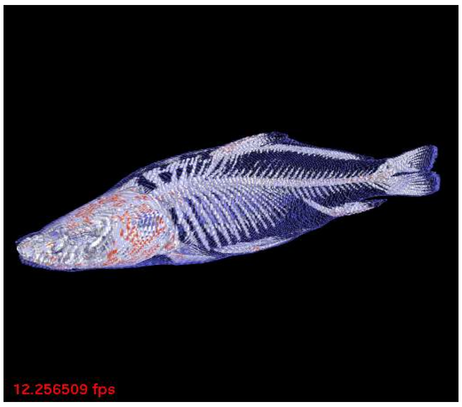
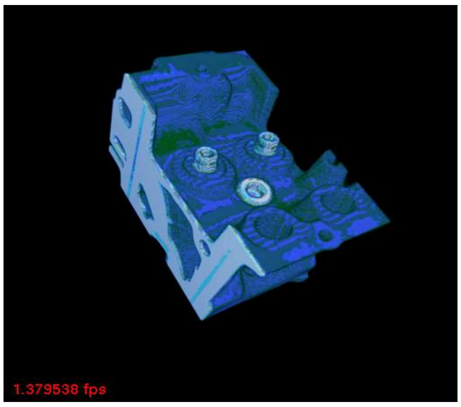
Volume Visualization

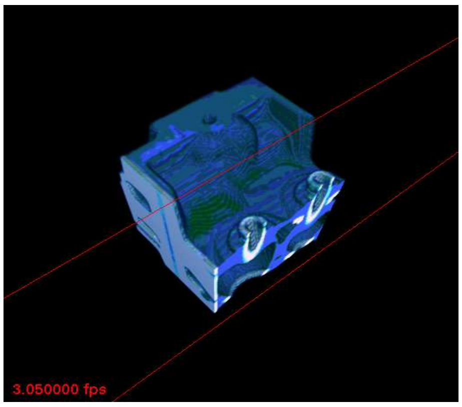
Volume visualization is used to create two-dimensional graphical representations from scalar datasets that are defined on three-dimensional grids. Examples of 3D data range from medical applications like CT, MRI scans, confocal microscopy over ultrasound and seismic data to fluid dynamics. There are two fundamental types of volume visualization: direct volume rendering (DVR) algorithms and indirect volume rendering- or surface-fitting (SF) algorithms.

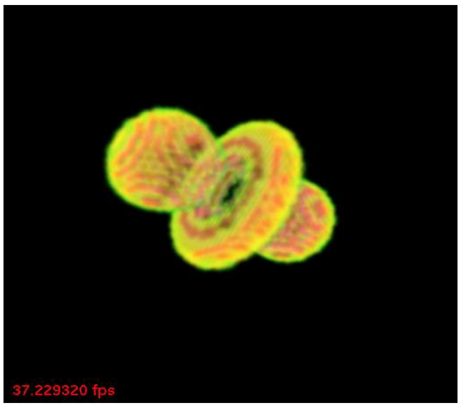
0.352437 fps

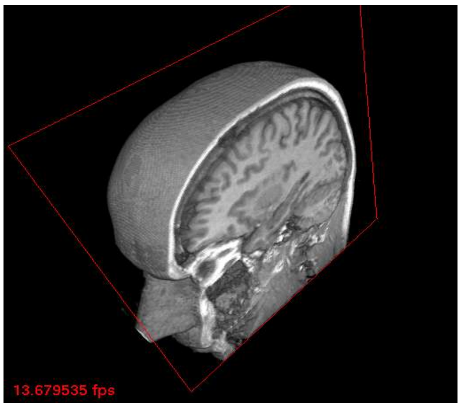


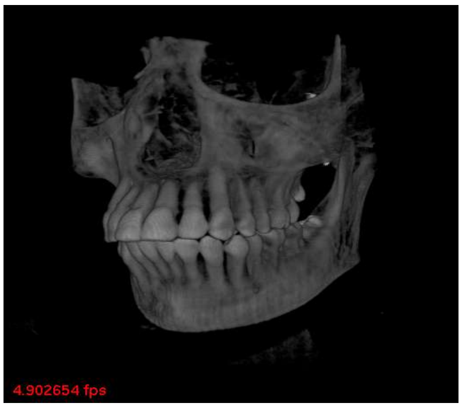
7.465051 fps

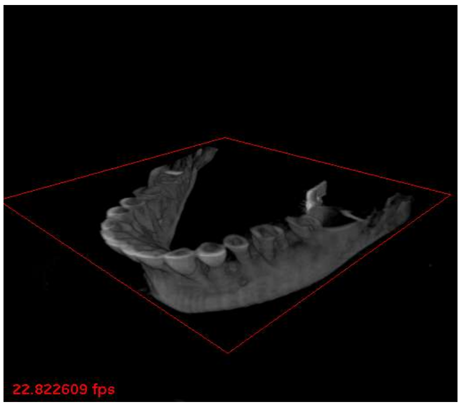












Standard surface modeling only defines the opaque outer surface of an object, so you can not see inside of it. The basic idea of volume visualization is to make the boundaries of an object transparent, so that one can see inside. Volume data consist of two different typical characteristics, which should be considered. They contain the essential interior information of an object, but geometric representation of fire, clouds or gaseous phenomena can not be described. So you have to distinguish between shape (given by the geometry of the grid) and appearance (given by the scalar values or color, texture, lighting conditions, etc.). Even if the data could be described geometrically, in general there are too many primitives to be represented.

In general, volume rendering can be classified in two groups, the direct and indirect techniques. Further on there exist techniques for 2D scalar fields, or techniques which reduce or convert volume data to an intermediate representation (surface representation), which can be rendered with traditional techniques. Another possibility is to consider the data as a semi-transparent gel with physical properties and directly get a 3D representation of it.

Slicing:

Slicing techniques are common methods for visualization, and are used to examine scalar fields. They display the volume data mapped to colors on a 2 dimensional slice plane.

Isosurfacing:

Isosurfaces are 2D surfaces that can be extracted from 3D (or higher dimensional) sample volumes. They generate opaque/semi-opaque surfaces.

One problem is that all voxels from the same isosurface have the same color, which only leads to a flat surface with a single color. A reasonable 3D effect originates only if one computes the lighting of the surface on the basis of the normal vectors.

Transparency effects:

Volume material attenuates reflected or emitted light.

Indirect volume rendering techniques:

The strategy of indirect volume rendering is to generate a surface model of the given volume data which is efficiently manageable and representable. In general the surface is opaque.

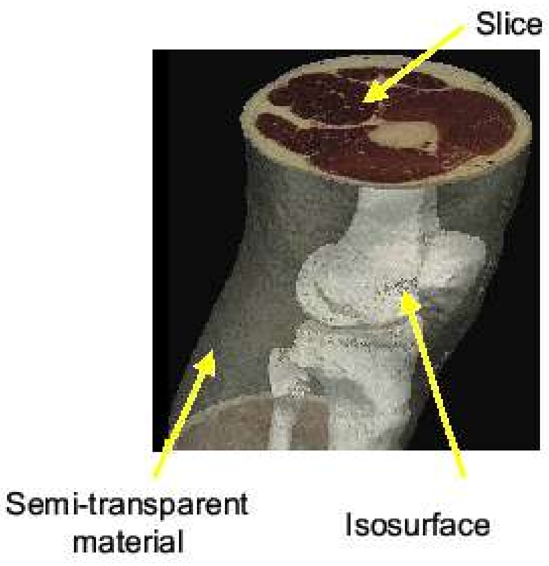
It is assumed that coherent structures (e.g. skin, bone) are represented by point sets with the same sampling rate. The surfaces of these pointsets are approximated by polygons. This often results in complex representations, where pre-processing of the surface representation might help. Even graphics hardware is used for interactive display.

In practice one starts with the volume data and tries to find a triangle mesh, that represents the volume as well as possible. Once the 2D mesh is found, it can be rendered with traditional techniques.

Direct volume rendering techniques:

The characteristic of direct volume rendering is the direct mapping of voxels on pixels of a 2D image plane. It allows for the "global" representation integrating physical characteristics, but prohibits interactive display due to its numerical complexity in general. Nowadays (2003) it is possible to realize interactive direct volume rendering on standard graphic hardware for a volume with approximately 256 cubes with up to 5 fps.

In practice the data can be considered as a semi-transparent gel and the user decides which parts of the object should be opaque or transparent. The final 2D image is computed by projecting, in visibility order, the voxels onto the image plane, and incrementally compositing the voxel's color and opacity into the final pixel.

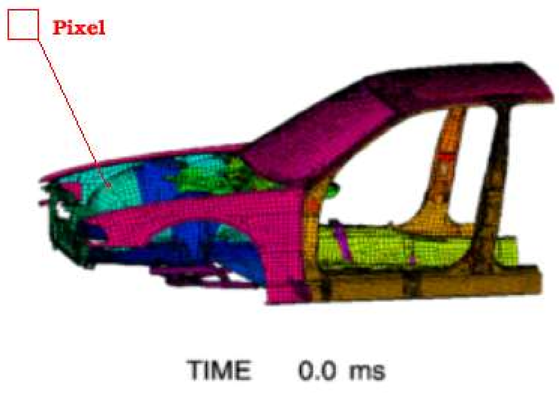


Opaque slice, opaque isosurface and semi-transparent tissue.

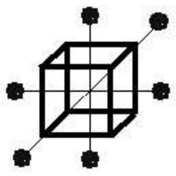
The goal of volume rendering is, to integrate all different techniques in order to represent the data as “good” as possible.
But you have to keep in mind, that the most correct method in terms of physical realism must not be the most optimal one in terms of understanding the data. Further, to render and display 3D values, you always have to create 2D images, which involves a projection and a loss of data, because you throw away one dimension.

Different grid structures:

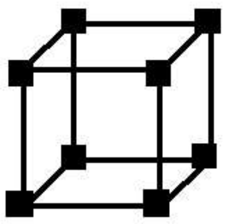
- Structured: uniform, rectilinear, curvilinear
- Unstructured
- Scattered data



Pixel is an abbreviation for picture element; a dot that represents the smallest graphic unit of display on the screen.



Voxel stands for volume element; which is equal to a pixel, but in 3D space. Values are constant within a region around a grid point.



A cell describes the volume framed by grid points, values between grid points are re sampled by interpolation.

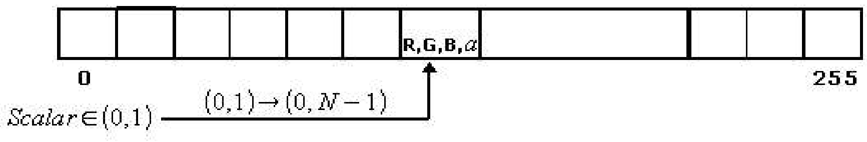
1 Classification

Important for visualization is the process of **classification**, which assigns a material characteristic to each voxel, based on any of a wide variety of data characteristics, such as data value (scalar or vector), derivative measures, or local histograms. The so created material occupancy assignment is called a classification-, or transfer function.

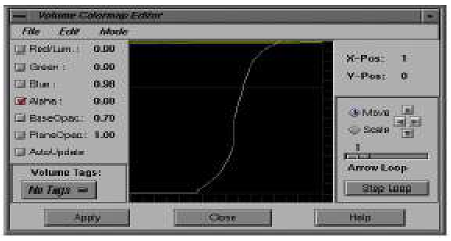
> **Transfer function**
>
> The transfer function describes the relationship between the input and the output of a system. Its role in volume rendering is to map the voxel information to renderable properties of opacity and color.

The classification of transfer functions is non trivial, it is often based on a color table and maps raw voxel value into presentable entities like color, intensity, opacity, etc. By extracting important features of the data set the user is empowered to recognize and select structures. Due to difficult finding of good transfer functions, it is often better to

support interactive manipulation, sometimes a histogram can be a useful hint. The most widely used approach for transfer functions is to assign each scalar value a different color value: $T: scalarvalue \rightarrow colorvalue$. A common choice for color representation is R, G, B, α , where the alpha value describes the opacity. The color values are coded into a color lookup table (LUT) whereby an on-the-fly update is possible.
A known problem of transfer functions is the so called **partial volume effect** which appears, when two or more substances mix in one voxel. In that case you cannot decide which material has to be represented. This problem can be solved with a good pre-classification, that means that each voxel has to be labeled with its associated material.



Coding scalar values into a color lookup table (LUT).


Volume Colormap Editor

File Edit Mode

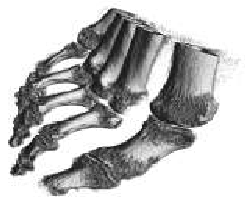
Red/Lum.: 0.90
Green: 0.90
Blue: 0.90
[✓ Alpha:] 0.00
BaseOpac.: 0.70
PlaneOpac.: 1.00
AutoUpdate

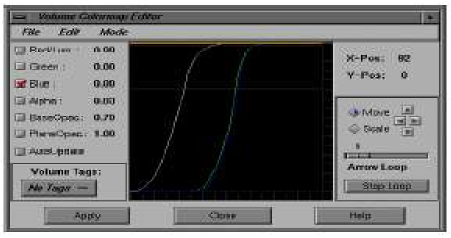
X-Pos: 1
Y-Pos: 0

Volume Tags: No Tags

Move
Scale
Arrow Loop
Stop Loop

Apply Close Help




Volume Colormap Editor

File Edit Mode

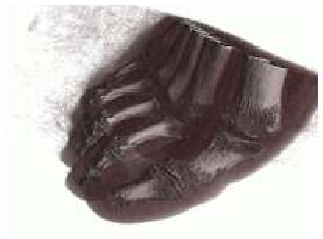
Red/Lum.: 0.00
Green: 0.00
[✓ Blue:] 0.00
Alpha: 0.00
BaseOpac.: 0.70
PlaneOpac.: 1.00
AutoUpdate

X-Pos: 82
Y-Pos: 0

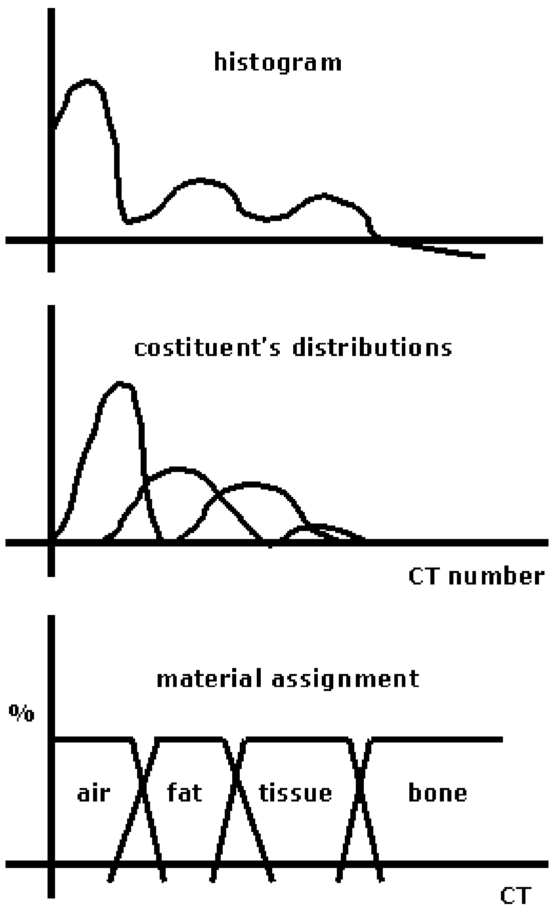
Volume Tags: No Tags

Move
Scale
Arrow Loop
Stop Loop

Apply Close Help

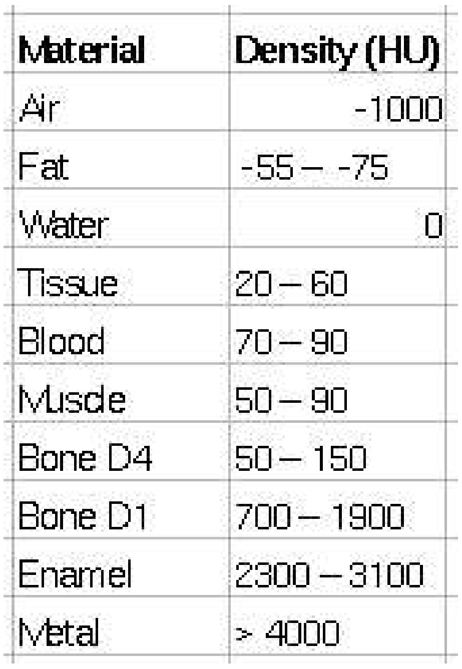


Interactive manipulation of transfer function with different results.



Heuristic approach, based on measurements of data sets.

For densitometry of materials exists the original measuring system of computer tomography by Hounsfield. A **Hounsfield unit** (**HU**) for CT data sets describes the density of material by a 12 bit CT-measurement and ranges from -1000 for air over 0 for water to values over 4000 for carbide. From the arising absorption differences Hounsfield set up a density scale, which itself initially moved from -1000 to +1000. With advanced development of computer and software the range of the firm body fabrics could be expanded further. Thus a very exact density allocation is possible.



Material	Density (HU)
Air	-1000
Fat	-55 – -75
Water	0
Tissue	20 – 60
Blood	70 – 90
Muscle	50 – 90
Bone D4	50 – 150
Bone D1	700 – 1900
Enamel	2300 – 3100
Metal	> 4000

Table with Hounsfield units (HU).

One existing problem with 12 bit CT datasets is that modern graphics hardware only supports a 8 bit color range. This means that for visualization, the 12 bit has to be reduced to 8 bit, which results in a loss of dynamic range.

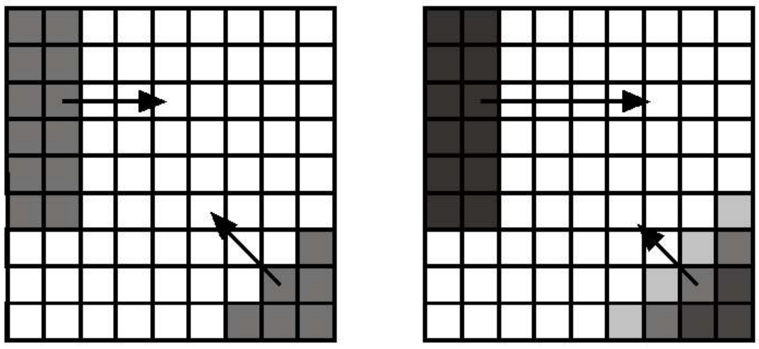
Pre-shading:

First the color values are assigned to the original function value of the lookup table, before texture interpolation is accomplished. This can lead to color-blending artifacts. In practice pre-classification results in a apparently smoothed transitions.

Post-shading:

First the scalar values are interpolated, then the appropriate color from the lookup table is assigned to the interpolated value. This makes a higher detail accuracy possible. In practice transitions of post-classification are much more discrete, but give more volume information of the rendered objects.

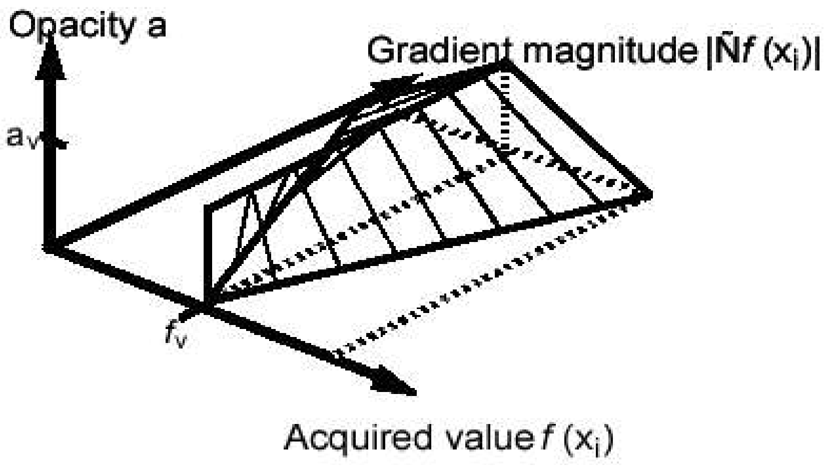
The general interest of volume visualization is not a particular isosurface but whole regions of change. This suggests a feature extraction with a high value of opacity in regions of change. Large homogeneous regions are less important than regions with strong structural changes. In order to emphasize changes it is useful to consider gradients of the scalar field, whereby the transfer function becomes two-dimensional.



The gradient of a pixel points to the direction of largest change. Transitions between same colors (e.g. same gray to white) result in same length of the gradient, no matter of its direction.

A multidimensional transfer function has been introduced by ***[Levoy-1988-DSV]***, he used the gradient magnitude for the second dimension as shown in the illustration below. The gradient as a vector represents the direction of strongest change in the scalar field, the gradient magnitude is a local property and gives us information about how fast values are changing.

At the point f_v in the image below the gradient is very low, which means that the changes in this region of data are very small and the structure one wants to detect here is very thin.
In contrast to that, points lying at the opposite side of the point f_v, have a very high gradient, which means that they cover a fuzzy range of structure and one can detect thick structure regions here.



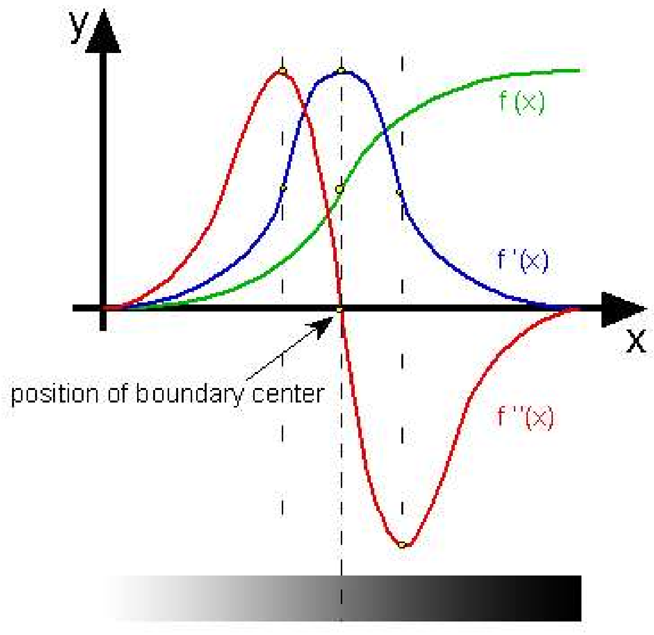
Scalar value and gradient of the scalar field in a transfer function to emphasize isosurfaces $\alpha(x_i)=\alpha_v(1-\frac{1}{r}|\frac{f_v-f(x_i)}{|f'(x_i)|}|)$.

Multidimensional transfer functions are of importance for volume visualization, because by having more variables they can differentiate better between various structures in volume data. Additionally to the gradient magnitude other values can be used to gain dimensions, e.g. the second derivation along the gradient direction. Further on one can use the result of an edge detection algorithm by Marr – Hildreth ***[Kniss-2001-IVR]*** or the Laplacian operator which uses the second derivation of the scalar field. Each variable represents one axis of the transfer function and thus stands for one dimension.

Approach for 3D transfer function can depend on:

- Scalar value
- Magnitude of the gradient
- Second derivative along the gradient direction

One decisive advantage that speaks in favor for using more dimensions is, that thereby the area of transition from e.g. air to bone can be represented, although these materials don't have overlapping HU values as shown in the heuristic approach diagram above.
A big problem of multidimensional transfer functions is, by adding more dimensions, you are adding an enormous number of freedom in which the user can get lost. It is already difficult for one dimension to find a good transfer function because each control point adds two degrees of freedom. Further, transfer functions are non-spatial which means that they do not include the spatial position in their domain.



In this graphic, $f(x)$ *shows the smooth transition between two materials.*

The first derivative $f'(x)$ *represents the gradient, which stands for the strength of magnitude.*

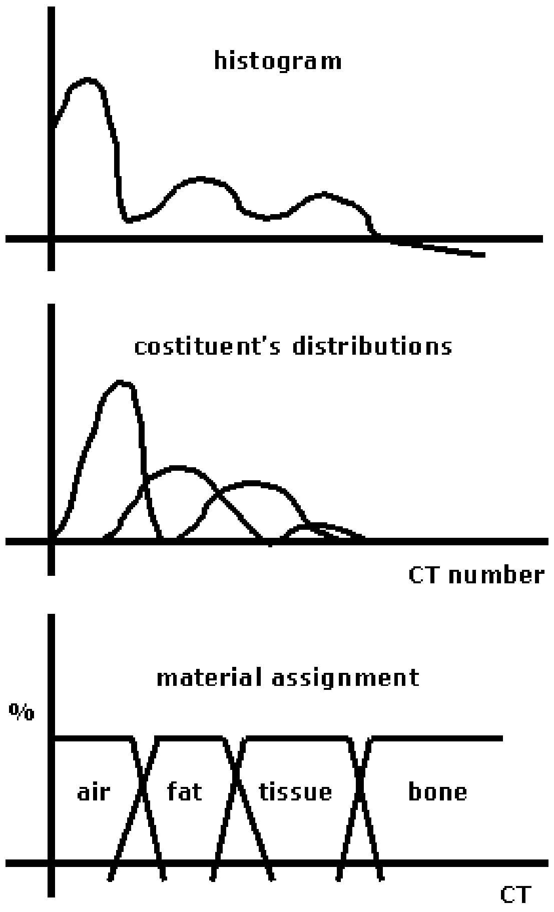
In certain cases this is not precise enough to detect boundaries, so the second derivative $f''(x)$ *can also be used for the transfer function.*

$f(x)$ $f'(x)$

$f''(x)$

2 Segmentation

Segmentation is a pre-processing method and needed for volume rendering to separate different objects from each other. Once the dataset is segmented, those quantities are easily measured. The difficult part of finding an accurate segmentation is that different materials can have the same scalar value, e.g. with a CT scan, different organs have similar X-ray absorption whereby a proper classification can not be distinguished. This is the reason why segmentation is mostly a semi-automatic or even manual process, and requires expert knowledge.

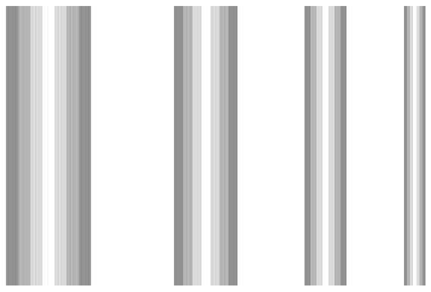


Heuristic approach, based on measurements of data sets.

3 Volumetric Shading

In general shading is used to visualize the 3D structure on a 2D plane. Without shading, different voxels of the same slice will have the same color after mapping by a transfer function. This leads to the fact that we notice the result as a plane 2D surface, although it is a 3D object. By shading this surface, one can pretend the human perception a 3D effect.
With **volumetric shading** techniques, it is possible to create scenes with effects like fog or smoke by simulating the scattering and reflection of light as it passes through the atmosphere. This takes effect on the color of each voxel in the volume dataset, which is generally represented as $RGB\alpha$ color vector. The most common form of shading function is $\vec{RGB\alpha} = \vec{F}_{RGB}(V(\vec{x})) + \vec{F}_{\alpha}(V(\vec{x}), |\vec{\nabla} V(\vec{x})|)$ where $\vec{F}_{RGB}$ and $\vec{F}_{\alpha}$ are vector functions for RGB and α respectively, $V(\vec{x})$ is the volume value and $|\vec{\nabla} V(\vec{x})|$ is the volume gradient length. This means that we are only using the scalar value to calculate the color and we are using both, the scalar value and the gradient magnitude to

calculate the opacity. By interpreting the intensity gradient we want to make use of the human visual system's ability to efficiently deal with shaded objects.



Review of the **Phong illumination** model:
In 3D graphics, the polygons that make up an object need to be shaded. One sophisticated lightning model that eliminates the faceted appearance of flat shading is the standard phong shading method. It is very similar to Gouraud shading, where for each vertex an average normal vector is computed out of the normal vectors of the adjacent surfaces. In addition to that Phong shading interpolates the vertex normals across the surface of a polygon to gain a surface normal at each point for illuminating each pixel. This kind of shading is very expensive and cannot be computed in real time on common hardware. That's why the most implementations are based on optimized approximations. The standard phong shading formula is made up of three components:

Ambient light: $C=k_a C_a O_d$

- k_a is ambient contribution
- C_a is color of ambient light
- O_d is diffuse color of object

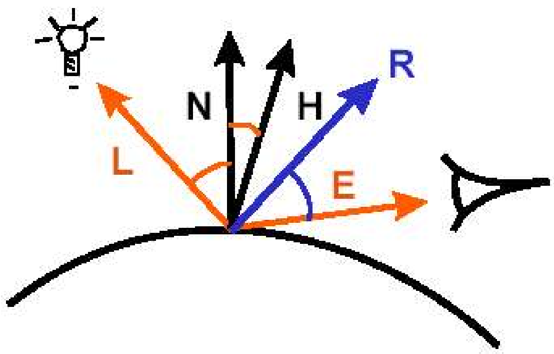
Diffuse light added: $C=k_a C_a O_d+k_d C_p O_d \cos(\theta)$

- k_d is diffuse contribution
- C_p is color of point light
- O_d is diffuse color of object
- $\cos(\theta)$ is the angle between normal vector and incoming light vector

Specular light added: $C=k_a C_a O_d+k_d C_p O_d \cos(\theta)+k_s C_p O_s \cos^n(\sigma)$

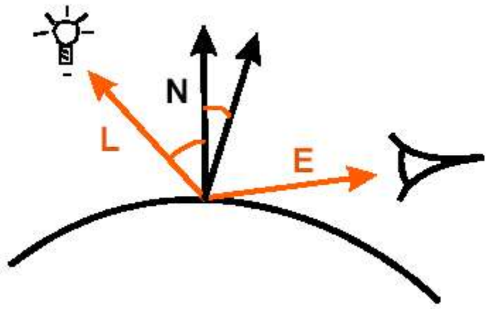
- k_s is specular contribution
- C_p is color of point light
- $\cos(\sigma)$ is the angle between normal vector and halfway vector

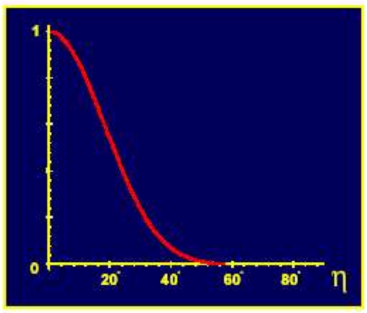
$\cos^n(\sigma)$ stands for the effect of the exponent of highlight. When $n \rightarrow \infty$ then $\cos^n a \rightarrow 0$, which means that the viewer comes close to the reflection-vector. The result is a sharp increase of the light intensity.



$$\mathbf{H} = \frac{\mathbf{L} + \mathbf{E}}{\|\mathbf{L} + \mathbf{E}\|}$$

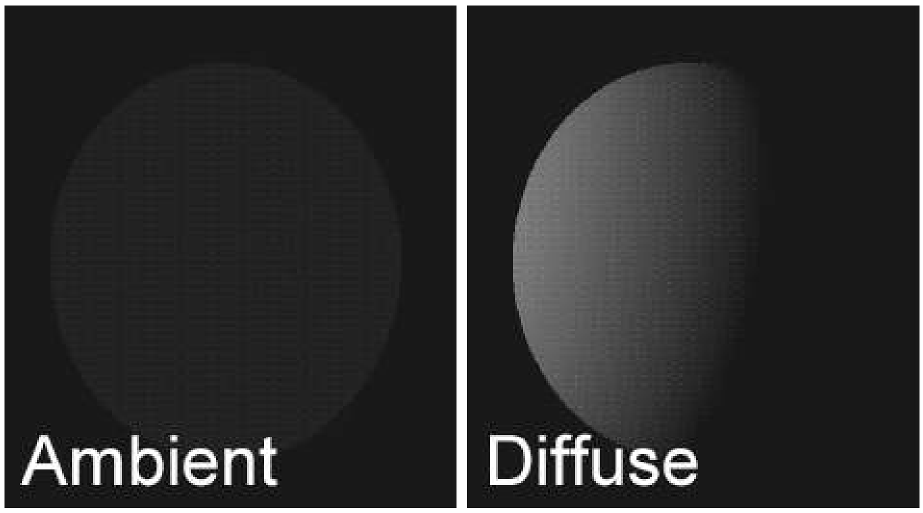
$\cos(\theta) = N * E$ *(Phong)*, $\cos(\sigma) = N * H$ *(Blinn-Phong)* , $H = \frac{L+E}{\|L+E\|}$.

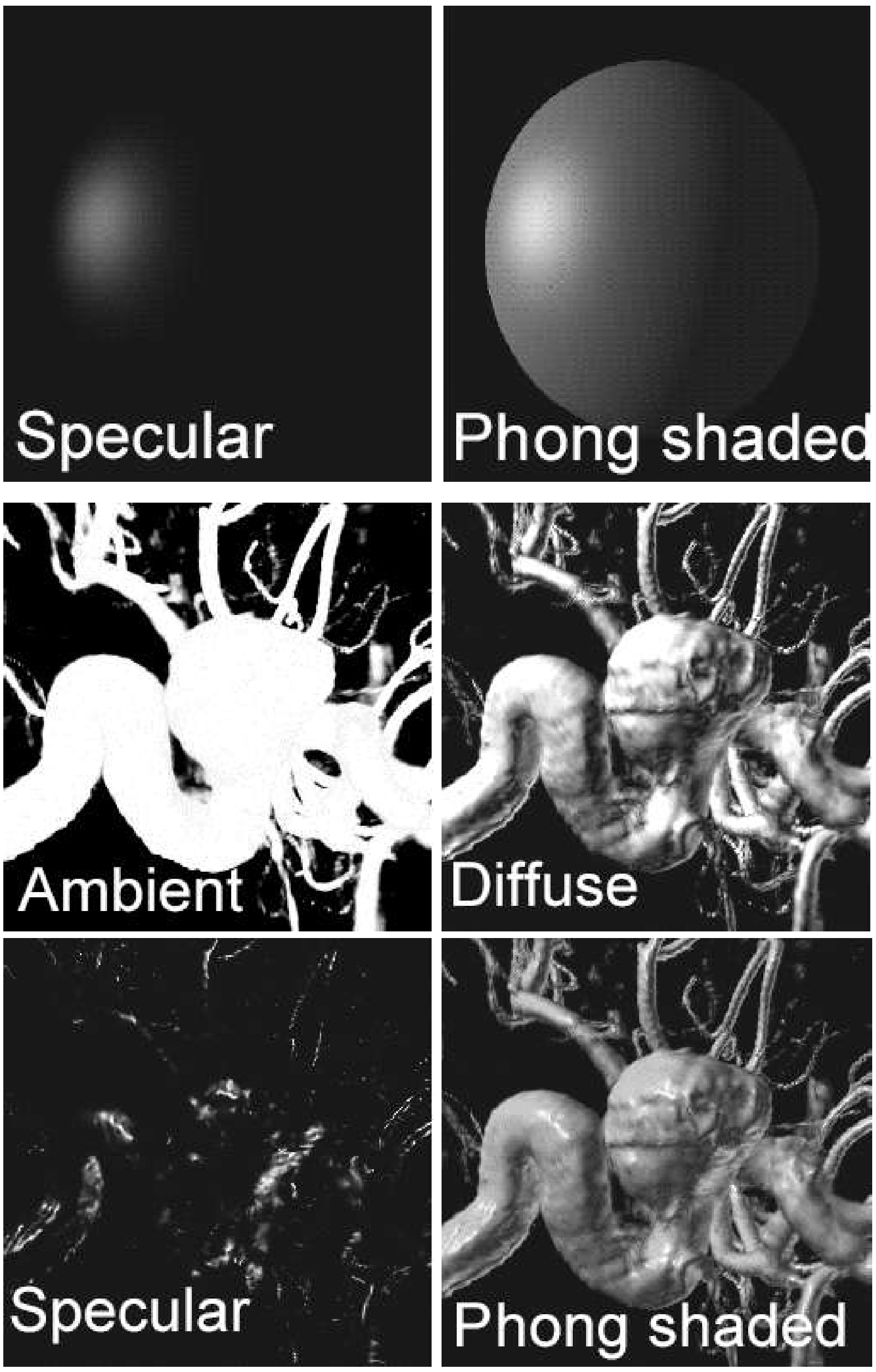




$$\cos(\sigma)^{10} = (\mathbf{N} * \mathbf{H})^{10}$$

Effect of the exponent of highlight. $\cos(\sigma)^{10} = (N * H)^{10}$.





$K_a = 0.1, K_d = 0.5, K_s = 0,4$

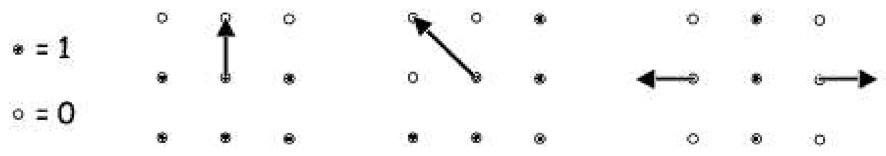
4 Numerical computation of the gradient:

> **Gradient in scalar fields:**
>
> The Gradient is the normal vector in a scalar field and stands perpendicular to the isosurface.

Central difference
Commonly used is the 6-point operator, because of it's fast and easy implementation:

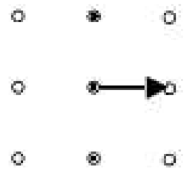
$G_x = V_{x+1,y,z} - V_{x-1,y,z}$

$G_y = V_{x,y+1,z} - V_{x,y-1,z}$
$G_z = V_{x,y,z+1} - V_{x,y,z-1}$
The convolution kernel is very simple, works with subtraction and one to one weighting: [-1 0 1]. It simply computes an average difference of values along each axis. Although this operator is not very accurate, it is a good estimation. The result is a kind of high-pass filter, which smooths noise.
The disadvantage of this operator is its non isotropic characteristic, the magnitude of gradient change with the orientation of boundary, which means the length of the gradient is at a ratio of 1 to $\sqrt{(2)}$ in D (look at the picture below). Further the gradient needs to be normalized.



Central difference. Orientation of the gradient in dependence of the scalar field.

Intermediate difference (forward/backward difference)
Slightly different is the intermediate differences approach. Here, the gradient is calculated right in between sample points and then interpolated, i.e.
$G_{x+\frac{1}{2}} = V_{x+1,y,z} - V_{x,y,z}$
$G_{y+\frac{1}{2}} = V_{x,y+1,z} - V_{x,y,z}$
$G_{z+\frac{1}{2}} = V_{x,y,z+1} - V_{x,y,z}$
This convolution kernel is very simple too: [-1 1]. It is very cheap for computation and it considers its own scalar value. Intermediate differences are more accurate and detect high frequencies.
The disadvantage of this operator is also its non isotropic characteristic and the susceptibility for noisy data makes it less good.



Example for a gradient computed with intermediate difference.

Sobel operator
The sobel operator has a $3 \times 3 \times 3$ convolution kernel like shown in the picture below. It is nearly isotropic and does not depend on the orientation of a structure or boundary in data set.
But because of its 3 dimensional kernel the operator is very expensive (multiple multiplications and summations) and has some additional smoothing in rendering.

$$\begin{array}{ccc} \text{Prev. slice} & \text{this slice} & \text{next slice} \\ \begin{bmatrix} -1 & 0 & 1 \\ -3 & 0 & 3 \\ -1 & 0 & 1 \end{bmatrix} & \begin{bmatrix} -3 & 0 & 3 \\ -6 & 0 & 6 \\ -3 & 0 & 3 \end{bmatrix} & \begin{bmatrix} -1 & 0 & 1 \\ -3 & 0 & 3 \\ -1 & 0 & 1 \end{bmatrix} \end{array}$$

partial derivative along the *z*-axis

other axes by rotation

The three slices of the sobel operators convolution kernel.

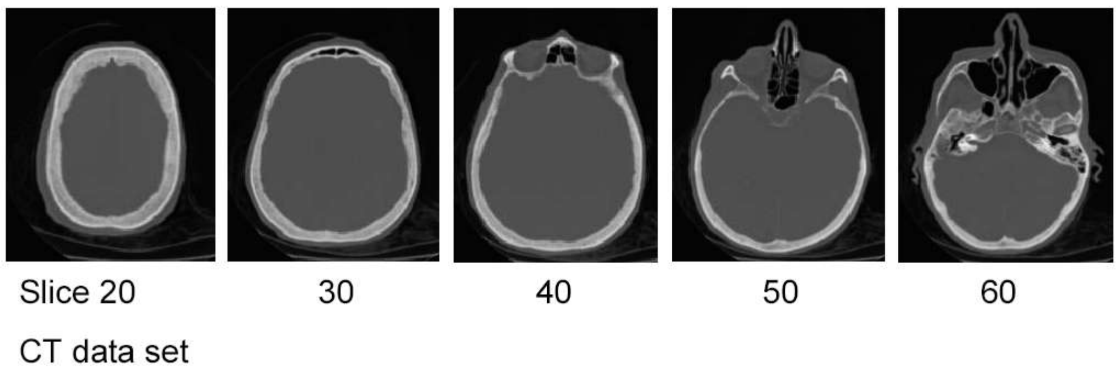
$$\begin{bmatrix} -1 & 0 & 1 \\ -3 & 0 & 3 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -3 & 0 & 3 \\ -6 & 0 & 6 \\ -3 & 0 & 3 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -3 & 0 & 3 \\ -1 & 0 & 1 \end{bmatrix}$$

5 Slicing

For indirect volume rendering there are two approaches, isosurfacing and slicing, which extract a subset of data and visualize the subset with traditional rendering techniques. Slicing can be divided again in two different procedures: Orthogonal and oblique slicing.

Orthogonal slicing

- Interactively resample the data on slices perpendicular to the x-,y-,z-axis
- Use visualization techniques for 2D scalar fields
    - Color coding
    - Isolines
    - Height fields

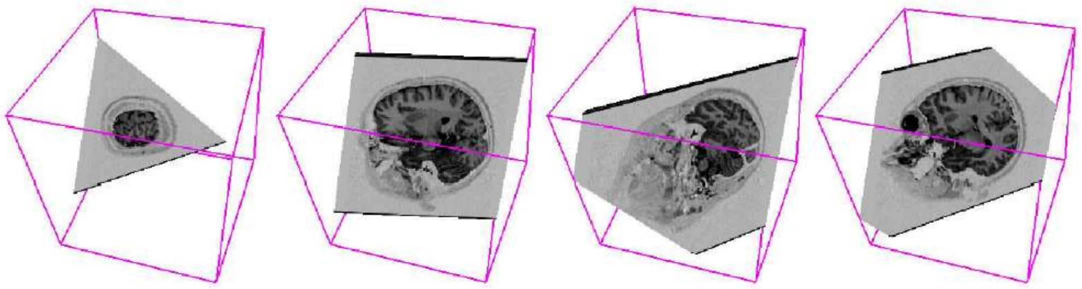


Result of orthogonal slicing.

Oblique slicing

- Resample the data on arbitrarily oriented slices
- Resampling in software or hardware
- Exploit 3D texture mapping functionality
    - Store volume in 3D texture
    - Compute sectional polygon (clip plane with volume bounding box)
    - Render textured polygon

Each pixel value of the generated slice plane can simply be found by taking the corresponding cell from the volume for the given x,y,z coordinate and interpolating the values given at the eight corners of the cell.



Examples of oblique slicing.

6 Indirect Volume Rendering

Contrary to direct volume rendering, **indirect volume rendering** techniques first transfer the volume dataset into a new domain, before it is rendered. These algorithms are often chosen because of their speed advantage, or a possible hardware acceleration, although they are not so precise. The general idea of those techniques is, if $f(x,y,z)$ is differentiable in every point, then the level-sets $f(x,y,z)=c$ are isosurfaces to the defined isovalue c. That means that the algorithm goes through all voxels and determines, if each voxel belongs to the isosurface with value c. Common indirect volume rendering techniques to determine and reconstruct isosurfaces from volume data are:

- Contour tracing
- Cuberille, opaque cubes
- Marching cubes/tetrahedra

Contour tracing
The contour tracing approach was often used in prominent medical applications before the marching cubes algorithm was invented. The simplified proceeding of contour tracing can be described as follows. It is a local operation on a (by a threshold value) binarized 2D slice of the volume dataset. By clockwise traversing the adjacent pixels of the contour, a chain of pixels is gained that forms a polyline. The proceeding to find isosurfaces from 2D contours can be as follows:

    - Segmentation: find closed contours in 2D slices and represent them as polylines
    - Labeling: identify different structures by means of the isovalue of higher order characteristics
    - Tracing: connect contours representing the same object from adjacent slices and form 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

$\rightarrow$ Tracing (assignment) becomes very difficult, so the main task of contour tracing is, how to correctly connect the vertices of the triangles on different isosurfaces.

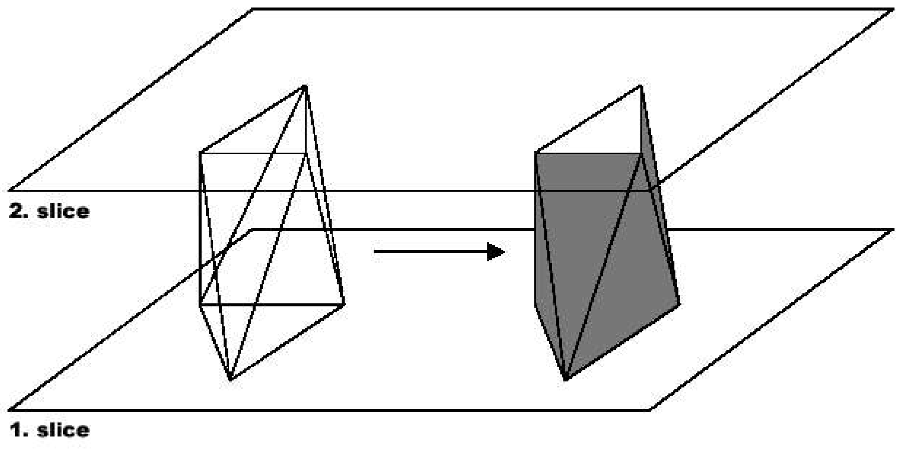
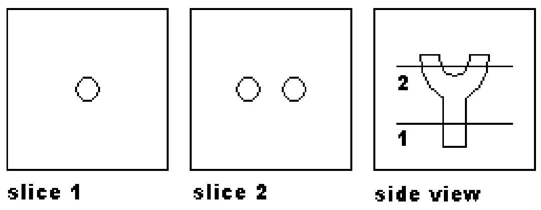


Illustration of contour tracing between two different slices.



Problem with contour tracing: labeling of a vessel branch in medical data volumes.

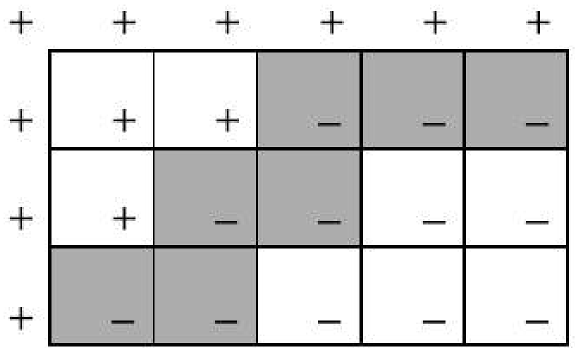
Of course there is not only one single way of representing a surface in indirect volume rendering, it always depends of the projection used. To make this clear two more methods will follow.

Generic **surface fitting** techniques

- Choose an isovalue (arbitrarily or from segmentation)
- Detect all cells the surface is passing through by checking the vertices
- Mark vertices with respect to $f(x,y,z) \geqslant c\ (+) \quad \vee \quad f(x,y,z) < c\ (-)$
- Consider all cells with different signs at vertices
- Place graphical primitives in each marked cell and render the surface

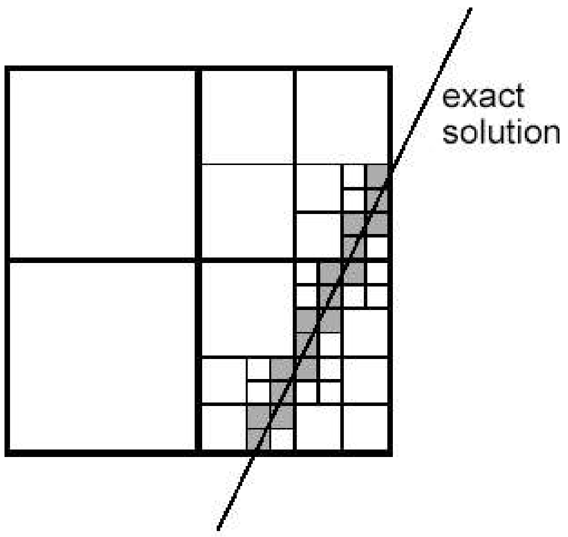
Cuberille (opaque cubes) approach ***[Herman-1979-DHO]***
(A) Binarization of the volume with respect to the isovalue
(B) Find all boundary front-faces
if the normal of each face points outward the cell, find all faces where the normal points towards the viewpoint $(\vec{N} \cdot \vec{V} > 0)$
(C) Render these faces as shaded polygons

- “Voxel” point of view: NO interpolation within cells. The approximated boundary is not very precise.



The cuberille approach does not use interpolation, it marks whole cells.

- Cuberille approach yields blocky surfaces
- Improve results by adaptive subdivision
- Subdivide each marked cube into 8 smaller cubes
- Use trilinear interpolation in order to reconstruct data values at new cell corners
- Repeat cuberille approach for each new cube until pixel size



Subdivision of the quadtree for the cuberille approach.

7 Marching Cubes

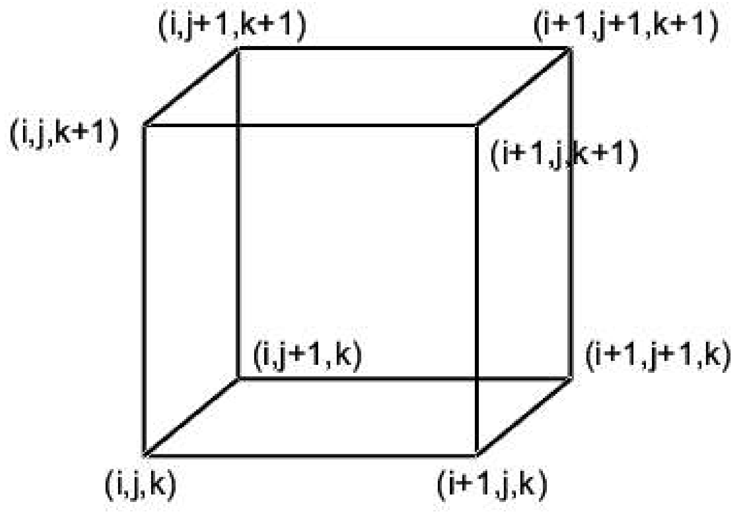
In order to get a better approximation for rendered isosurfaces of volumetric data, the **Marching Cubes** (**MC**) algorithm was developed by ***[Lorenson-1987-MCA]***.
The algorithm works on the original volume data, it defines a voxel (cube) by the pixel values at the eight corners of the cube. This cube is “marching” through the whole volume dataset and subdivides space into a series of cubes. At each step we classify each vertex of the cube as inside or outside the isosurface. Edges that are adjacent to one “inside” and one “outside” classified vertex are intersected by the isosurface and we can create a triangle patch whose vertices are found by linear interpolation along those cell edges. Further we use the gradients as normals of the triangle surfaces. By connecting the patches from every step of the cube we get an approximated isosurface represented by a triangle mesh. We gain efficient computation by means of lookup table that stores all

possible constellations of triangle patches.
MC is THE standard geometry-based isosurface extraction algorithm!

The core MC algorithm:

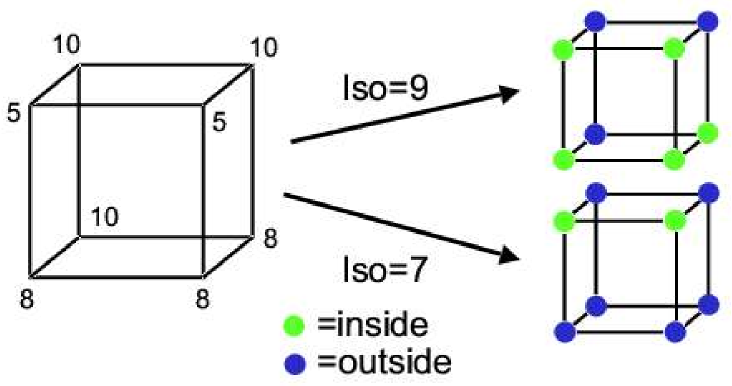
- Cell consists of 4(8) pixel (voxel) values: $(i+[01], j+[01], k+[01])$

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

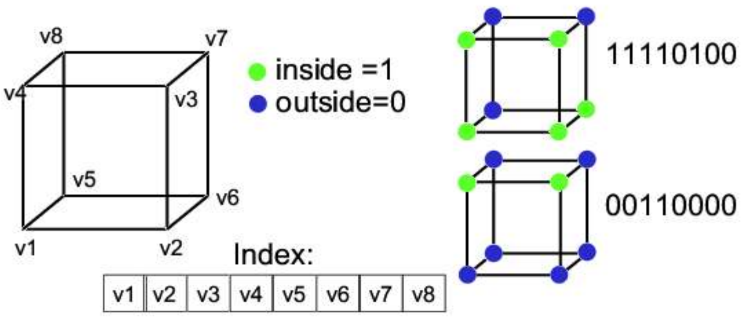


(i,j,k) $(i+1,j,k)$ $(i+1,j,k+1)$ $(i,j,k+1)$ $(i,j+1,k)$
$(i+1,j+1,k)$ $(i+1,j+1,k+1)$ $(i,j+1,k+1)$

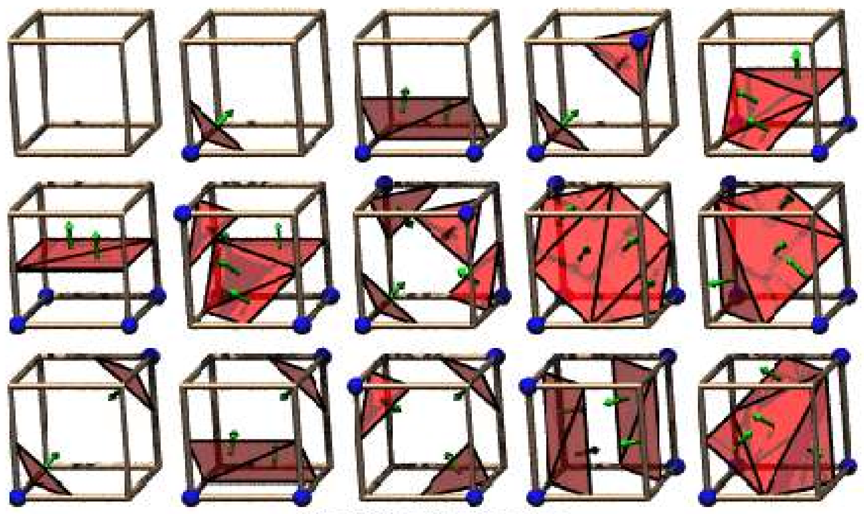
- Step 1: Consider a cell defined by eight data values.



- Step 2: Classify each voxel according to whether it lies
- outside the surface, when $voxel\,value > iso\,value\,c$ (+ or 0)
- inside the surface, when $voxel\,value \leq iso\,value$ (- or 1)

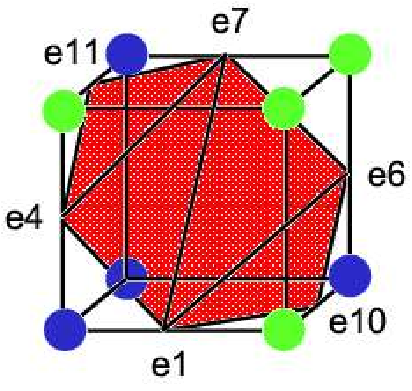


- Step 3: Use the binary labeling of each voxel to create an index

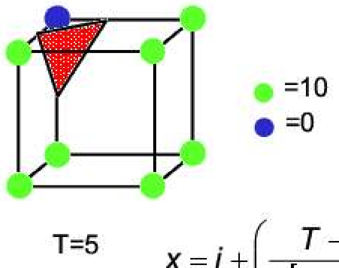

The 15 Cube Combinations

- Step 4: For a given index, access an array storing a list of edges
    - All 256 cases can be derived from 15 base cases due to symmetries.
    - Get edge list from table
    - Example for Index = 10110001

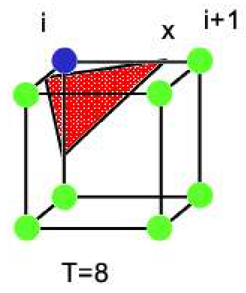
    triangle 1 = e4,e7,e11
    triangle 2 = e1, e7, e4
    triangle 3 = e1, e6, e7
    triangle 4 = e1, e10, e6



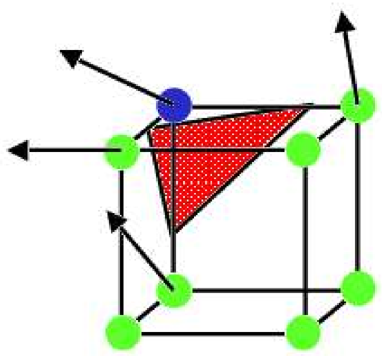
With this code you know for each triangle, which edge is intersected, but you don't know where exactly it intersects.



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



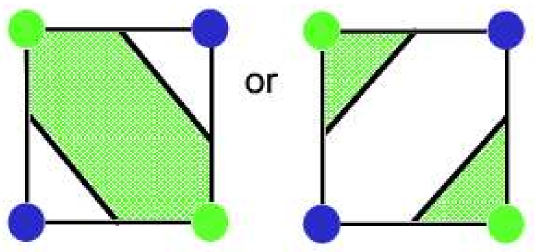
- Step 5: For each triangle edge, find the vertex location along the edge using linear interpolation of the voxel values
- This step has to be done for all cubes and all edges that intersect the current cube.

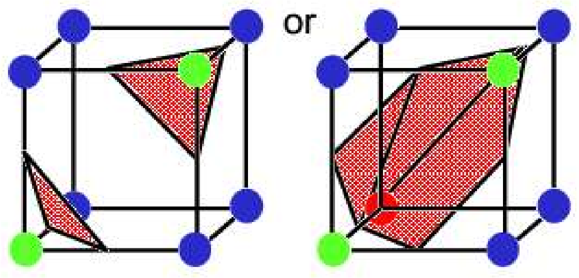


- Step 6: Calculate the normal at each cube vertex

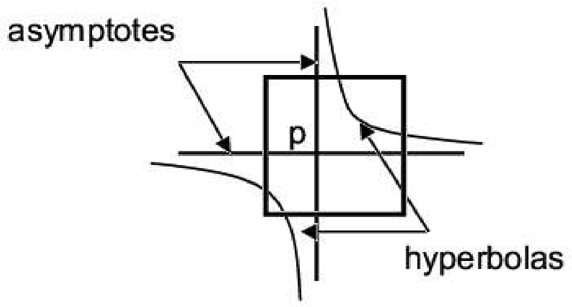
$$G_x = V_{x+1,y,z} - V_{x-1,y,z}$$
$$G_y = V_{x,y+1,z} - V_{x,y-1,z}$$
$$G_z = V_{x,y,z+1} - V_{x,y,z-1}$$

- Use linear interpolation to compute the polygon vertex normal (of the isosurface). After that normalization is needed to do smooth shading (gouraud shading).
- Note that different isosurfaces can never intersect, because each one has only one unique isovalue.



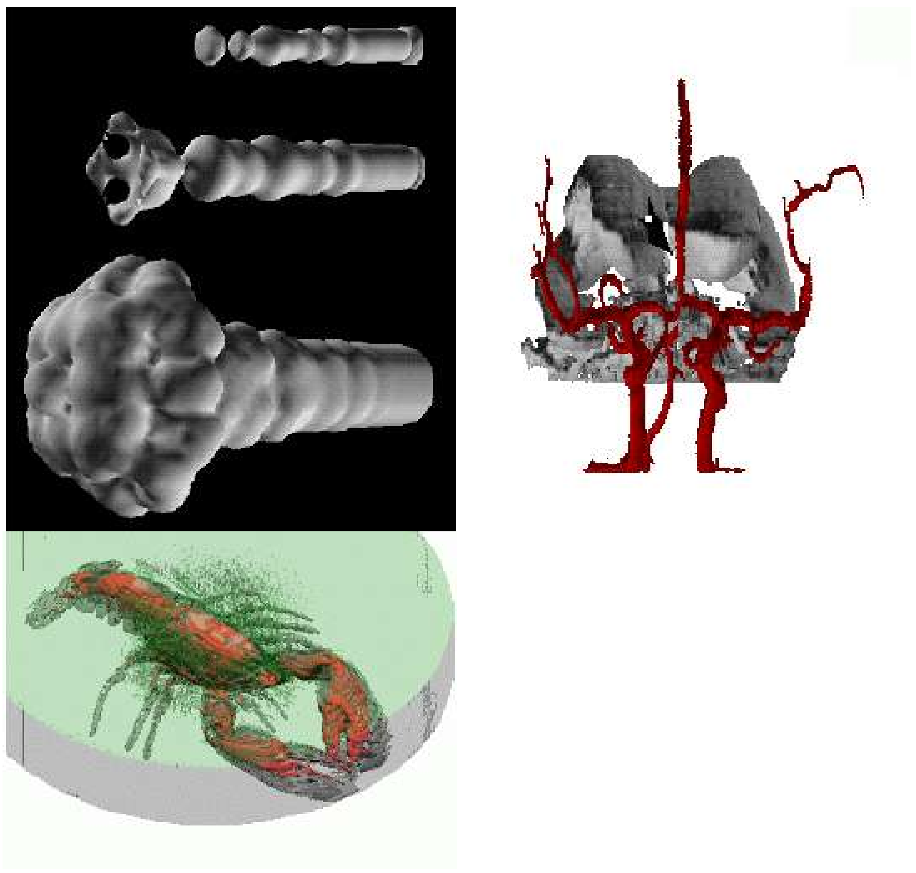


- Step 7: Consider ambiguous cases
    - Ambiguous cases: 3, 6, 7, 10, 12, 13
    - Adjacent vertices: different states
    - Diagonal vertices: same state
    - Resolution: decide for one case
    - Asymptotic Decider ***[Nielson-1991-TAD]***
    - Assume bilinear interpolation within a face
    - Hence isosurface is a hyperbola
    - Compute the point p where the asymptotes meet on the face
    - Sign of S(p) decides the connectivity



- **Summary:**
    - 256 Cases
    - Reduce to 15 cases by symmetry
    - Ambiguity resides in cases 3, 6, 7, 10, 12, 13
    - Causes holes if arbitrary choices are made
- Up to 5 triangles per cube
- Dataset of 512×3 voxels can result in several million triangles (many MB)
- Semi-transparent representation $\rightarrow$ sorting

- **Optimization:**
    - Reuse intermediate results
    - Prevent vertex replication
    - Mesh simplification

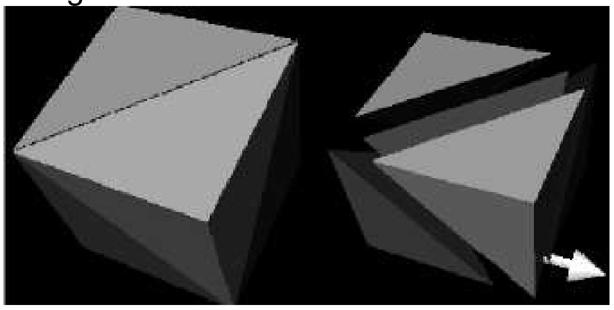

Example for 1 isosurface, 2 isosurfaces and 3 isosurfaces.

8 Marching Tetrahedra

The **Marching Tetrahedra** algorithm was developed by ***[Shirley-1990-PAR]***, it is very closely related to the Marching Cube algorithm because the fundamental idea is the same. Primarily it was used for unstructured grids, by splitting the cell into tetrahedras it was easier to handle. Due to the simpler geometry of the tetrahedra we only have three different cases how the isolines can intersect.

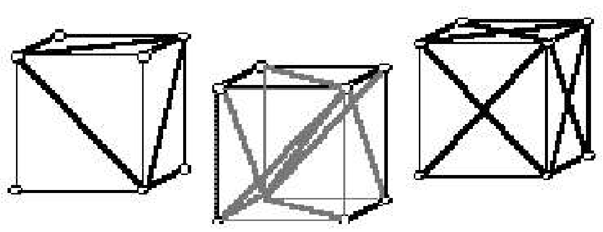
1. No intersection
2. One vertex negative (-) and three vertices positive (+) or vice versa, so the surface is defined by one triangle.
3. Two vertices negative (-) and two vertices positive (+), so the surface is defined by a quadrilateral that can be divided in two triangles using the shorter diagonal.

Further interpolation became much easier because we can use linear interpolation on triangular surfaces.


Split the cell into 5-6 tetrahedras.

Properties of Marching Tetrahedra:

- Fewer cases, i.e. 3 instead of 15
    - no problems with consistency between adjacent cells
- Number of generated triangles might increase considerably compared to the MC-algorithm due to splitting into tetrahedra
- Huge amount of geometric primitives
- But, several improvements exist:
    - Hierarchical surface reconstruction
    - View-dependent surface reconstruction
    - Mesh decimation



Mesh decimation for Marching Tetrahedra.

An interesting question is, how to shade isosurfaces generated by Marching Tetrahedra? The answer is in deed not very difficult, since the function for interpolating the colors varies linear between the coordinates of the vertices, the gradient is the first derivative and it will be constant. So we compute an average gradient for each vertex using each surface normal (which is the gradient) of all adjacent triangles.

9 Dividing Cubes

Dividing Cubes was established by ***[Cline-1988-ROT]*** it is one acceleration approach of the standard marching cubes algorithm. Nowadays it is not used any longer, but however it is historically worth mentioning. The algorithm works on uniform grids and takes the advantage of the observation that the size of generated triangles, when rendered and projected, is often smaller than the size of a pixel. So the basic idea is to create surface points instead of triangles, that means the input volume is subdivided down, until a cube has approximately the same size as a pixel. This allows a point based rendering. The surface normal which is needed for rendering is the averaged normal of the cubes corner normals.

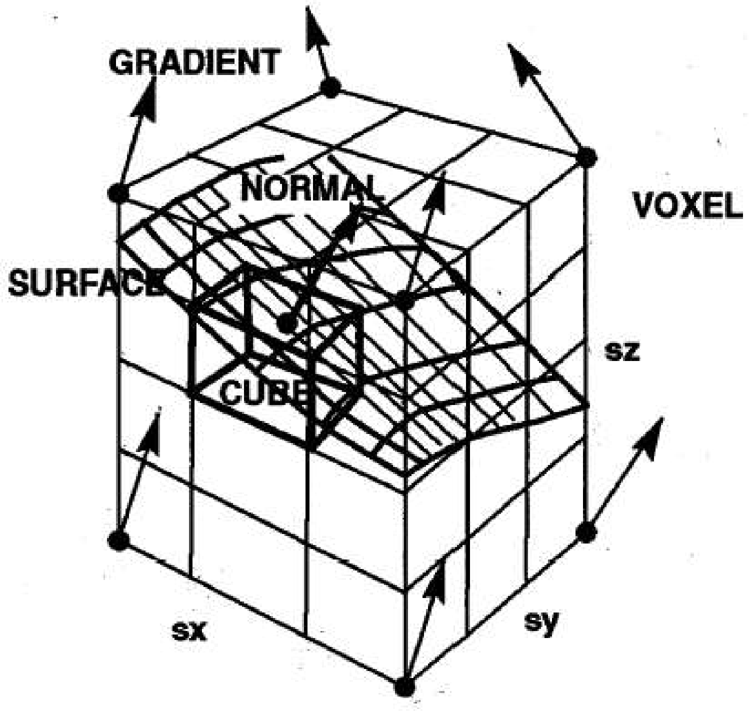
The algorithm works as follows:

- Choose a cube
- Classify, whether an isosurface is passing through it or not
- If (surface is passing through)
    - Recursively subdivide cube down to pixel size
- Compute normal vectors at each corner of the cube
- Render shaded points with averaged normal

Properties:

- View dependent load balancing, that means when you look at the object from a certain side, the subdivision of the cubes can stop earlier, which leads to a speed up.
- Better surface approximation due to trilinear interpolation within cells.
- Only good for rendering, but since no surface representation is generated it does not allow further computations on the surface. So once a model is rendered it cannot be

scaled for various resolutions anymore since it was generated with a particular display resolution.
- Eliminates scan conversion step
- Point cloud rendering randomly ordered points
- No topology

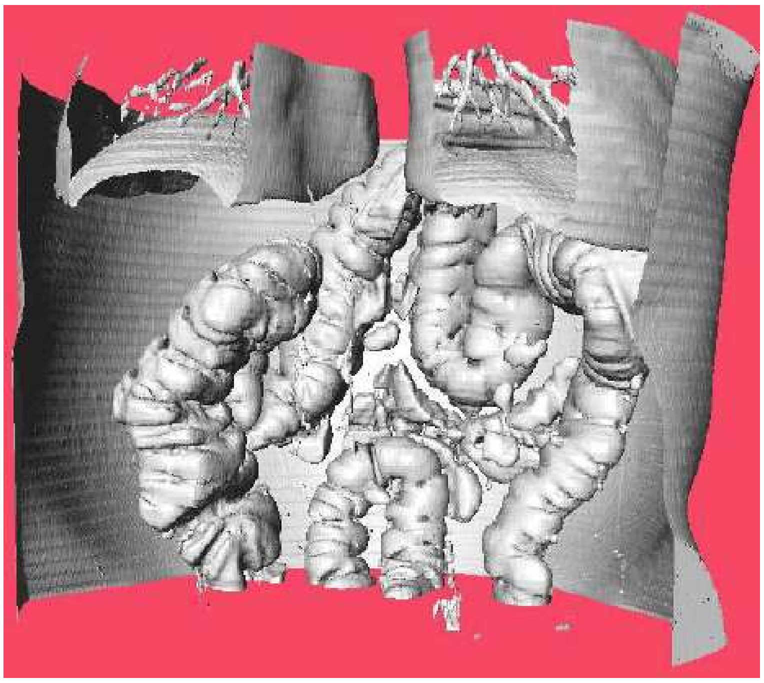


Subdividing the voxel into pixel sized cubes.

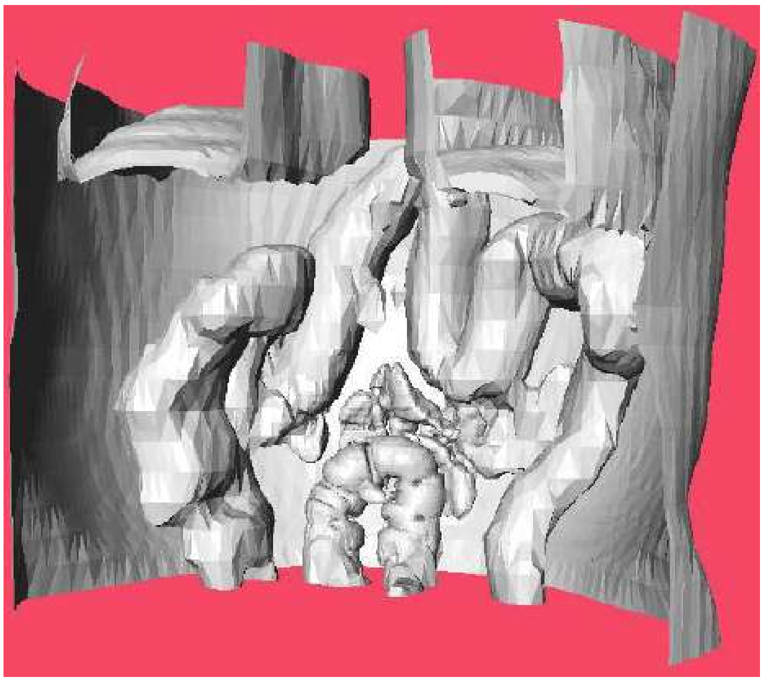
10 Optimization of Fitted Surfaces

All surface fitting techniques produce a huge amount of geometric primitives which can be a problem for interactive rendering. Therefore several improvements exist like:
- Hierarchical surface reconstruction
- View-dependent surface reconstruction
- Mesh decimation

Hierarchical surface reconstruction
Try to reconstruct the surface hierarchically by generating copies of the dataset at different resolutions. For lower resolutions down sample the eight neighbor voxels into one, and then compute Marching Cubes.
For displaying select level-of-detail (**LOD**) based on error criterion like the distance of approximation to "original" surface. If things are far away, an object is rendered with fewer polygons.

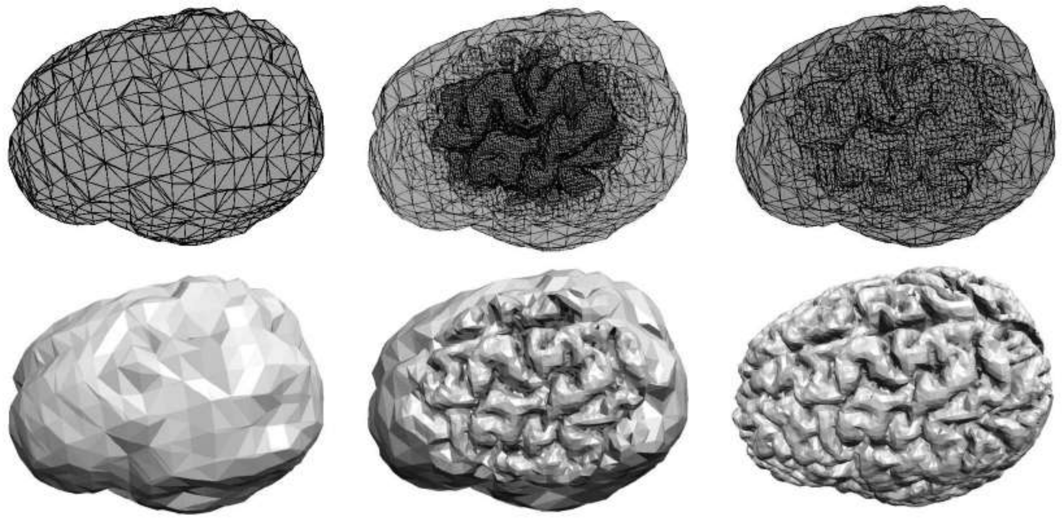


Full reconstruction of a human abdomen with 6M triangles.



Reconstruction of the same abdomen with 123K triangles.

View-dependent surface reconstruction
Here we have a user defined *level-of-detail* (focus point oracle, like a lens that could be moved across the volume, points near the focus we have better resolution parts further away we have inferior resolution).
With *view frustum culling* regions that are outside the viewing pyramid are avoided to reconstruct. This makes close-up scenes with few objects much faster.
Further on *occlusion culling* avoids the reconstruction of regions that are already occluded by a surface (this implies front-to-back traversal). Efficient occlusion culling is very difficult and only done as pre-processing step.
Finally there is the dividing cubes idea, which means to avoid the reconstruction in cells that are below pixel size.


View-dependent surface reconstruction with focus point oracle in the middle of the brain. It is needful to show the surroundings, so that the user has a feeling for the context.

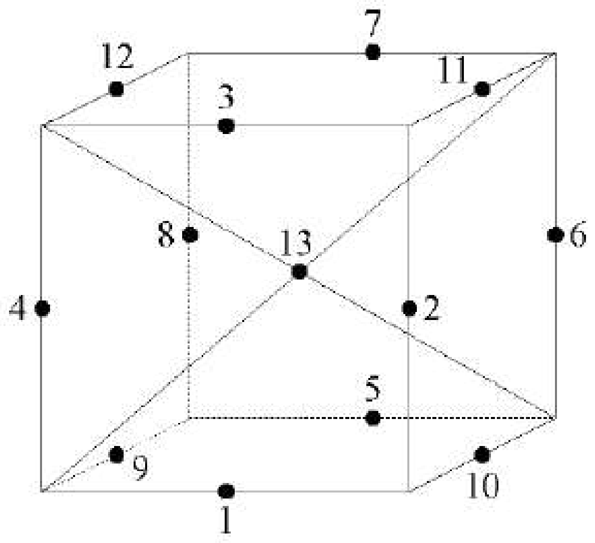
Mesh decimation
Mesh decimation algorithms are usually applied to geometric models, nevertheless they can be used to simplify isosurfaces. For example you first use Marching Cubes to generate an isosurface and then use the decimation algorithm to minimize the quantity of triangles. The basic concept is to remove triangles, end up with polygons and re-triangulate these with less triangles. Consider deviation between mesh before and after decimation, then generate a hierarchical mesh structure as a post-process and switch to appropriate resolution during display.

Picture of a buddah statue with differently strong mesh decimations.

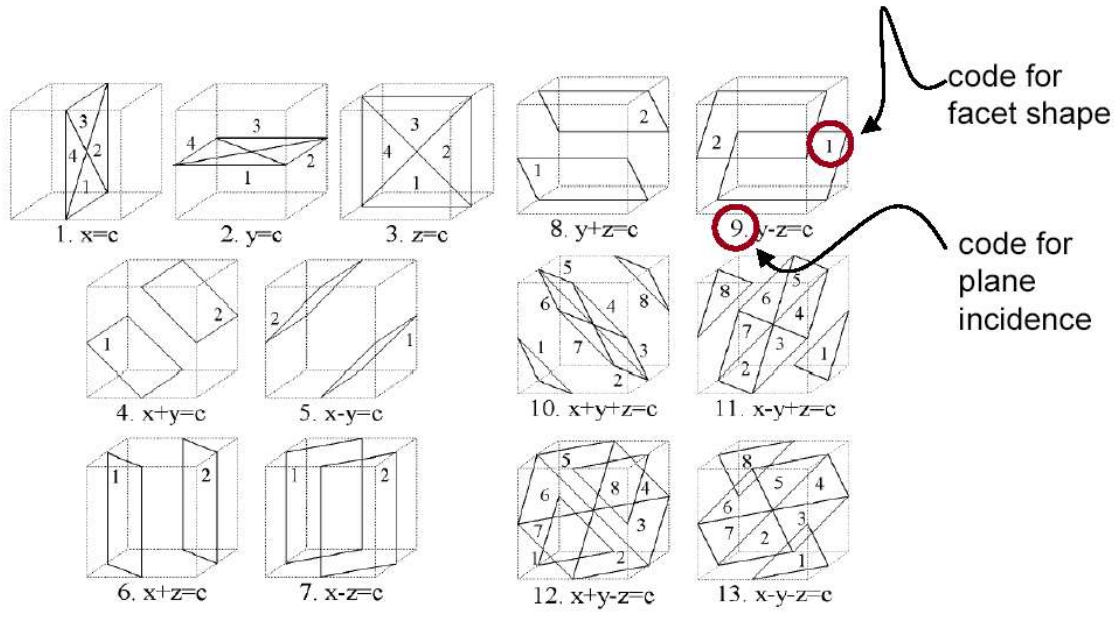
11 Discretized Marching Cubes

Decisive for the idea of the **Discretized Marching Cubes** algorithm was the ambition to find an algorithm that does not create so many triangles. One way but the wrong way was to run marching cubes first to create isosurfaces and to apply Mesh Decimation to the result. But it turned out very soon that this way was very slow and not efficient. The Discretized Marching Cubes (**DiscMC**) is a mixture in-between the Cuberille approach

(constant scalar value on each voxel) and the Marching Cubes (trilinear interpolation in cells), it was published by ***[Montani-1994-DMC]*** and is based on the following idea.
If the cube is not much larger than a pixel, it is not useful to create much smaller triangles. To accelerate the standard MC algorithm the DiscMC specifies that if an isosurface intersects the cube, it always intersects the middle of the edges, thus one saves the linear interpolation. Further this constraint leads to a limited set of planes with also restricted orientations.

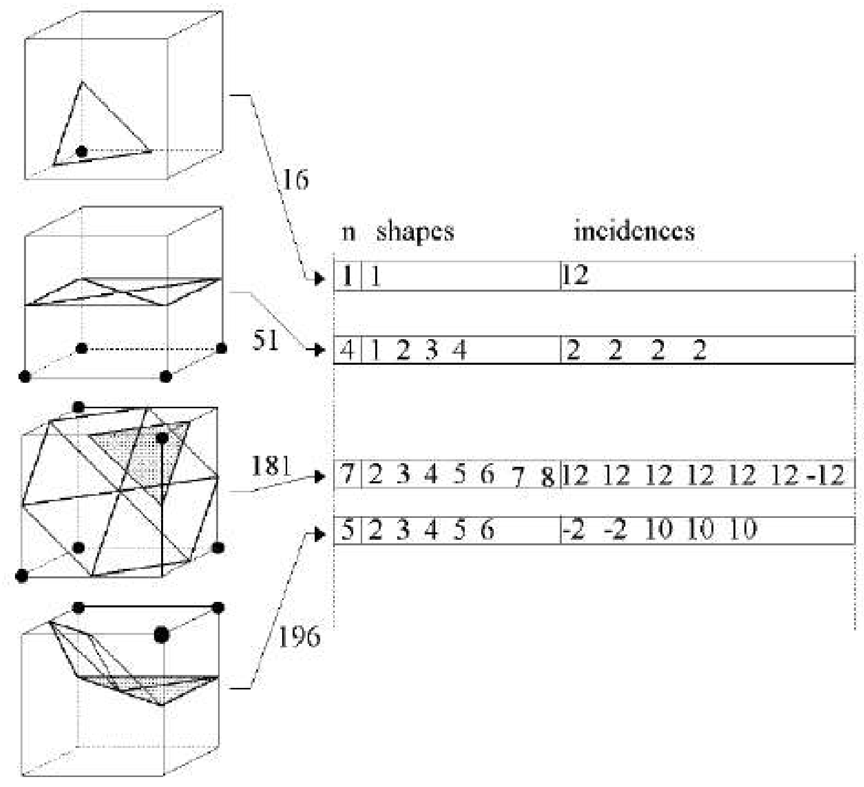


For each cube there are 13 different vertex positions, 12 edge-midpoints + 1 centroid.



Finite set of planes on which faces can lie.

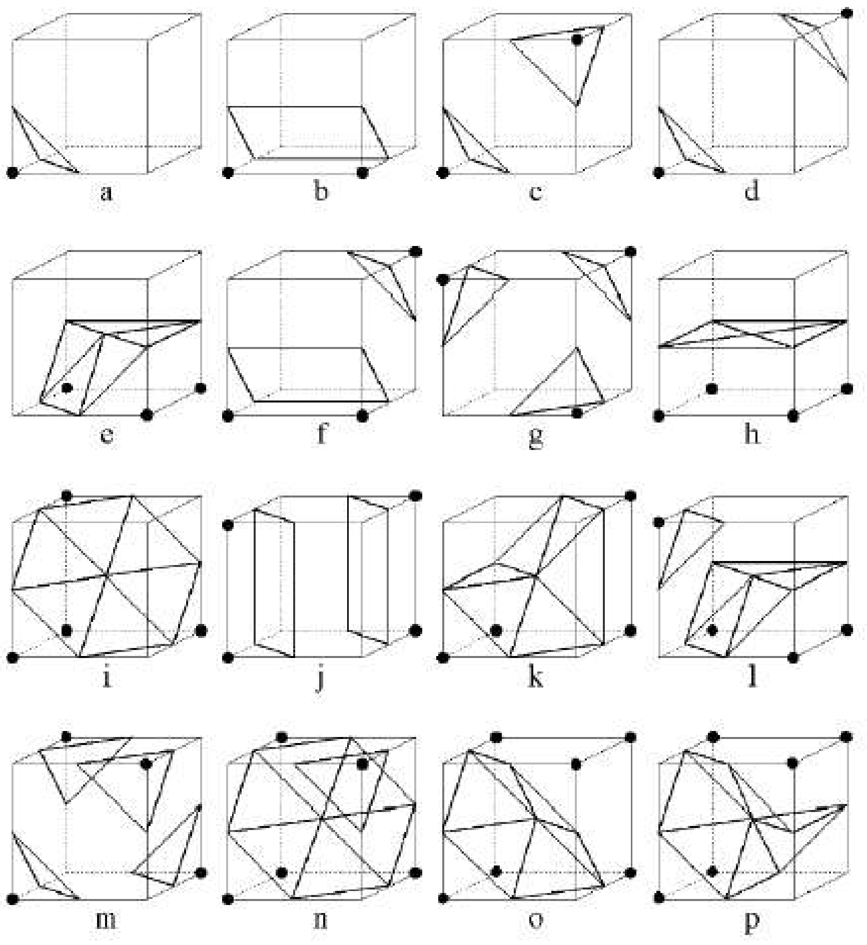
- Classification of a facet by
    - Plane incidence (code of the orientation of the normal) and
    - Shape
- Sign of incidence determines orientation of facet
- Classification of isosurface fragment (facet set)
    - Indices to incidences and shapes



Classification of the facets.

- 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



All remaining possibilities stored in a Lookup table.

Algorithm:

- Analogously to MC: traversing the grid
- Normal vectors based on gradients (same as MC)
- Post processing: merging facets and edges

Advantages of MC:

- 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 results

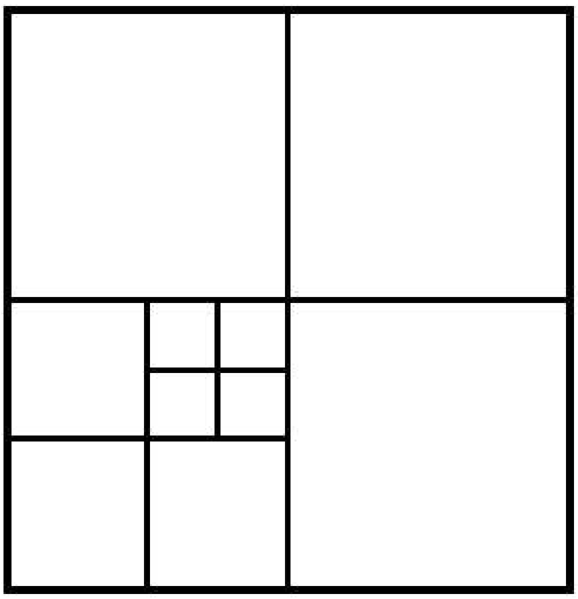
Good to know:

- Triangles or planes in neighboring cubes continue with the same surface structure, because of limited number of orientation.
- For shading it is not necessary to use the normals of the discretized triangles, one can use the information from the original data set.
- It turned out that although of all this significant improvement, in industry no medical product implemented this algorithm.

12 Octree-Based Isosurface Extraction

In order to accelerate the MC algorithm there exist two more interesting approaches, the

domain spaced approaches which operate on the voxel values and the range query approaches based on scalar values. All these algorithms have in common that they search as fast as possible for cells, which contribute to isosurfaces. We will start with a domain based approach.
The **Octree**-based approach was published by ***[Wilhelms-1992-FIG]***, and it works with a spatial hierarchy on a grid that is constructed as a tree. During generation of the tree, for each node the minimum and maximum scalar value is stored by looking at the values of all 8 children (for the 3 dimensional approach). Once this tree is computed, it can be traversed (bottom up) to search for an isosurface with a specified isovalue. Obviously the isosurface cannot be inside of a voxel, if the isovalue does not lie in-between the stored minimum and maximum values of the node. In this case we can skip those parts of the tree.


Geometrical structure of an quadtree in 2 dimensional space.

It is important to find the right data structure for the octree. If you have a data set with one million voxels and you plan to use a pointer structure to save the volume, you have to allocate one pointer for each voxel. This means that you have one million pointers, each pointer itself has a size of a double word which corresponds to 32Bit. So you end up with 3.9MB only for the data structure. So it is better to use a simple array like structure, especially when you work with a full octree, because you just have to allocate one array, which is much smaller since it only includes start point and size.

Advantages when using full octree:

- Simple array-like structure and organization
- No pointers needed
- Number of nodes in full octree:

$$n_{nodes} = i=0\left(\log_2 s\right)-18^i = s^3 - \frac{1}{7} \approx 0.14n_{datapoints}$$

→ optimal ratio is data $\frac{n_{nodes}}{n_{datapoints}} \approx 0.14$

Problem with memory consumption of complete octree:
- Ideal: grid size of $2^n \cdot 2^n \cdot 2^n$
- Normally different resolutions that are not powers of two

Example:

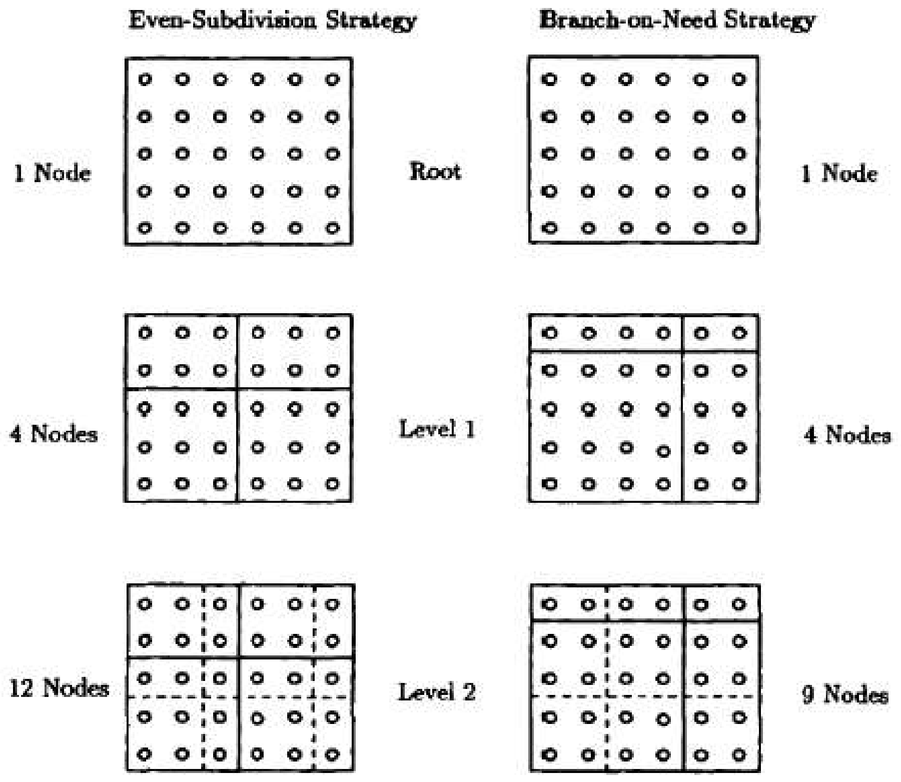
- Data set: 32032040
- 4M data points
- Full octree: $1+2^3+4^3+\ldots+256^3=20M$ elements (nodes)
- 2 values per element: minimum and maximum values

Solution: Branch on Need Octree (BONO)

- Consider octree as conceptionally full
- Avoid allocating memory for empty subspaces
- Delay subdivision until needed
- Allocate only dimensions of powers of two
- Aspects of a bottom-up approach

- For above example: approx. 585k nodes (opposed to 20M nodes)

- Ratio almost optimal: $\frac{n_{nodes}}{n_{datapoints}} \approx 0.1428$
- Ratio never exceeds 0.162 (~16% memory overhead)

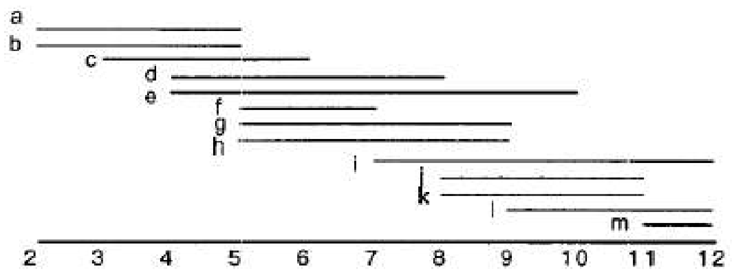


Example for Even-Subdivision and BONO Strategy.

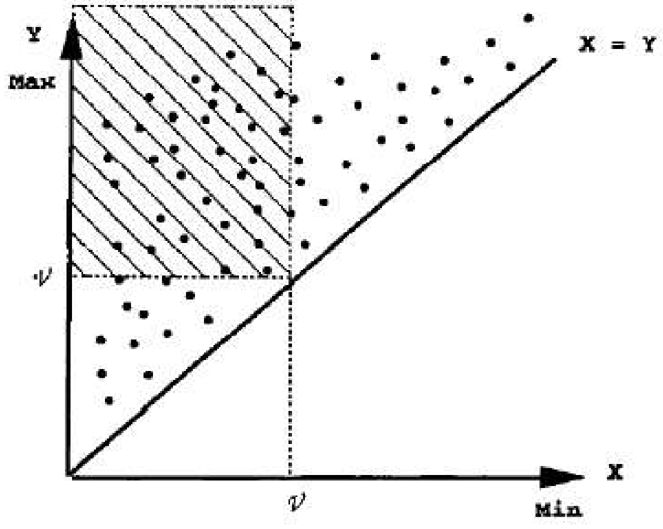
13 Range Query for Isosurface Extraction

The second alternative to accelerate MC are **range query** approaches, which data structures are based on scalar values and not like before on domain decomposition .

The one dimensional interval space approach works as follows:
All possible isovalues are listed along the x-axis. For each cell the min and max value is determined, considering all vertices that belong to the cell. The computed min and max values for each cell are stored in a sorted list (for example sorted by min value). Now you can extract an isosurface with a specific isovalue by searching the list for cells with a smaller min and a greater max value. E.g. a search for isovalue 7 in the graphic below returns the cells d, e, f, g, h, i.

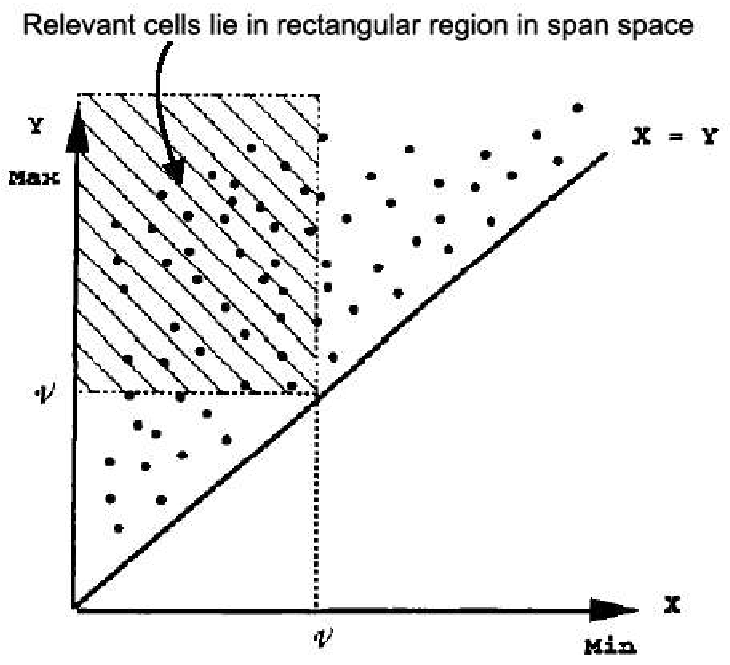


One dimensional interval structure for min / max values.



In two dimensional span space.

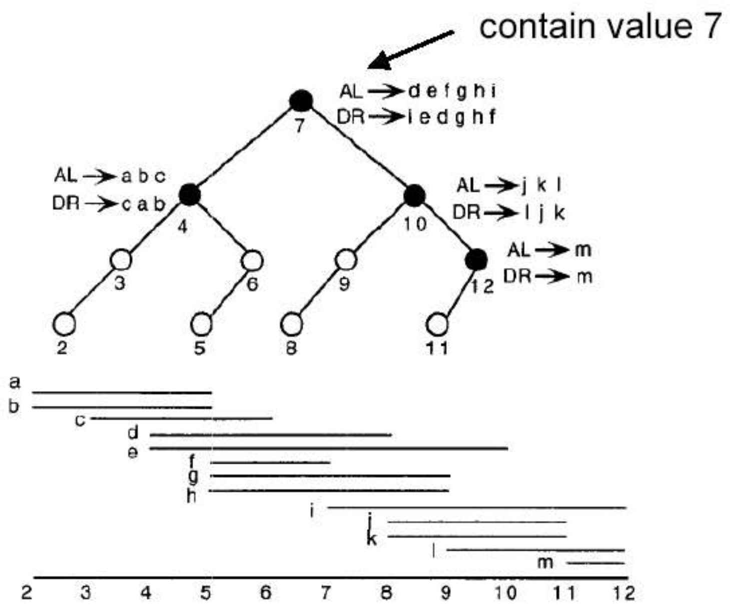
The two dimensional span space approach bases on the following idea:
Each cell is represented by one point, whose x value corresponds to the minimum and y value to the maximum. All points lie above the main bisecting line, because otherwise they would have a larger min then max value. If you search for a specific isovalue you know that all relevant points lie in a rectangular region in span space.



Span space approach.

The Problem for both one-, and two dimensional approaches is how to find the corresponding cells efficiently. One way to do so is the “optimal isosurface extraction from irregular volume data” published by ***[Cignoni-1996-OIE]***. This method uses a so called interval tree with the following characteristics:

- h different extreme scalar values
- Balanced tree: $height = \log h$
- Bisecting the discriminant scalar value
- Node contains:
    - Scalar values
    - Sorted intervals AL (ascending left)
    - Sorted (same) intervals DR (descending right)

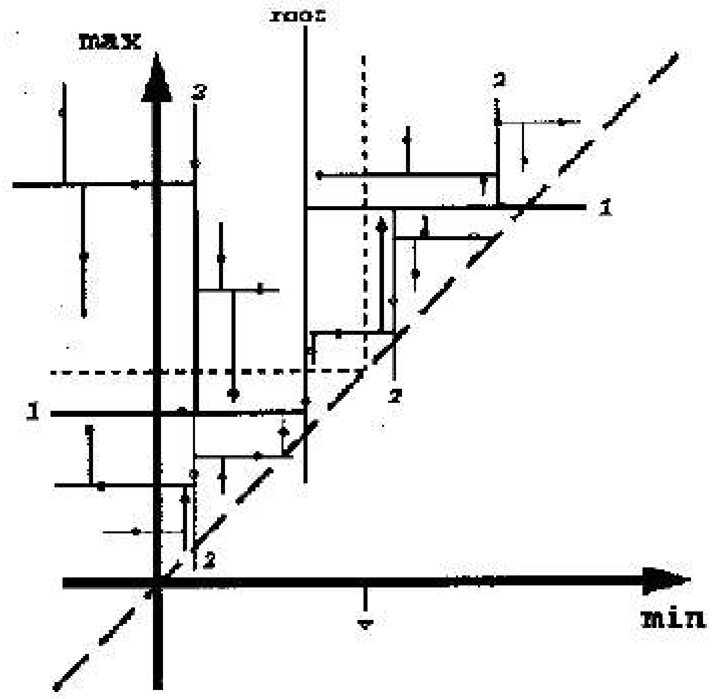


Example for sorted interfals data structure for ascending left (AL) and descending right (DR). The root node in this graphic has an error in the DR case.

- Running time: $O(k+\log h)$ due to
    - Traversal of interval tree: $\log h$ (height of the tree)
    - k intervals in the node = number of relevant cells (i.e., output sensitive)

Variations of the above range query based on interval trees:
- Near optimal isosurface extraction (NOISE) ***[Livnat-1996-IEA]***
- Isosurfacing in span space with utmost efficiency (ISSUE) ***[Shen-1996-ISS]***

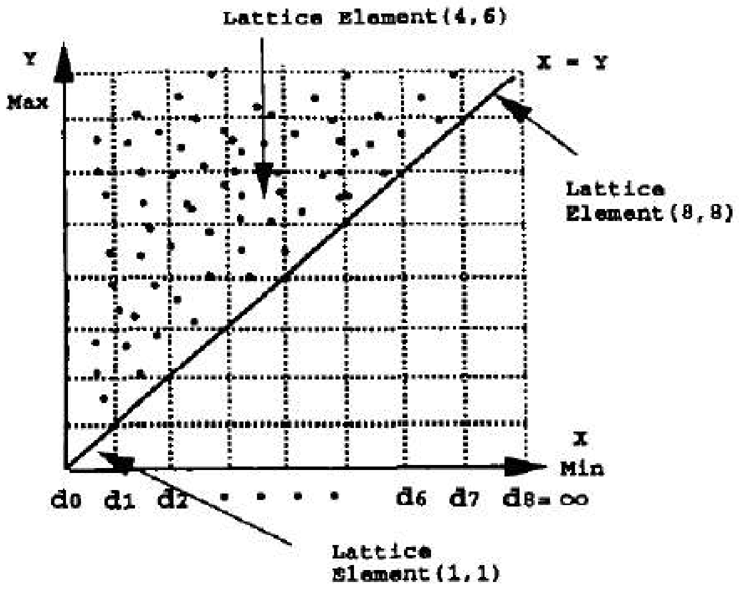
NOISE:
- Based on span space
- Create Kd-tree for span space, here you begin with one axis perpendicular to either x or y axis and divide further (see picture below). The result is a balanced tree.
- Worst case running time: $O(k+\sqrt{n})$ where
- k = number of relevant cells (with isosurface)
- n = total number of grid cells



Near optimal isosurface extraction. Kd-tree with its axes perpendicular to x or y axis.

ISSUE: Isosurfacing in span space with utmost efficiency
- Based on span space
- Lattice subdivision on span space
- Average running time: $O\left(k+\log\left(\frac{n}{L}\right)+\frac{\sqrt{n}}{L}\right)$
- L = dimension of grid in x and y

- All range-query algorithms suitable for structured and unstructured grids.



Example for Isosurfacing in span space with utmost efficiency.

14 Contour Propagation

Contour propagation is an acceleration of cell traversal and proceeds as follows. Look for a seed cell from which one knows it contains an isosurface and from there on, visit all adjacent cells but avoid visiting empty cells. Problem is that isosurfaces can consist of several not connected components, and for every one a seed cell must be found or whole parts of the isosurface will be lost.

Algorithm:
- Trace isosurface starting at a seed cell
- Breadth-first traversal along adjacent faces
- Finally, cycles are removed, based on marks at already traversed cells

Similar to 2D approach
- Same problem:
    - Find ALL disconnected isosurfaces
    - Issue of optimal seed set