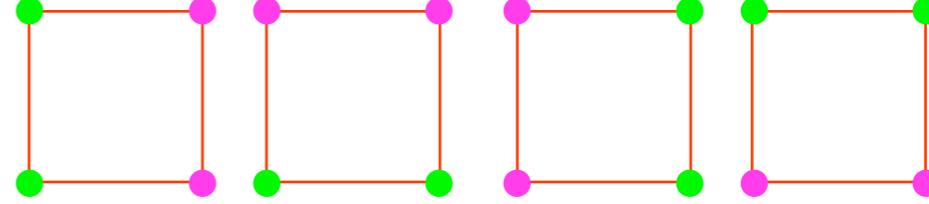
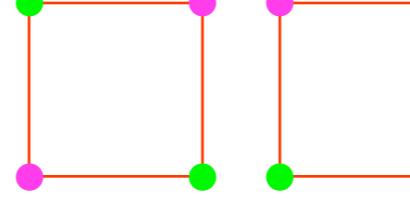


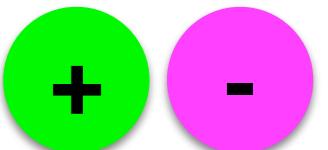
Approach to Contouring in 2D

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



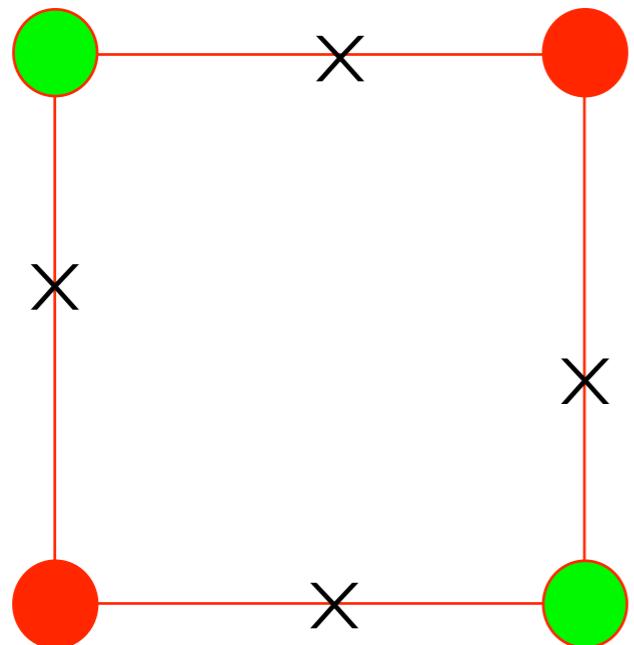
Cases

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	
				$16 = 2^4$



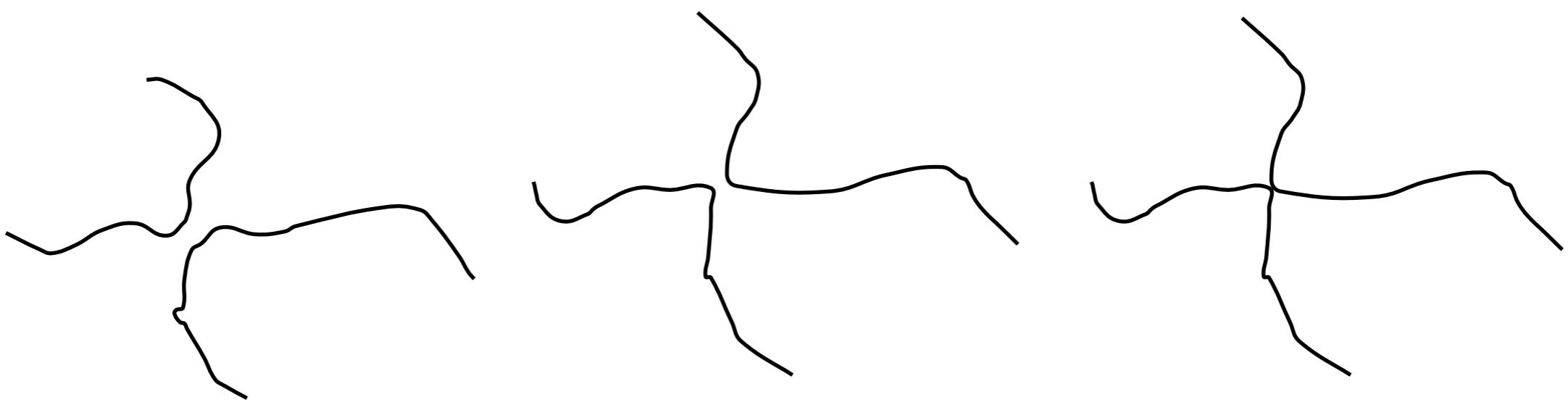
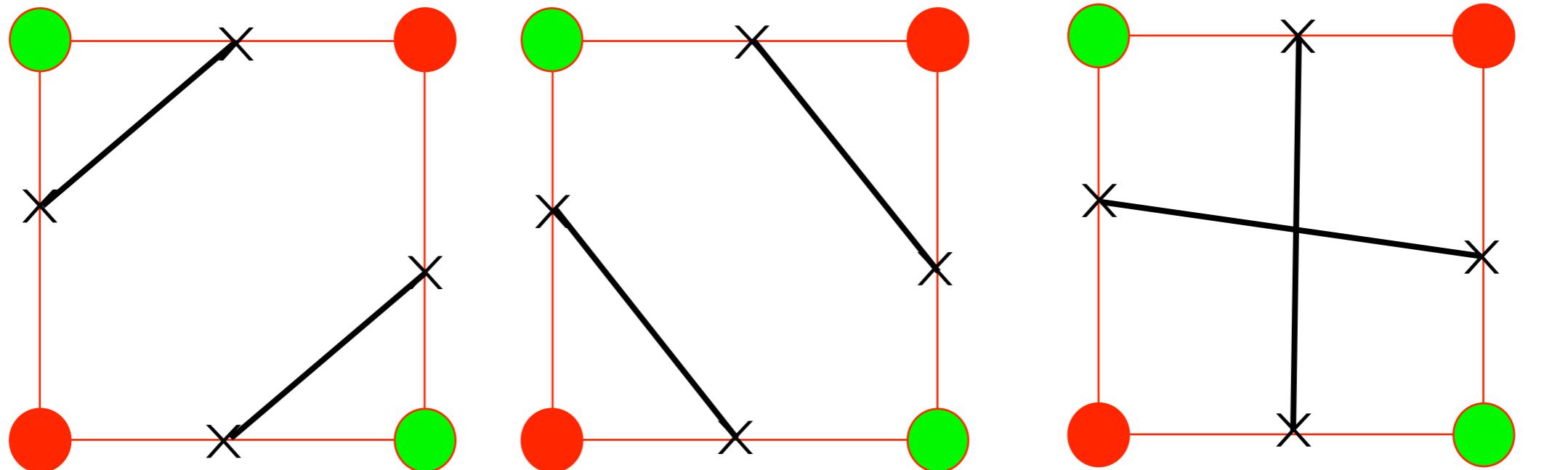
Ambiguities

- How to form lines?



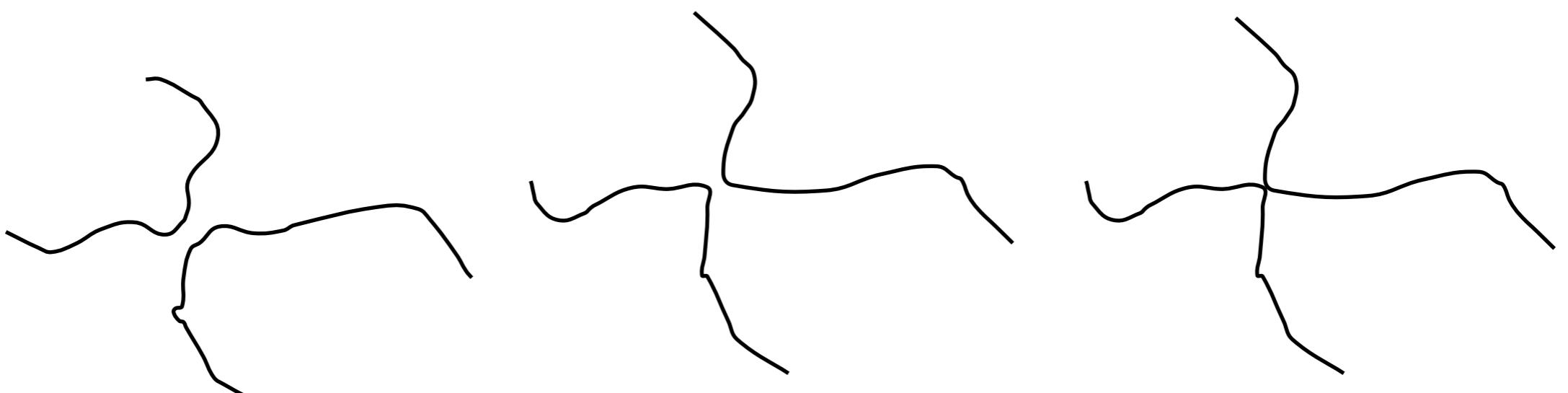
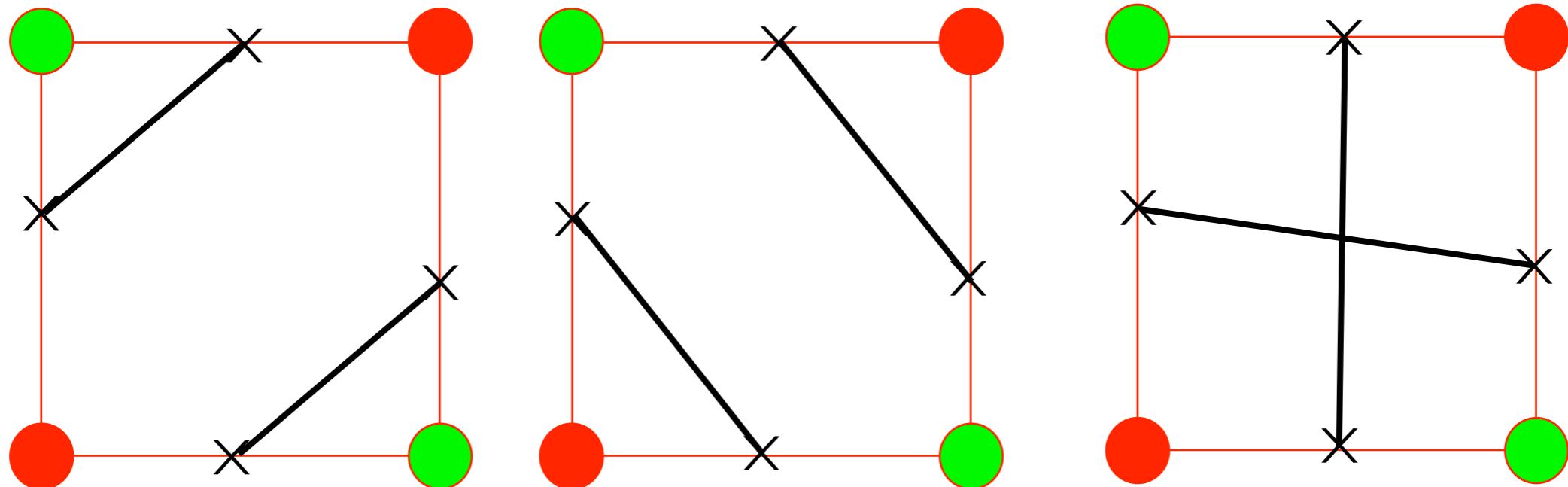
Ambiguities

- Right or Wrong?

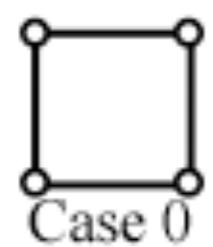


Ambiguities

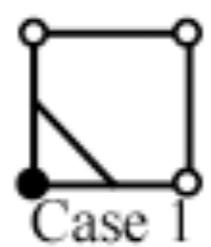
- Right or Wrong?



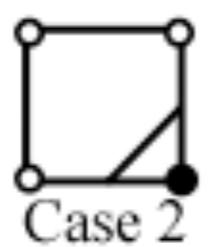
More on this later... let's go to 3D!



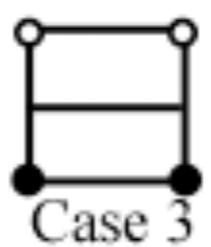
Case 0



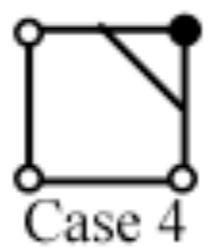
Case 1



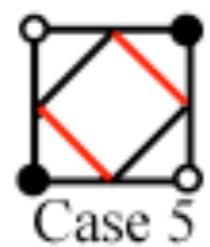
Case 2



Case 3



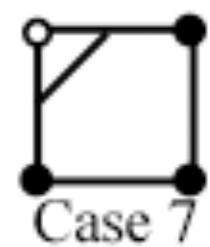
Case 4



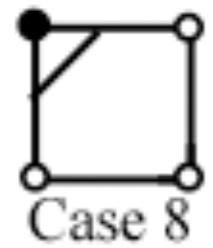
Case 5



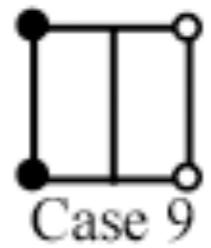
Case 6



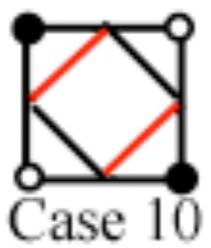
Case 7



Case 8



Case 9



Case 10



Case 11



Case 12



Case 13



Case 14

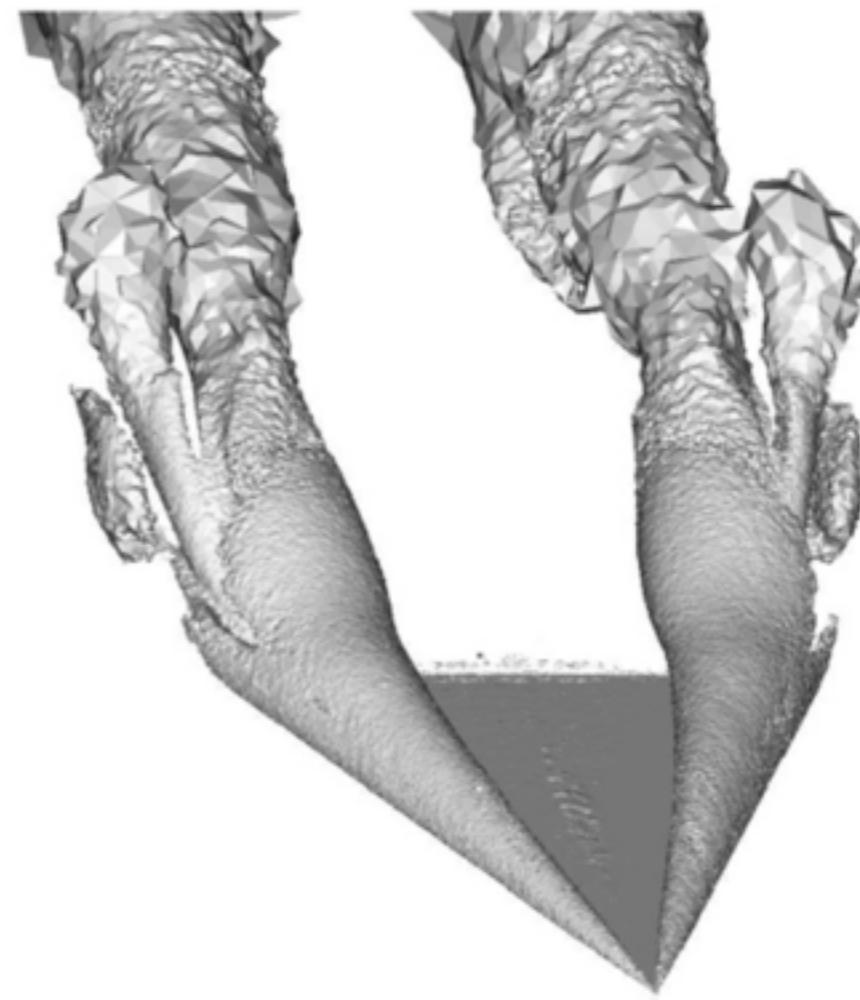
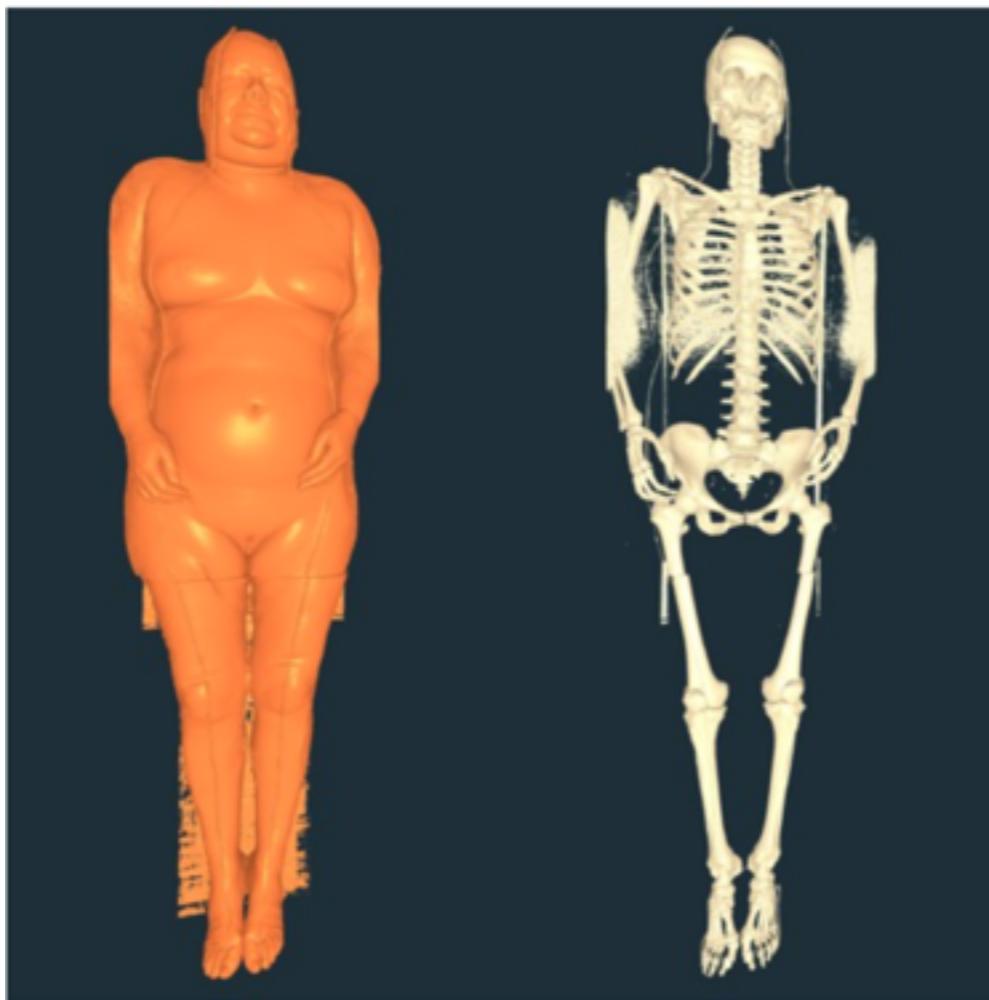


Case 15

Isosurfaces

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 $F(x,y,z) = v$.



Level sets

- It's easier to use some mathematical terminology to generalize contours.
- A **level set** of a function $f : R^n \rightarrow R$ is the set of points \mathbf{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**,

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

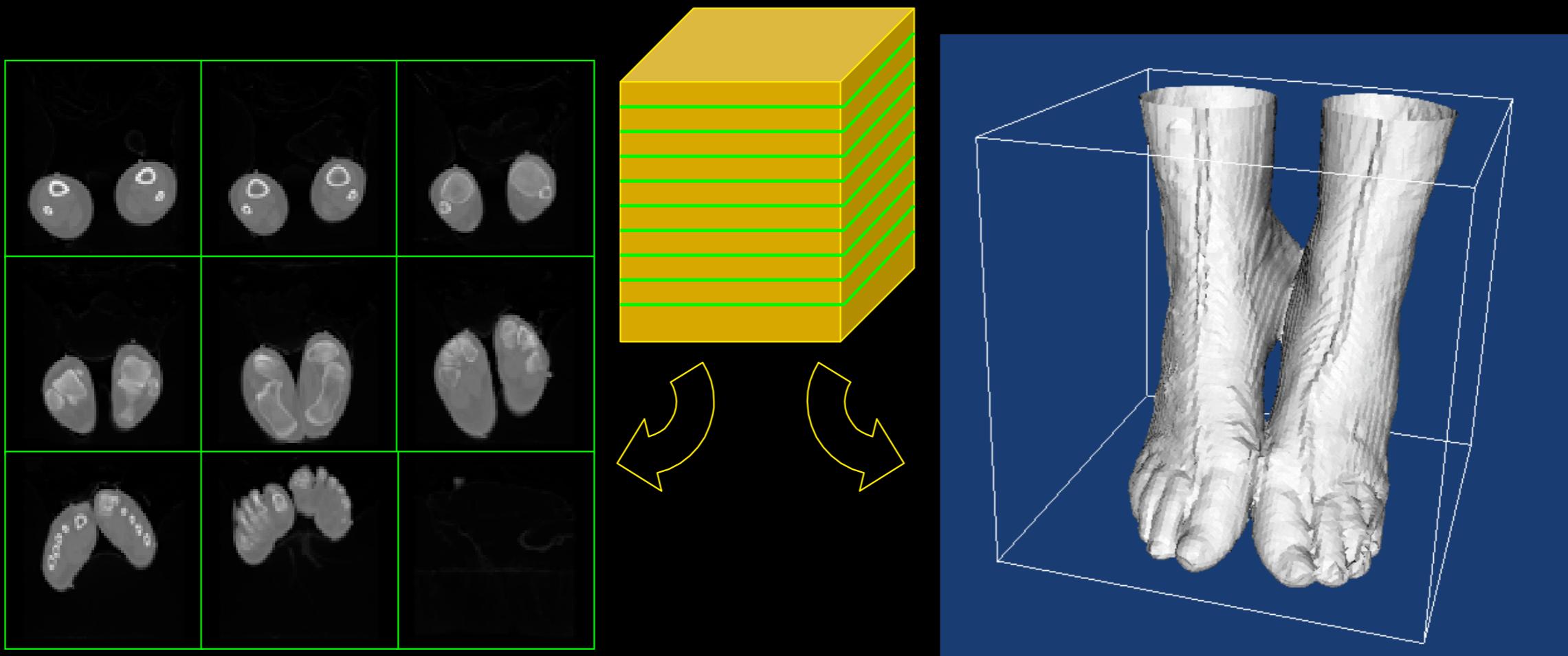
- and the **superlevel set**,

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

both bounded manifolds in R^n .

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.

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

Figure 17. 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 produces multiple 2D slices of information. The data sam-

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.

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 il-

Nov 10, 2015: 11814 cites on Google Scholar!

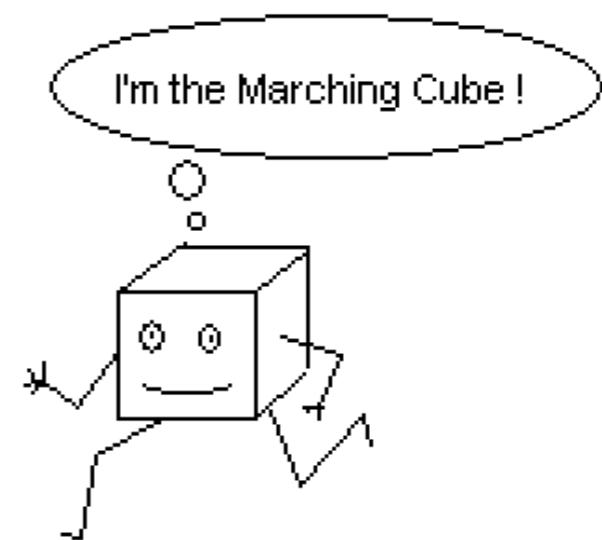
from multiple 2D slices of computed tomography (CT), magnetic resonance (MR), and single-photon emission computed tomography (SPECT), help physicians to understand the complex anatomy present in the slices. Interpretation of 2D

1. Data acquisition.

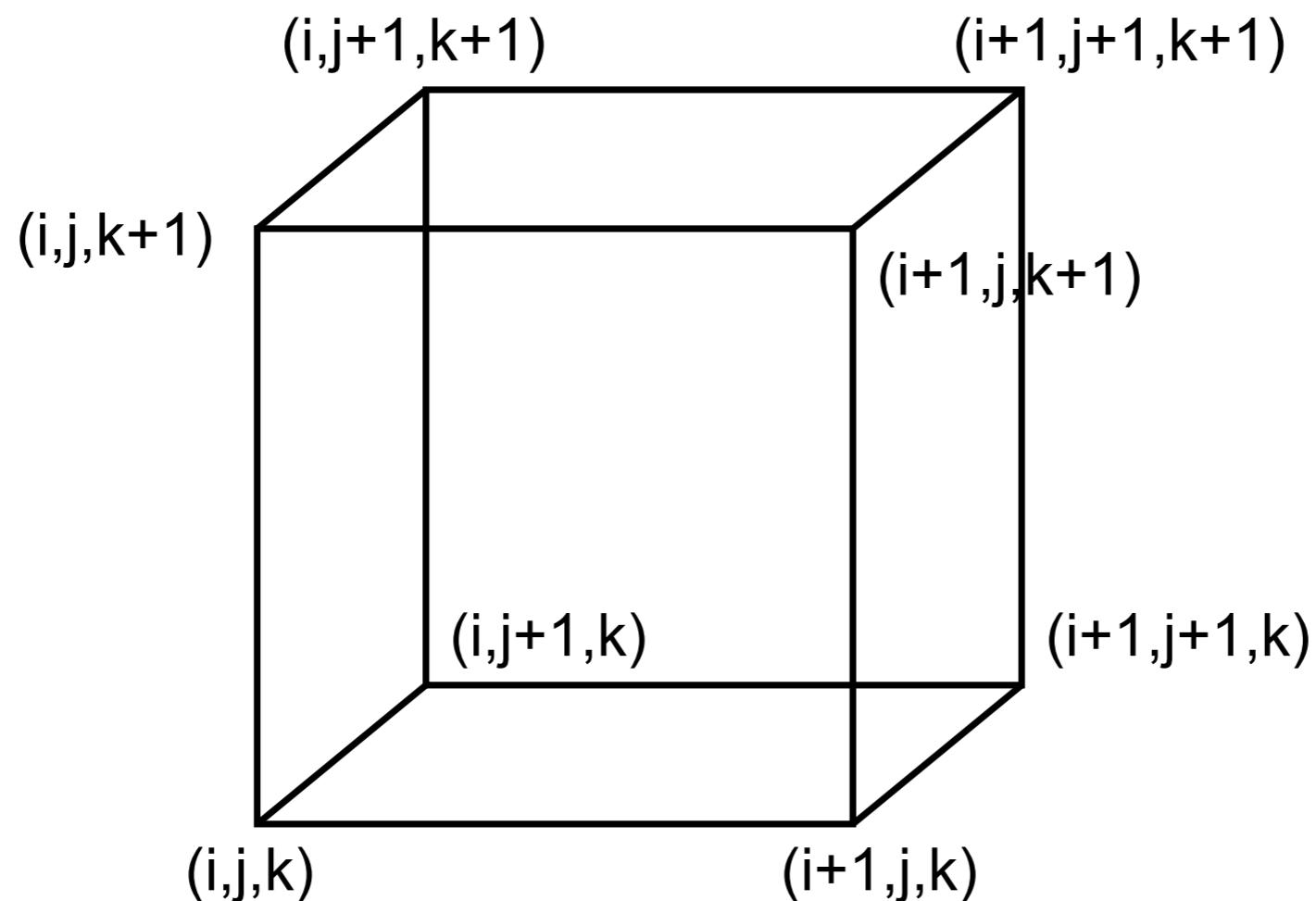
This first step, performed by the medical imaging hardware, samples some property in a patient and produces multiple 2D slices of information. The data sam-

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

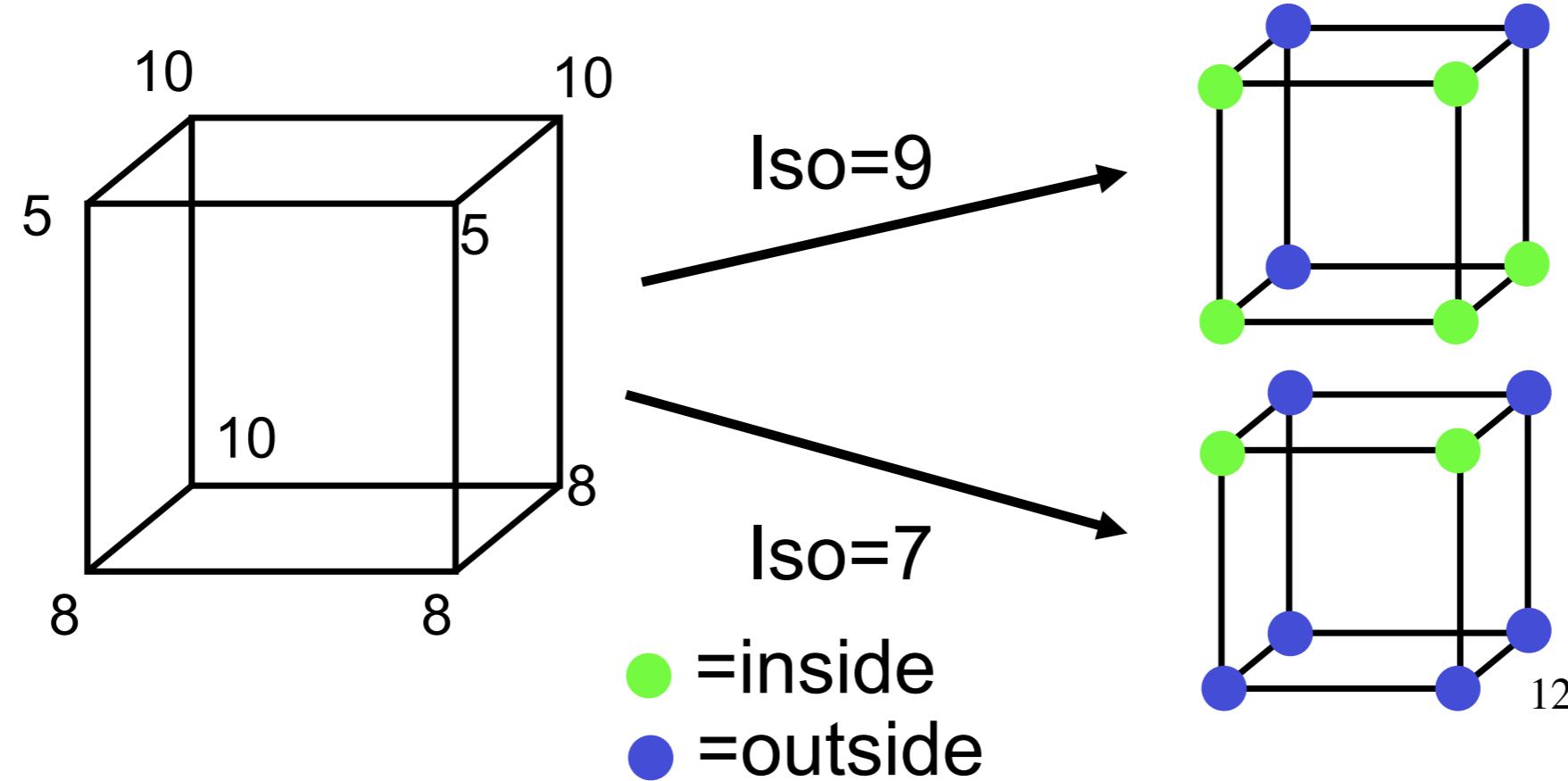
1. Consider a cell
2. Classify each vertex as inside or outside
3. Build an index
4. Get edge list from table[index]
5. Interpolate the edge location
6. Compute gradients
7. Consider ambiguous cases
8. Go to next cell



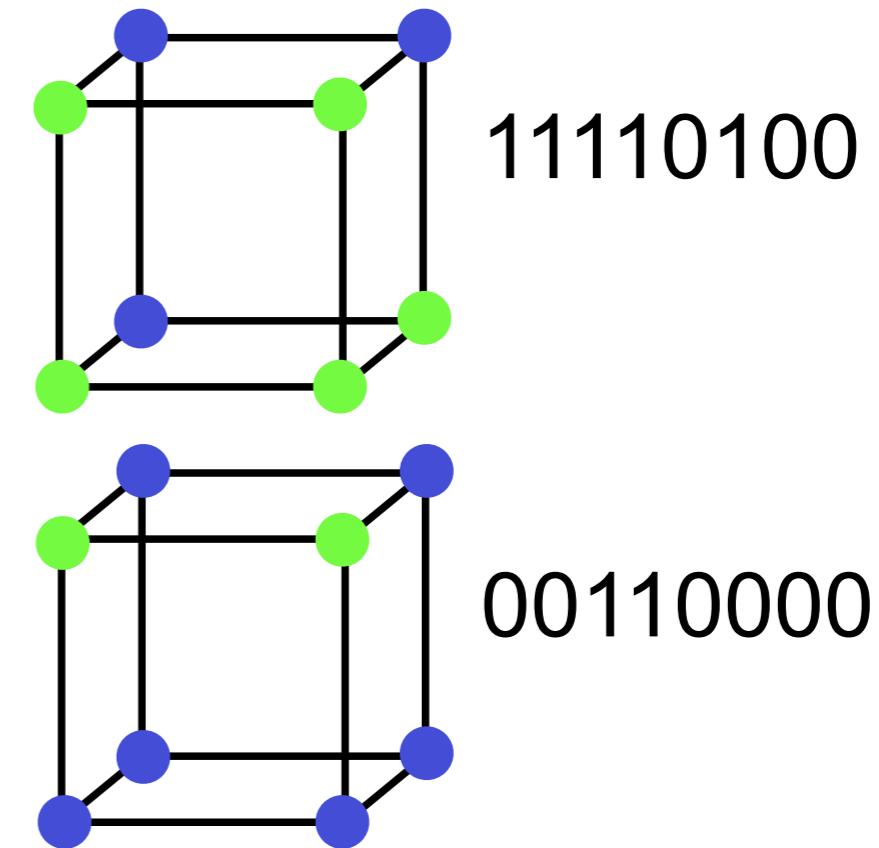
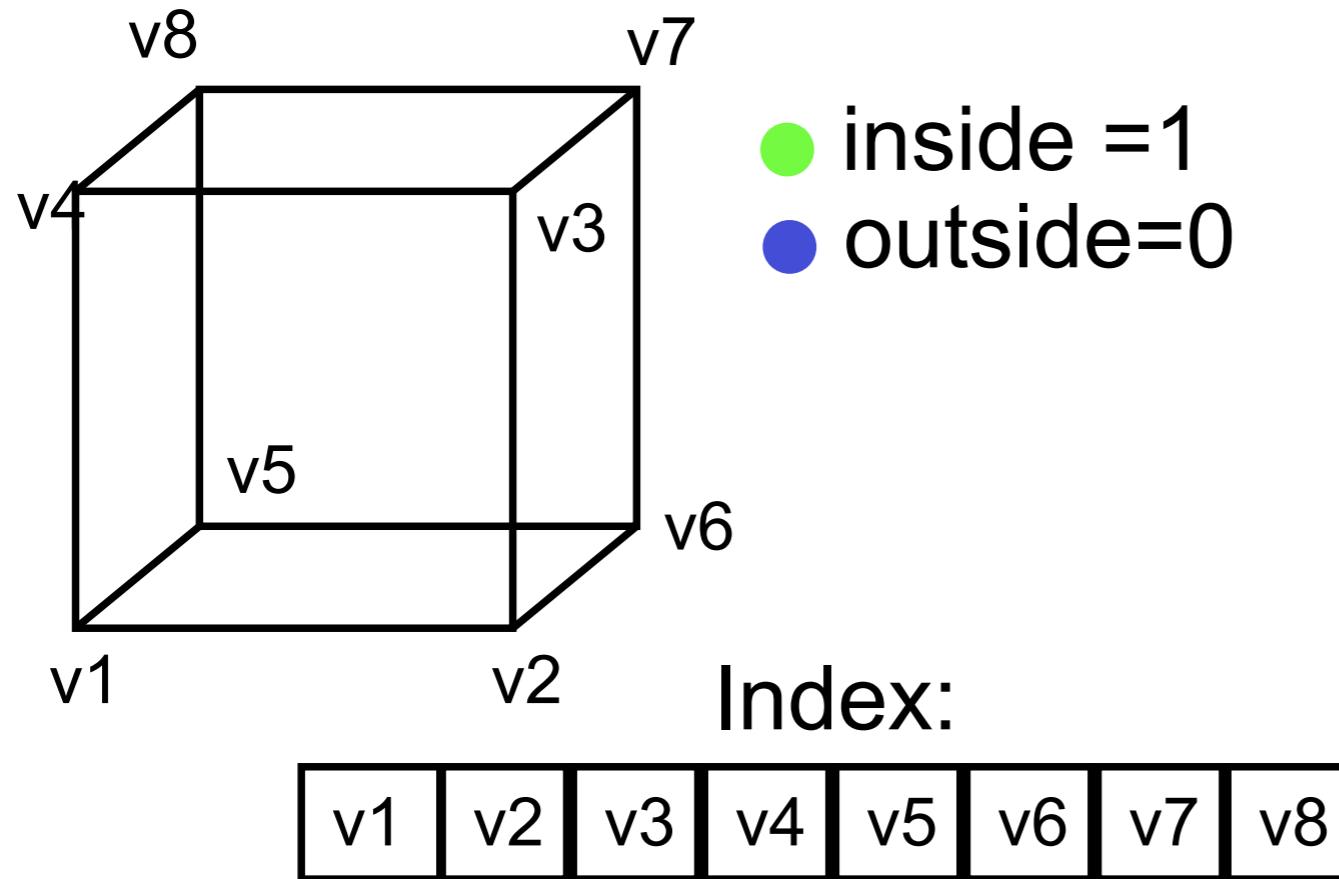
- Step 1: Consider a cell defined by eight data values



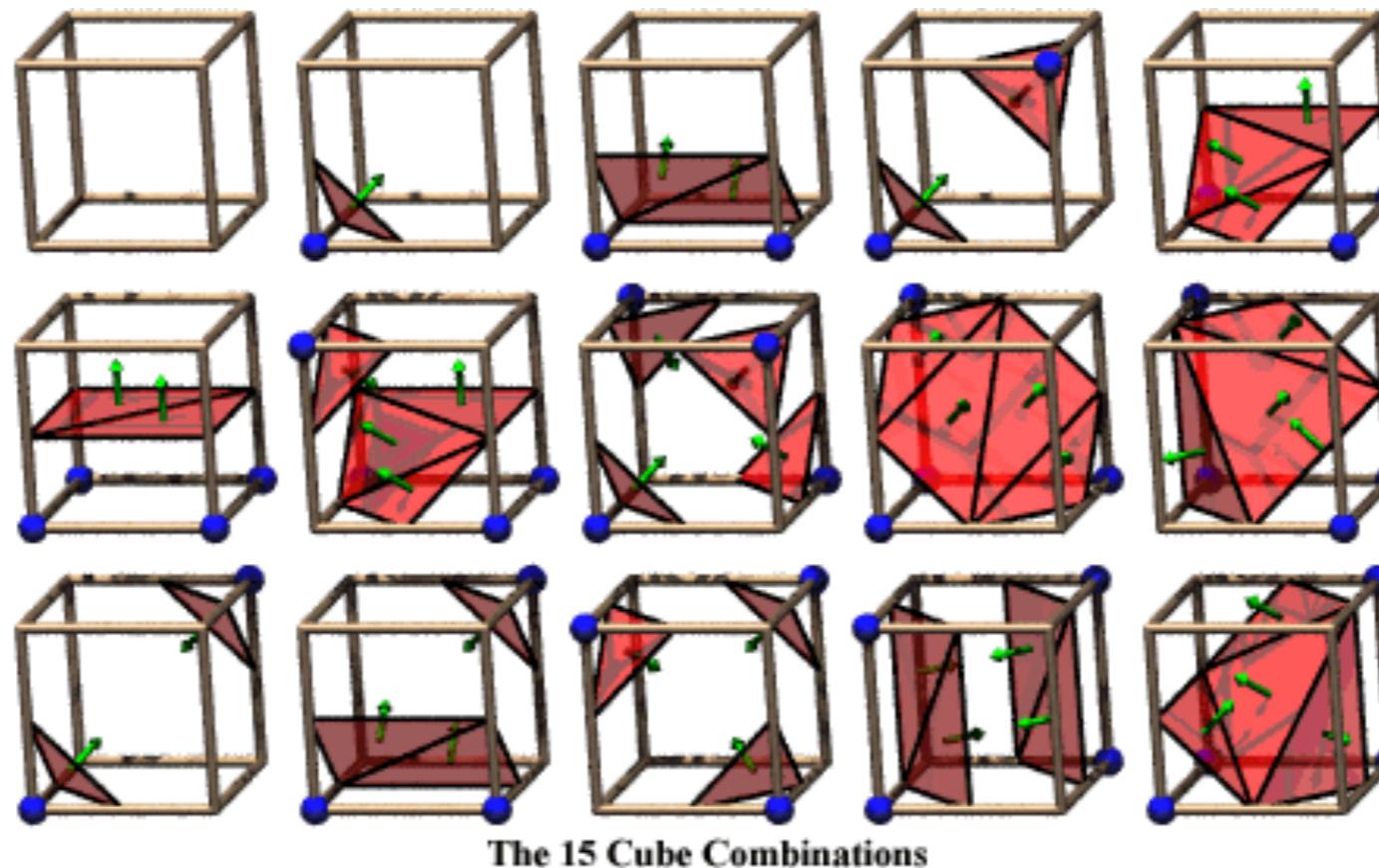
- Step 2: Classify each voxel according to whether it lies
 - Outside the surface ($\text{value} > \text{isosurface value}$)
 - Inside the surface ($\text{value} \leq \text{isosurface value}$)



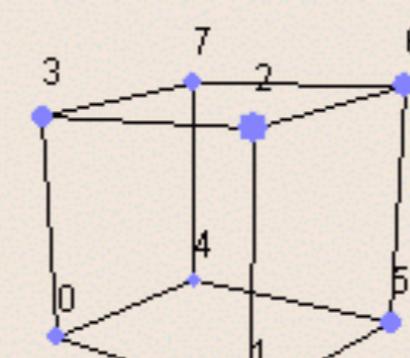
- Step 3: Use the binary labeling of each voxel to create an index



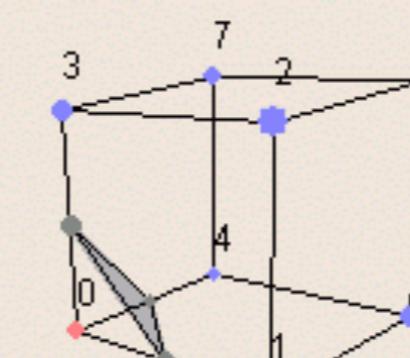
- Step 4: For a given index, access an array storing a list of edges
 - All 256 cases can be derived from $1+14=15$ base cases due to symmetries



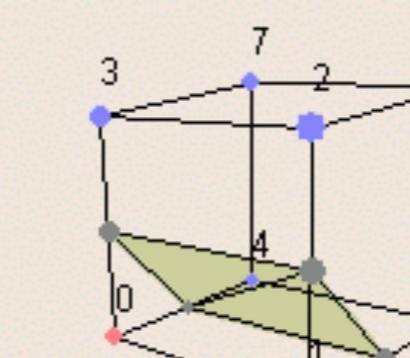
Case Table



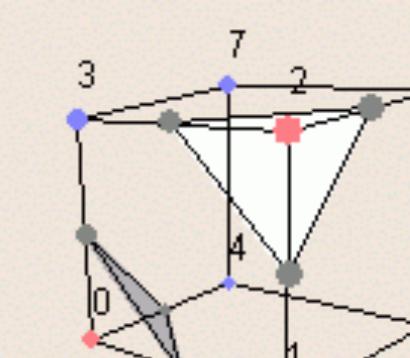
Case 0



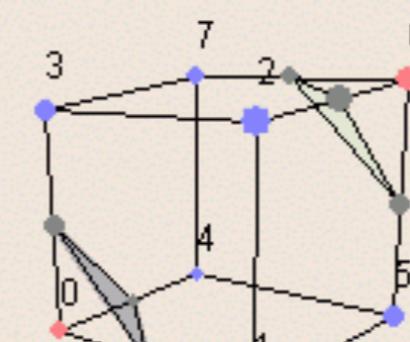
Case 1



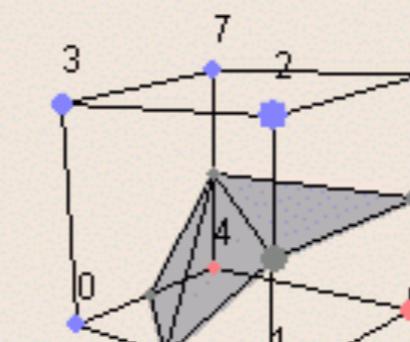
Case 2



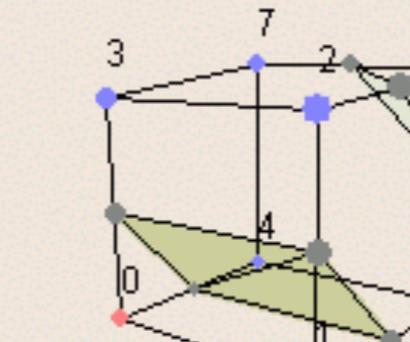
Case 3



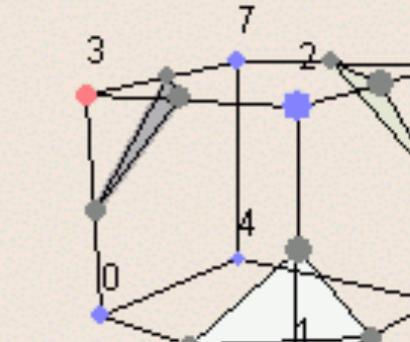
Case 4



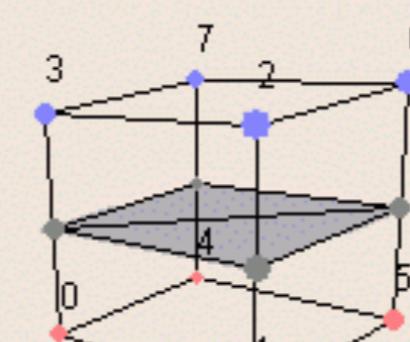
Case 5



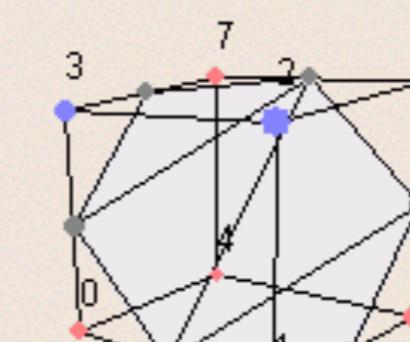
Case 6



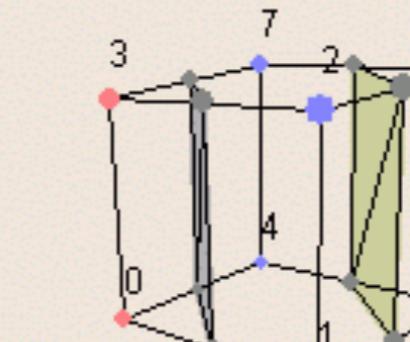
Case 7



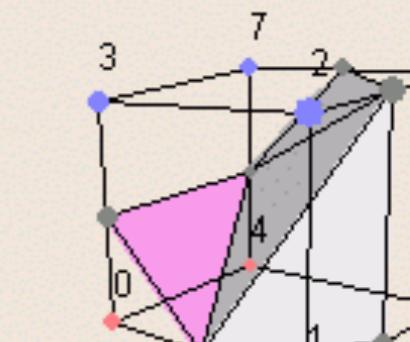
Case 8



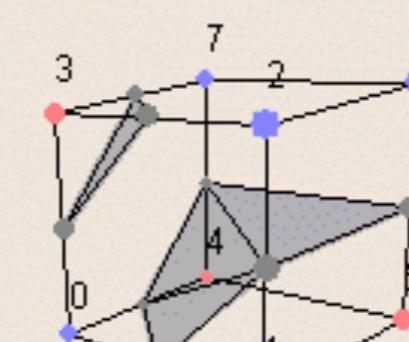
Case 9



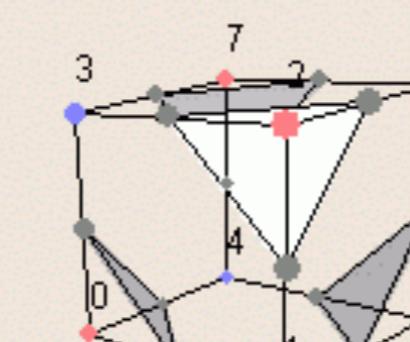
Case 10



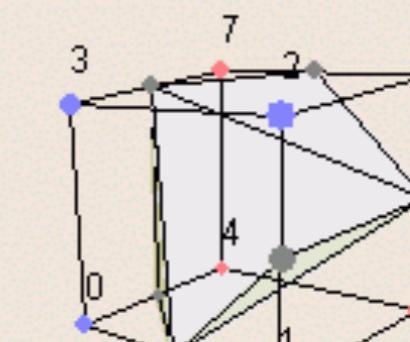
Case 11



Case 12



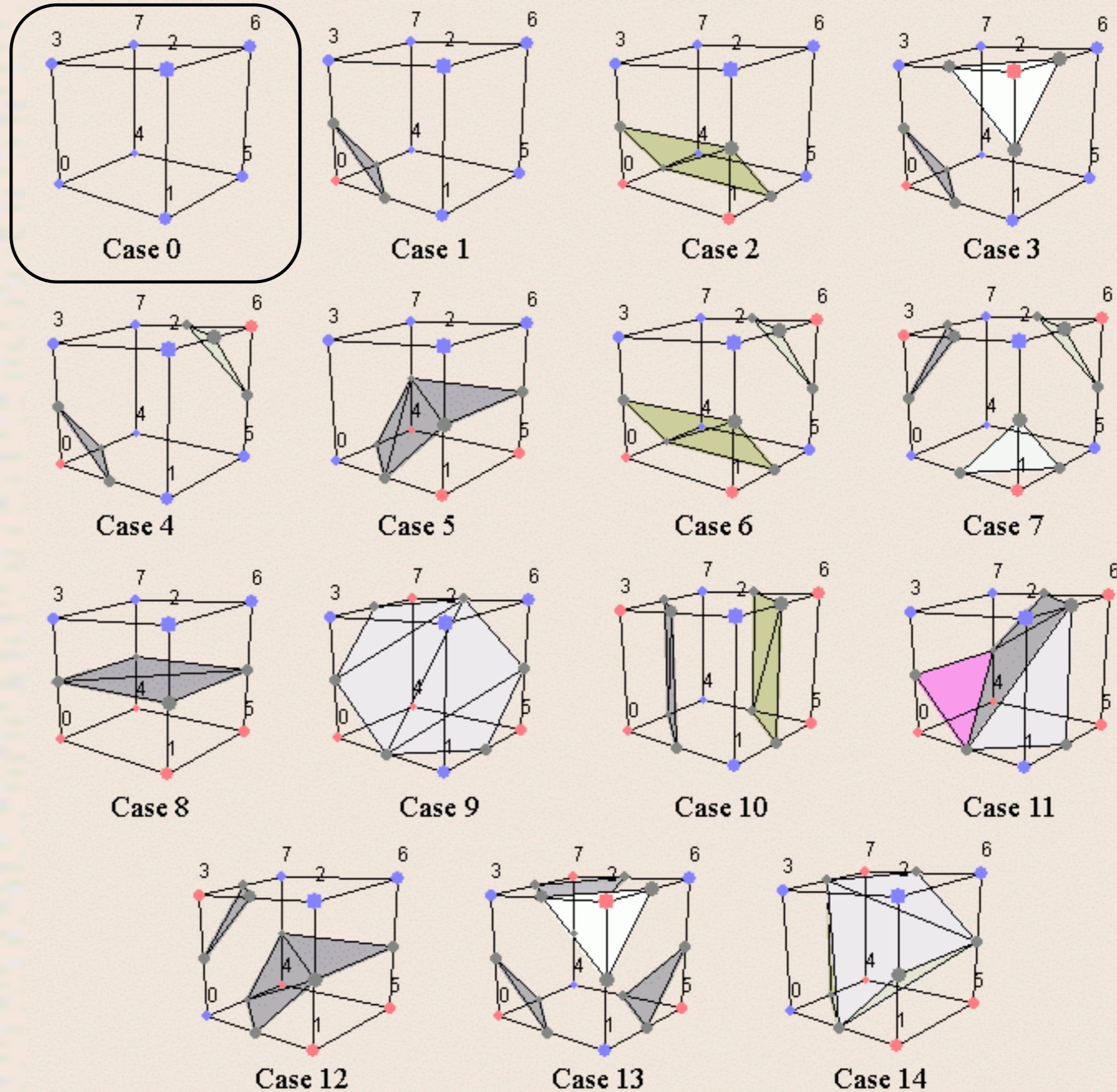
Case 13



Case 14

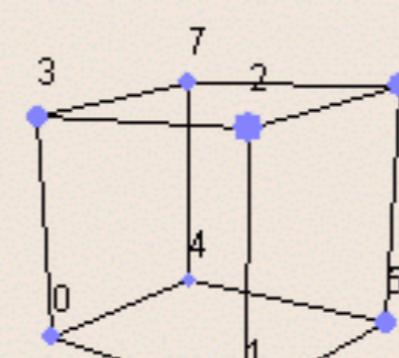
8 Above
0 Below

1 case

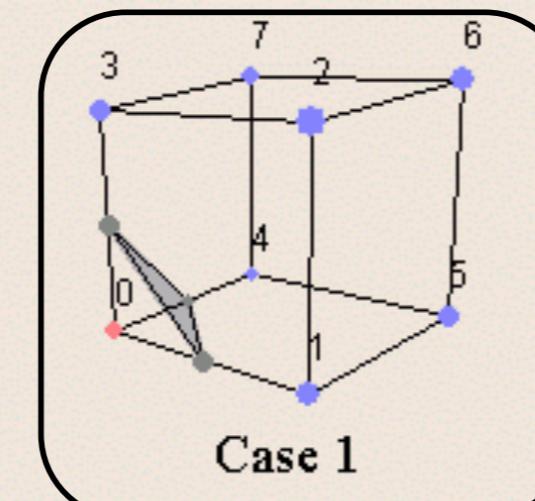


• 7 Above
• 1 Below

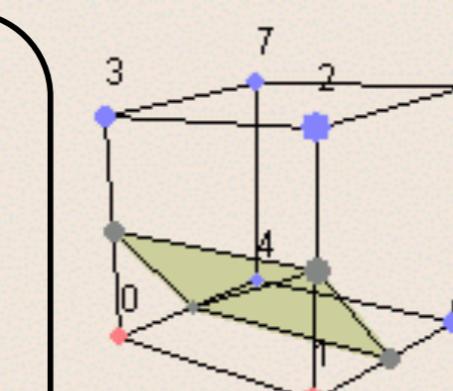
1 case



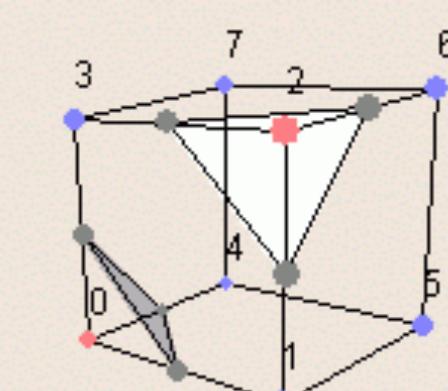
Case 0



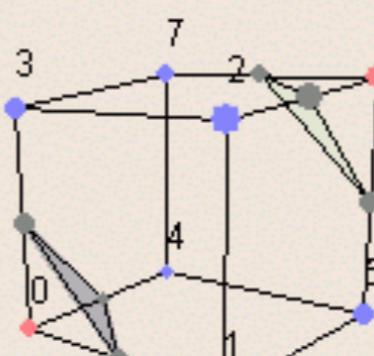
Case 1



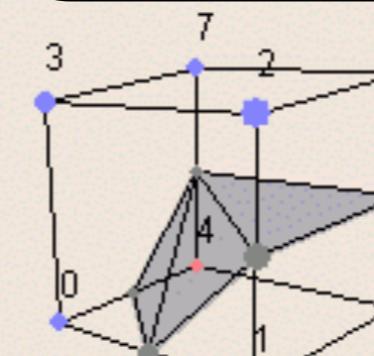
Case 2



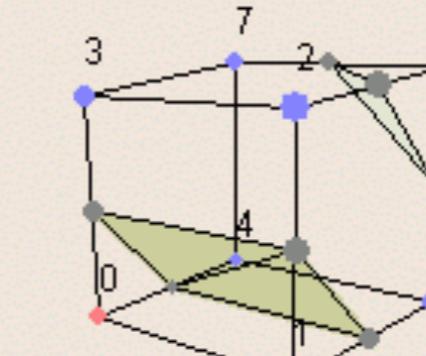
Case 3



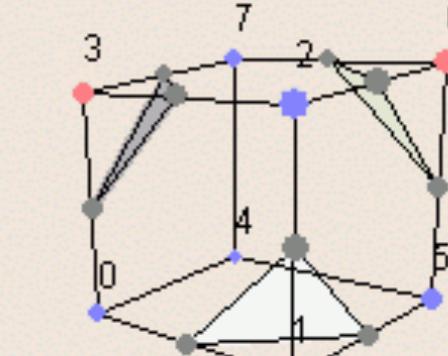
Case 4



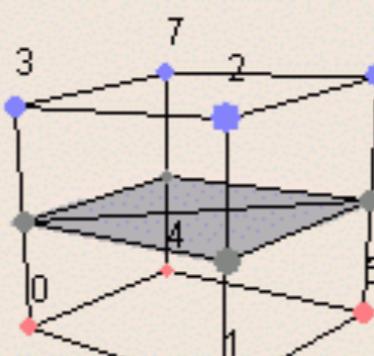
Case 5



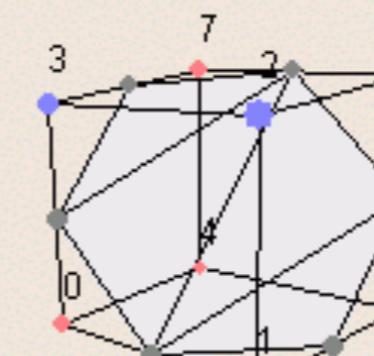
Case 6



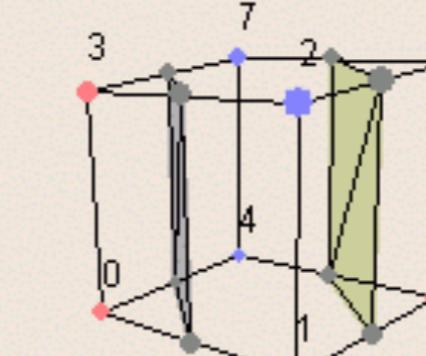
Case 7



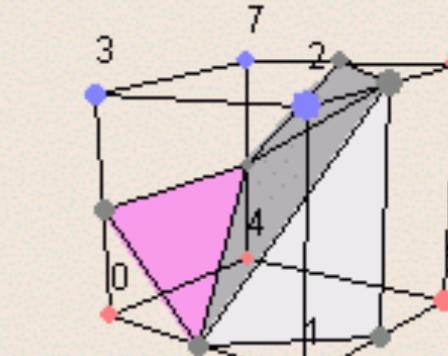
Case 8



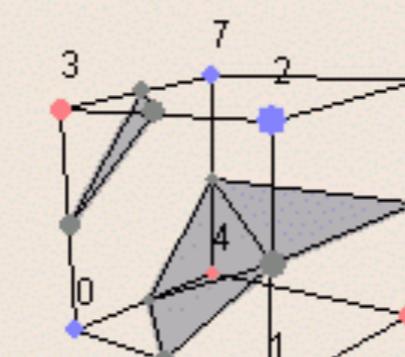
Case 9



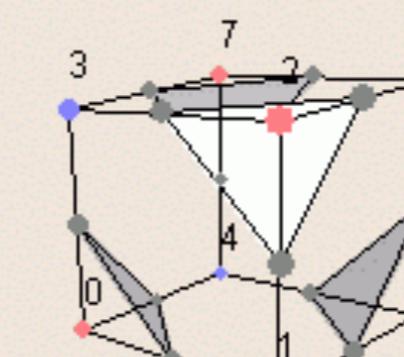
Case 10



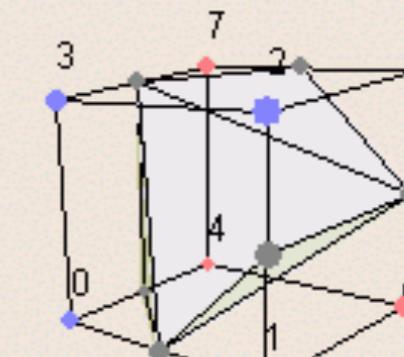
Case 11



Case 12



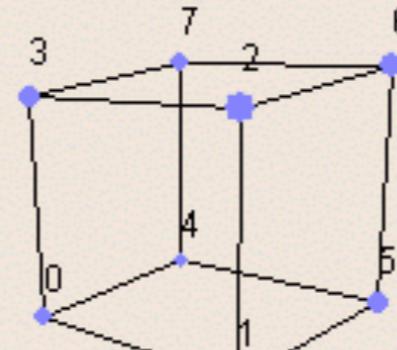
Case 13



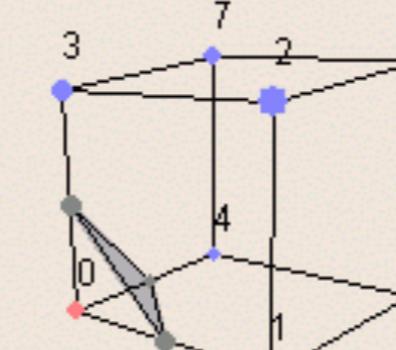
Case 14

- 6 Above
- 2 Below

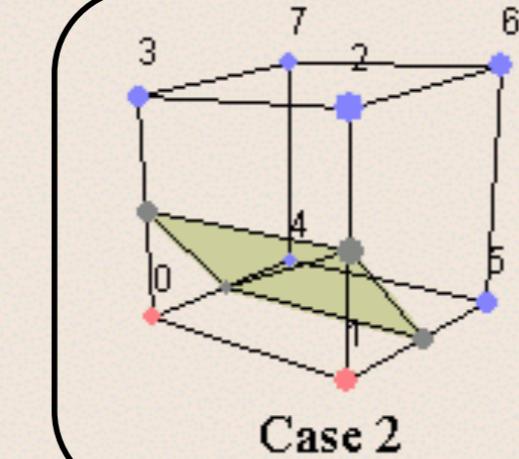
3 cases



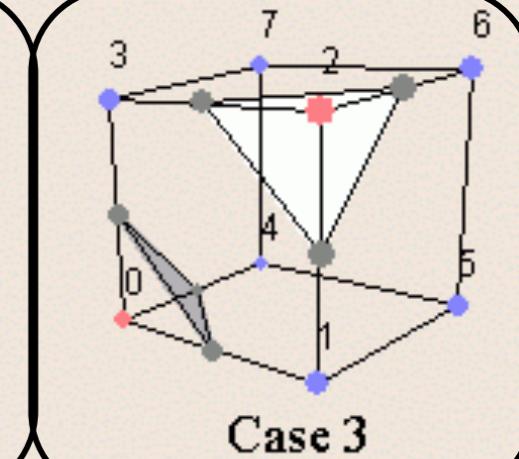
Case 0



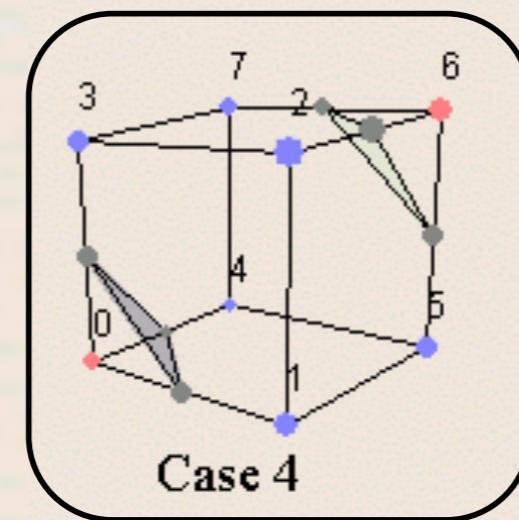
Case 1



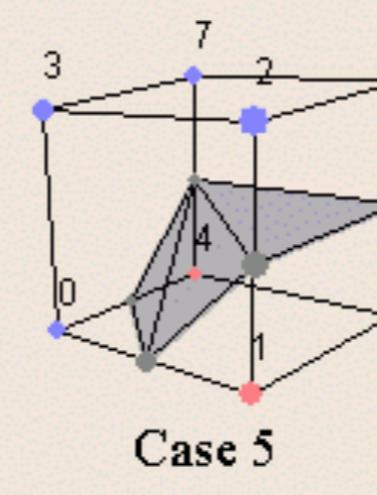
Case 2



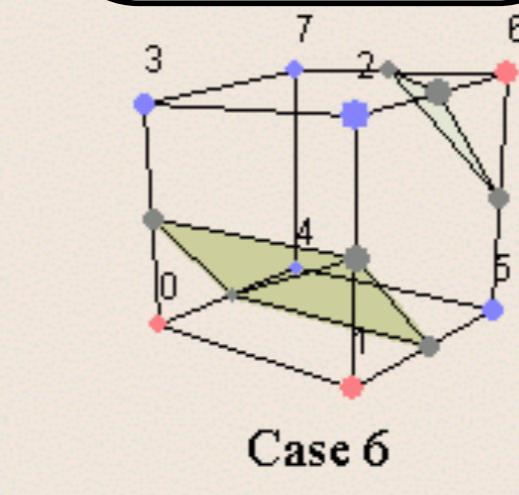
Case 3



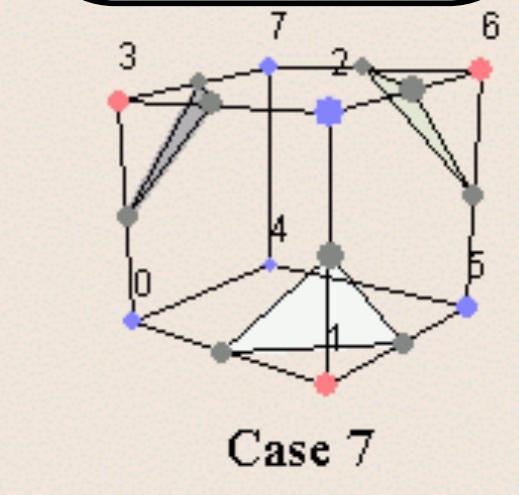
Case 4



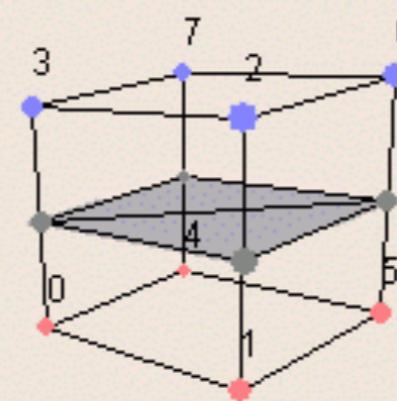
Case 5



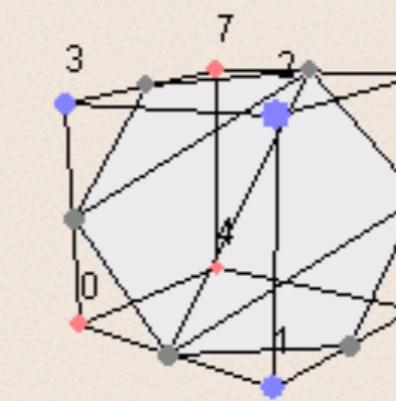
Case 6



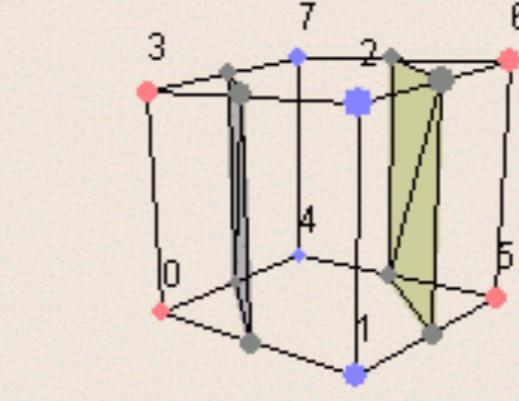
Case 7



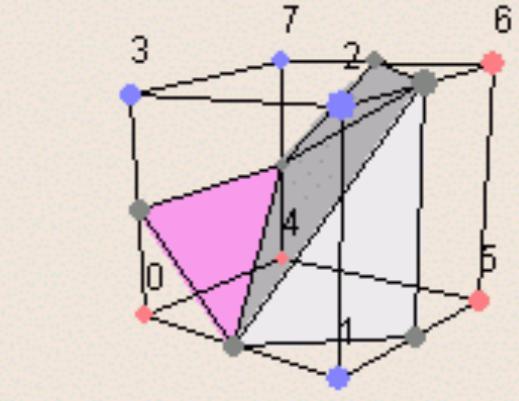
Case 8



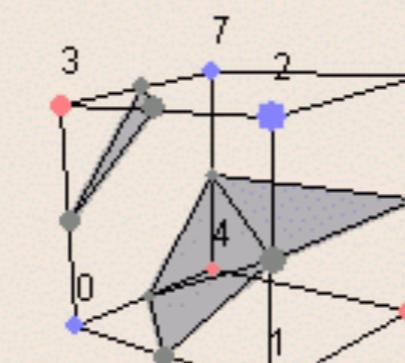
Case 9



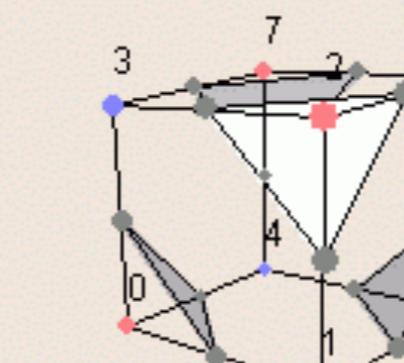
Case 10



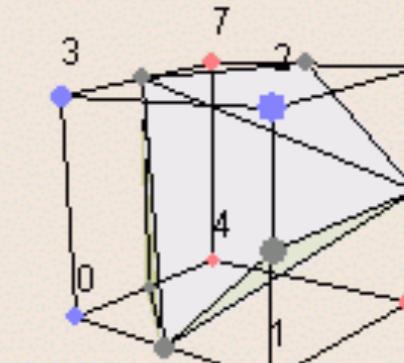
Case 11



Case 12



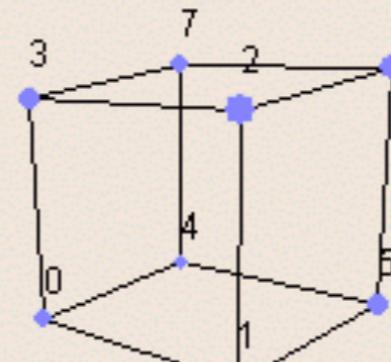
Case 13



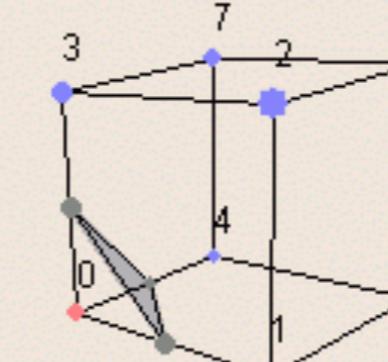
Case 14

- 5 Above
- 3 Below

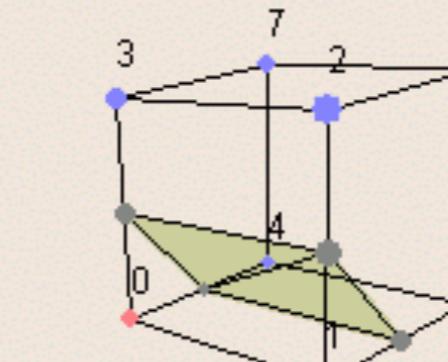
3 cases



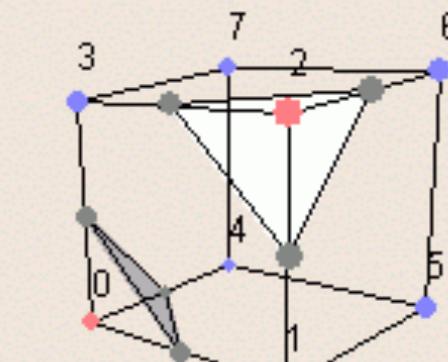
Case 0



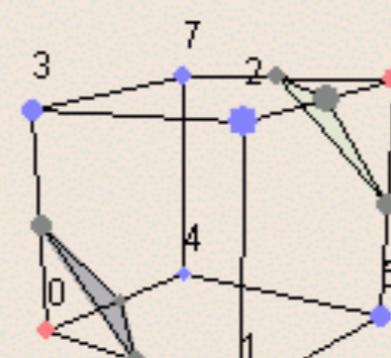
Case 1



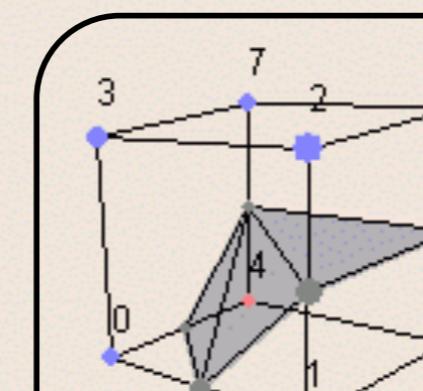
Case 2



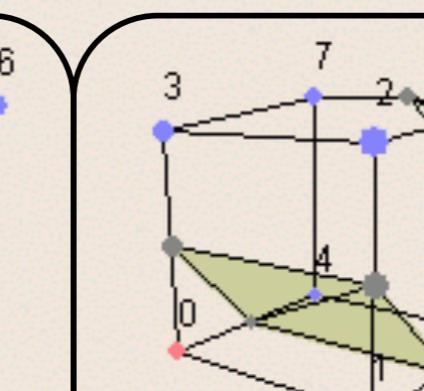
Case 3



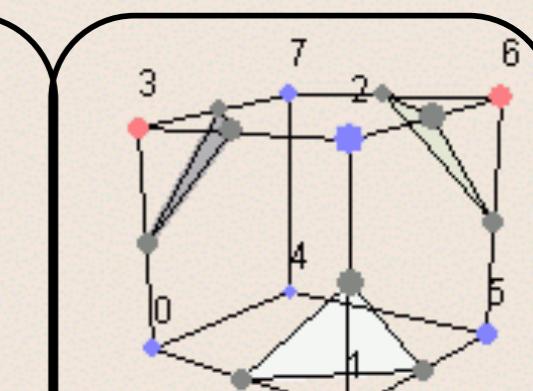
Case 4



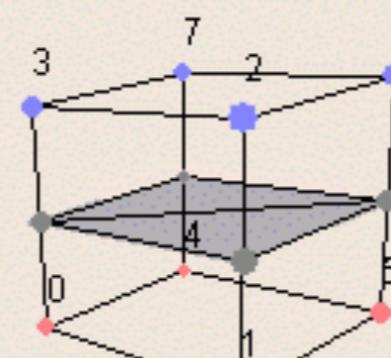
Case 5



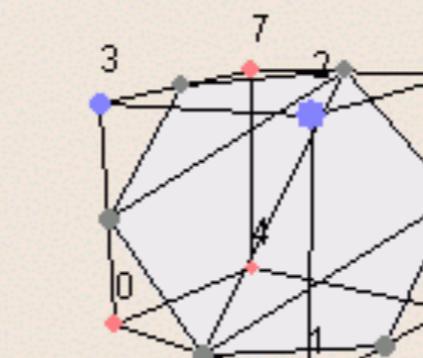
Case 6



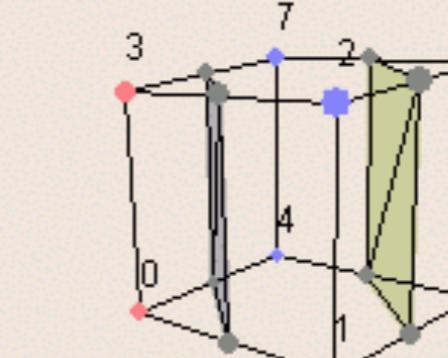
Case 7



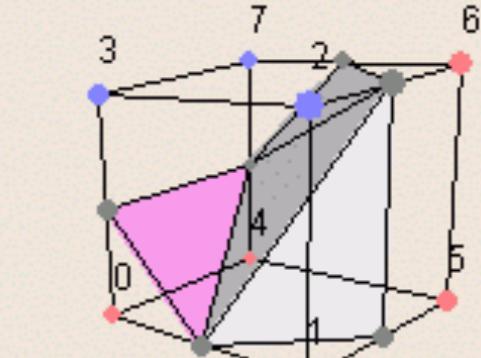
Case 8



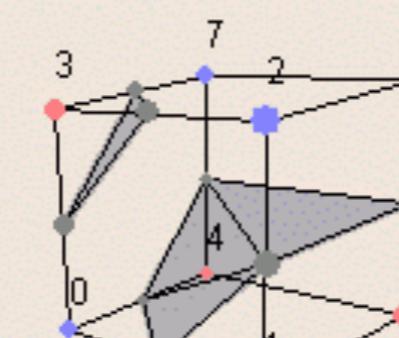
Case 9



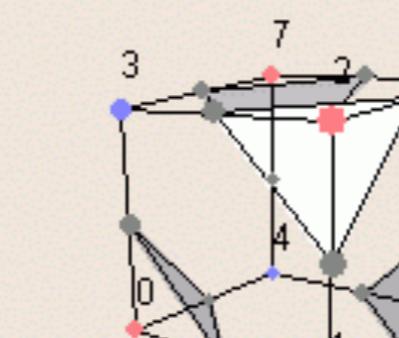
Case 10



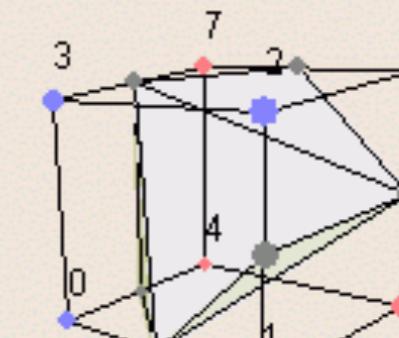
Case 11



Case 12



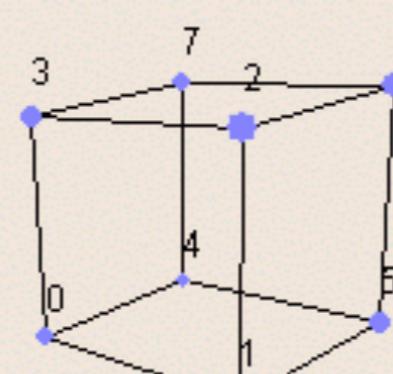
Case 13



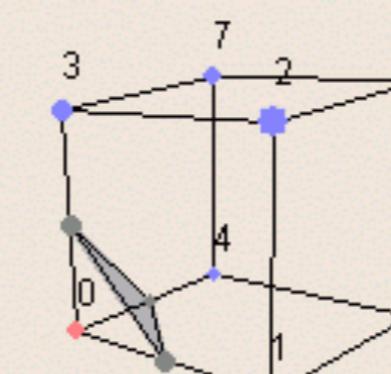
Case 14

- 4 Above
- 4 Below

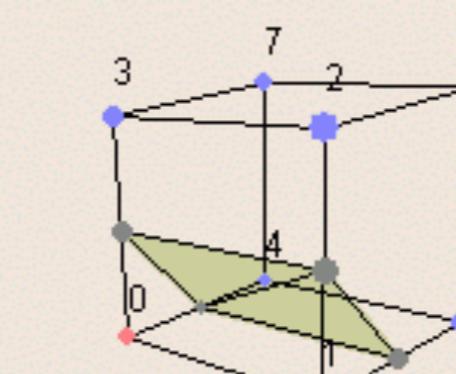
7 cases



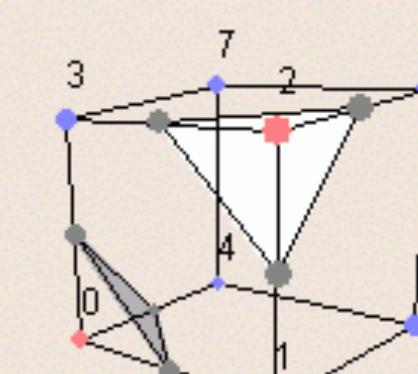
Case 0



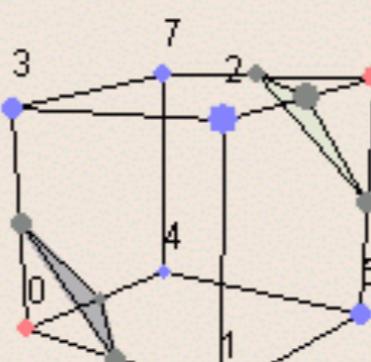
Case 1



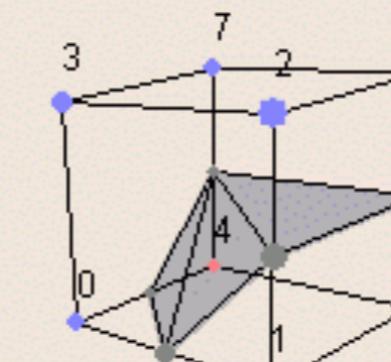
Case 2



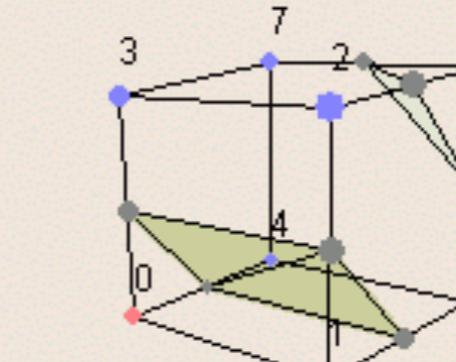
Case 3



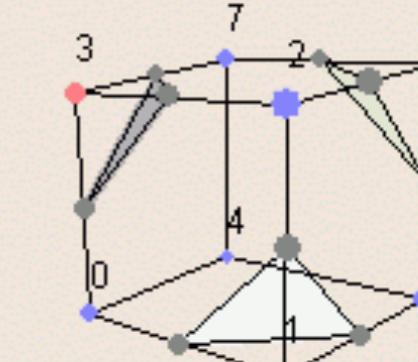
Case 4



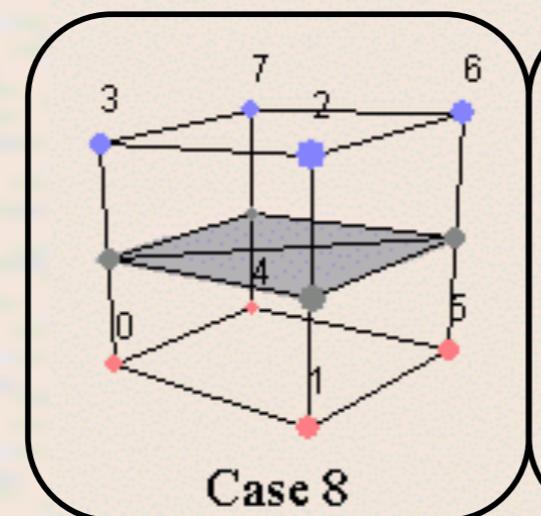
Case 5



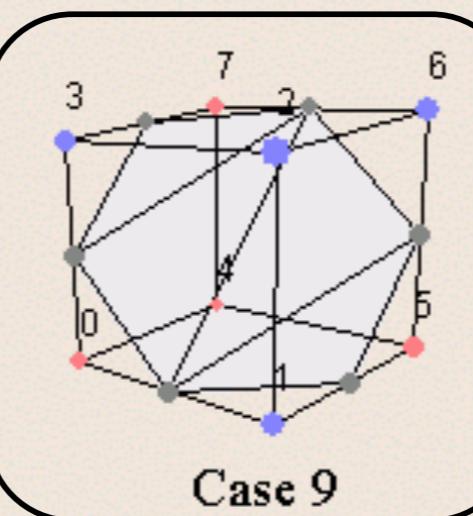
Case 6



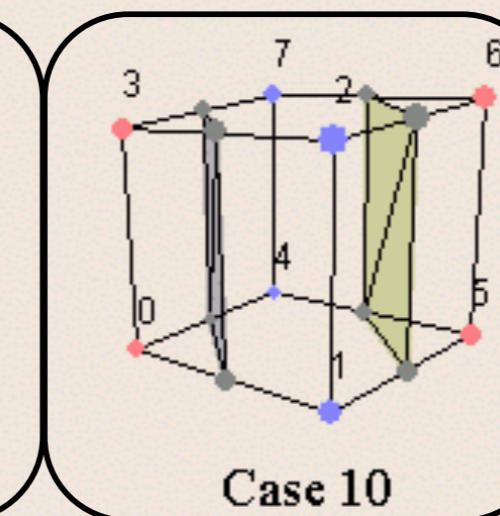
Case 7



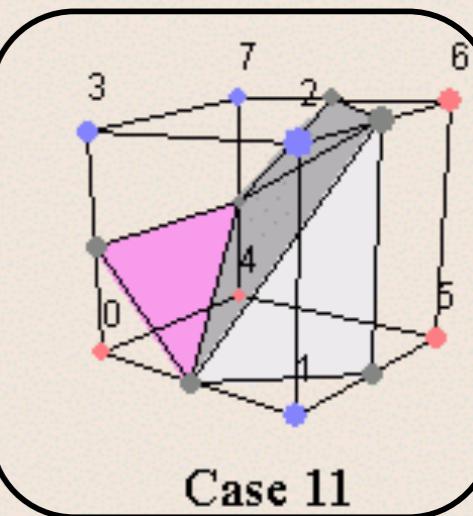
Case 8



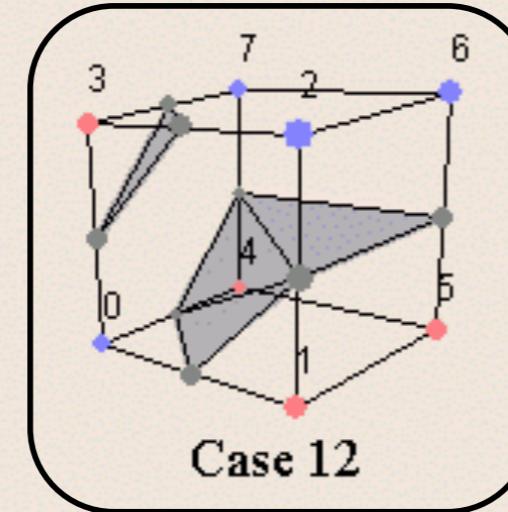
Case 9



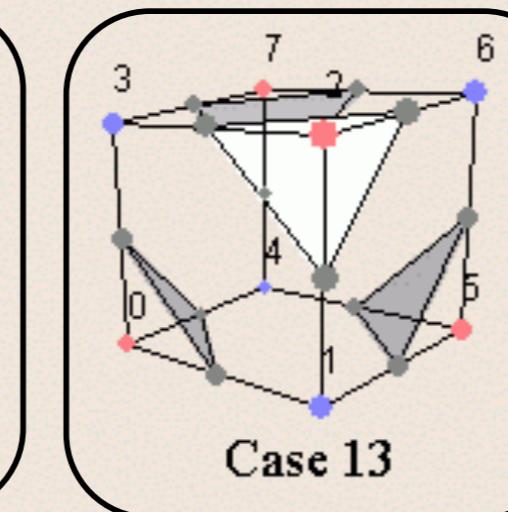
Case 10



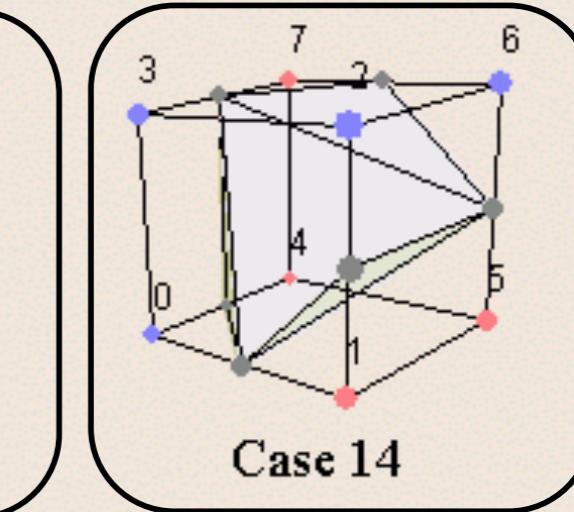
Case 11



Case 12



Case 13



Case 14

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

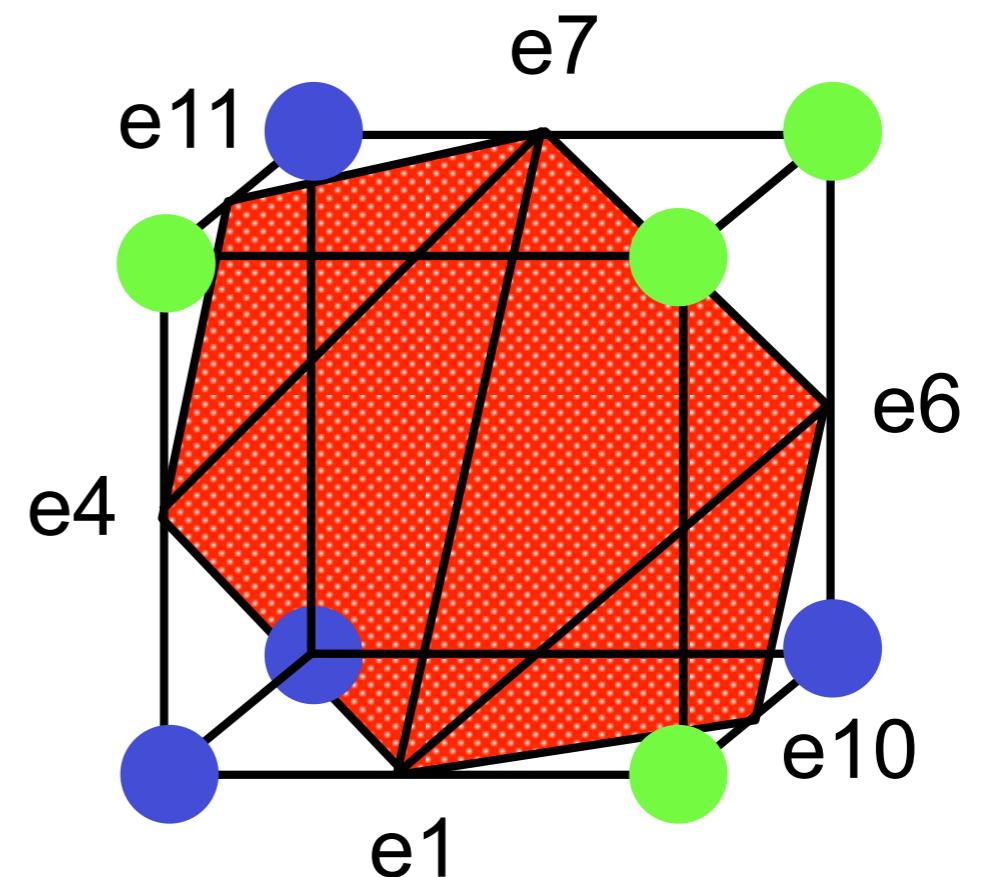
Index = 10110001

triangle 1 = e4,e7,e11

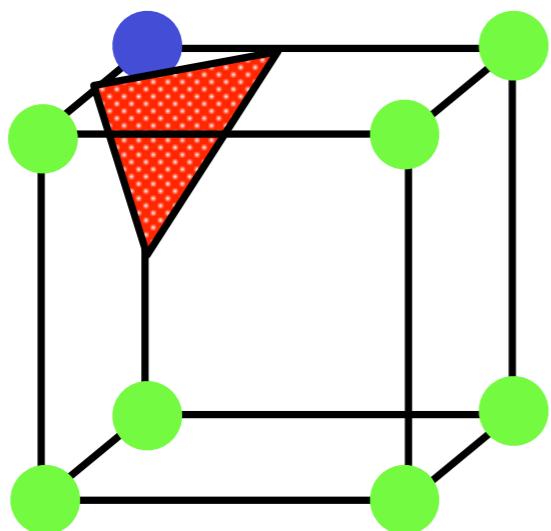
triangle 2 = e1, e7, e4

triangle 3 = e1, e6, e7

triangle 4 = e1, e10, e6

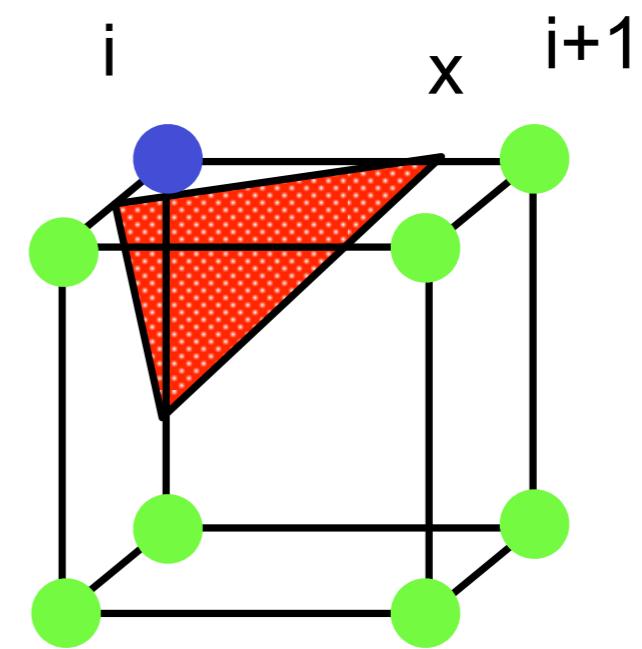


- Step 5: For each triangle edge, find the vertex location along the edge using linear interpolation of the voxel values



$T=5$

● = 10
● = 0



$T=8$

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

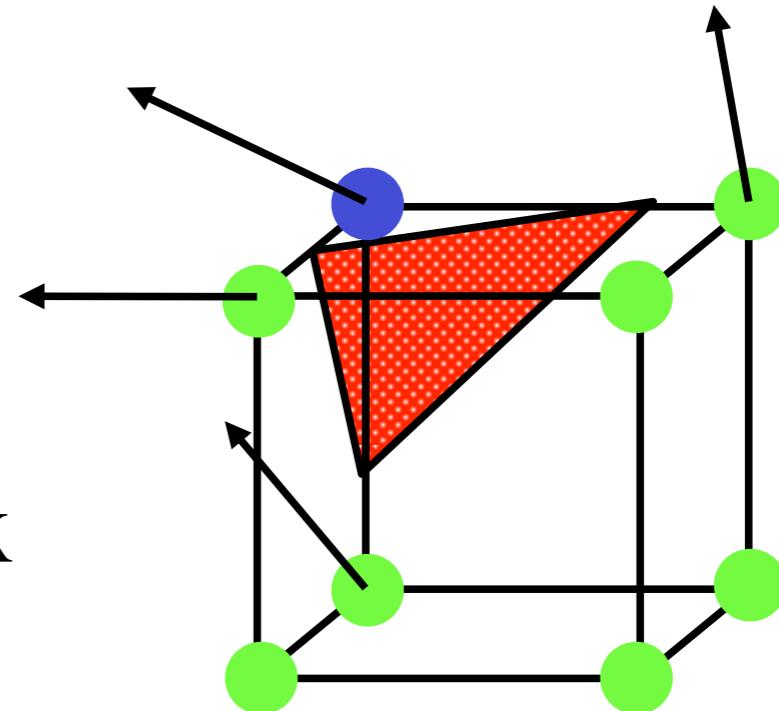
- Step 6: Calculate the normal at each cube vertex (central differences)

$$- G_x = V_{x+1,y,z} - V_{x-1,y,z}$$

$$G_y = V_{x,y+1,z} - V_{x,y-1,z}$$

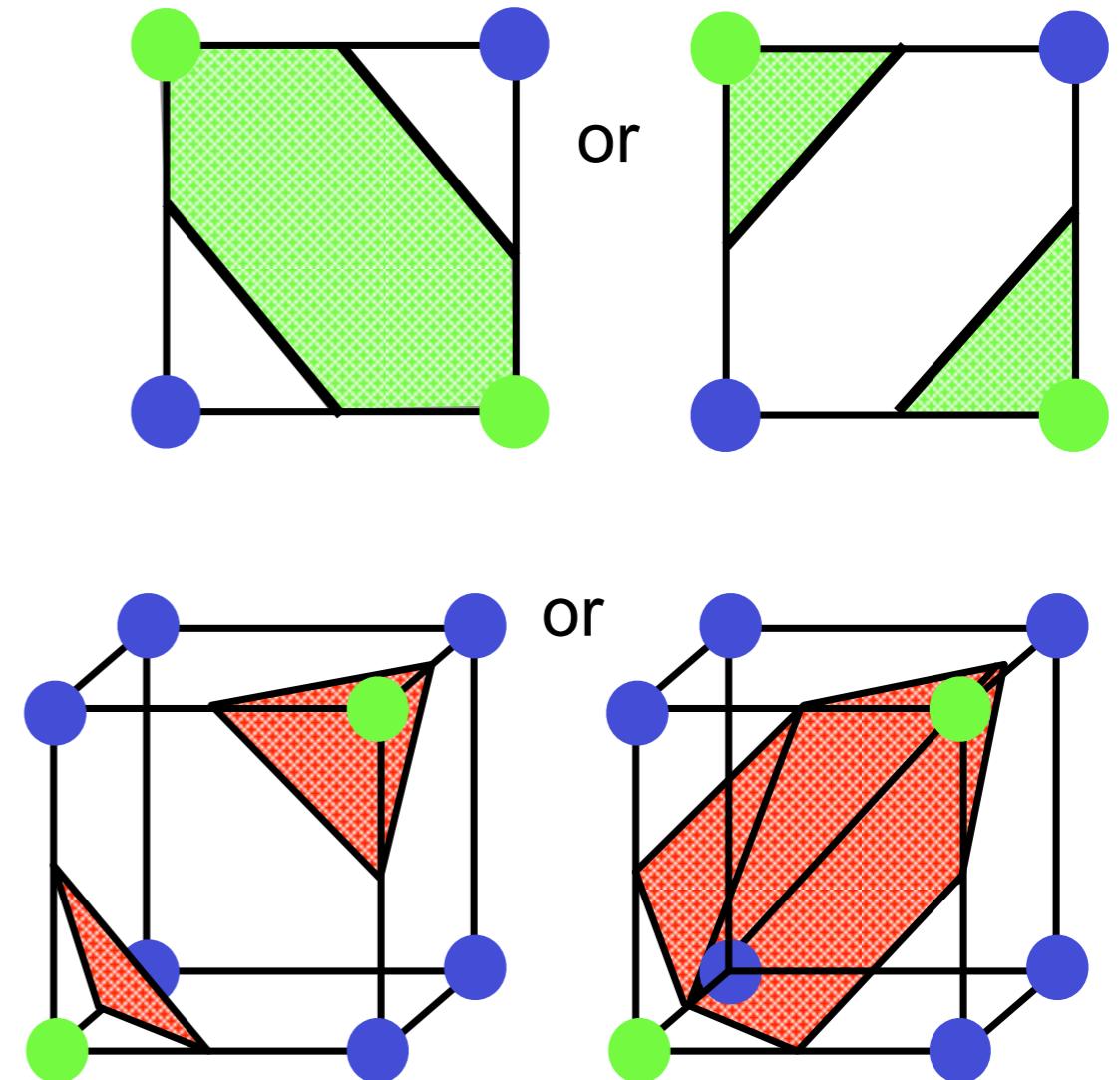
$$G_z = V_{x,y,z+1} - V_{x,y,z-1}$$

- 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

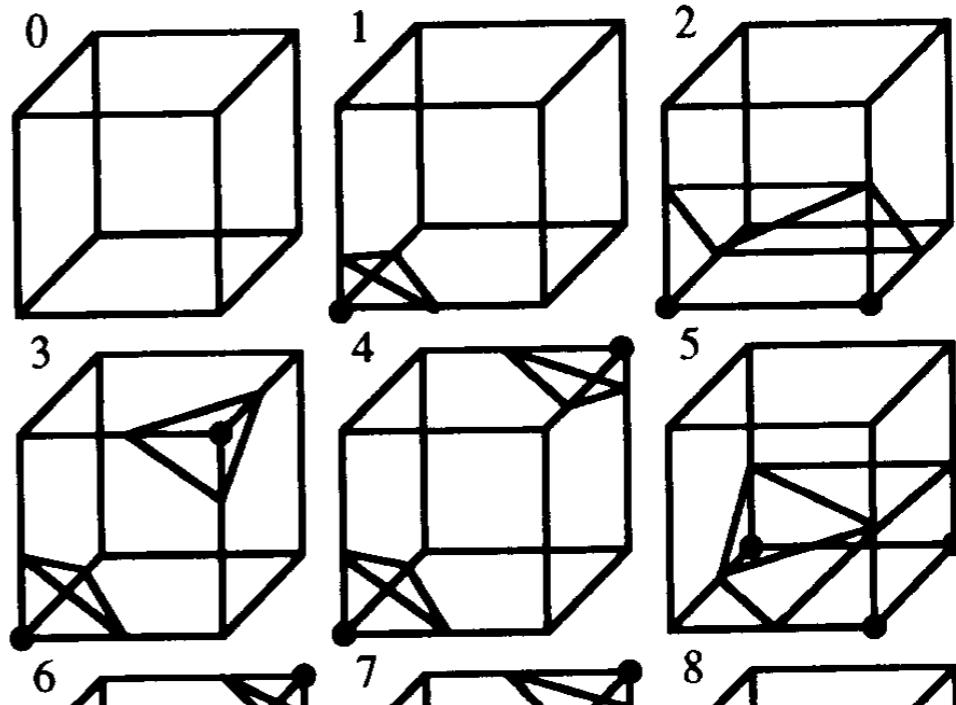
A method for computing isovalue or contour surfaces of a trivariate function is discussed. The input data are values of the trivariate function, F_{ijk} , at the cuberille grid points (x_i, y_j, z_k) and the output is a collection of triangles representing the surface consisting of all points where $F(x, y, z)$ 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 $F_{ijk} = F(x_i, y_j, z_k)$; $i = 1, \dots, N_x$, $j = 1, \dots, N_y$, $k = 1, \dots, N_z$. 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

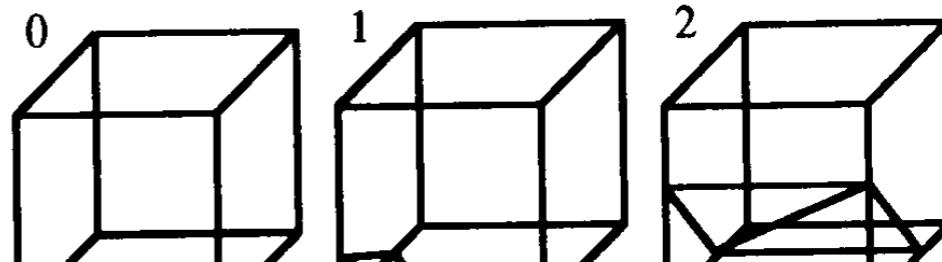
Abstract

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

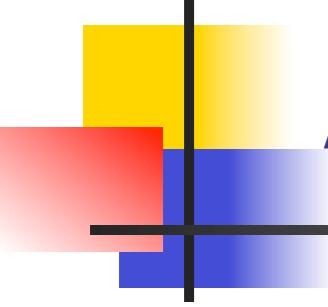
The purpose of this paper is to describe a method for computing contour or isovalue surfaces of a trivariate function $F(x, y, z)$. It is assumed that the function is continuous and that samples over a

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.



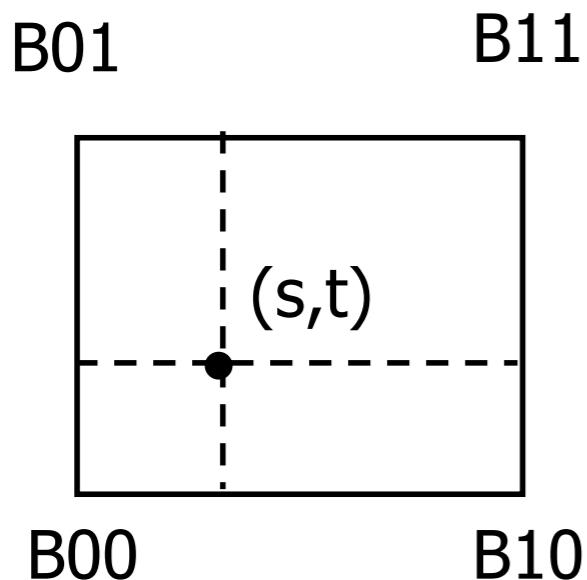
Nov 10, 2015: 626 cites on Google Scholar!





Asymptotic Decider (1)

- Based on bilinear interpolation over faces

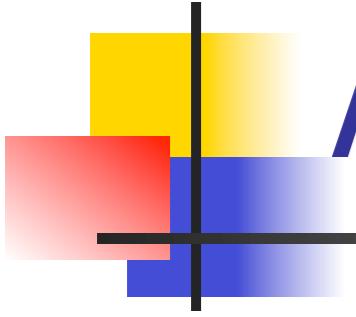


$$B(s,t) = (1-s, s) \begin{vmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{vmatrix} \begin{vmatrix} 1-t \\ t \end{vmatrix}$$

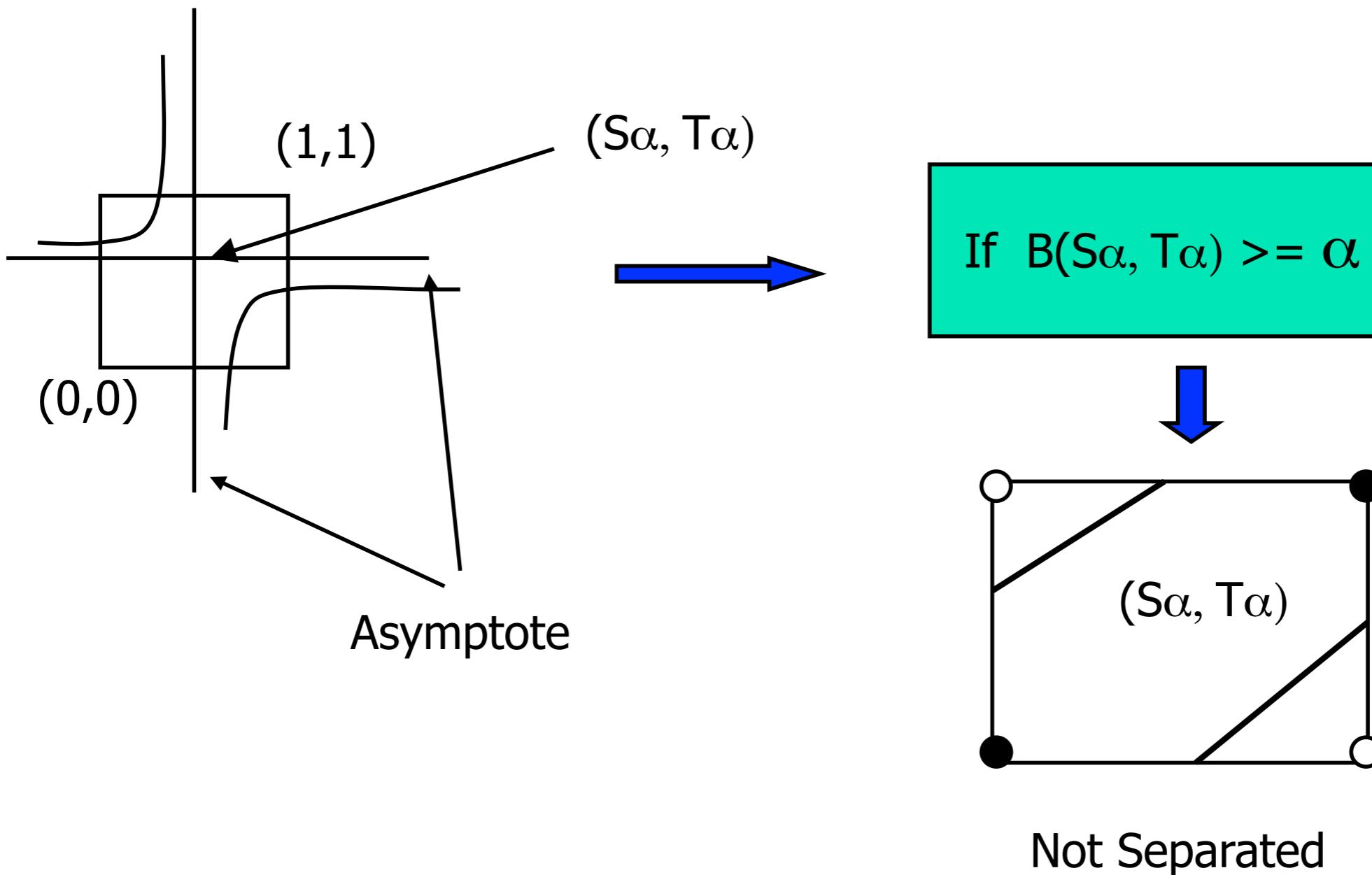
$$= B_{00}(1-s)(1-t) + B_{10}(s)(1-t) + \\ B_{01}(1-s)(t) + B_{11}(s)(t)$$

The contour curves of B :

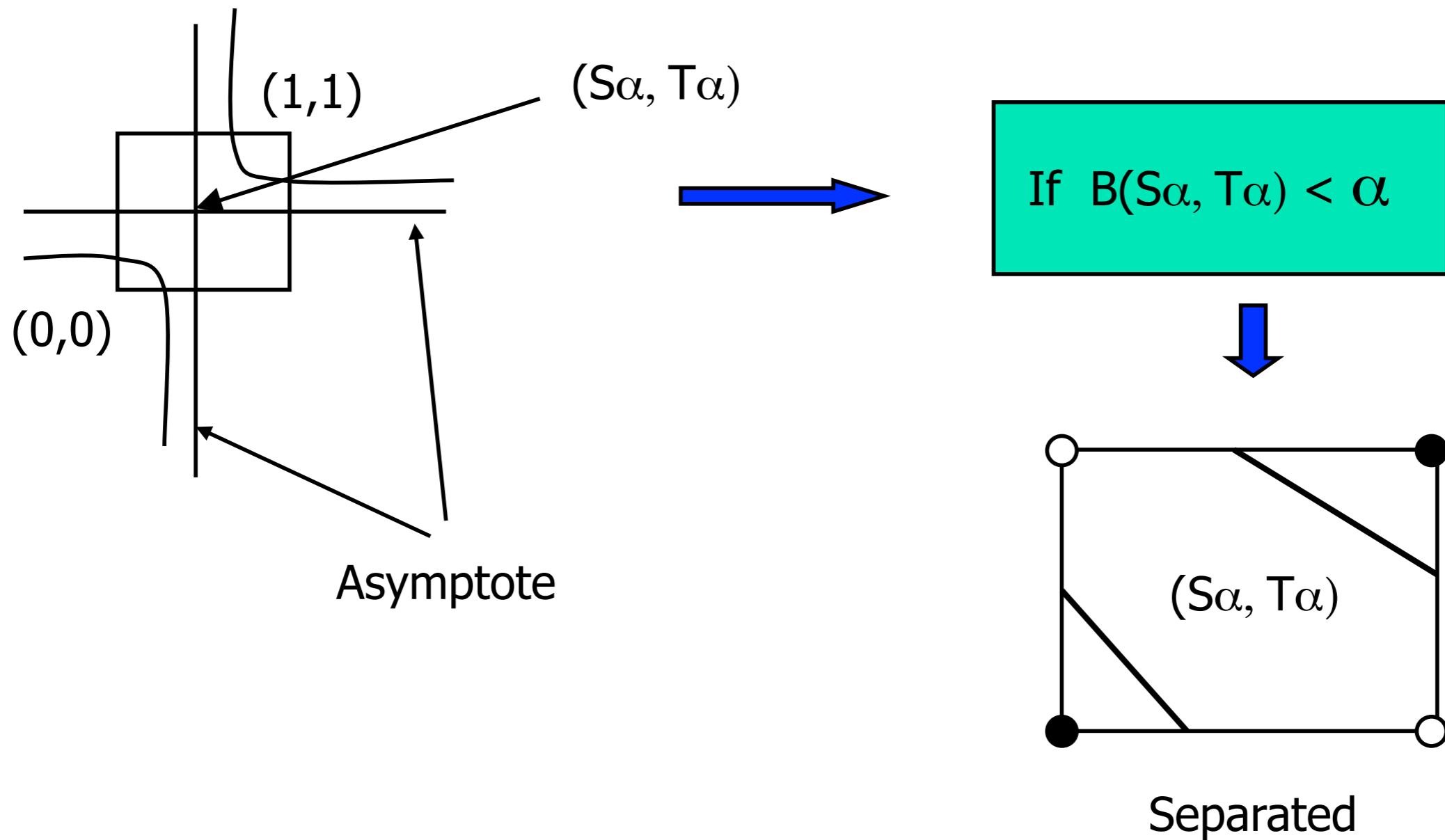
$\{(s,t) \mid B(s,t) = \alpha\}$ are hyperbolas



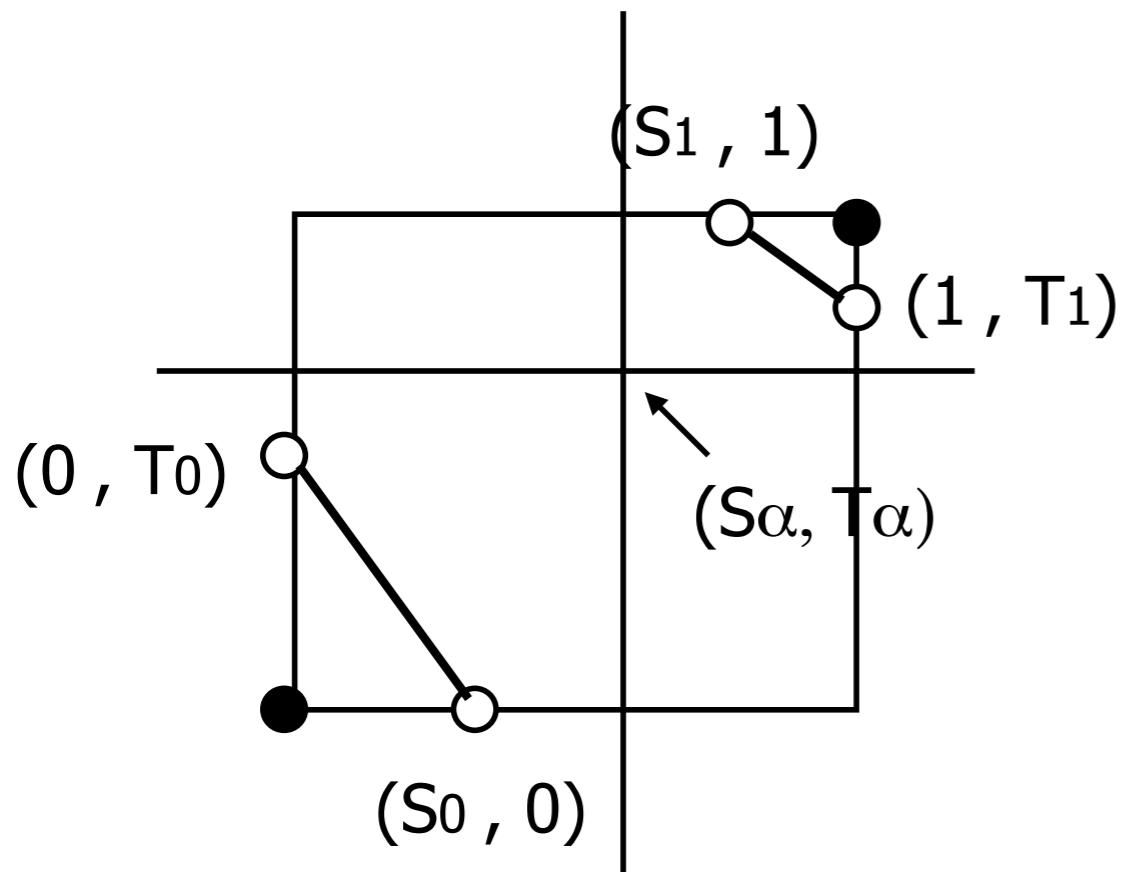
Asymptotic Decider (2)



Asymptotic Decider (3)



Asymptotic Decider (4)



$$B(S_\alpha, 0) = B(S_\alpha, 1)$$

$$B(0, T_\alpha) = B(1, T_\alpha)$$

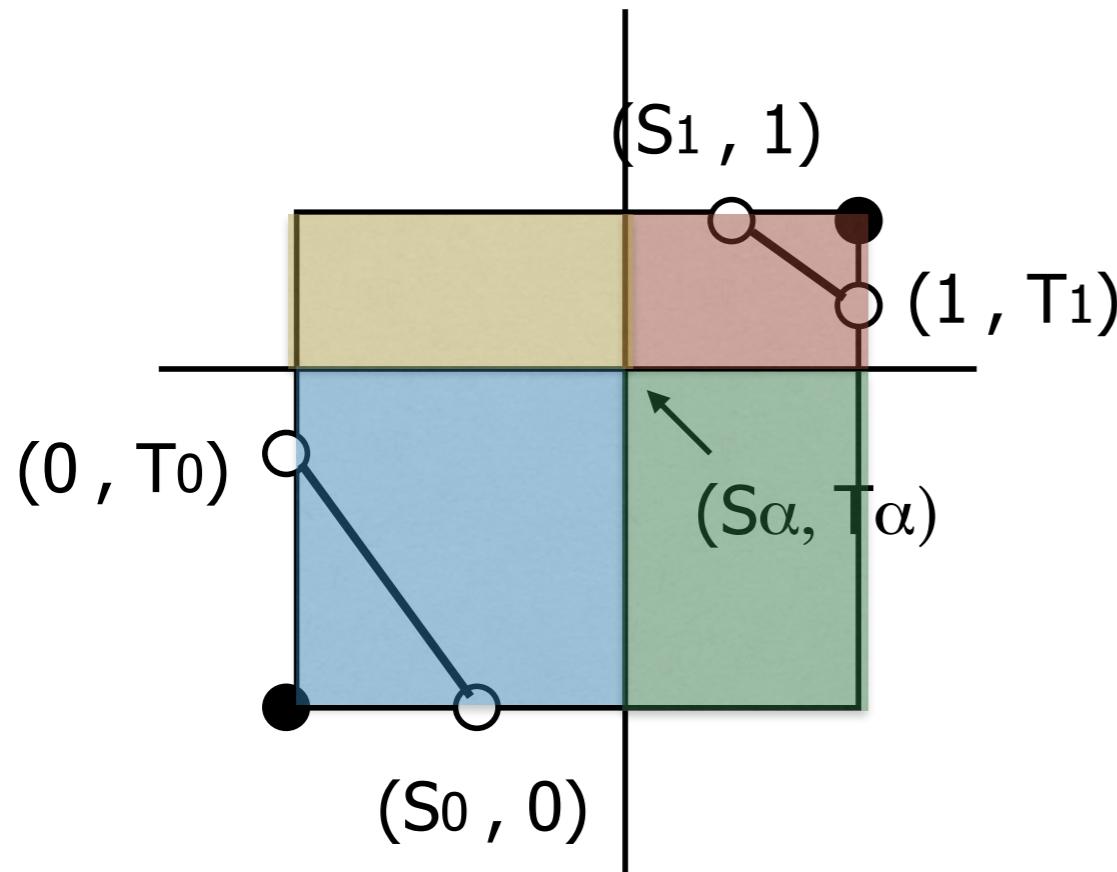


$$S_\alpha = \frac{B_{00} - B_{01}}{B_{00} + B_{11} - B_{01} - B_{10}}$$

$$T_\alpha = \frac{B_{00} - B_{10}}{B_{00} + B_{11} - B_{01} - B_{10}}$$

$$B(S_\alpha, T_\alpha) = \frac{B_{00} B_{11} + B_{10} B_{01}}{B_{00} + B_{11} - B_{01} - B_{10}}$$

Asymptotic Decider (4)



$$B(S_\alpha, 0) = B(S_\alpha, 1)$$

$$B(0, T_\alpha) = B(1, T_\alpha)$$



$$S_\alpha = \frac{B_{00} - B_{01}}{B_{00} + B_{11} - B_{01} - B_{10}}$$

$$T_\alpha = \frac{B_{00} - B_{10}}{B_{00} + B_{11} - B_{01} - B_{10}}$$

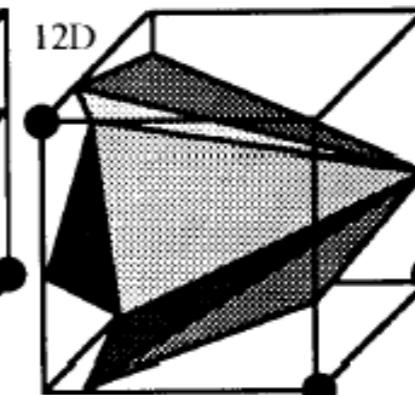
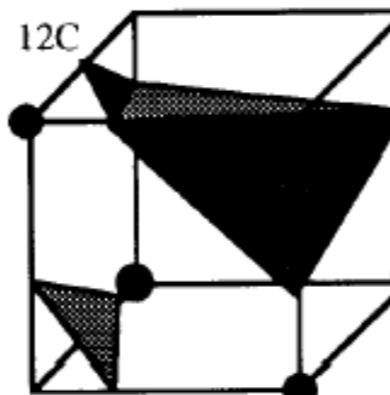
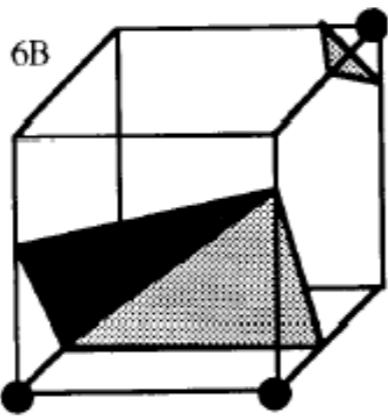
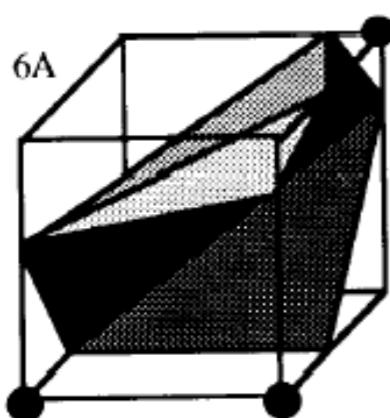
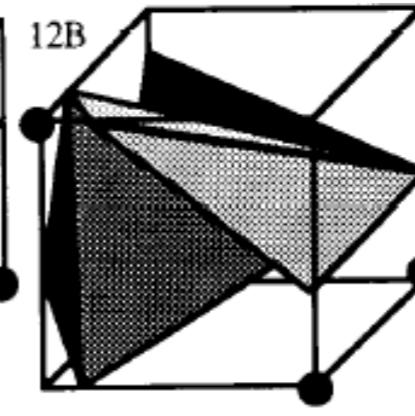
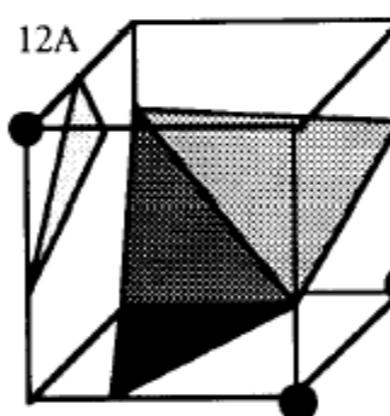
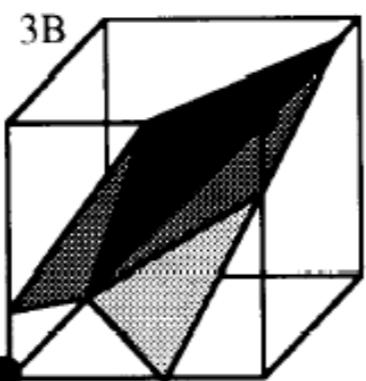
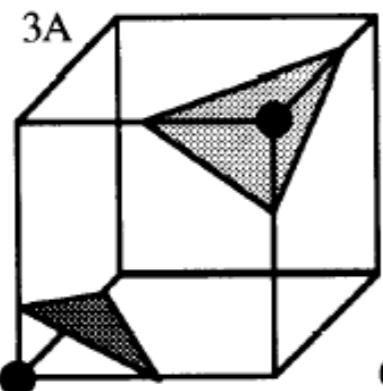
$$B(S_\alpha, T_\alpha) = \frac{B_{00} B_{11} + B_{10} B_{01}}{B_{00} + B_{11} - B_{01} - B_{10}}$$

i.e., use S_α, T_α to divide the face into

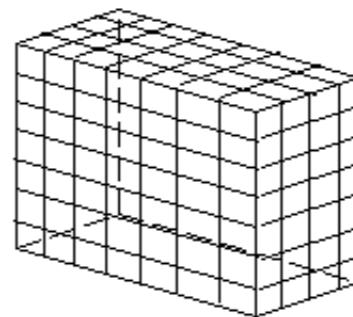
4 unambiguous rectangles — only 2 of which we need to evaluate.

Asymptotic Decider (5)

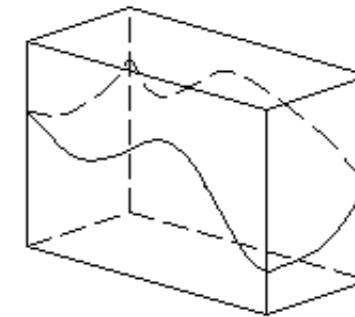
- case 3, 6, 12, 10, 7, 13
 - (These are the cases with at least one ambiguous faces)



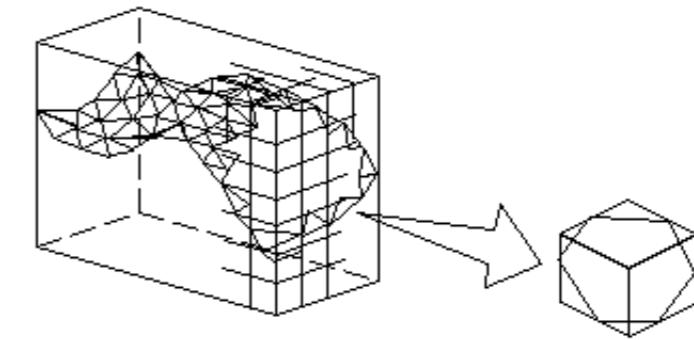
- Summary
 - 256 Cases
 - Reduce to 15 cases by symmetry
 - Ambiguity in cases
3, 6, 7, 10, 12, 13
 - Causes holes if arbitrary choices are made
- Up to 5 triangles per cube
- Several isosurfaces
 - Run MC several times
 - Semi-transparency requires spatial sorting



(a) Volume data



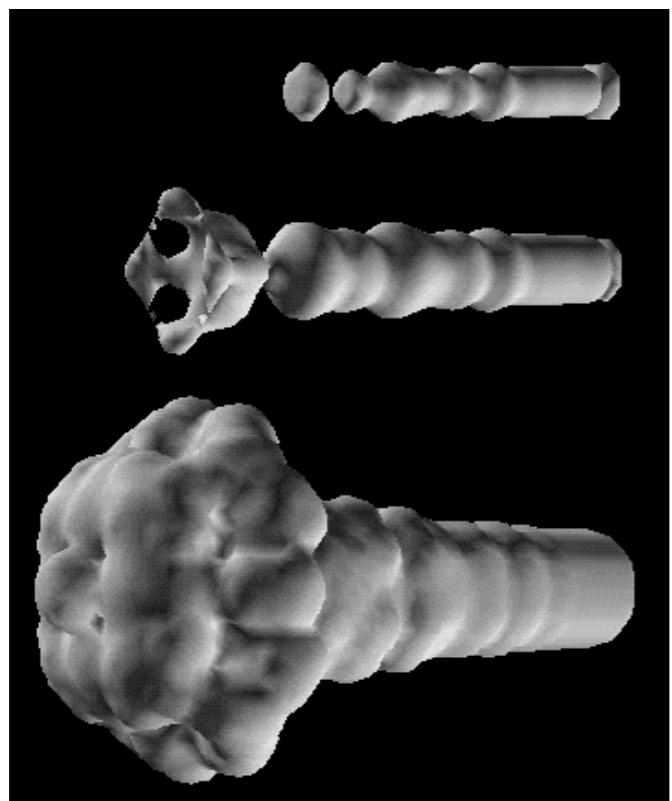
(b) Isosurface
 $S = f(x, y, z)$



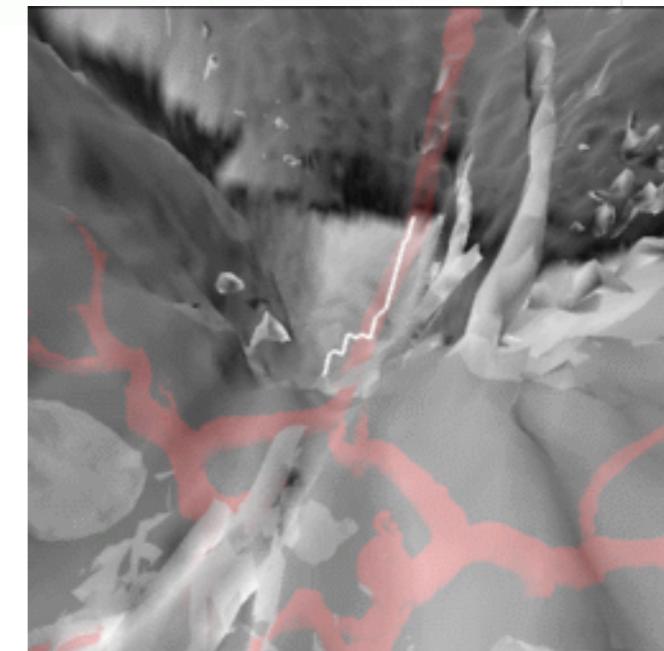
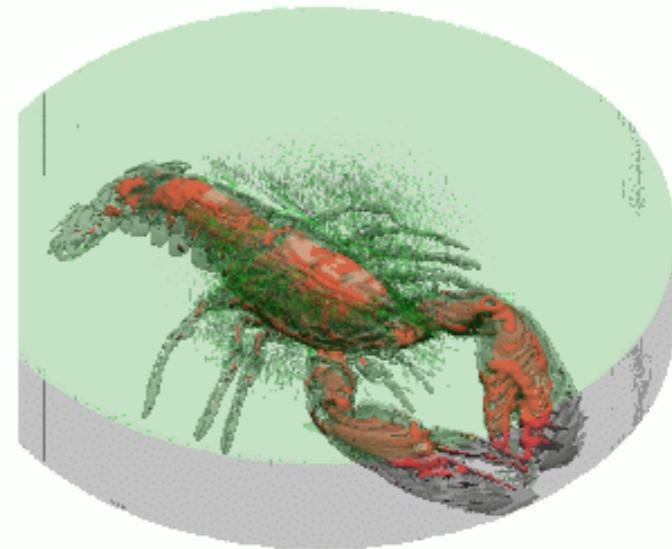
(c) Polygonal Approximation

- Examples

1 Isosurface



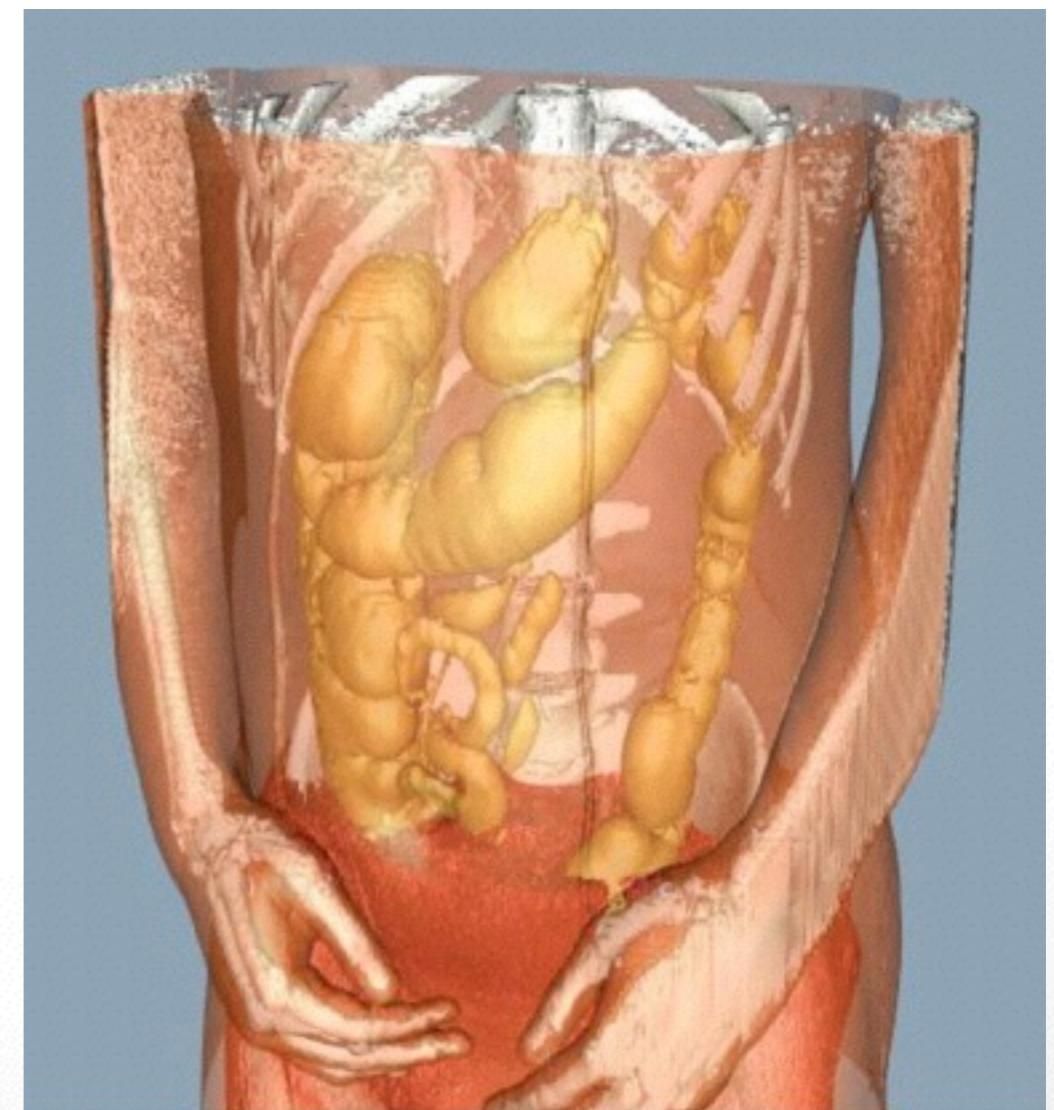
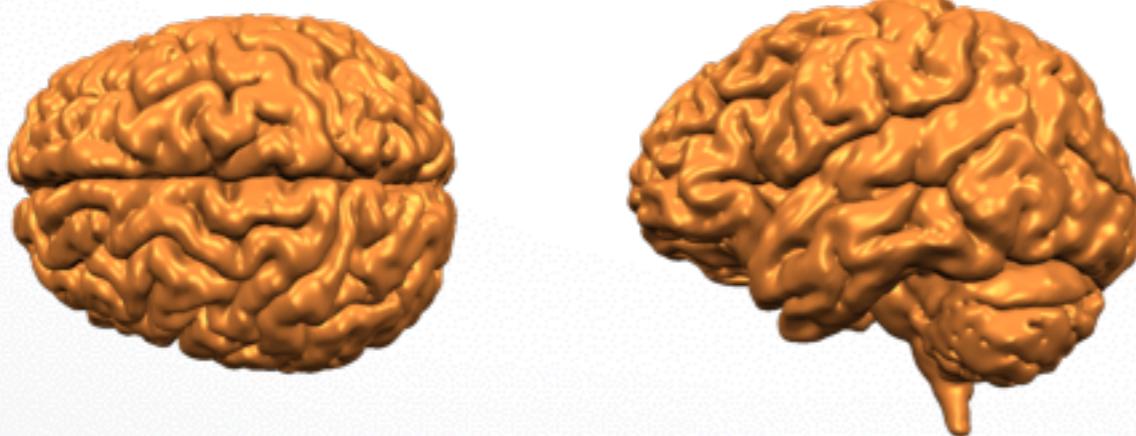
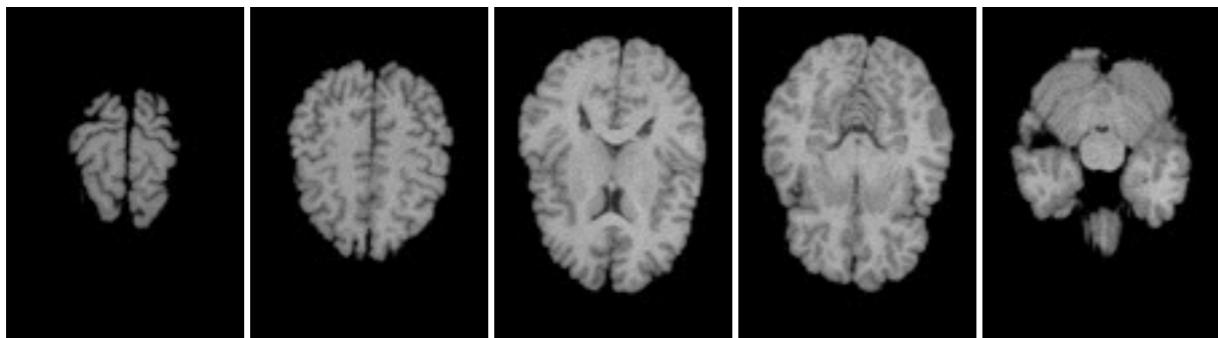
3 Isosurfaces



2 Isosurfaces

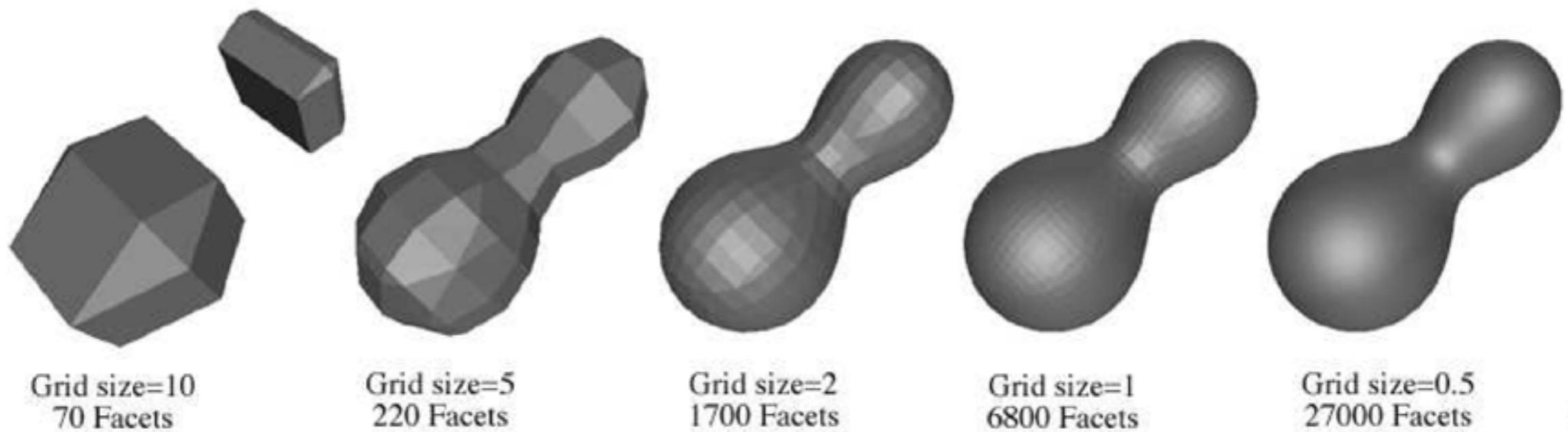
Marching Cubes

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



Marching Cubes

Effect of grid size



Marching Cube Variants

The “first Marching Cubes” paper

Wyvill, McPheeeters & Wyvill 86.

Visual Computer

Data structure for soft objects

Geoff Wyvill¹, Craig McPheeeters²,
and Brian Wyvill²

¹ Department of Computer Science,
University of Otago,
Box 56, Dunedin, New Zealand

² Department of Computer Science,
University of Calgary,
2500 University Drive N.W. Calgary, Alberta,
Canada, T2N 1N4

We introduce the concept of *soft* objects whose shape changes in response to their surroundings. Established geometric modelling techniques exist to handle most engineering components, including ‘free form’ shapes such as car bodies and telephones. More recently, there has been a lot of interest in modelling natural phenomena such as smoke, clouds, mountains and coastlines where the shapes are described stochastically, or as fractals. None of these techniques lends itself to the description of *soft* objects. This class of objects includes fabrics, cushions, living

The *Graphicsland* project group (Wyvill 1985a) at the University of Calgary has developed an organised collection of software tools for producing animation from models in three dimensions. The system allows the combination of several different kinds of modelling primitive (Wyvill et al. 1985b). Thus polygon based models can be mixed freely with fractals (Mandelbrot 1983, Fournier 1982) and particles (Reeve 1983) in a scene. Motion and camera paths can be described, and animation generated. Note that we do not include the use of a two dimensional ‘paint’ system. Our objective is always to construct views of a full three dimensional model. An important class of objects in the everyday world is *soft*. That is, the shape of the object varies constantly because of the forces imposed on it by its surroundings. A bouncing ball is a simple example: as it strikes the ground, it flattens. The smoothly covered joints of animals change shape with seamless continuity, and liquids mould themselves to their surroundings and even break into separate droplets. Even apparently rigid objects deform in some circumstances. Trees, for example, bend in the wind.

To date, there seem to have been few attempts to model *soft* objects for computer graphics. Possibly, this is because *soft* objects are less important in engineering. But it is also true that much effort in computer graphics has been directed to producing still pictures and you cannot tell that an object is *soft* until it moves. Clouds (Gardner 1985) and particles (Reeve 1983) come close, but there is nothing in either of these papers which deals with the interaction of particles with surrounding objects.

We have been experimenting with a general model for *soft* objects which represents an object or collection of objects by a scalar field. That is a mathematical function of space which defines



Nov 10, 2015: 1033 cites on Google Scholar

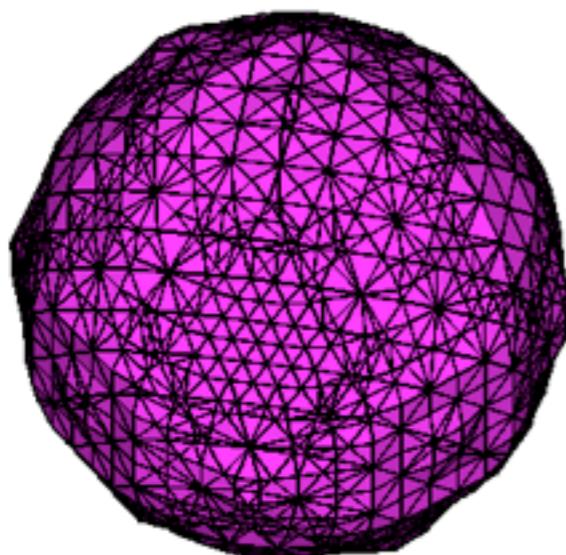
in internal memory usage.

Key words: Soft objects – Geometric modelling – Computer animation

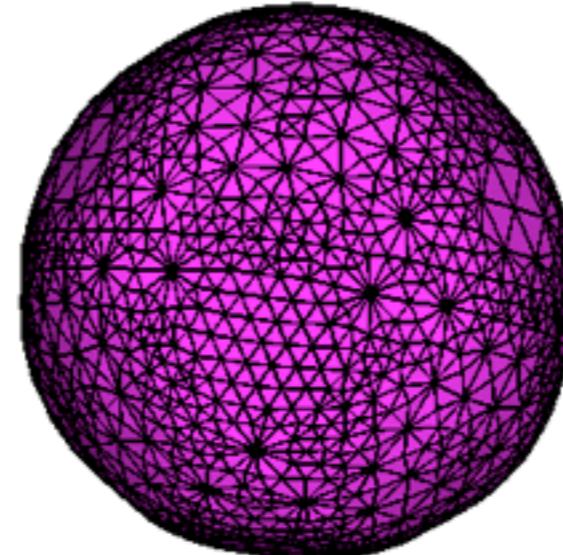
cations and describes a direct rendering technique using an elegant set of sorted lists. A similar technique has been used for some years in the LINKS project at the University of Osaka (Nishimura 1985). Ken Perlin has used a modification of

Marching tetrahedra

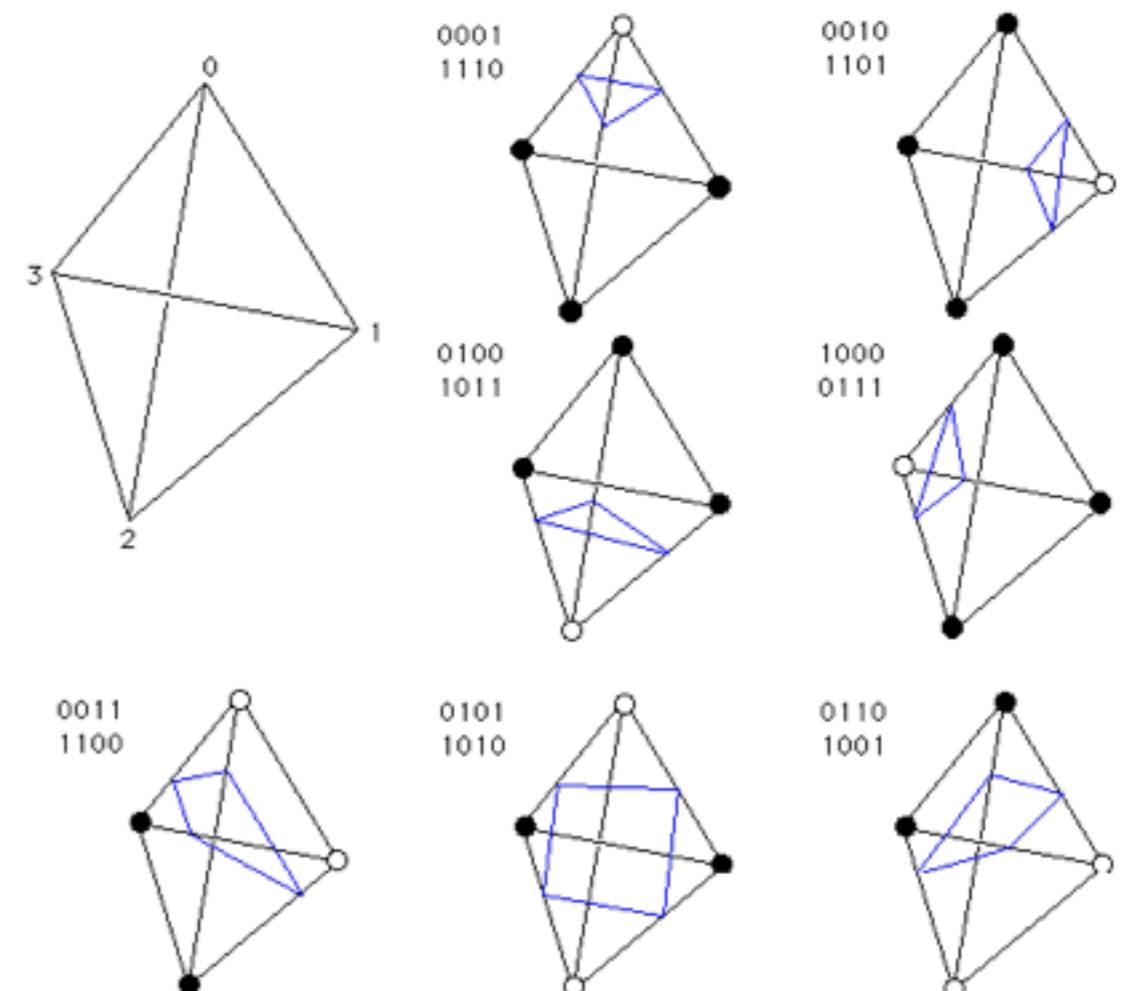
- Only 8 cases to consider in a tetrahedron
- Correct piecewise-linear interpolants on tet meshes!
- Generates *horrible* triangles.



marching tets

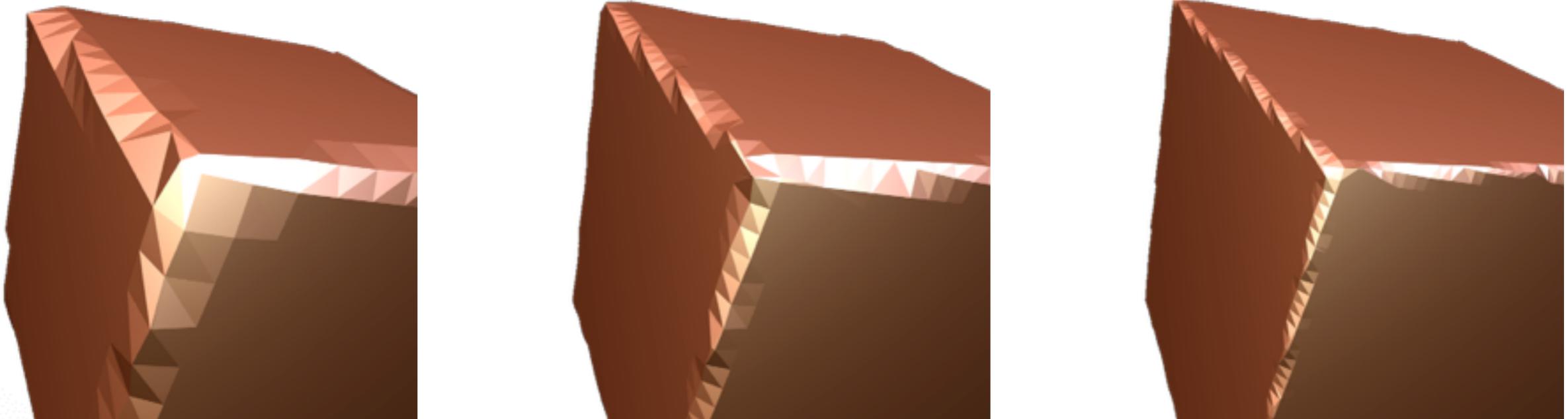


after 30 iterations of Laplacian smoothing...



Doi and Koide. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. IEICE Transactions on Information and Systems, 1991

Increasing Resolution

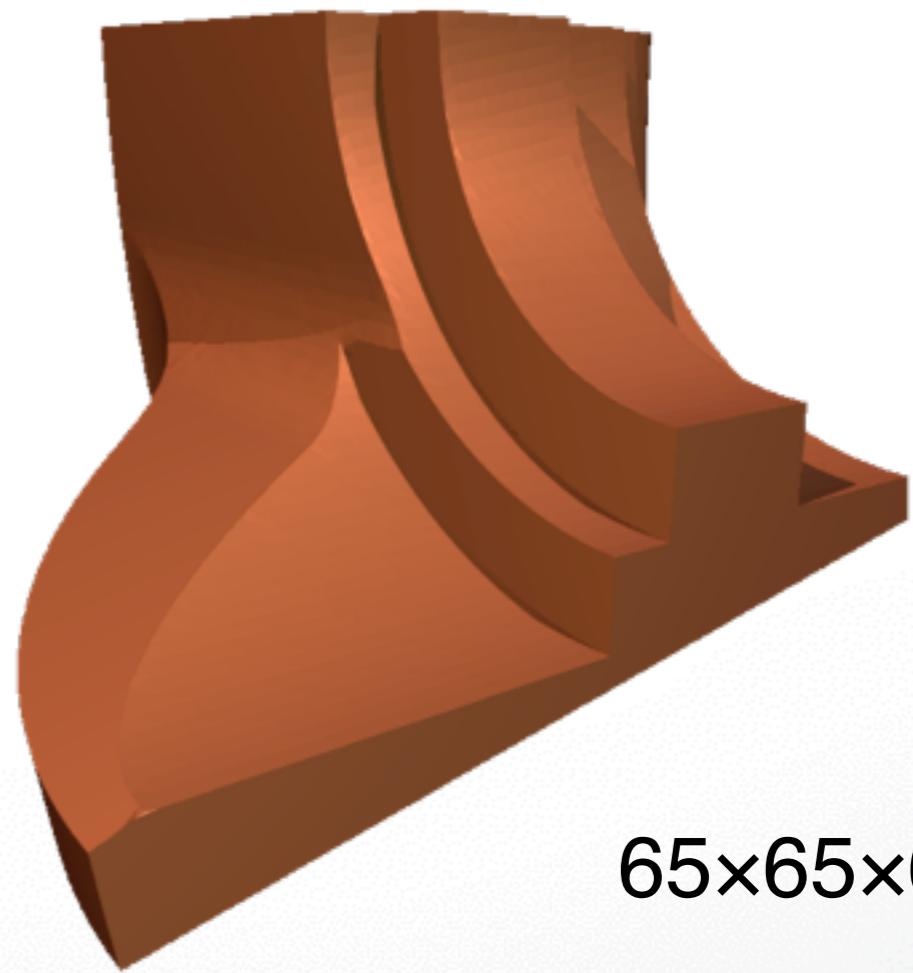
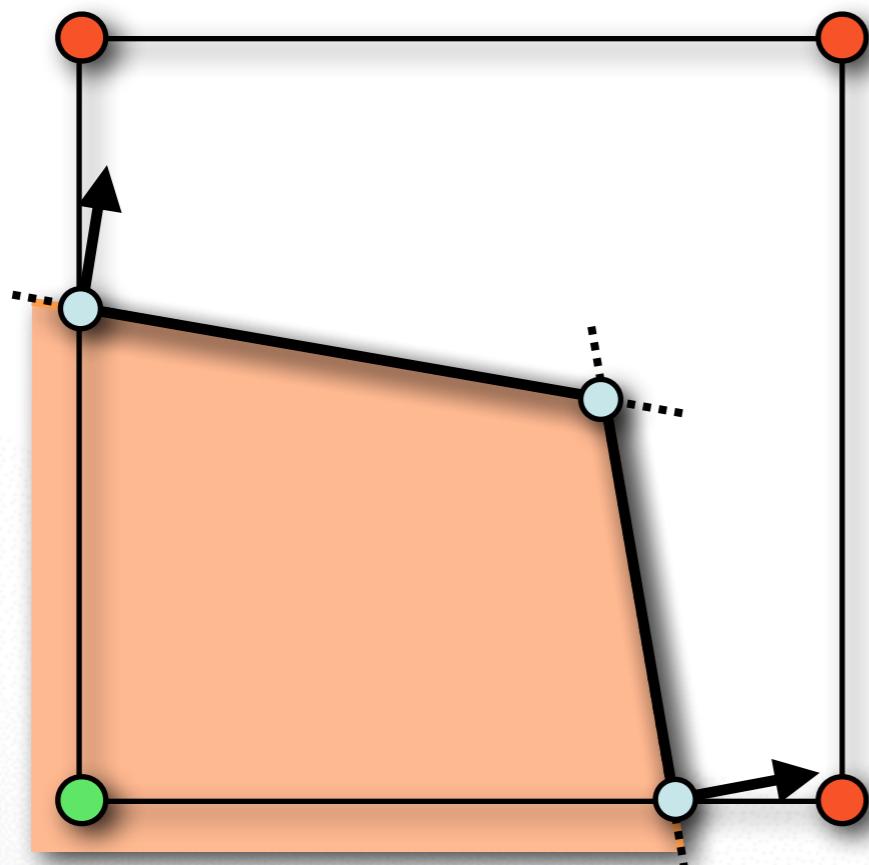


Does not remove alias problems!

Extended Marching Cubes

Locally extrapolate distance gradient

Place samples on estimated features

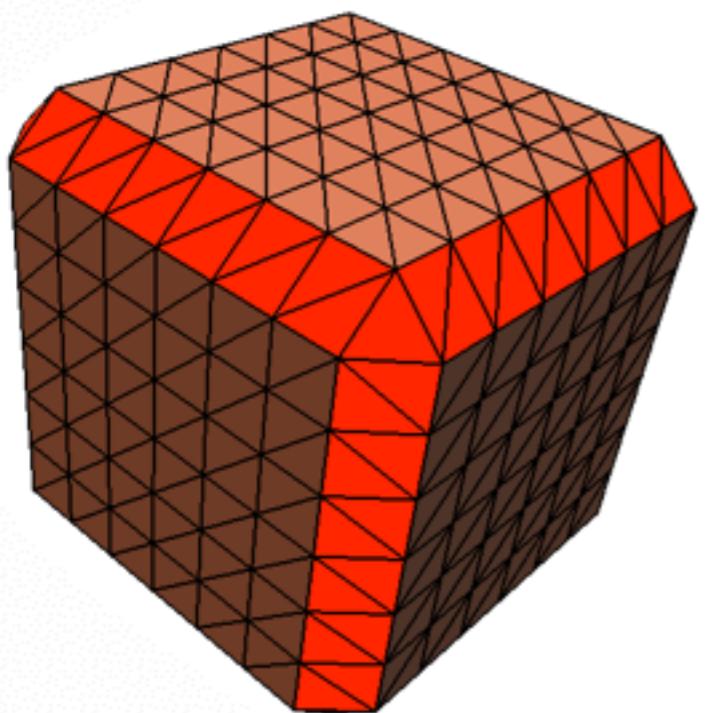


L Kobbelt, M Botsch, U Schwancke, HP Seidel. Feature Sensitive Surface Extraction from Volume Data. Siggraph 2001.

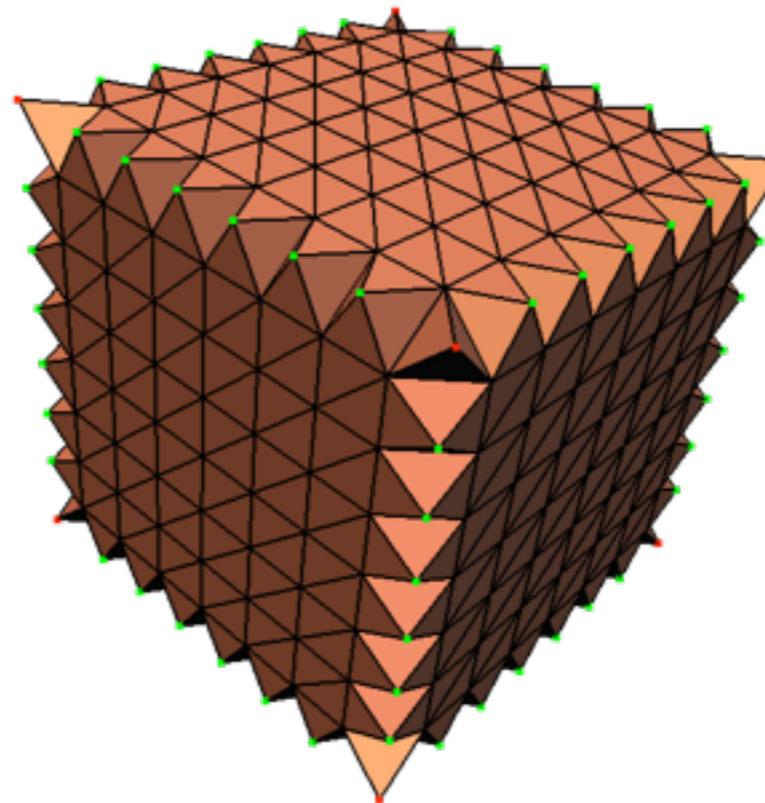
Slides: Hao Li, USC

58

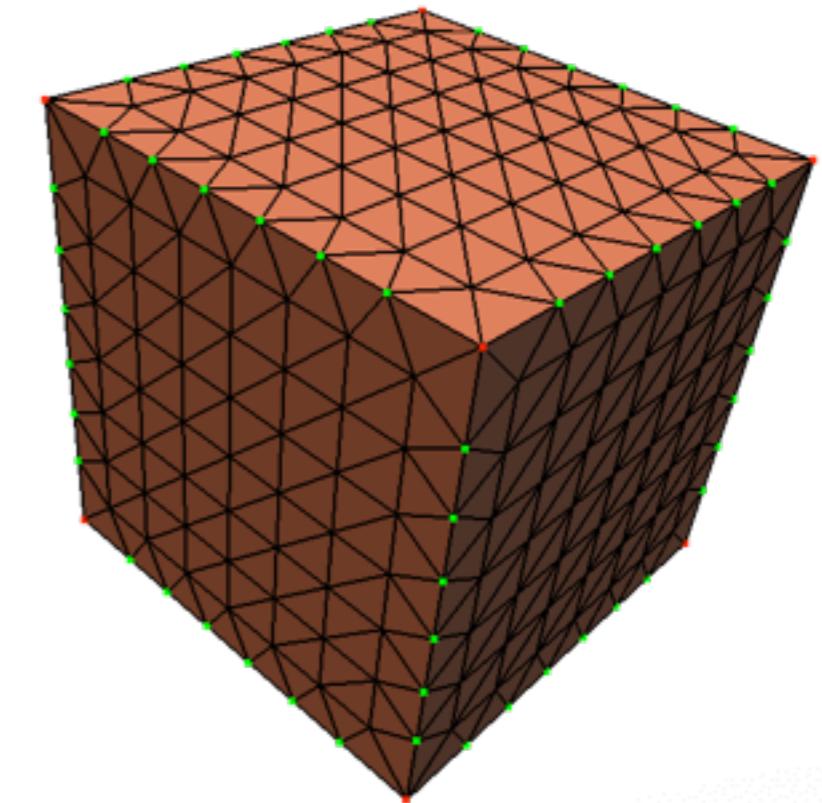
Extended Marching Cubes



**Feature
Detection**



**Feature
Sampling**

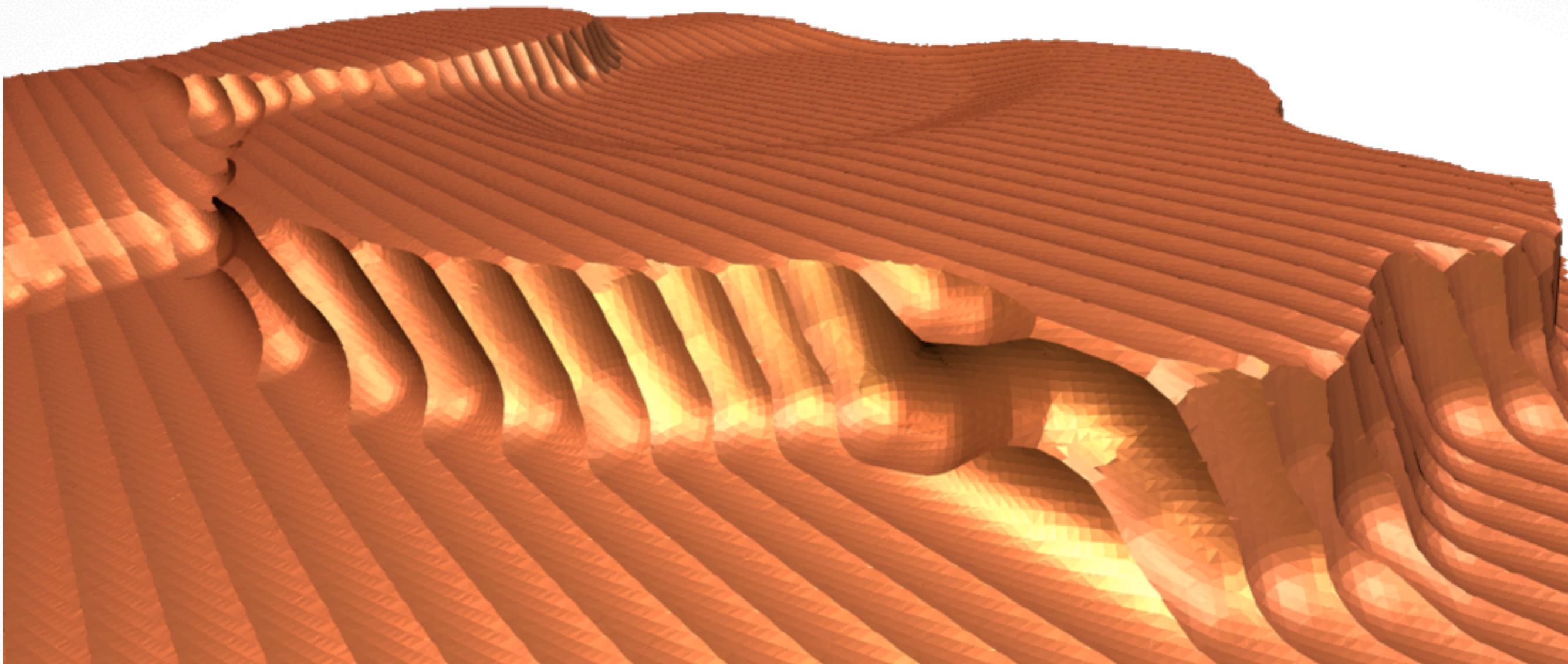


**Edge
Flipping**

L Kobbelt, M Botsch, U Schwancke, HP Seidel. Feature Sensitive Surface Extraction from Volume Data. Siggraph 2001.

Slides: Hao Li, USC

Milling Simulation



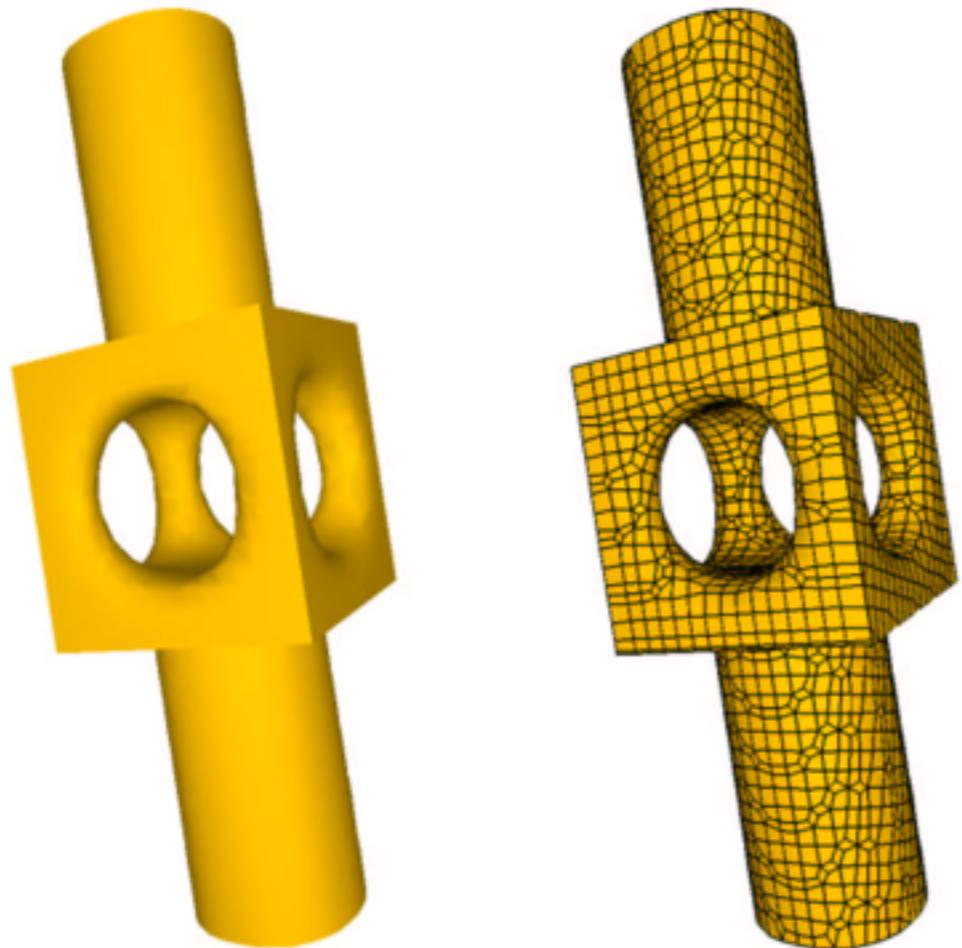
257×257×257

L Kobbelt, M Botsch, U Schwancke, HP Seidel. Feature Sensitive Surface Extraction from Volume Data. Siggraph 2001.

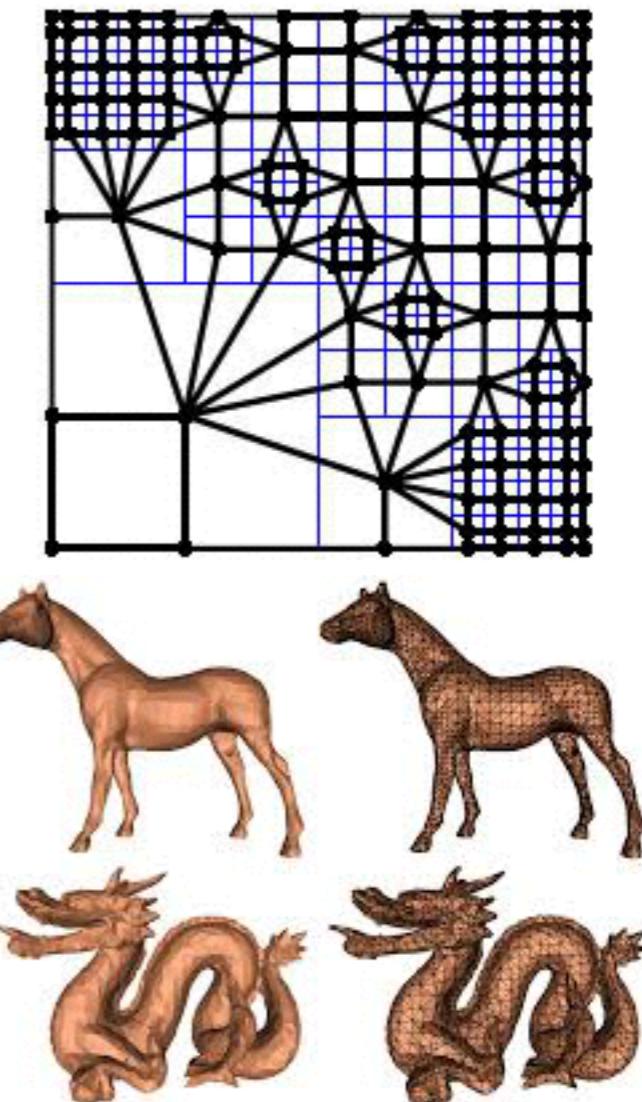
Slides: Hao Li, USC

63

Dual Contouring



Ju et al., Dual Contouring of Hermite Data,
Siggraph 2002



Schafer and Warren., Dual Marching Cubes:
Primal Contouring of Dual Grids. Computer
Graphics Forum, 2004

Edge Groups

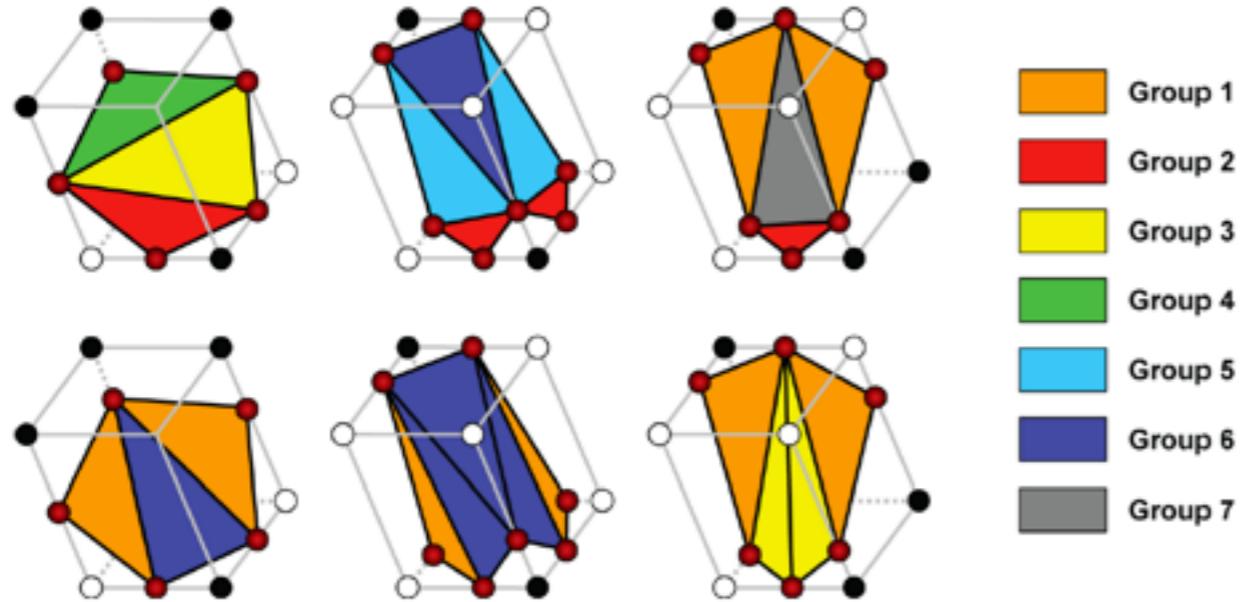


Fig. 4. Replacing cases in MC. The top row shows the original triangulation in 3 MC cases (case 5, complement of case 6 and case 11, respectively [20]), while the bottom row shows the modified connectivity. The reconnection of the cut points removes the edge group 2 from these cells, reducing the probability of generating low quality triangles.

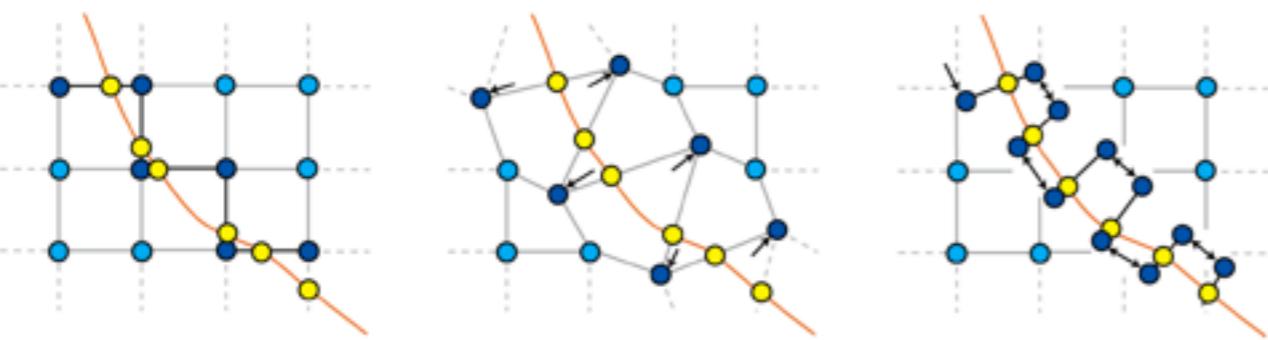
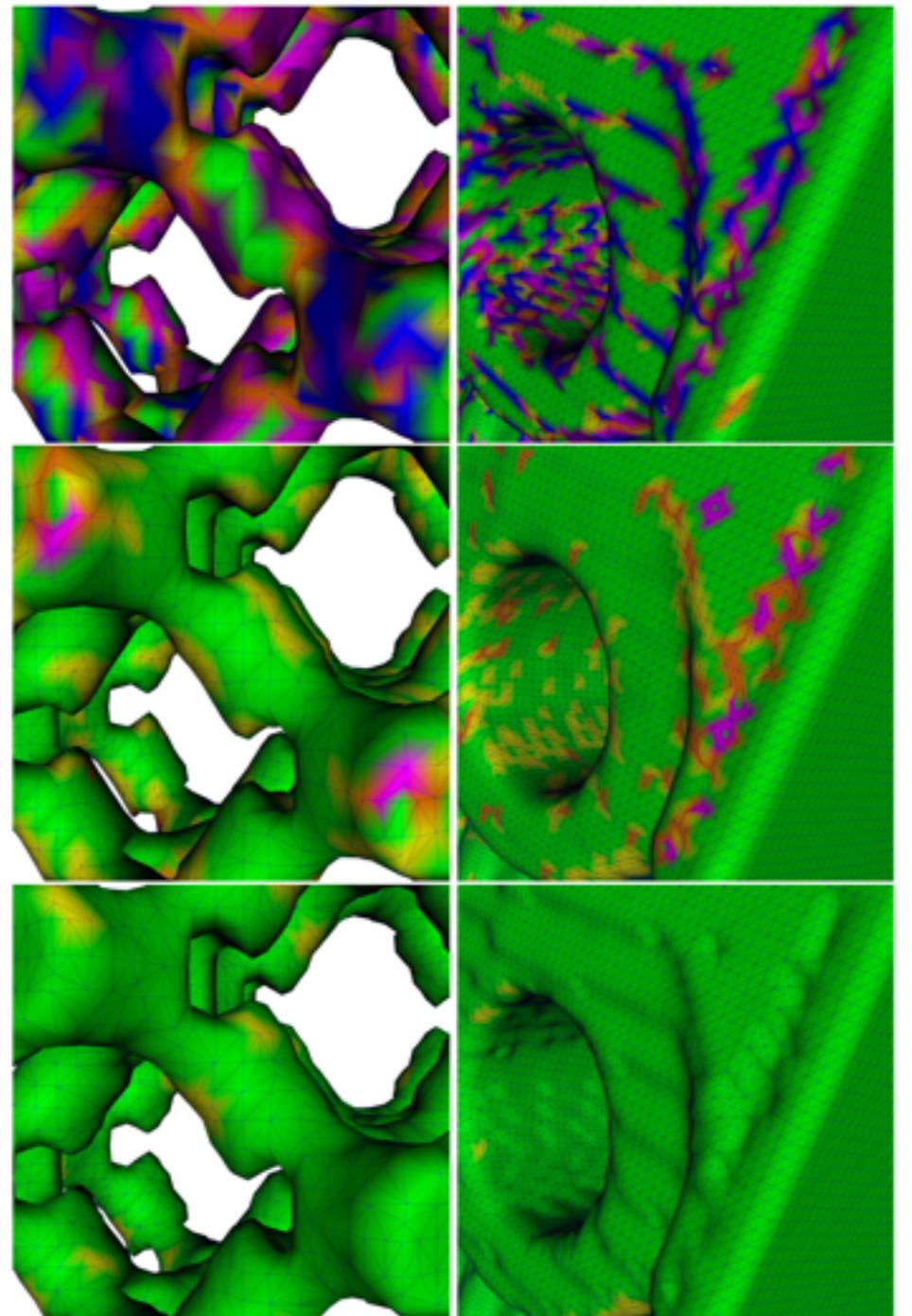


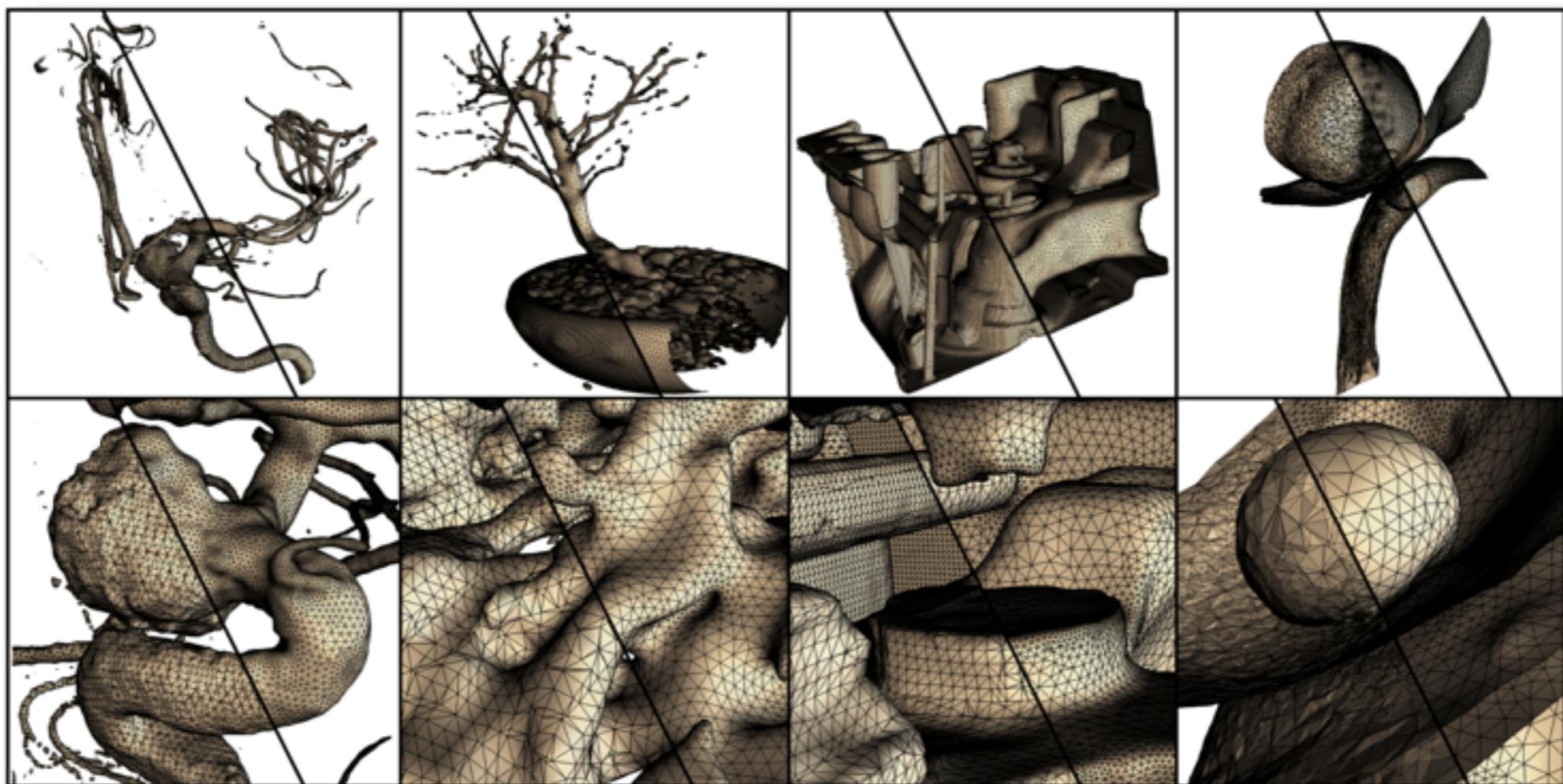
Fig. 5. Intuition behind Macet: small changes to grid vertices positions (left) may improve triangle quality. Moving vertices along the gradient (center) or along tangential paths (right) improve triangle quality.



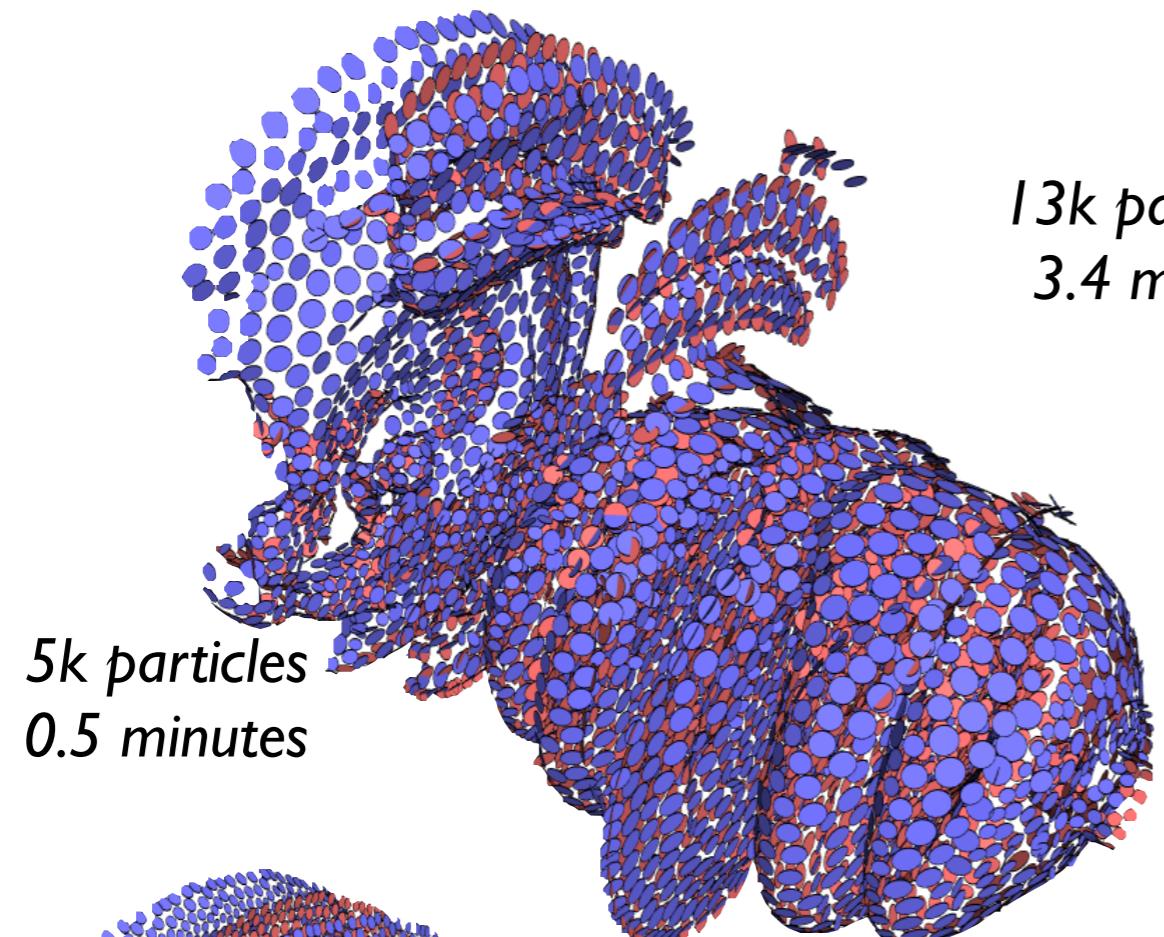
Particle-driven mesh extraction

Advancing Front

- Starting from a seed point, use curvature of the implicit surface to determine local feature size (LFS), find next seed points, and from those create a guidance field locally resampling the scalar field.
- Continue the guidance field until all fronts merge, and the mesh is done.

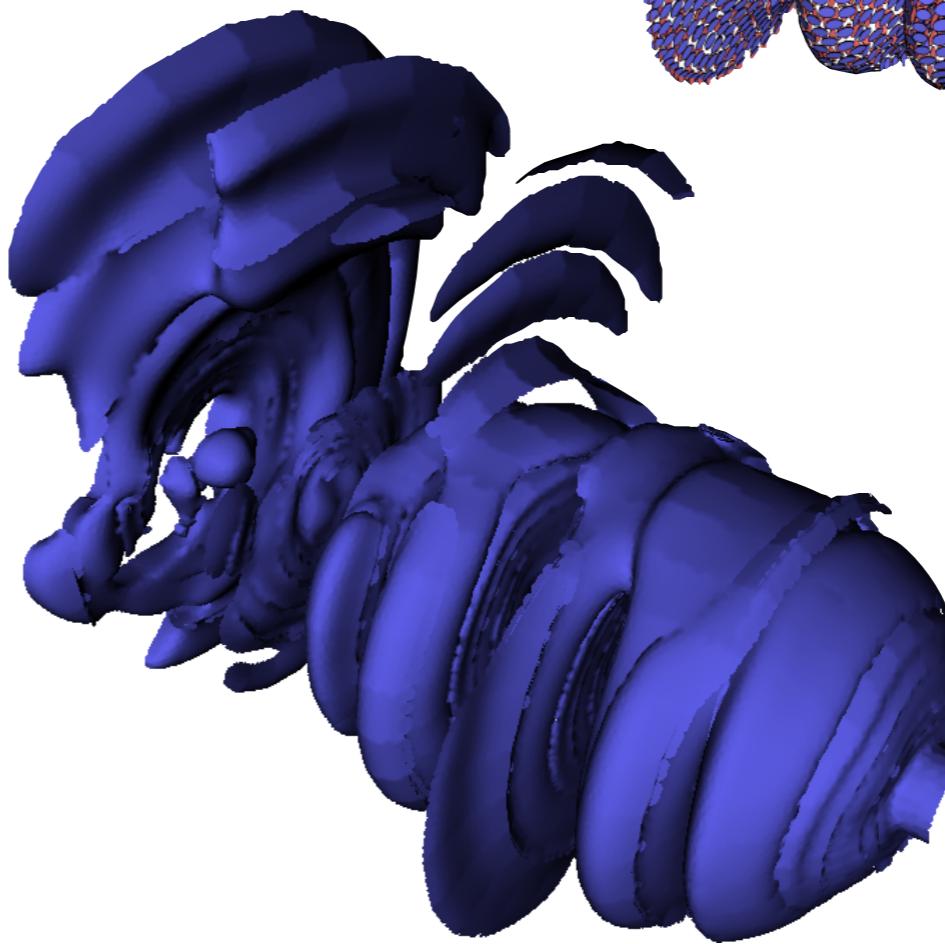
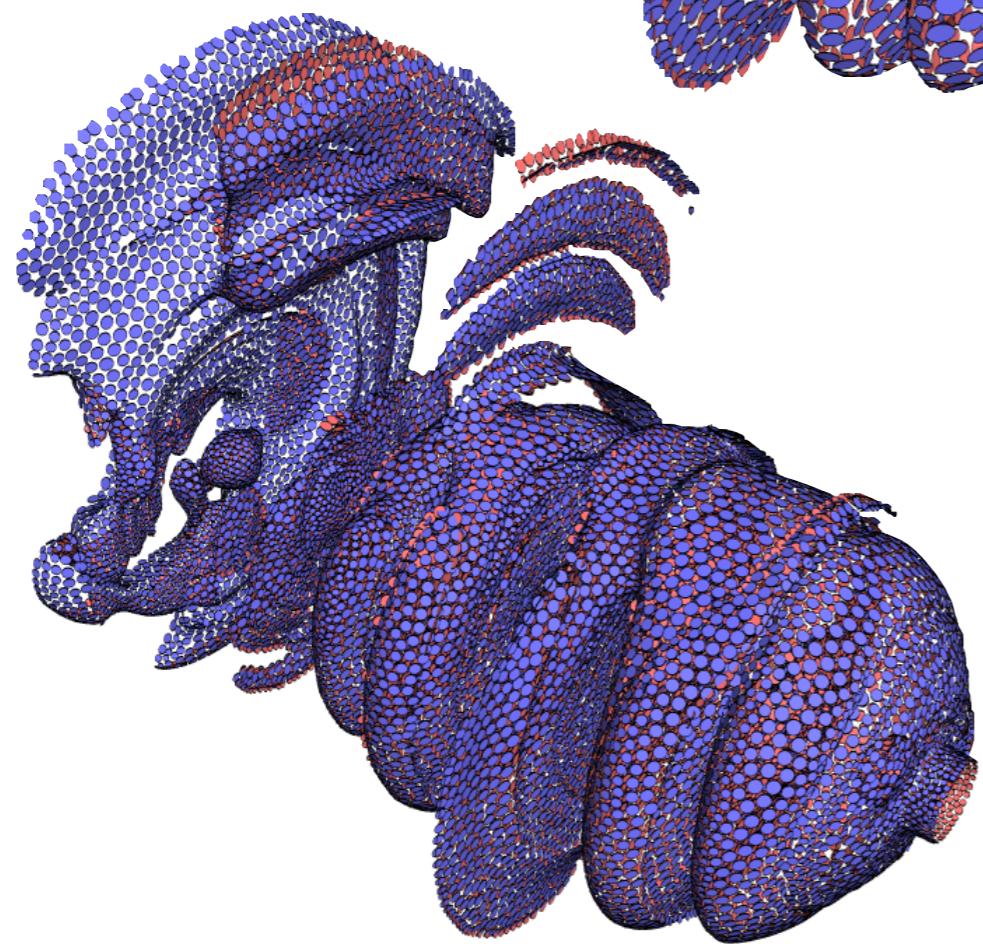


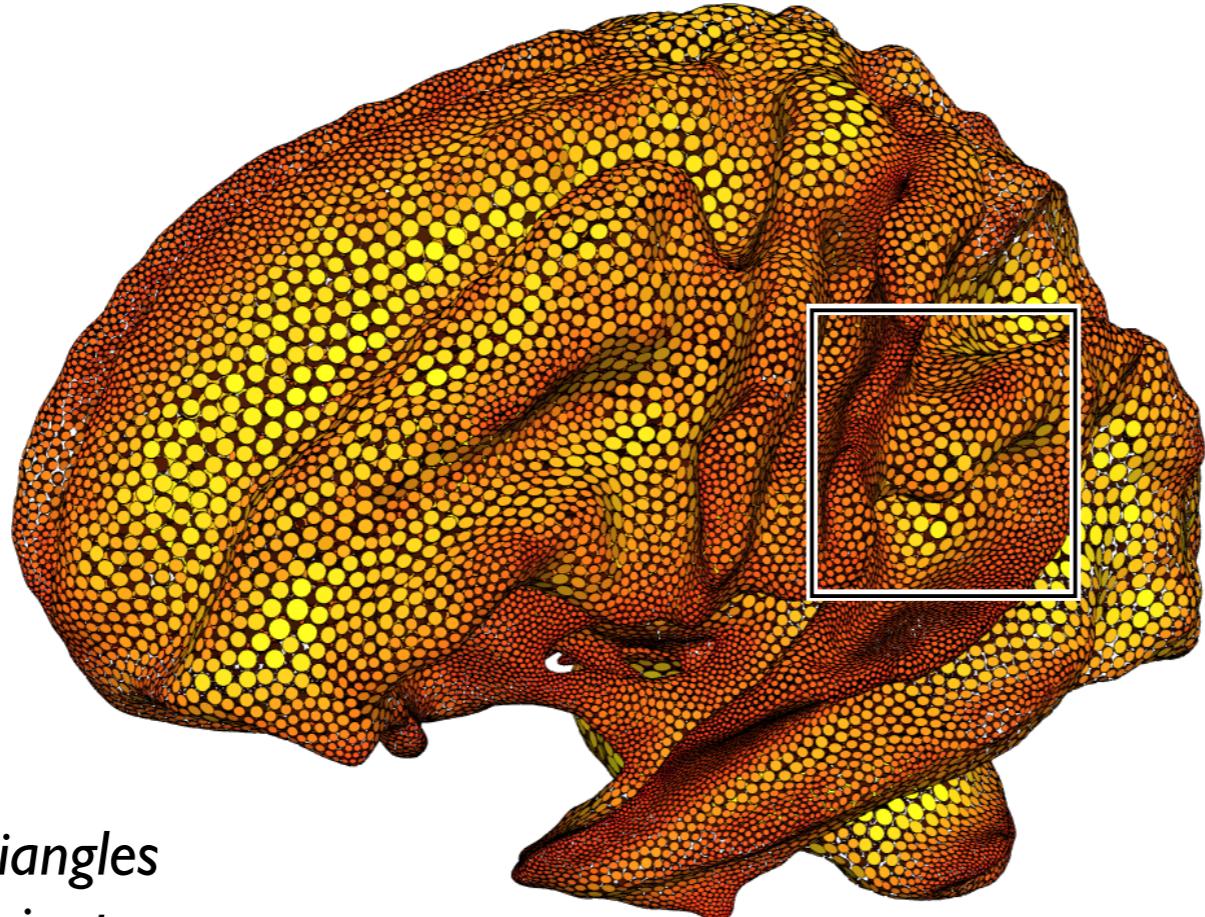
Schreiner et al. High-Quality Extraction of Isosurfaces from Regular and Irregular Grids. IEEE Visualization 2006



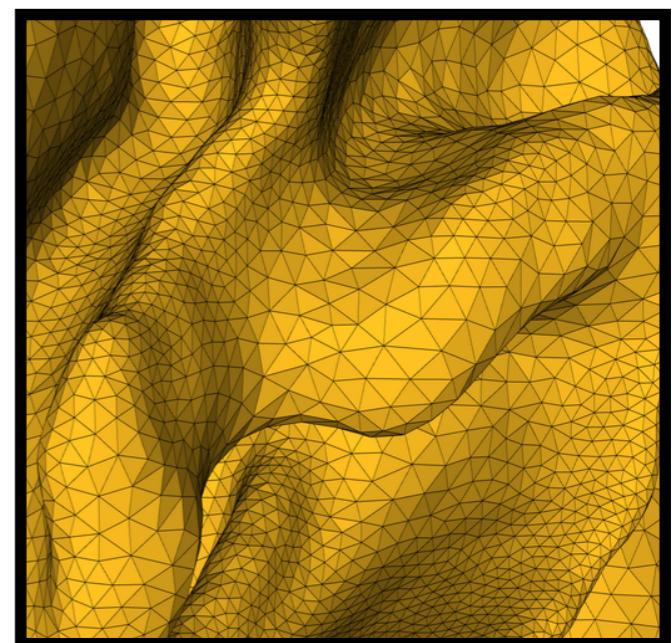
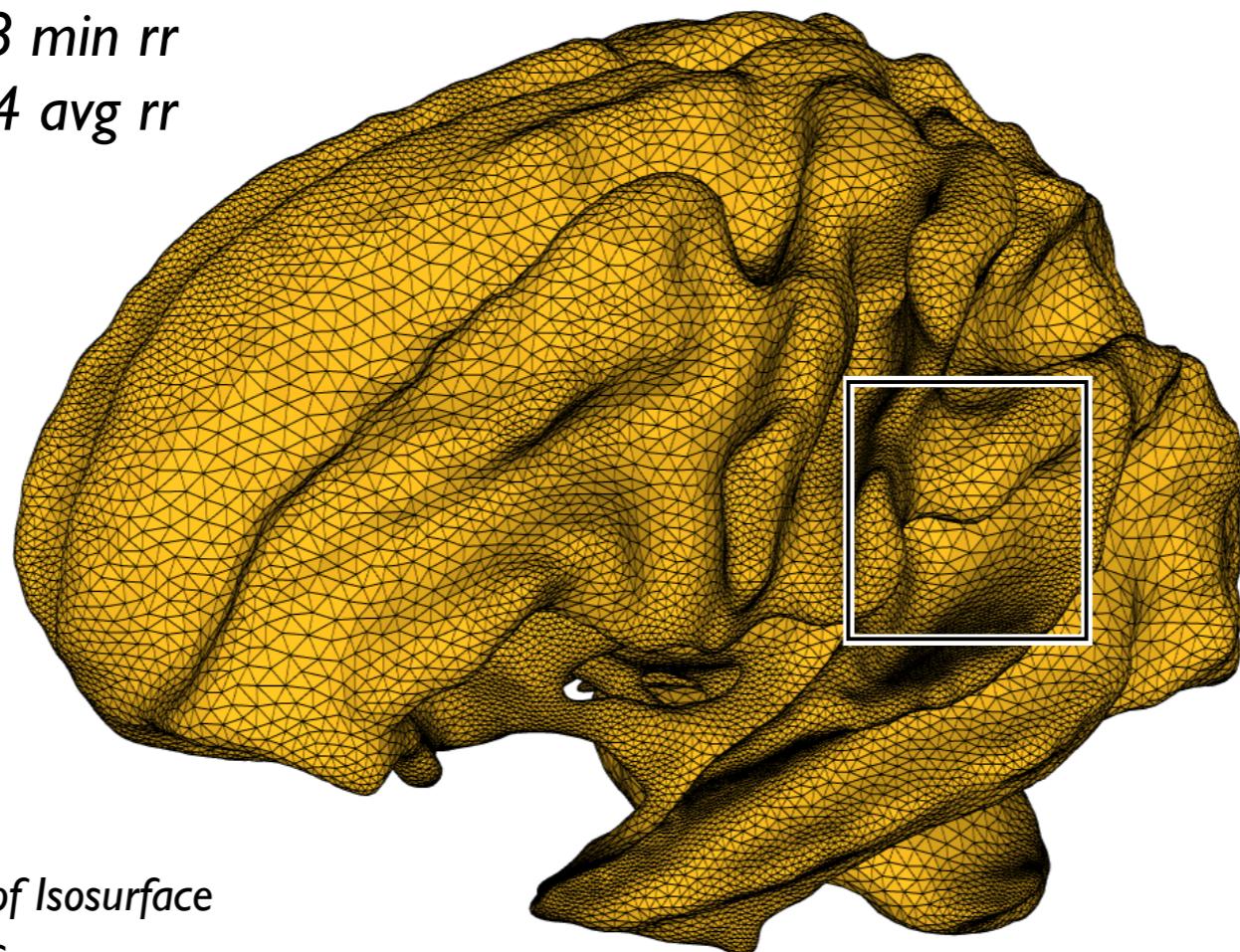
*5k particles
0.5 minutes*

*13k particles
3.4 minutes*



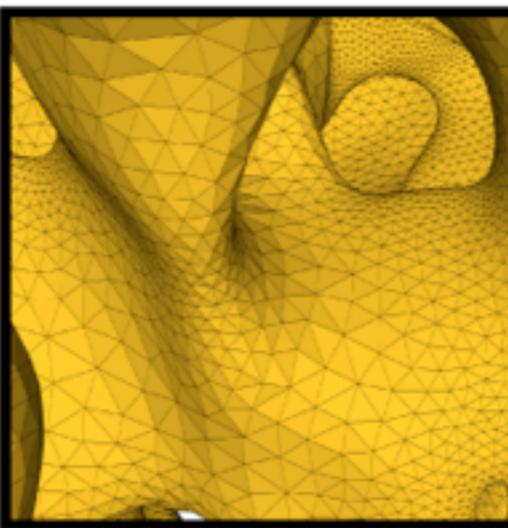
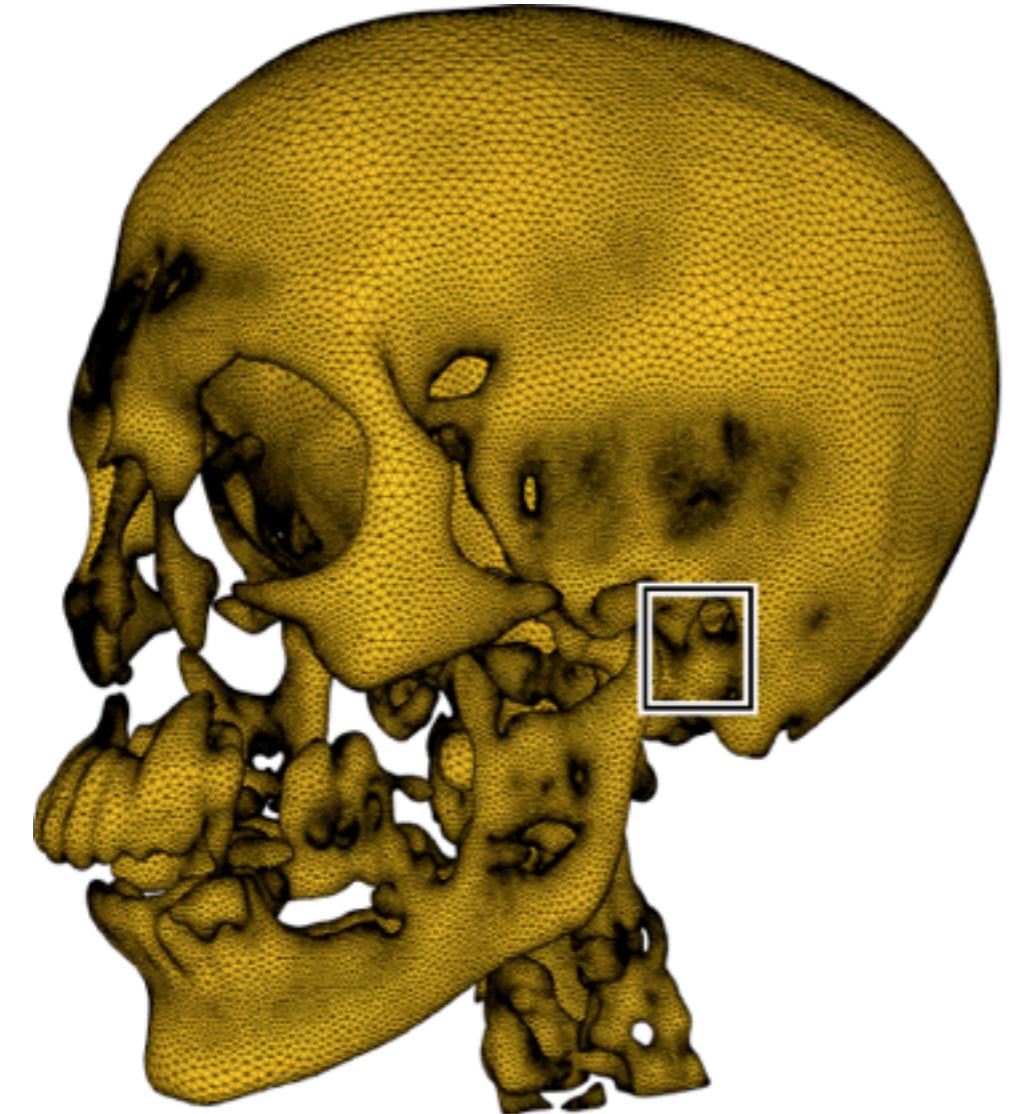


*182k triangles
41 minutes
0.18 min rr
0.94 avg rr*

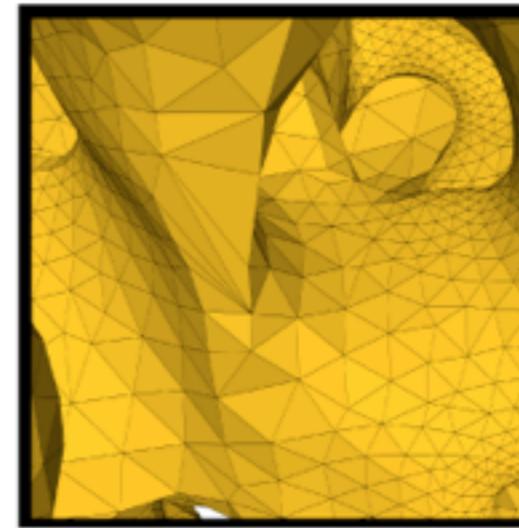


*Topology, Accuracy, and Quality of Isosurface
Meshes Using Dynamic Particles.*

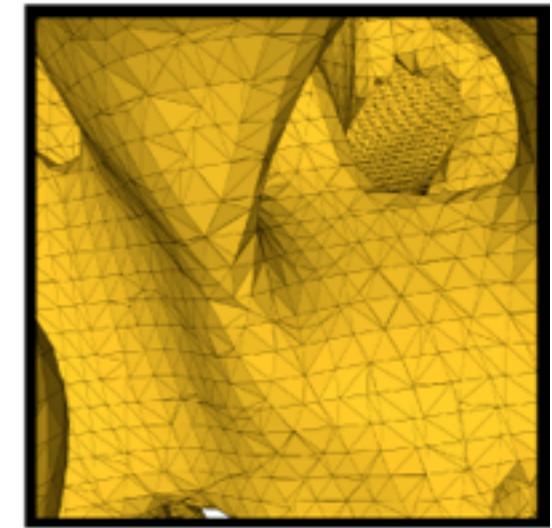
M. Meyer et al., Vis 2007.



particle system



advancing front



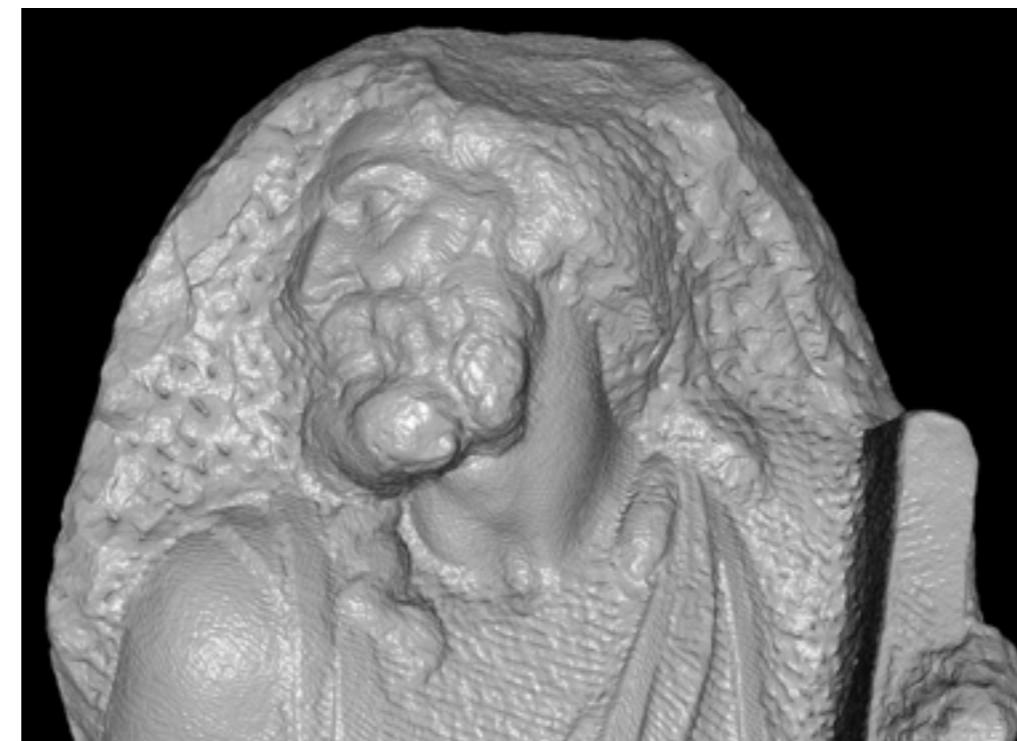
marching cubes

Surface splatting

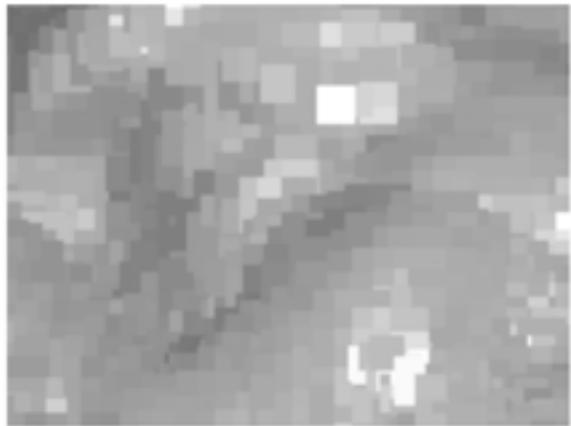
QSplat

- The first (large) pure point-based system, built on a bounding sphere hierarchy.

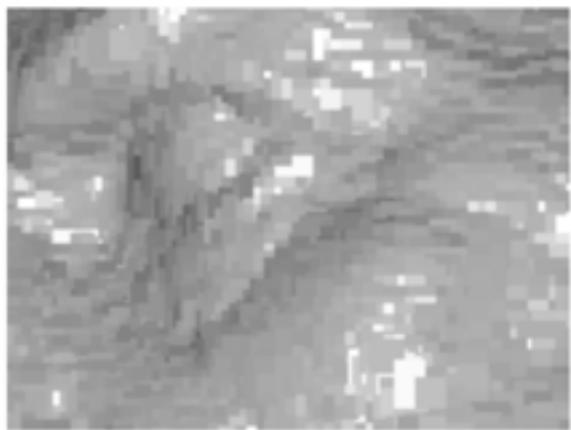
- Stores vertices and normals up to full resolution
- Explicit geometry only — no mesh!



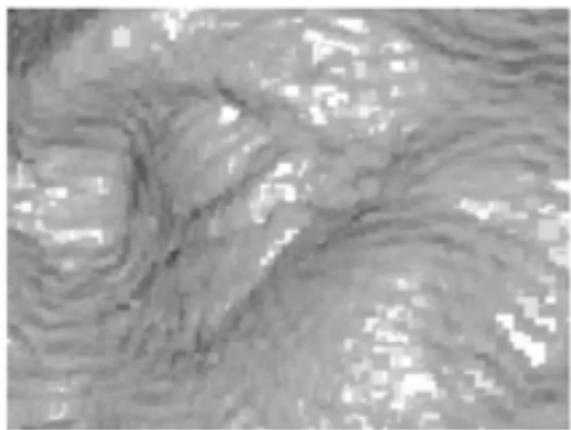
15-pixel cutoff
130,712 points
132 ms



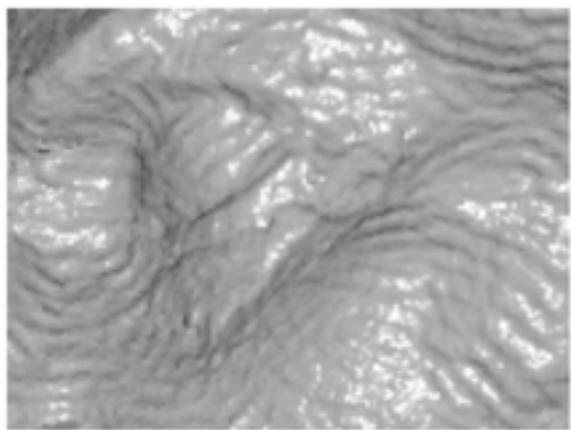
10-pixel cutoff
259,975 points
215 ms



5-pixel cutoff
1,017,149 points
722 ms



1-pixel cutoff
14,835,967 points
8308 ms



Iso-splatting

- Create approximate points near the isosurface using pre-classified points inside the volume
- Optionally, use Newton-Raphson to better fit samples to the isosurface.

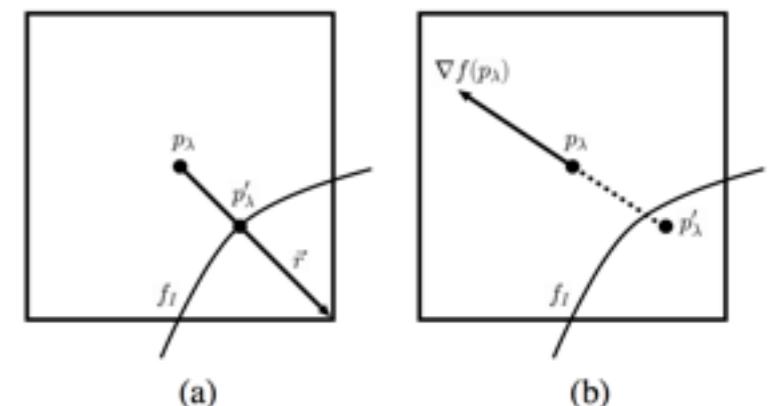
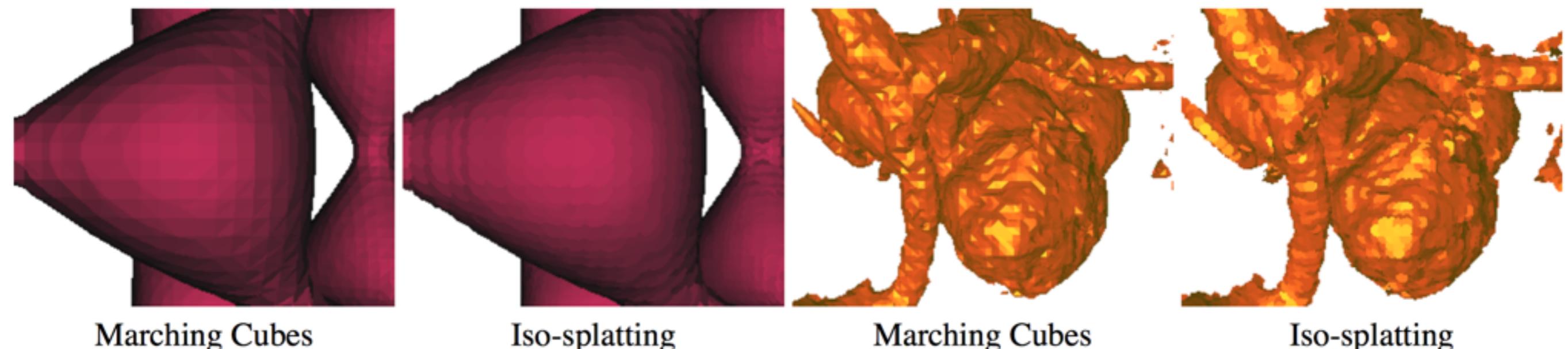


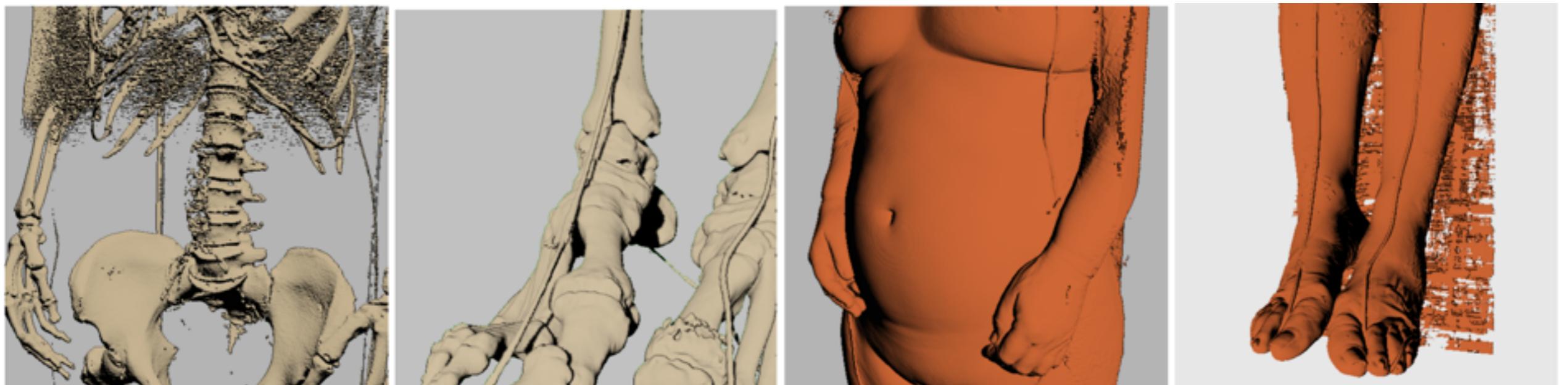
Figure 2. Projection of sample point p_λ onto isosurface f_I , producing point p'_λ using (a) exact projection and (b) approximate projection.



C Co, B Hamann, K Joy. Iso-splatting: A Point-based Alternative to Isosurface Visualization. IEEE Vis 2003.

Hybrid splatting + extraction

- View dependent splatting
- Builds on the point hierarchy idea of QSplat, and view-dependent marching cubes.
- Very fast for its time — but complicated.

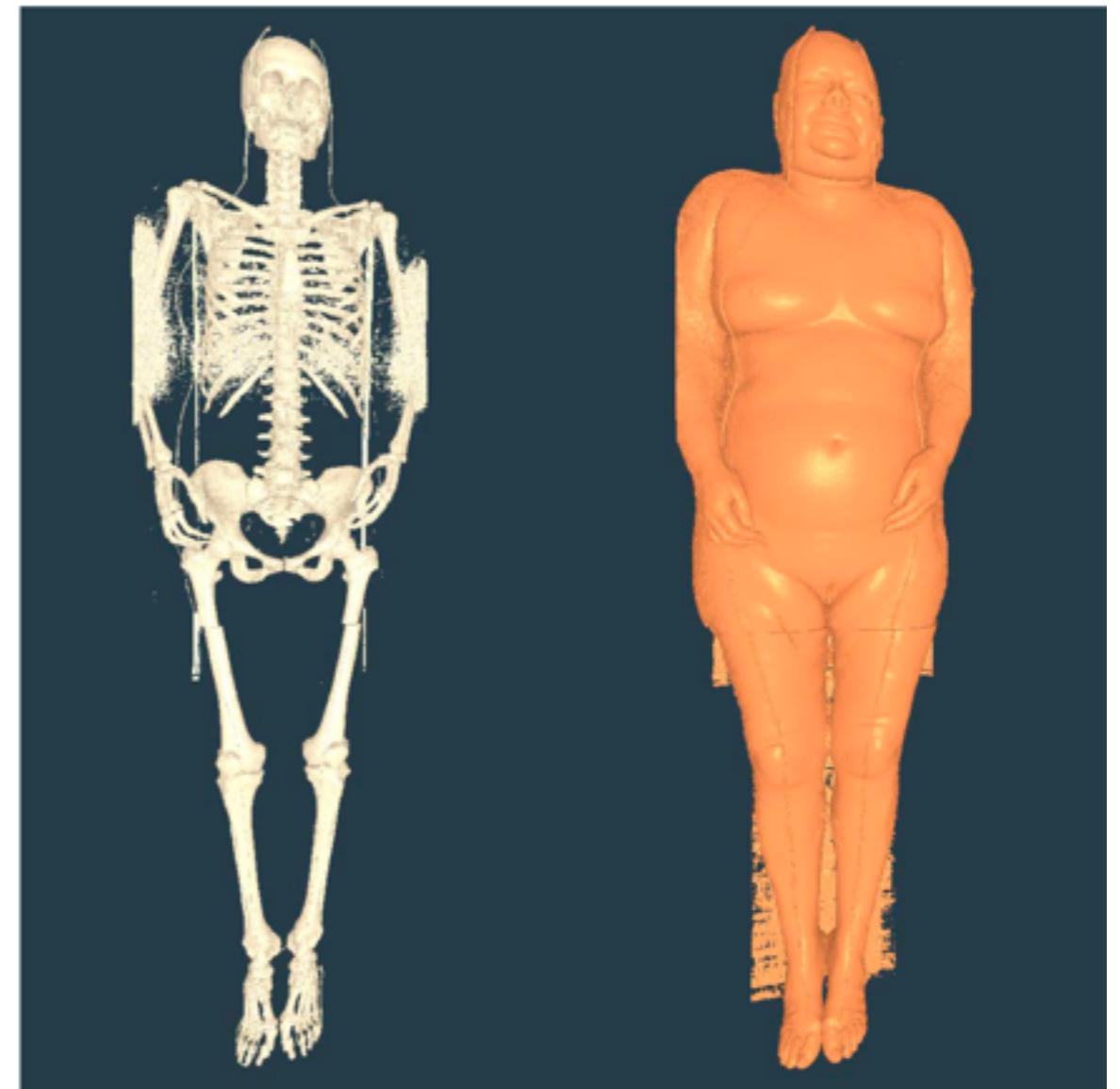


Y Livnat and X Tricoche. Interactive Point-Based Isosurface Extraction. IEEE Vis 2005.

Ray casting / tracing

RTRT

- Accelerate volume with a two-level uniform grid of interval values
- Direct numerical solution for ray intersection with the trilinear isosurface patch
- 1 GB visible female, rendered interactively on an SGI shared-memory machine.



Ray-trilinear isosurface intersection

$$\rho(x_a + tx_b, y_a + ty_b, z_a + tz_b) - \rho_{\text{iso}} = 0.$$

The intersection with the isosurface $\rho(\vec{p}) = \rho_{\text{iso}}$ occurs where:

$$\rho_{\text{iso}} = \sum_{i,j,k=0,1} (u_i^a + tu_i^b)(v_i^a + tv_i^b)(w_i^a + tw_i^b) \rho_{ijk}$$

This can be simplified to a cubic polynomial in t :

$$At^3 + Bt^2 + Ct + D = 0$$

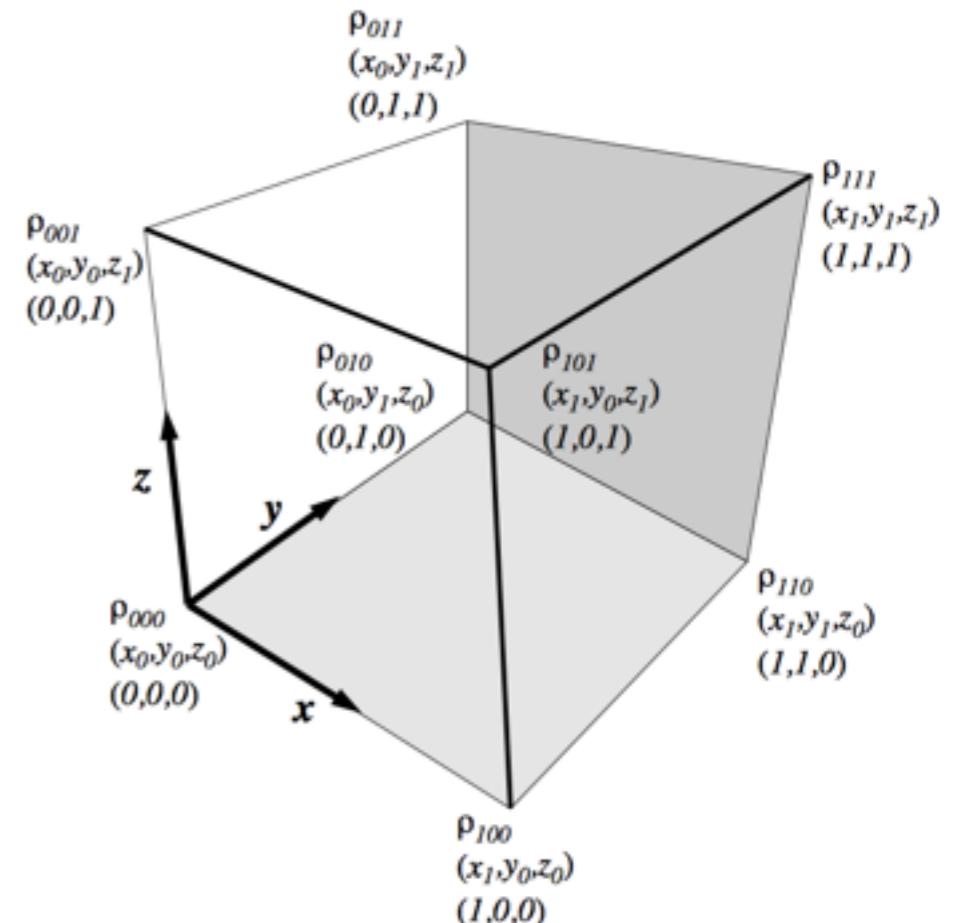
where

$$A = \sum_{i,j,k=0,1} u_i^b v_i^b w_i^b \rho_{ijk}$$

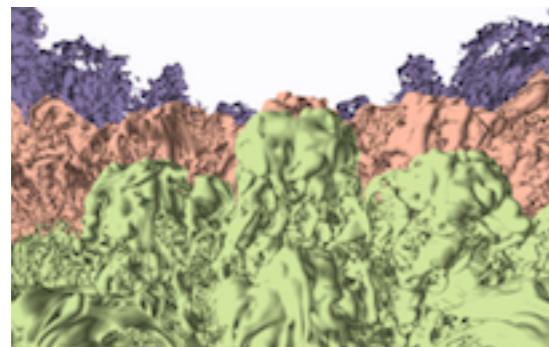
$$B = \sum_{i,j,k=0,1} (u_i^a v_i^b w_i^b + u_i^b v_i^a w_i^b + u_i^b v_i^b w_i^a) \rho_{ijk}$$

$$C = \sum_{i,j,k=0,1} (u_i^b v_i^a w_i^a + u_i^a v_i^b w_i^a + u_i^a v_i^a w_i^b) \rho_{ijk}$$

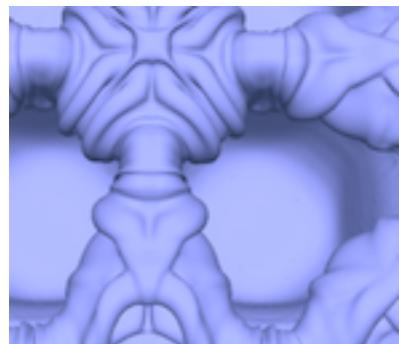
$$D = -\rho_{\text{iso}} + \sum_{i,j,k=0,1} u_i^a v_i^a w_i^a \rho_{ijk}$$



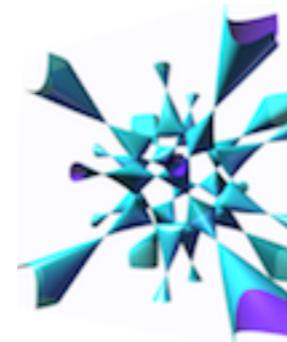
Fast CPU Isosurface Ray Tracing — 2004-2008



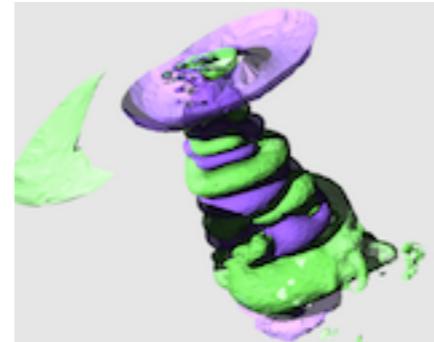
Aaron Knoll, Charles Hansen, and Ingo Wald
Coherent Multiresolution Isosurface Ray Tracing
The Visual Computer 2009



Aaron Knoll, Ingo Wald, Steven Parker, and Charles Hansen.
Interactive Isosurface Ray Tracing of Large Octree Volumes
Proceedings of the IEEE Symposium on Interactive Ray Tracing, Salt Lake City, 2006



Aaron Knoll, Younis Hijazi, Andrew Kensler, Mathias Schott, Charles Hansen and Hans Hagen
Fast Ray Tracing of Arbitrary Implicit Surfaces with Interval and Affine Arithmetic.
Computer Graphics Forum, 2009

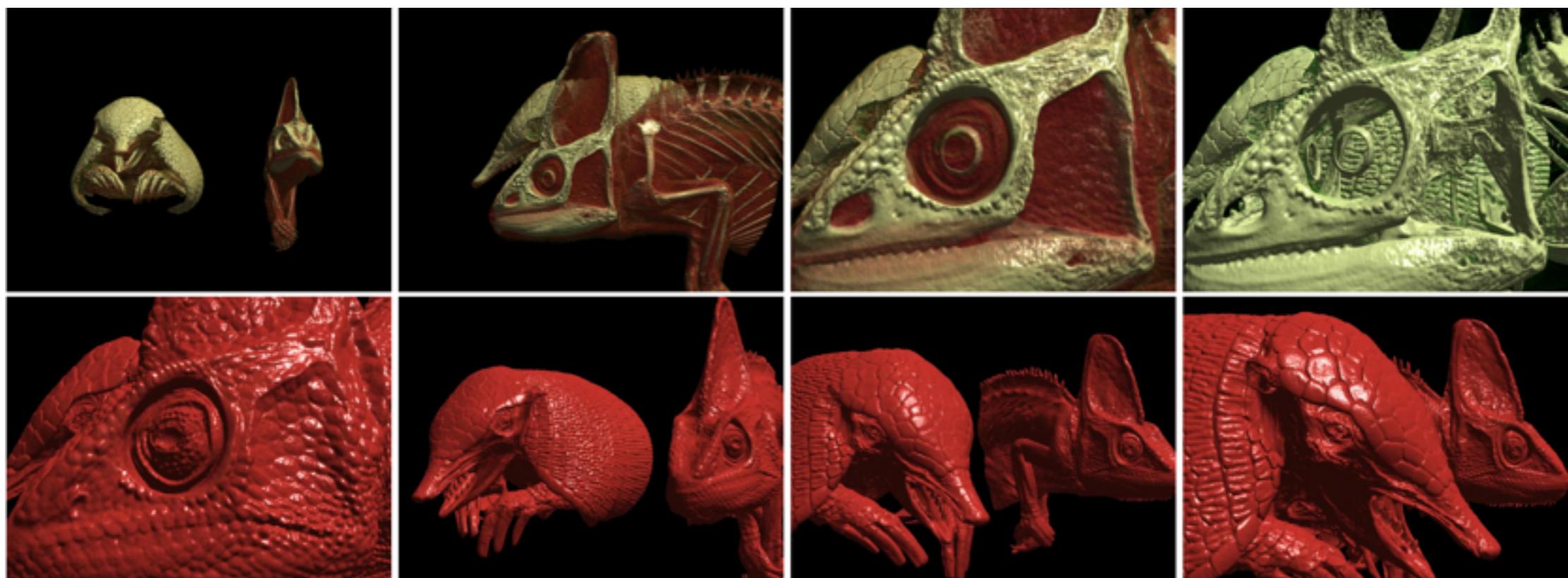
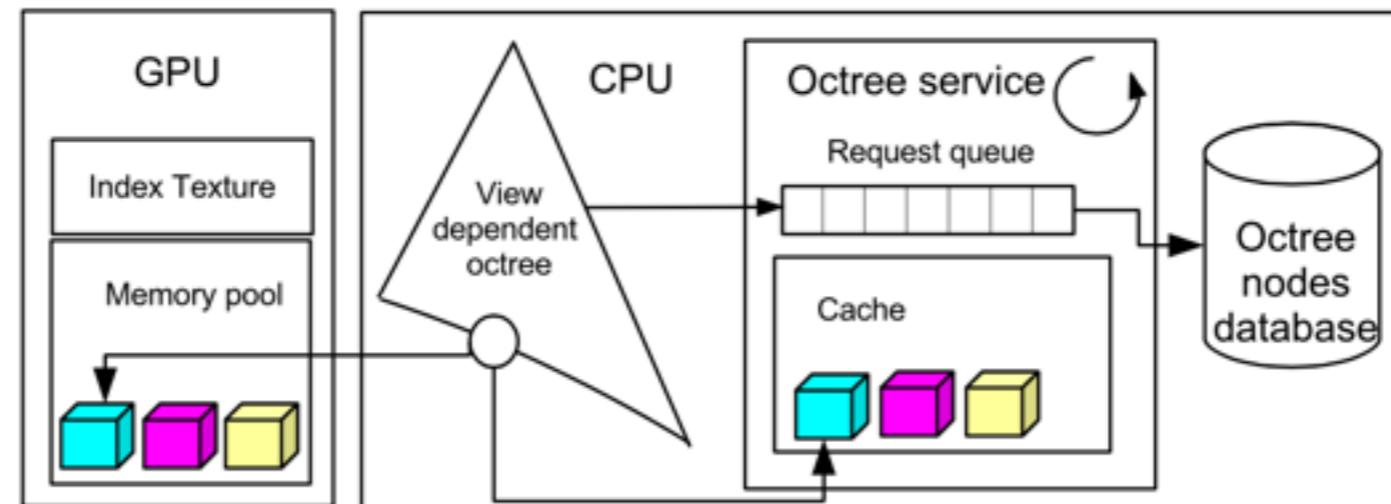


Ingo Wald, Heiko Friedrich, Aaron Knoll, and Charles D. Hansen
Interactive Isosurface Ray Tracing of Time-Varying Tetrahedral Volumes
IEEE Visualization 2007



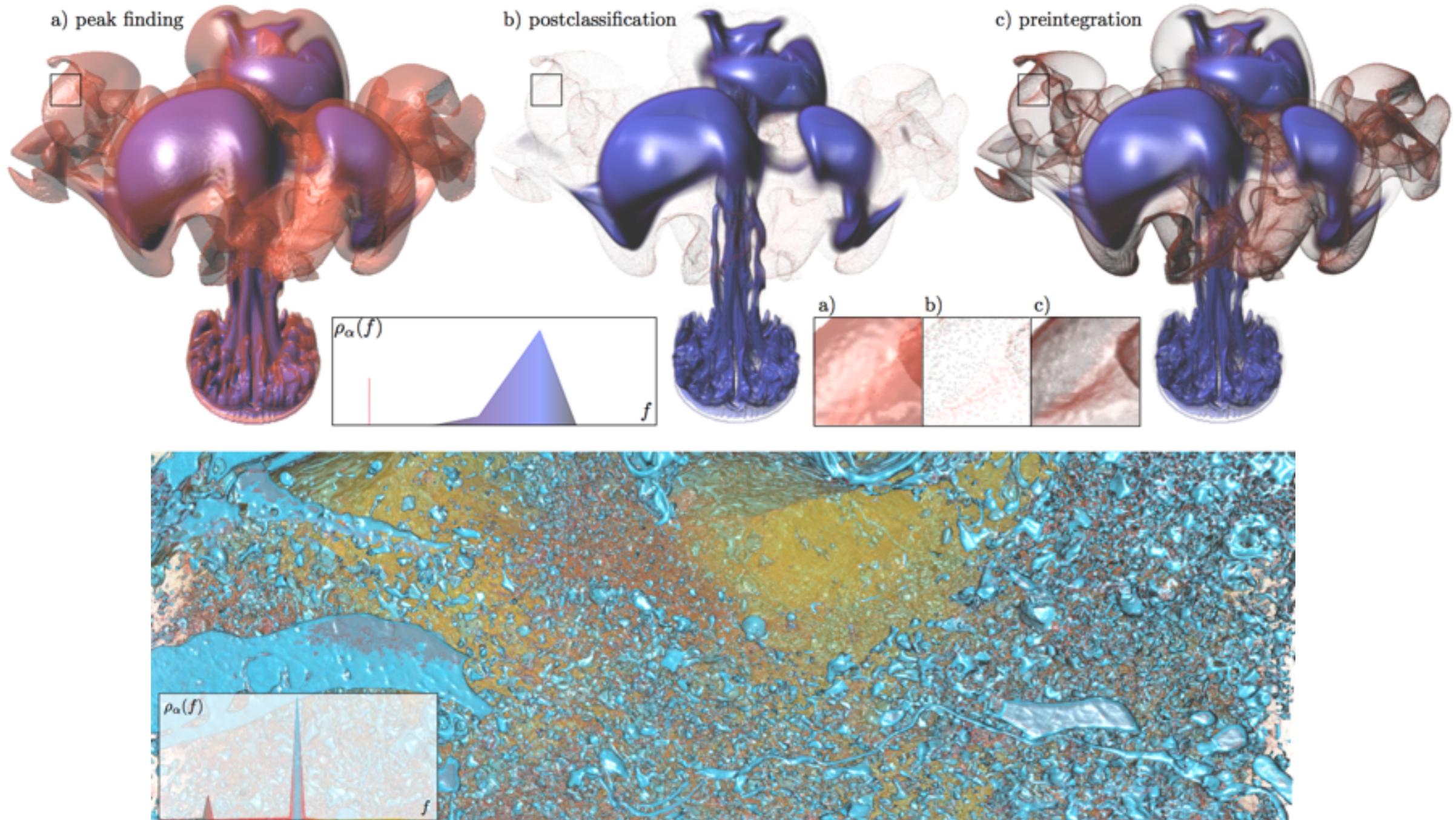
Chris Wyman, Steven Parker, Pete Shirley, and Charles Hansen.
Interactive display of isosurfaces with global illumination.
IEEE TVCG 2006.

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

Peak finding: combining isosurfacing and volume rendering



A Knoll et al. Volume Ray Casting with Peak Finding and Differential Sampling.
IEEE Visualization 2009.

Thoughts on isosurfacing

- 2012: “No one uses volume rendering, it’s too slow and hard.”
- 2015: “No one uses isosurfacing, it’s too ugly and limiting.”
- As a pure visualization modality, isosurfacing is on its way out
 - Volume rendering is no longer “expensive”, has decent implementations in Paraview and VisIt, as well as Voreen, ImageVis3D, Intel OSPRay, NVIDIA IndeX, etc.
 - Users are starting to “get” transfer functions...
 - Marching cubes is still nasty and expensive, and has pitfalls
- As a core technique, isosurfacing will be here for a long time
 - Schroeder et al. “Flying Edges: A High Performance Scalable Isocontouring Algorithm”, IEEE LDAV 2015.
 - We’ll always need ways to convert from implicit to explicit and visa versa, for computation.

Next up:

- Thursday 11-12: Vector and tensor field visualization.