Isosurface Rendering

What is Isosurfacing?

- An isosurface is the 3D surface representing the locations of a constant scalar value within a volume
  - A surface with the same scalar field value

- Isosurfaces form the 3D analogy to the isolines that form a contour display on the surface

- Isosurfaces have the root in medical imaging where surfaces of constant density are often generated
  - Bone skeletons, organ boundaries

Marching Cubes Algorithm

- To approximate an isosurface of a 3D scalar field or function
  - Input:

    Cubic grid data (voxels)

    Isovalue
  - Output:

    Set of triangles approximating surface for a given isovalue

- March through each of the cubes (voxels) replacing the cube with appropriate set of triangles
  - Determine if and how an isosurface would pass through it
  - Generate polygonal isosurface on a voxel-by-voxel basis

- References:

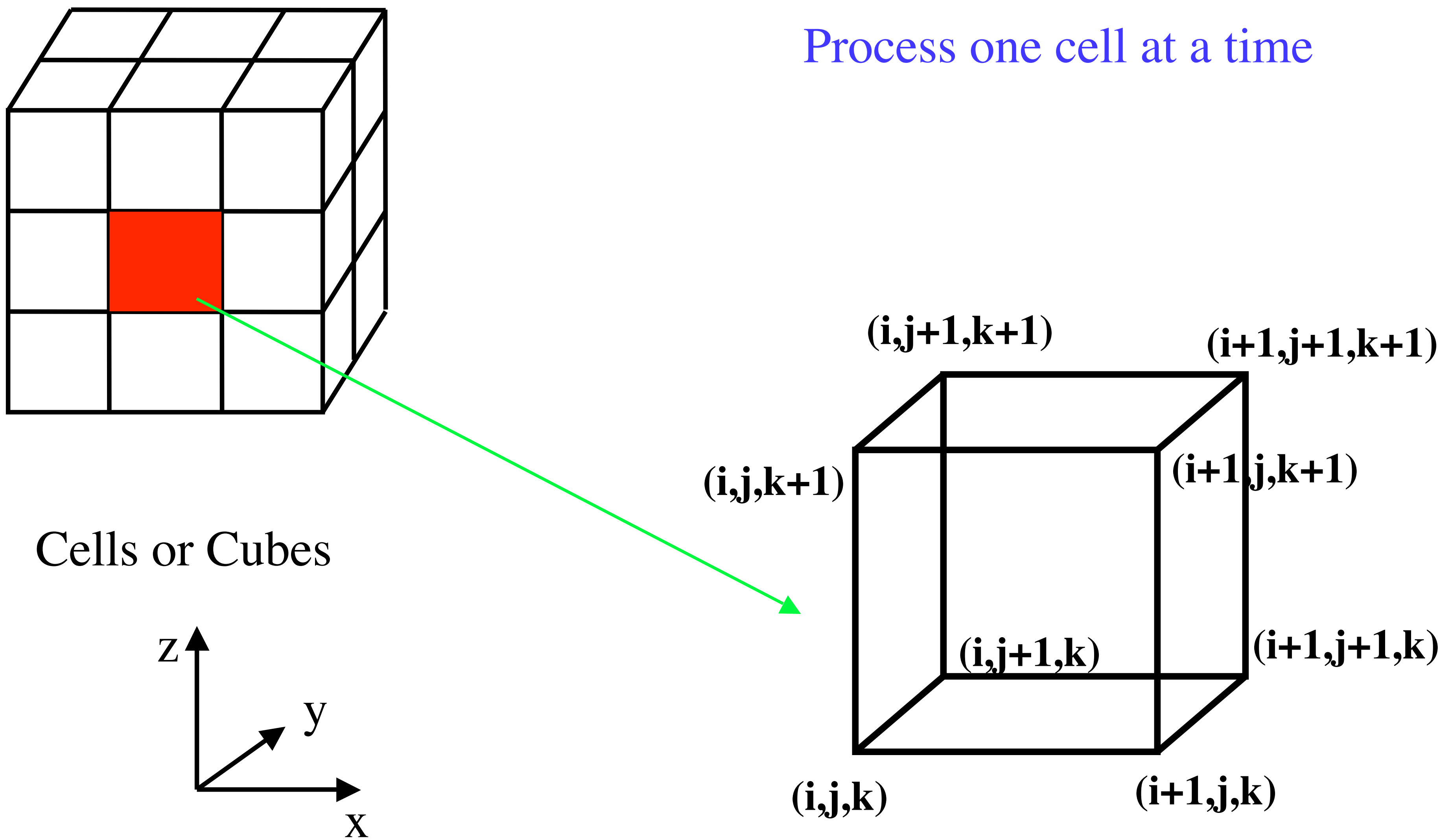
  - Lorensen and Cline, "Marching Cubes: A High-resolution 3D surface construction algorithm" *Computer Graphics*, 21(3), 163, July 1987
  - Neilson and Hamann, "The Asymptotic Decider: Resolving the ambiguity in Marching Cubes" Proc. Vis. 1991, San Deigo, CA, Oct. 22-25.
  - Sharman, "The Marching Cubes Algorithm," 1998 www.exaflop.org/docs/marchcubes

Basic MC Algorithm

- Select a cell
  - Process each cell, one at a time
- Classify the inside/outside state of each vertex
- Create an index
  - Find equivalent basic configuration by switching marked points or rotation
- Get edge list from the table
  - Produce a set of triangles
- Interpolate the edge location
  - Mid-edge (default)
  - Linear interpolation along edge
- Go to the next cell

Step 1: Select a Cell

Process one cell at a time

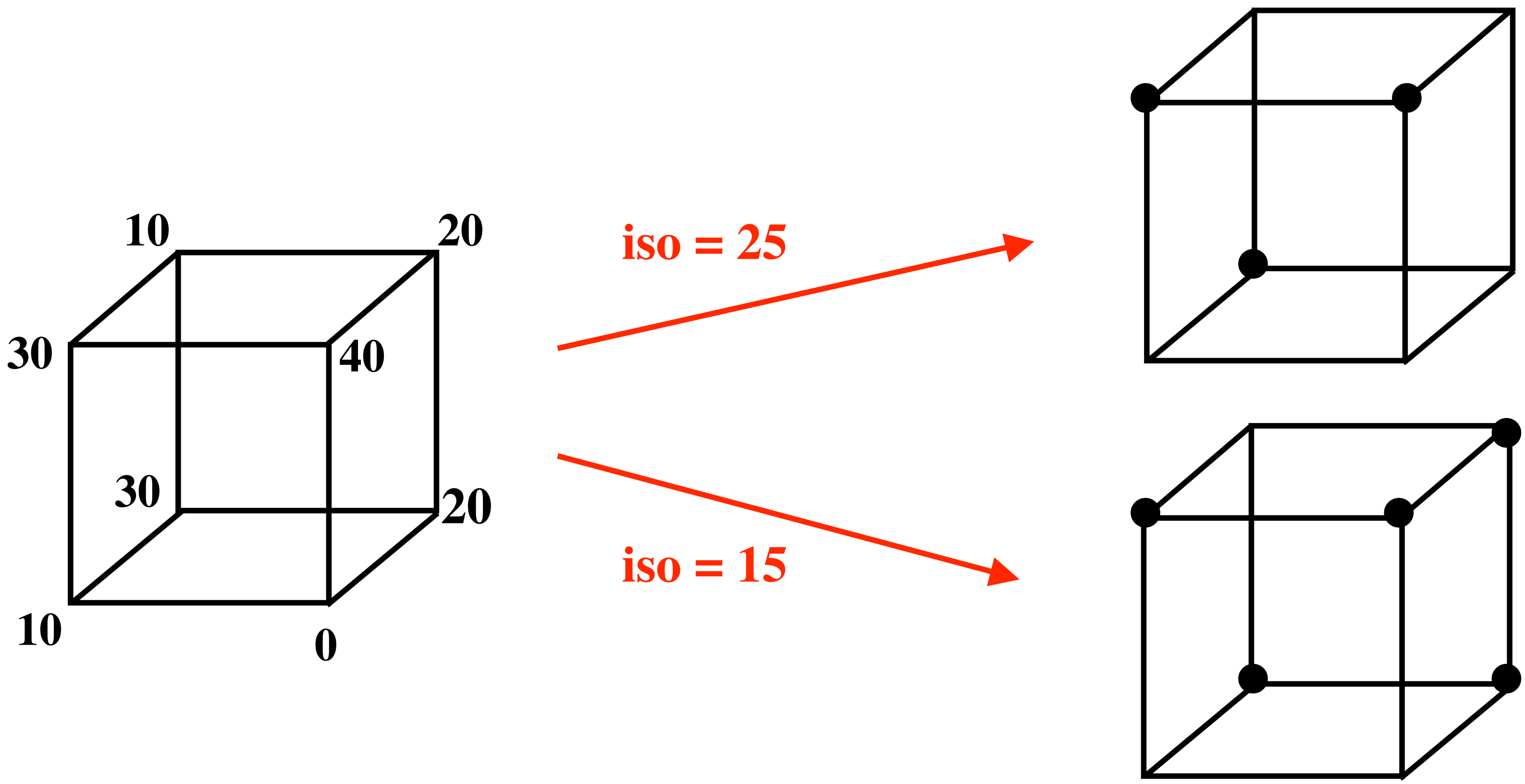


Step 2: Classify States of Vertices

- Determine the inside/outside state of each vertex of the cell:

  Inside isosurface (value >= iso-value) **● = inside**

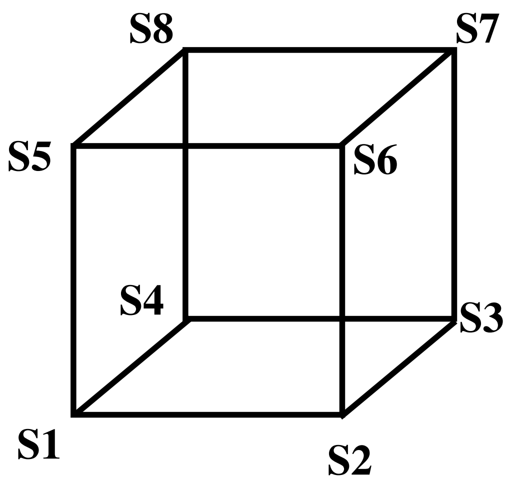
  Outside isosurface (value < iso-value) **unmarked = outside**

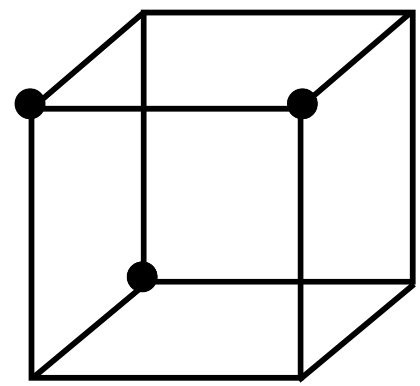


Step 3: Create Index

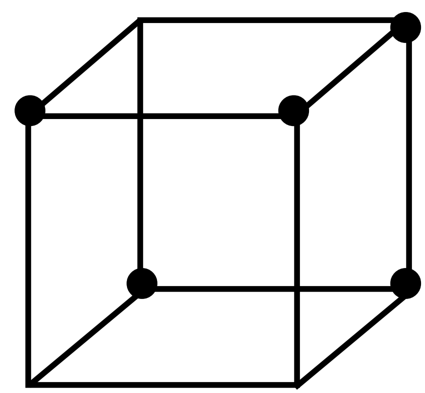
Marked vertex by ● = inside = 1

Unmarked vertex = outside = 0





00011100



?

index	S1	S2	S3	S4	S5	S6	S7	S8

or

index	S8	S7	S6	S5	S4	S3	S2	S1

Forms the bits of a binary number between 0 and 255 for an 8-vertex cube

Step 4: Get Edge List

- An index corresponds to a list of edges the isosurface cuts through
  - ➢ Given an index, get edge list from table which is pre-created

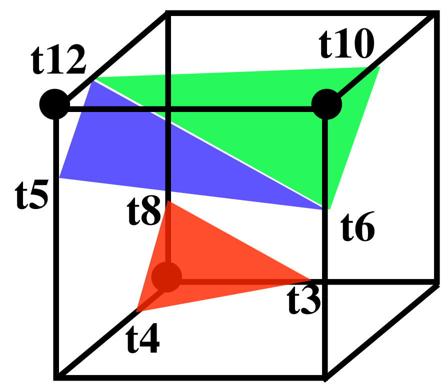
- 2D cell index: 4 bits, 2^4 (16) cases

  3D cell index: 8 bits, 2^8 (256) cases

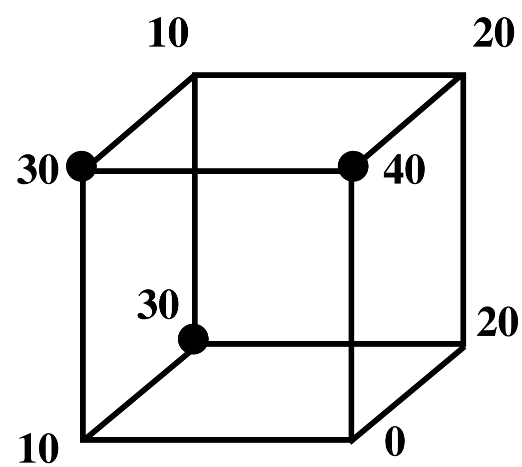
Example:

Index = 00011100

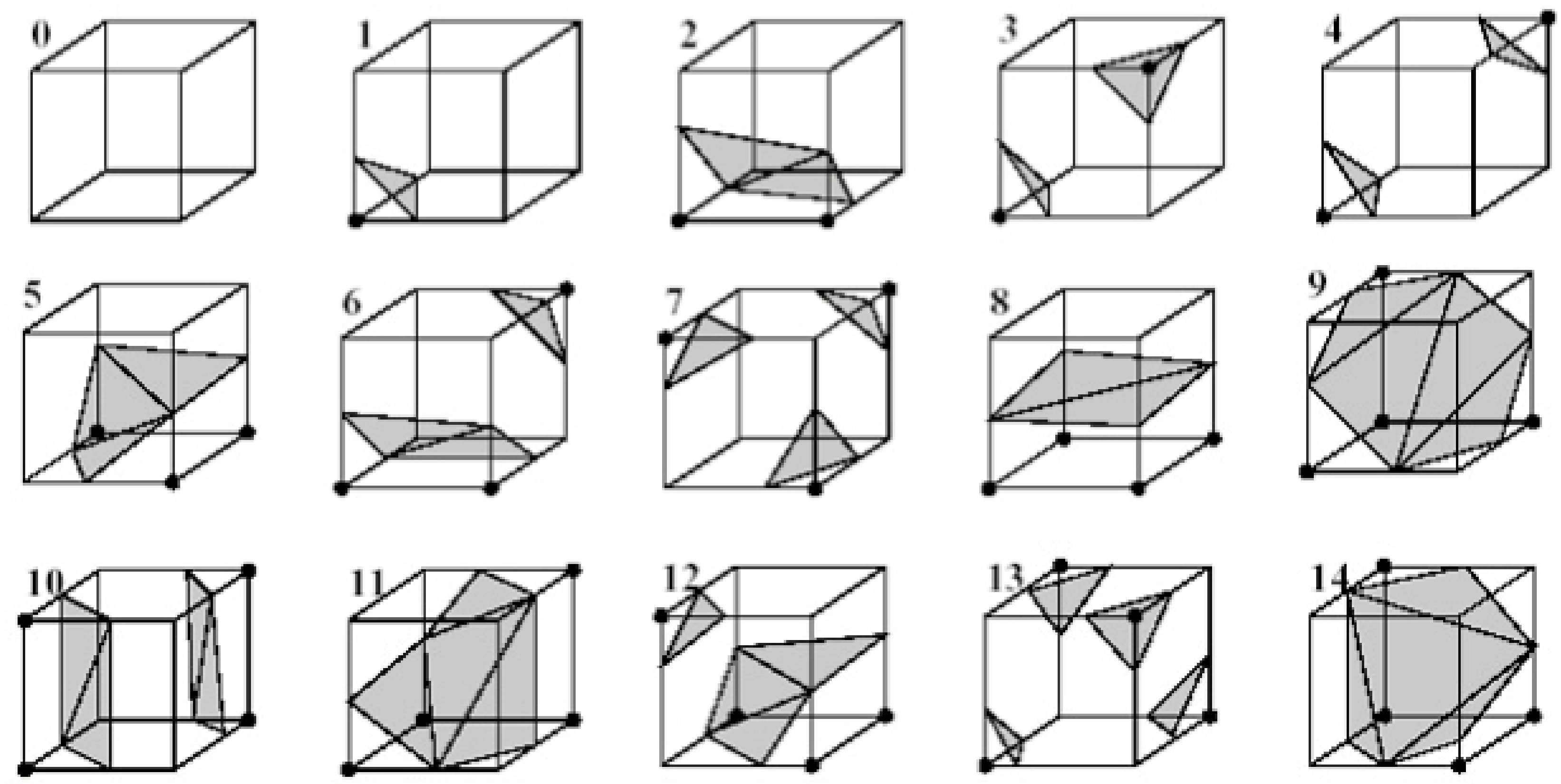
triangle 1 = t3, t4, t8
triangle 2 = t5, t6, t12
triangle 3 = t6, t10, t12



For isovalue of 25



15 Basic Cases of 3D Cells



Symmetries: Complementary and rotations

Pre-defined look-up table enumerates

a) how many triangles will make up the isosurface segment passing through the cube

b) which edges of the cubes contain vertices of triangles, and in what order

Step 5: Interpolation of Triangle Vertices

- For each triangle, find an vertex location along the edge using linear interpolation of the values at the edge's two end points

$$x = x(i) + fac * \delta x$$
$$y = y(i) + fac * \delta y$$
$$z = z(i) + fac * \delta z$$

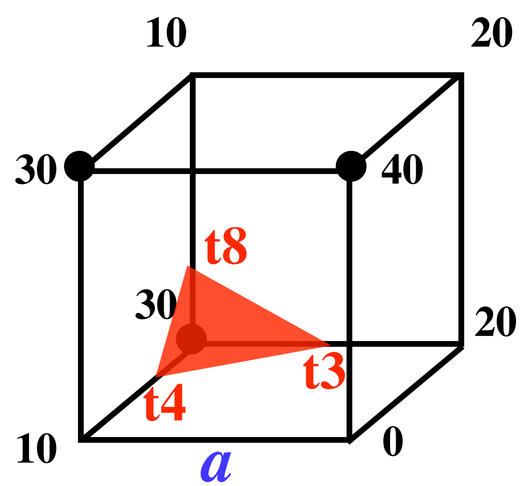
where $fac = \left(\dfrac{S(i+1) - S_{iso}}{S(i+1) - S(i)} \right)$

- Vertices of triangle

t3 = $(x(i) + a/2, y(i), z(i))$

t4 = $(x(i), y(i) + a/4, z(i))$

t8 = $(x(i), y(i), z(i) + a/4)$



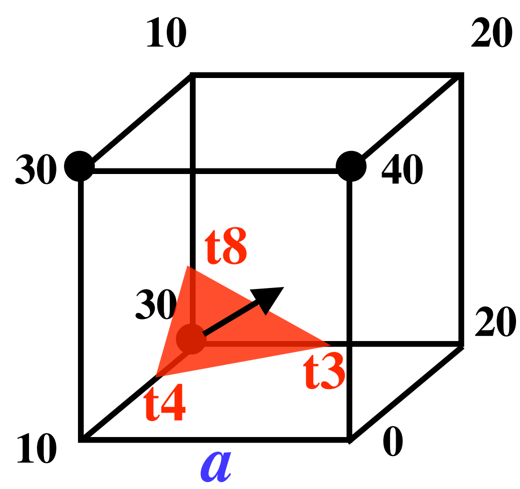
Surface Normals

- Smooth shading of isosurface segments requires the normal to the surface
  - Calculate a unit normal at each cube vertex using central differences.
  - Interpolate the normal to each triangle vertex.

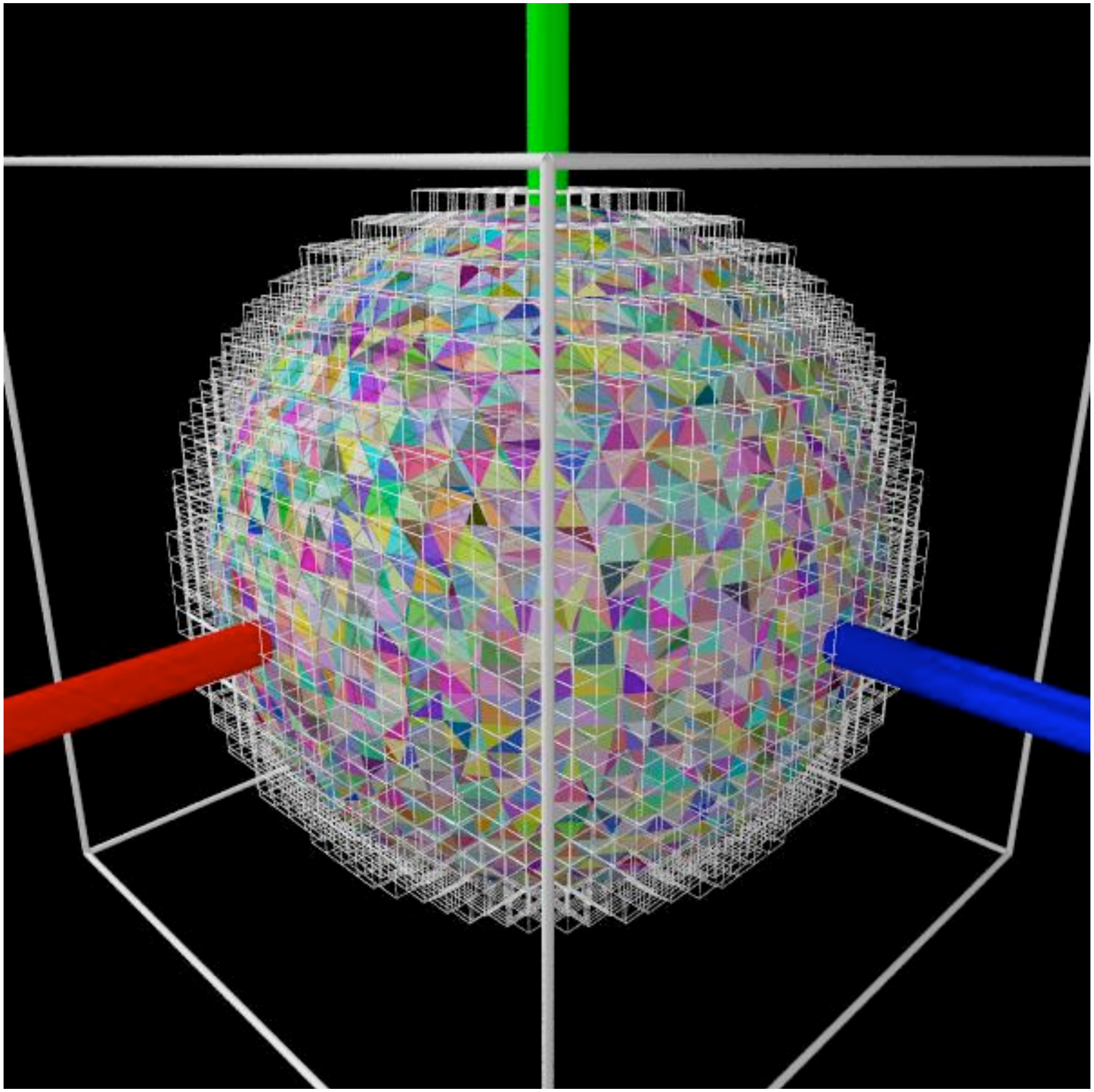
$$\left[\frac{dS(x,y,z)}{dx},\frac{dS(x,y,z)}{dy},\frac{dS(x,y,z)}{dz}\right]$$

Where *dx, dy, dz* are the lengths of the cube; and *dS*'s are the central differences.

- A normal vector: a perpendicular distance to the triangle from the marked vertex pointing away



A Spherical Isosurface



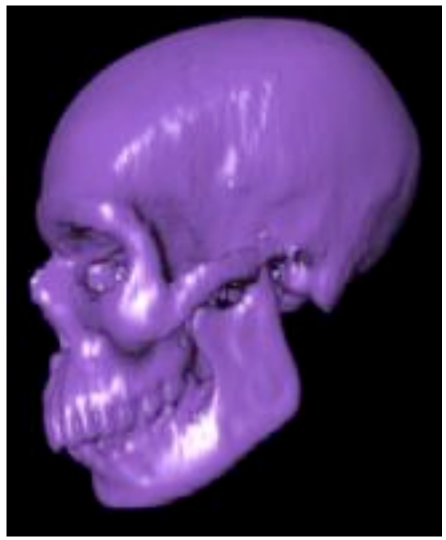
Scalar function:
$f = \sqrt{(x^2+y^2+z^2)}$

Shown are the cells where the field is being evaluated

Triangles are randomly colored.

www.cs.ubc.ca

Images Produced by Marching Cubes



www.erc.msstate.edu

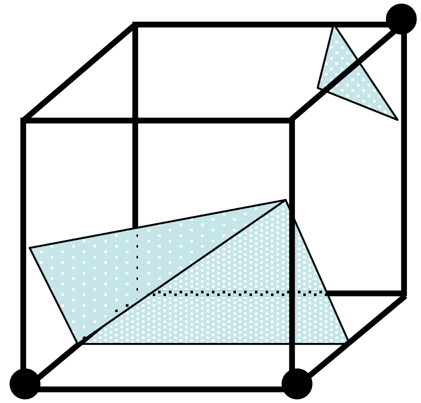
MC's Performance

- Benefits:
  - High quality images:
    - Original data and structure is preserved
    - Gradient data reflected in normal vectors
  - Divide and conquer:
    - good for parallel implementation

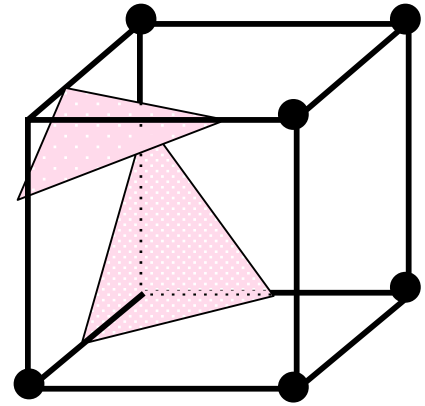
- Problems:
  - Inefficient:
    - Slow in computation and large in memory requirement
      - large number of triangles generated
    - 100^3 dataset requires several megabytes memory
  - Missing voxels
    - How to fill up the data
  - Ambiguities
    - Isosurface polygons may be discontinuous across two adjacent cells
    - Triangles smaller than a single pixel

Ambiguity in Marching Cubes

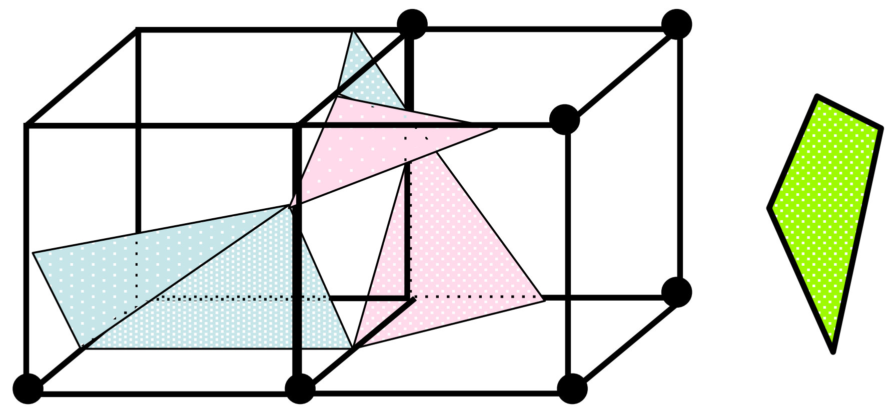
- Ambiguous cases:
  - **3, 6, 7, 10, 12, 13**
- Adjacent vertices in different states, but diagonal vertices in the same state
- Ambiguous cases may cause holes



case 6



case 3c



Isosurface polygons are disjoint across the common element surface

Resolving the Ambiguity

- Using different triangulations, leading to consistency

  - Asymptotic deciders

  - Improved Marching Cubes

  - Marching tetrahedra

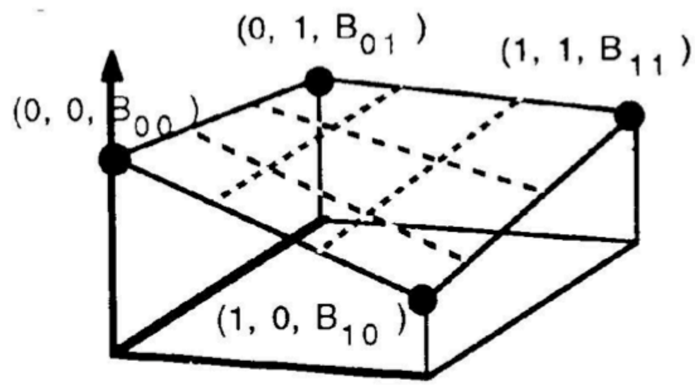
The Asymptotic Decider

- Techniques for choosing which vertices to connect on ambiguous face (Nielson and Hamann, 1991)

- Uses bilinear interpolation over ambiguous face

- Consider:
  - Face is unit square
  - B_{ij} values of four corners
  - $\{(s,t): 0 <= s <= 1, 0 <= t <= 1\}$

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



AD (Contd.)

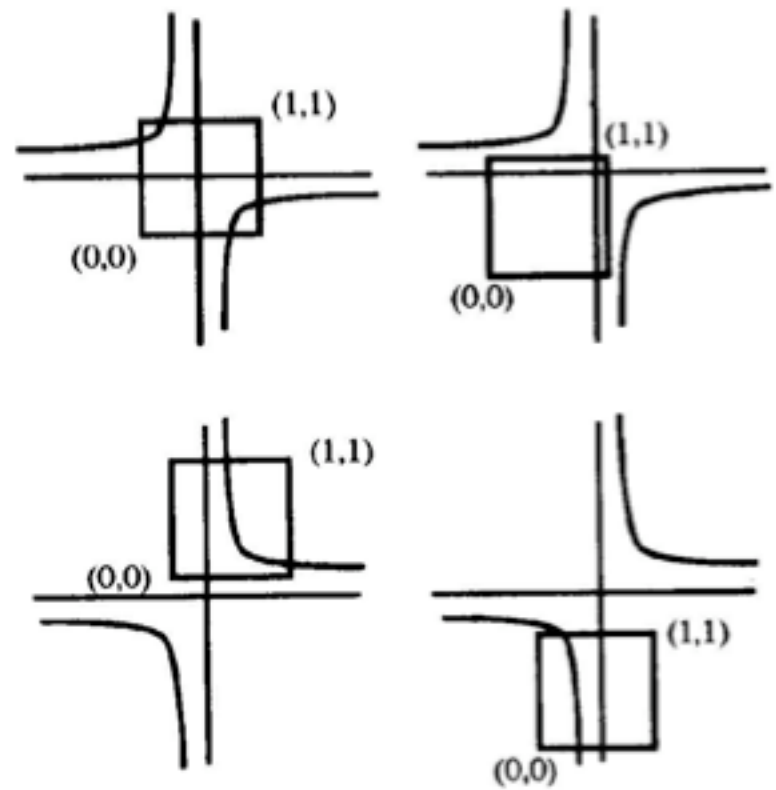
- Contour curves of B are hyperbolas

  $\{(s,t): B(s,t) = a\}$,

  where a is isovalue

- Ambiguous case: both components of hyperbolas intersect the domain

- Criteria for connecting vertices based on whether they are joined by a component of hyperbola



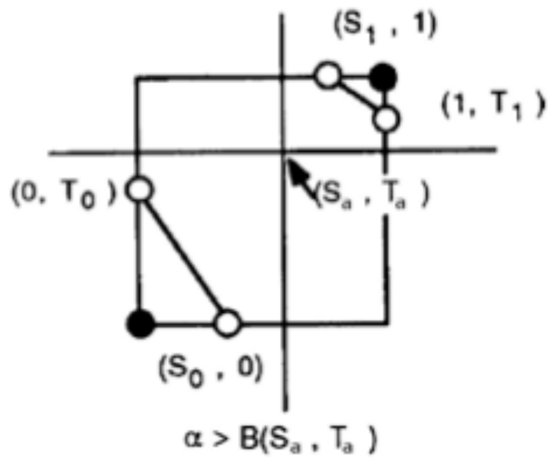
AD (Contd.)

- Selection determined by comparing values a and $B(S_a,T_a)$
  - a = contour value
  - $B(S_a,T_a)$ = value of bilinear interpolant at intersection point of the asymptotes

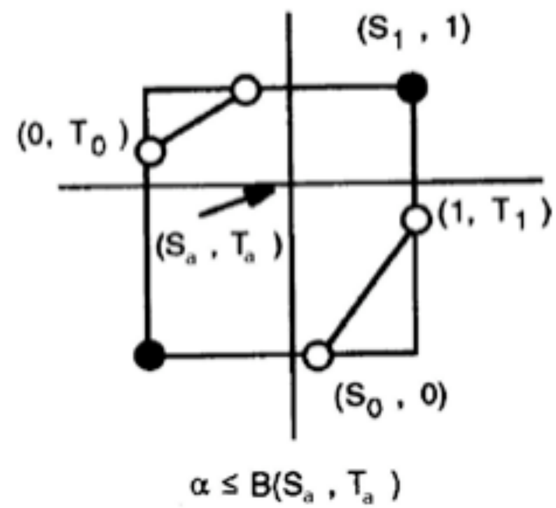
- If $a > B(S_a,T_a)$
  - connect $(S_1,1)$ to $(1,T_1)$ and $(S_0,0)$ to $(0,T_0)$

  else
  - connect $(S_1,1)$ to $(0,T_0)$ and $(S_0,0)$ to $(1,T_0)$

- Possible triangulations
  - Two or more.



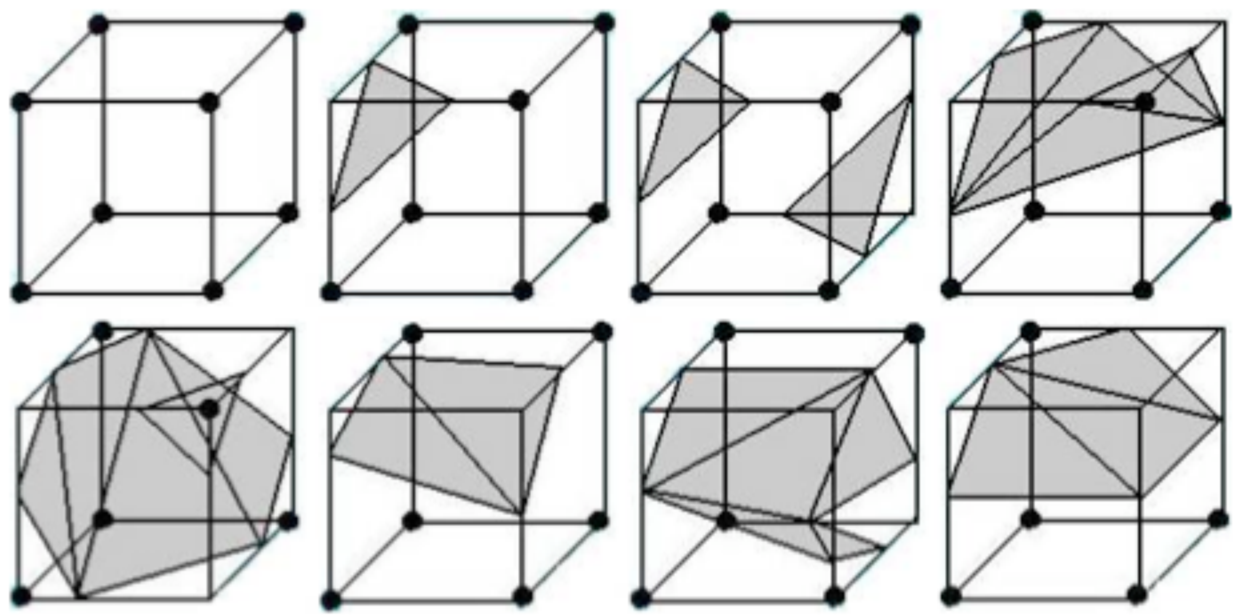
Separated



Not Separated

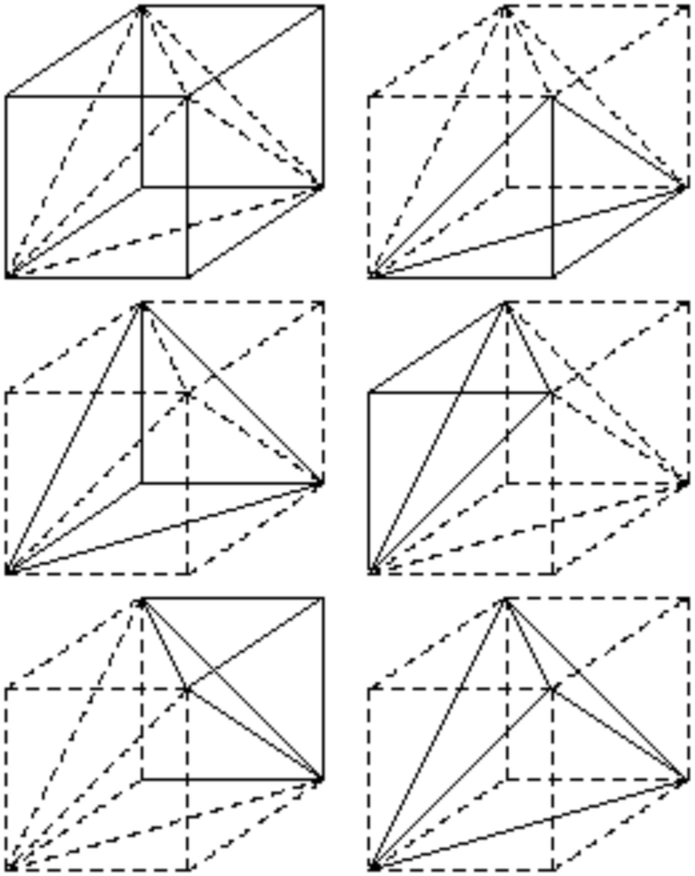
Improved Marching Cubes

- 8 extra cases to consider (Shoeb, 1998)
  - They do not assume the complimentary cases to be equivalent
- Choose cases so that shared sides have same connections between vertices



Marching Tetrahedra

- Tessellates the cube with tetrahedron
  - Every tetrahedron has four nodes and six edges
  - 5 tetrahderons
  - Requires more triangles
- No ambiguous cases exist
- May result in artificial bumps in the isosurface
  - Interpolation along the face diagonals



Trilinear Interpolant within the Cell

- Improve the representation of the surface in the interior of each grid cell
  - Model the topology of trilinear interplolant within the cell

$$S(x,y,z) = a + bx + cy + ez + gxy + fxy + dyz + hyxz$$

Where $a = S_{000}$, $b = S_{001}$-S_{000}, $c = S_{010}$-S_{000}, and so on

- Represent different topologies including the possibility of tunnels
  - To deal with interior ambiguity

- Make surface visually continuous as the data and threshold change in value.

Implicit Isosurfaces

Particle Sampling

- Volume data is sampled at regular points, and the results of the sampling are displayed as dots
- Using point primitives for display
  - Display consists of a dense group of points which imply the surface
  - Rendering points faster than rendering polygons
  - Geometric operations such clipping and merging data are simple with points
- Color and density of points can vary with the magnitude of the scalar value within the specified range
- Display the points of constant scalar value within the entire 3D volume as an implicit isosurface

Shape Function Interpolation

- Shape functions are used to interpolate the element data values
- Generate a continuum of points at any desired density by using a small increment in the parametric *u*, *v* and *w* values
- A linear 8 vertex shape function

$$S(u,v,w) = \sum_{i=1}^{8} \frac{1}{8} S(i)\left[(1 + uu(i))(1 + vv(i))(1 + ww(i))\right]$$

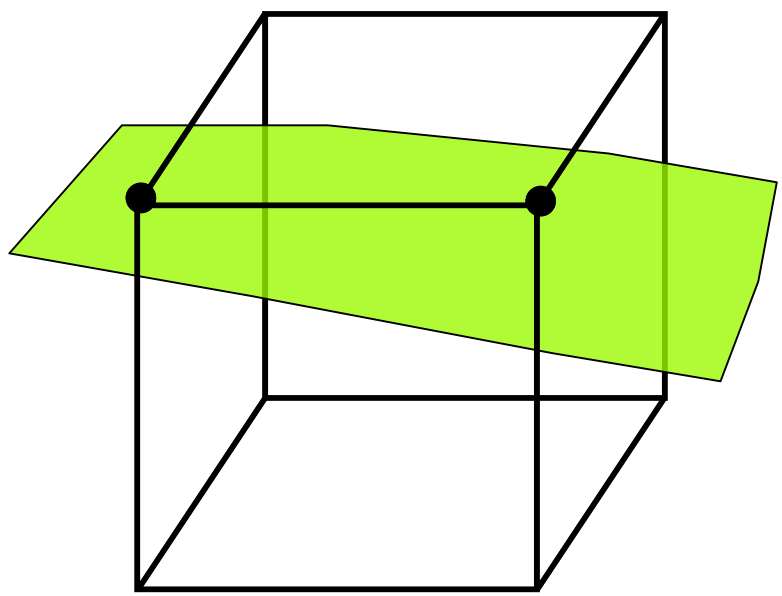
Dividing Cubes Algorithm

- Generates isosurface using dense cloud points
- Use point primitives unlike triangles in Marching Cubes
- Conditions
  - Large number of points
  - Density of points >= screen resolution
  - Lighting and shading calculations

H. Cline, W. Lorensen, S. Ludke, C. Crawford, and B. Teeter, "Two algorithms for the three-dimensional reconstruction of tomographs" *Medical Physics*, vol. 15, no. 3, May 1988

Find Intersecting Voxel

- Select a voxel (cell) and determine whether the isosurface passes through it
  - ➢ Whether there are scalar values at vertices both above and below the iso-value

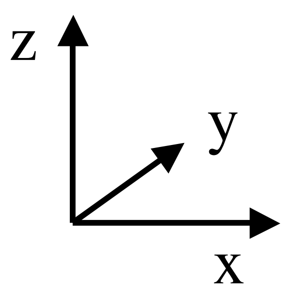


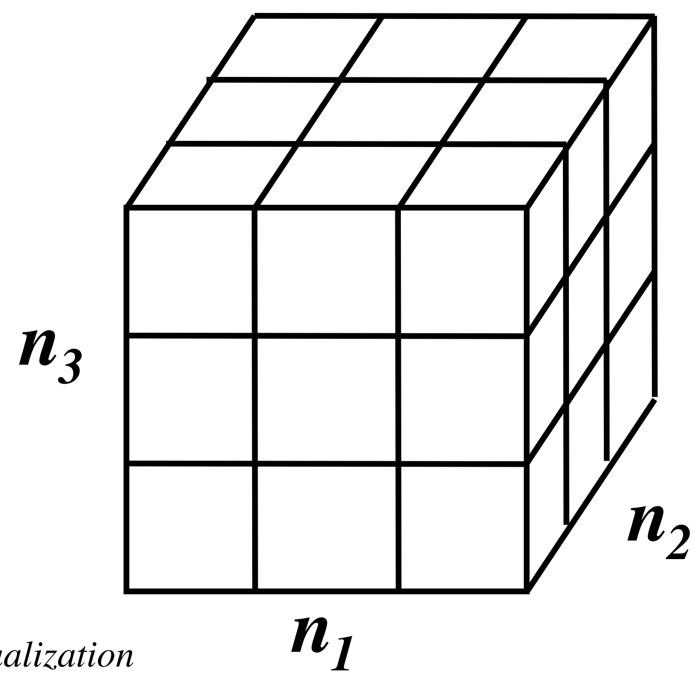
- Inside isosurface

Subdivide Voxel

- The voxel is subdivided into a regular grid of $n_1 \times n_2 \times n_3$ subvoxels

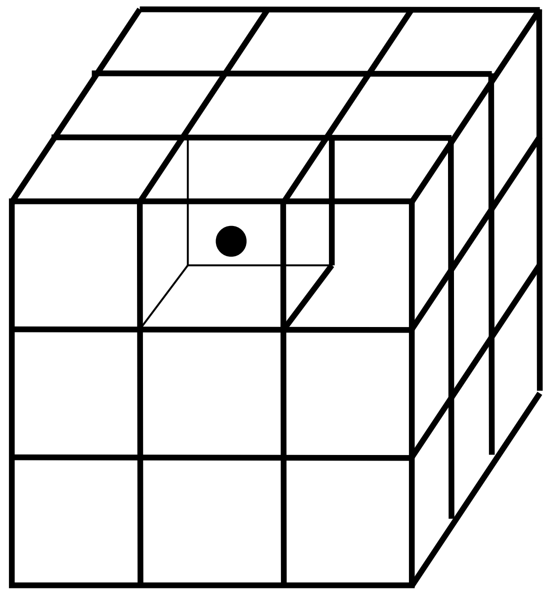
- $n_i = w_i / R$,
  where R is screen resolution and w_i is width of the voxel





Generate Points

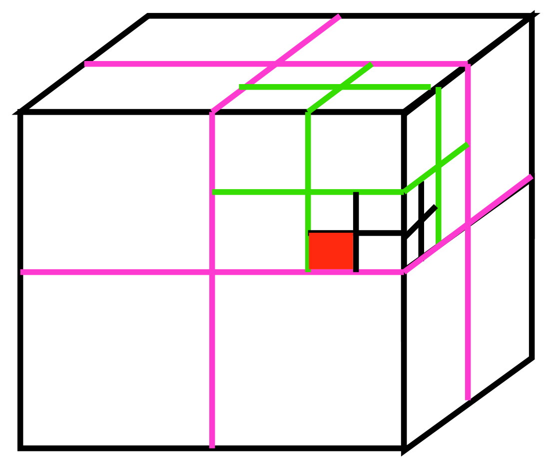
- Scalar values at the subpoints are generated using the interpolation function
- Find whether the isosurface passes through each sub-voxel
- If it does, generate a point at the center of the subvoxel and compute its normal
- Collection of all such points compose the Dividing Cubes' isosurface



Recursive Implementation

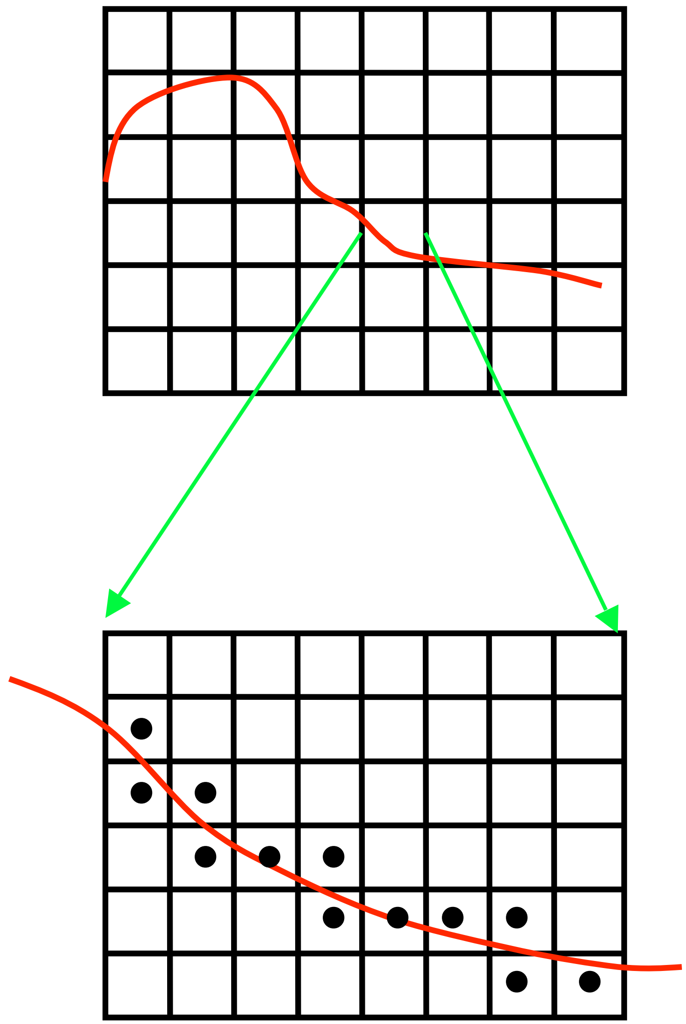
- Recursively divide the voxel as in octree decomposition
- Scalar values at the new points are interpolated
- Process repeats for each sub-voxel if the isosurface passes through it
- This process continues until the size of the subvoxel =< ***R***

  A point is generated at the center of the sub-voxel



Hierarchy of spatial subdivisions to form an octree

Dividing Squares' Contour



Dividing Cubes' Image

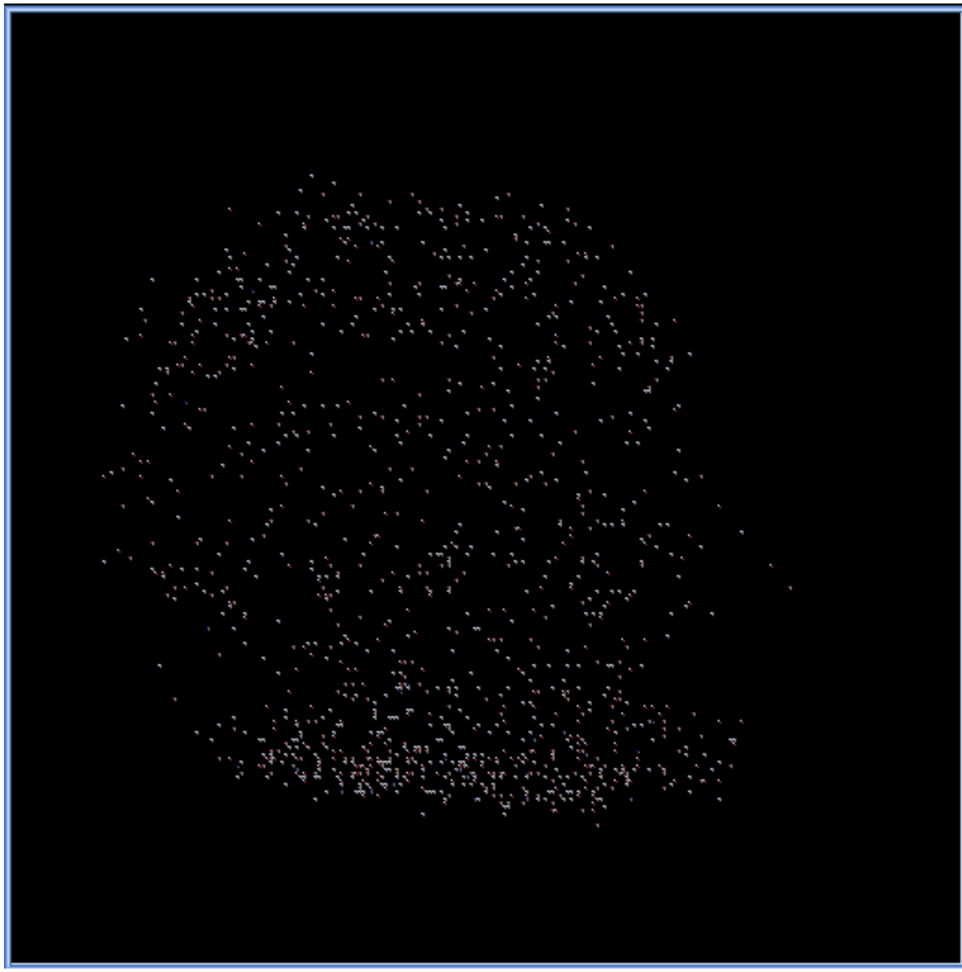


Image of human head

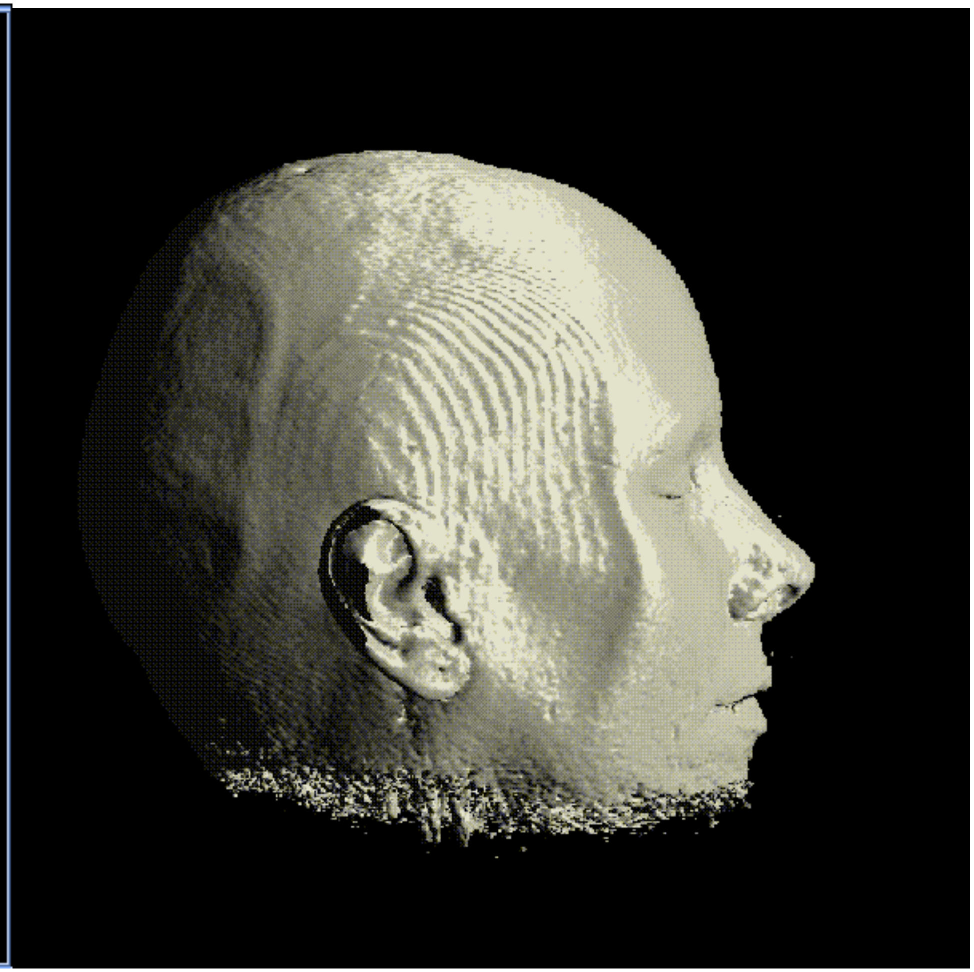


Image with voxel subdivision into 4x4x4 cubes

www.cs.umbc.edu

Fast Isosurface Extractions

- View dependent isosurface extraction
  - Very large and complex isosurfaces

    Multiple non-overlapping polygons may project onto individual pixels

    Some sections may be occluded by the other sections of the isosurface
  - Extract only the visible portions of the isosurface.

- Interactive ray tracing of isosurfaces
  - Generate a single image of isosurface from a given viewpoint

    No geometry generated but an analytical isosurface intersection computation done
  - Use ray-tracing in which one or more rays are sent from viewpoint through each pixel of the screen and into the scene

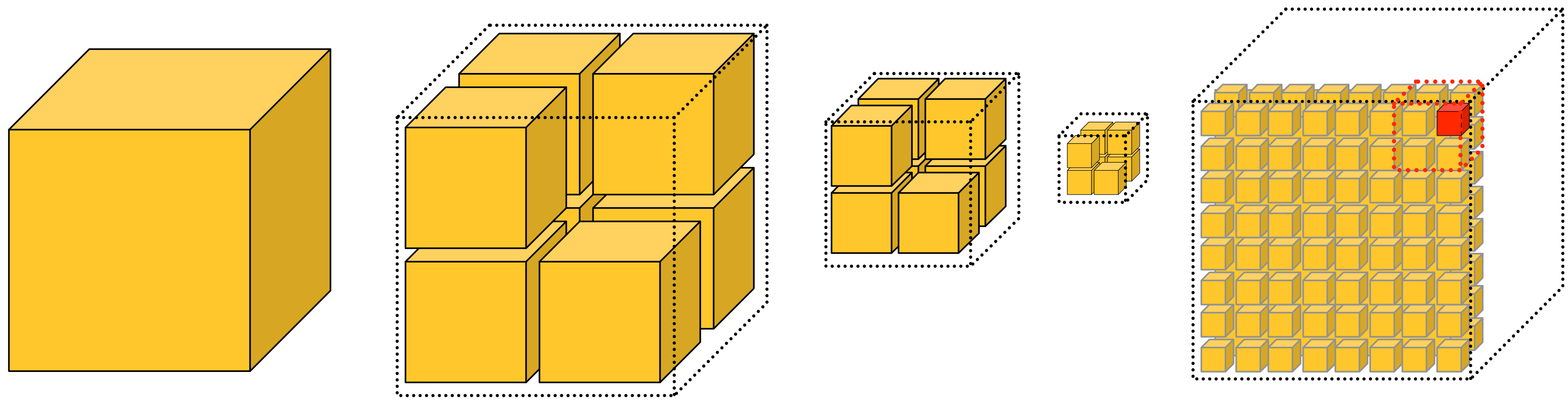
    Parallel processing.

- Near optimal isosurface extraction (NOISE)
  - Maps the search phase onto a two-dimension space (the span space)

    Time complexity: $O(\sqrt{n}+k)$ or $O(\log n = k)$, where k is the size of the isosurface and n is the size of the data set.

Octree-Based Isosurface Extraction

- Octree with Marching Cubes Algorithm *Wilhelms and van Gilder ACMTG 1992*
- Construct an octree (min and max values)
- Skip nodes (cells within) if they do not contribute to the isosurface
- Perform local triangulation in each contributing cell



Octree root Octree internal nodes (coarser grids) Original cell (fine grid)

Isosurfacing in Higher Dimensions

- Marhcing Cubes like algorithm for hypercubes of any dimension
  - 4-dimensional isosurfaces (space + time)
  - 216 possible vertex labels
  - 222 basic cases (after the symmetry)

- Isosurfacing in R^d
  - 2^{2^d} possible cases
  - Locate the d-cubes which are intersected by the isosurface

- 4D isosurfacing provides
  - Smooth animation
  - Slicing through oblique hyper-planes to study time-evolving features