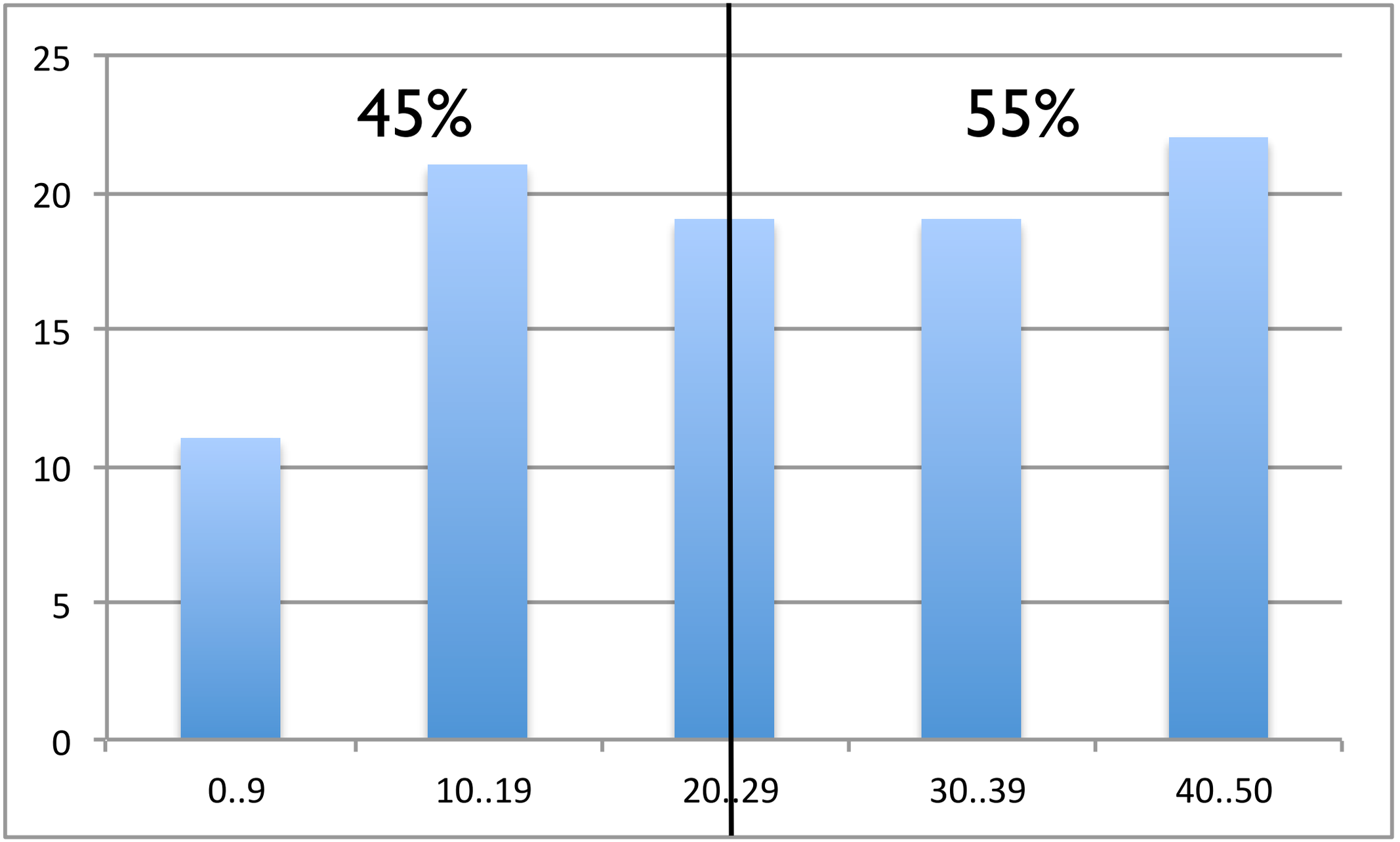
Computer Graphics

-Volume Rendering-

Oliver Bimber

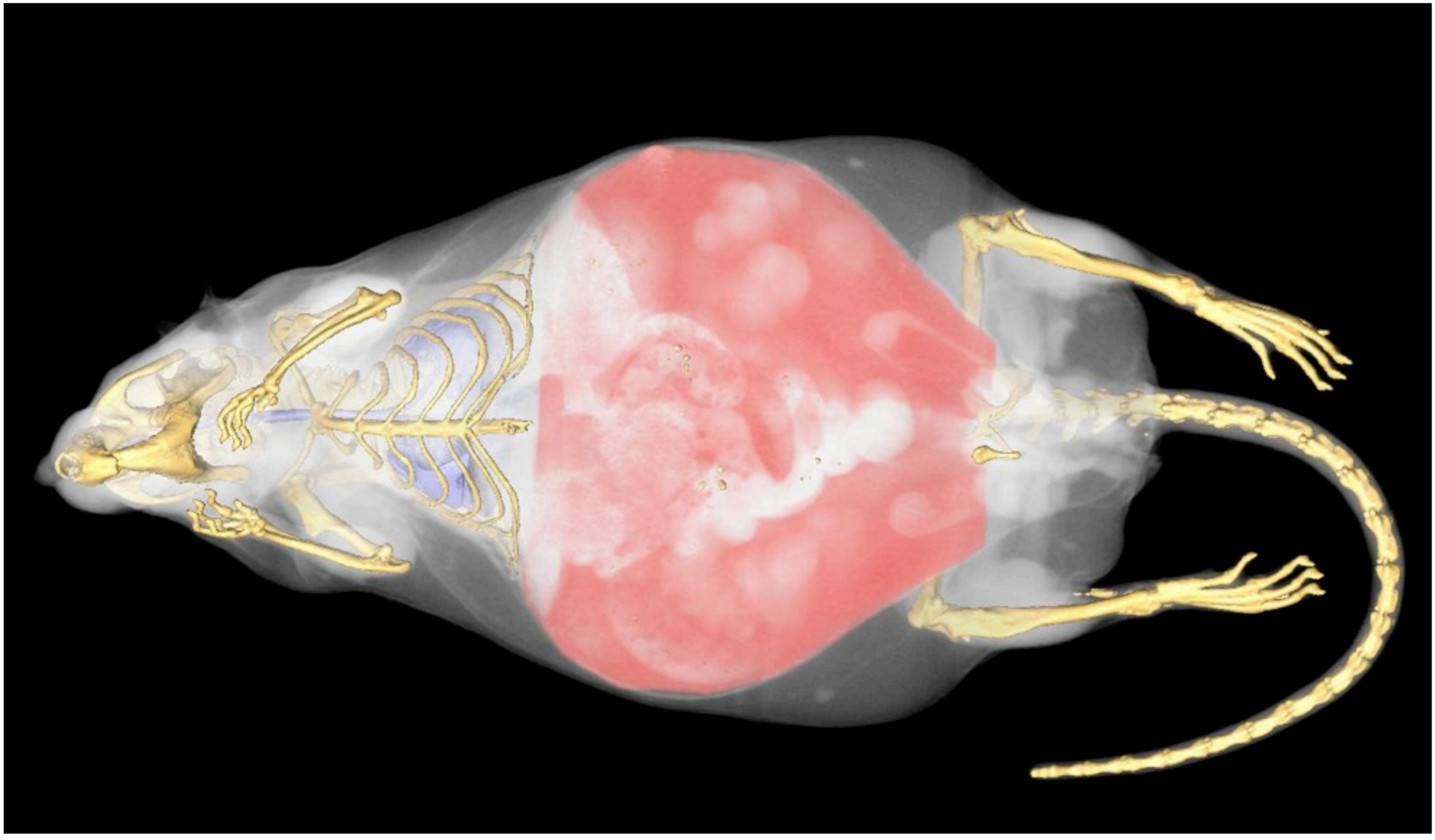
Intermediate Exam Results



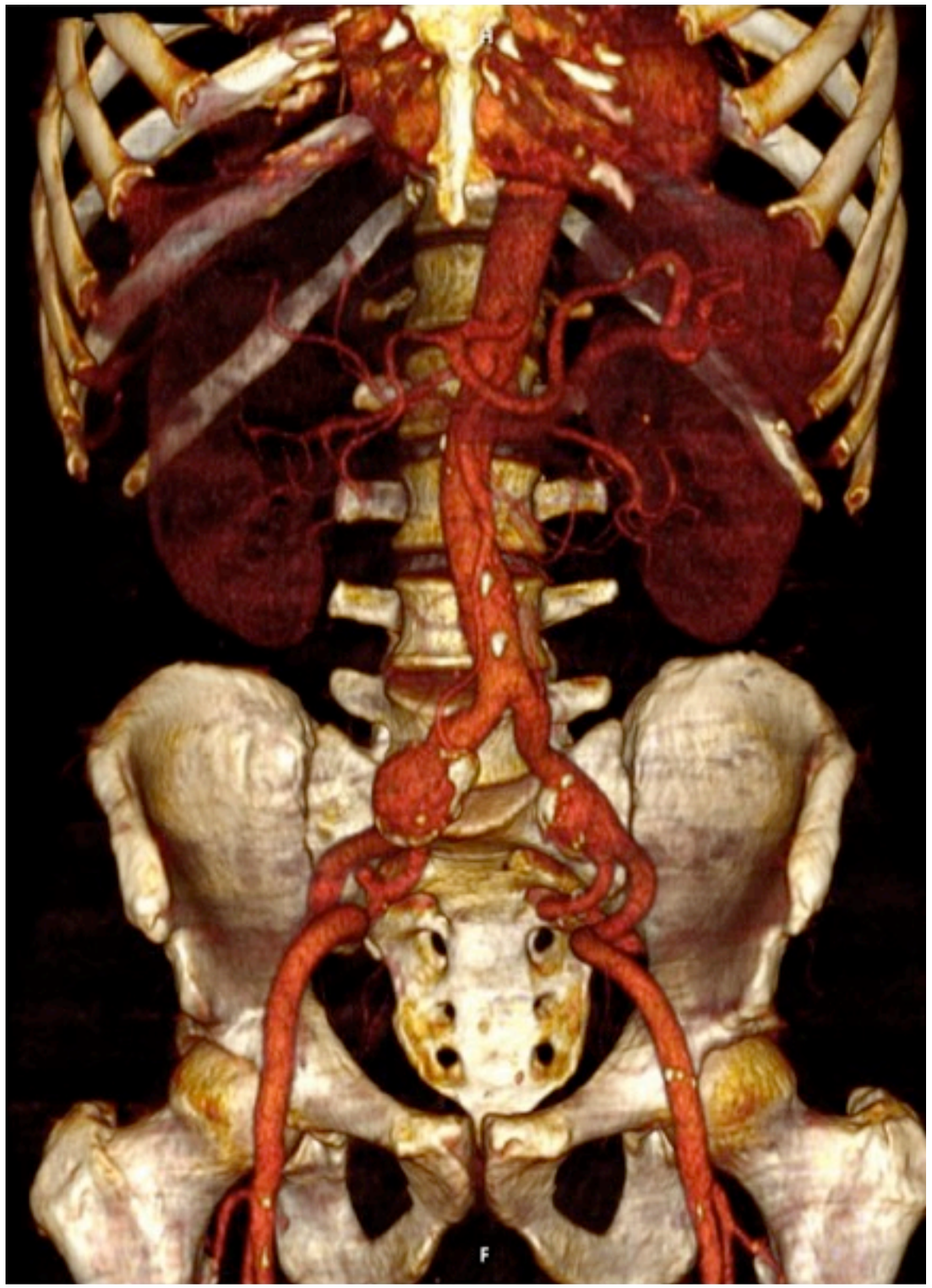
Course Schedule

Type	Date	Time	Room	Topic	Comment
C1	01.03.2016	13:45-15:15	HS 18	Introduction and Course Overview	Conference
C2	15.03.2016	13:45-15:15	HS 18	Transformations and Projections	Easter Break
C3	05.04.2016	13:45-15:15	HS 18	Raster Algorithms and Depth Handling	
C4	12.04.2016	13:45-15:15	HS 18	Local Shading and Illumination	
C5	19.04.2016	13:45-15:15	HS 18	Texture Mapping Basics	
C6	26.4.2016	13:45-15:15	HS 18	Advanced Texture Mapping & Graphics Pipelines	
C7	03.05.2016	13:45-15:15	HS 18	Intermediate Exam	
C8	09.05.2016	17:15-18:45	HS 18	Global Illumination I: Raytracing	
C9	10.05.2016	13:45-15:15	HS 18	Global Illumination II: Radiosity	Conference / Holiday
C10	31.05.2016	13:45-15:15	HS 18	Volume Rendering	
C11	07.06.2016	13:45-15:15	HS 18	Scientific Data Visualization	
C12	14.06.2016	13:45-15:15	HS 18	Curves and Surfaces	
C13	21.06.2016	13:45-15:15	HS 18	Basics of Animation	
C14	28.06.2016	13:45-15:15	HS 18	Final Exam	
C15	04.10.2016	13:45-15:15	TBA	Retry Exam	

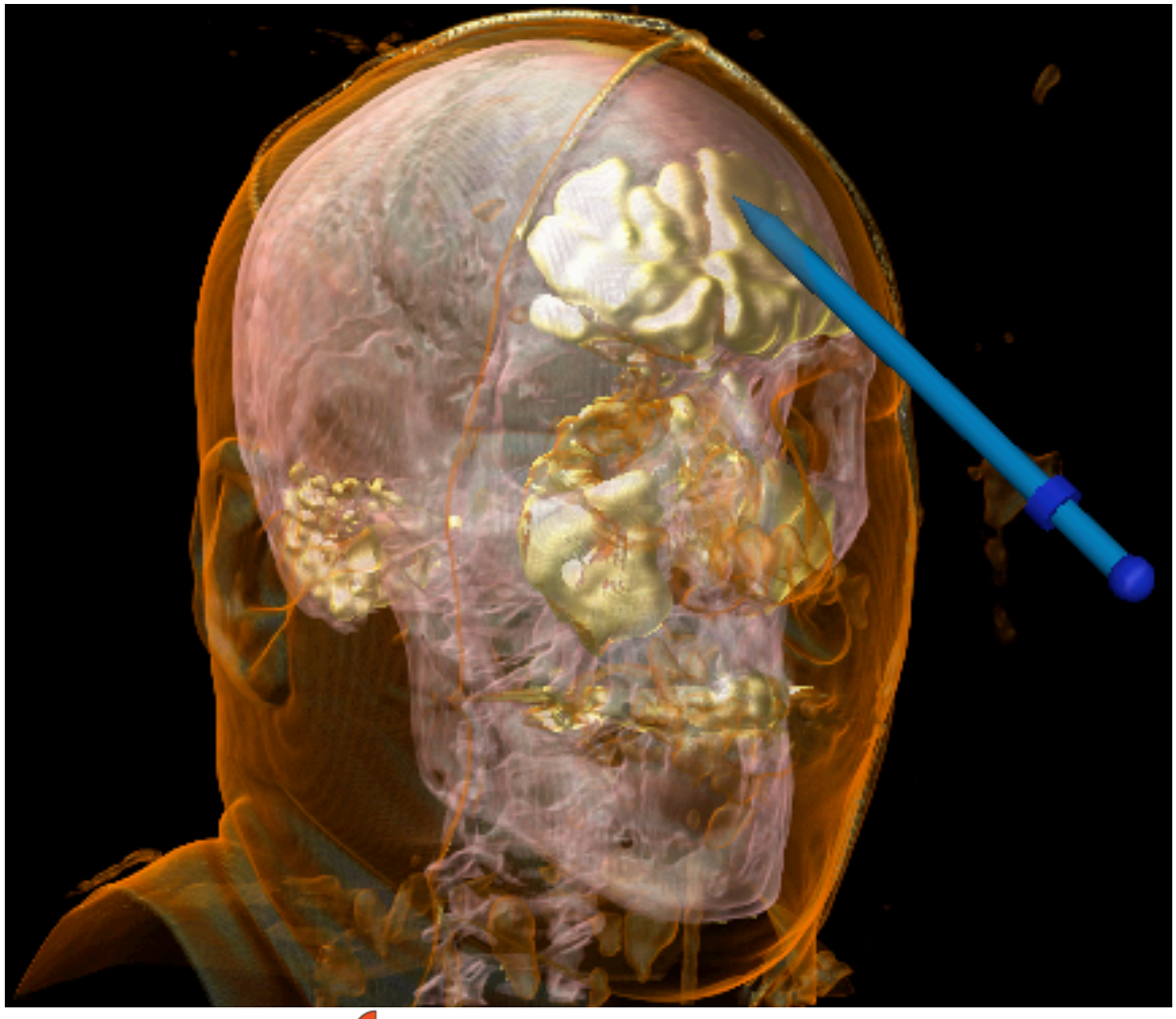
Volume Rendering



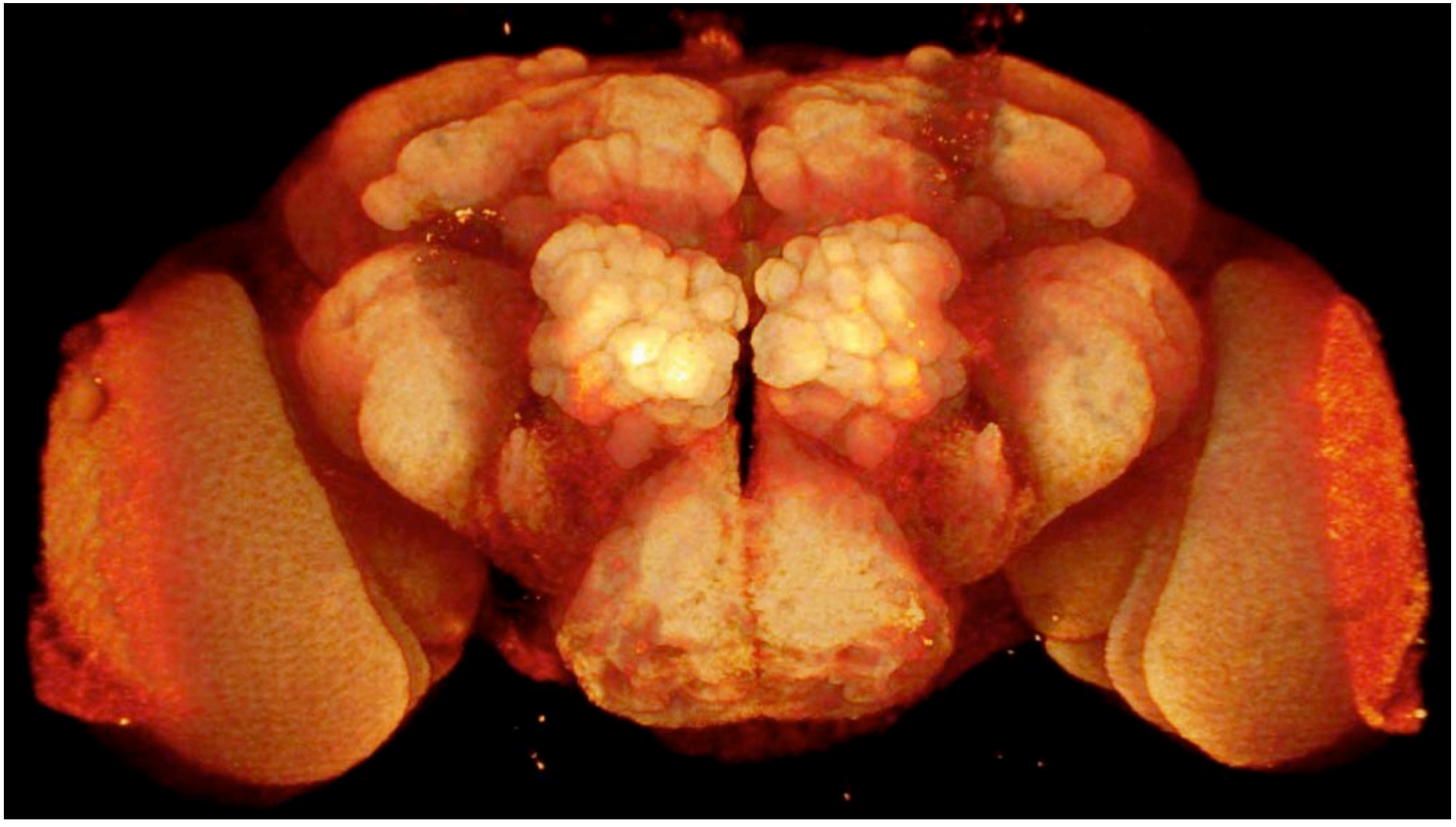
Volume Rendering



Volume Rendering

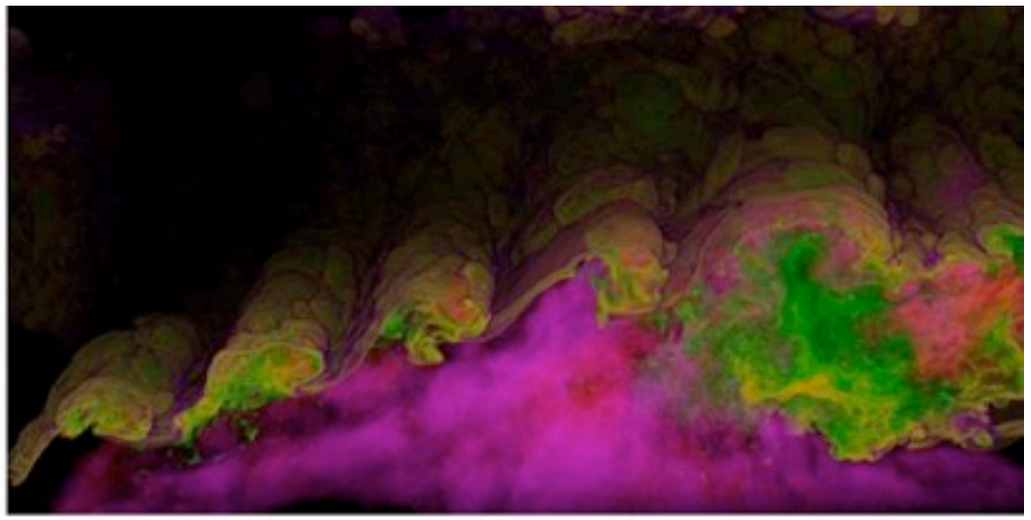


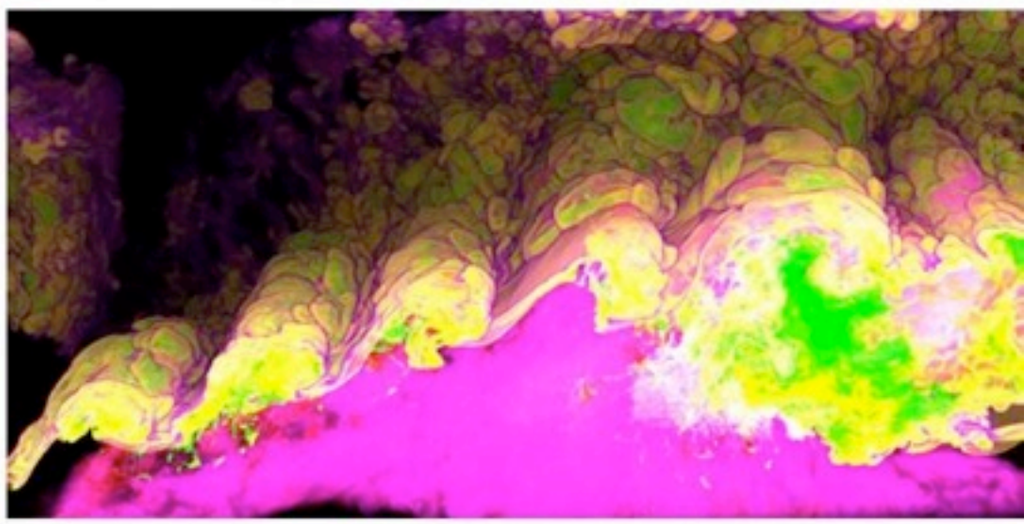
Volume Rendering

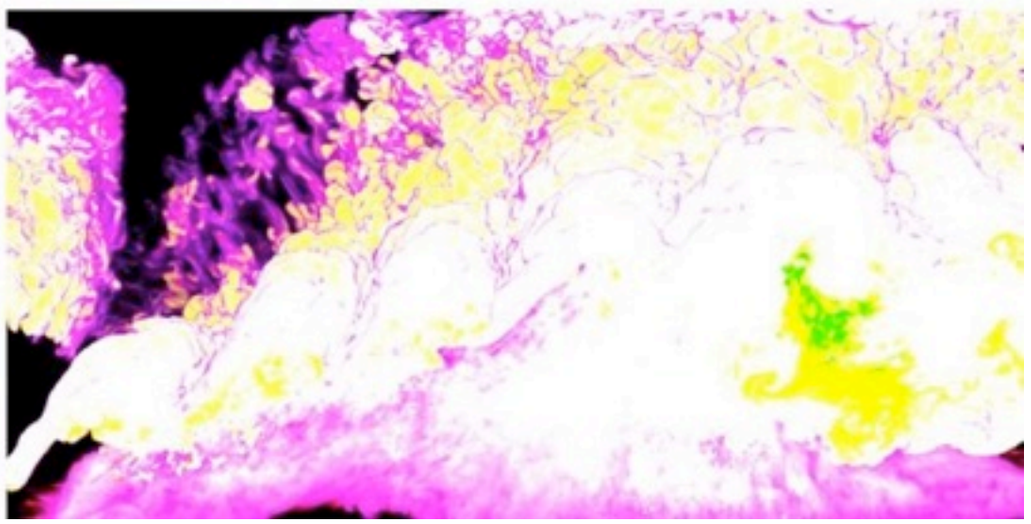


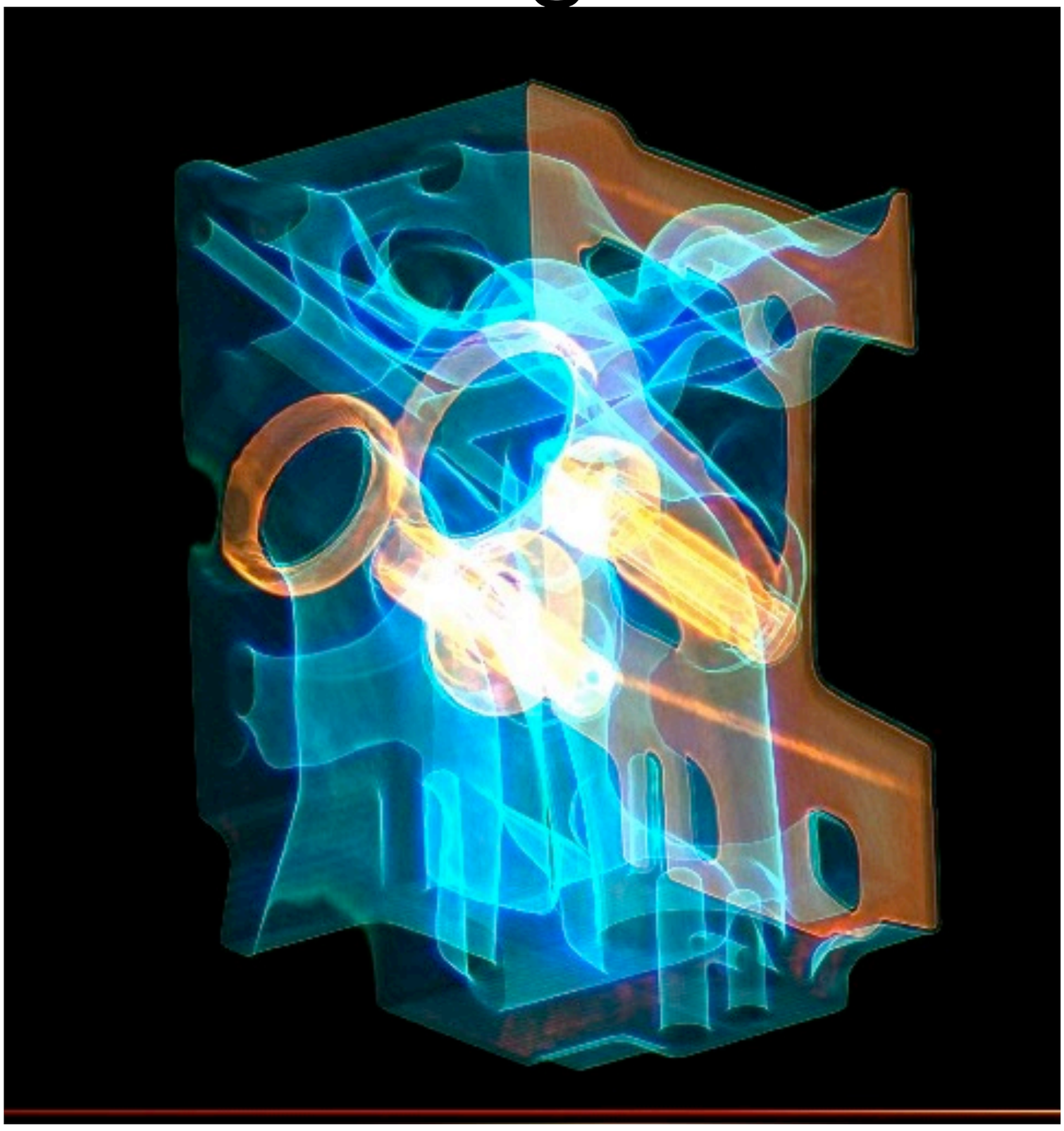
Volume Rendering

Volume Rendering

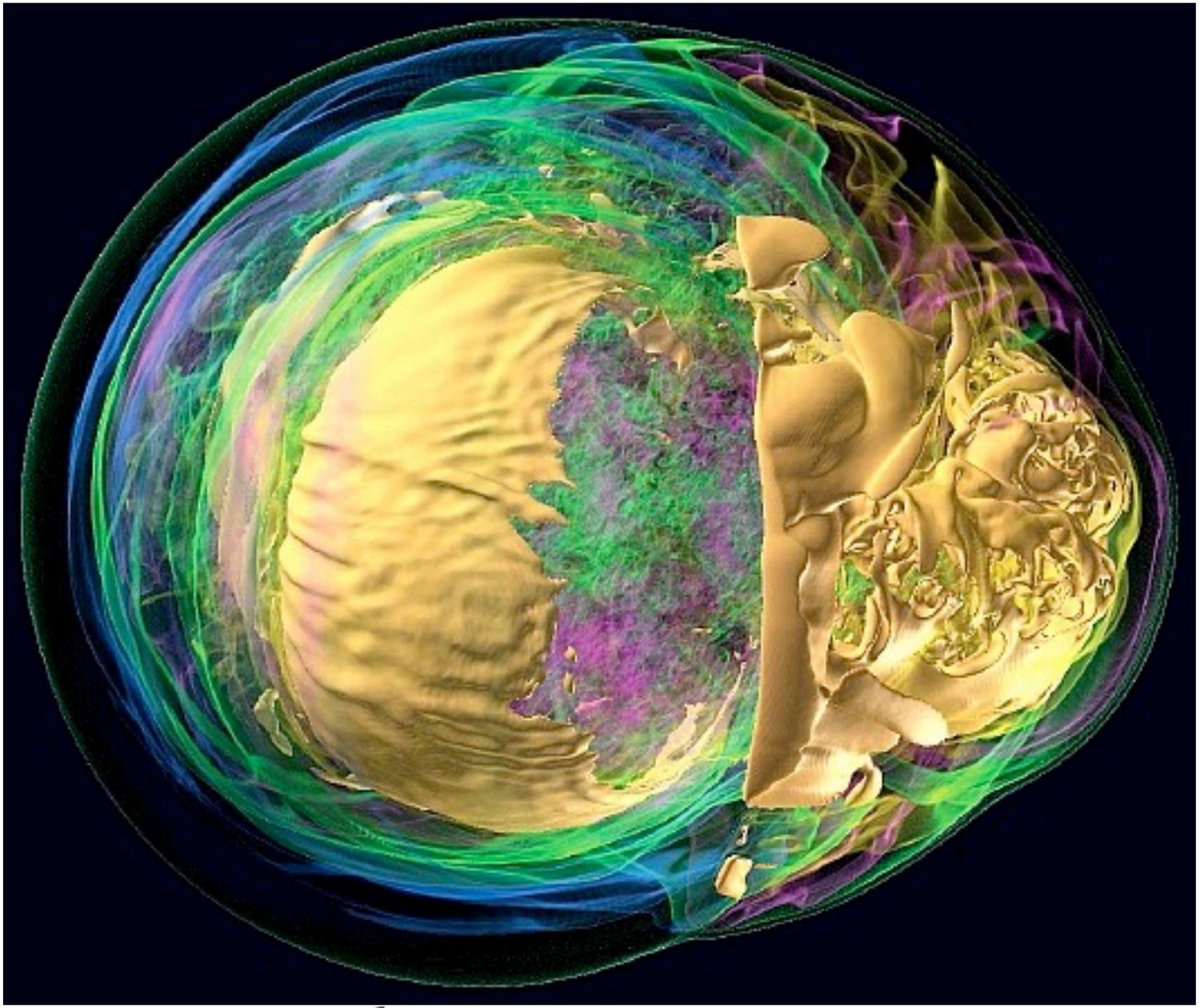






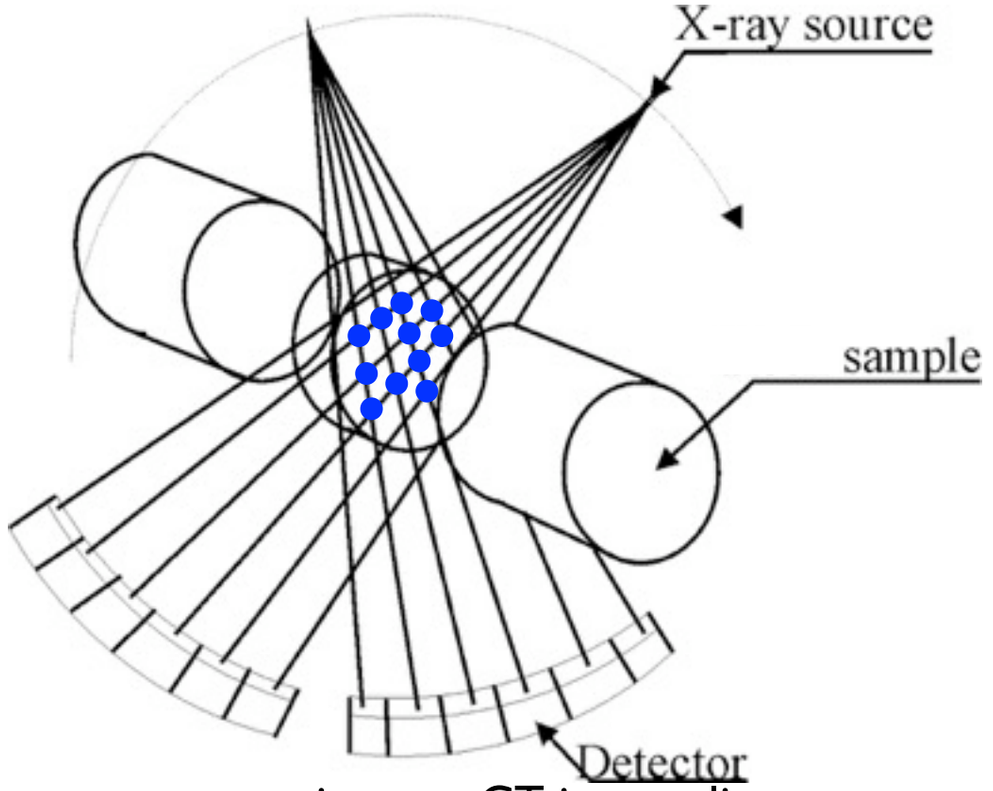


Volume Rendering

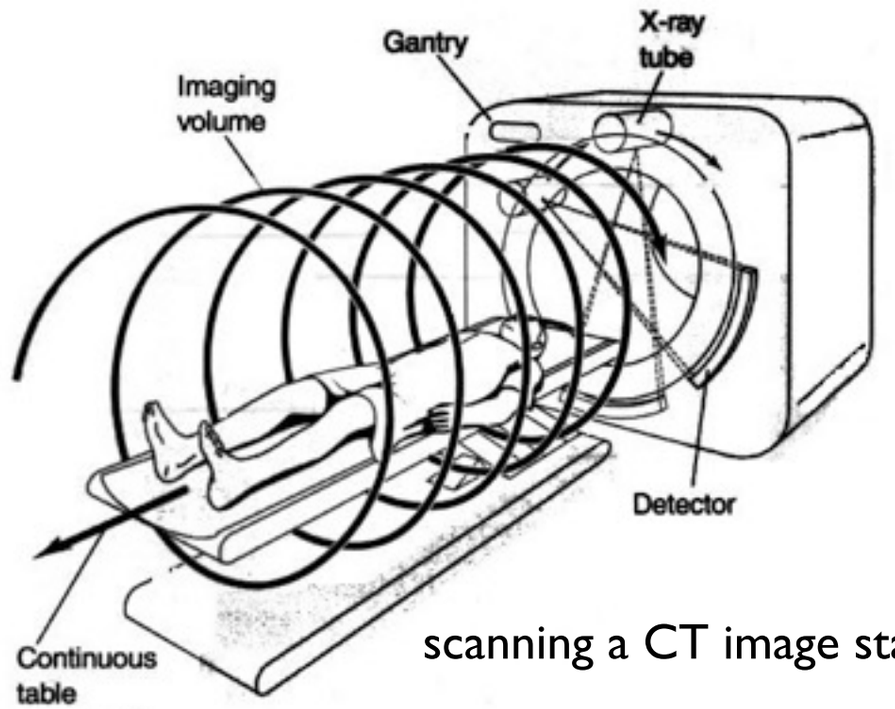


Example: Computer Tomography

- A classical application for volume rendering is computer tomography (CT)
- X-rays are fired from a source in a transverse plane, and are detected by detectors
- The detectors measure the amount of absorption through the sample
- Rotating the X-ray source around the sample, generates many rays (with their corresponding absorption values being 1D projected on the detectors) on the same plane
- From the intersections of the rays and the measured absorptions at the 1D projections, absorption coefficients on the 2D plane can be constructed (this gives one image at the corresponding plane)
- The sample is then moved through the system to measure a series of images (image stack)



scanning one CT image slice



scanning a CT image stack

Generating Volumetric Models

- Volumetric models can be computed (e.g., simulations) or measured (e.g., scanned)
- Imagine a stack of correlated images
- Each image displays a slice (cutting plane) through a volume
- How would you render this volumetric data set?
- You could display it image by image (slice by slice) - but this would limit rendering to a perspective that is co-axial to the slicing direction
- You can think of volume rendering as a form of image-based rendering
- The different image slices can, for example, be rendered with alpha blending and texture mapping - but other possibilities exist

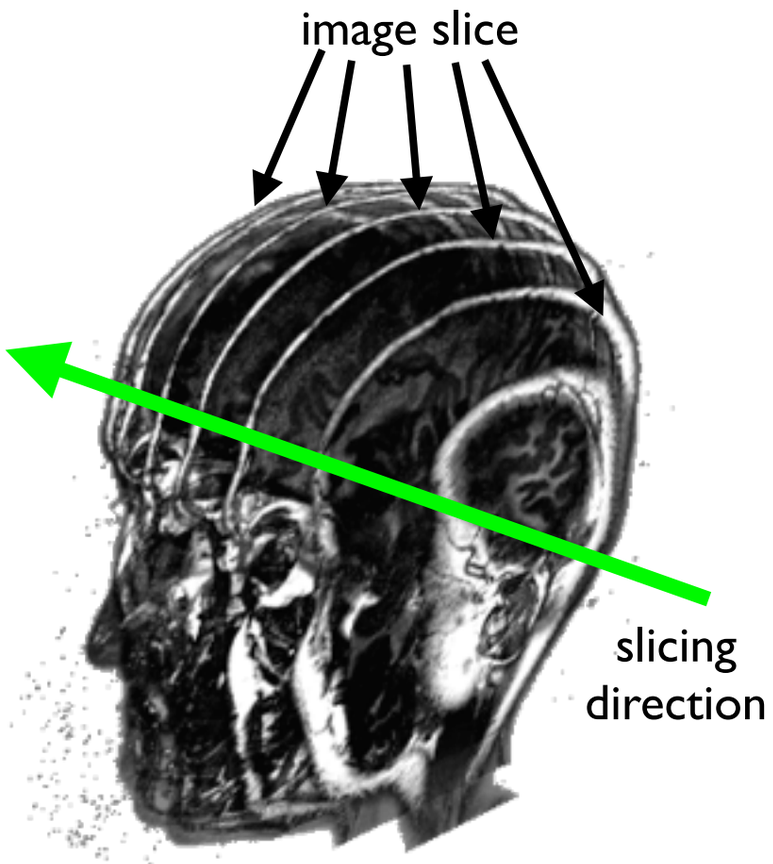


image stack of a volumetric data set


volumetric rendering

Transfer Functions

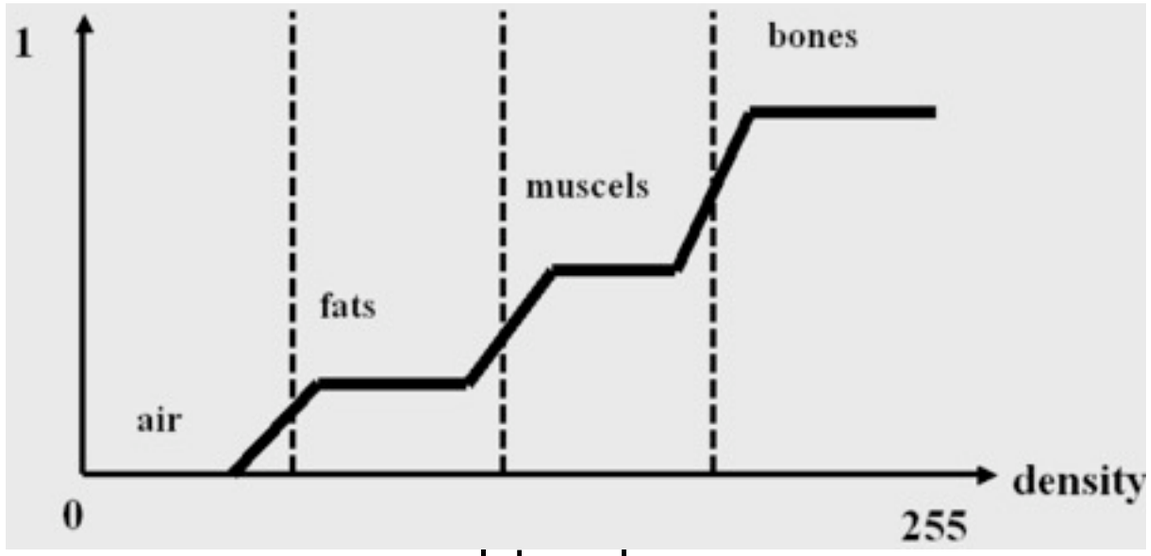
- Before rendering the volume, the opacity values in the volume dataset (which corresponds to density on a CT scan) can be re-mapped

- The function for re-mapping these original values is called transfer function

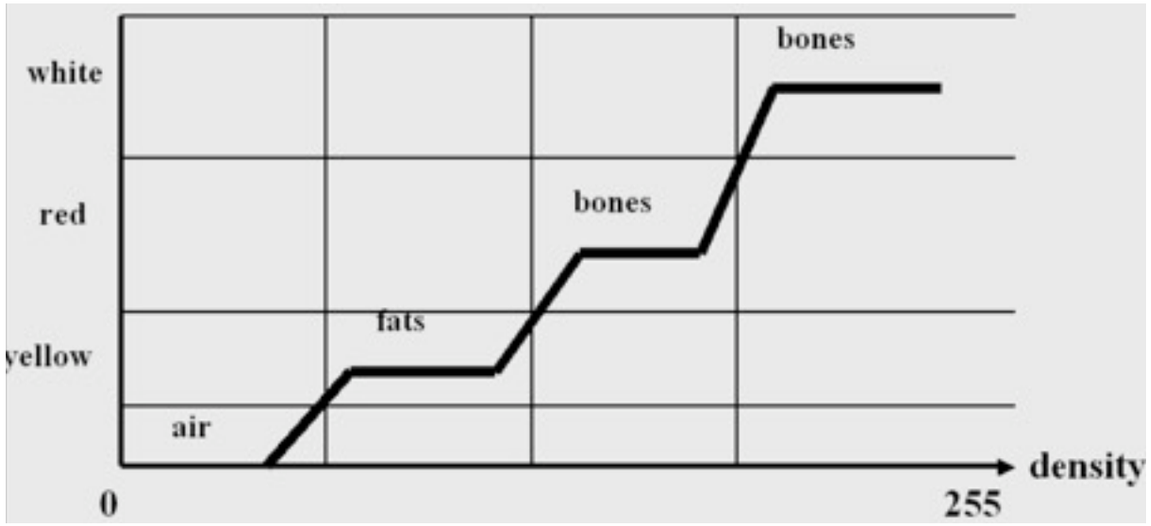
- Transfer functions take (multiple) scalars of the data and map them to RGBA

- For example, color and transparency (alpha) values can be assigned based on density values

- They can be simple look-up tables, or more complicated functions



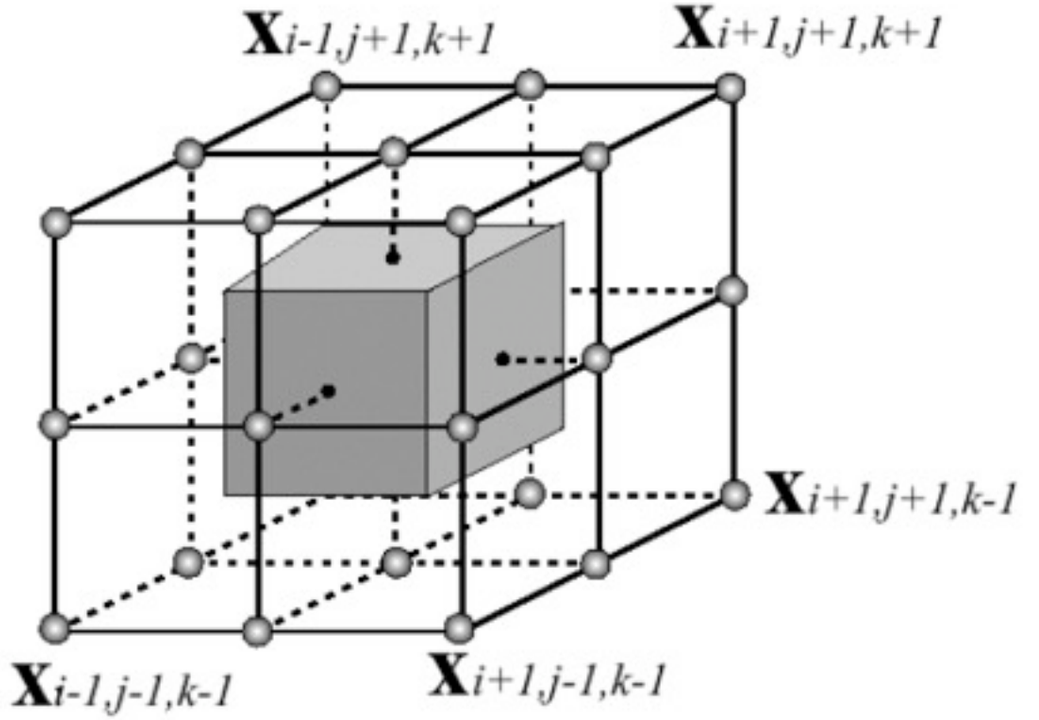
alpha values



color values

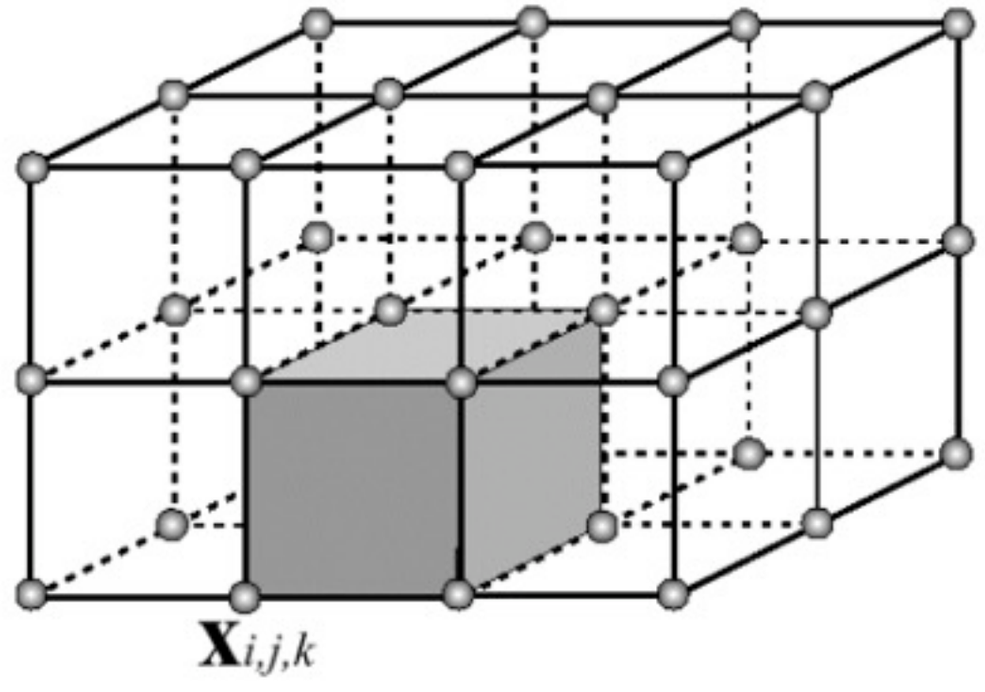
Volumetric Dataset

- Scanning an image stack is discrete (images consist of discrete entries and stack of discrete images - all sampled at fixed resolution and spacing)
- One entry in a volumetric dataset is called voxel
- Voxels are homogeneous and represent the measurement or computations of one discrete sample



a voxel is homogeneous

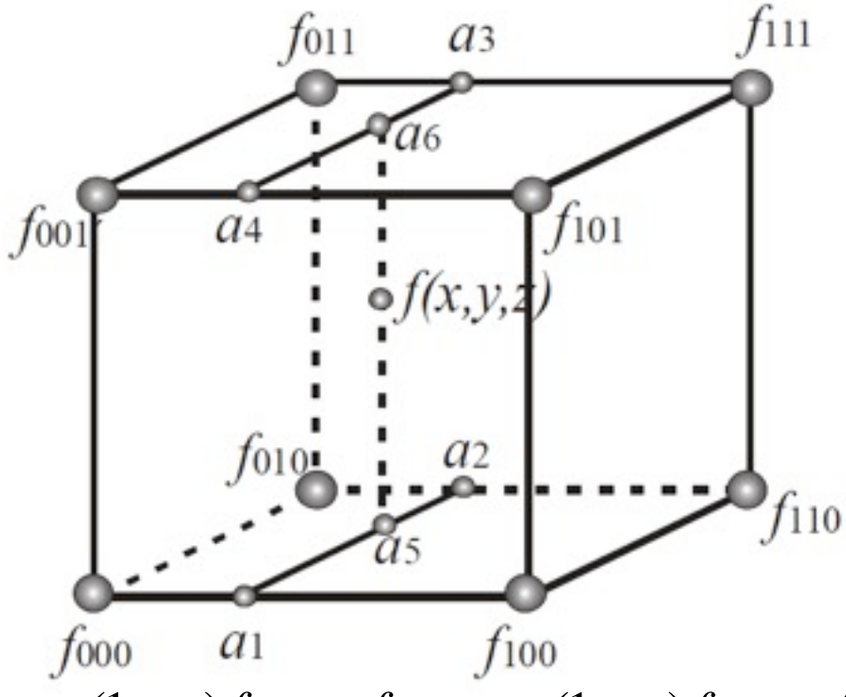
- How to represent intermediate values (in between image entries or image planes)?
- They can be interpolated (trilinear interpolation)
- A cell connects the eight neighboring voxels - thus, they are inhomogeneous
- Intermediate values within a cell can be computed through interpolation



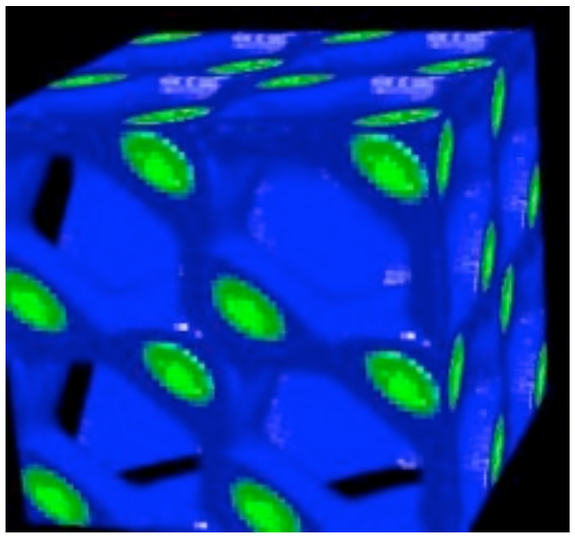
a cell is inhomogeneous

Trilinear Interpolation

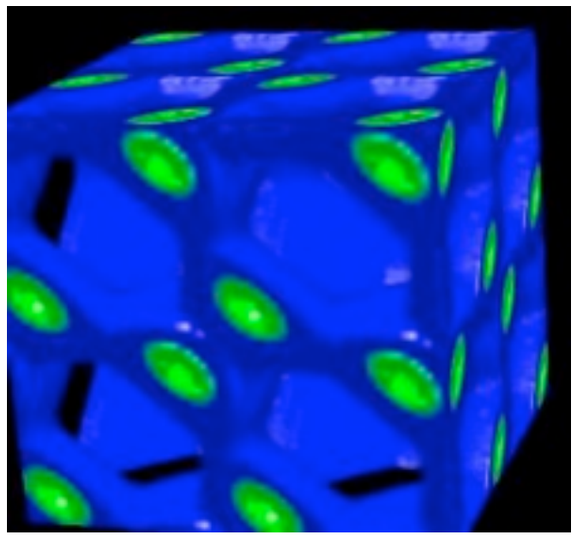
- We have discussed interpolation in triangles already
- Instead of using the voxel values directly (which would equal a nearest neighbour selection for intermediate volume positions), trilinear interpolation leads to smoother results and less aliasing
- Intermediate values are interpolated from the eight voxel vales at the corners of each cell
- Note, that each cell is assumed to be a unit-cell (i.e., x,y,z are normalized to 0..1)
- The voxel values at each corner are indicated with f in this example



$$a_1 = (1-x)f_{000} + xf_{100}, a_2 = (1-x)f_{010} + xf_{110}$$
$$a_3 = (1-x)f_{011} + xf_{111}, a_4 = (1-x)f_{001} + xf_{101}$$
$$a_5 = (1-y)a_1 + ya_2, a_6 = (1-y)a_4 + ya_3$$
$$f(x,y,z) = (1-z) * a_5 + za_6$$



nearest neighbour



trilinear interpolation

Recap: Alpha Blending

- Besides the three color channels (RGB) an additional channel is usually supported by common rendering pipeline (actually, there are several more as we will see later)

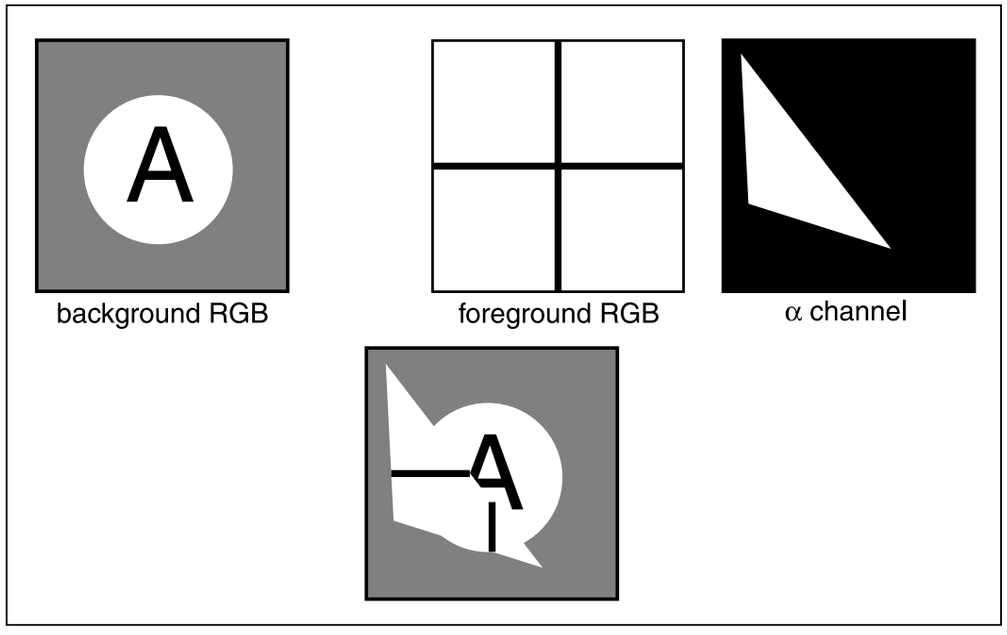
- It is used for blending together differently rasterized portions (i.e., allows to simulate simple transparency effects)

- It is called alpha channel and stors one normalized weight (alpha) per pixel

- For example, to blend the overlapping region of an opaque background with a semi-transparent foreground, their colors are weighted and added during rasterization

- There are several different blending functions

- If alpha is 0 or 1, we simply stencil our portions instead of blending them



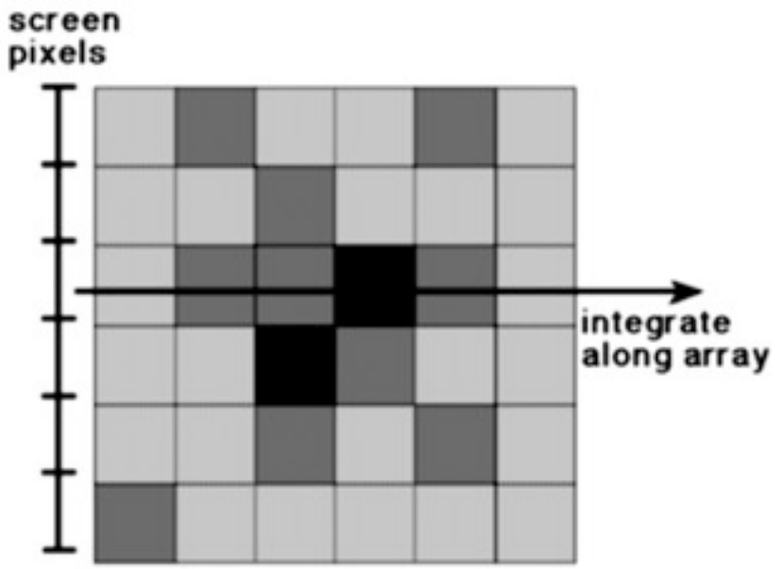
$$c = \alpha c_f + (1 - \alpha) c_b$$

c_f is the color of the foreground and
c_b is the color of the background

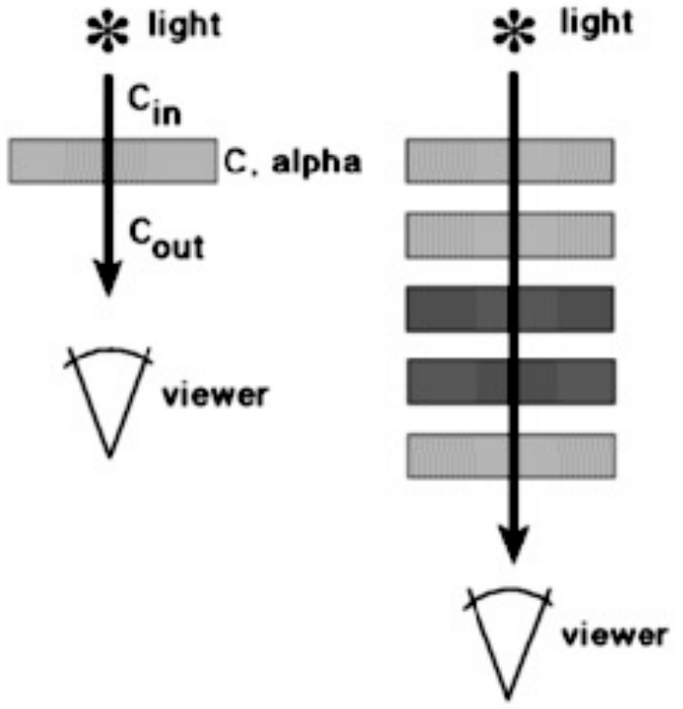
...more cool graphics (with alpha blending)

Image-Order Rendering

- Ray-casting can be used for rendering volumetric datasets
- Since ray-casting is a pixel-by-pixel operation, this class is called image-order rendering, image/pixel space traversal, or back projection
- A ray is cast through the volume for every pixel in the image plane / screen
- By traversing the volume, the different absorption coefficients (alpha) and colors (C) on the ray are accumulated
- In fact, the rays are traverse from back to front (i.e., from behind the volume towards the image plane)
- This enables a iterative accumulation of absorption coefficients and color
- Initially, alpha and C can both be 1



example for ray-casting principle

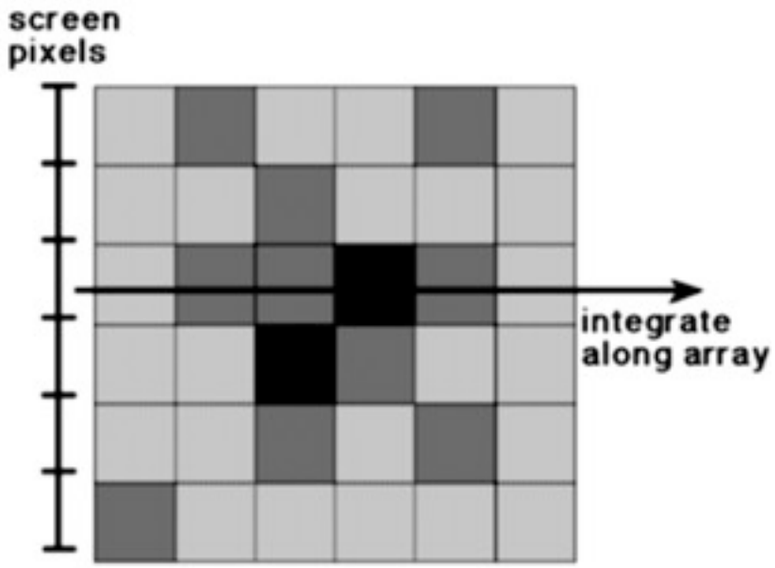


$C(i)_{out} = C(i)_{in} * (1\text{-alpha}(i)) + C(i) * \text{alpha}(i)$

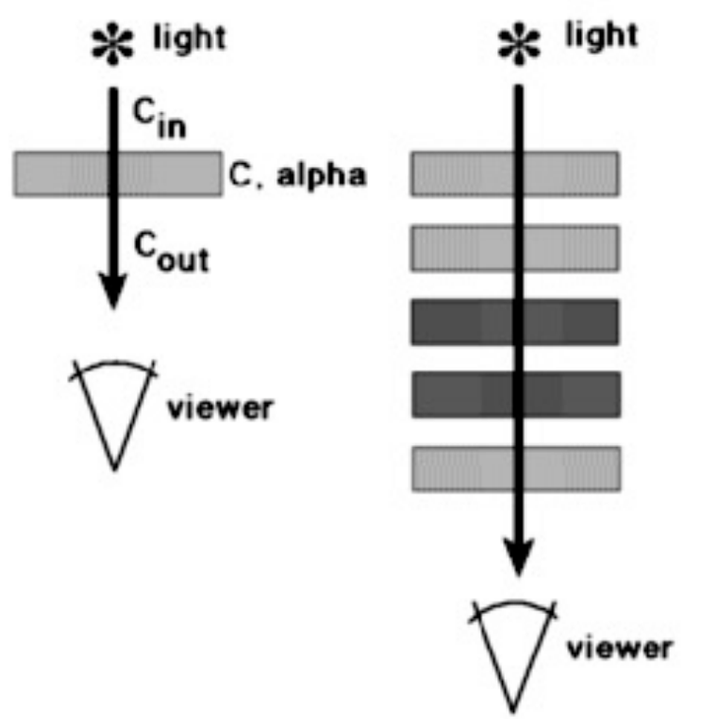
iterative accumulation of values during traversal

Image-Order Rendering

- This equals an orthographic projection of the volumetric dataset onto the image plane
- The difference to ray-tracing is, that in ray-casting, only one ray is traced in one direction
- Before ray-casting samples the data, the volume can be transformed to be aligned to the image plane with the desired perspective
- This can be done in two ways:
  - warp the entire voxel grid in 3D (called volume transform)
  - inversely transform rays while leaving the volume untransformed (called ray transform)



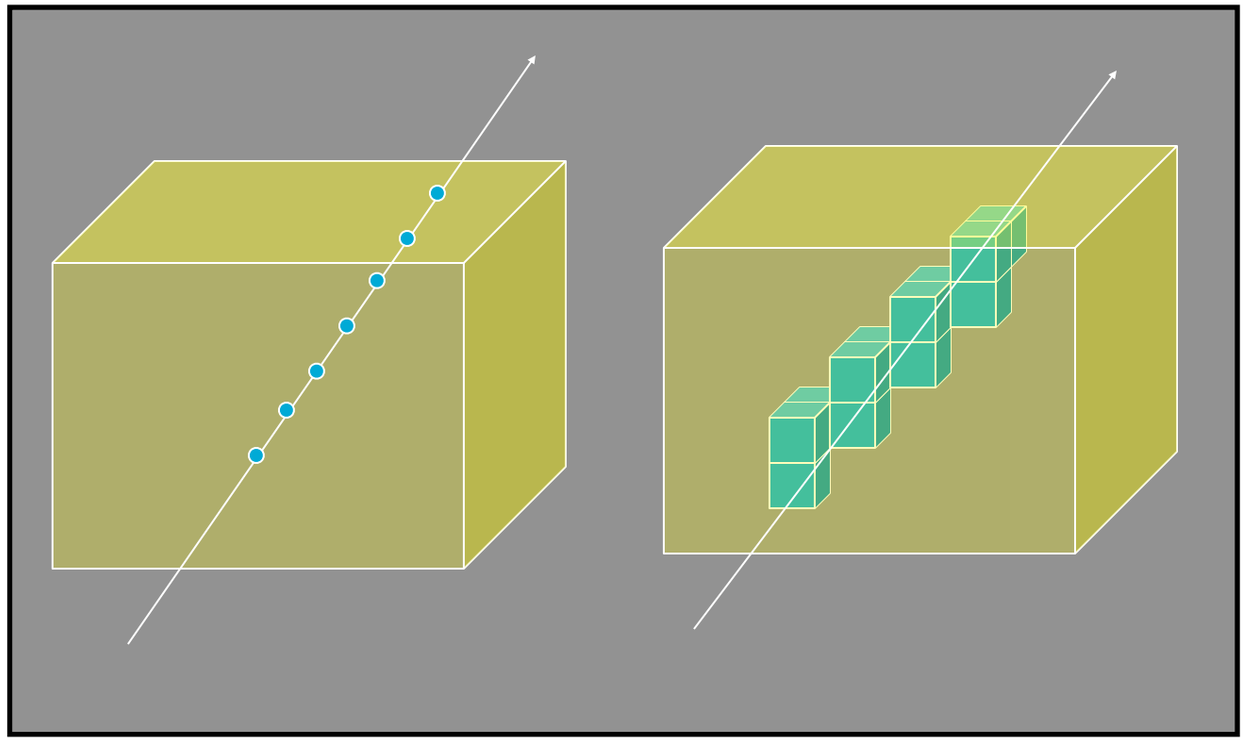
example for ray-casting principle



iterative accumulation of values during traversal

Ray Transform

- If the volume remains untransformed, then each inversely transformed ray has to be traversed through the volume grid
- Each can be sampled with a uniform distance
- The corresponding cell is determined for each ray point
- The parameters (C, alpha) for each ray point are computed within the corresponding cell via trilinear interpolation
- The parameters for all ray points are accumulated as described earlier
- Resampling and rendering steps are merged
- Since intersections of rays in cells are different, the interpolation computations have to be carried out for each ray in each cell
- The step size of ray points have to be chosen appropriate (if too large - small details are missed, if too short - rendering is too slow)
- Since ray-cell intersections are different, a constant step-width might not be optimal
- Appropriate 3D scan conversion techniques should be used that balance speed vs. quality

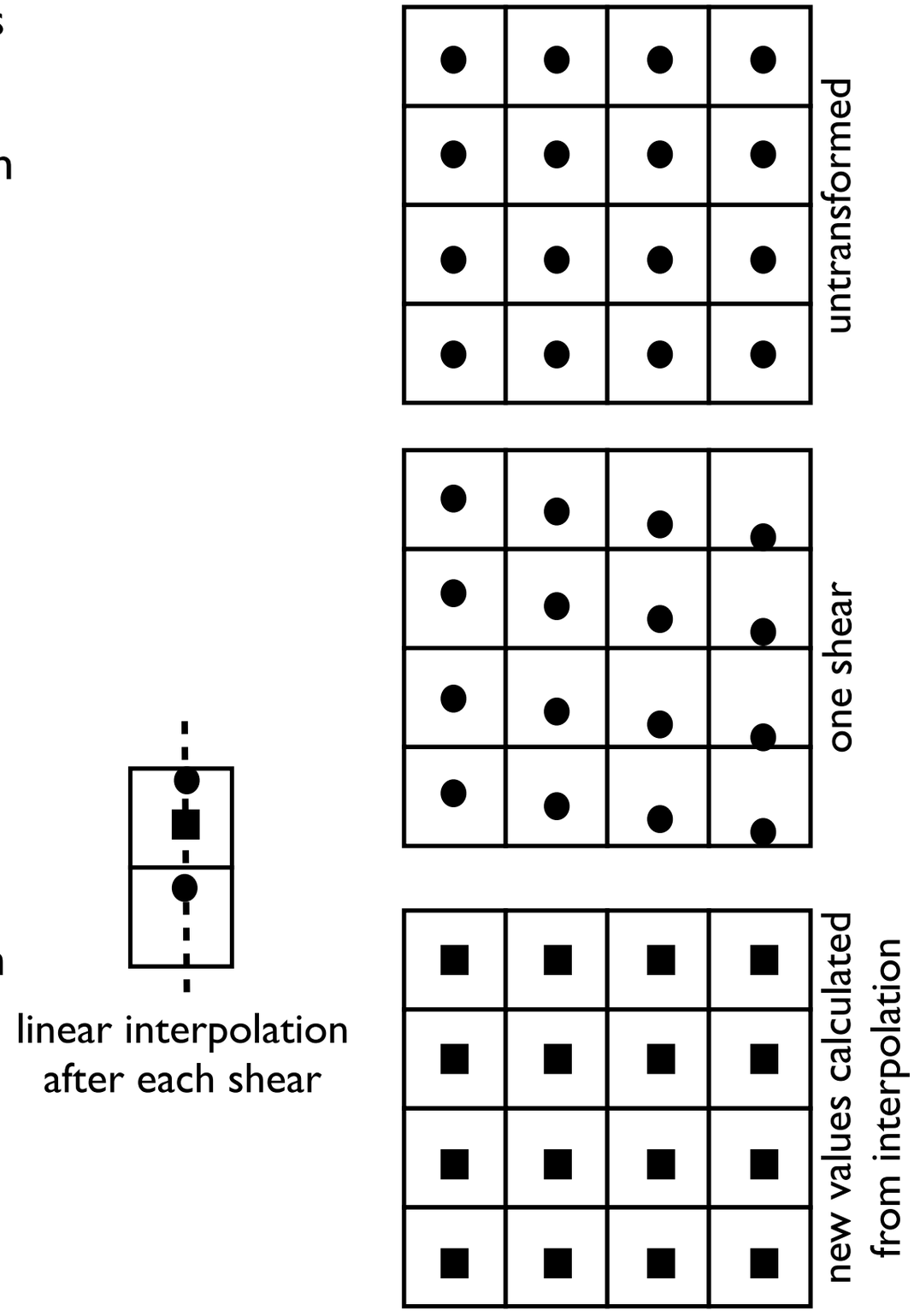


sampling a ray through the volume

results of increasing (from left to right) sampling resolutions (i.e., decreasing step-with on ray)

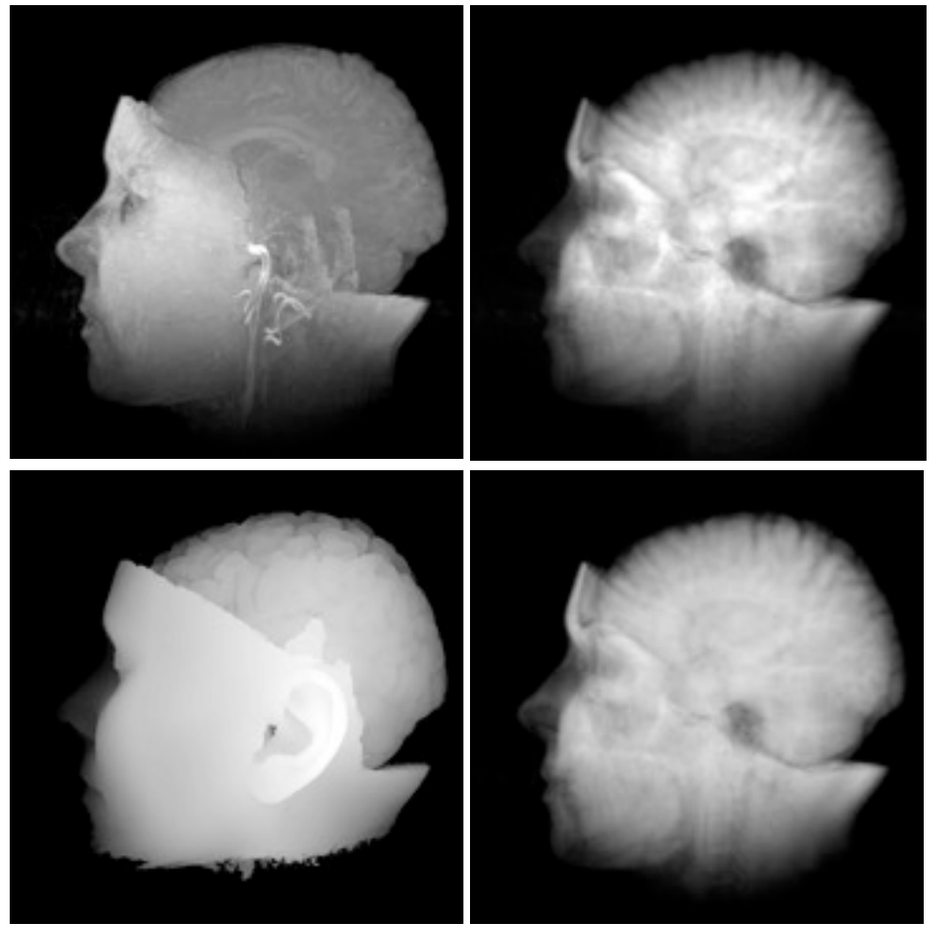
Volume Transform

- In this case, the volume data set is pre-transformed and the un-transformed rays are cast
- The 3D transformation of the volume can be decomposed in a set (actually nine - three for each axis) of shear transformations (this is efficient to be implemented on graphics hardware)
- Resampling of the volume data has to be performed after each shear transformation (before rendering)
  - the basic cell grid through which rays are cast remains untransformed
  - transforming the volume means transforming the voxels
  - the voxels might not be centered in the cell grid after transformation
  - new voxel values for the grid centers have to be interpolated
- Constant step width along rays can be adjusted better

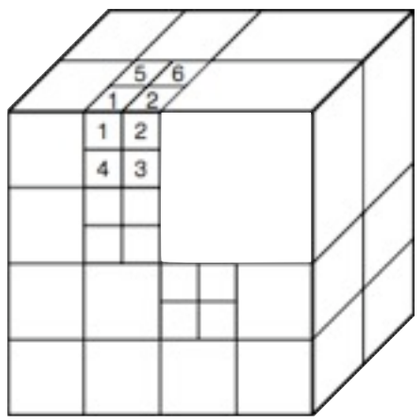


Acceleration and Storage Techniques

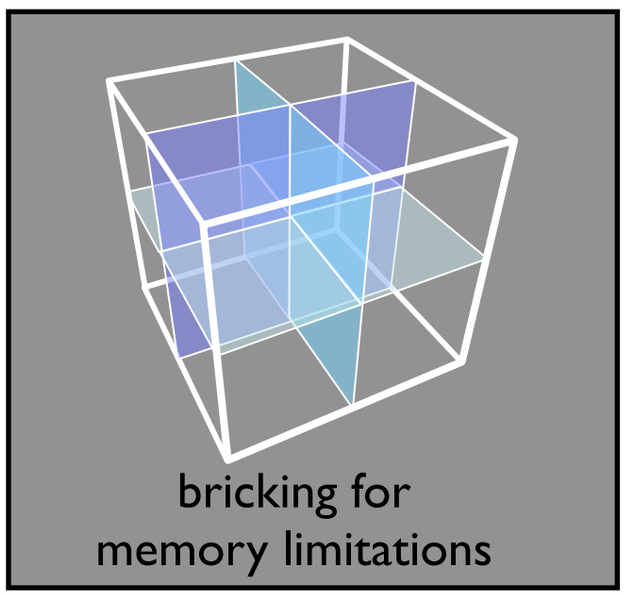
- When traversing rays, different functions can be used to compose the resulting pixel value
- Examples are average, maximum (maximum intensity projection, MIP), distance, or alpha compositing
- The opacity of a point in the ray is given by its corresponding alpha value (alpha=0.3 means 30% opacity)
- Early stopping: ray traversal can be stopped, when enough opacity is accumulated (e.g., when alpha = 1)
- Empty space (i.e., space with an alpha less than a given threshold) does not contribute to the final result
- We don‘t have to sample the ray in these areas
- Space-skipping: use space partitioning, such as octree to cluster areas with similar opacity
- Bricking: if whole volume data does not fit in memory, subdivide into smaller chunks (bricks) and render sequentially



different composition functions



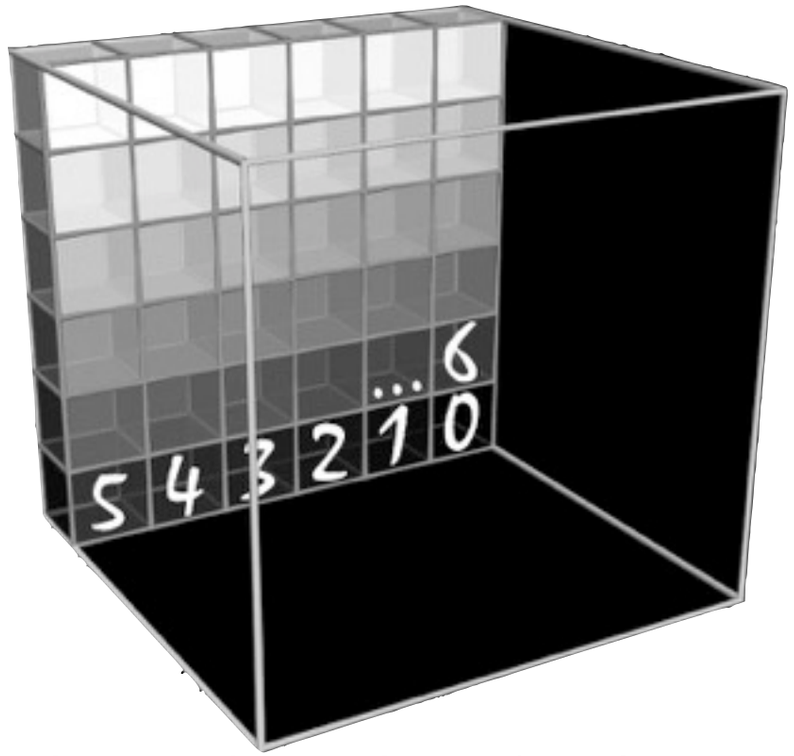
octree used for space-skipping



bricking for memory limitations

Object-Order Rendering

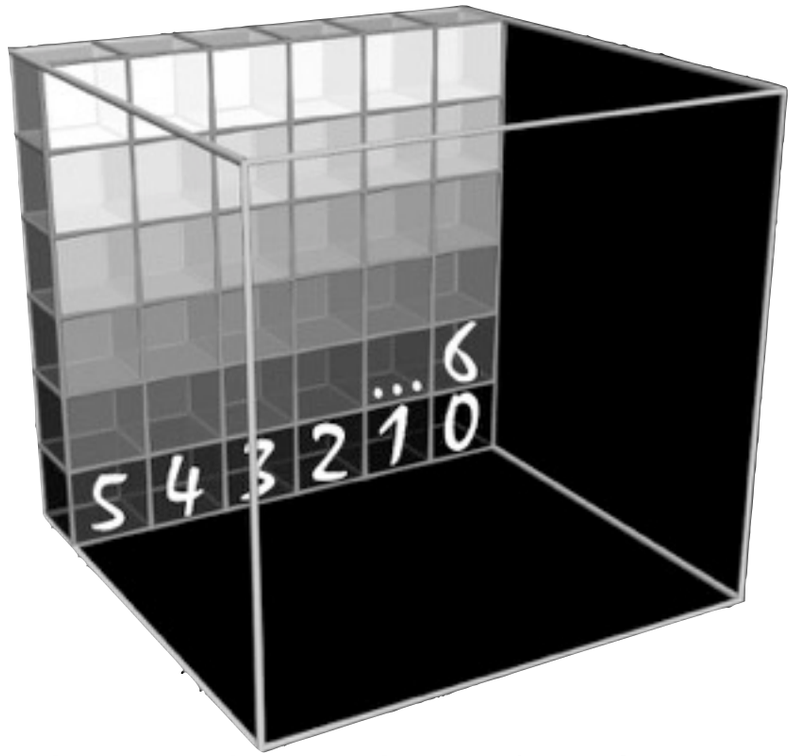
- Object-order rendering is also called voxel projection, object/voxel space traversal, or forward projection
- In this case, voxel planes are traversed, and are projected to image plane
- The frame buffer is accumulated
- We can traverse from back-to-front or front-to-back
  - front-to-back is fast, since we can stop when enough opacity is accumulated (e.g. alpha=1)
  - back-to-front can show all details (sequentially) - even if hidden in final stage
- One point at the center of a particular voxel projects to a single pixel (this might cause holes on the image plane)



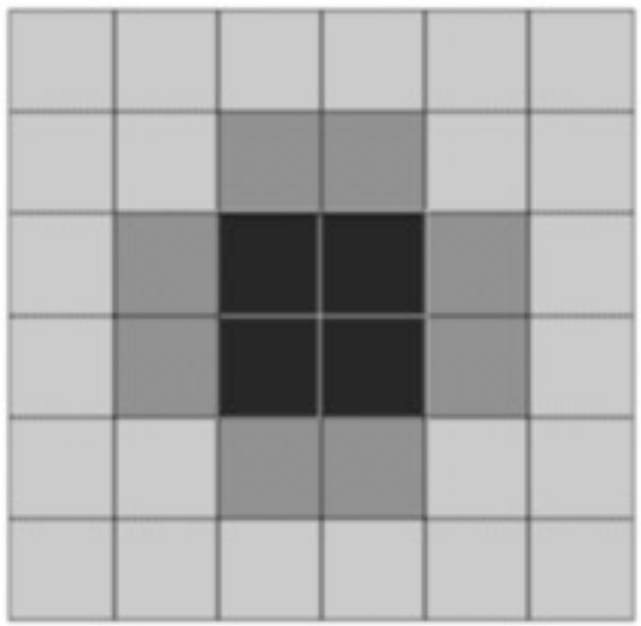
object-order rendering (voxel projection) in back-to-front order

Object-Order Rendering

- To determine the filled pixel values on the image plane, a 3D spherical region surrounding the sampled voxel can be filtered to consider the contribution of neighbours
- If the 3D filter function is 3D Gaussian, then this projects to a circular function
- Thus, this is equivalent to taking the sample voxel and spread this over a number of pixels on the image plane - this is called splatting
- Consequently, we can project and filter the data by taking the voxel value and splatting it into the image plane by convolving it with the filter weights (2D Gaussian) and accumulating these values
- The 2D filter is called footprint of a voxel
- For orthogonal projection, each voxel has the same footprint - thus it can be pre-computed and stored in a look-up-table



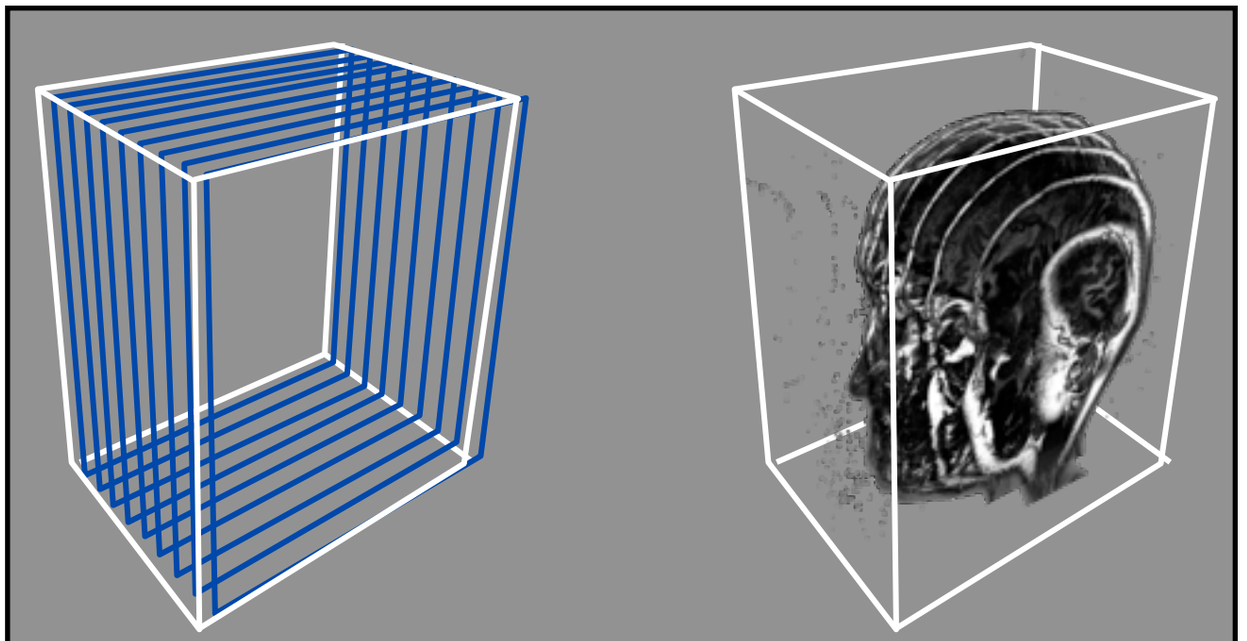
object-order rendering (voxel projection) in back-to-front order



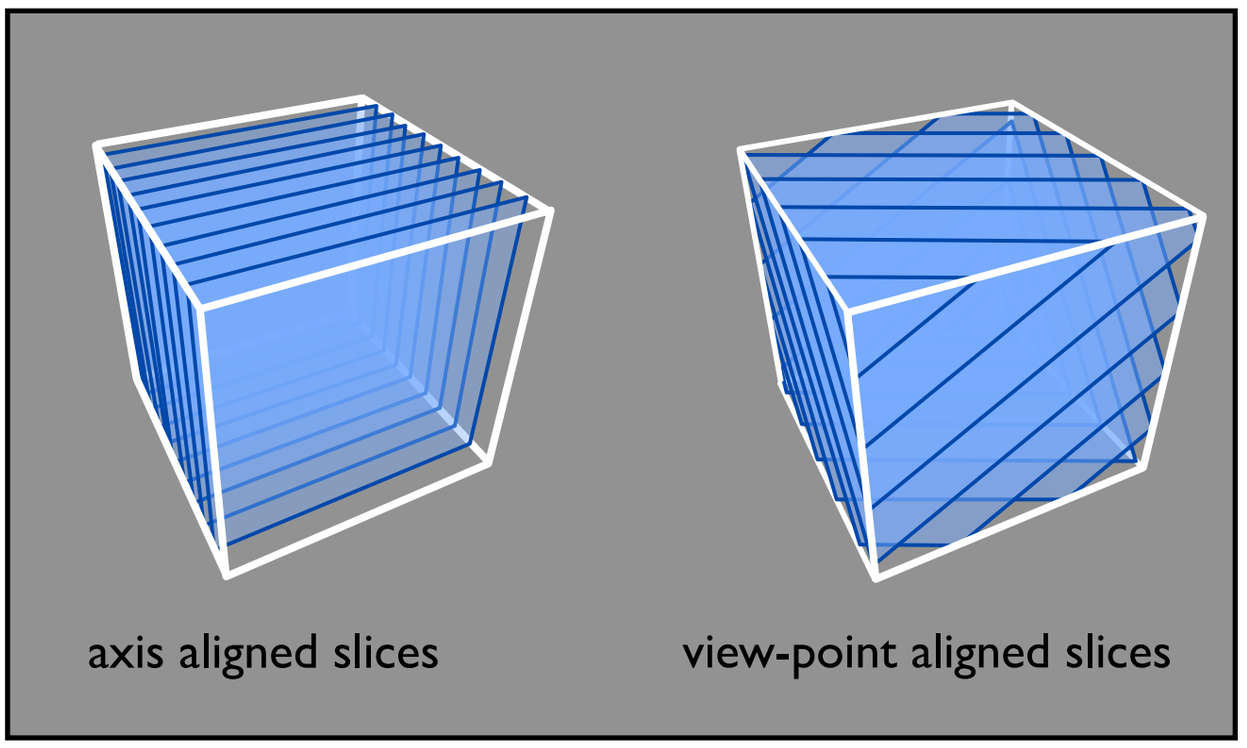
footprint of weights (close up of splat)

Texture-Based Rendering

- Three-dimensional textures (volume textures) can be used for volume rendering (see texture mapping)
- Three-dimensional texture-mapping is hardware accelerated
- A set of parallel polygons that are texture mapped with the 3D texture (assigne corresponding 3D texture coordinates polygon vertices)
- The polygons are rendered in back-to-front order
- Alpha texture mapping is required for transparency
- Polygons can be axis aligned or view-point aligned (constant sampling - but has to be re-mapped constantly)



parallel polygons are texture-mapped with 3D textures



axis aligned slices

view-point aligned slices

Surfaces in Volumes

- If we assume that a voxel contains a part of a surface, the question is how to extract and render it

- If surface information (e.g., normal vector, reflectance) can be recovered, regular shading models can be used for rendering these surfaces

- The computed shading components can be used for C(i) during the compositing operation (discussed earlier)

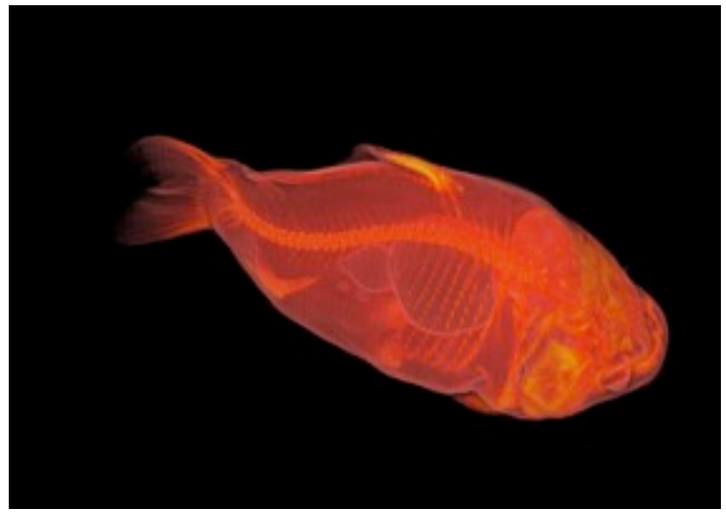
- One possibility is to compute the 3D gradient (equals surface normal N) within local neighbourhoods of the volume data

- If the material within the neighborhood is homogeneous (INI=0), then the voxel at x,y,z does not contain a surface fragment

- Thus, N/INI can be used for shading (e.g., Phong shading) and INI can be used for weighing the contribution of the shading component

$$N_x = R(x+1,y,z) - R(x-1,y,z)$$
$$N_y = R(x,y+1,z) - R(x,y-1,z)$$
$$N_z = R(x,y,z+1) - R(x,y,z-1)$$

gradient in local neighbourhood of voxel at x,y,z in volume dataset (R is the reflectance/material times the density)



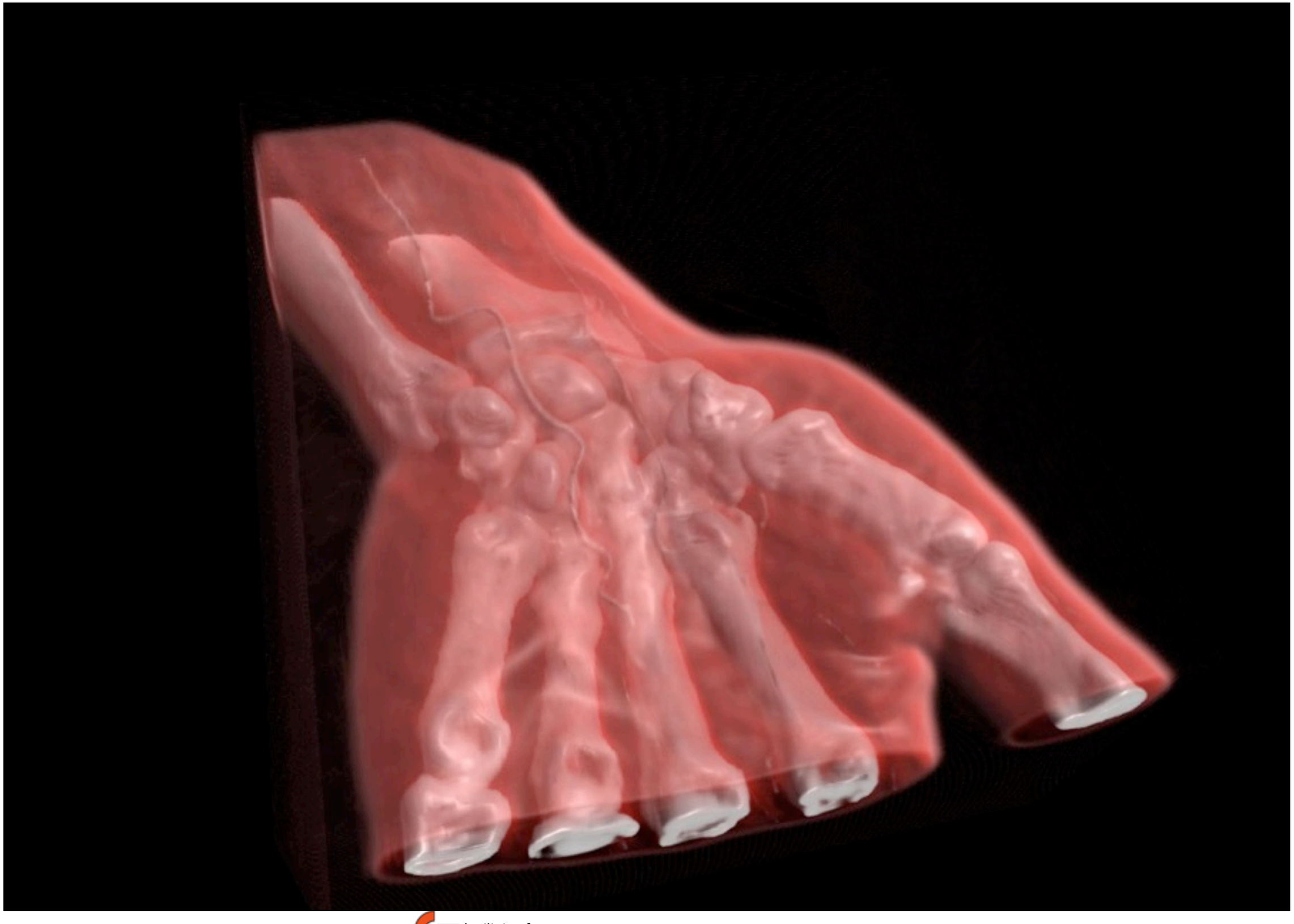
without Phong shading

with Phong shading

Surfaces in Volumes

without Phong shading

Surfaces in Volumes



with Phong shading

Extraction of Isosurfaces

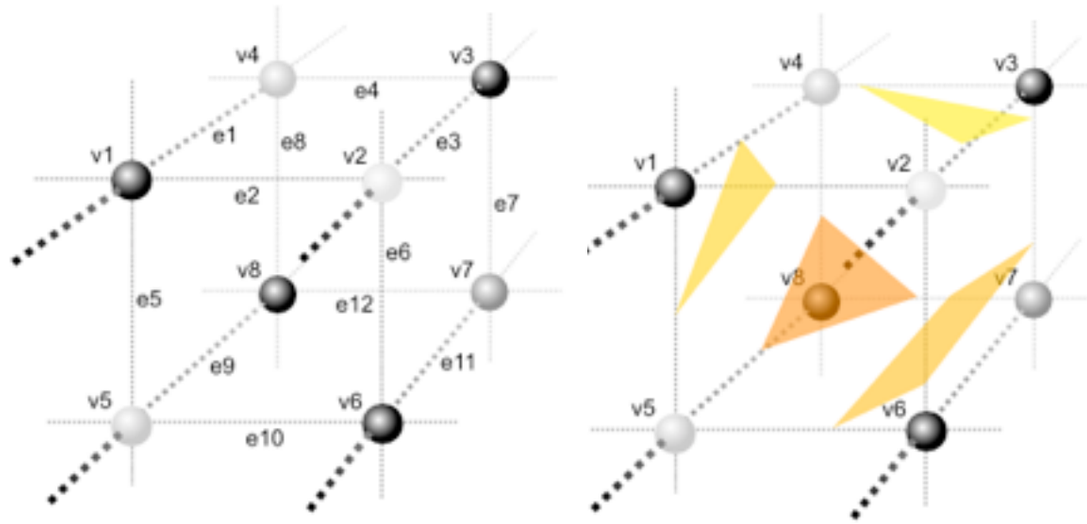
- The problem with using only the computed gradients for shading is, that no occlusions from light source to voxel are considered (the surface is still unknown)

- But we can recover the isosurface from a volumetric dataset with a technique known as marching cubes - the result is a polygon mesh that can be rendered in the usual way

- The volume data set is separated into a grid of cubes (possibly down to cell size) and the algorithm processes one cube after the other (marching cube) to determine how the contained surface intersects the cube

- Thus, we have to fit (a) polygon(s) through each cube that approximate the contained surface boundary

- Which polygon(s) are used and where they are placed inside the cube can be determined from the field values (the corresponding voxel density) at the 8 cube vertices



separating cube corners with polygons that are on one side of a surface from corners that are on the other side (based on the corners field values)

Marching Cubes

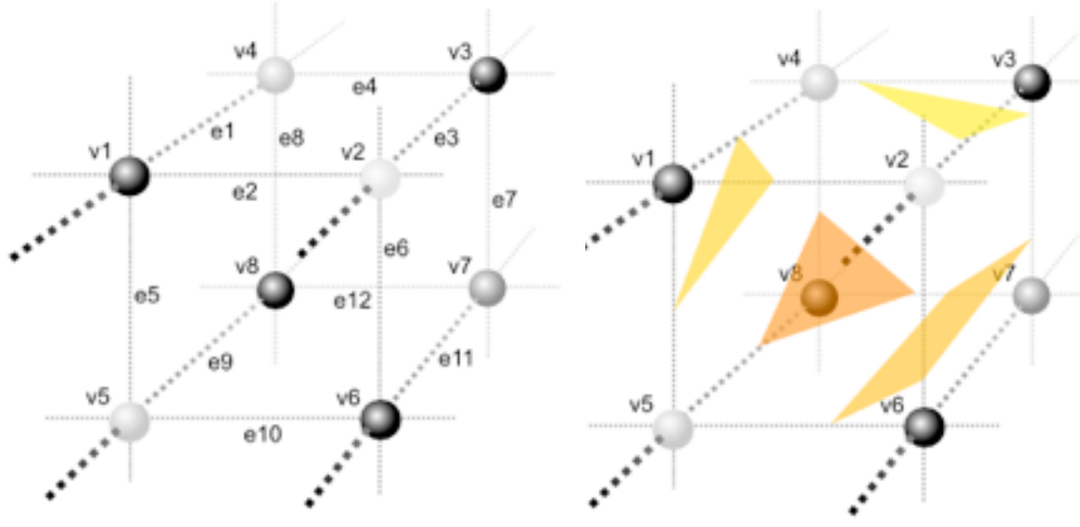
- Depending on a user-defined threshold, corners with a field value above the threshold are separated from corners that are below the threshold (i.e., we separate the corners that are in front of the surface from the ones that are behind the surface)

- The polygons are placed within the cube in such a way these two sets of corners are separated

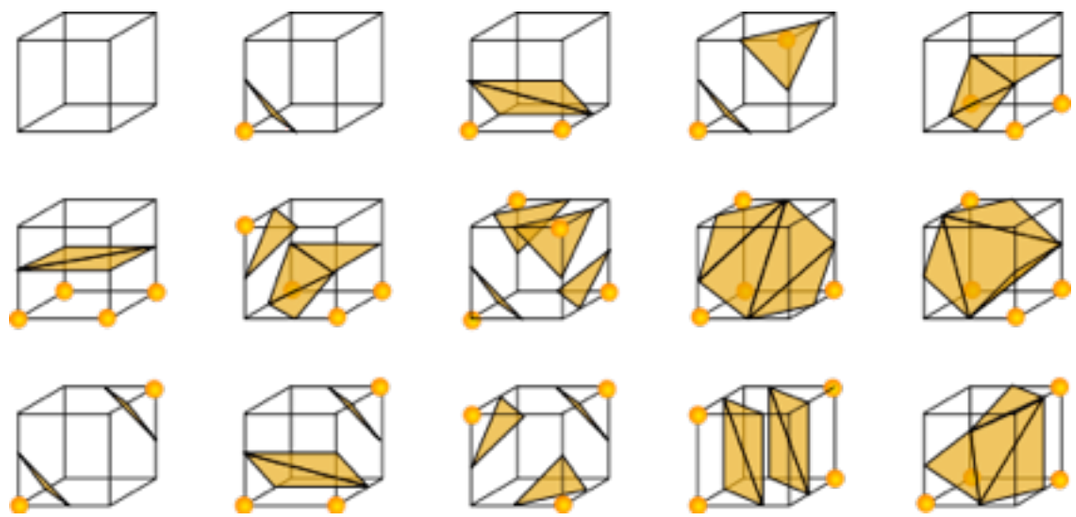
- There are 2^8=256 different separation - but due to symmetry (i.e., if symmetric cases are not considered) then only 15 different possibilites remain

- The exact positions and orientations of the polygons (i.e., their accurate intersections with their cubes) are determined form interpolating between the field values at the corners

- The final polygons represent fragments of the final surface mesh (normals are computed for each fragment in addition)



separating cube corners with polygons that are on one side of a surface from corners that are on the other side (based on the corners field values)

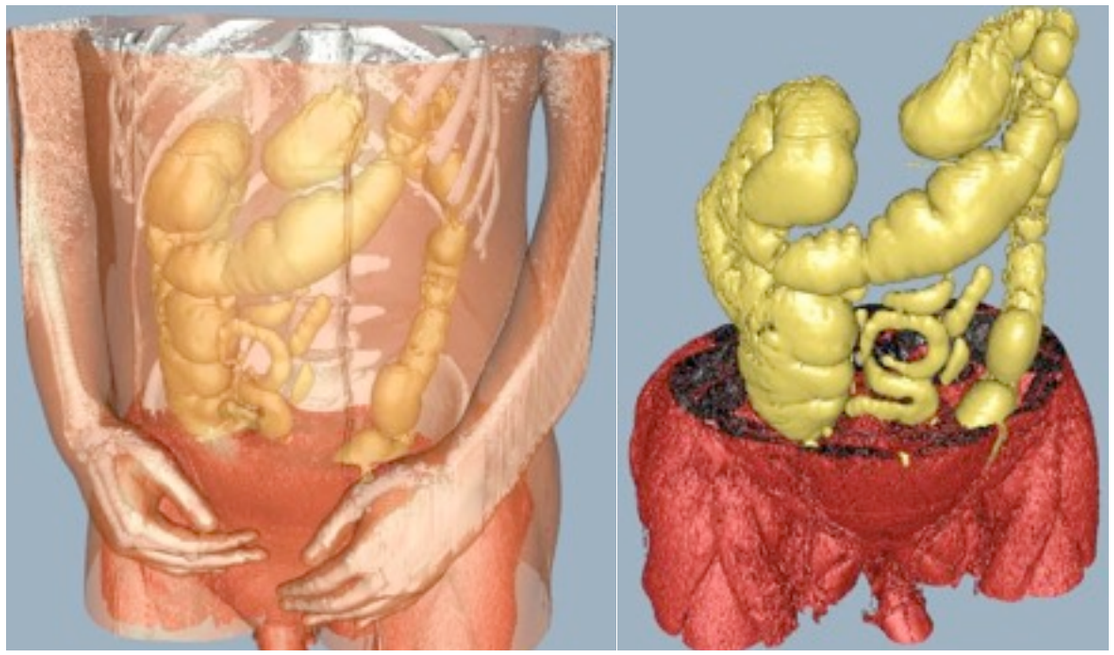


the 15 possibilites in the marching cube algorithm

Marching Cubes

- Depending on a user-defined threshold, corners with a field value above the threshold are separated from corners that are below the threshold (i.e., we separate the corners that are in front of the surface from the ones that are behind the surface)

- The polygons are placed within the cube in such a way these two sets of corners are separated

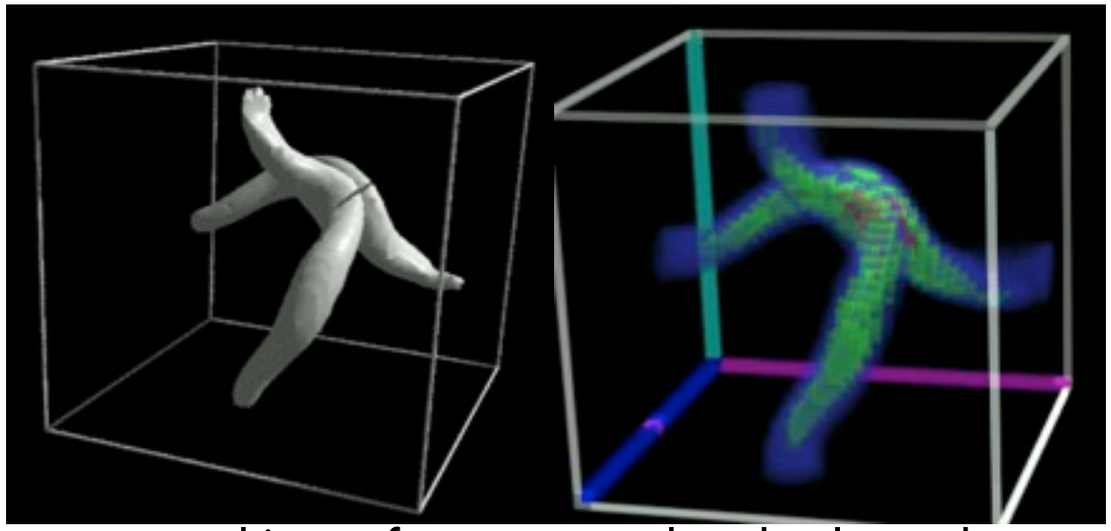


example for marching cube results

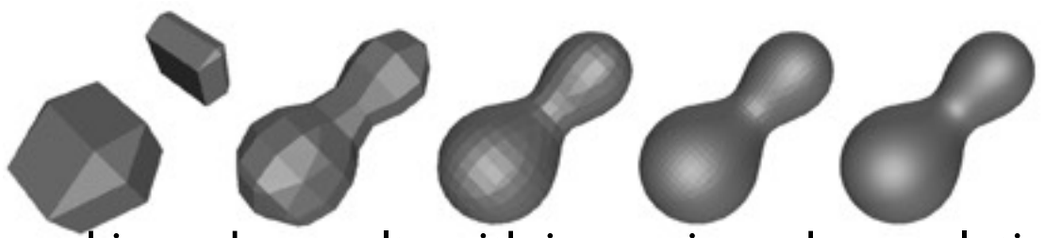
- There are 2^8=256 different separation - but due to symmetry (i.e., if symmetric cases are not considered) then only 15 different possibilites remain

- The exact positions and orientations of the polygons (i.e., their accurate intersections with their cubes) are determined form interpolating between the field values at the corners

- The final polygons represent fragments of the final surface mesh (normals are computed for each fragment in addition)



ray-traced isosurface rendered volume data



marching cube results with increasing cube resolution

Course Schedule

Type	Date	Time	Room	Topic	Comment
C1	01.03.2016	13:45-15:15	HS 18	Introduction and Course Overview	Conference
C2	15.03.2016	13:45-15:15	HS 18	Transformations and Projections	Easter Break
C3	05.04.2016	13:45-15:15	HS 18	Raster Algorithms and Depth Handling	
C4	12.04.2016	13:45-15:15	HS 18	Local Shading and Illumination	
C5	19.04.2016	13:45-15:15	HS 18	Texture Mapping Basics	
C6	26.4.2016	13:45-15:15	HS 18	Advanced Texture Mapping & Graphics Pipelines	
C7	03.05.2016	13:45-15:15	HS 18	Intermediate Exam	
C8	09.05.2016	17:15-18:45	HS 18	Global Illumination I: Raytracing	
C9	10.05.2016	13:45-15:15	HS 18	Global Illumination II: Radiosity	Conference / Holiday
C10	31.05.2016	13:45-15:15	HS 18	Volume Rendering	
C11	07.06.2016	13:45-15:15	HS 18	Scientific Data Visualization	
C12	14.06.2016	13:45-15:15	HS 18	Curves and Surfaces	
C13	21.06.2016	13:45-15:15	HS 18	Basics of Animation	
C14	28.06.2016	13:45-15:15	HS 18	Final Exam	
C15	04.10.2016	13:45-15:15	TBA	Retry Exam	

Thank You!