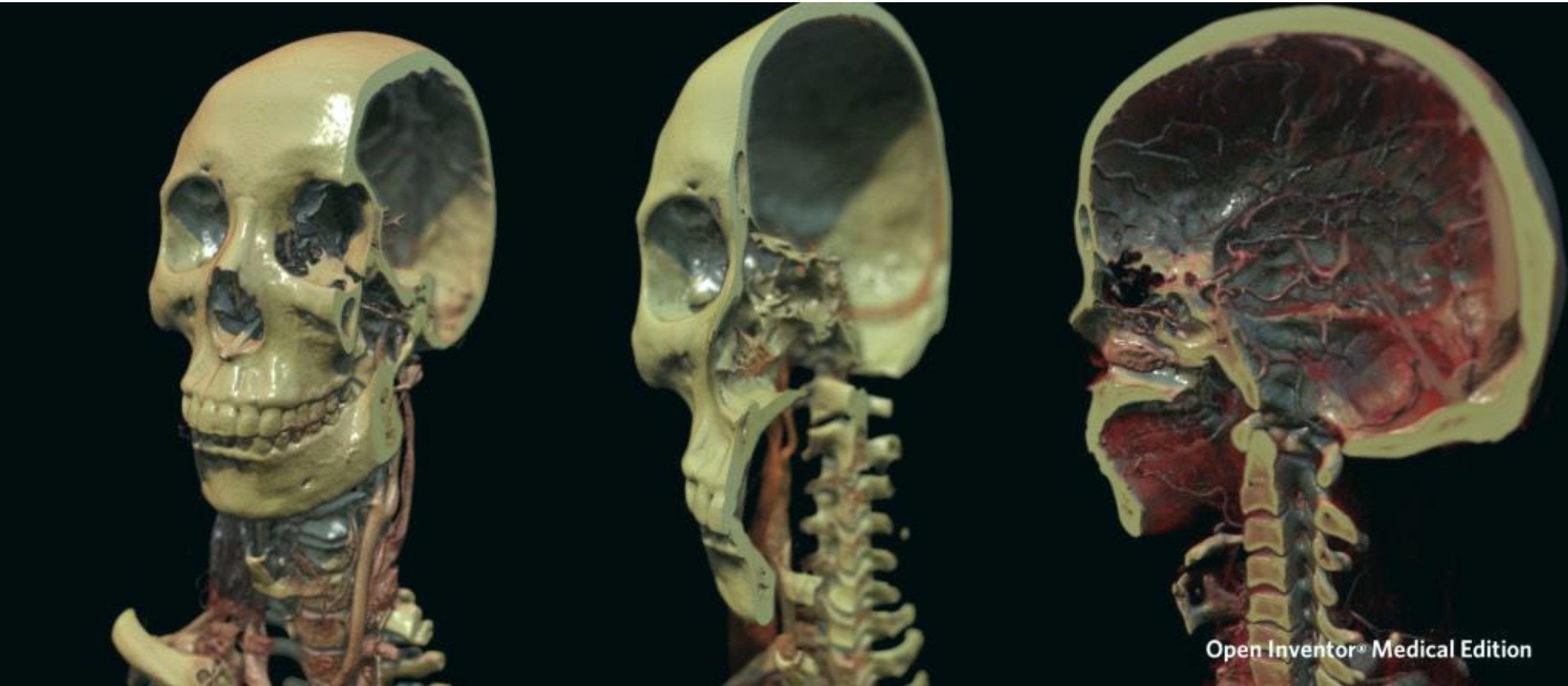
Volume Rendering

Computer Animation and Visualisation
Lecture 11

Taku Komura

Institute for Perception, Action & Behaviour
School of Informatics



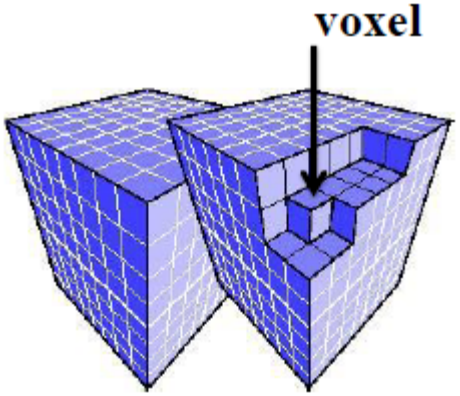
Overview

- Introduction to Volume Rendering
- Image order rendering
- Converting the profile into intensity
  - MIP, average, composite
- Composite:
  - Back-to-front accumulation
  - Front-to-back accumulation
- Image order trade-offs
  - Interpolation scheme, step-size
  - Sampling/traversal/shear warping

Volume Data

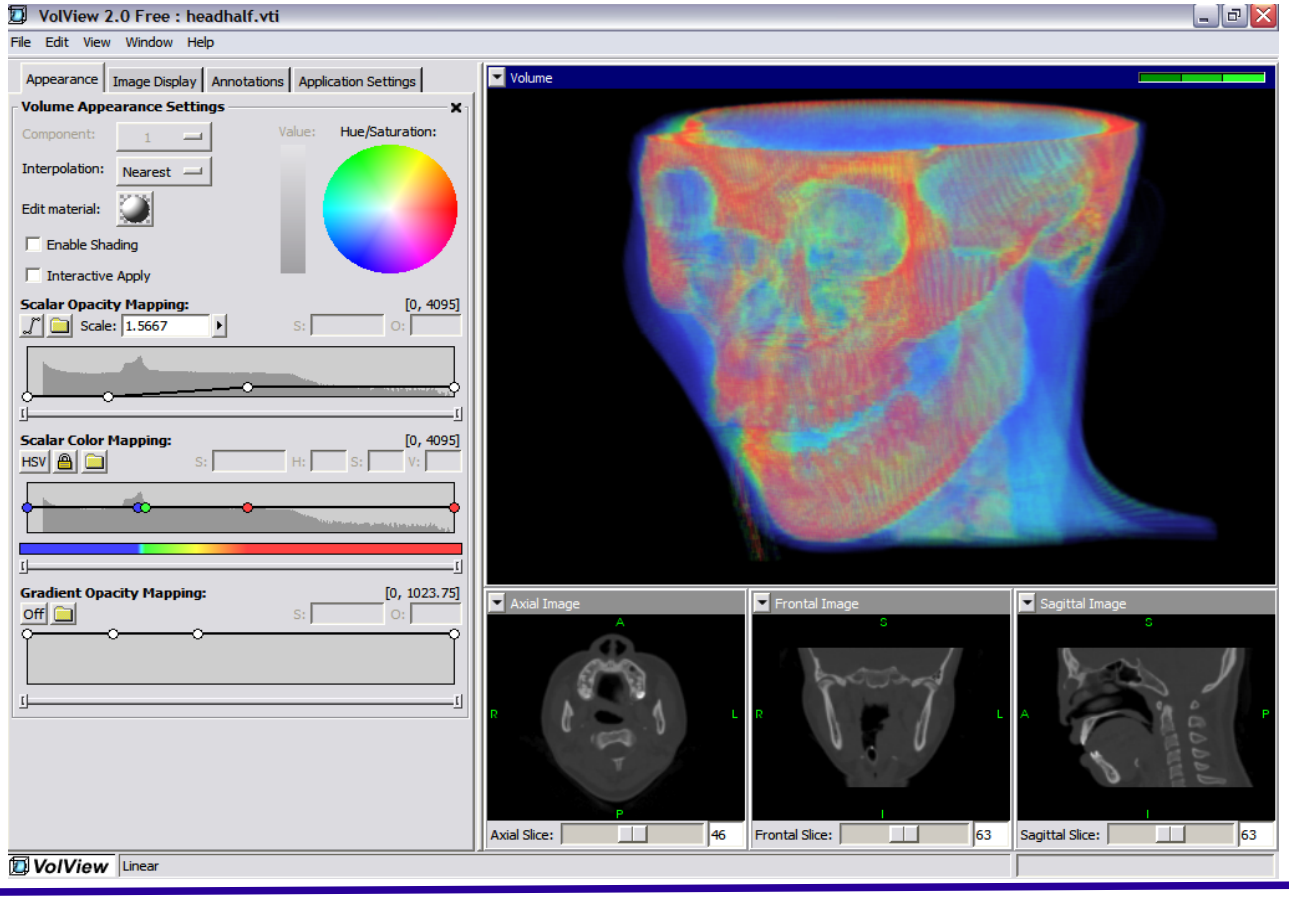
Usually, a data uniformly distributed in the 3D space

- *The volume is composed of cells called voxels*
- *The attributes can be of any dimensions (scalar, vector, tensor), but here we assume it is a scalar*
- *Sources:*
  - *Medicine (CT, MRI, ultrasound)*
  - *3D Simulation (Fluid simulation, FEM).*



Volume Rendering

- Display the **information inside the volume using colours and transparency**
    - **(Defining a colour / opacity transfer functions)**
- direct display : volumetric data → screen pixels



Volume Rendering – Why is it hard?

- *Classification*
  - assignment of **colour and transparency** to volume regions
    - defining a **transfer function**
- *Efficiency & Compactness*
  - volumetric data is ***BIG***
    - surfaces : 1 million polygons is considered big
    - 1 million voxels is considered small (100x100x100) – billions of voxels not uncommon
  - visualisation demands interaction

Rendering Transparent Objects

- *Need to be able to draw semi-transparent objects*
  - **Light is transmitted through the object**
- *In ordinary graphics, only drawing opaque objects*

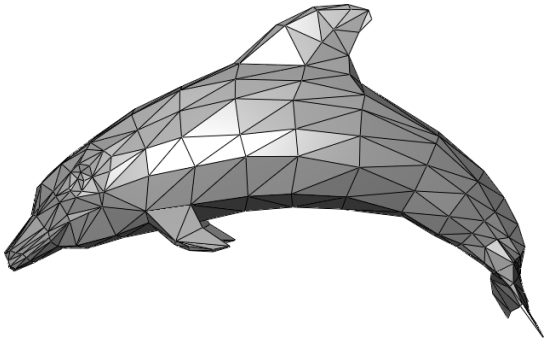
- *Transparent vs. Opaque*
  - ***transparency*** ***:*** the property of transmitting light
  - ***opacity*** ***:*** the property of not transmitting light

Overview

- Introduction to Volume Rendering
- **Image order rendering**
- **Converting the profile into intensity**
  - **MIP, average, composite**
  - **Composite:**
    - Back-to-front accumulation
    - Front-to-back accumulation
- Image order trade-offs
  - Interpolation scheme, step-size
  - Sampling/traversal/shear warping

Volume Rendering Techniques

- *Object Order Rendering*
  - scene drawing takes place 1 object at a time
  - geometric rendering



- *Image Order Rendering*
  - scene drawing takes place 1 pixel at a time

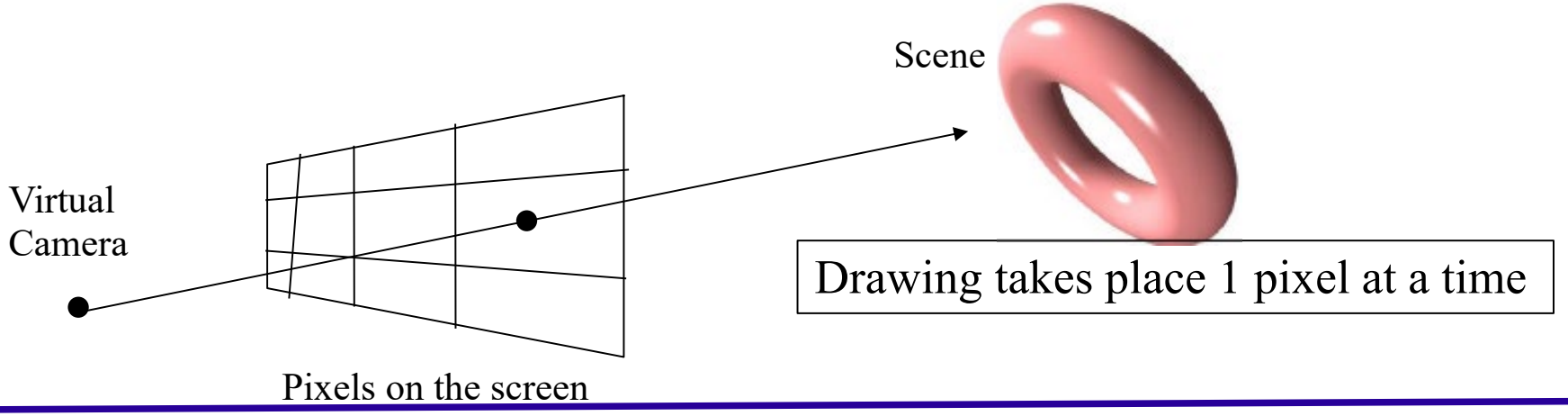


Image Order Rendering

- *Also known as Ray-tracing, or Ray casting*
- *For traditional Graphics scene*
  - Project an imaginary ray from the centre of projection (the viewers eye) through the centre of each 2D image pixel into the 3D scene.
  - Find the point on the object this ray intersects
  - Calculate shade for the point the ray hits by lighting equations

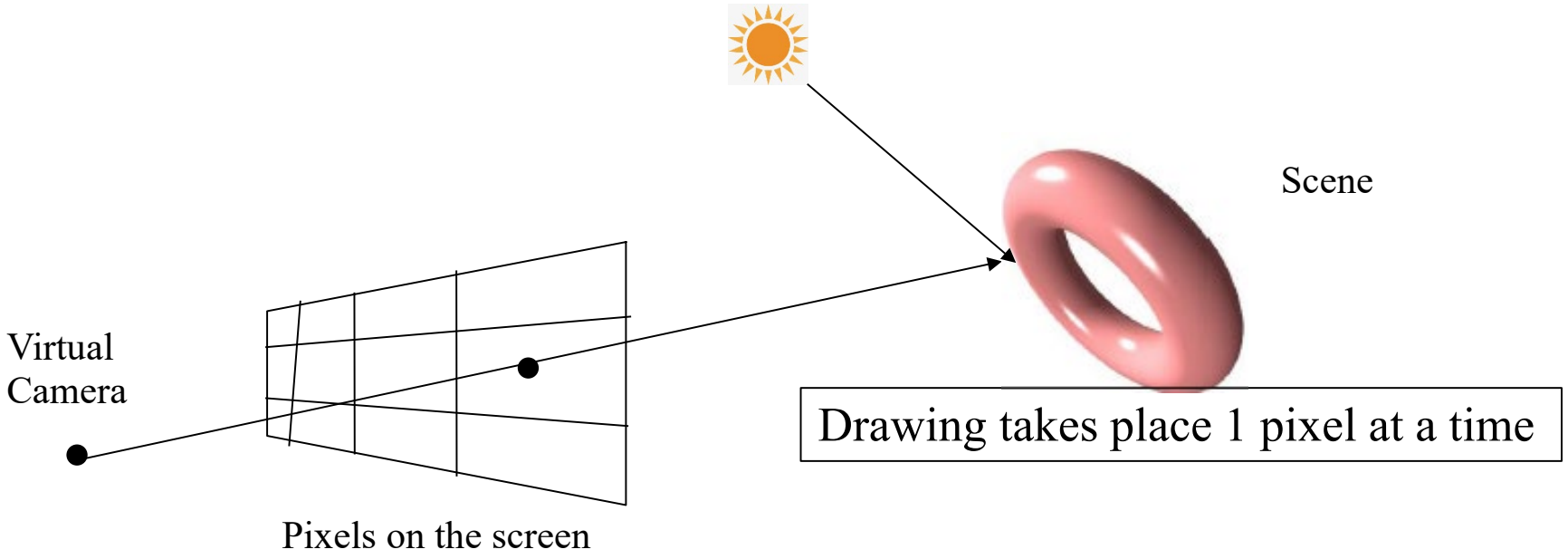
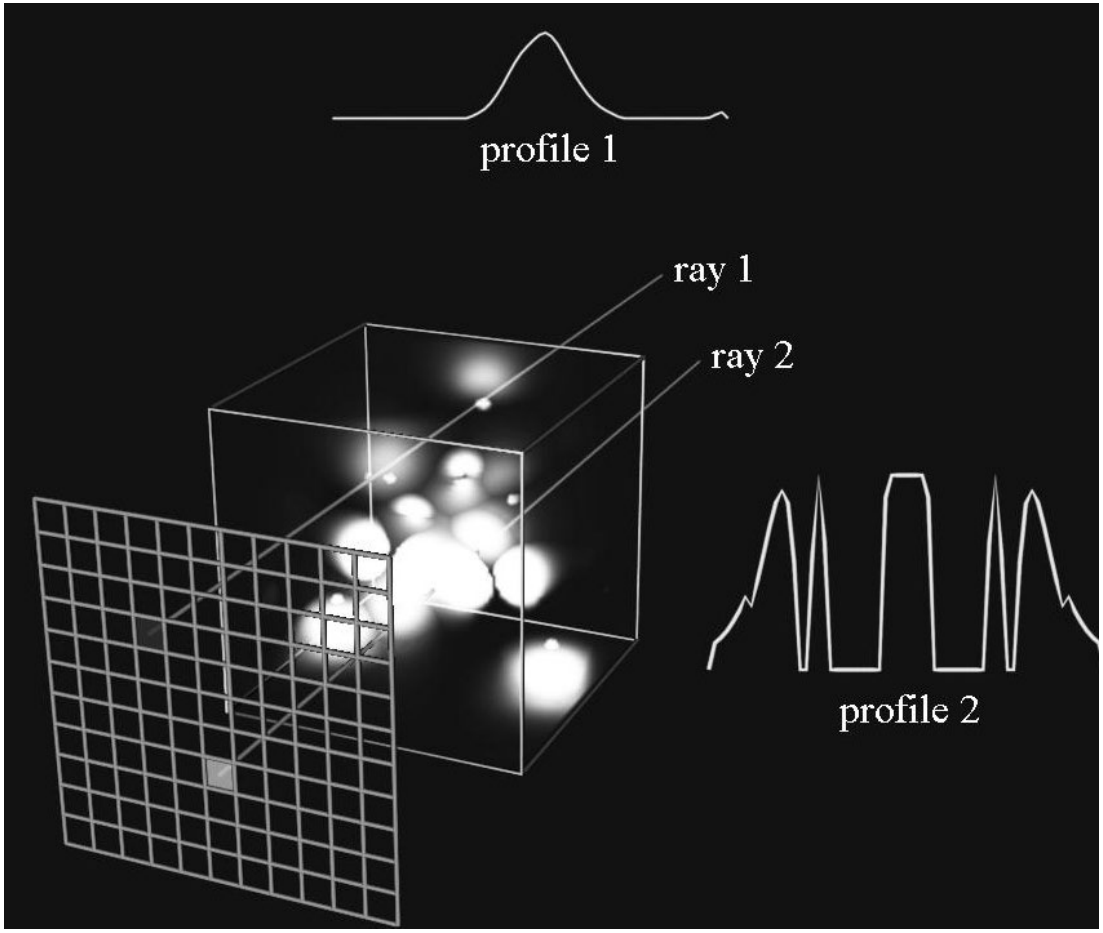
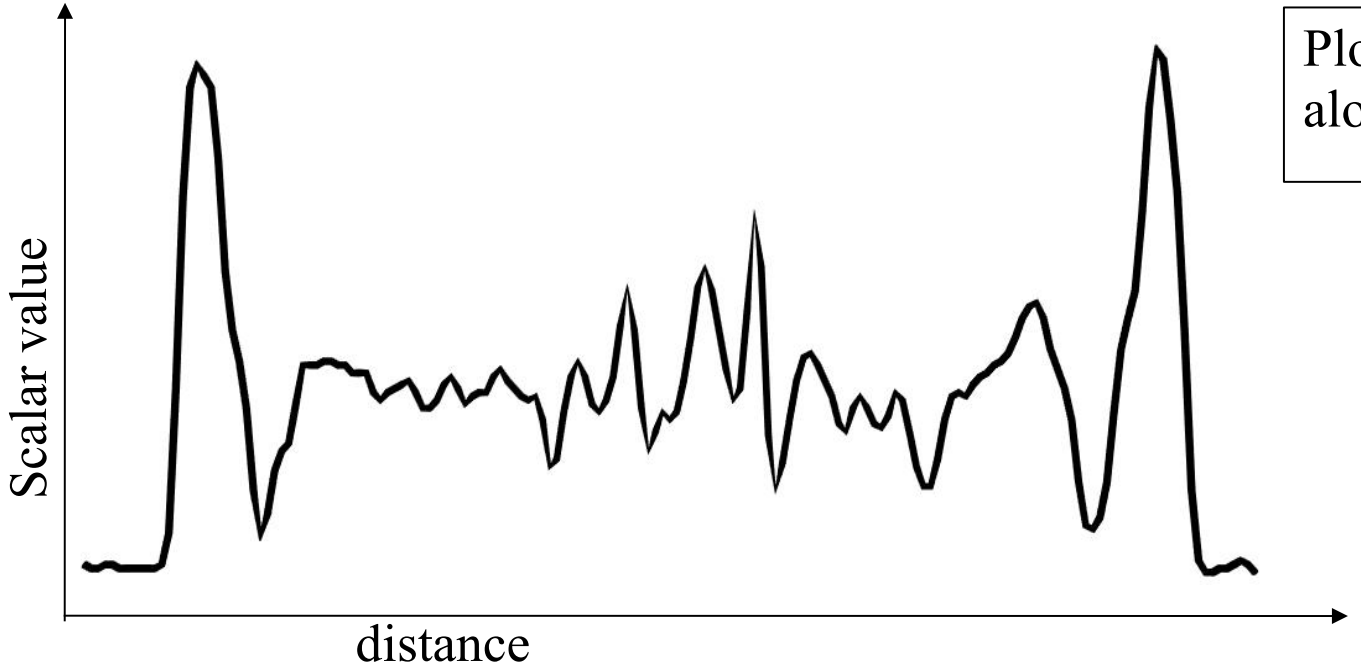


Image Order Volume Rendering

- *Cast rays through the pixels into 3D volume according to camera parameters*
- *Evaluate the scalar values encountered in the volume using a specific function*
- ***How do we convert this profile of scalar values to an intensity for display?***

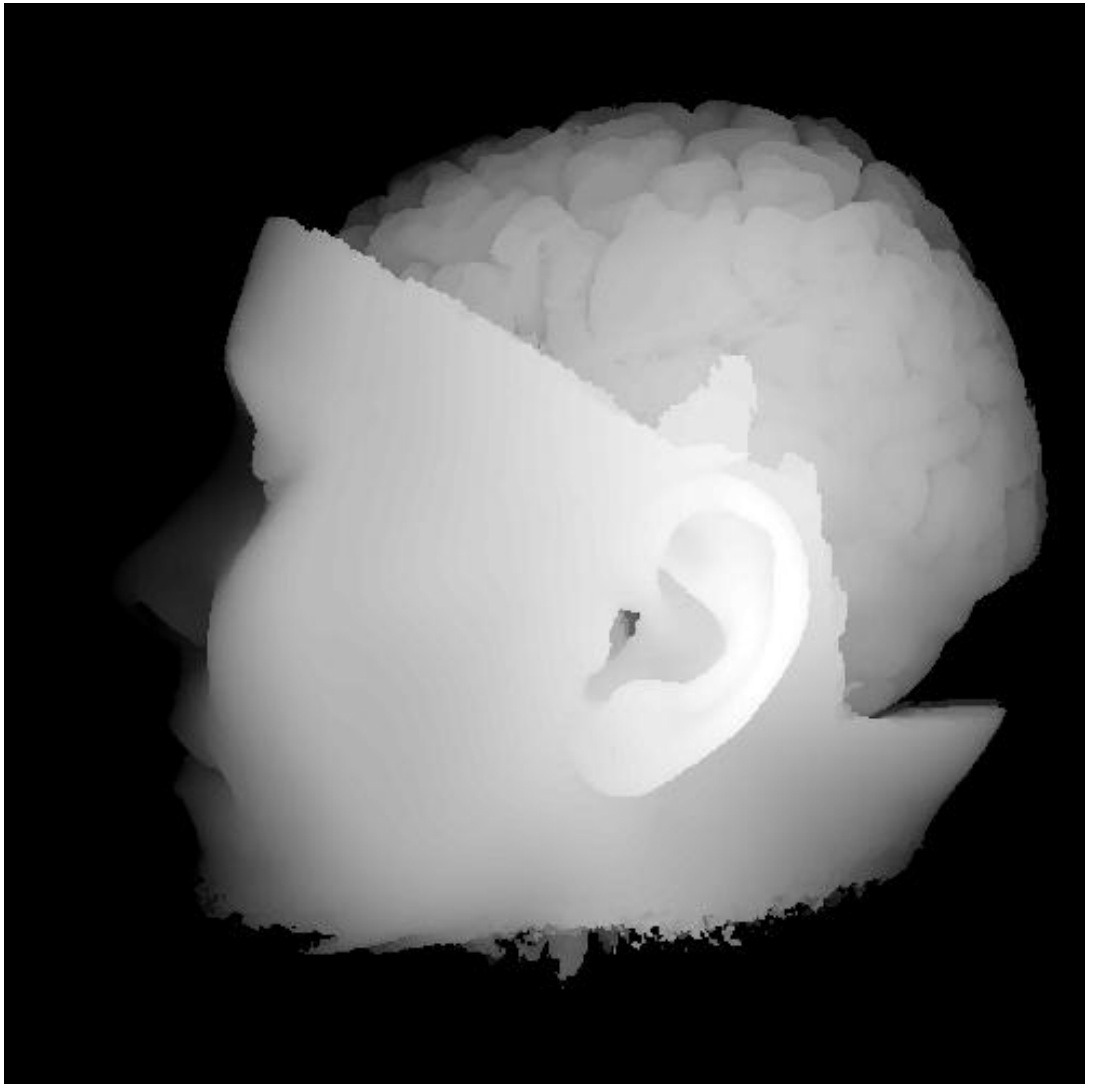


Q: How to convert the scalar value profile into an intensity **?**



- *Assume : voxels emit light proportional to their scalar value*
  - similar to greyscale image (no colour at the moment)

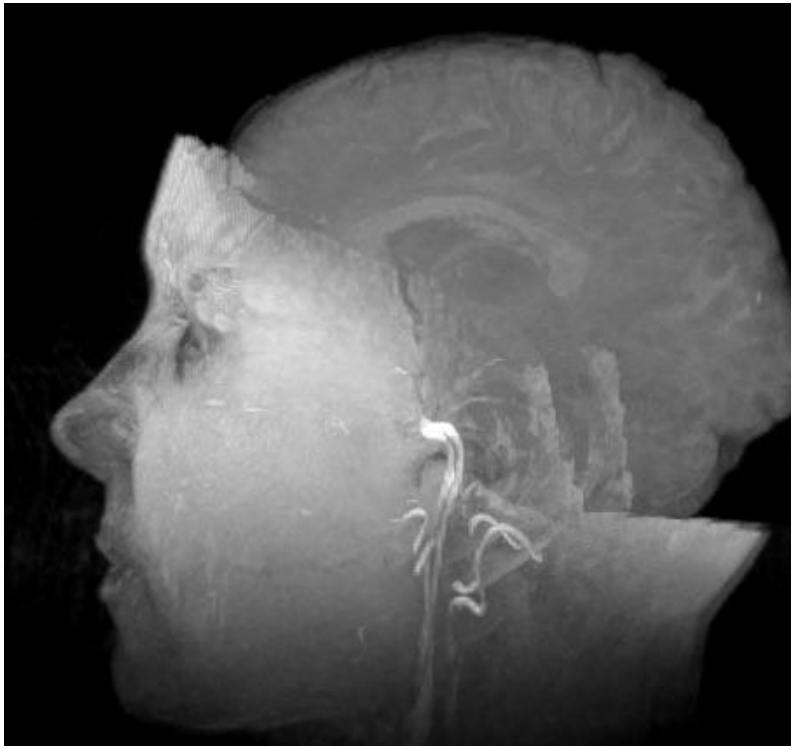
Example : 3D head volume

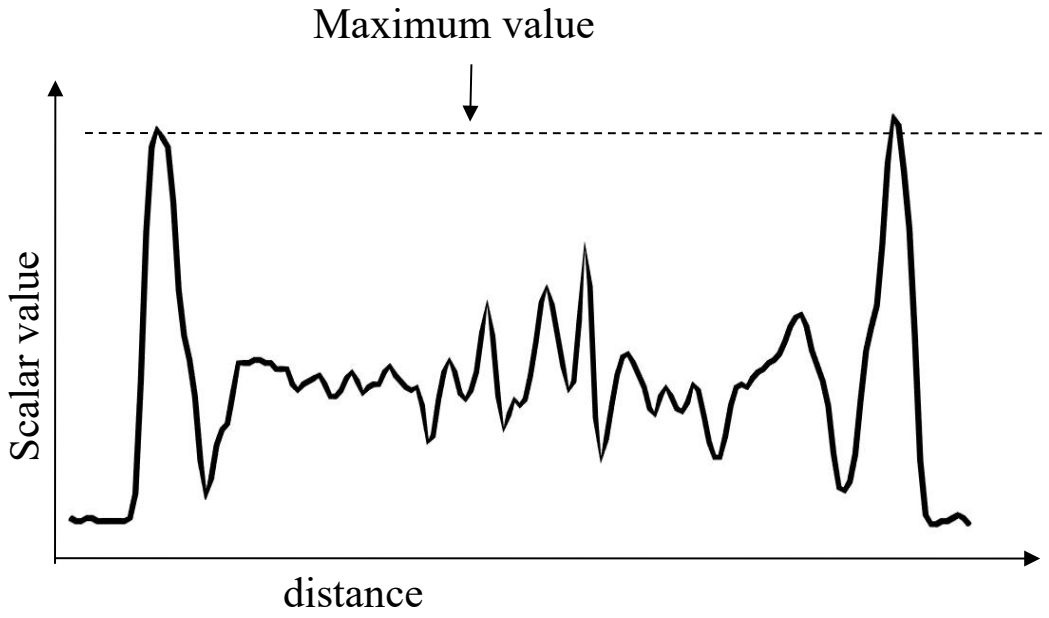


Rendered with no transparency : intensity ≈ distance

Option 1 : Maximum Value

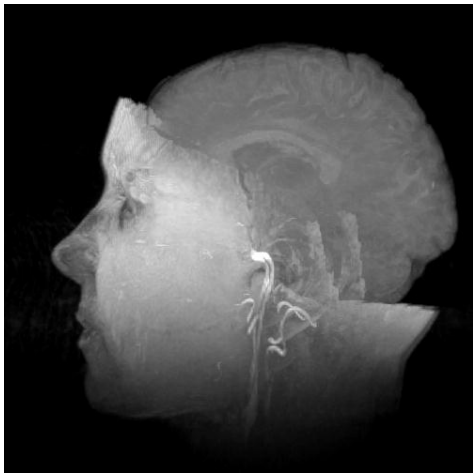
- *No blending or transparency*
  - intensity ≈ max. scalar value encountered on ray
  - **maximum intensity projection**





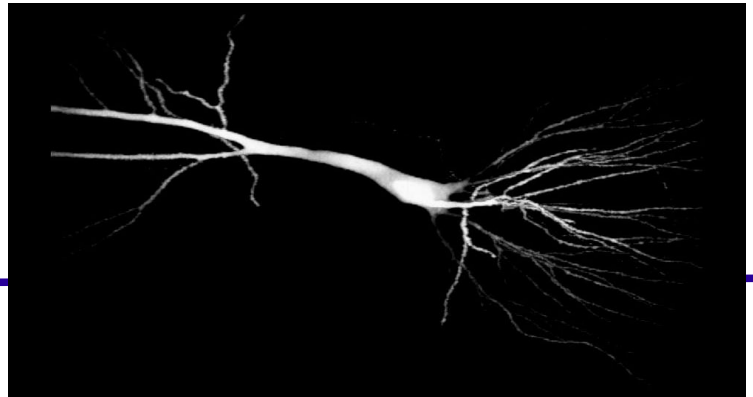
Maximum Intensity Projection (MIP)

- *Simplest method of Image Order Volume Rendering*
  - Pixel values ≈ maximum voxel values
- *with CT data, resembles a conventional X-ray image*
  - Useful for finding some unusual objects
  - Limitation : perception of depth is lost
    e.g. front/back ambiguity

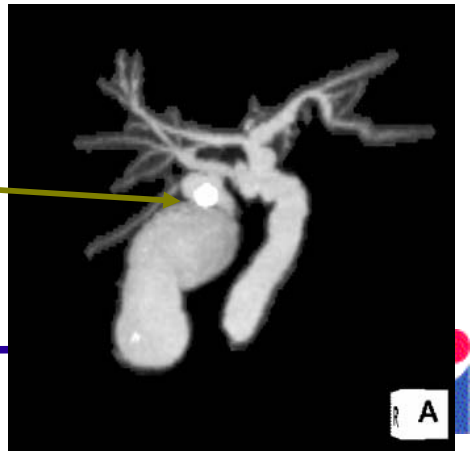


MIP rendering of a nerve cell.

. Which branches are in front, and which behind ?

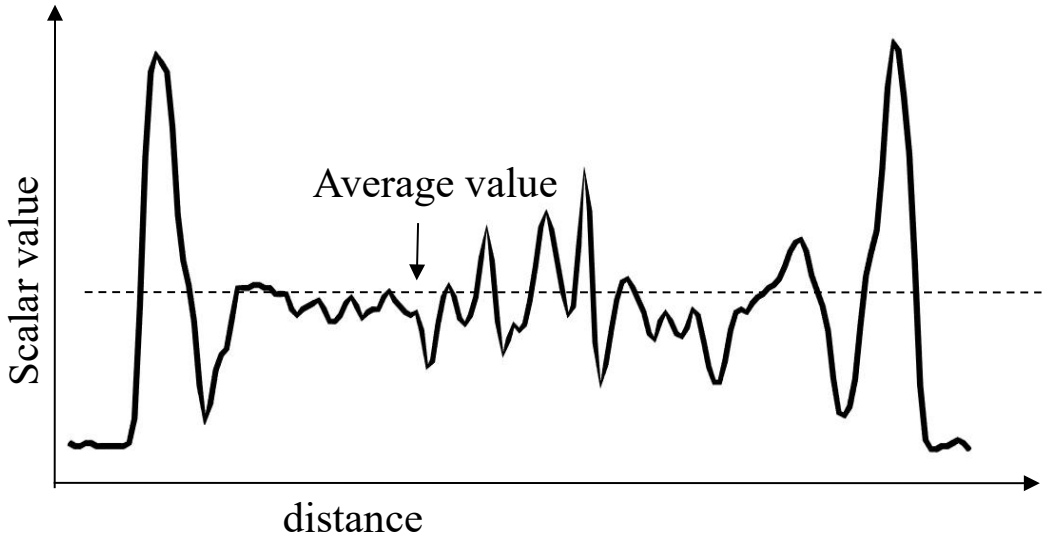


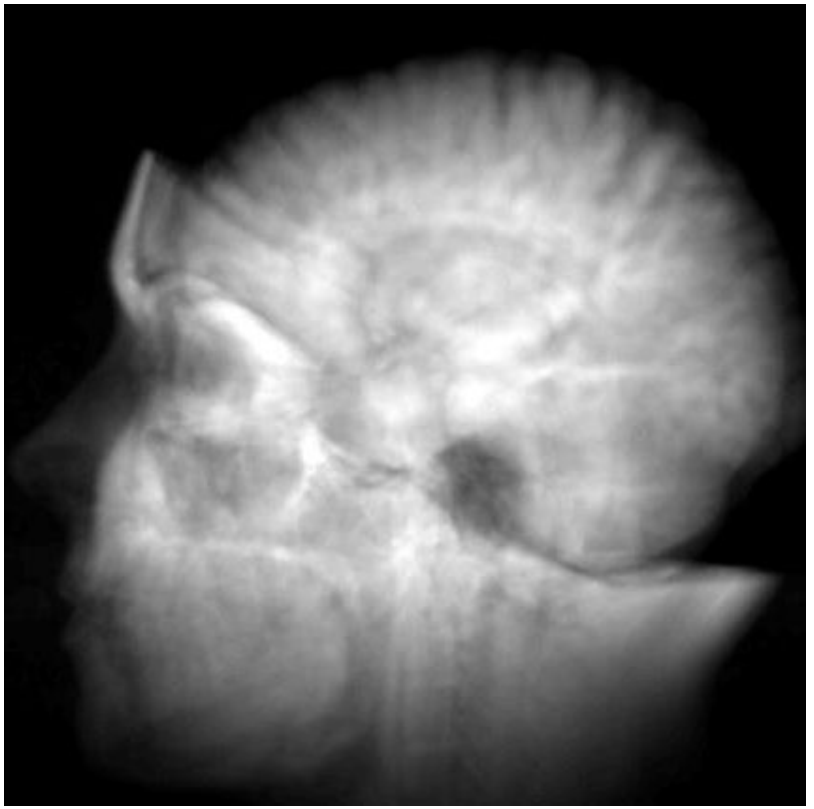
Bilestone in the cholecystitis



Option 2 : Average Value

- *Average value along the ray*
  - intensity ≈ mean scalar value
  - All the voxels will affect the result
  - Depth information is lost

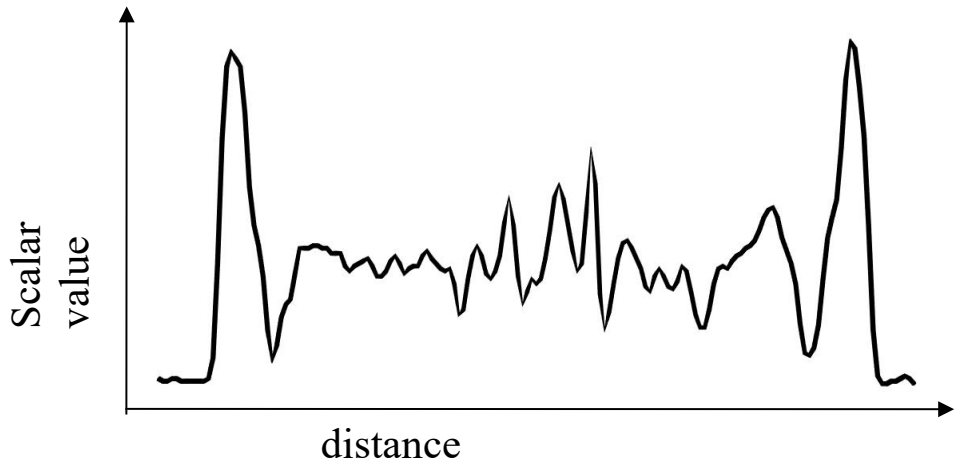


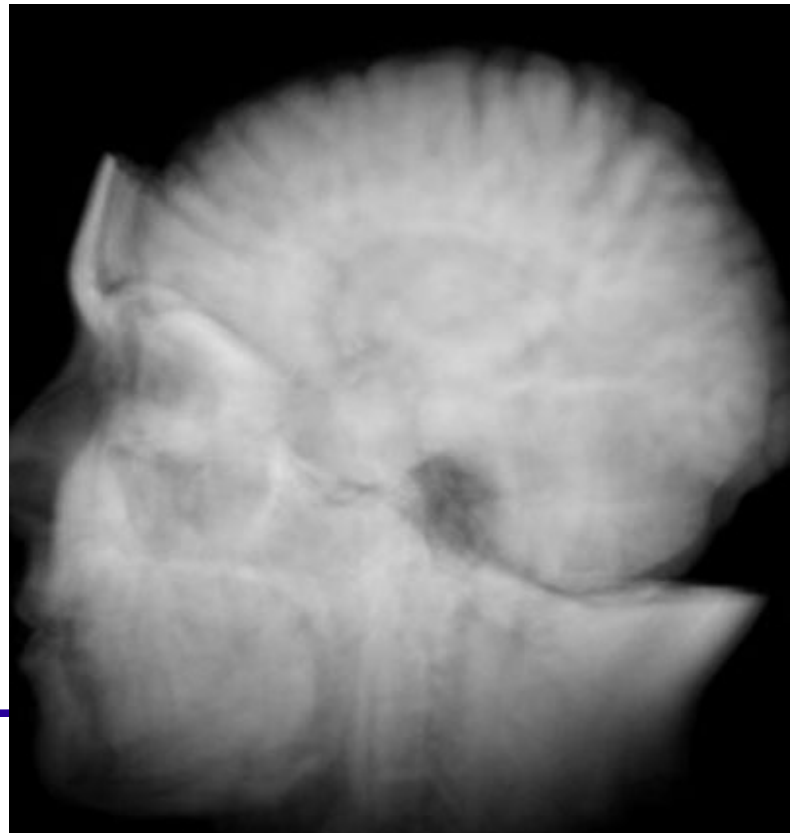


Option 3 : Composite Value

- *Value accumulated along the ray*
- *Intensity = scalar value accumulated along the ray*
- *Also treating scalar values as alpha transparency value at each voxel*
- *Closer objects have stronger influence*

(=> more bright = more visible)

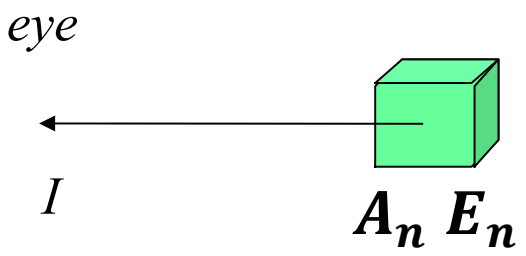




Composite Intensity Projection: Rendering with Transparency 1

'Light' that arrives at the eye is 'brightness' of cell + light transmitted

- multiply brightness by opacity (0 = transparent → 1 = opaque)
- If cell is totally transparent, cannot see cell brightness
- If cell is totally opaque, can see all of cell brightness

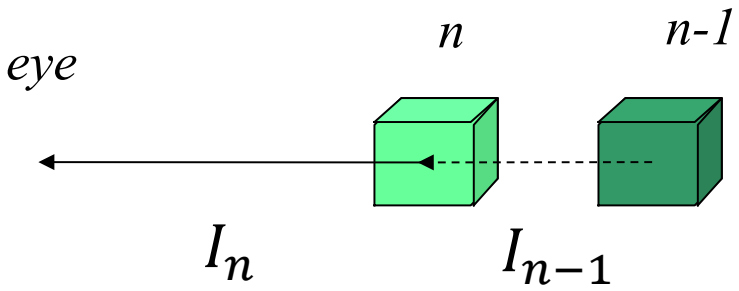


A_n: opacity, E_n: brightness

Rendering with Transparency 2

- *Back to front ray casting*
  - Only need to store current value of I

$$I_n = A_n\, E_n + (1 - A_n) I_{n-1}$$



Subscript *n* refers to cell n.

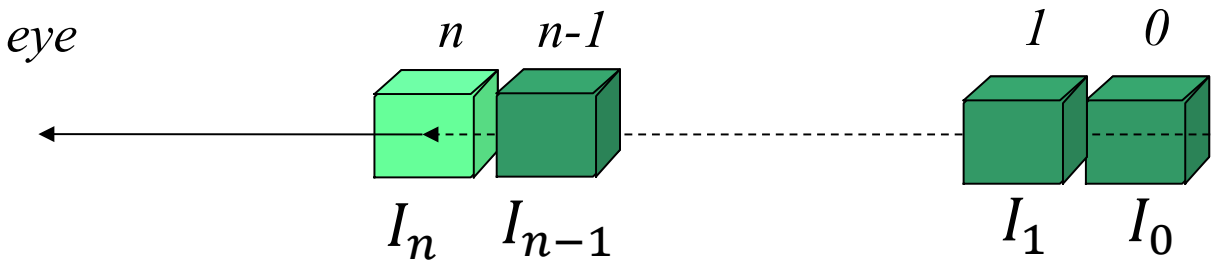
A refers to object *opacity.*

- Start with furthest away cell and blend towards the camera.
- I_n corresponds to current contents of the frame buffer.
- E_n Light emitted from cell n

Rendering with Transparency 2

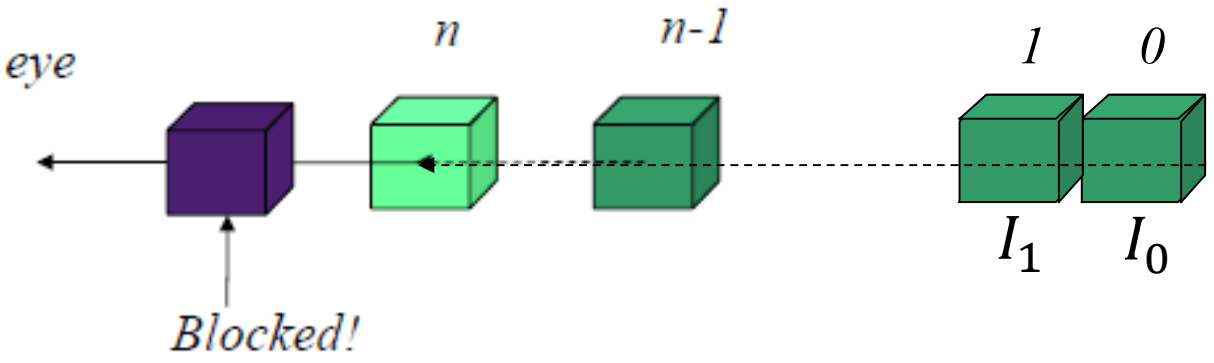
- *Back to front ray casting*
  - For calculating the colour of the pixel, we need to accumulate the brightness, starting from voxel 0

$$I_n = A_n\, E_n + (1 - A_n) I_{n-1}$$



Problem : Back to Front

If an opaque voxel is near the eye, all the previous computation is wasted



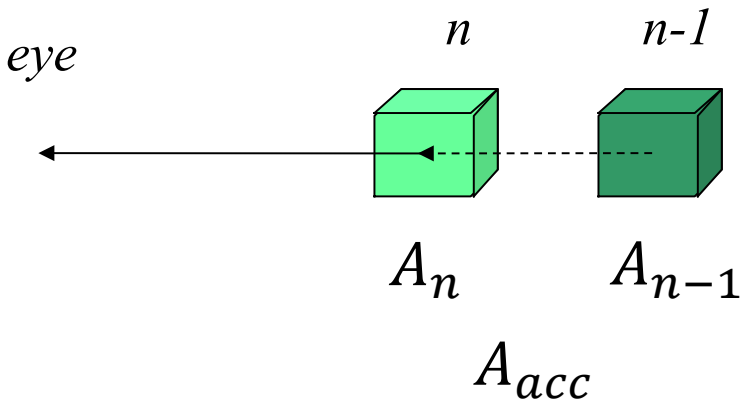
Accumulating the Alpha

- *Let's focus on how the light passes through the voxels*

$$I_n = A_n E_n + (1 - A_n) I_{n-1}$$

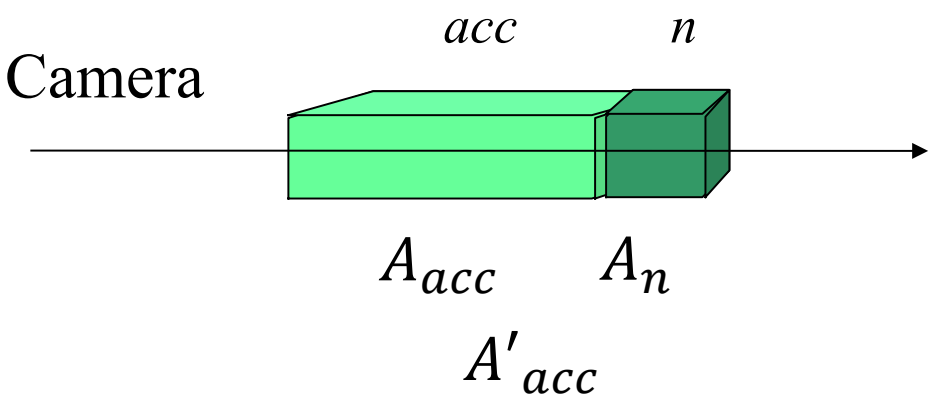
$$I_n = (1 - A_n) I_{n-1}$$

$$I_{n+1} = (1 - A_{n+1}) I_n$$
$$= (1 - A_{n+1})(1 - A_n) I_{n-1}$$
$$= (1 - A_{n+1} - A_n + A_{n+1} A_n) I_{n-1}$$
$$= (1 - A_{acc}) I_{n-1}$$



$$A_{acc} = A_{n+1} + A_n - A_{n+1} A_n$$
$$= A_{n+1} + (1 - A_{n+1}) A_n$$

Forward casting of ray



If we **use a buffer to store current alpha, we can accumulate starting at the nearest cell instead.**

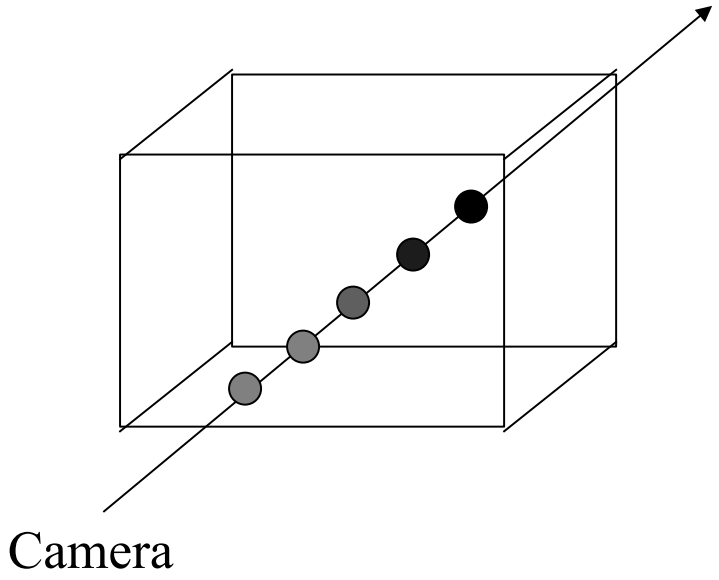
Supports early termination if opacity approaches 1.0

New alpha = Current Alpha + (1-Current Alpha) * Accumulated Alpha

$$A'_{acc} = A_{acc} + (1 - A_{acc})\, A_n$$

$$I'_{acc} = I_{acc} + (1 - A_{acc})\, A_n\, E_n$$

Early termination of ray



Current Alpha	Accumulated Alpha
0.2	0.2
0.4	0.4+(1- 0.4)*0.2=0.52
0.5	0.5+(1- 0.5)*0.52=0.76
0.4	0.4+(1- 0.4)*0.76=0.86
0.6	0.6+(1- 0.6)*0.86=0.94

- *Stop when opacity (alpha) reaches ~1.0 as nothing behind is visible*
  - ***In practice if remaining opacity (alpha) change < 1/256 no pixel change will occur → stop***

How to decide the transparency & brightness of the voxels

Deciding the {Color | Transparency} Transfer Function

- *Not very easy -> Lots of things embedded into each other*

Increase the opacity of the area where you want to see

Decrease the opacity where you do not want to see

Design a table of colors and brightness

- *Bones : white / high opacity*
- *Muscles: red / middle opacity*
- *Fat : values to beige / mostly transparent*

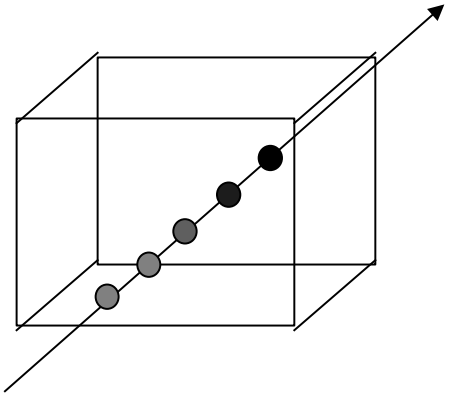
Try Volview

http://www.kitware.com/products/volview.html

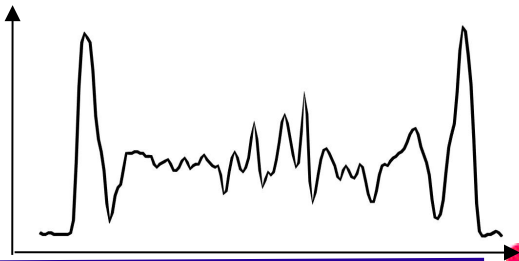
Overview

- Introduction to Volume Rendering
- Image order rendering
- Converting the profile into intensity
- MIP, average, composite
- Composite:
  - Back-to-front accumulation
  - Front-to-back accumulation
- **Image order trade-offs**
  - **Interpolation scheme, step-size**
  - **Sampling/traversal/shear warping**

Image Ordered Trade-offs

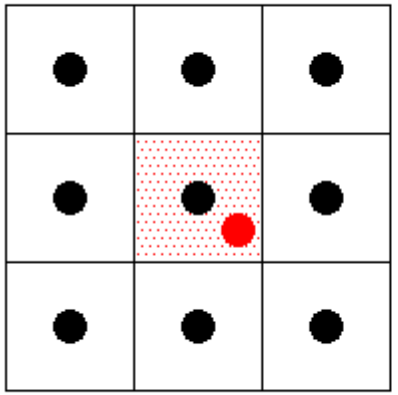


- *Interpolation method*
  - ray intersects cells not points
  - linear interpolation across cell (expensive)
  - Shortcut : nearest-neighbour (use nearest value)
- *Step size (along ray)*
  - **Large step size** = less computation = **faster rendering**
  - More artefacts
- *Sampling method*
  - **Uniform sampling** across voxel gives smooth results
  - Shortcut : **voxel-by-voxel sampling** is faster

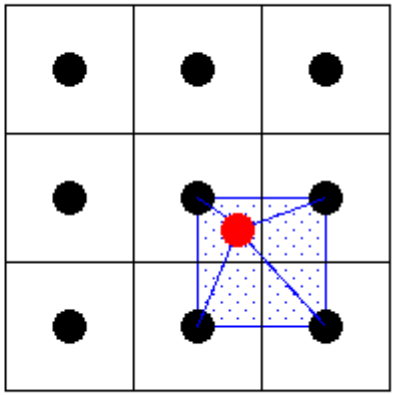


Interpolation & Sampling

Nearest neighbour



Linear interpolation

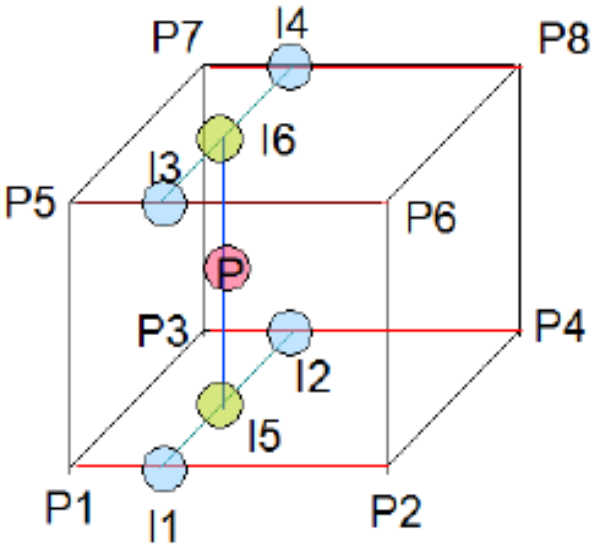


Trilinear Interpolation

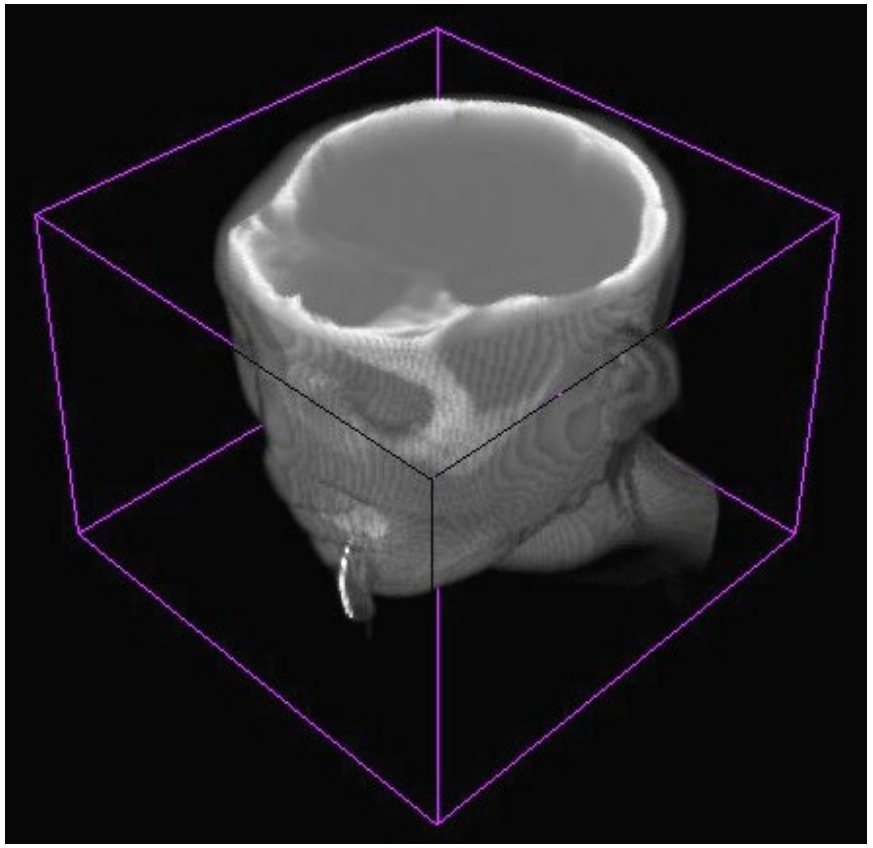
First interpolate between the vertices and find the projected point on the edges

Then interpolate those to find the projected points on the face

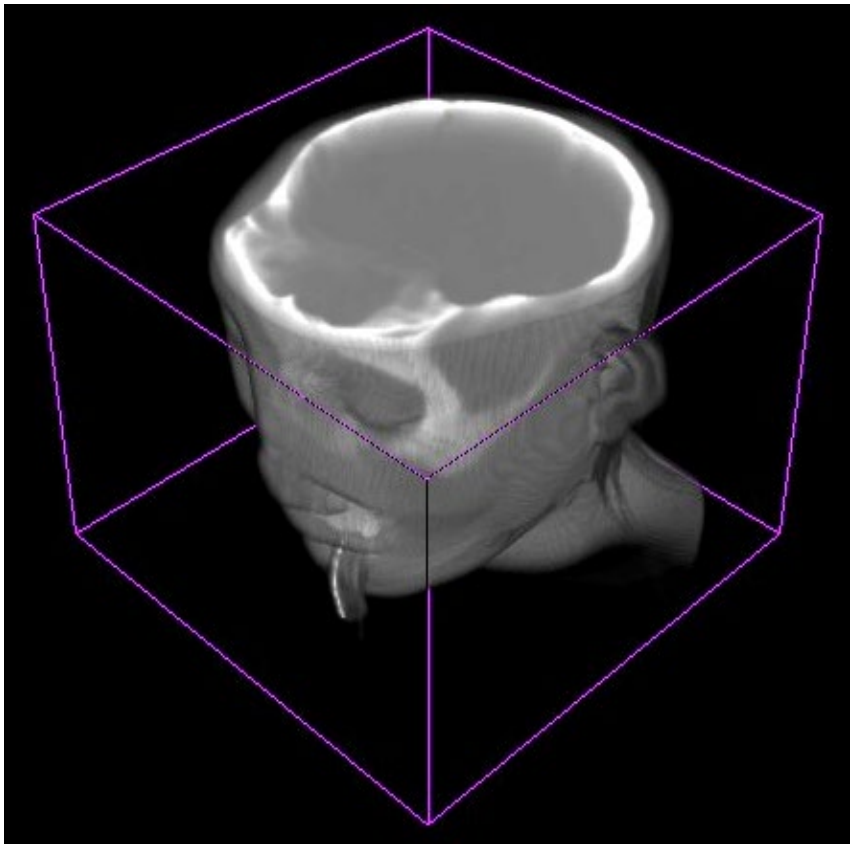
Finally interpolate them to find the value at the point



Effect of interpolation



Nearest neighbour
(visible artefacts in rendering)

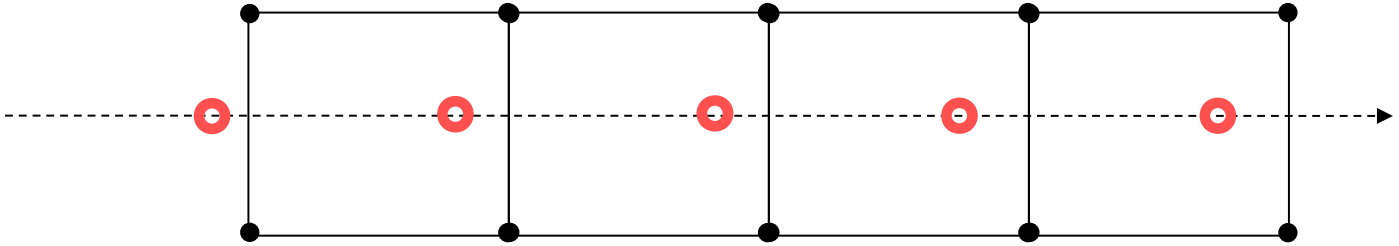


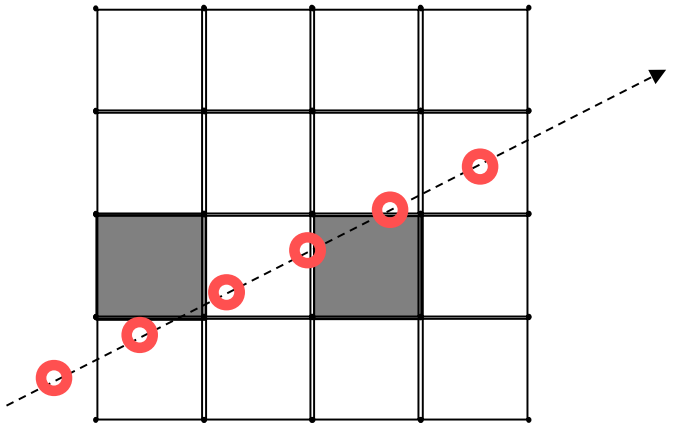
Tri-linear interpolation (i.e. in X,Y & Z)
(smoother, no artefacts)

Step = 1.0

Effect of Step Size 1

Step size and interpolation method have little effect if the ray lies along an axis of the grid.

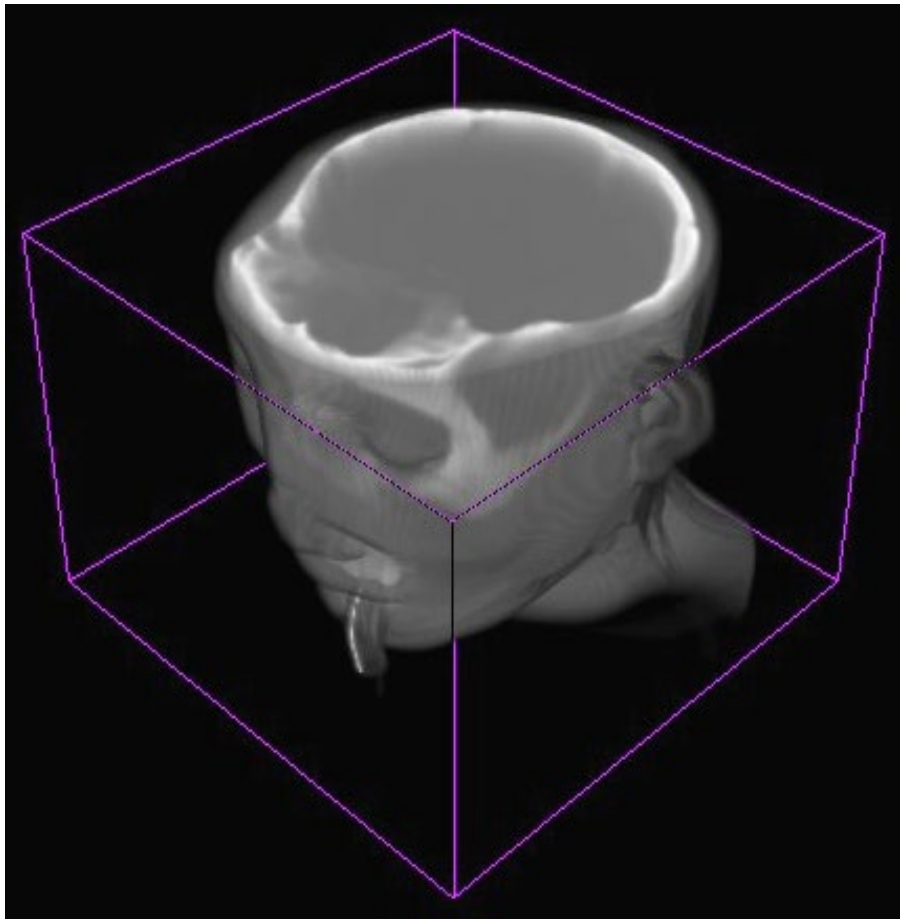




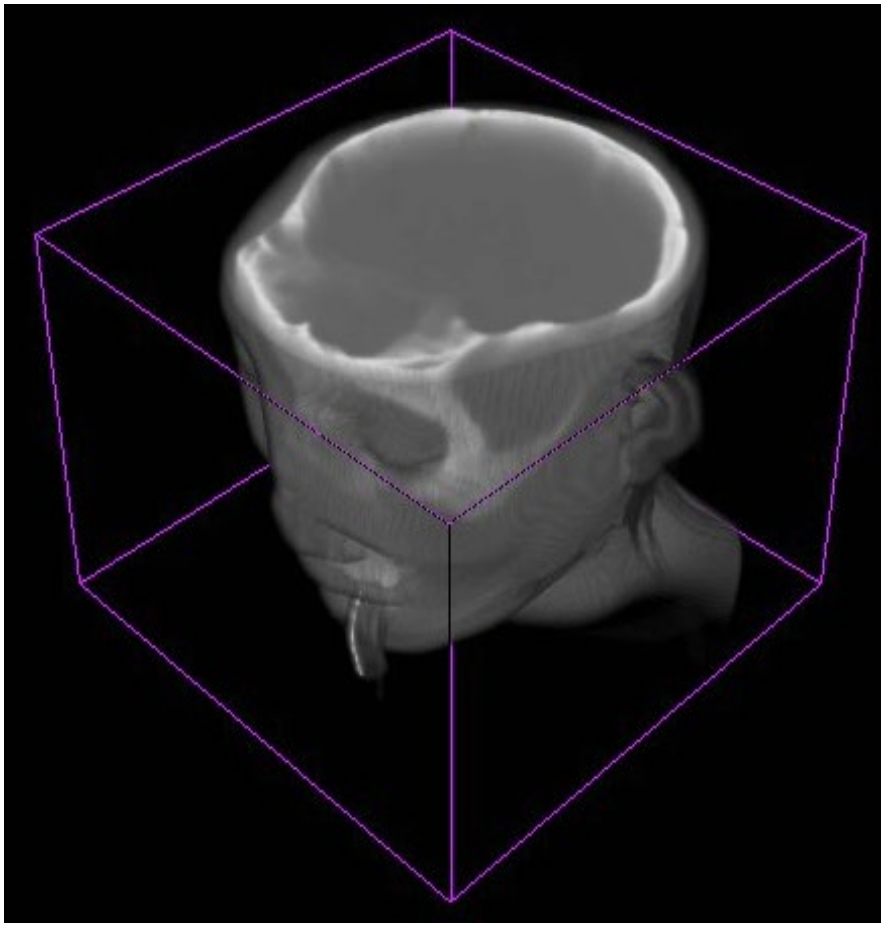
But in the general case, some voxels can be missed due to sampling, even if the spacing is 1 voxel width.

- Leads to artefacts appearing.

Effect of Step Size 2

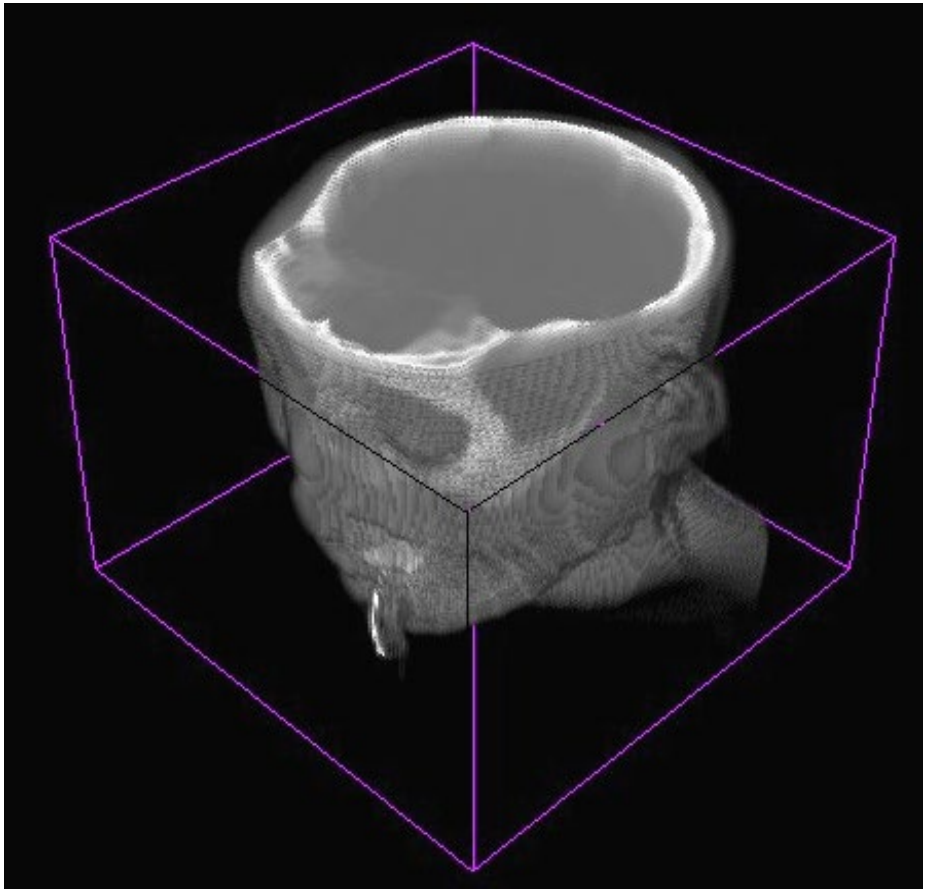


Step = 0.2

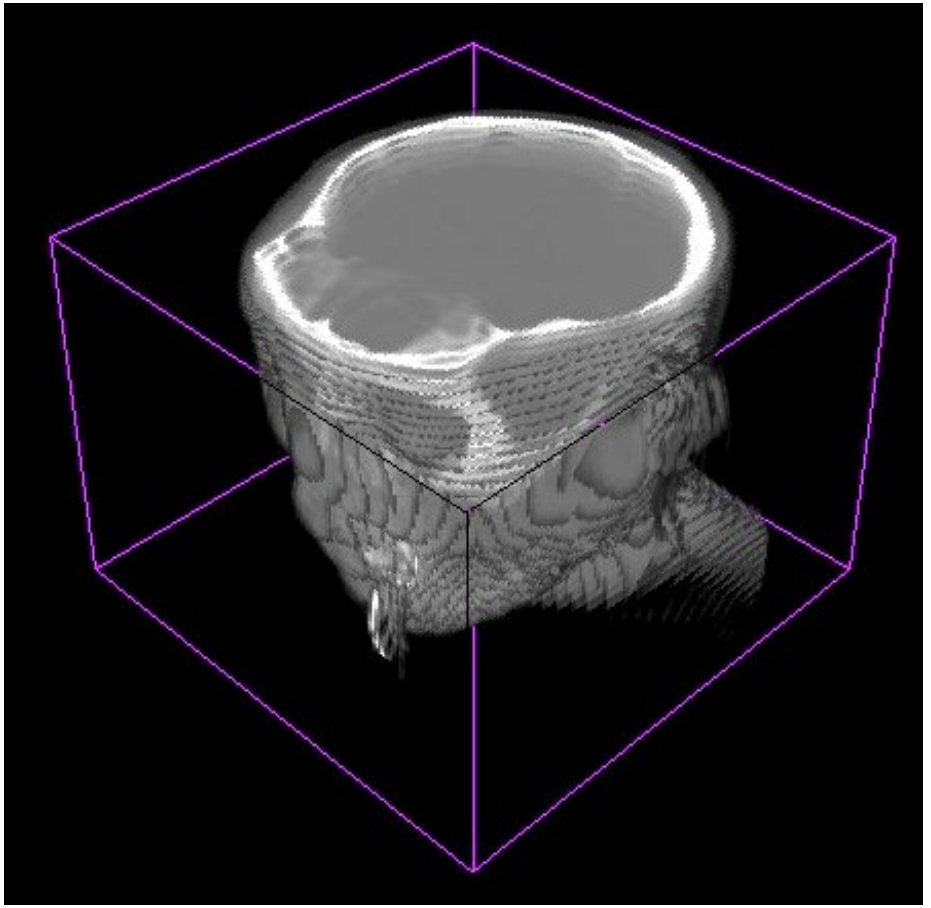


Step = 1.2
(mild
artefacts)

Effect of Step Size 3



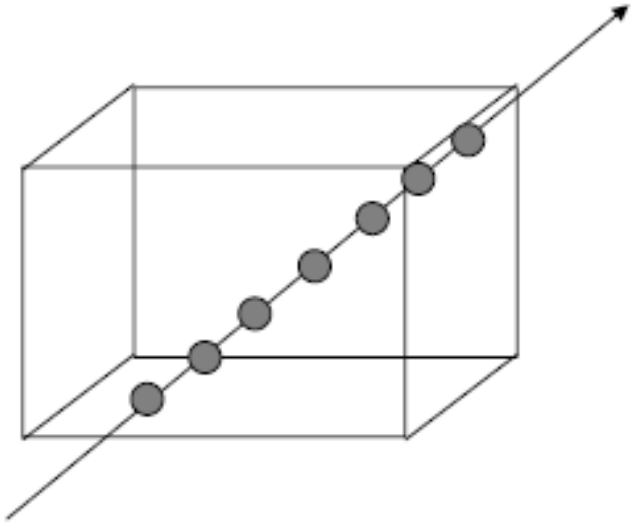
Step = 3.5
(visible artefacts)



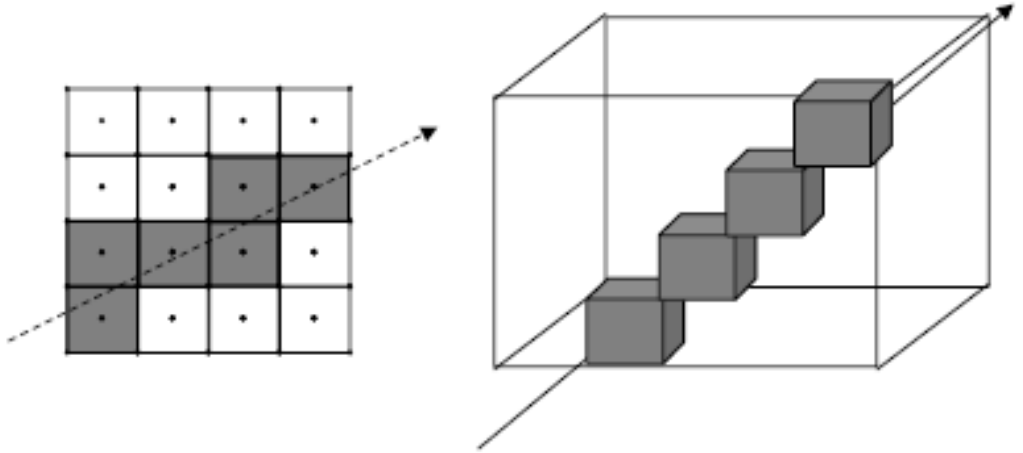
Step = 7.0
(highly visible artefacts
- distract from data itself)

Uniform / Voxel-by-voxel sampling

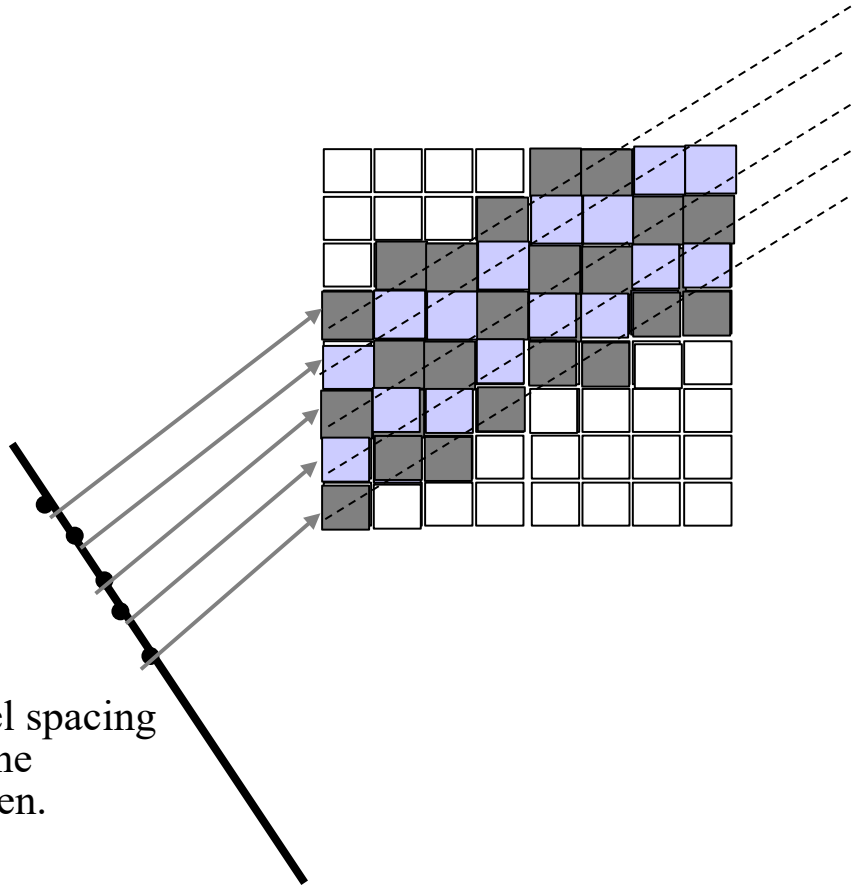
Uniform sampling



Voxel-by-voxel traversal



Voxel Traversal

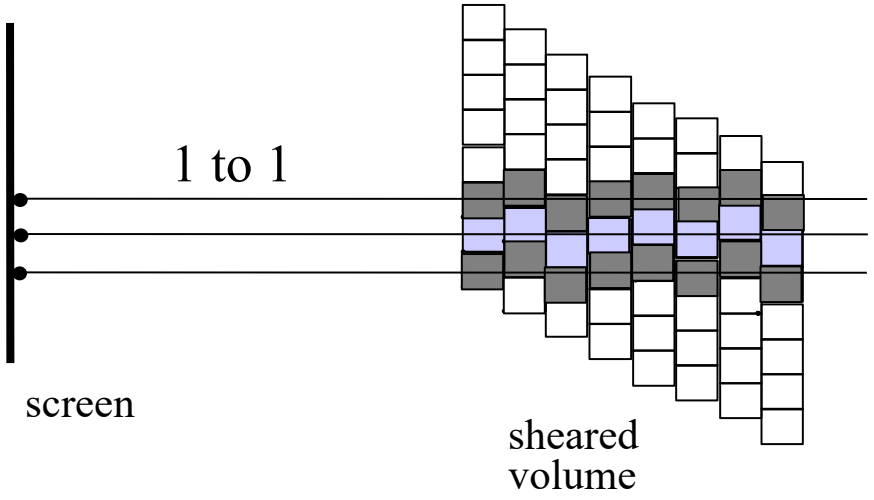


If we originate rays in each voxel at the same position of each grid, each ray forms an identical path of voxels through the grid

- Path known as a **template**
- All identical, pre-compute, save computation

Shear-warp factorisation

- *Instead of traversing along rays, visit voxels in a plane*
  - regardless of camera viewing angle [Lacroute '94]
- *Perform final warp on the image due to the shear process*
- *Assumes a orthographic camera model*
  - *projection perpendicular to image plane*

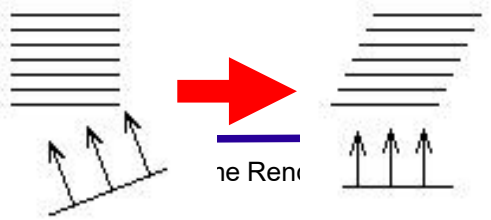


· Rays are cast from the base plane voxels at the same place.

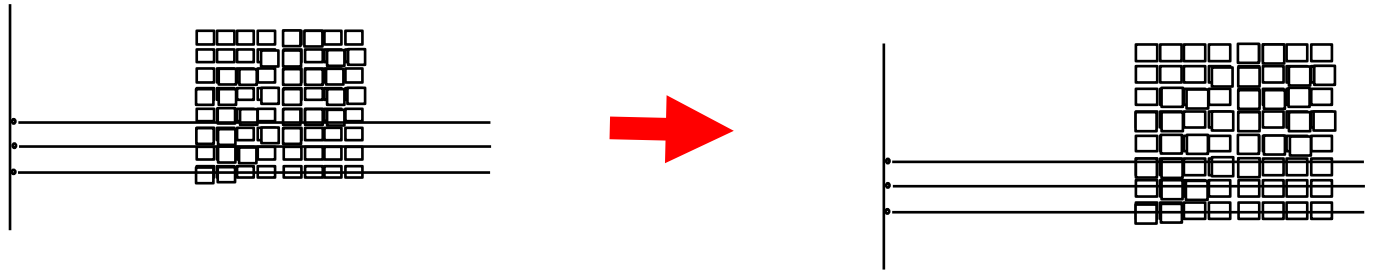
· They **intersect voxels on subsequent planes in the same location.**
The weights (the contribution of each voxel) are easy to compute

· Only **one set of interpolation weights needs to be computed for all the voxels in a slice**

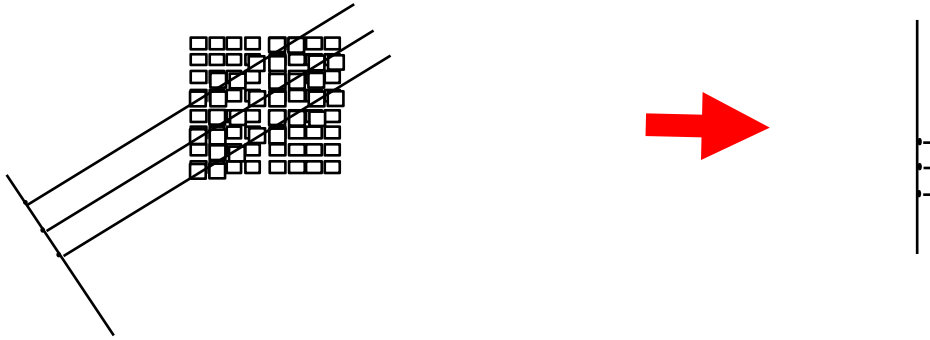
- **volume is sheared** so that rays remain perpendicular to base plane
  - lead to **efficient ray computation**



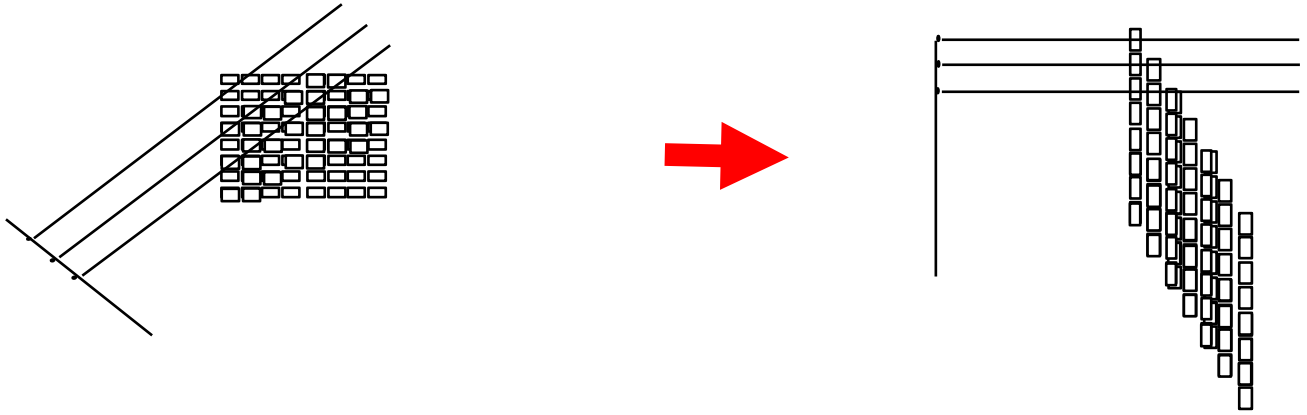
Rotation of camera with *shear-warp*



No rotation



Small angle

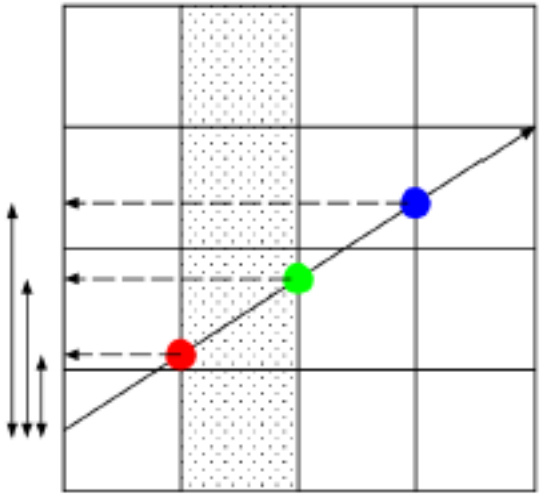


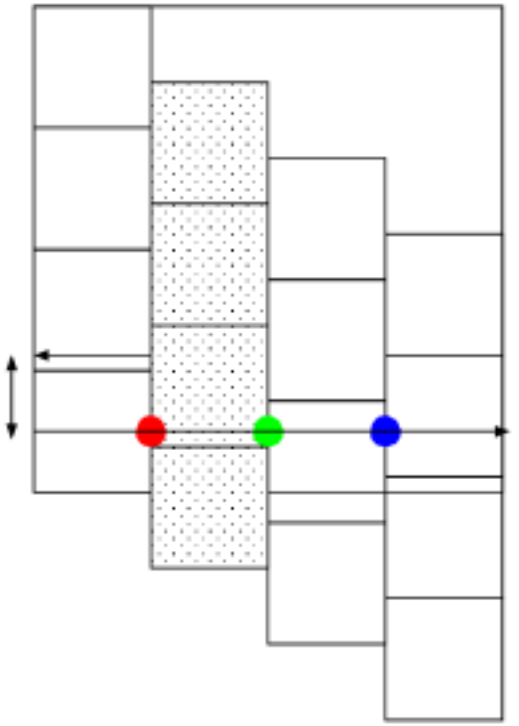
Large angle

How to decide how much to shear?

Calculate the intersection of the ray and the front plane

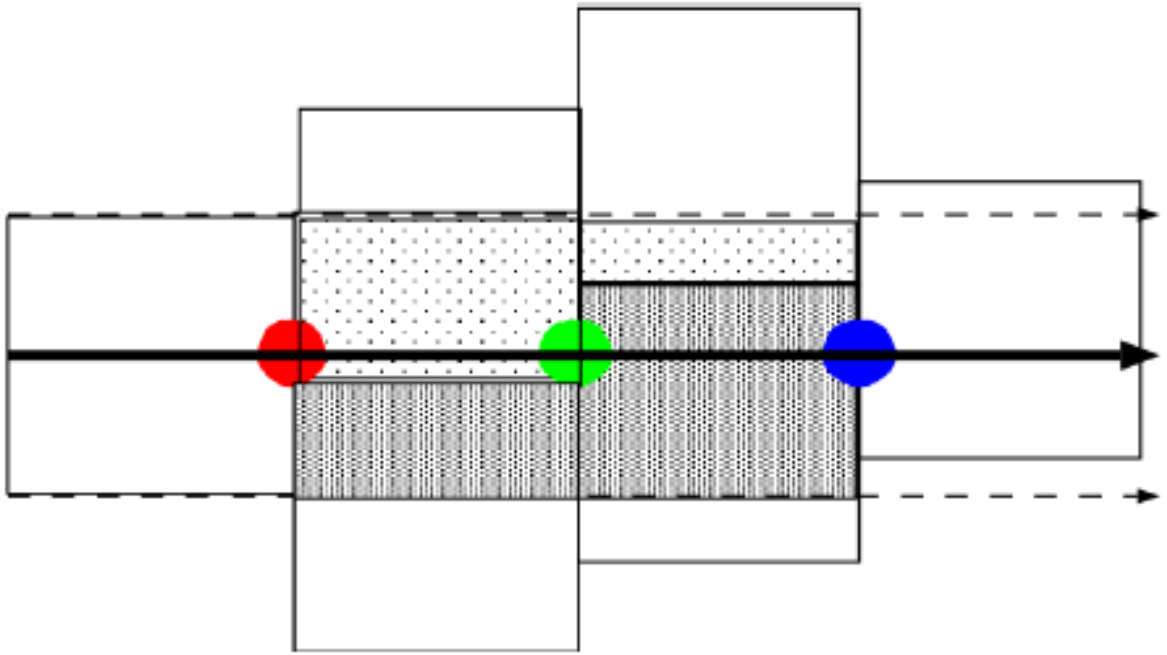
That gives the offset





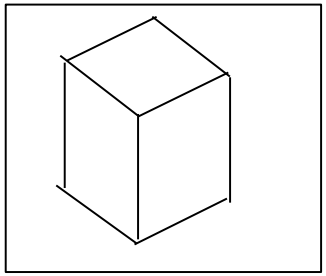
How to decide weights?

Can compute it based on how much the pixel is overlapped with the voxel

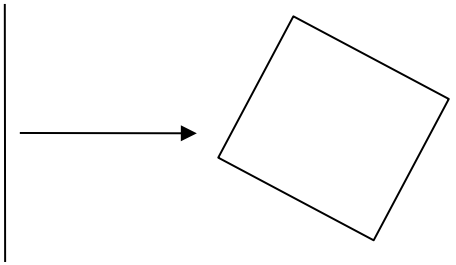


Final stage of shear-warp

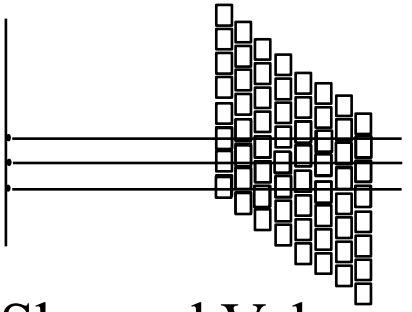
- *Final stage of re-sampling (warp) required*
  - ***transform resulting image from sheared space to regular Cartesian space (image plane).***



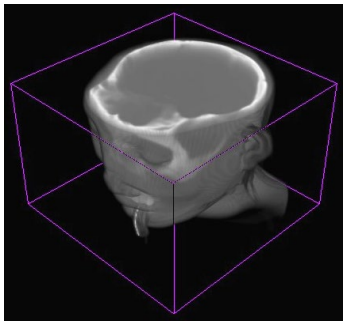
View of Volume

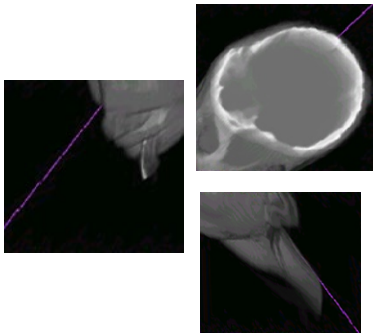


Top down view

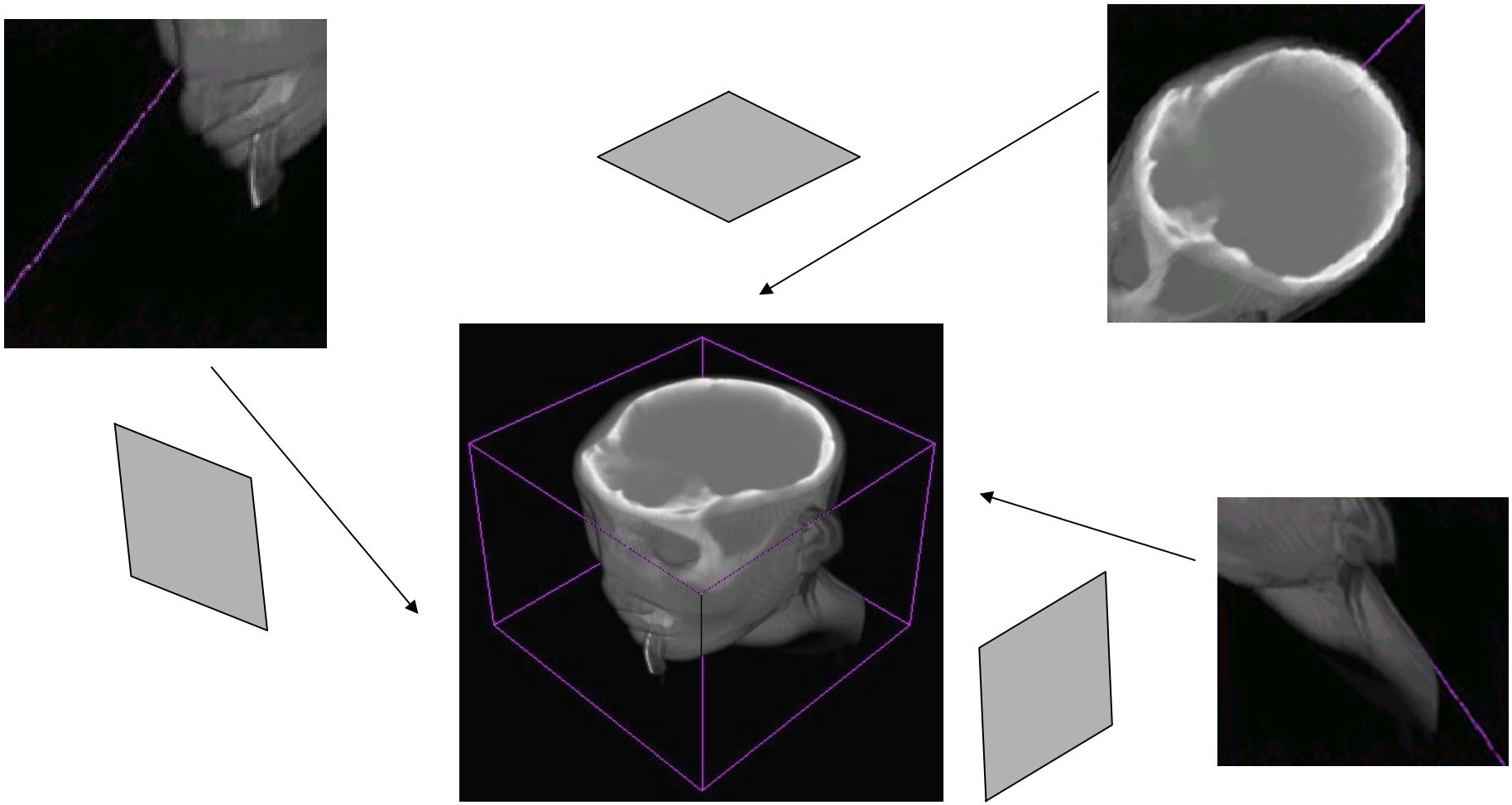


Sheared Volume





Warping the images



- Perform warping to get samples back into 2D image grid correctly
  - amount of warp required dependant on viewing angle to volume

Summary

- *Volume Rendering*
  - display the information inside the volume
  - use of transparency (/opacity)
- *Image Order Volume Rendering*
  - ray casting
  - intensity transfer function (scalar ray profile → intensity)
  - Opacity transfer function (scalar ray profile → opacity)
  - ray casting with transparency
  - trade-offs in image based : interpolation, sampling, step-size

Readings

- *Marc Levoy, "Display of Surfaces from Volume Data", IEEE CG&A, May 1988.*
- *Lacroute and Levoy, Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation, SIGGRAPH '94*
- *Westover, "Footprint evaluation for volume rendering", SIGGRAPH '90*