Volume Rendering

Visualisation – Lecture 10

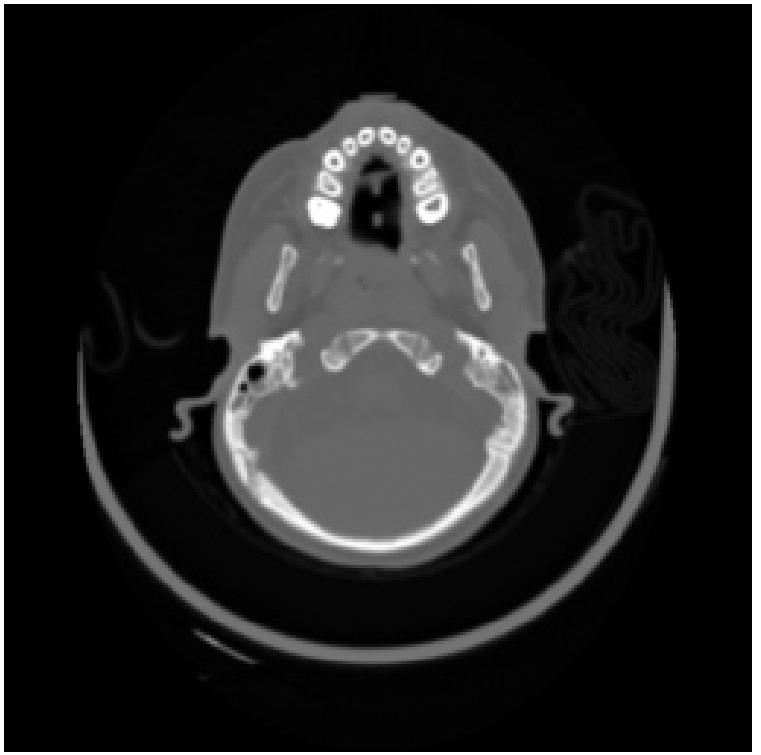
Taku Komura

Institute for Perception, Action & Behaviour
School of Informatics

Previously .... contour or image rendering in 2D

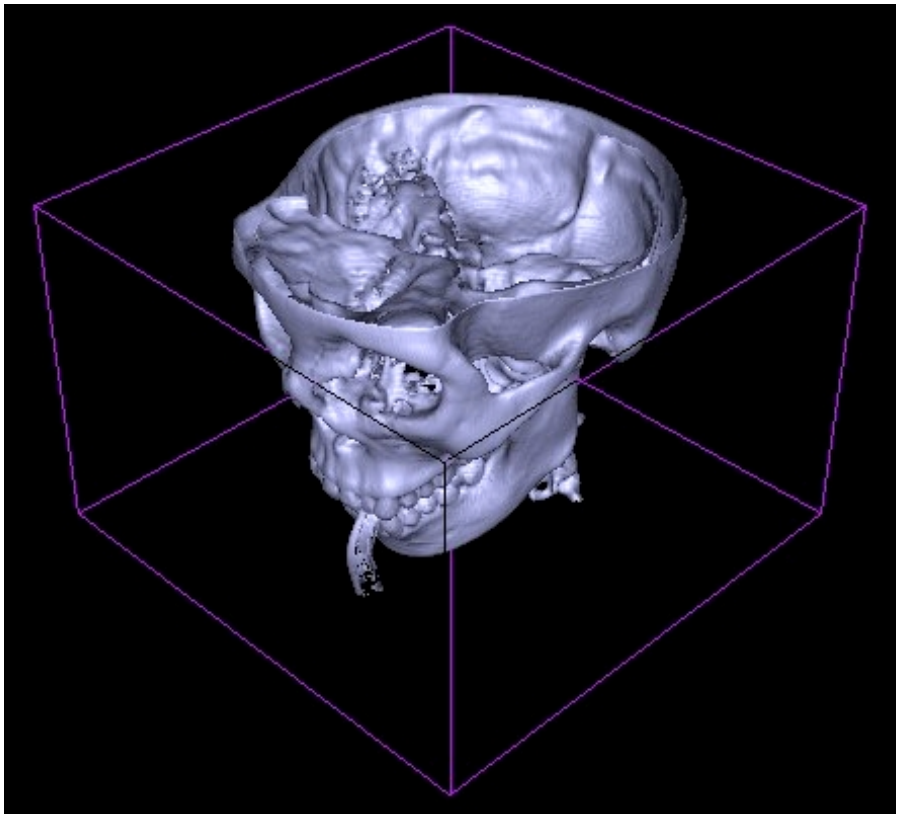
2D Contour line

2D greyscale image

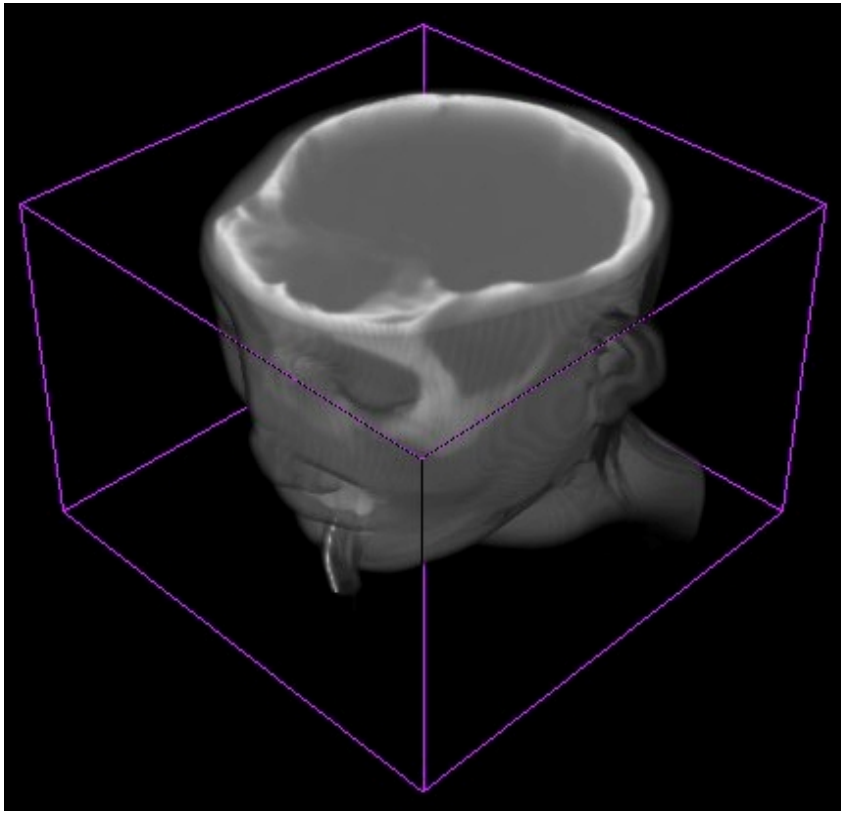


3D Equivalent – Volume Rendering

Shaded iso-surface – surface based rendering



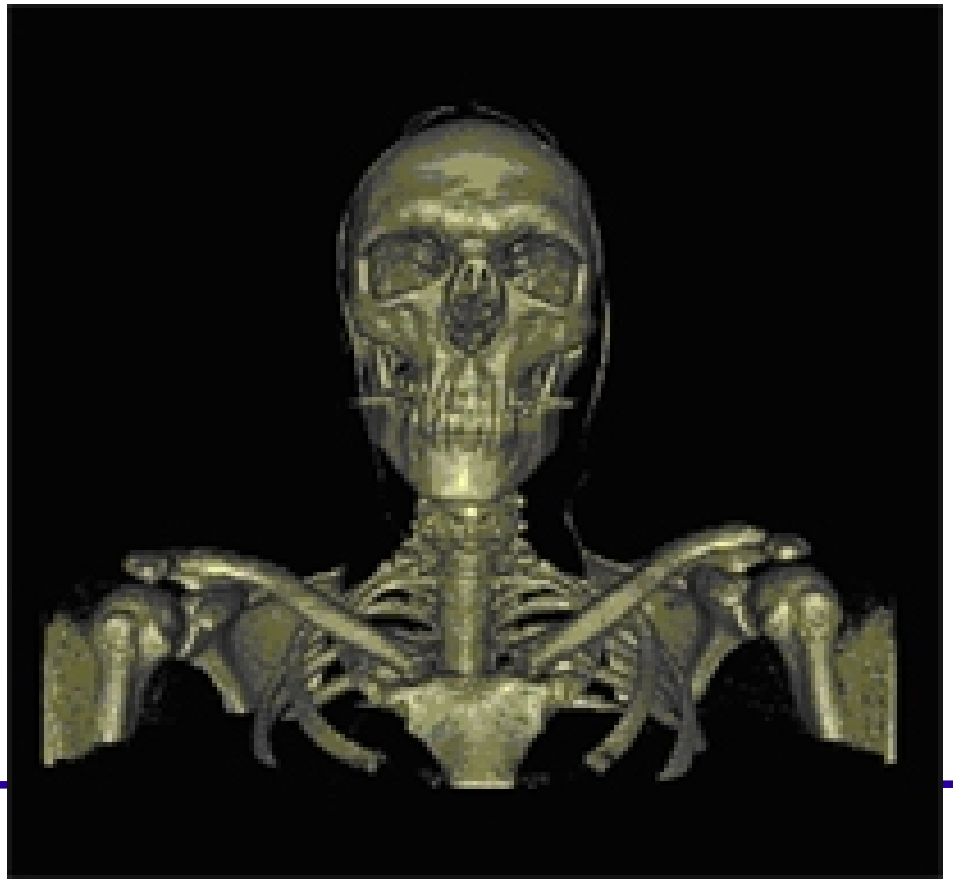
Volume Rendering – voxel based rendering



Data : 3D dense volume;
structured **3D grid of voxels**
regular geometry and topology

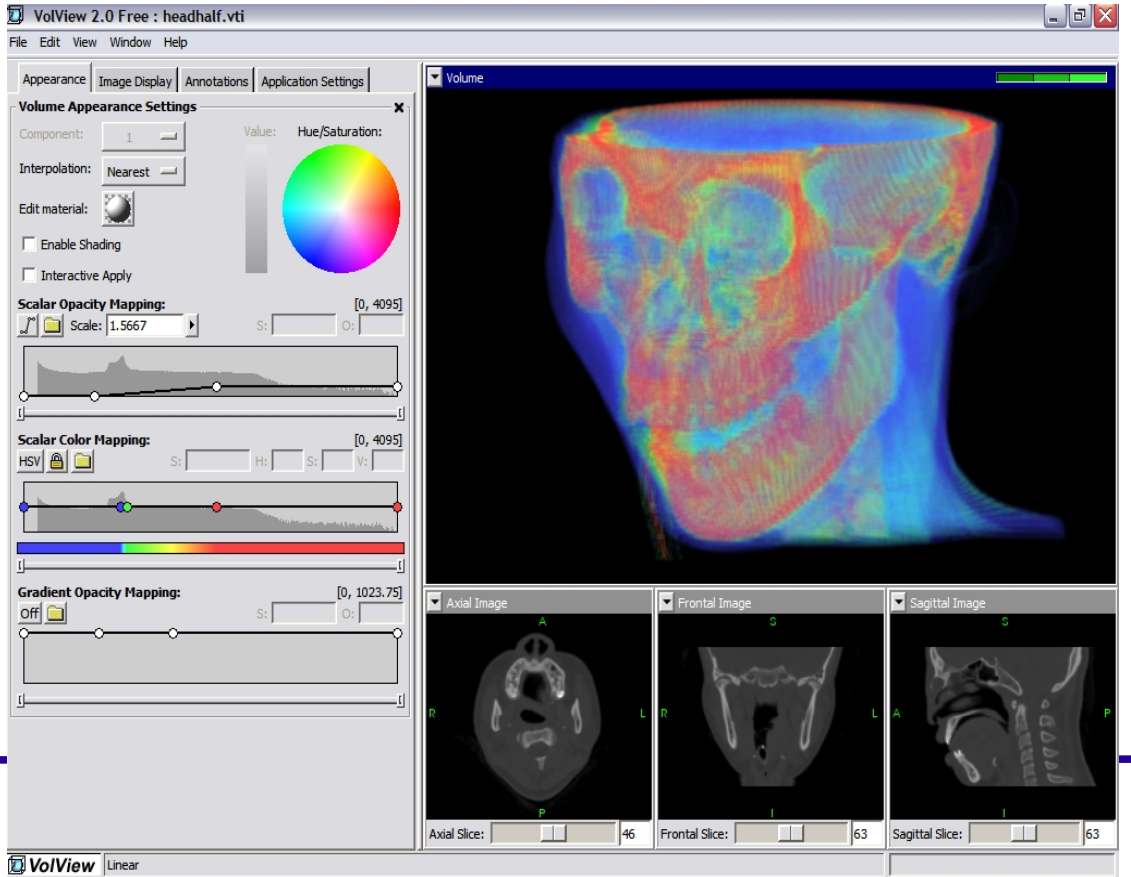
Previous techniques

- concentrated on **visualisation via surface shading**
- **volumetric data → geometric primitives → screen pixels**
- not always possible, or appropriate for 'best' visualisation



Volume Rendering

- display the **information inside the volume**
- direct display : **volumetric data → screen pixels**
- use **transparency** *(seeing through volumes)*



Volume Rendering – Why is it hard?

- **Classification**
  - assignment of **colour and transparency to volume regions**
- **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

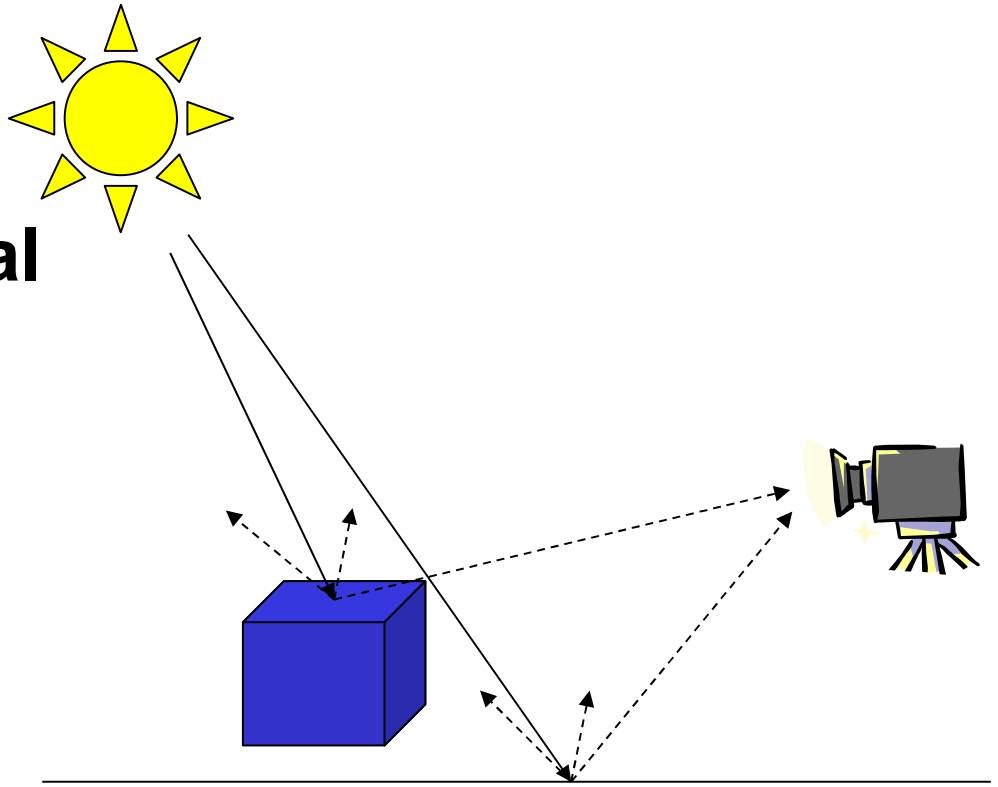
- need to be able to **draw semi-transparent objects**
- **Previously** - only considering drawing **opaque objects**
  - where surfaces reflect, or absorb light, but **no light is transmitted through the object**
- Transparent Vs. Opaque
  - **transparency** : the property of **transmitting light**
  - **opacity** : the property of **not transmitting light**
- Transparency : - frequently referred to as *Alpha* or *Opacity* in CG

Previously : surfaces only

So far .... **light reflected from surfaces**

For volumes we need to consider how **light behaves inside a material**

- i.e. objects that transmit light



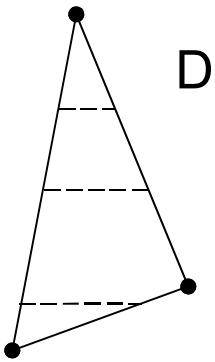
Volume Rendering Techniques

- **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**

- ***Purpose :*** *effectively convey information in the volume data*

Reminder : Object Order Rendering

- **Traditional approach** in Graphics
  - Scanline, or object order rendering :
    - **Transform** and project the polygon into 3D space
    - **Determine colour** of each polygon by lighting equations
    - **Draw the polygon**, checking to see if anything in front of it has already been drawn



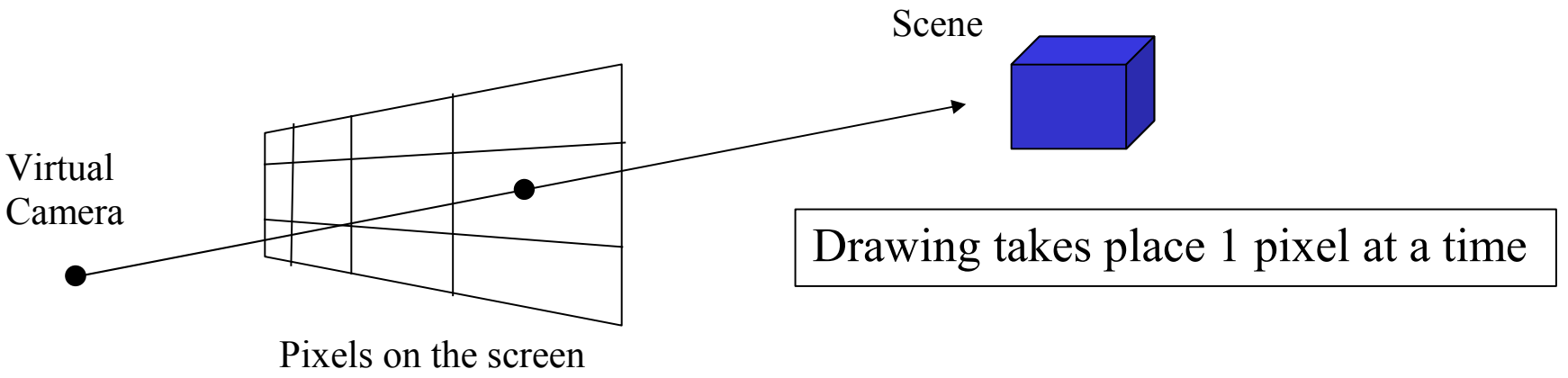
Draw a triangle at a time in rows of pixels, or *scanlines*.

Drawing takes place 1 object at a time
(in Volume Rendering voxels = objects)

  - *From the **3D scene to 2D image***

Reminder : Image Order Rendering

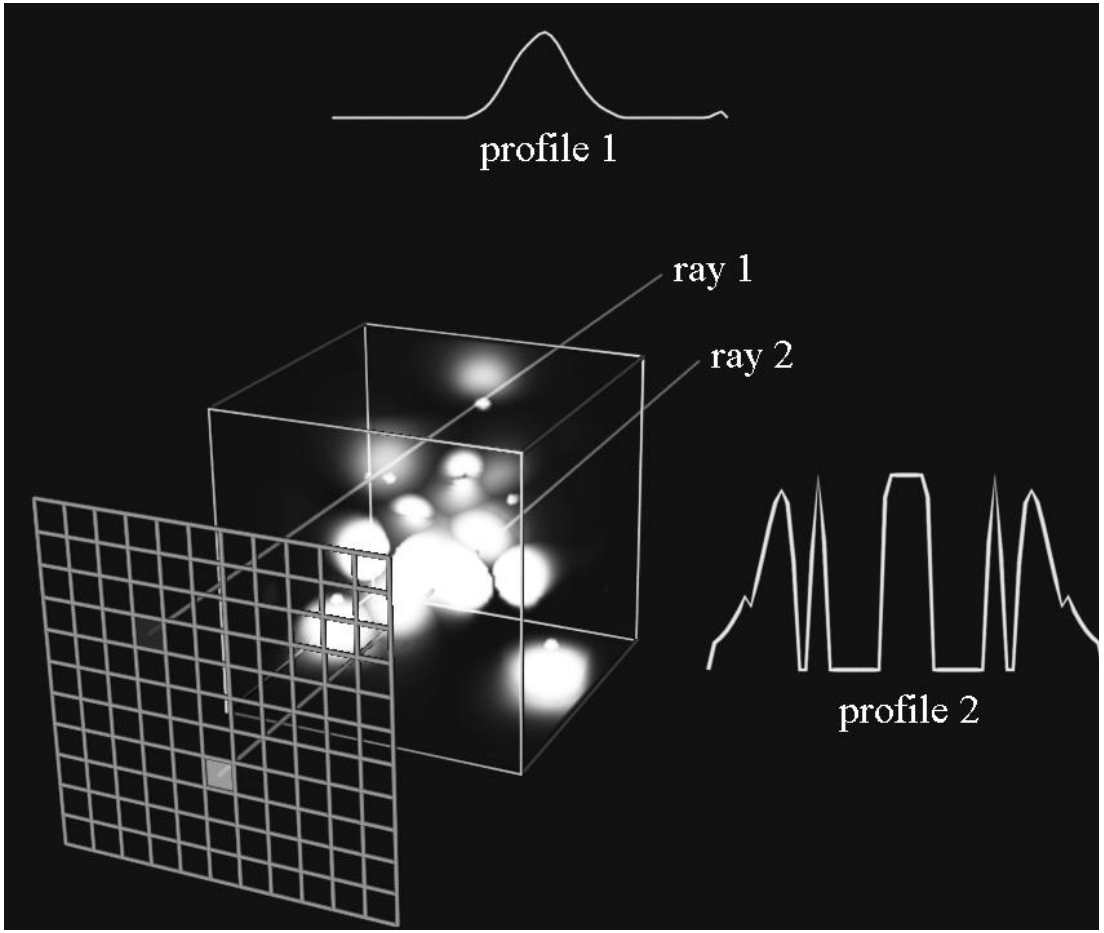
- Also known as *Ray-tracing,* or ***Ray casting***
- For traditional Graphics scene
  - Project an **imaginary ray** from the centre of projection (the viewers eye) **through the centre of each 2D image pixel into the 3D scene.**
  - **Find the point** on the object this **ray intersects**
  - **Calculate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



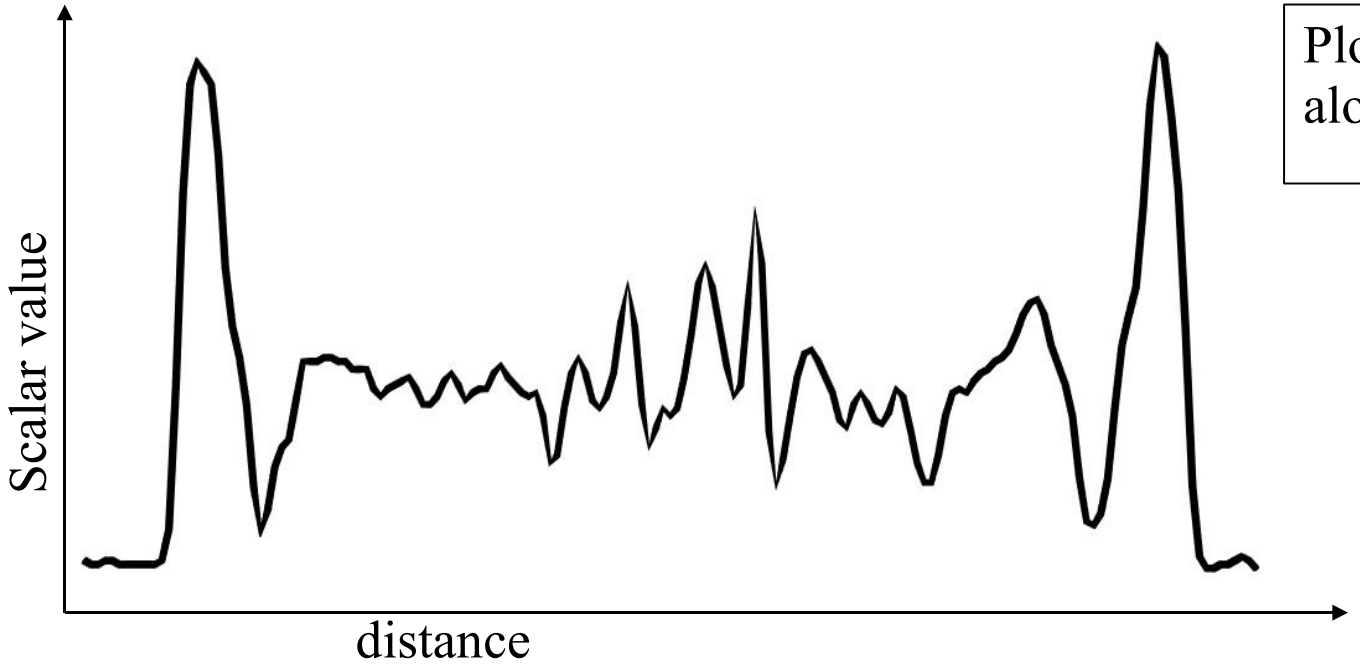
  - *From **2D image into 3D scene*** (i.e. volume)

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?
  - **intensity transfer function**

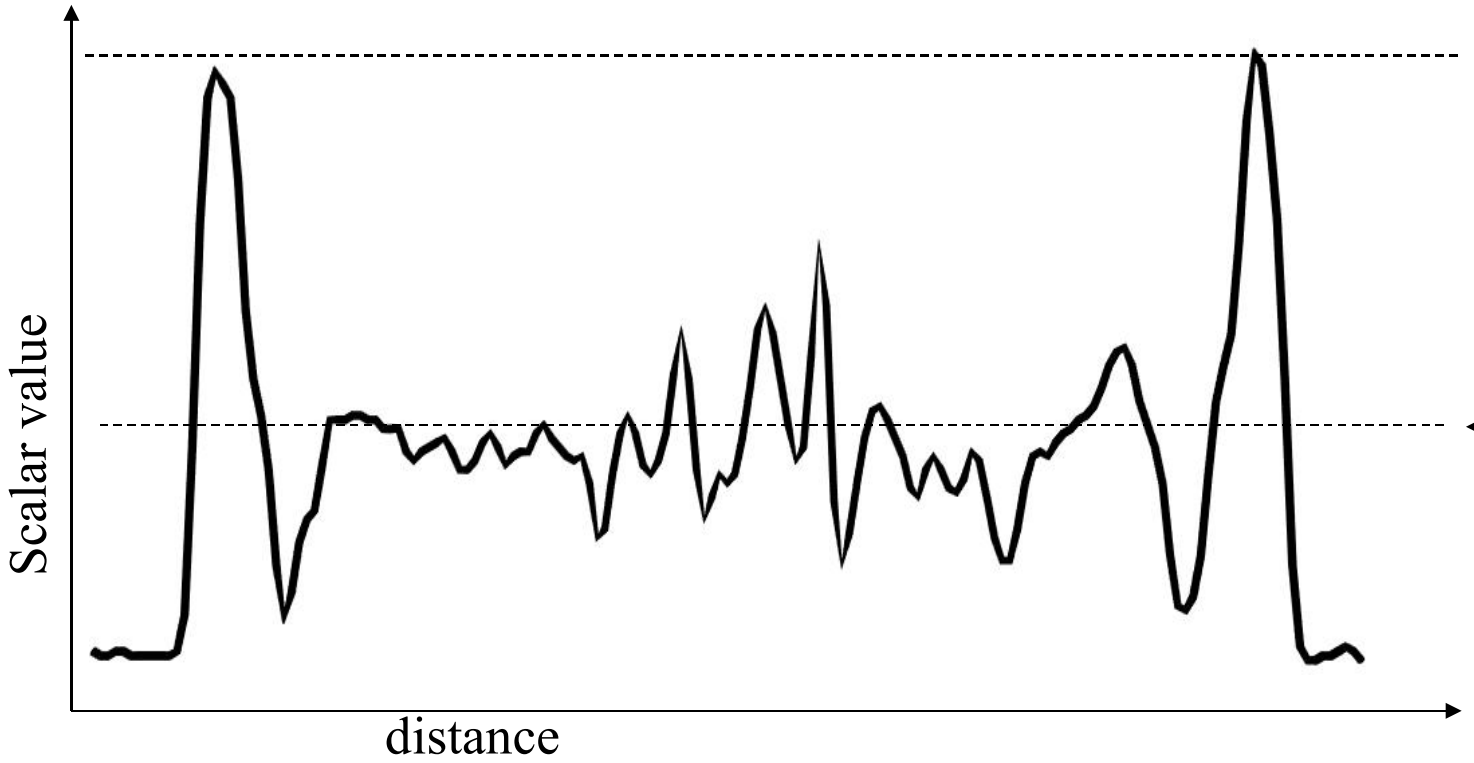


Q: What function to use ?



- **Need** : a lighting model for volumetric data
- **Assume** : voxels emit light proportional to their scalar value
  - similar to greyscale image *(no colour at the moment)*

A : Options for blending

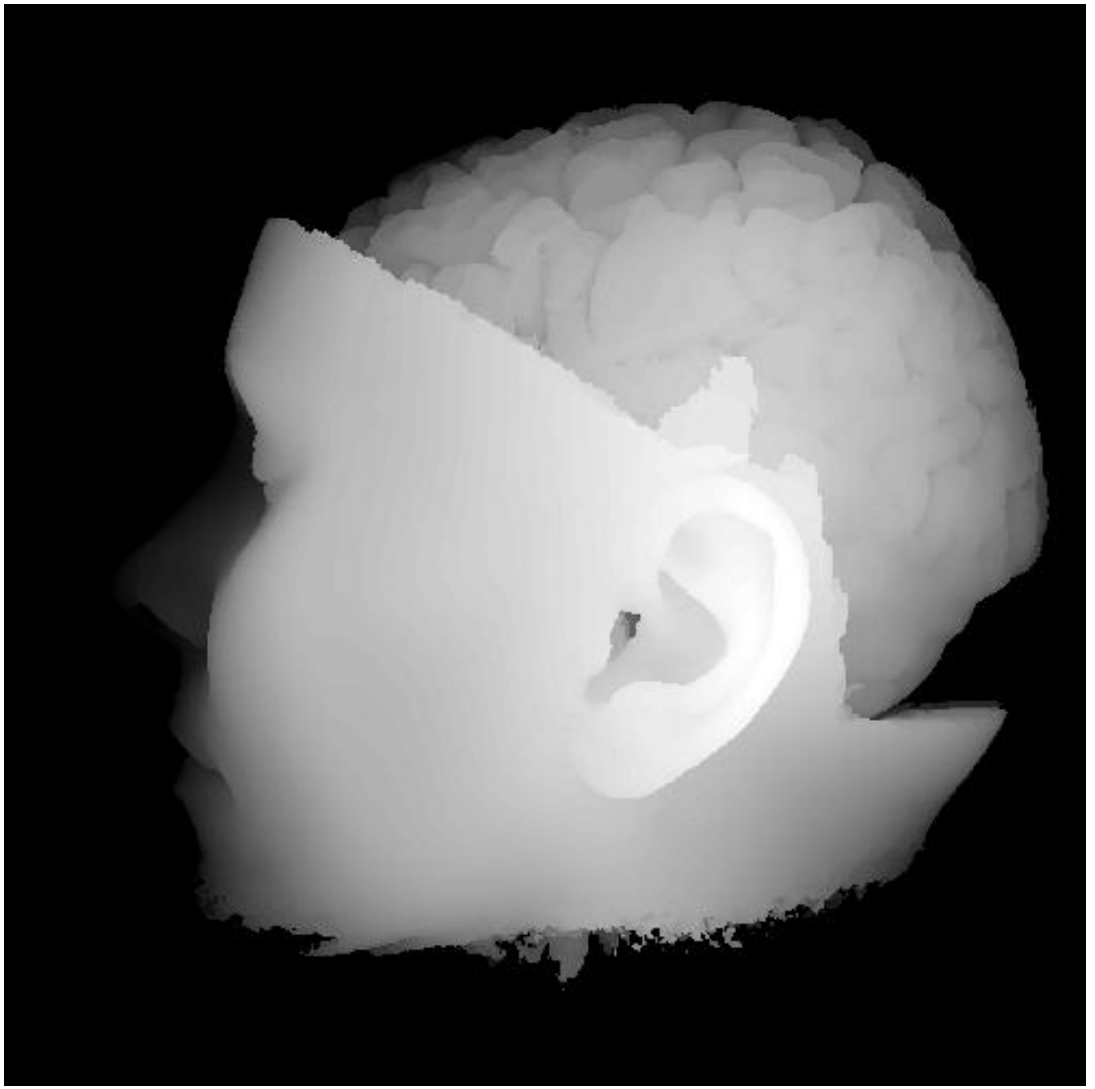


Maximum value

'Composite' Value

- assume the transparency of the cell is proportional to the value.

Example : 3D head volume

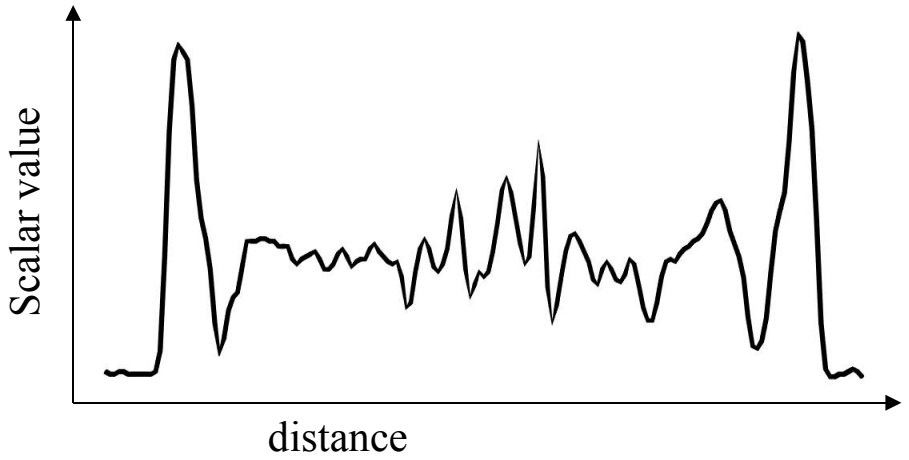


Rendered with no transparency : intensity ≈ distance

Option 1 : Composite Value

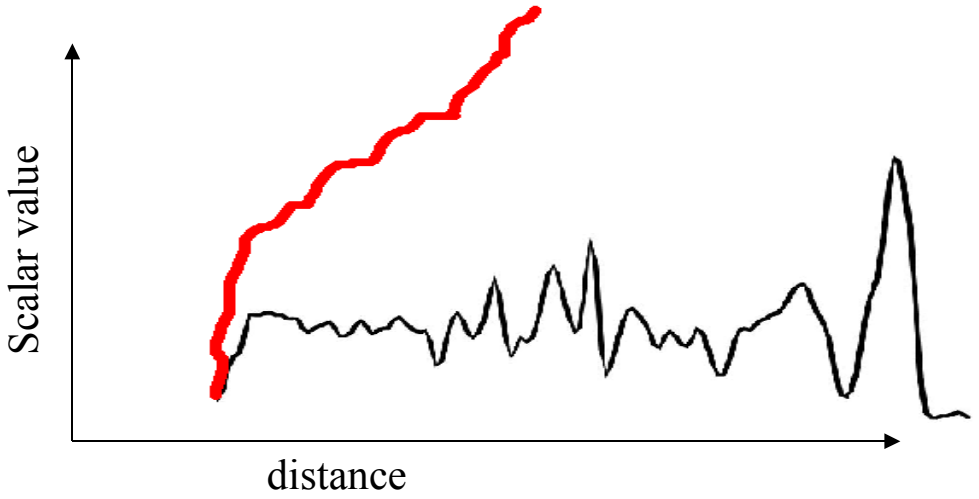
- Value accumulated along the ray
- treating scalar values as alpha transparency value at each voxel

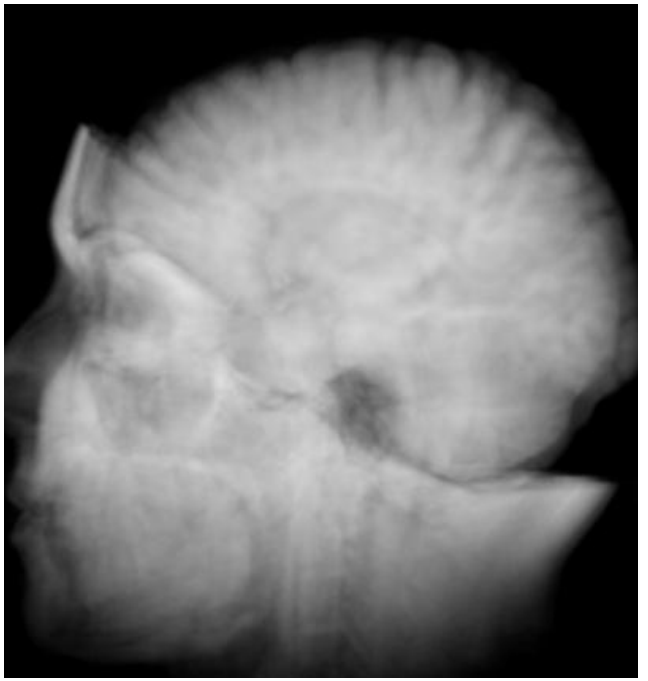
(=> more bright = more visible)



Composite Intensity Projection

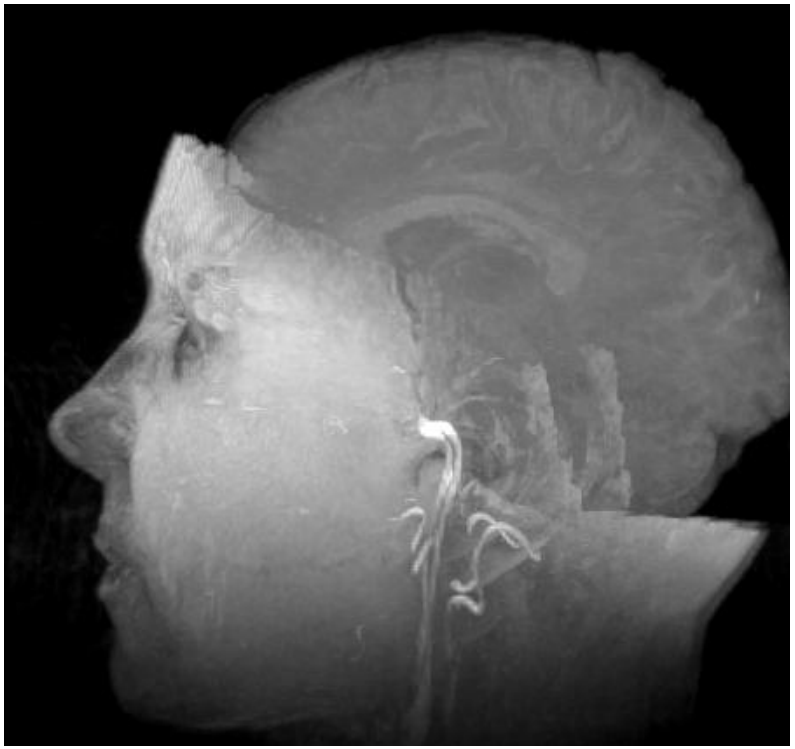
- Intensity = scalar value accumulated along the ray
  - treating scalar values as alpha transparency at each voxel

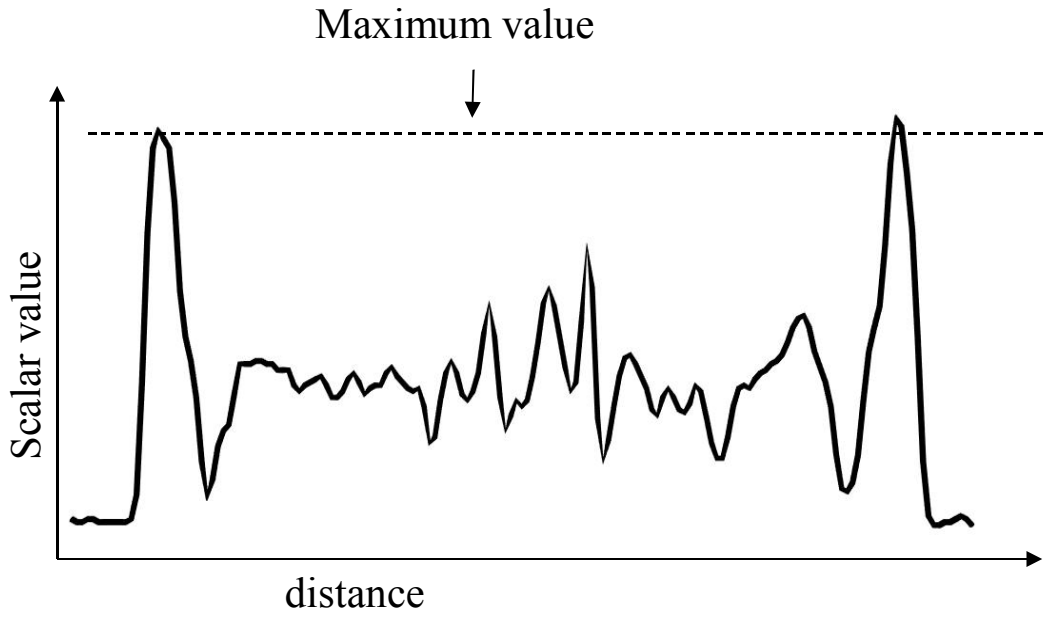




Option 2 : Maximum Value

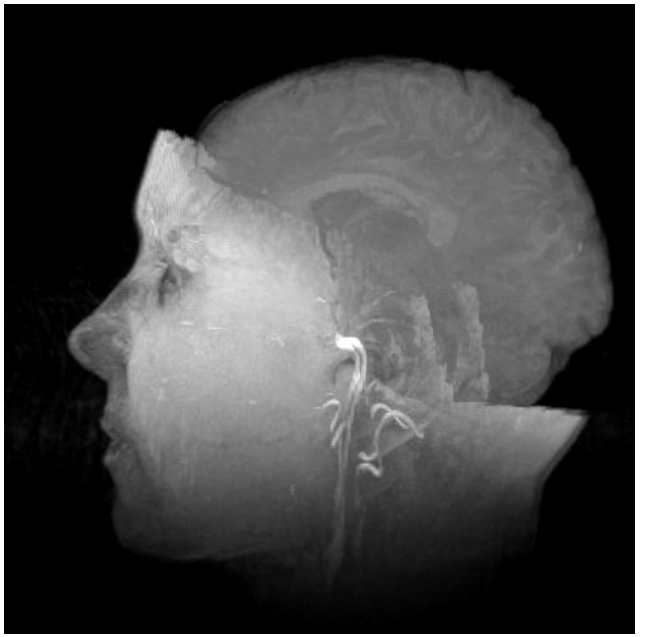
- No blending or transparency
  - intensity ≈ max. scalar value encountered on ray
  - **maximum intensity projection**
  -





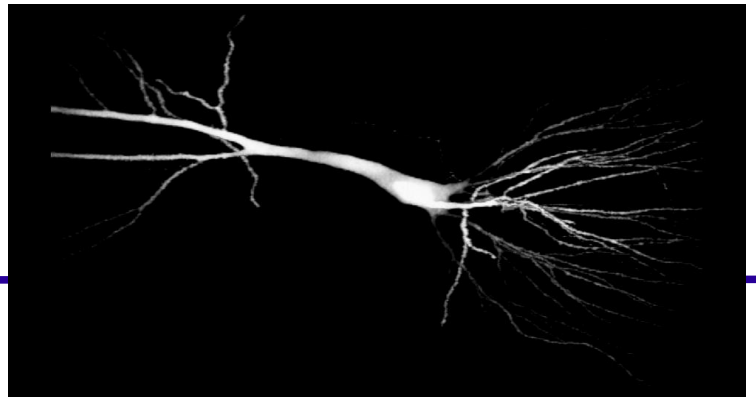
Maximum Intensity Projection (MIP)

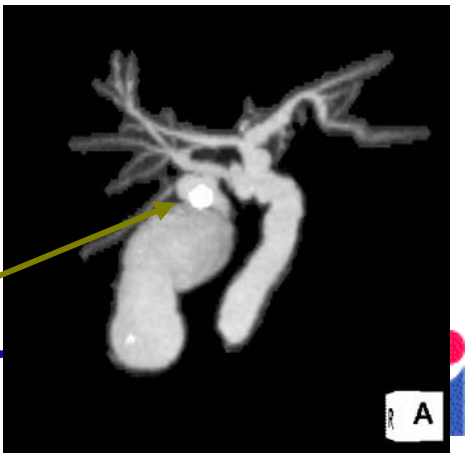
- **Simplest method of Image Order Volume Rendering**
  - Pixel values ≈ maximum voxel values
  - with CT data, resembles a conventional X-ray image
  - **Useful for finding some unusual objects**
  - **Limitation** : perception of depth
    e.g. front/back ambiguity



MIP rendering of a nerve cell.

• Which branches are in front, and which behind ?

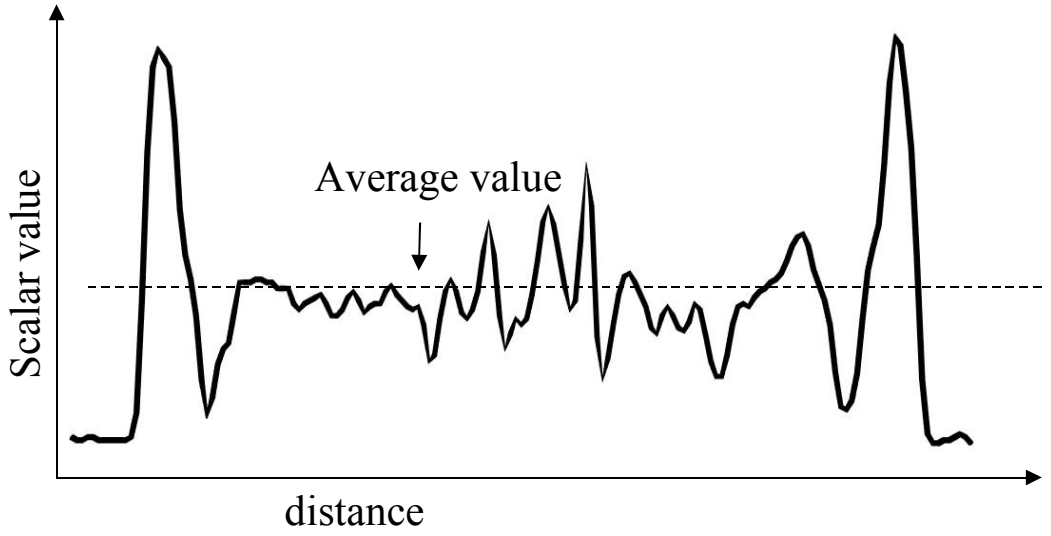


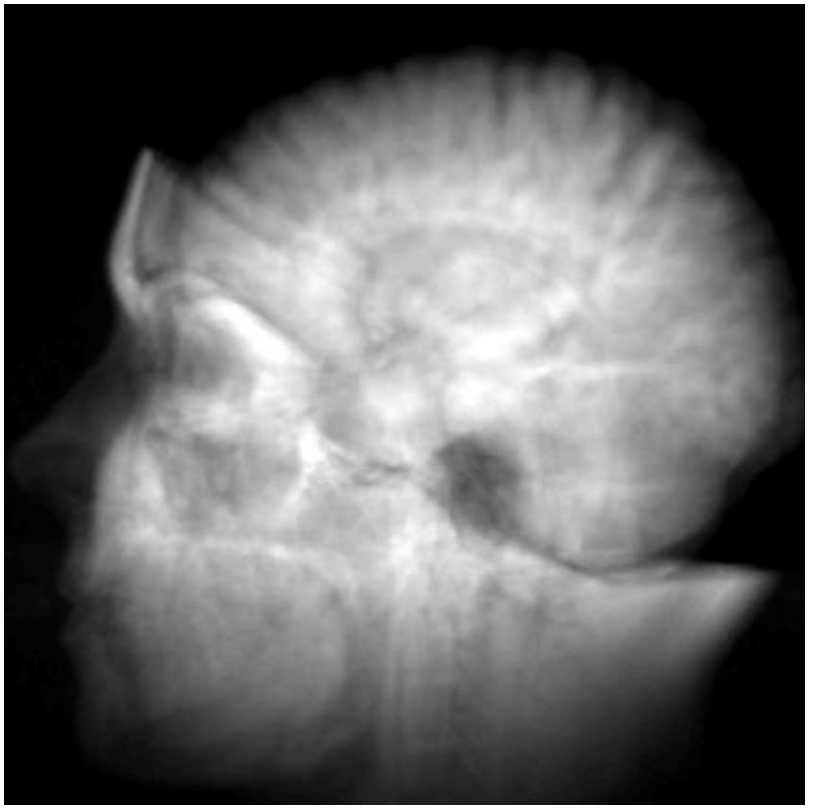


Bilestone in the cholecytis

Option 3 : Average Value

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





How to decide the transparency & brightness of the voxels

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

Some easy solution :

setting them proportional to the scalar value

Rendering with Transparency 1

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

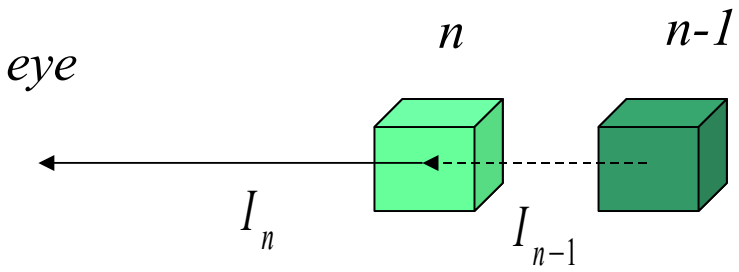
- **multiply brightness by opacity** (0 = transparent → 1 = opaque)
- If cell is **totally transparent, cannot see cell brightness**
- If cell is **totally opaque, can see all of cell brightness**

A_n: opacity, E_n: brightness

Rendering with Transparency 2

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

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

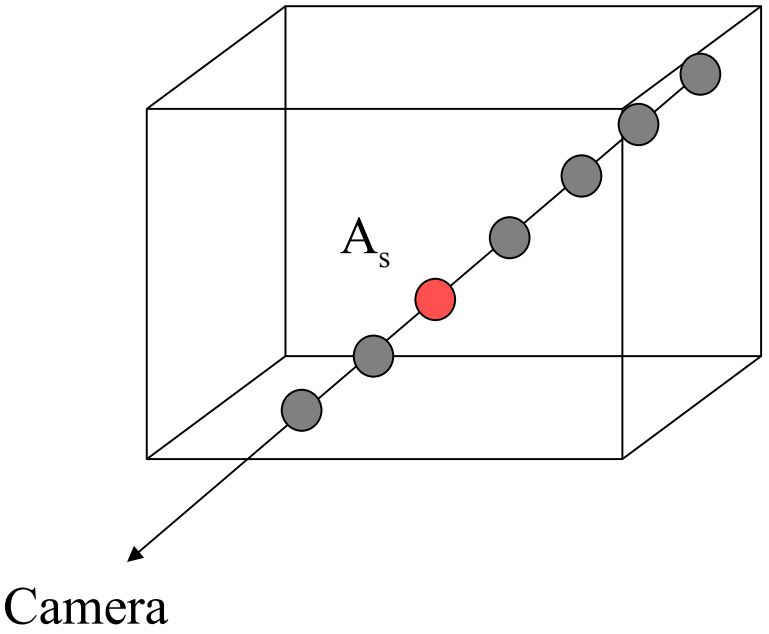


Subscript *n* refers to cell n.

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

Which way to cast the ray ?

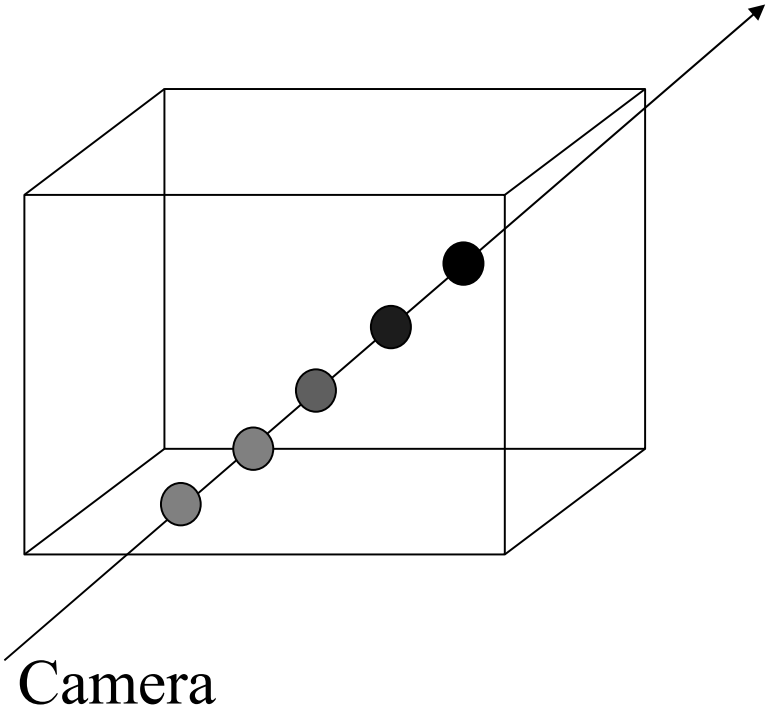


$$I = A_s I_s + (1 - A_s) I_b$$

Accumulate values **starting at furthest voxel and work towards camera** – transparency correctly handled

- **Principle** : *evaluate brightness behind voxel and blend using opacity of current voxel*
- ***Compositing*** *: convention back-to-front*

Forward casting of ray



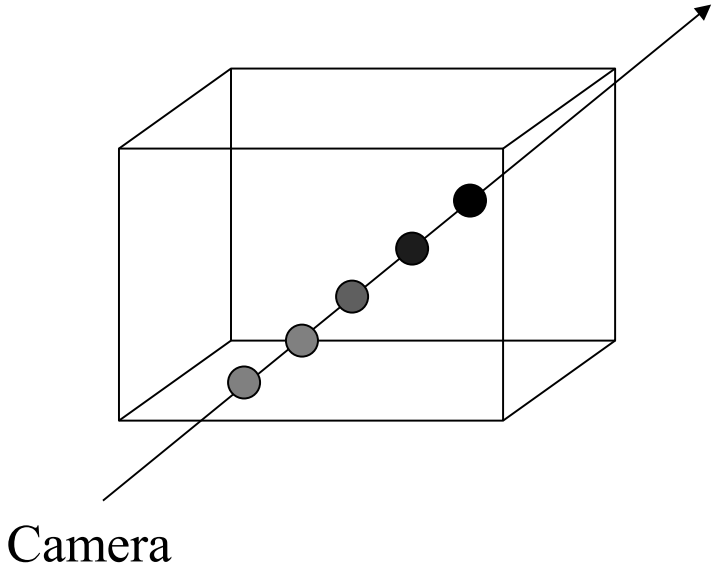
If we **use a buffer to store current alpha, we can accumulate starting at the nearest cell instead.**

-Supports early termination if opacity approaches 1.0

alpha = opacity value

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

Early termination of ray

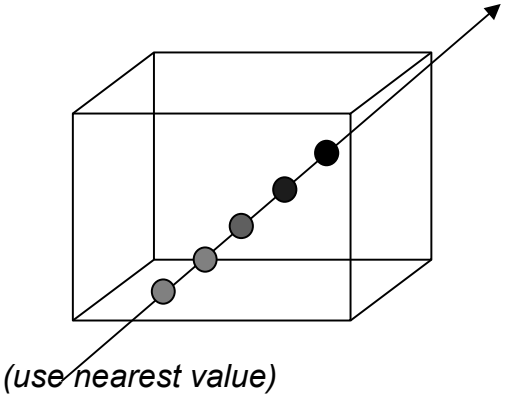


Current Alpha	Accumulated Alpha
0.2	0.2
0.4	0.4+(1- 0.4)*0.2=0.52
0.5	0.5+(1- 0.5)*0.52=0.76
0.4	0.4+(1- 0.4)*0.76=0.86
0.6	0.6+(1- 0.6)*0.86=0.94

- 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

Image Ordered Trade-offs

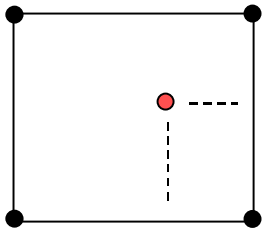
- **Interpolation** method
  - *ray intersects cells not points*
  - **linear interpolation** across cell *(expensive)*
  - Shortcut : **nearest-neighbour interpolation**



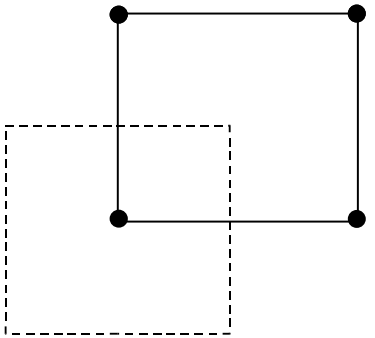
- **Sampling** method
  - **uniform sampling** across voxel gives smooth results
  - Shortcut : **voxel-by-voxel sampling** is faster
- **Step size** (along ray)
  - **larger step size** = less computation = **faster rendering**
  - larger step size gives **more image artefacts**

Interpolation & Sampling

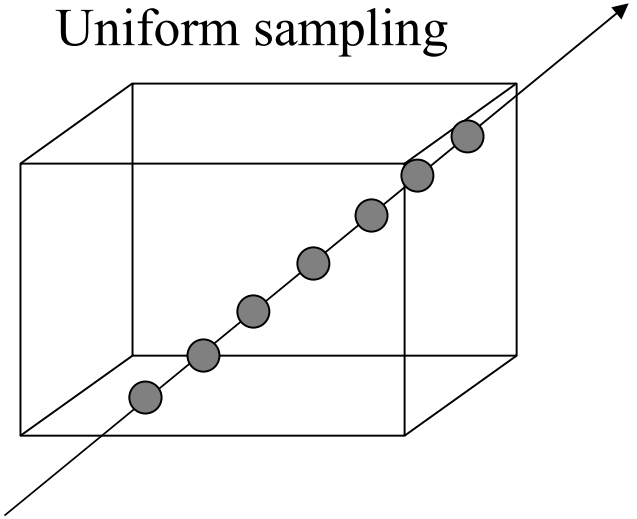
Linear interpolation



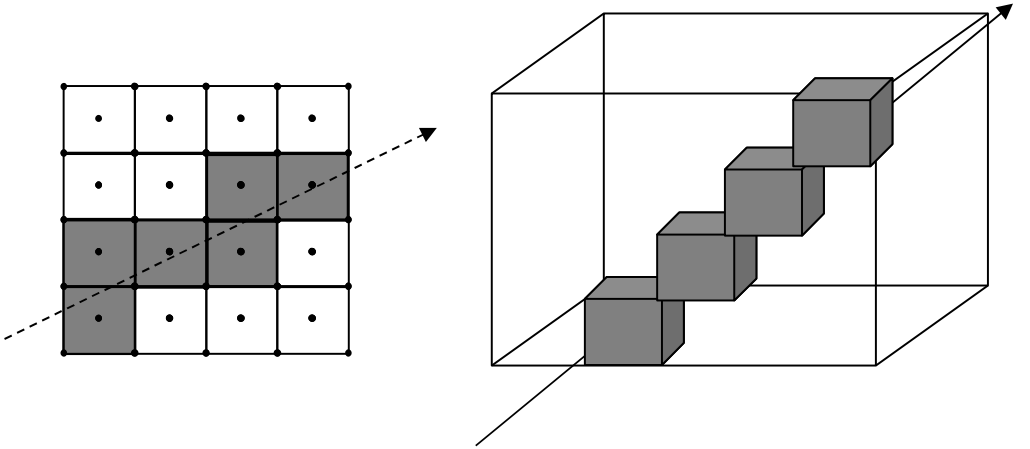
Nearest neighbour interpolation



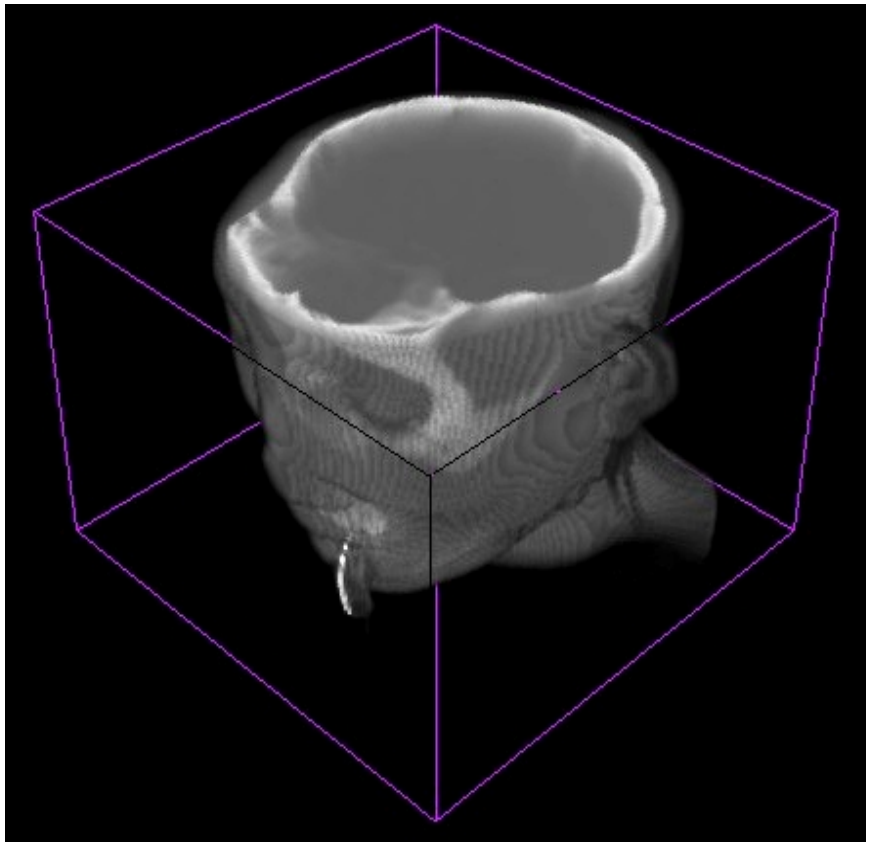
Uniform sampling



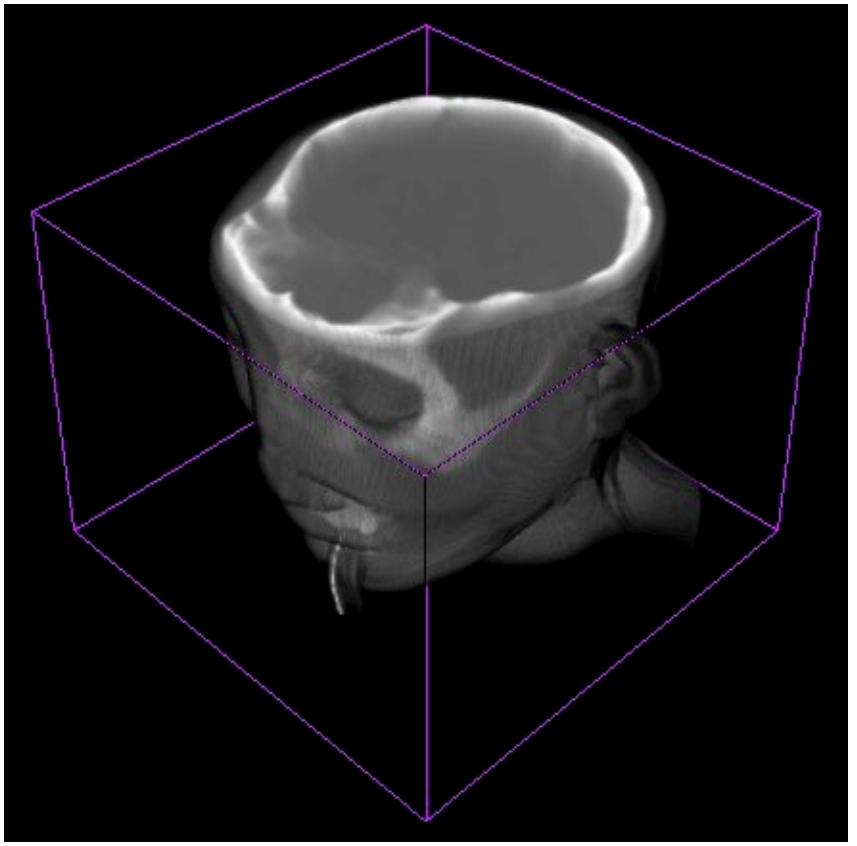
Voxel-by-voxel traversal



Effect of interpolation



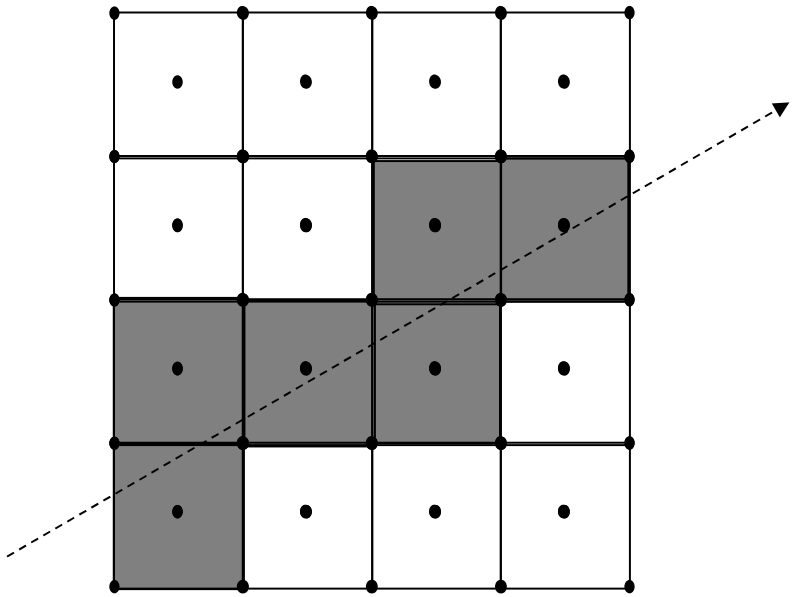
Nearest neighbour
(visible artefacts in rendering)



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

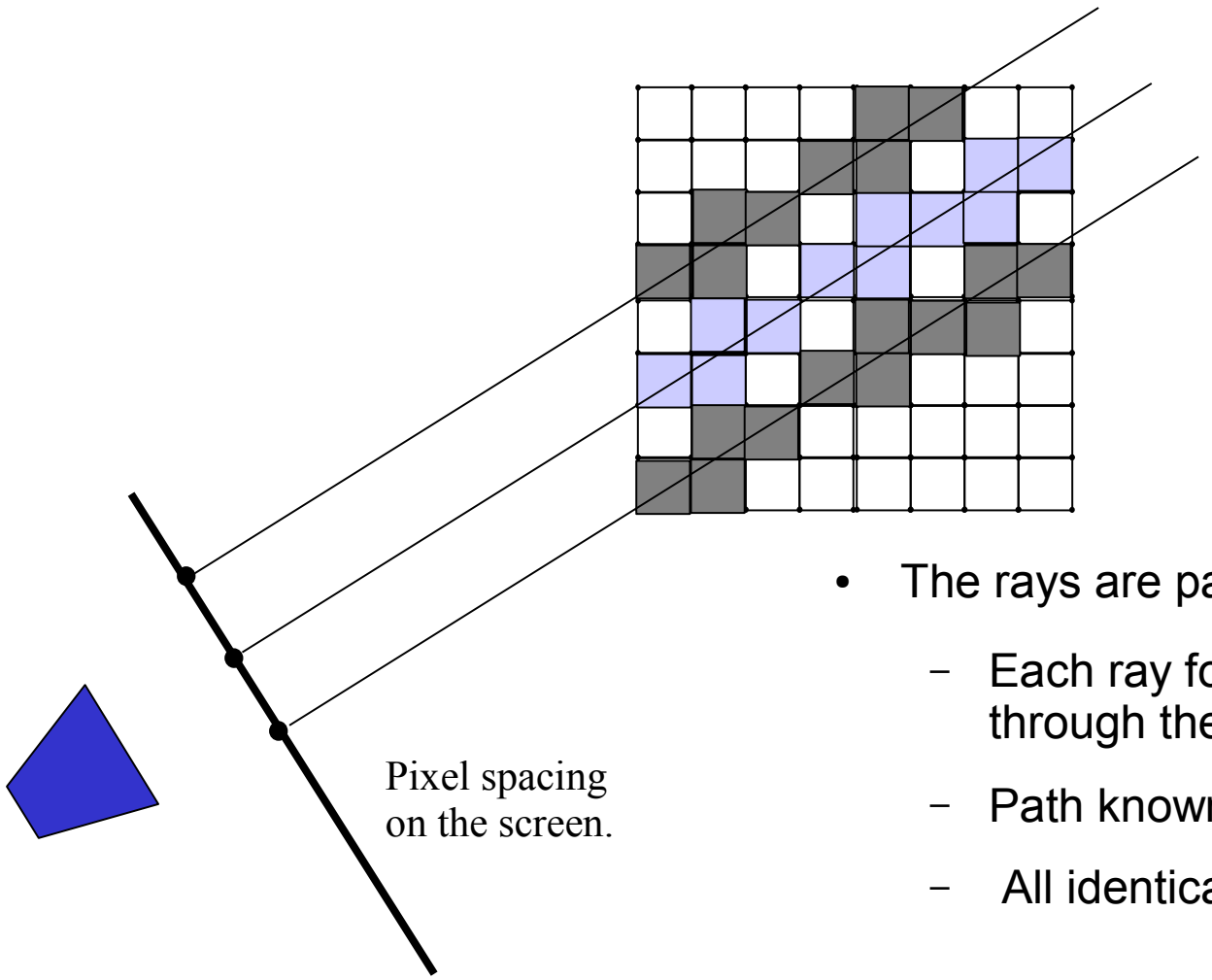
Step = 1.0

Voxel Traversal



We can shift the whole grid by ½ a voxel size, and consider the volume of influence to be the cell.

Voxel Traversal with Templates

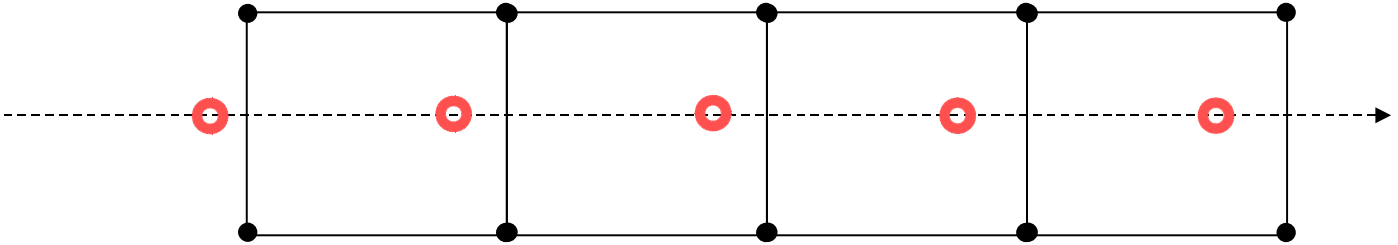


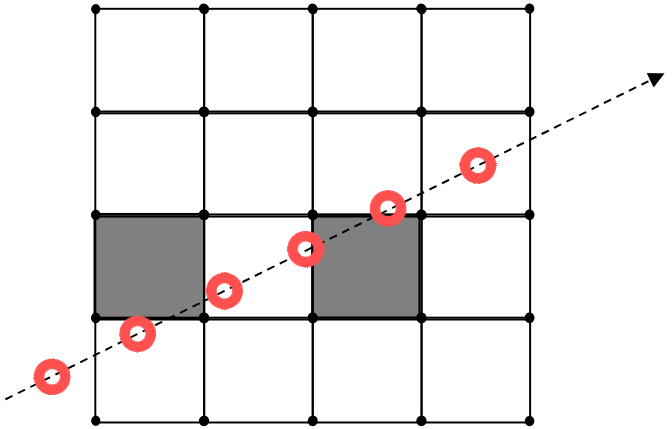
- The rays are parallel
  - Each ray forms an identical path of voxels through the grid
  - Path known as a **template**
  - All identical, pre-compute, save computation

Note : some voxels are missed.

Effect of Step Size 1

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

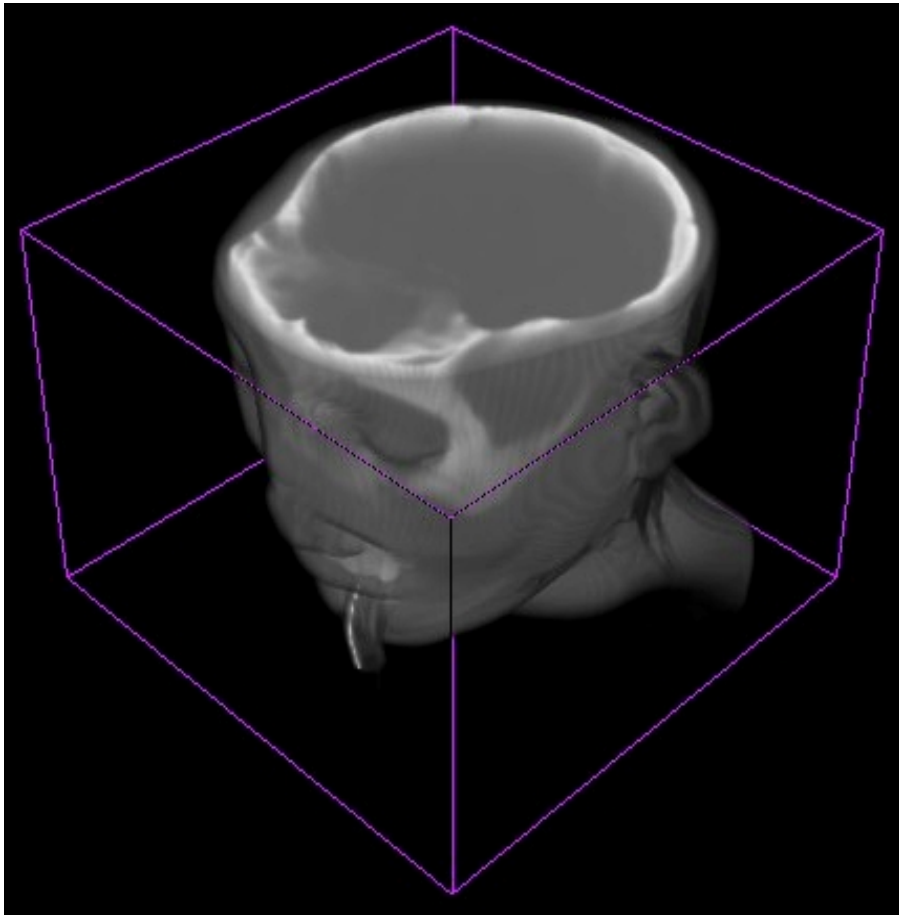




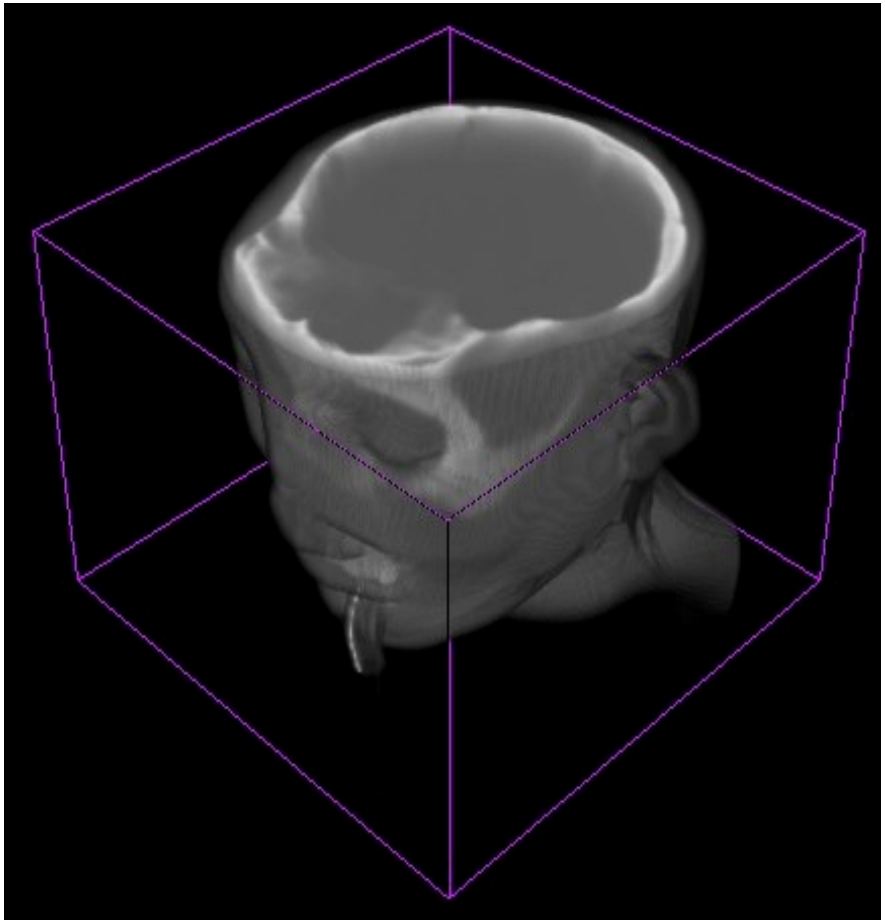
But in the general case, some voxels can be missed due to sampling, even if the spacing is 1 voxel width.

- **Leads to artefacts appearing.**

Effect of Step Size 2

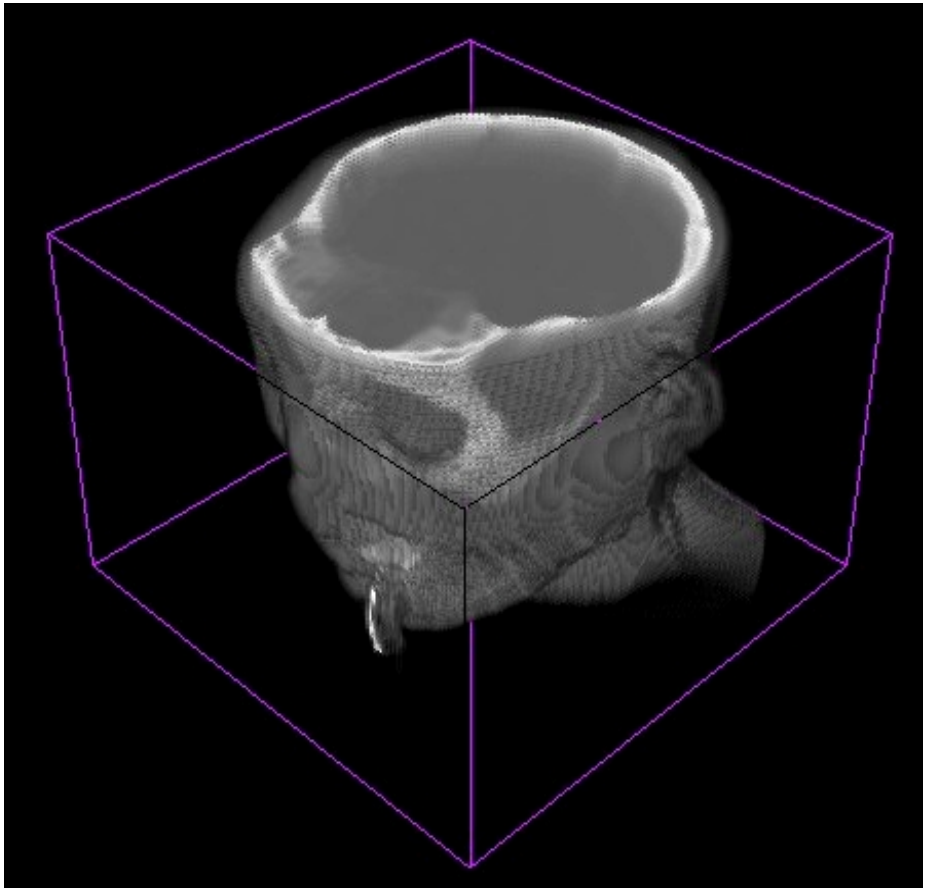


Step = 0.2

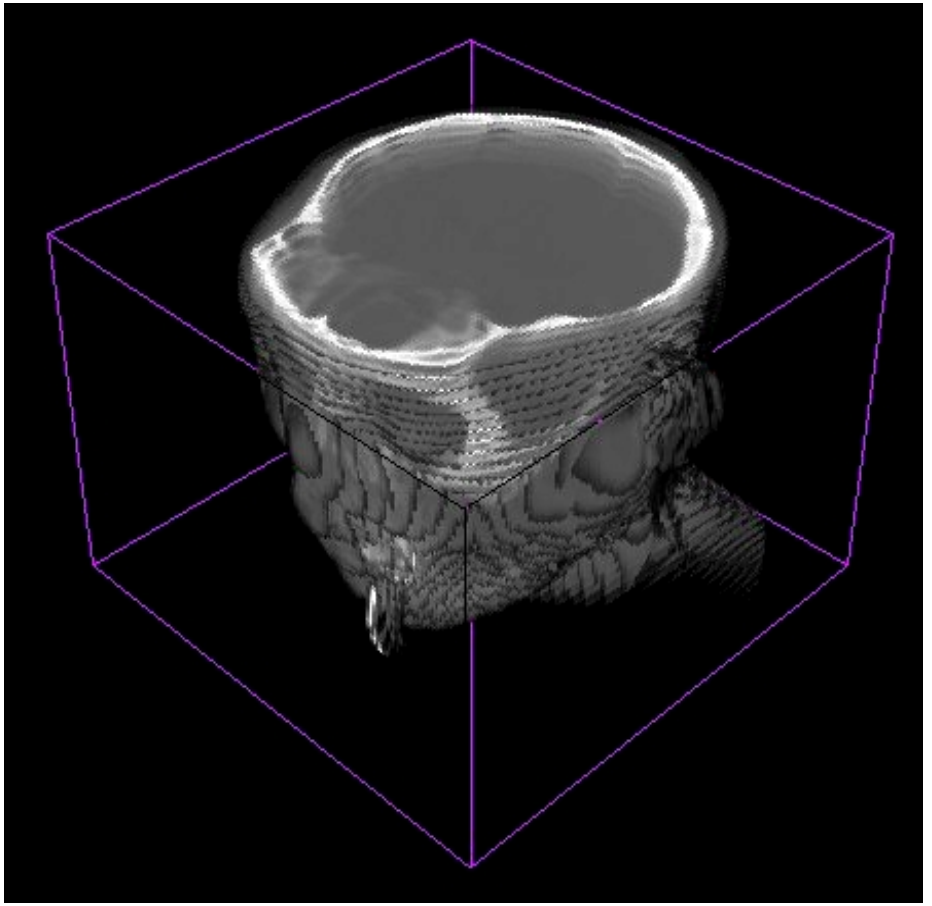


Step = 1.2
(mild artefacts)

Effect of Step Size 3



Step = 3.5
(visible artefacts)



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

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 transer function** (scalar ray profile → opacity)
  - ray casting with **transparency**
  - **trade-offs** in image based : interpolation, sampling, step-size

Next lecture : *Volume Classification / Object Order*