EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Indirect Volume Visualization

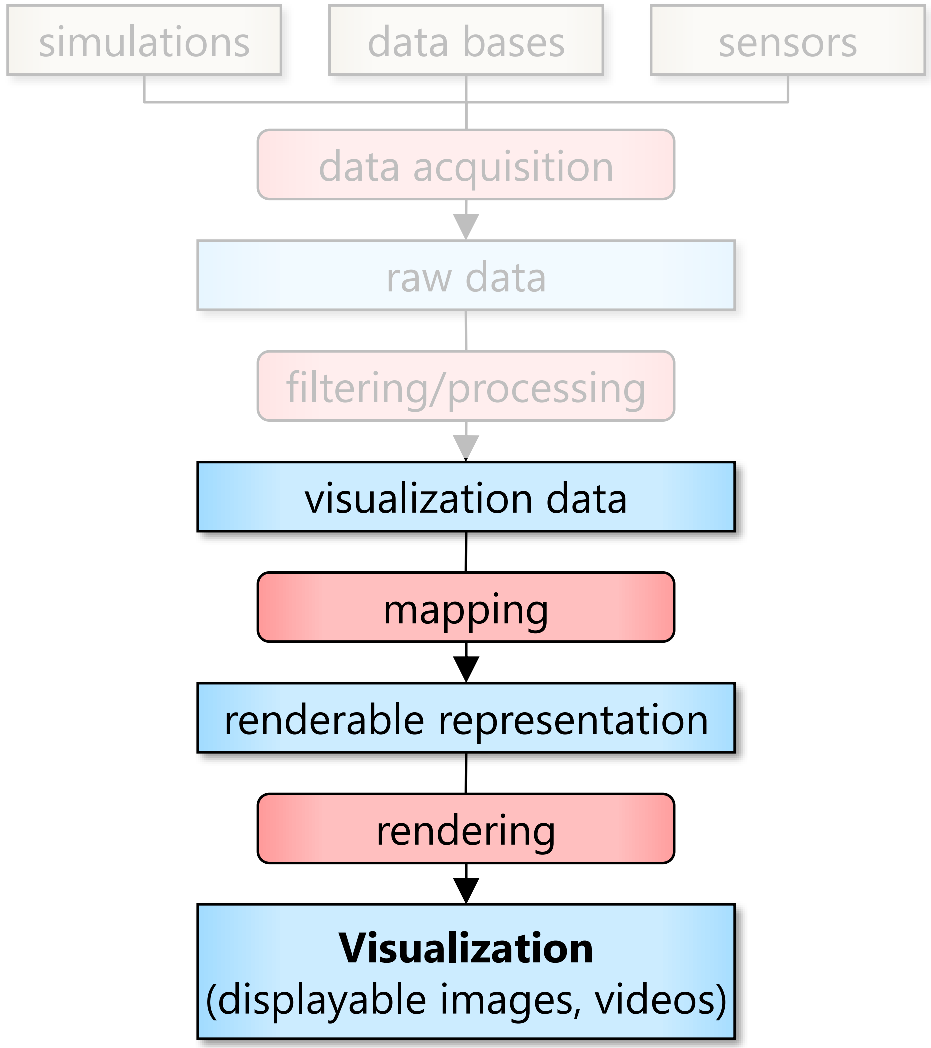
Scientific Visualization – Summer Semester 2021

Jun.-Prof. Dr. **Michael Krone**

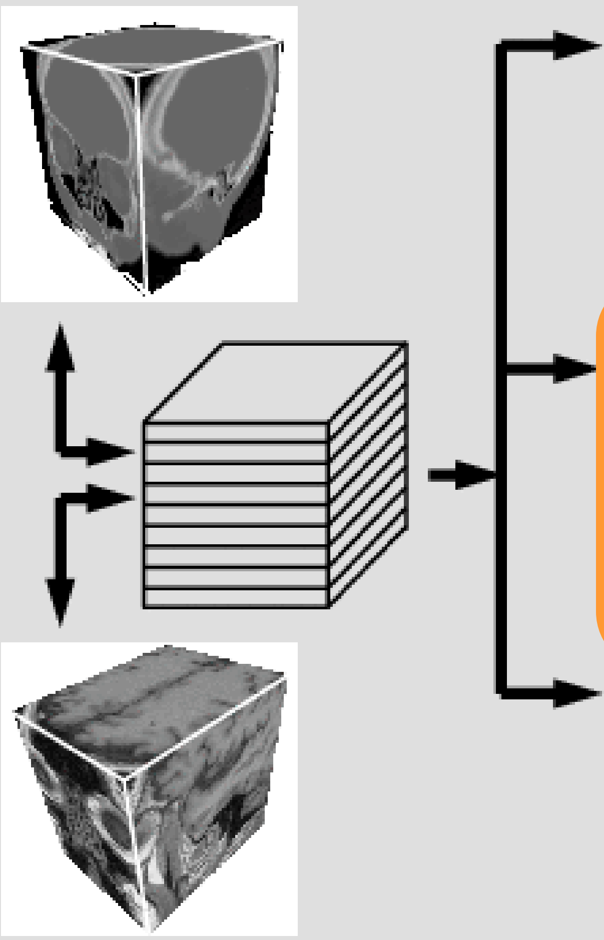
Contents

- Contour tracing
- Cuberille approach
- Marching cubes
- Marching tetrahedra
- Dividing cubes
- Optimization (Discretized MC, octree-based, range query)

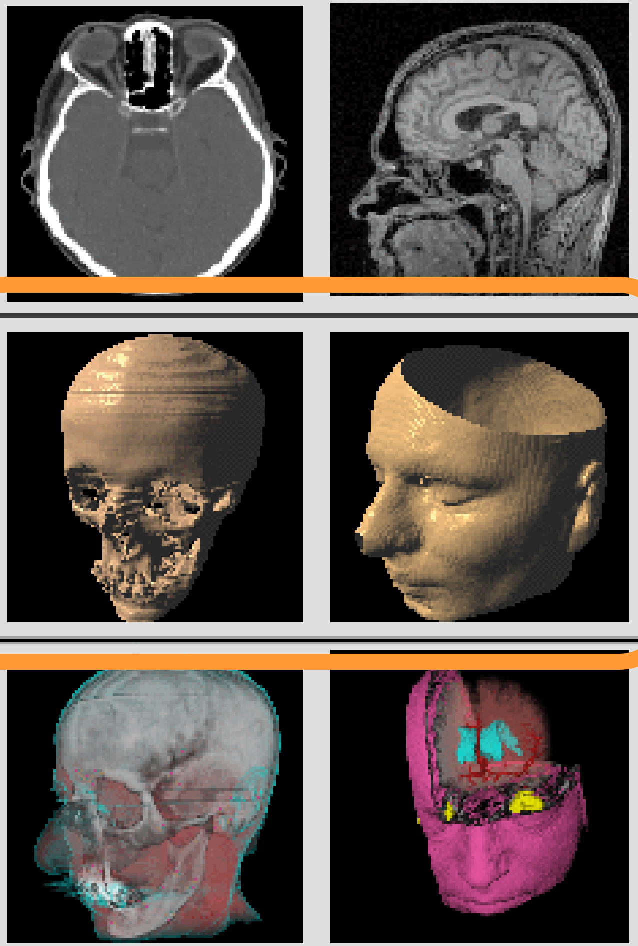
Focus:
Second step of visualization pipeline



Overview – Volume Visualization



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



Isosurface Rendering

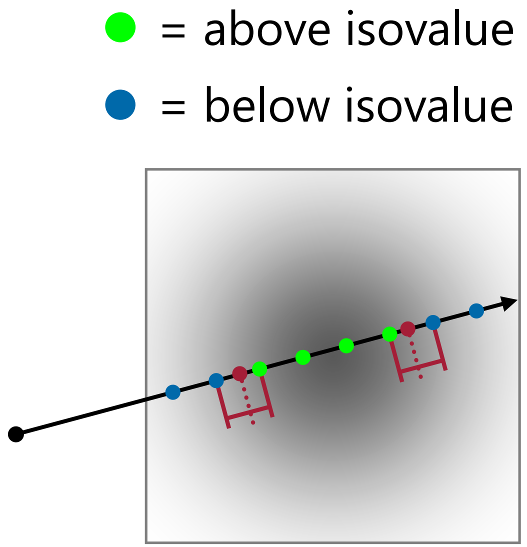
- Via direct volume rendering / ray casting
  - March along ray (front-to-back) until and check the value at each sample point and compare it to the value of the previous sample point

```
if (value < isovalue): do nothing
if (value ≥ isovalue):
    Check if (previous value < isovalue): draw surface
```

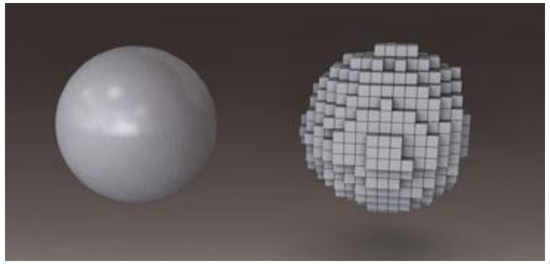
→Same procedure the other way round (i.e., leaving the isosurface)

→Linearly interpolate exact location of isosurface between samples

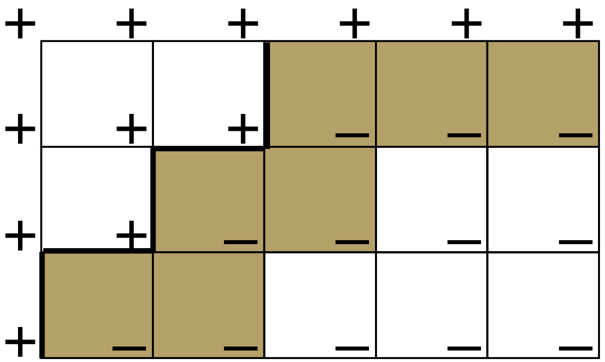
  - Surface normal at isosurface via central differences
  - Pros:
    - High quality (if step size is sufficiently small)
  - Cons:
    - View-dependent, costly per-pixel ray marching, no surface mesh for further computations

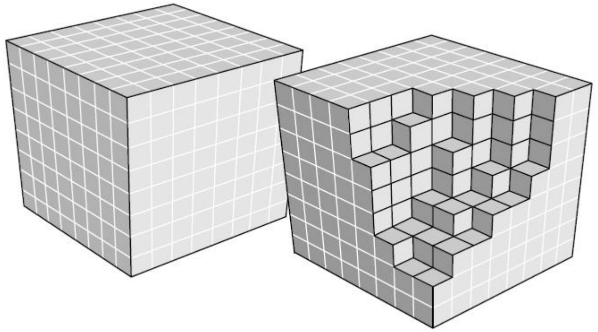


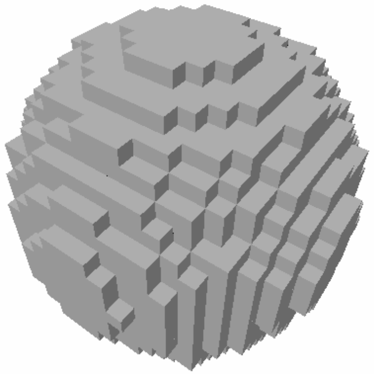
Cuberille Approach



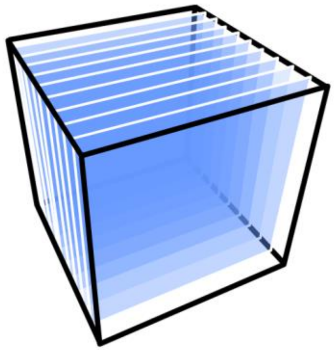
- Cuberille (opaque cubes) approach [Herman 1979]
  - Binarization of the volume with respect to the isovalue
  - Find all boundary front-faces
    - All faces where the normal points towards the viewpoint ($N \bullet V > 0$) and where normal points outwards the cell
  - Render these faces as shaded polygons (cubes)
- "Voxel" point of view: **NO** interpolation within cells





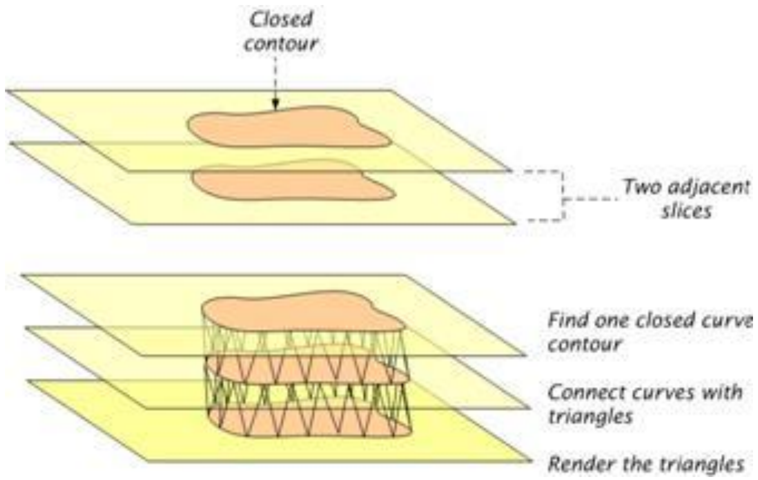


Contour Tracing



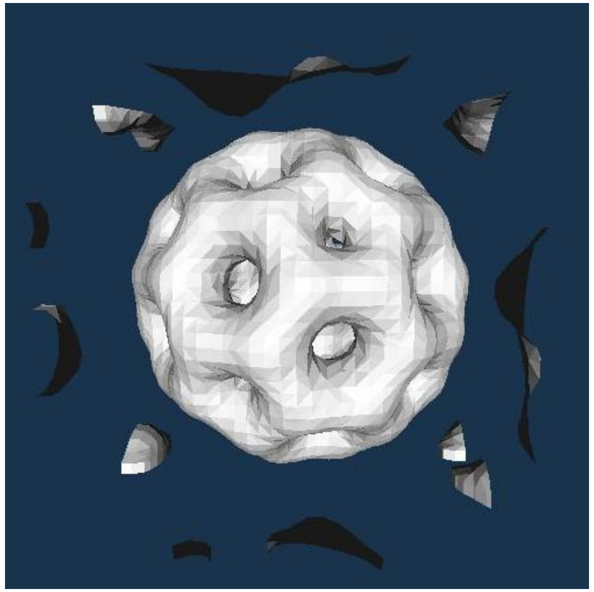
- Find isosurfaces from 2D contours
  - *Segmentation:* find closed contours in 2D slices (polylines)
  - *Labeling:* identify different structures (isovalue of higher-order characteristics)
  - *Tracing:* connect contours of the same object from adjacent slices via triangles
  - *Rendering:* display triangles
- Choose topological or geometrical reconstruction
- Problems:
  - Sometimes there are many contours in each slice or there is a high variation between slices

    → Tracing (assignment) becomes very difficult

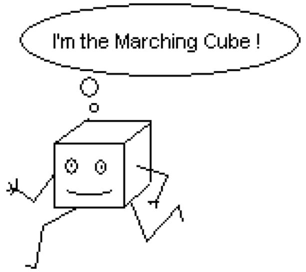


Marching Cubes

- Better approximation of the "real" isosurface by the **Marching-Cubes** (MC) algorithm [Lorensen, Cline 1987]
  - 3D analog to Marching Squares
  - Works on the original data
  - Approximates the surface by a triangle mesh
  - Surface is found by linear interpolation along cell edges
  - Uses gradients (e.g., central diff.) as the normal vectors of the isosurface
  - Efficient computation by means of lookup tables
- ***The*** standard geometry-based isosurface extraction algorithm
  → *more than 17,000 citations!*



Marching Cubes

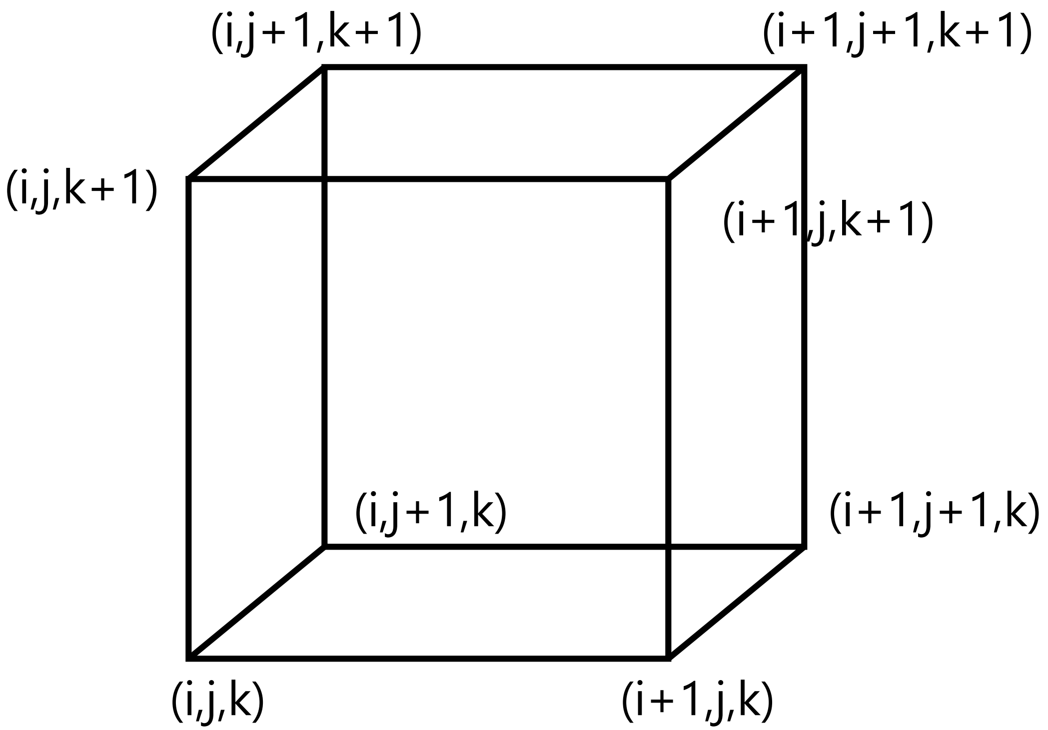


- The core MC algorithm
  - Cell consists of 8 node values: (i+[0,1], j+[0,1], k+[0,1]) → cube

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 (optional)
7. Consider ambiguous cases (optional)
8. 8. Go to next cell

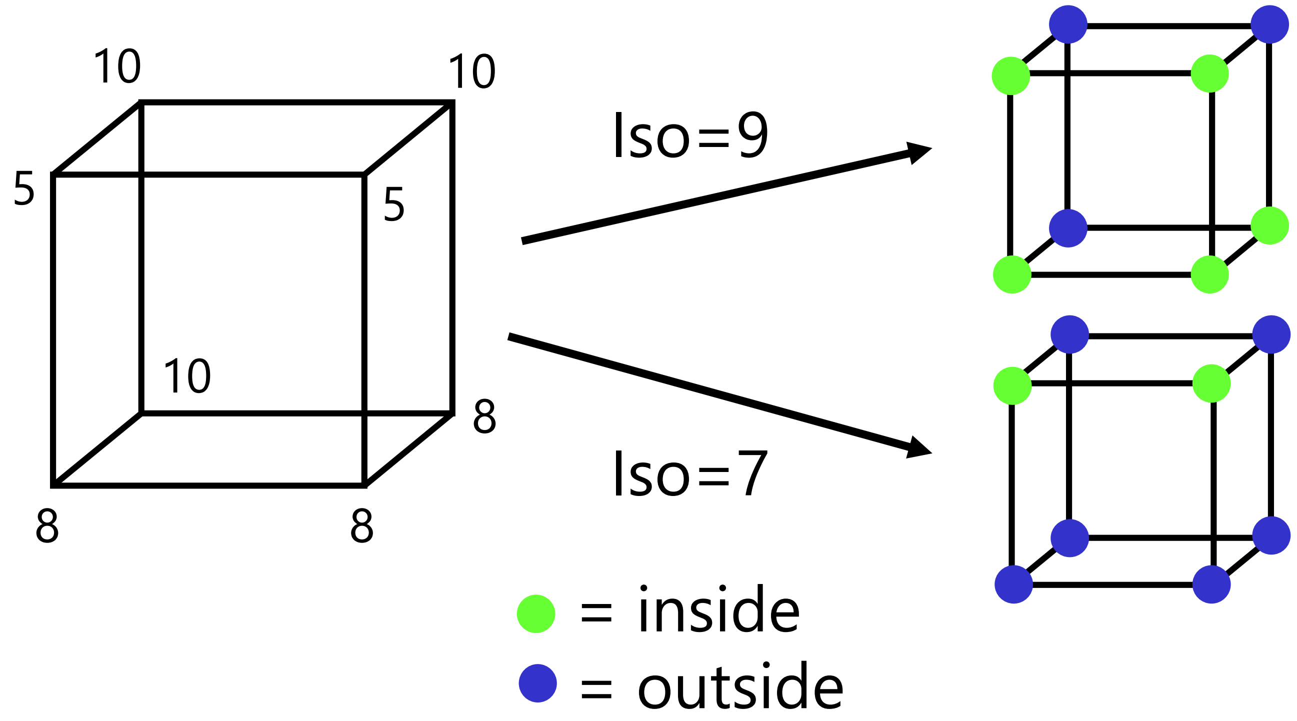
Marching Cubes

- Step 1: Consider a hexahedral cell with eight node values



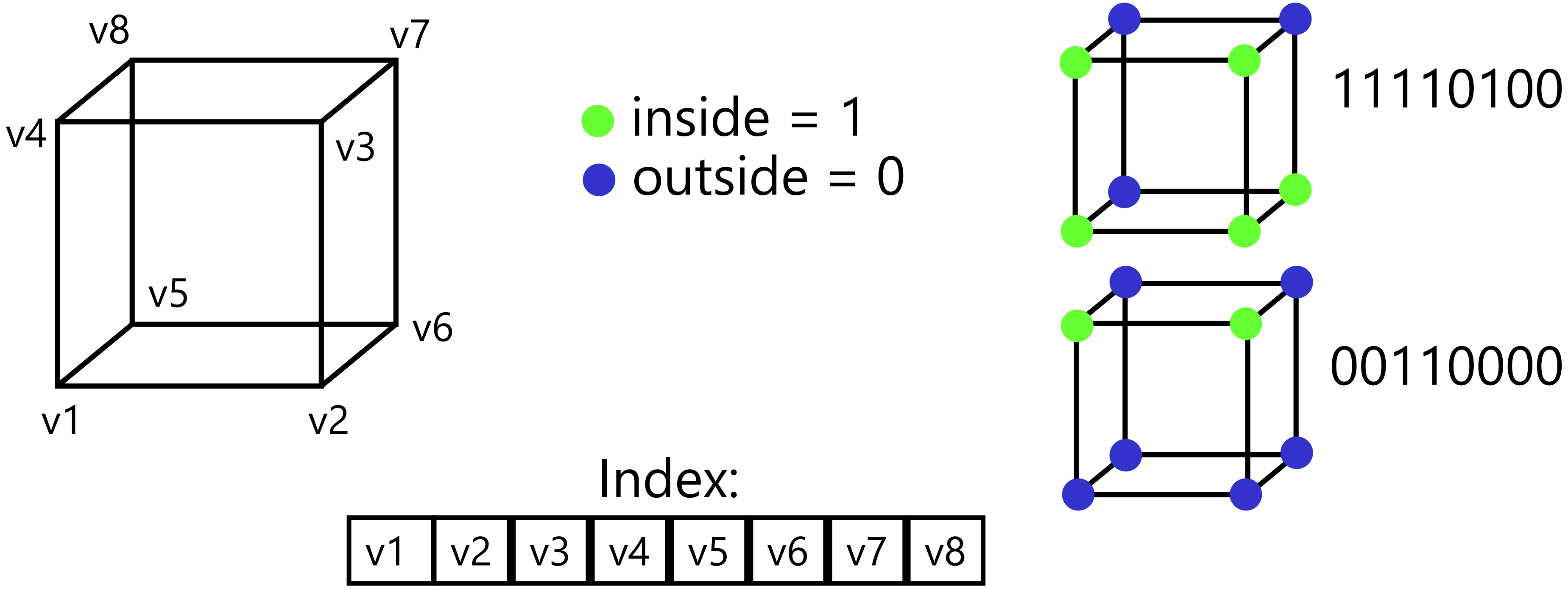
Marching Cubes

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



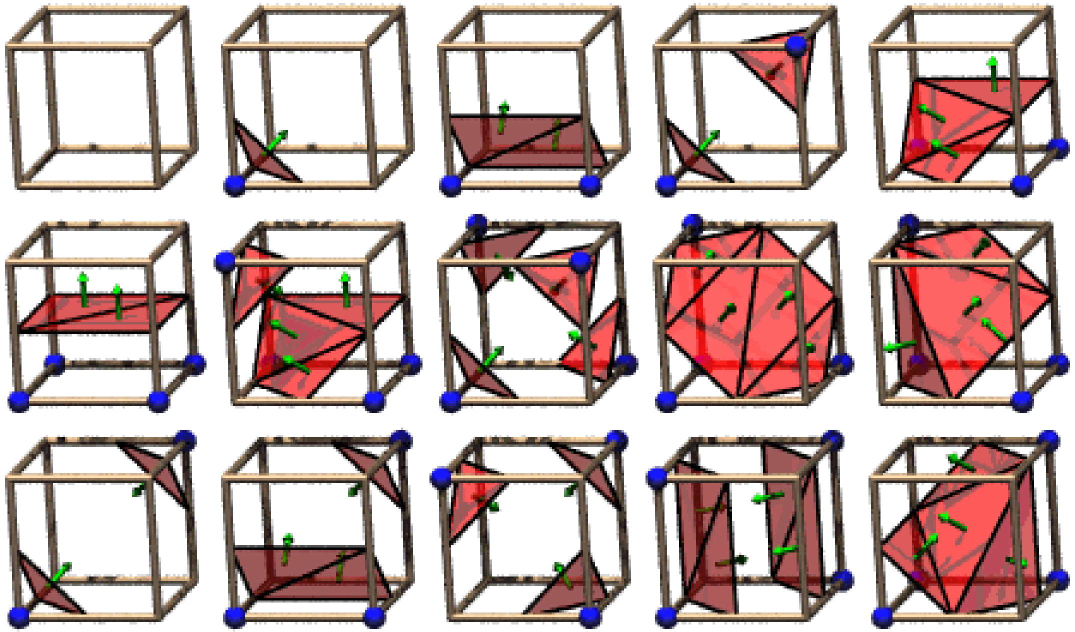
Marching Cubes

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



Marching Cubes

- 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
  - Lookup table (LUT) stores all 256 cases
  - Each case creates at most 5 triangles (dual cases for inverted signs)



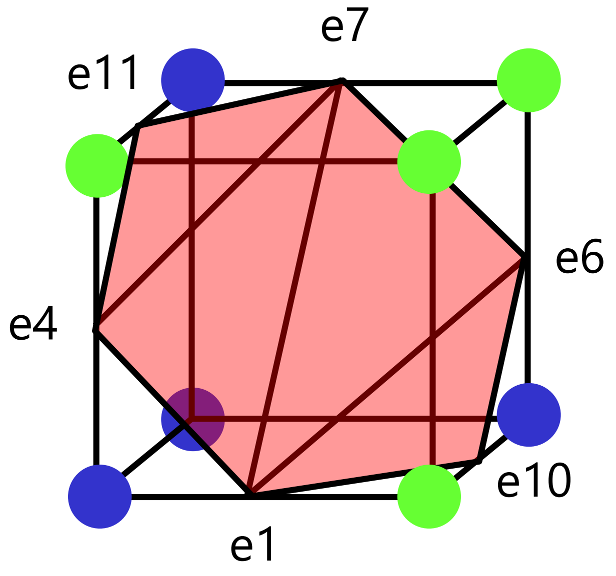
The 15 Cube Combinations

Marching Cubes

- Step 4 *cont.*: Get triangle list from table
  - Example for index = 10110001

    triangle count = 4

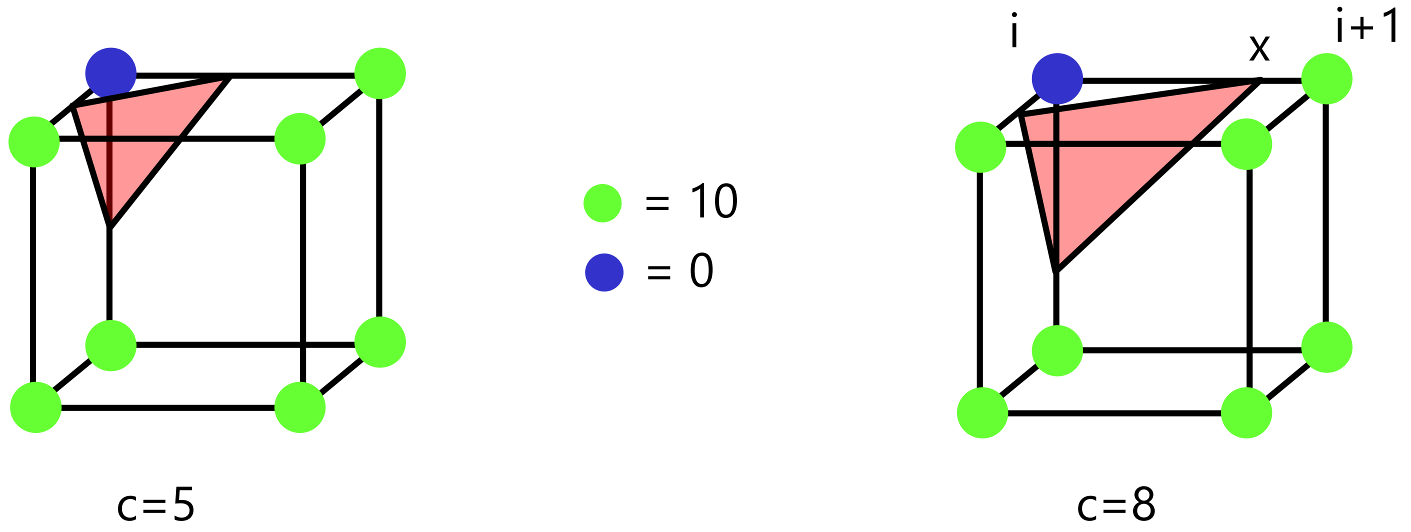
    triangle 1 = e4,e7,e11
    triangle 2 = e1, e7, e4
    triangle 3 = e1, e6, e7
    triangle 4 = e1, e10, e6



  - Face normals encoded implicitly by order of vertices
  - Normal points to higher (or lower) values of the field -> inside/outside

Marching Cubes

- Step 5: For each triangle edge, find the vertex locations along the cell edges using linear interpolation of the node values

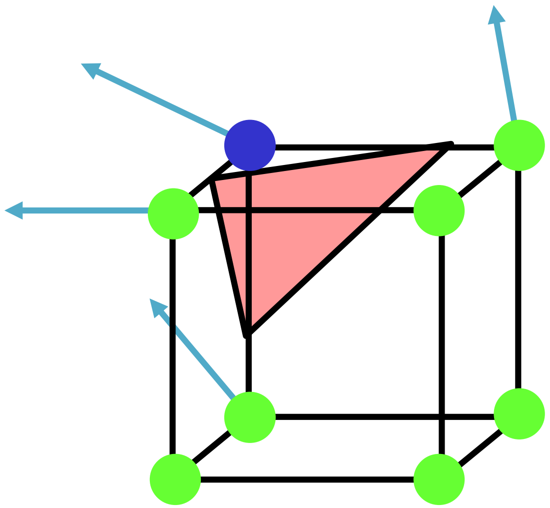


$$x = i + (c - v[i]) / (v[i+1] - v[i])$$ if all edge lengths = 1

$$x = [(v[i+1] - c)x[i] + (c - v[i])x[i+1]] / (v[i+1] - v[i])$$ otherwise

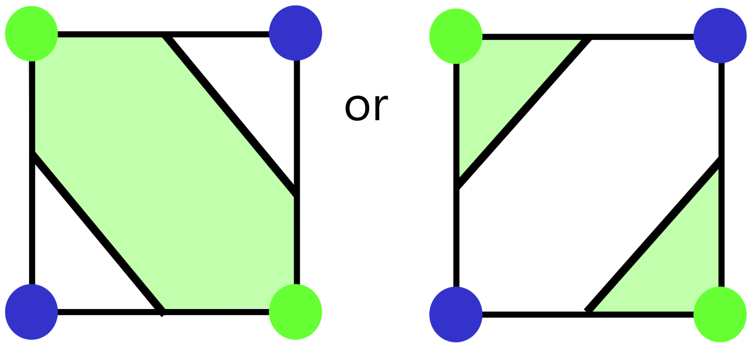
Marching Cubes

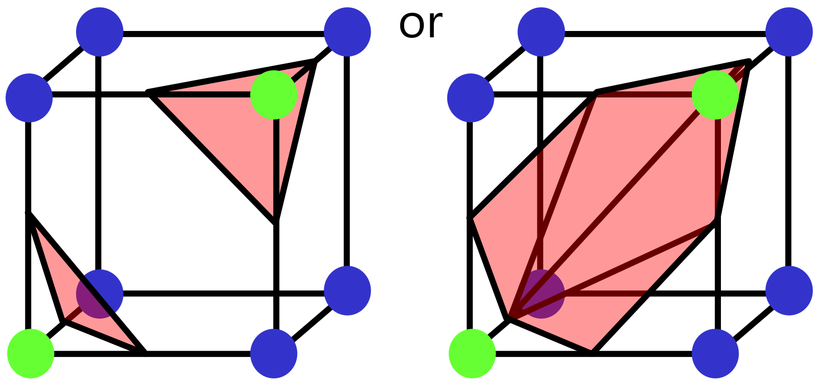
- Step 6: Calculate the normal at each node (central differences)
  - $G_x = V_{x+1,y,z} - V_{x-1,y,z}$
    $G_y = V_{x,y+1,z} - V_{x,y-1,z}$
    $G_z = V_{x,y,z+1} - V_{x,y,z-1}$
  - Use linear interpolation and normalization to compute the normal at the vertex
  - ***Alternative*** *(OpenGL/WebGL)*: compute central differences at vertex position (3D texture lookup)



Marching Cubes

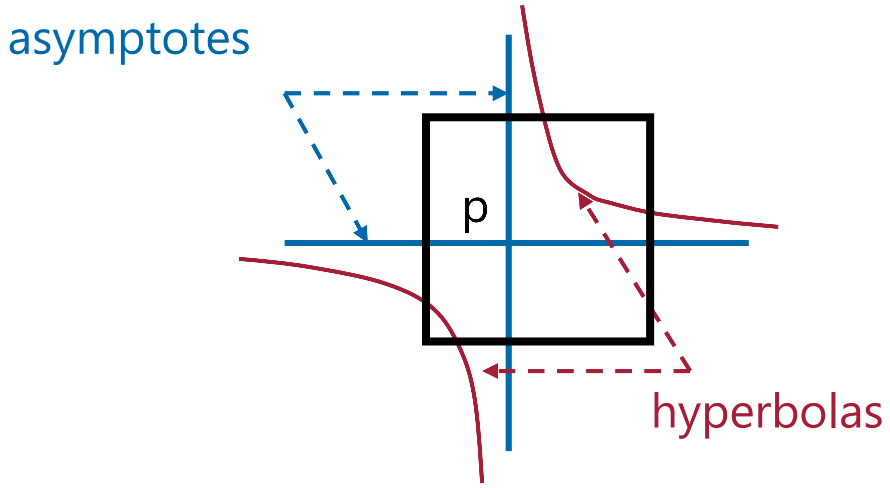
- Step 7: Consider ambiguous cases
  - Ambiguous cases: 3, 6, 7, 10, 12, 13
  - Adjacent vertices: different states
  - Diagonal vertices: same state
  - Resolution: choose one case (the right one!)





Marching Cubes

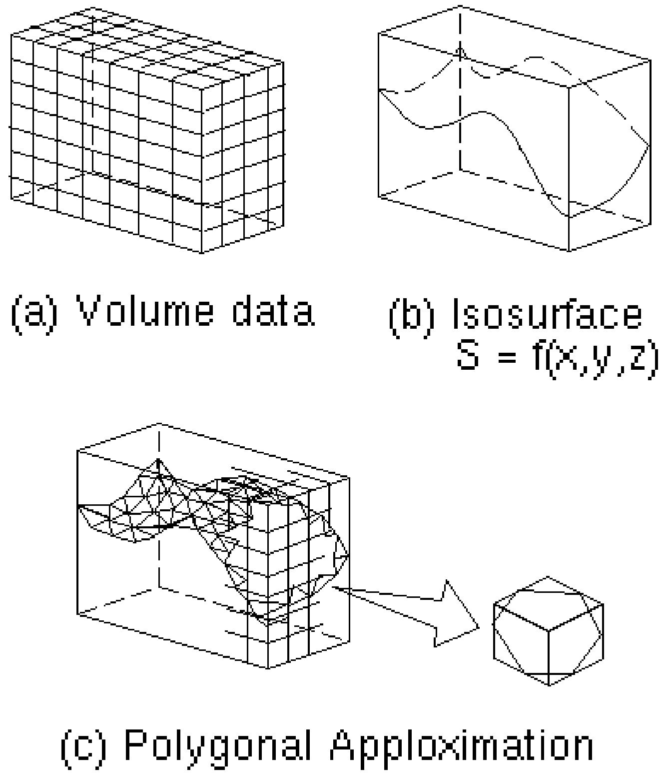
- Step 7 *cont.*: Consider ambiguous cases
- Asymptotic Decider [Nielson, Hamann 1991]
  - Assume bilinear interpolation within a face
  - Hence face-intersection of isosurface is a hyperbola
  - Compute the point p where the asymptotes meet
  - Sign of $S(p)$ decides the connectivity
    (→ see Marching Squares)



Marching Cubes

- 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

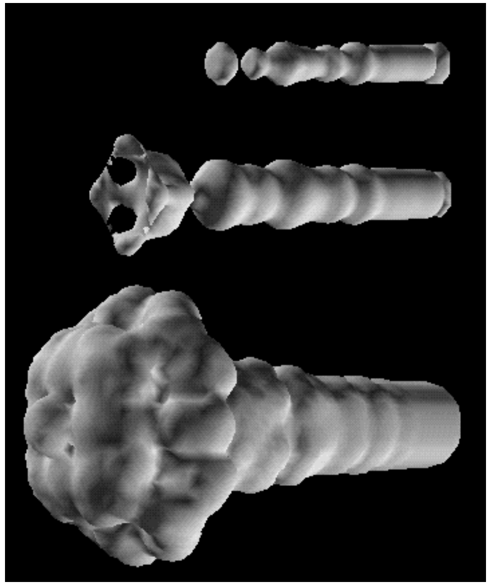
- Several isosurfaces
  - Run MC several times with different isovalues
  - Semi-transparent rendering in OpenGL/WebGL requires spatial sorting



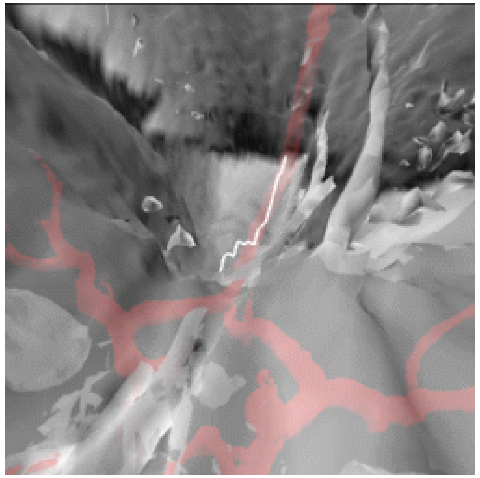
Marching Cubes

- Examples

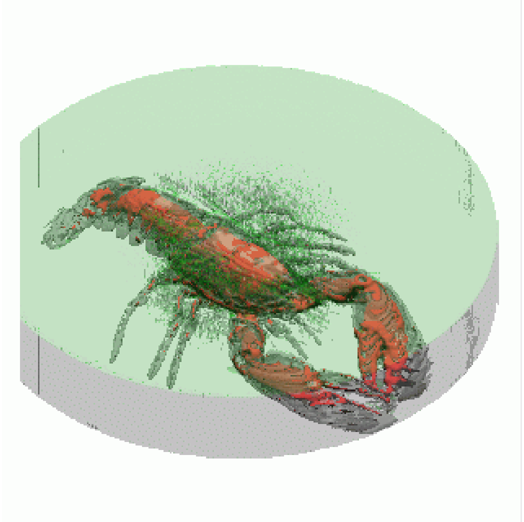
1 Isosurface



2 Isosurfaces

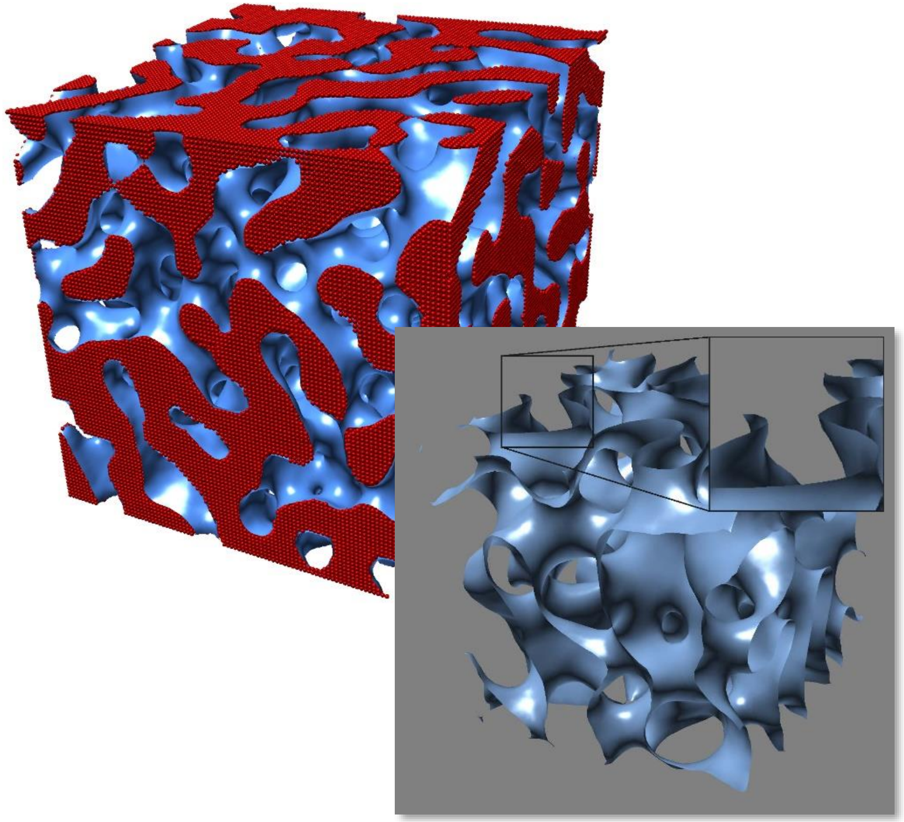


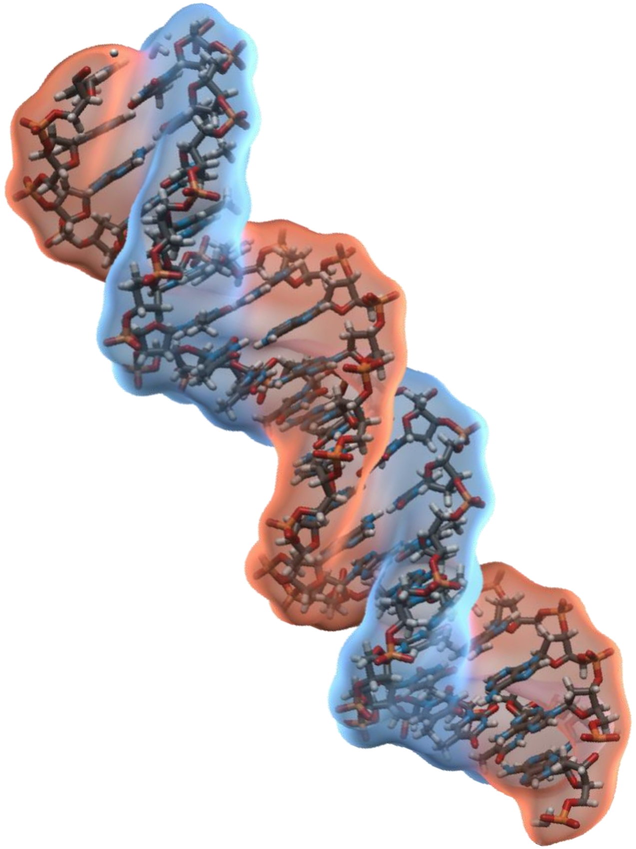
3 Isosurfaces

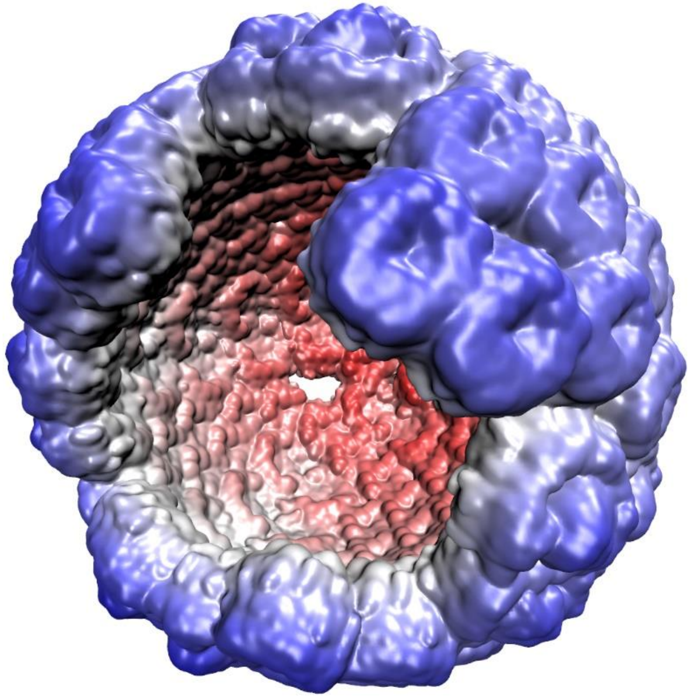


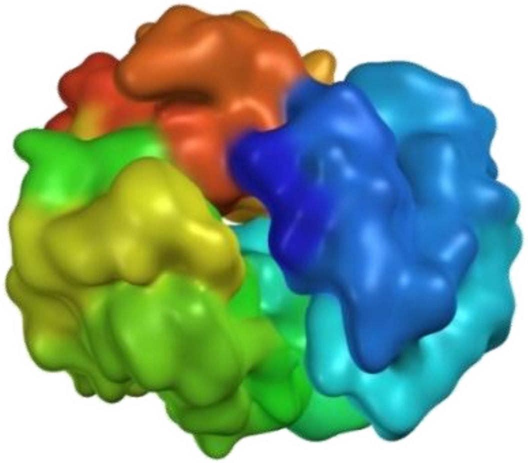
Marching Cubes

- Examples



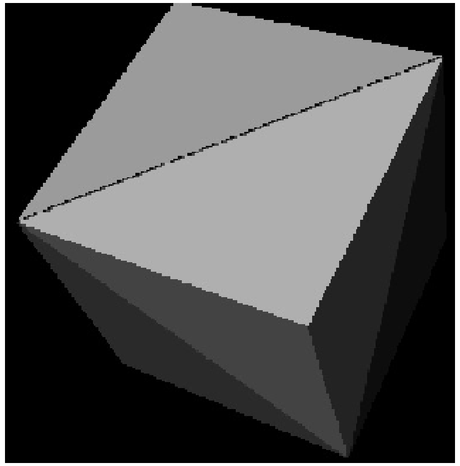


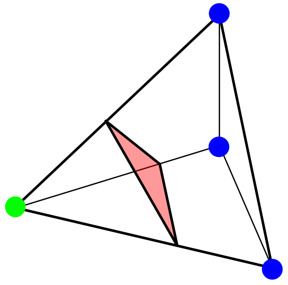


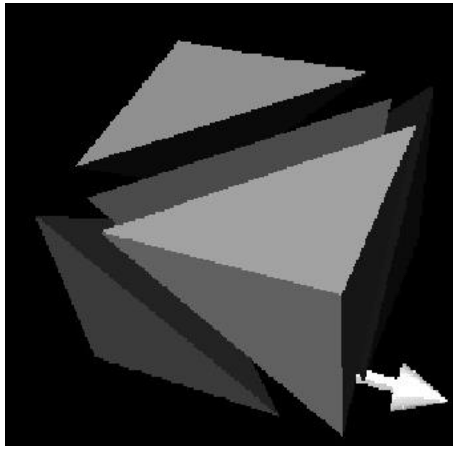


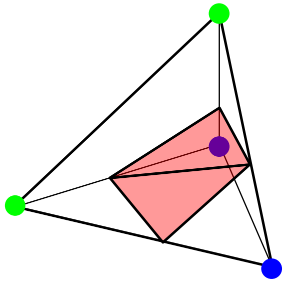
Marching Tetrahedra

- Primarily used for unstructured grids
  - May split other cell types into tetrahedra
    → introduces (usually small) error
- Process each tetrahedron similarly to MC algorithm
- Two different cases:
  - One (–) and three (+) (or vice versa)
    - The surface is defined by one triangle
  - Two (–) and two (+)
    - Sectional surface given by a quadrilateral
    - Split quad into two triangles using the shorter diagonal



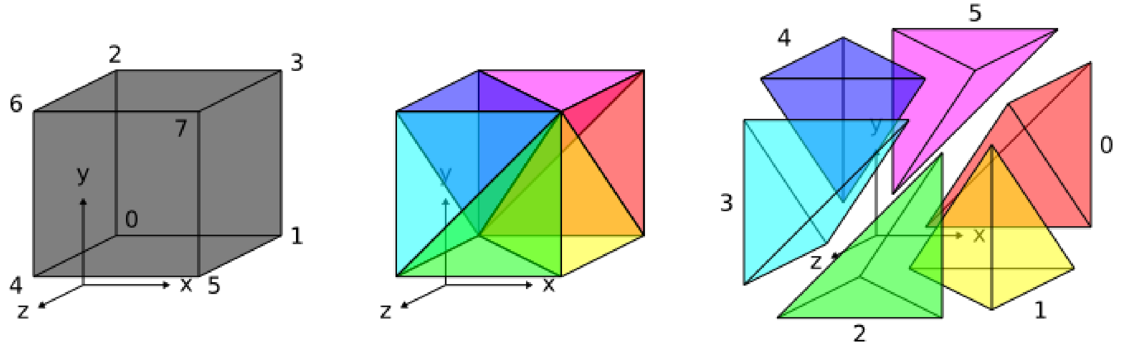






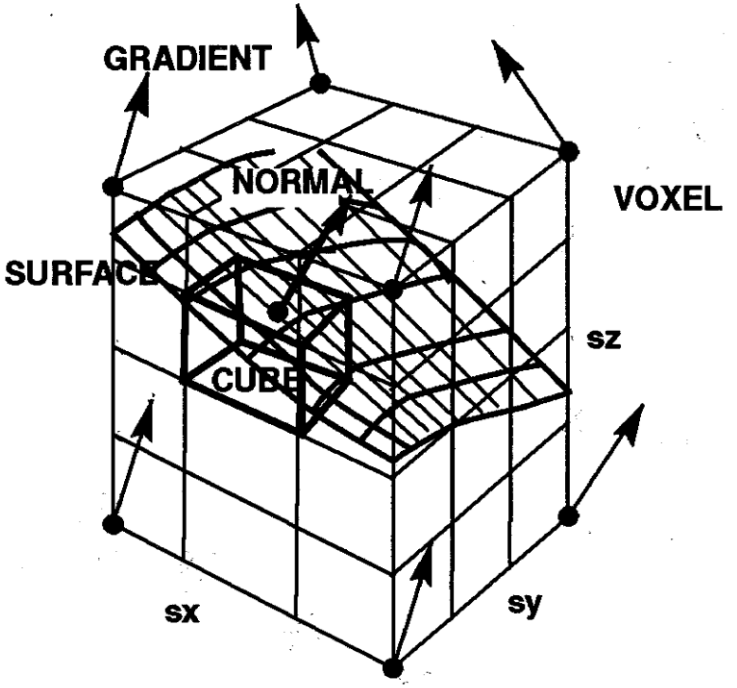
Marching Tetrahedra

- Properties
  - Fewer cases: 3 instead of 15
    - Linear interpolation within tetrahedra
    - No problems with consistency between adjacent cells
    - Vertex gradient via weighted cell gradients (constant)
  - Number of generated triangles might increase considerably compared to the MC algorithm due to splitting into tetrahedra
  - But, several improvements exist:
    - Hierarchical surface reconstruction
    - View-dependent surface reconstruction
    - Mesh decimation



Dividing Cubes [Cline, Lorensen 1988]

- Uniform grids
- Basic idea
  - Create “surface points" instead of triangles
  - Associate surface normal with each surface element
  - Surface points (when shaded and rendered) are pixel-sized
  - Subdivide cells as necessary
  - View-dependent technique



Dividing Cubes

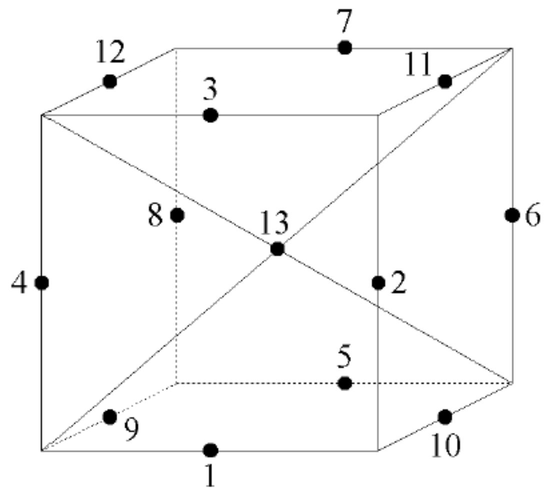
- Algorithm:
  - Choose a cube
  - Classify, whether an isosurface is passing through it or not
  - if (surface is passing through)
    - Recursively subdivide cube until pixel size
  - Compute normal vectors at each node (e.g., via central differences)
  - Render shaded points (cube centers) with interpolated normal

Dividing Cubes

- Properties
  - View-dependent load balancing
  - Better surface approximation due to interpolation within cells
  - Only good for rendering → no triangle mesh computation
    - Since no surface representation is generated, it does not allow for further computations on the surface
  - Eliminates scan-conversion step (generation of pixels from triangles)
  - Point-cloud rendering of randomly ordered points
  - No topology

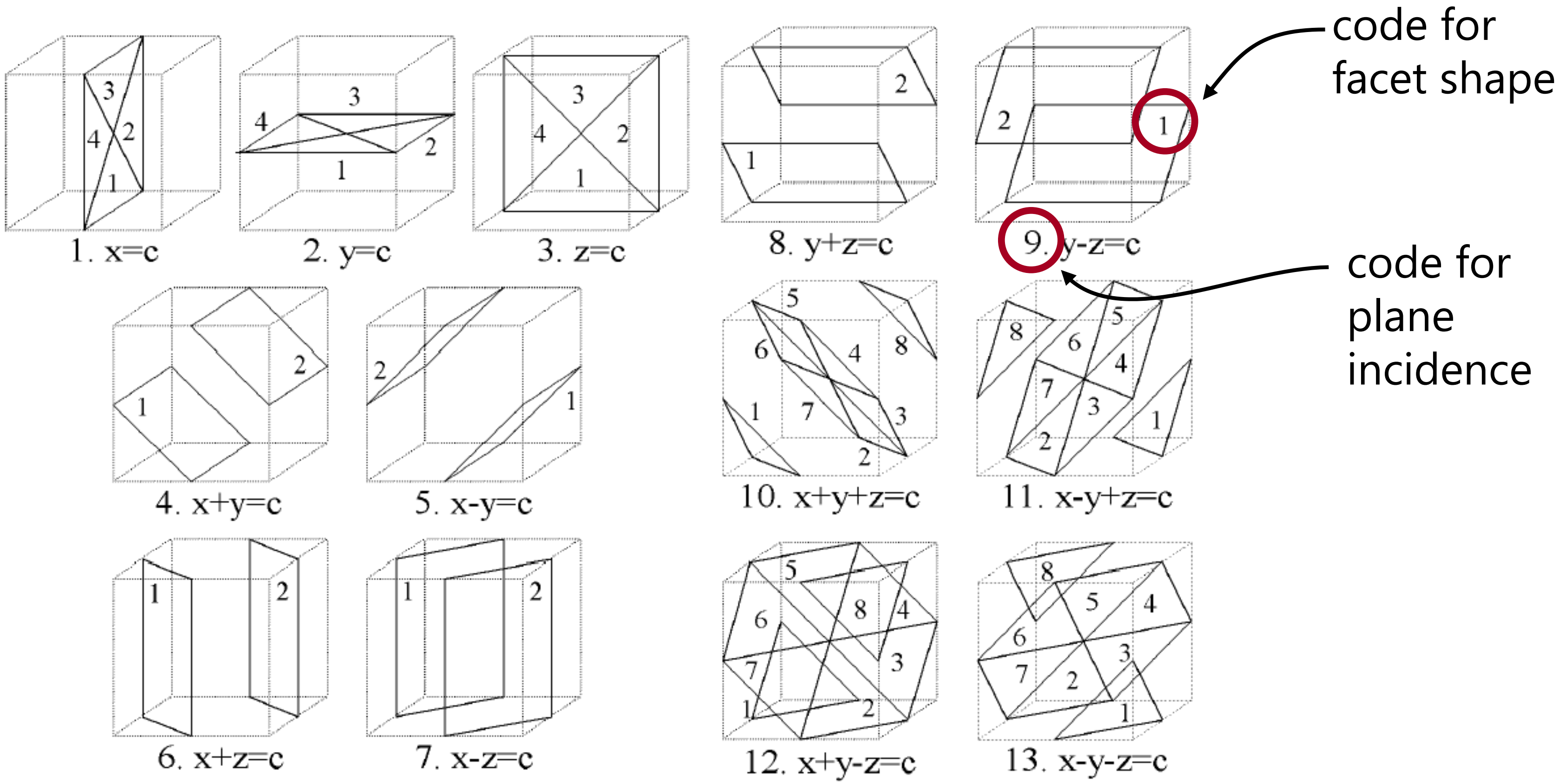
Discretized Marching Cubes

- Discretized Marching Cubes (DiscMC) [Montani et al. 1994]
- Accelerate standard MC
- Mixture in-between:
  - Cuberille approach (constant scalar value for each voxel)
  - Marching Cubes (trilinear interpolation in cells)
- Approximation of MC: discrete positions for vertices of isosurface
  - 13 different vertex positions
  - 12 edge-midpoints + 1 centroid



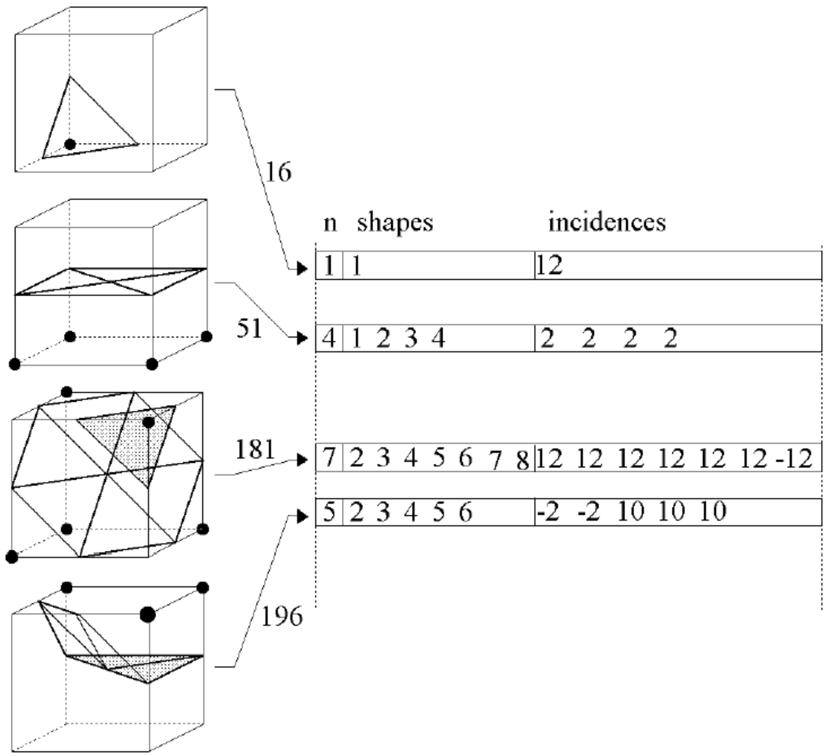
Discretized Marching Cubes

- Finite set of planes on which faces can lie



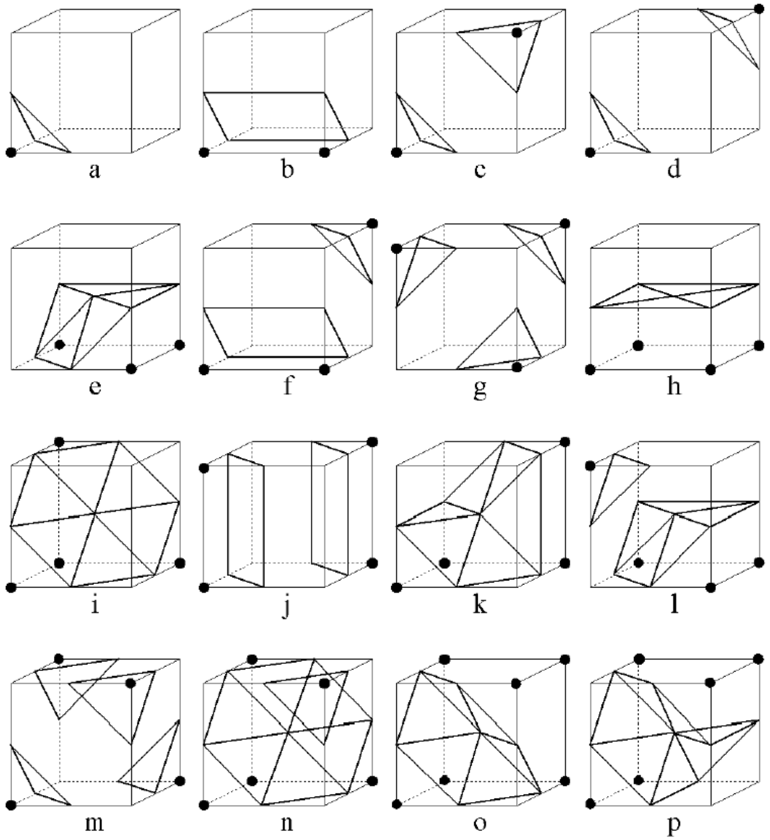
Discretized Marching Cubes

- Classification of a facet by
  - Plane incidence
    and
  - shape
- Sign of incidence determines orientation of facet
- Classification of isosurface fragment (facet set)
  - Indices to incidences and shapes



Discretized Marching Cubes

- Lookup table
  - Based on MC LUT
  - Simple reorganization
  - Indices as above
- Vertex positions of facet determined by vertex configuration of cell
- No linear interpolation needed

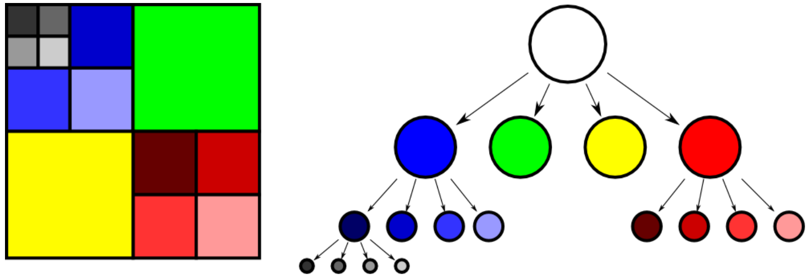


Discretized Marching Cubes

- Algorithm:
  - Analogously to MC: traversing the grid
  - Normal vectors based on gradients (same as MC)
  - *Optional postprocessing*: merge facets and edges (i.e., adjacent planar triangles)
- Advantages:
  - Simple classification of facet sets
  - Many coplanar facets due to small number of plane incidences
    → significantly reduces number of triangles after merge
  - No interpolation needed, i.e., only integer arithmetic
  - Still quite good visual results
    (in particular after shading due to gradient-based normals)

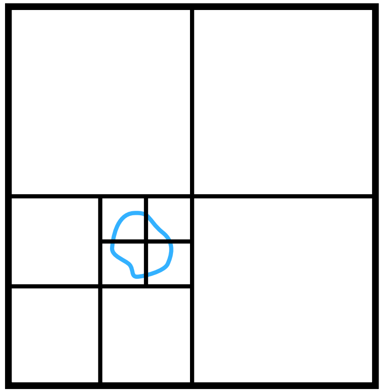
Octree-based Isosurface Extraction

- Acceleration of MC (and similar methods)
- Domain search – space query



Quadtree

- Octree-based approach [Wilhelms, van Gelder 1992]
  - Spatial hierarchy on grid (tree)
  - Store minimum and maximum scalar values for all children with each node
  - While traversing the octree, skip parts of the tree that cannot contain the specified isovalue (space query)



Octree-based Isosurface Extraction

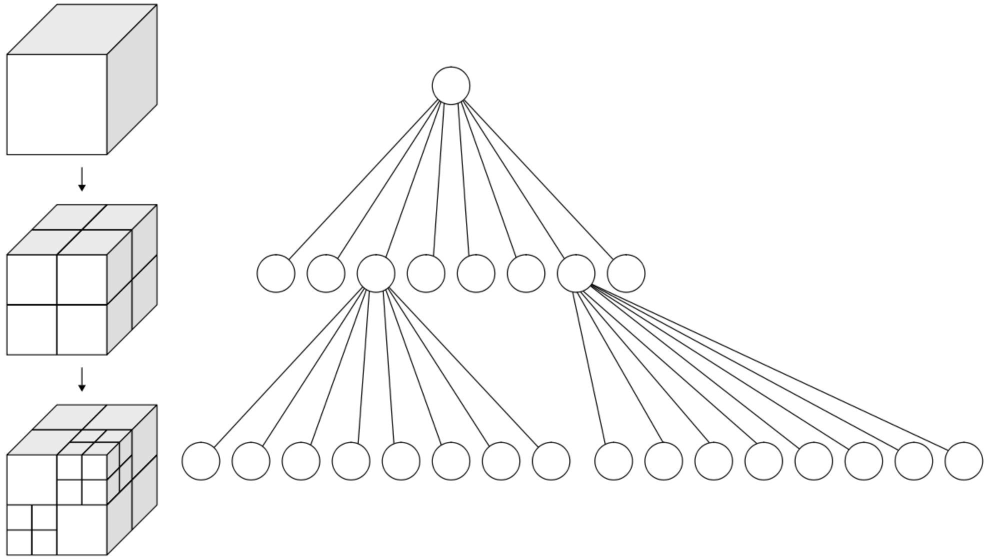
- What data structure for octree?
- Advantages of full octree:
  - Simple array-like structure and organization
  - No pointers needed

- Number of nodes in full octree:

$$n_{\text{nodes}} = \sum_{i=0}^{\lceil \log_2 s \rceil - 1} 8^i = \frac{8^{\lceil \log_2 s \rceil} - 1}{8-1} \approx \frac{s^3 - 1}{7} \approx 0.14\, n_{\text{data points}}$$

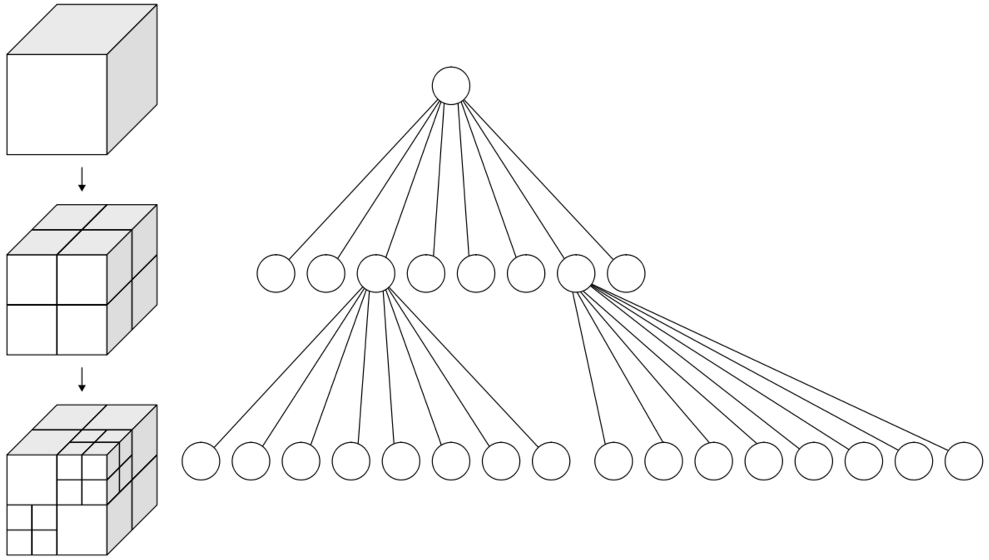
s: resolution in each dimension

→ optimal ratio is $n_{\text{nodes}} / n_{\text{data points}} \approx 0.14$



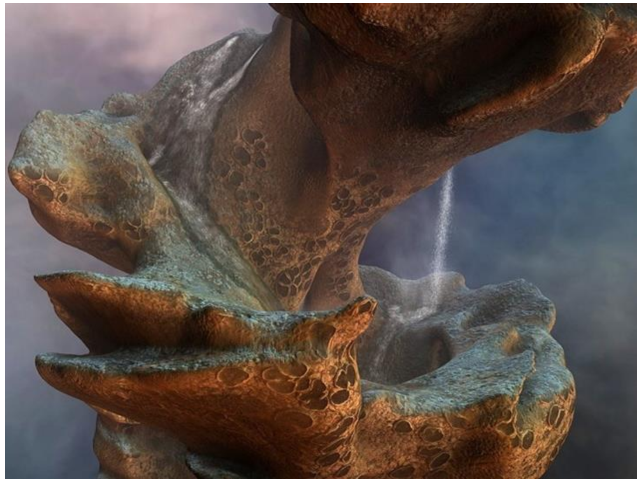
Octree-based Isosurface Extraction

- Problem with memory consumption of complete octree:
  - Ideal: grid size of $2^n \times 2^n \times 2^n$
  - Usually different resolutions that are not powers of two
- **Example:**
  - Data set: $320 \times 320 \times 40$
  - 4M data points (4,096,000 voxels)
  - Full octree:
    $1 + 2^3 + 4^3 + \ldots + 256^3 = 20\text{M}$ nodes
  - 2 values per element:
    minimum and maximum values



GPU Implementation of Marching Cubes

- Geometry Shader
  - GPU shader stage between vertex and fragment shader than can create new triangles → *not (yet) available in WebGL*
  - Input: primitives (points, trianges,…)
  - For each cube (2×2×2 voxel): geometry shader reads 8 voxel values at corners, classifies them and generates the MC triangles (0-5) according to LUT
- **Example:** Nvidia Demo "Cascades" (2007)
  - Procedurally generated 3D terrain rendered via Geometry Shader Marching Cubes
  - Showcase for Geforce GTX 8000 series



https://developer.nvidia.com/gpugems/gpugems3/part-i-geometry/chapter-1-generating-complex-procedural-terrains-using-gpu