CSE 564
Visualization & Visual Analytics

Medical & Scientific Visualization

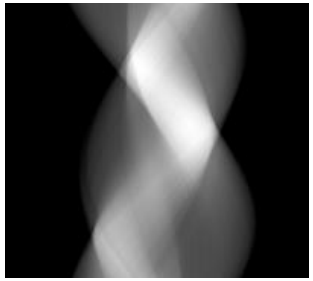
Klaus Mueller

Computer Science Department
Stony Brook University

Medical Imaging: Overall Concept

human (in pain)

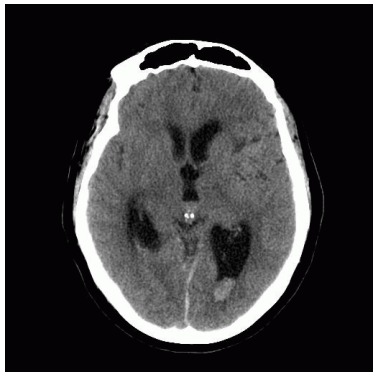
imaging device



data

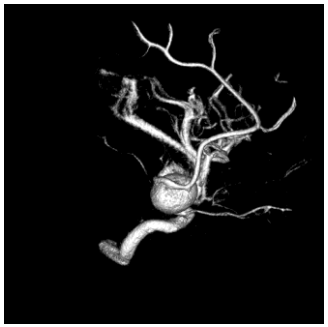
imaging algorithm

reconstructed cross-sectional image



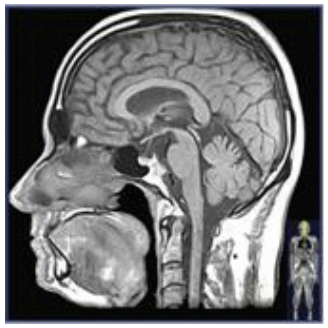
Imaging Modalities Overview

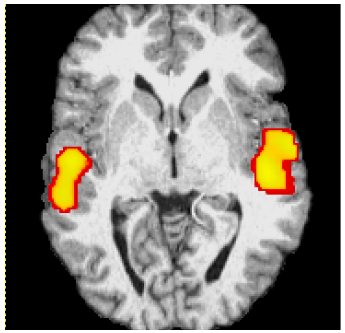
CT



X-ray

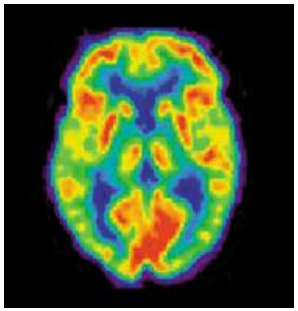
MRI / fMRI

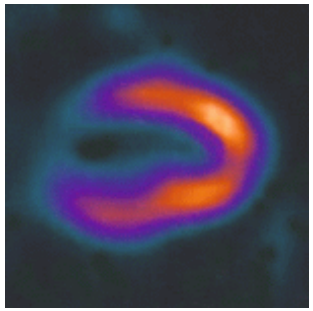




magnetic spin

Nuclear



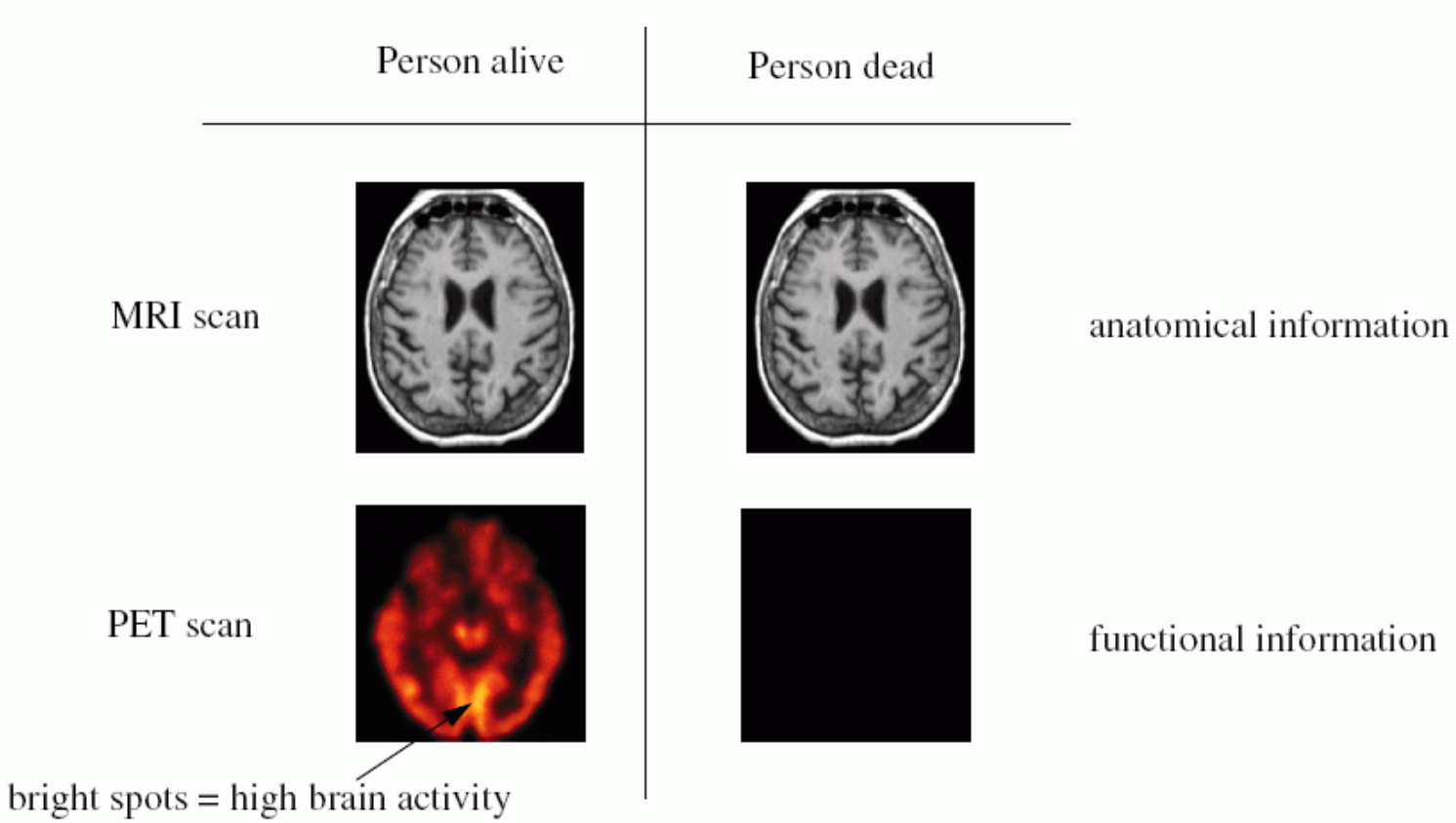


metabolic tracer X-ray emission

Ultrasound

sound waves

Anatomic vs Functional Imaging



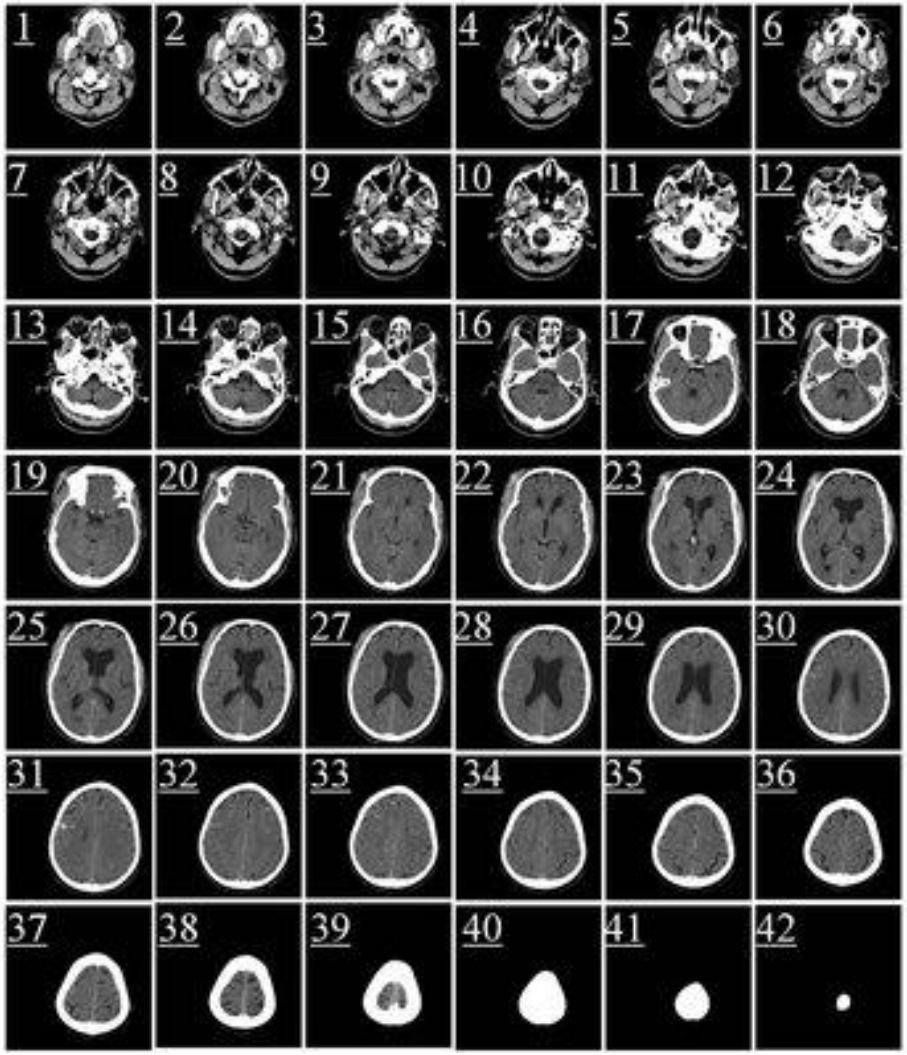
An MRI scan shows you that you have a brain

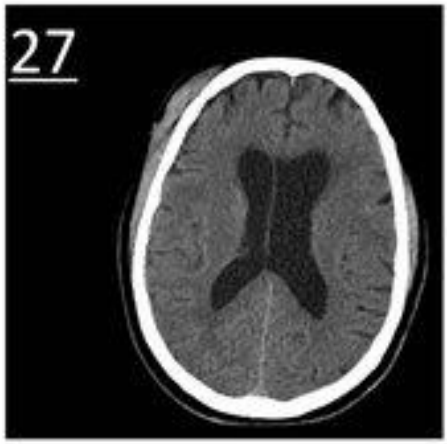
A PET scan shows that you use it

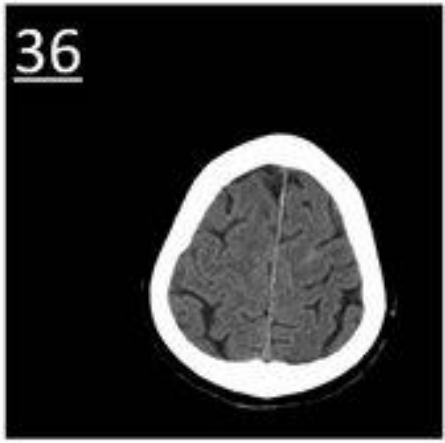
Reviewing Radiographs

Would 3D visualization help?

Slice Matrix







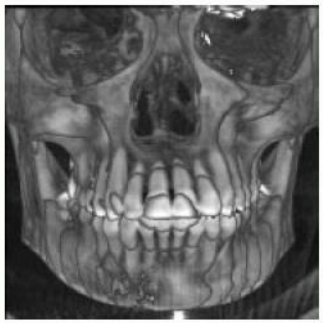
Would 3D visualization help?

3D Visualization via Volume Rendering

Reconstructed object enables:

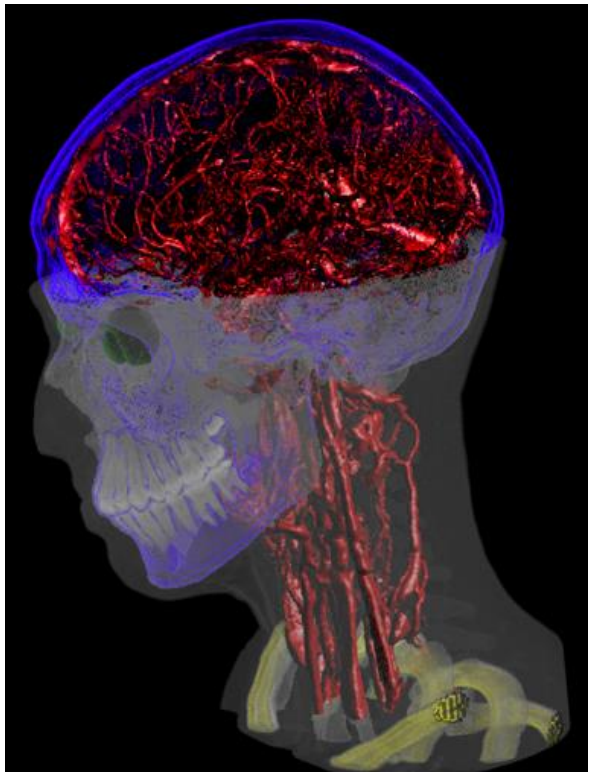
- Enhanced X-ray visualization from novel views:


- Maximum Intensity (MIP) visualization:

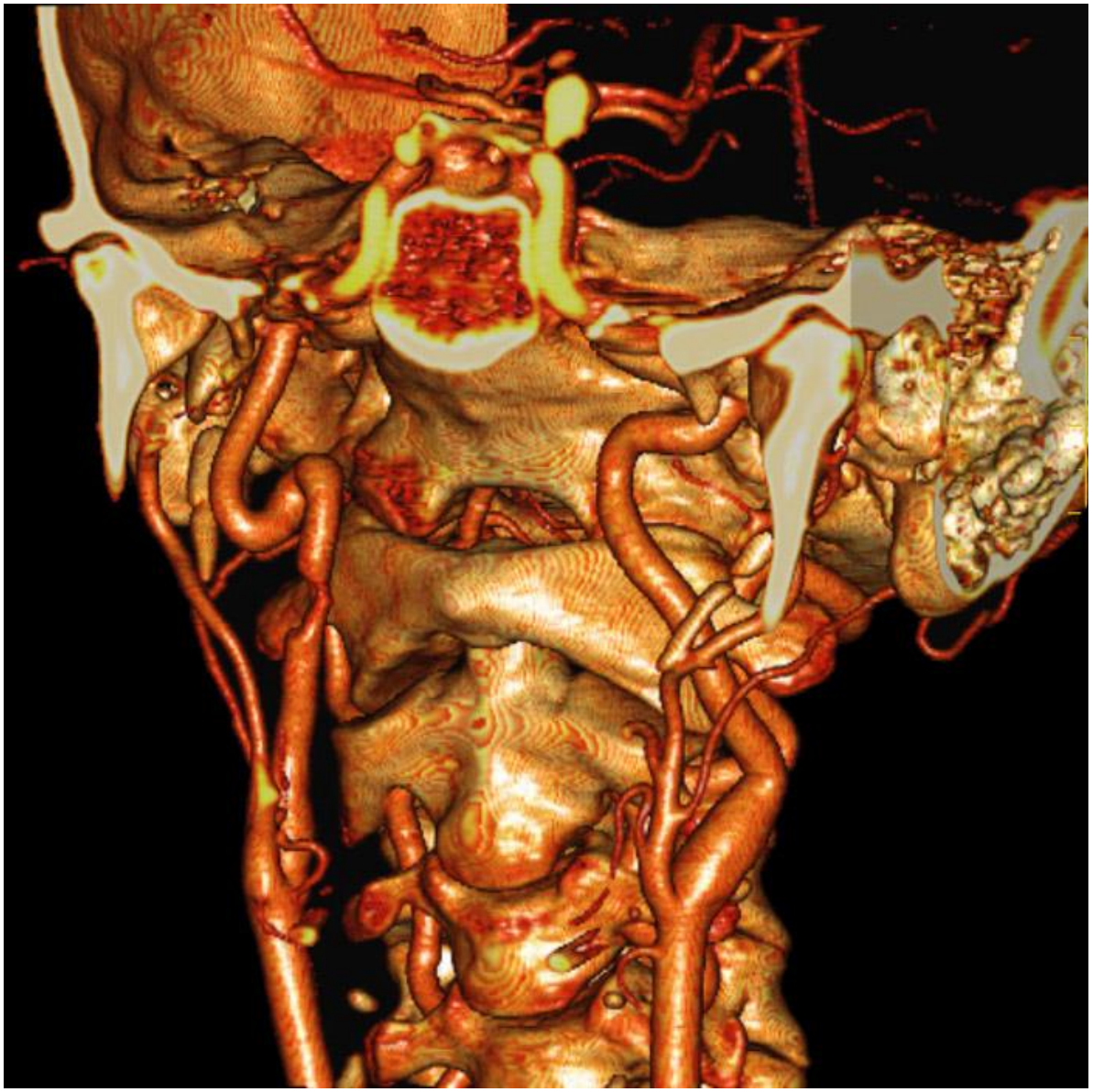

- Shaded object display:



Aortic Stent and Arterial Vessels

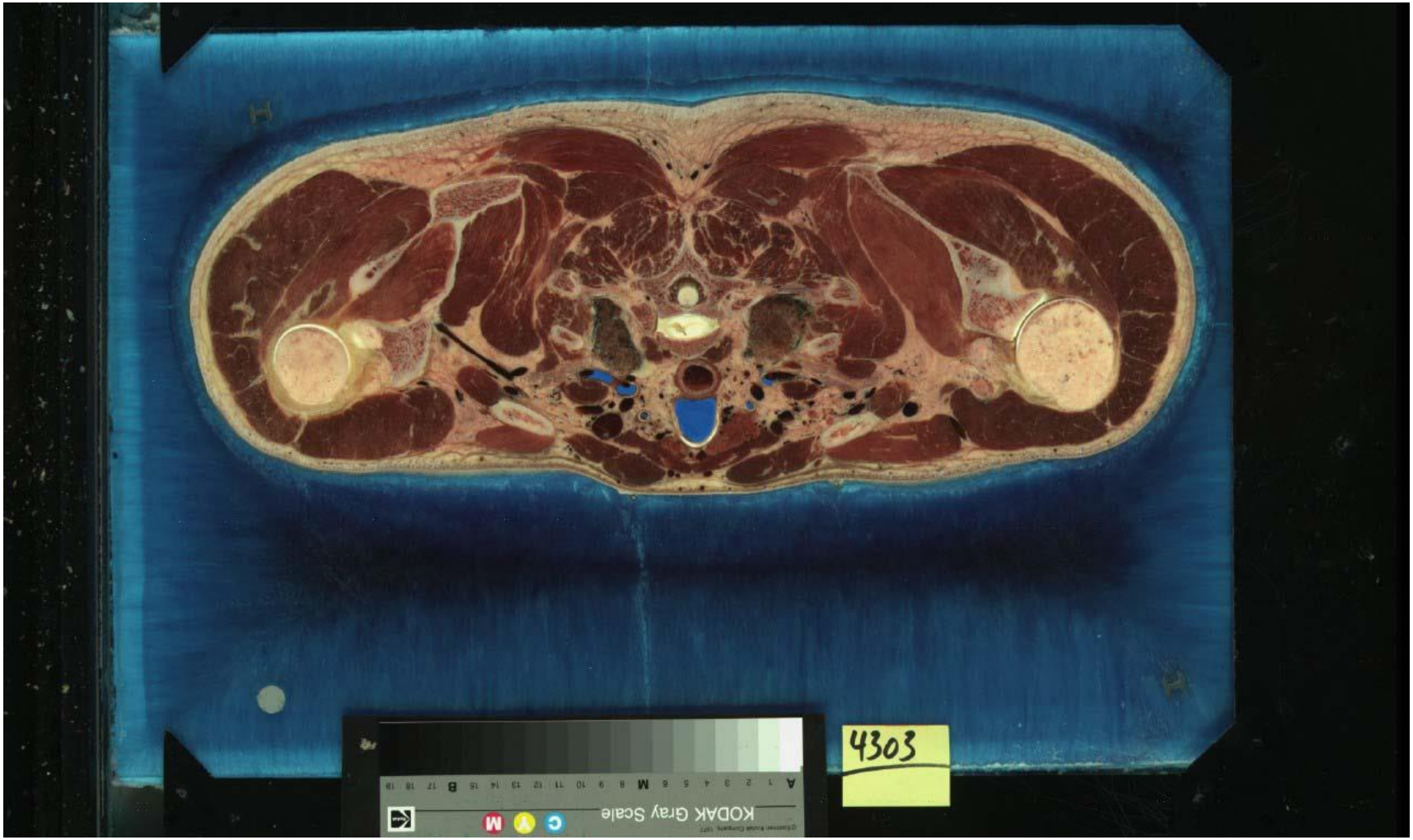
Cartotid Stenosis



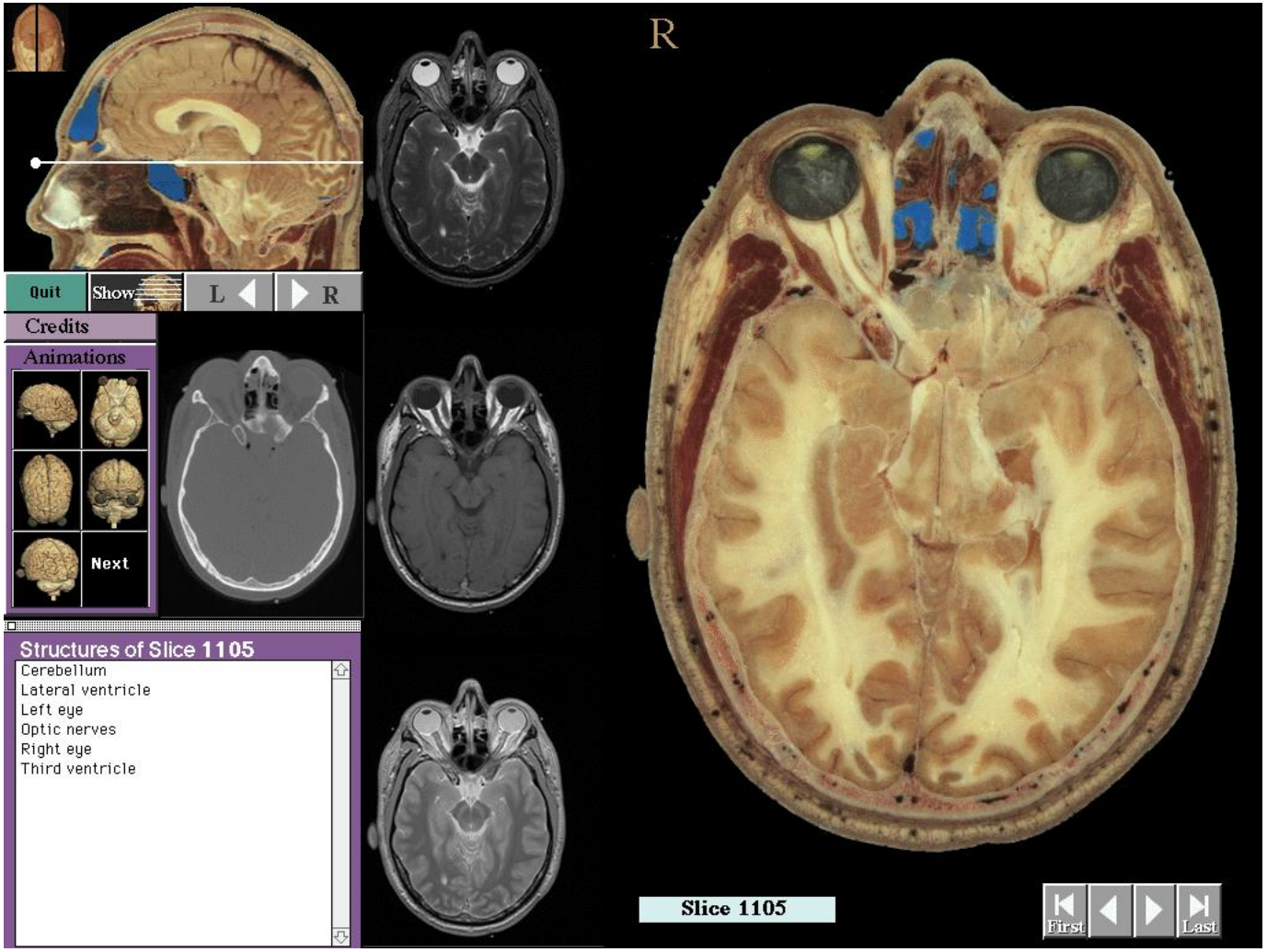
Virtual Colonoscopy

Virtual endoscopy, arthroscopy, etc.

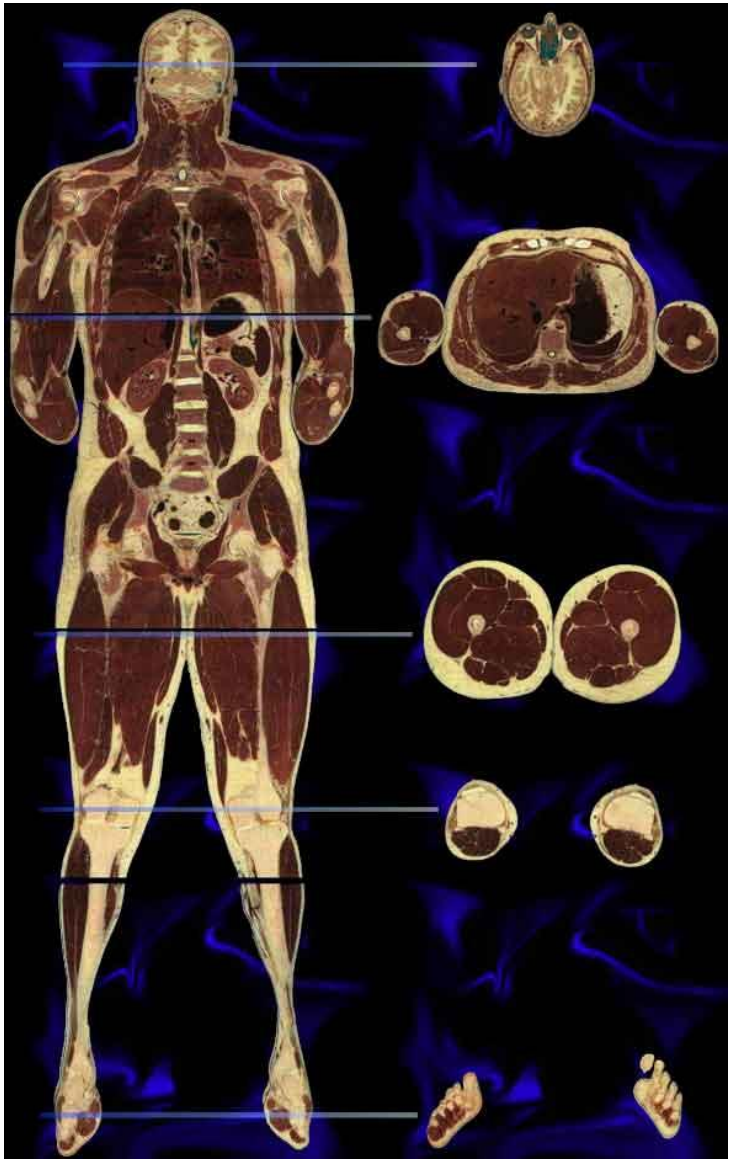
Dataset

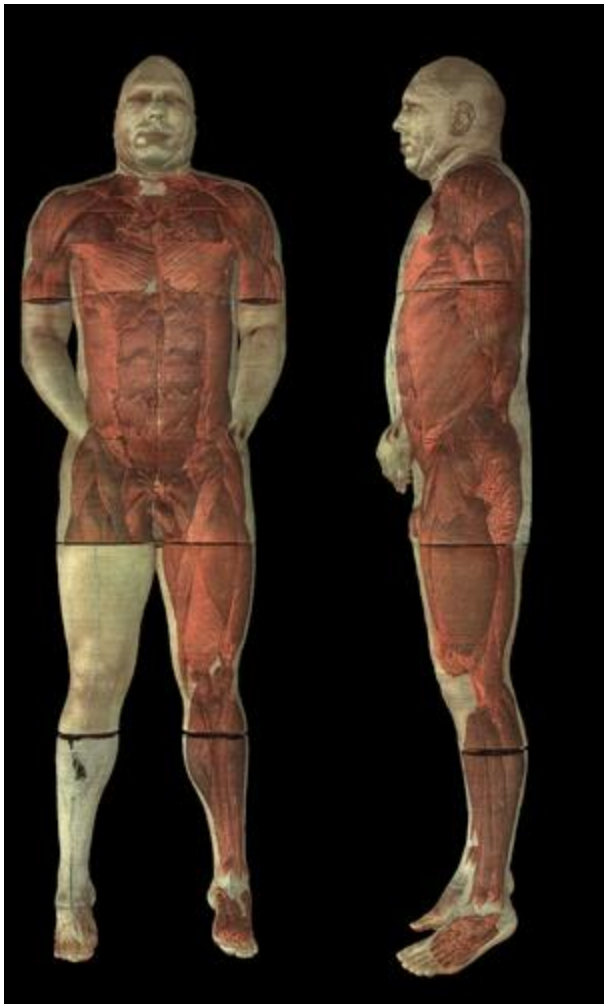


Renderings

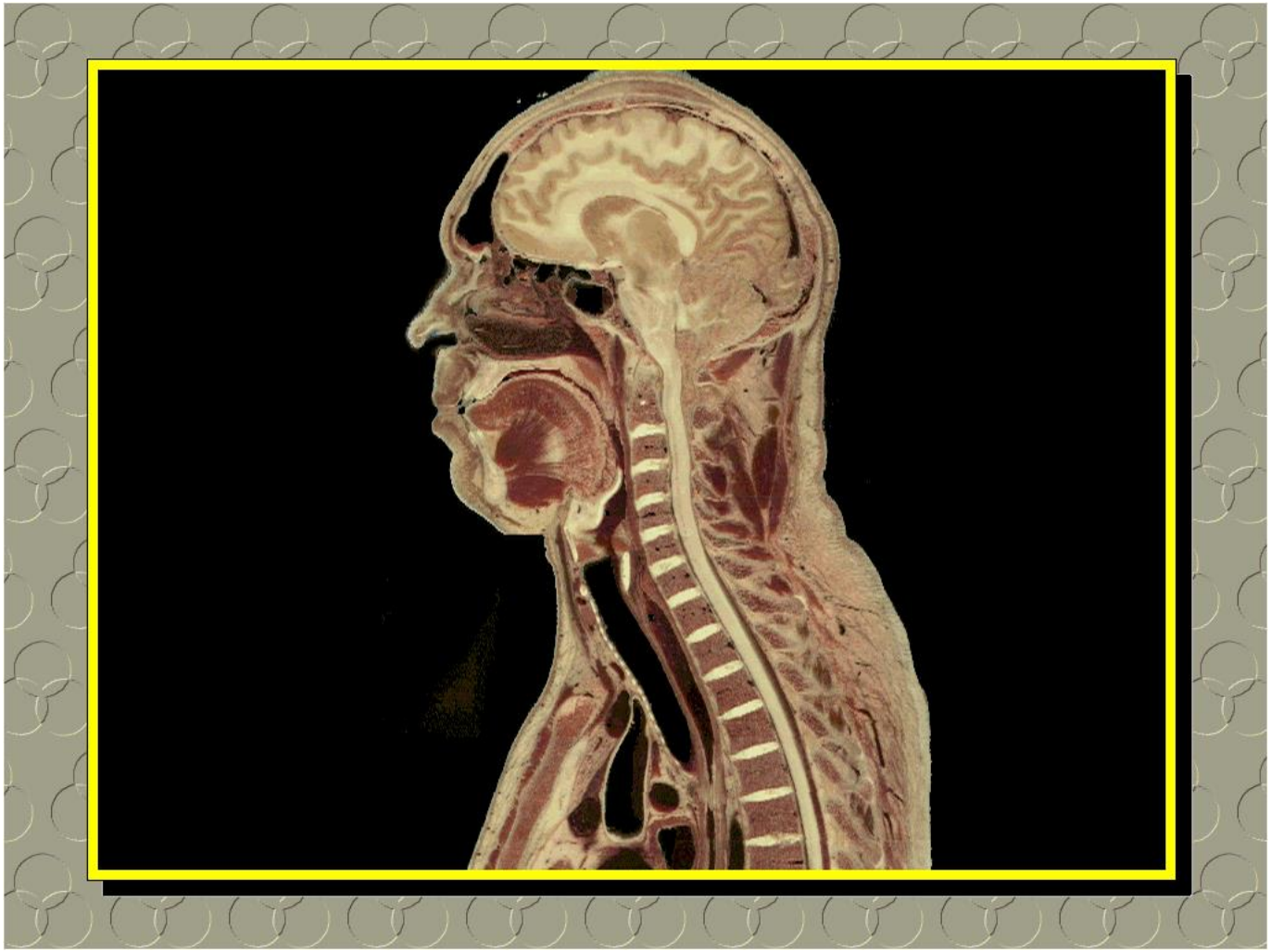

Quit
Show
L
R
Credits
Animations
Next
Structures of Slice 1105
Cerebellum
Lateral ventricle
Left eye
Optic nerves
Right eye
Third ventricle
R
Slice 1105
First
Last

Renderings





Renderings

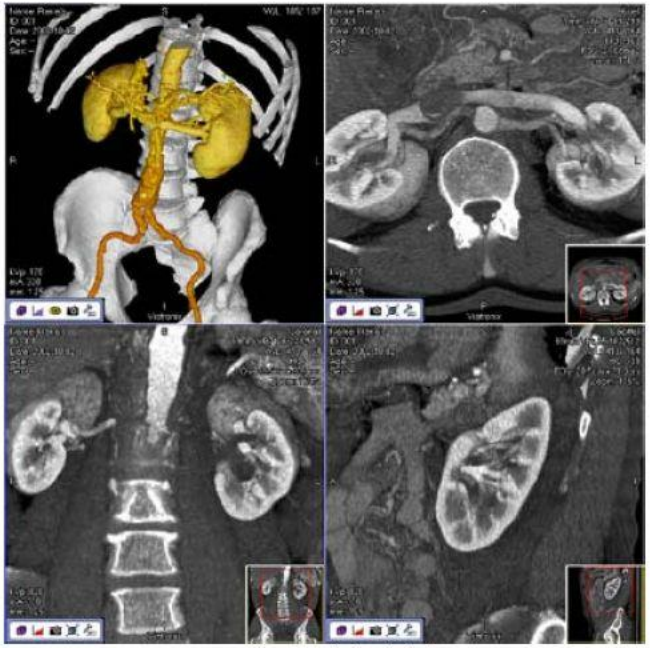


Renderings

- Data scanned with medical scanners (MRI, CT, PET, SPECT, etc.)

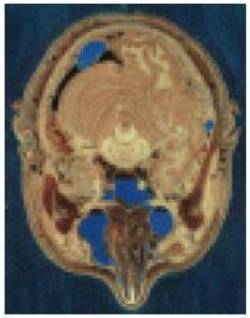
aortic aneurism

heart

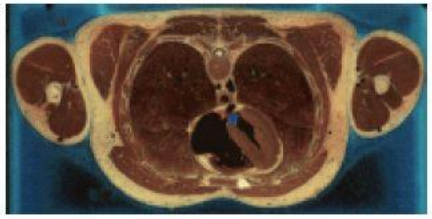


renals (with kidneys)

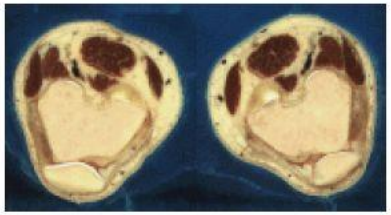
- Data photographed from histological slices (NIH-NLM *Visible Human*)



head

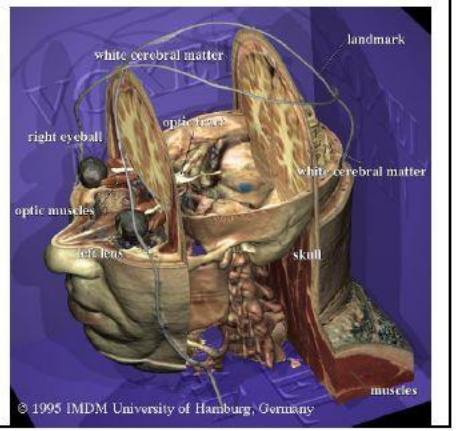


thorax

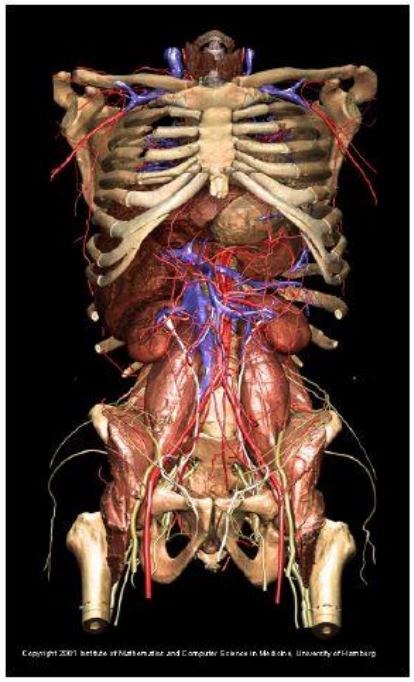


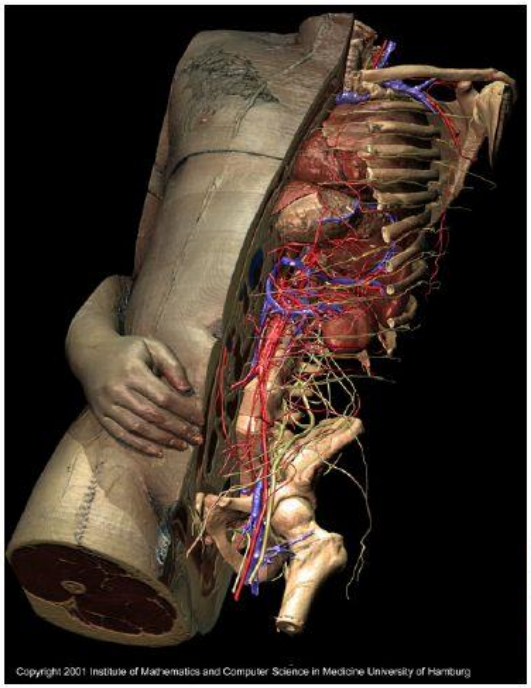
feet

atlas created from ~1700 1/3 mm slices

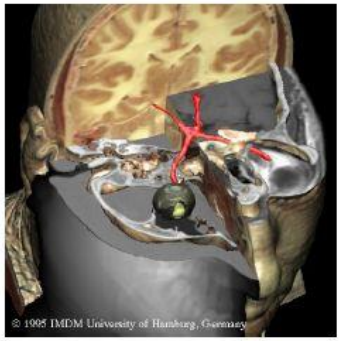


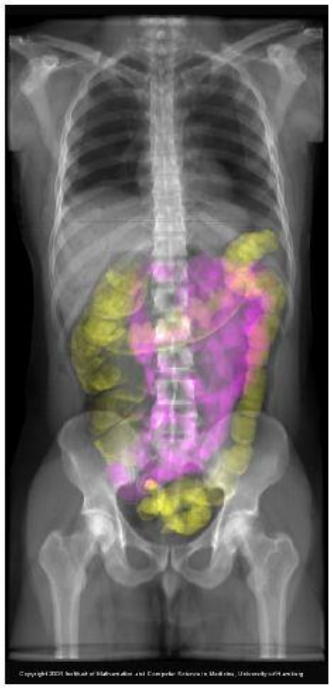
Renderings

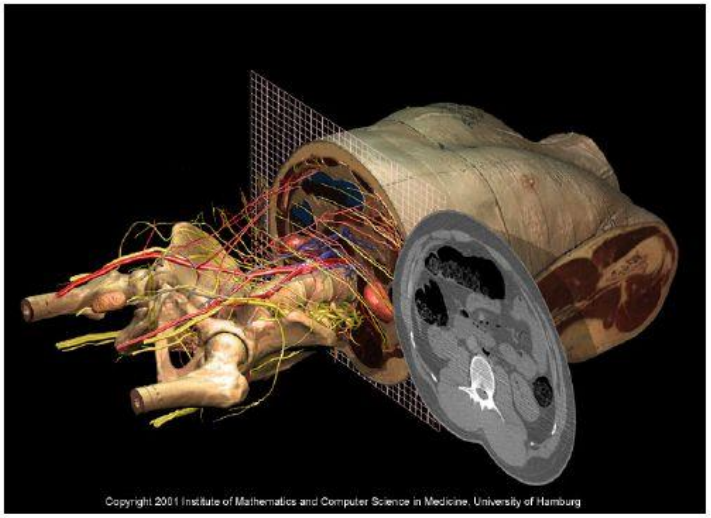

Copyright 2001 Institute of Mathematics and Computer Science in Medicine, University of Hamburg

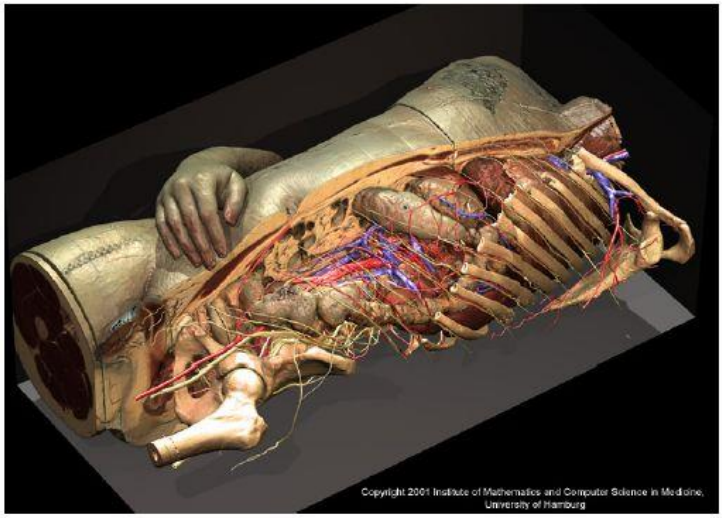

Copyright 2001 Institute of Mathematics and Computer Science in Medicine University of Hamburg


© 1995 IMDM University of Hamburg, Germany

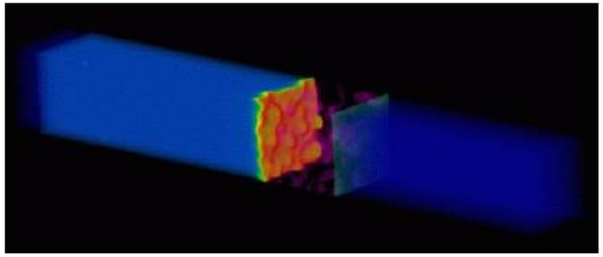

© 1995 IMDM University of Hamburg, Germany




Copyright 2001 Institute of Mathematics and Computer Science in Medicine, University of Hamburg

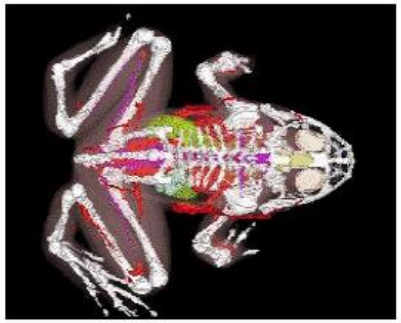

Copyright 2001 Institute of Mathematics and Computer Science in Medicine, University of Hamburg

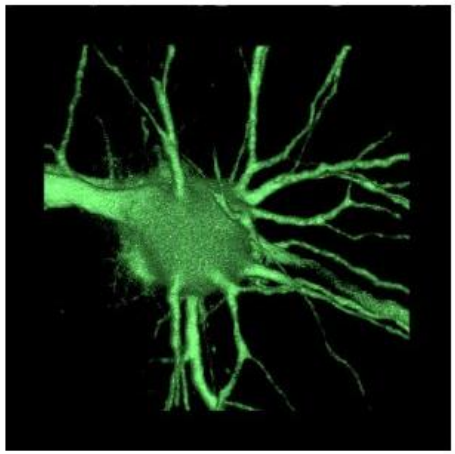
Scientific Visualization



shock wave

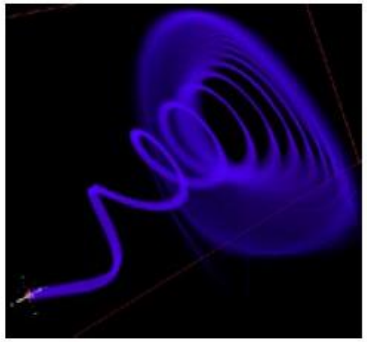
virtual frog

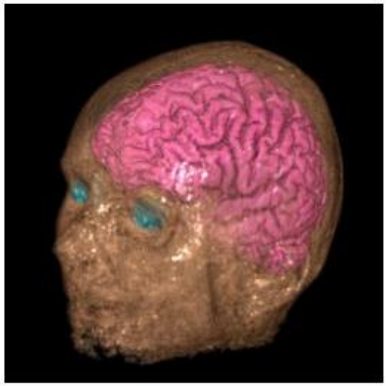




nerve cell

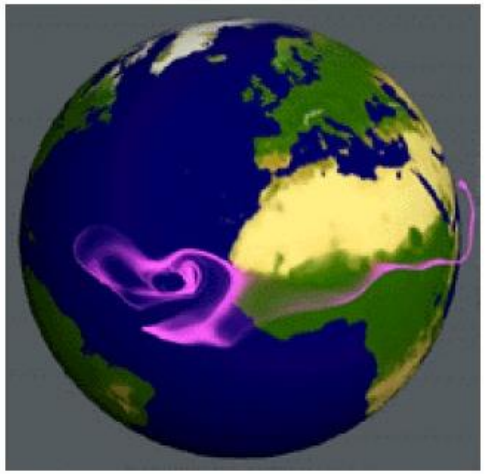
spiral flow





transparent MRI head

wind flow



MRI head

semi-transparent tomato

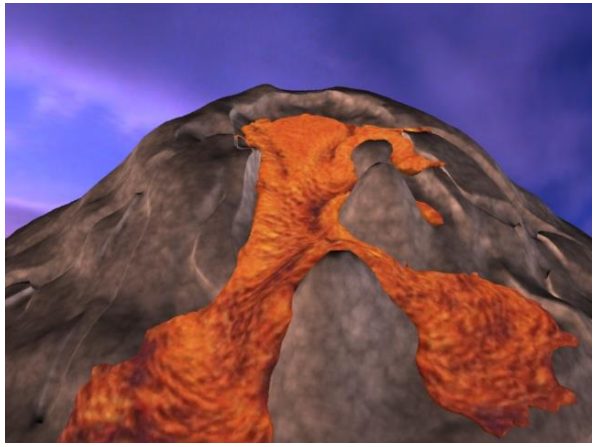
Fluid Dynamics Simulations

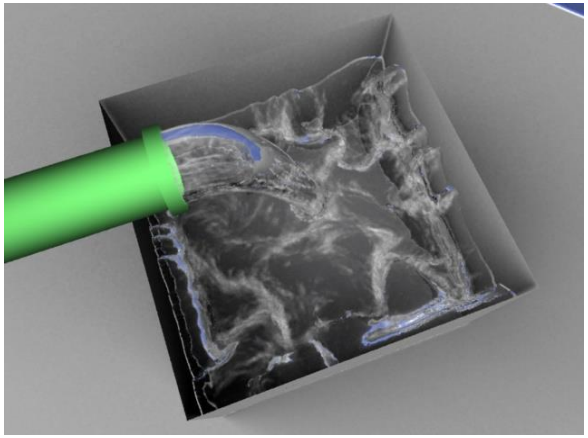
Navier-Stokes equations for viscous, incompressible liquids.

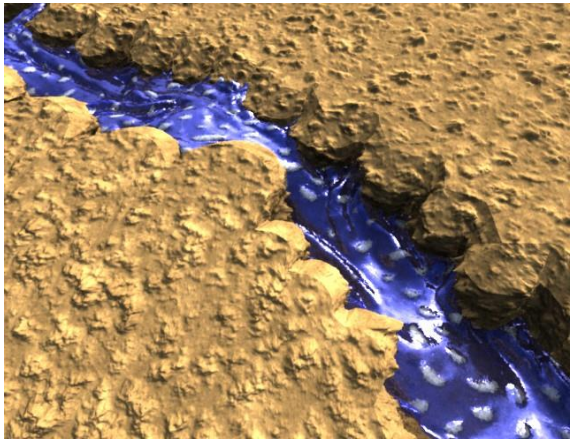
$$\nabla \cdot \mathbf{u} = 0$$ Conversation of mass

$$\mathbf{u}_t = -(\mathbf{u} \cdot \nabla)\mathbf{u} + \nu \nabla^2 \mathbf{u} - \frac{1}{\rho} \nabla p + \mathbf{f}$$

Advection Diffusion Pressure





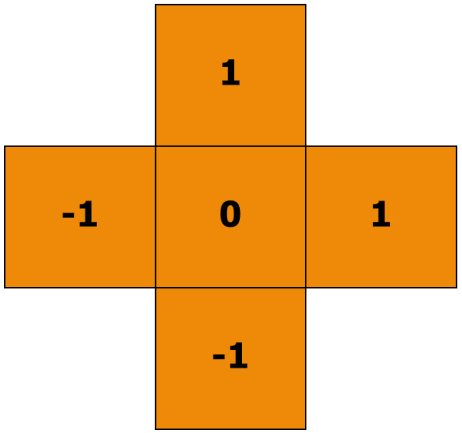


Navier-Stokes Solution

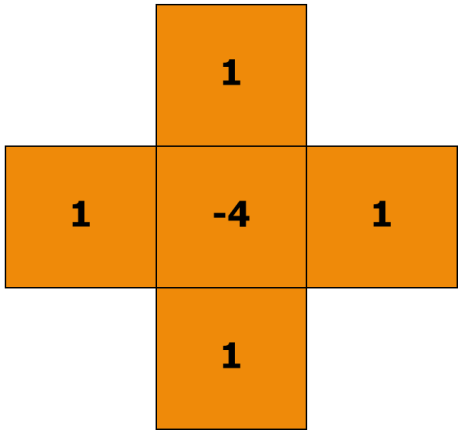
Via finite differencing

It all boils down to Ax=b.

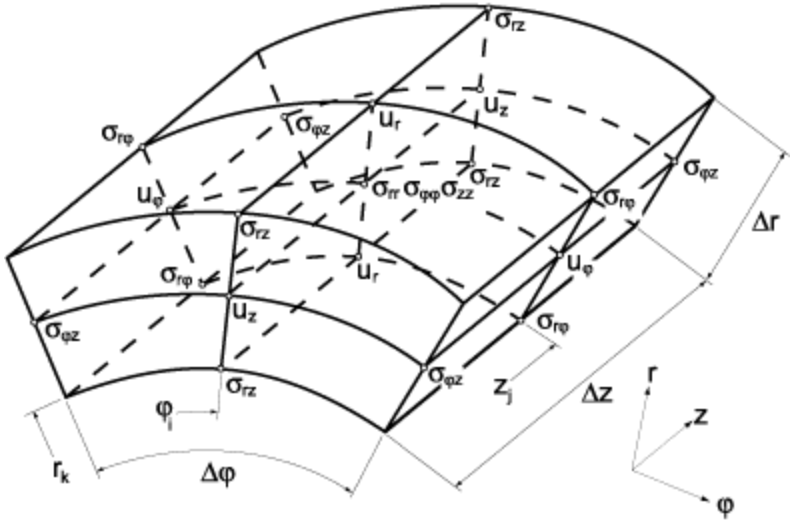
Divergence Operator



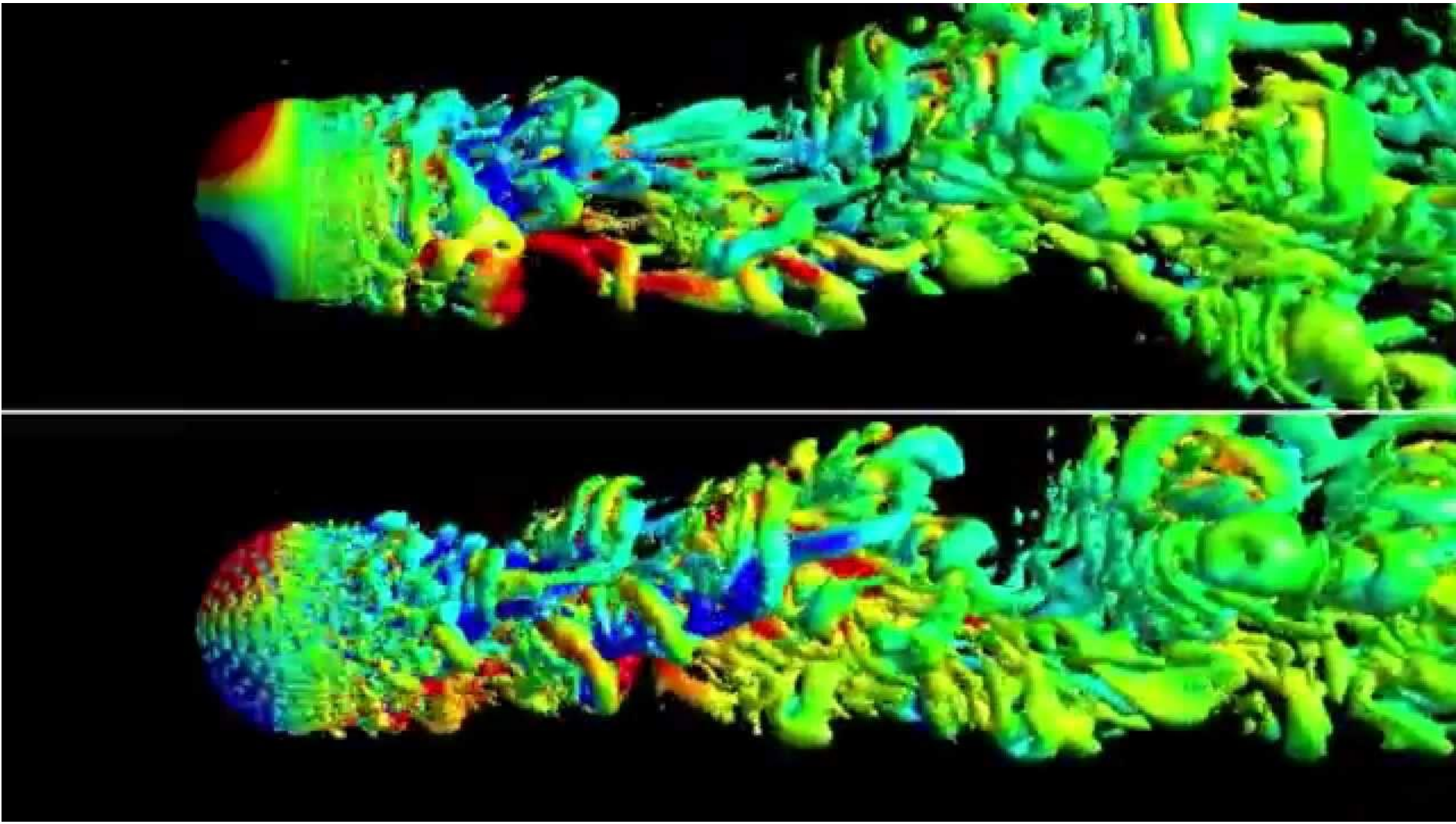
Laplacian Operator



$$\begin{bmatrix} ? & ? & \cdots & \cdots & ? \\ ? & ? & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ ? & \cdots & \cdots & \cdots & ? \end{bmatrix}_{n^d x n^d} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_{n^d} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ x_{n^d} \end{bmatrix}$$



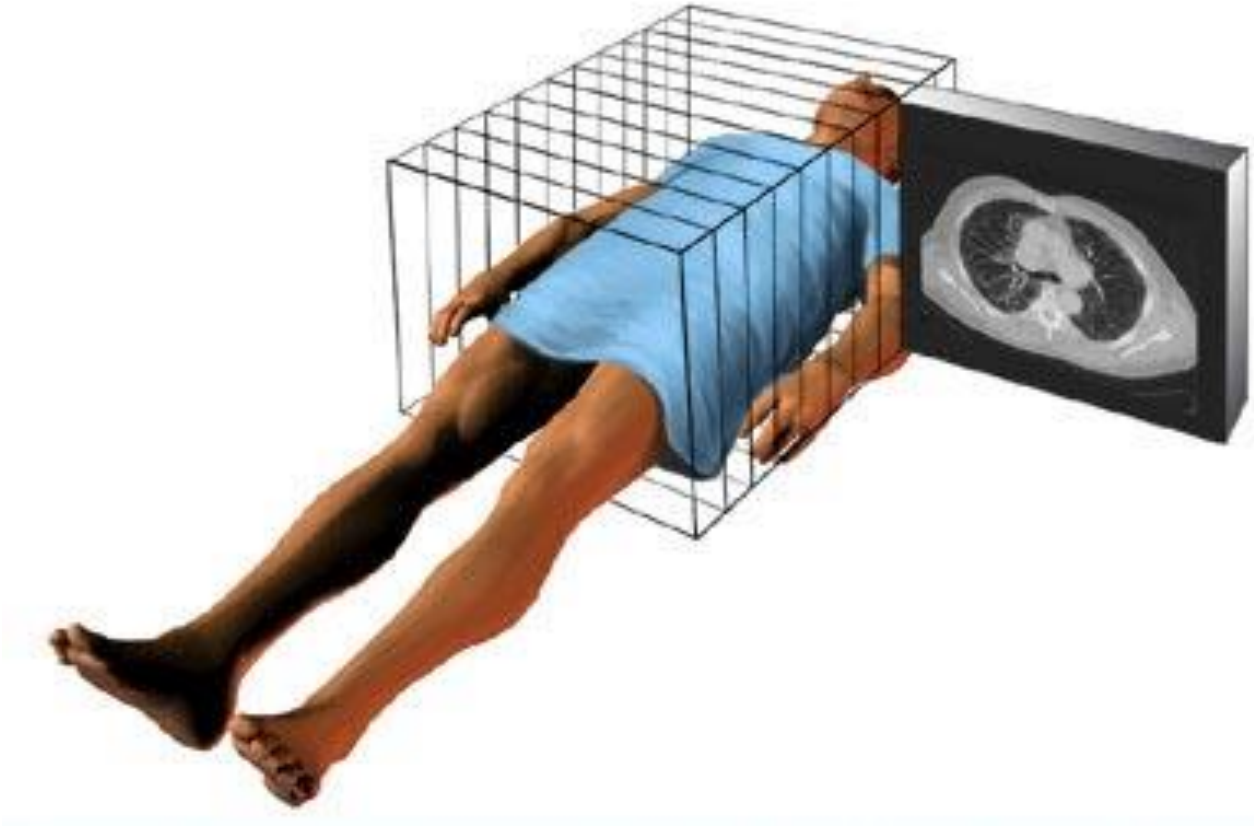
Visualize via Volume Rendering



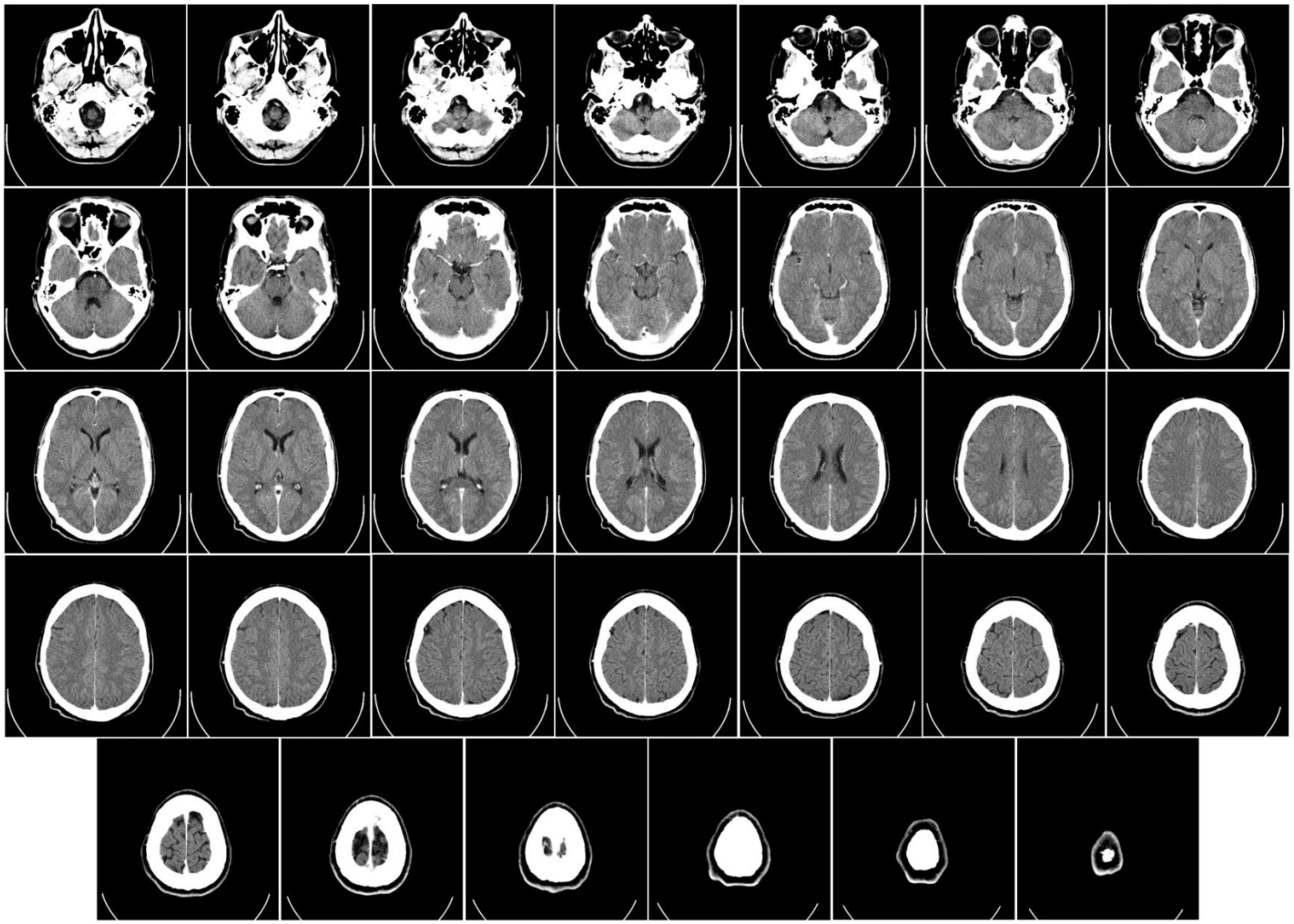
Volume Data Generation

Often obtained by scanning

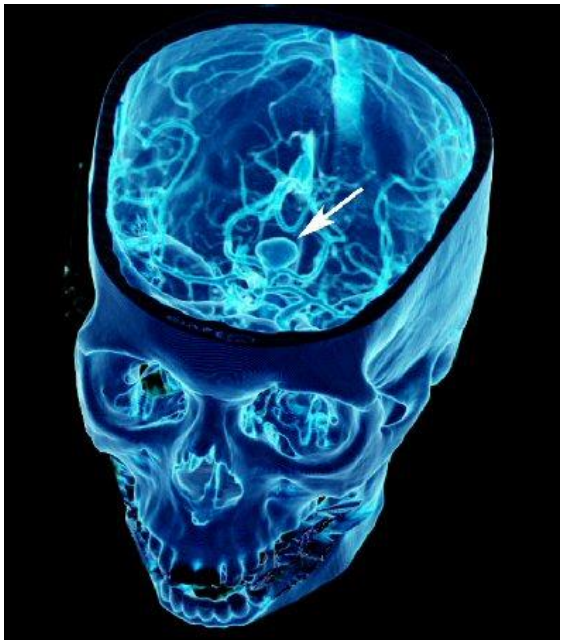
- for example, X-ray CT



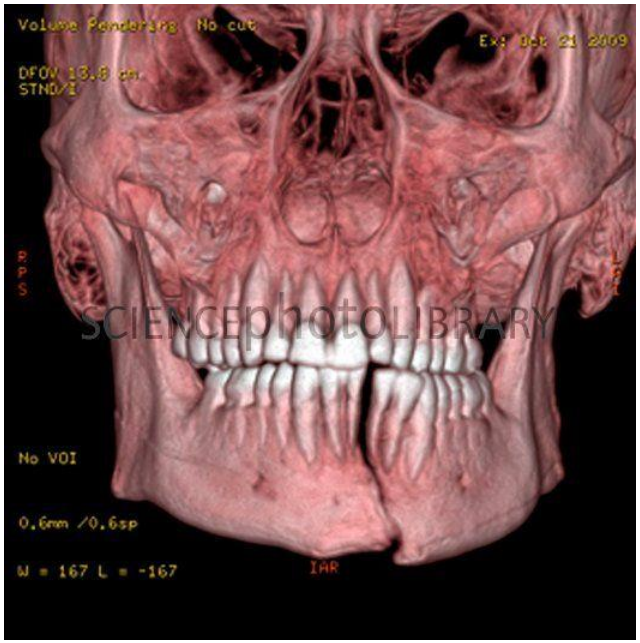
Volume Data – 2D Slice View



Volume Data – 3D Rendered View



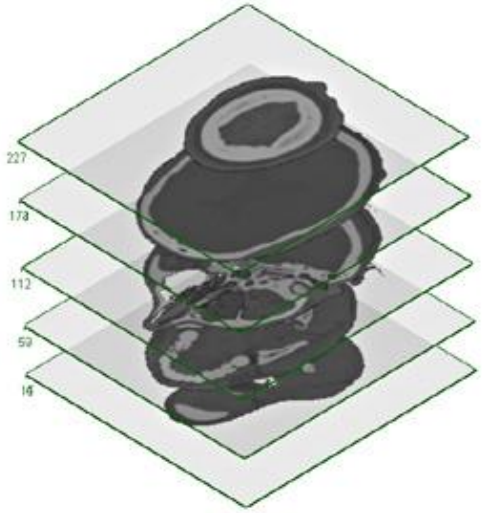
aneurism



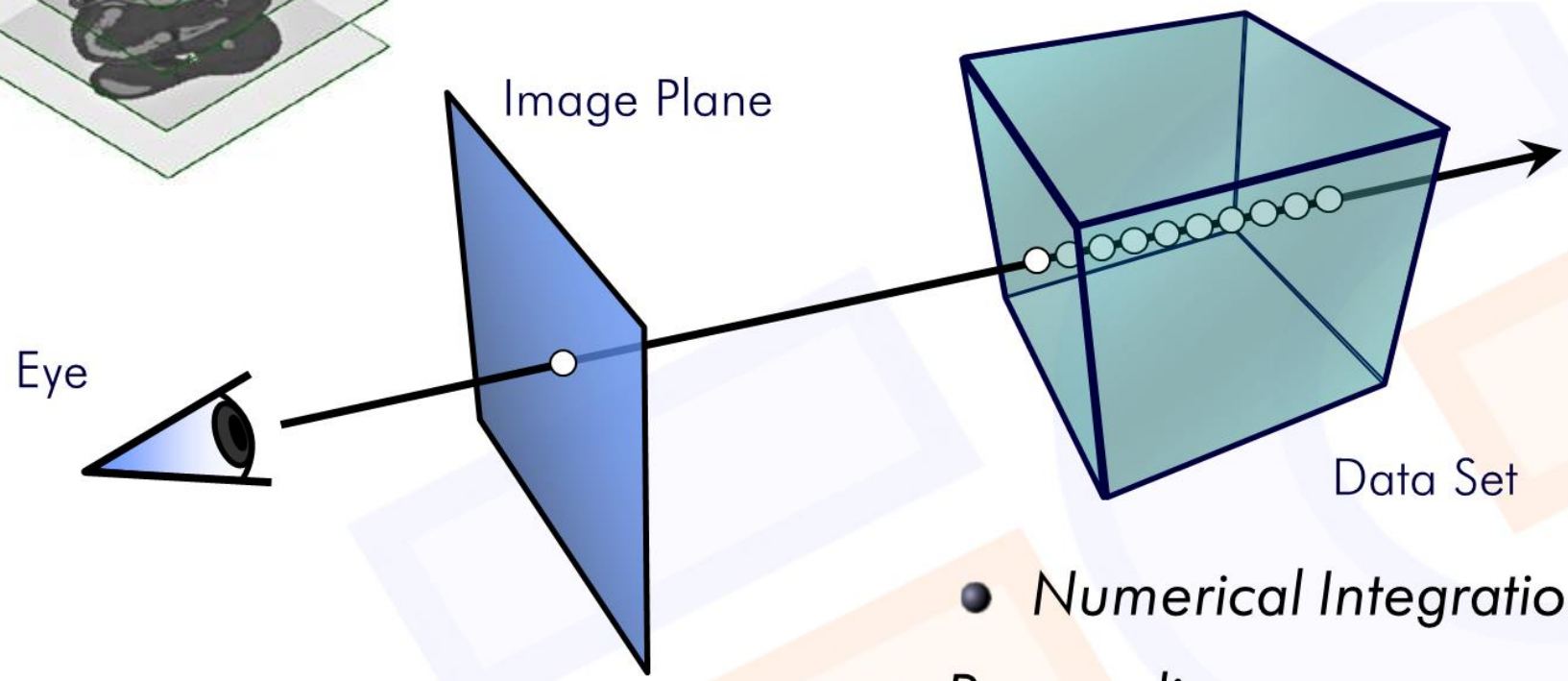
broken jaw

carotid arteries

Which do you prefer: 2D or 3D

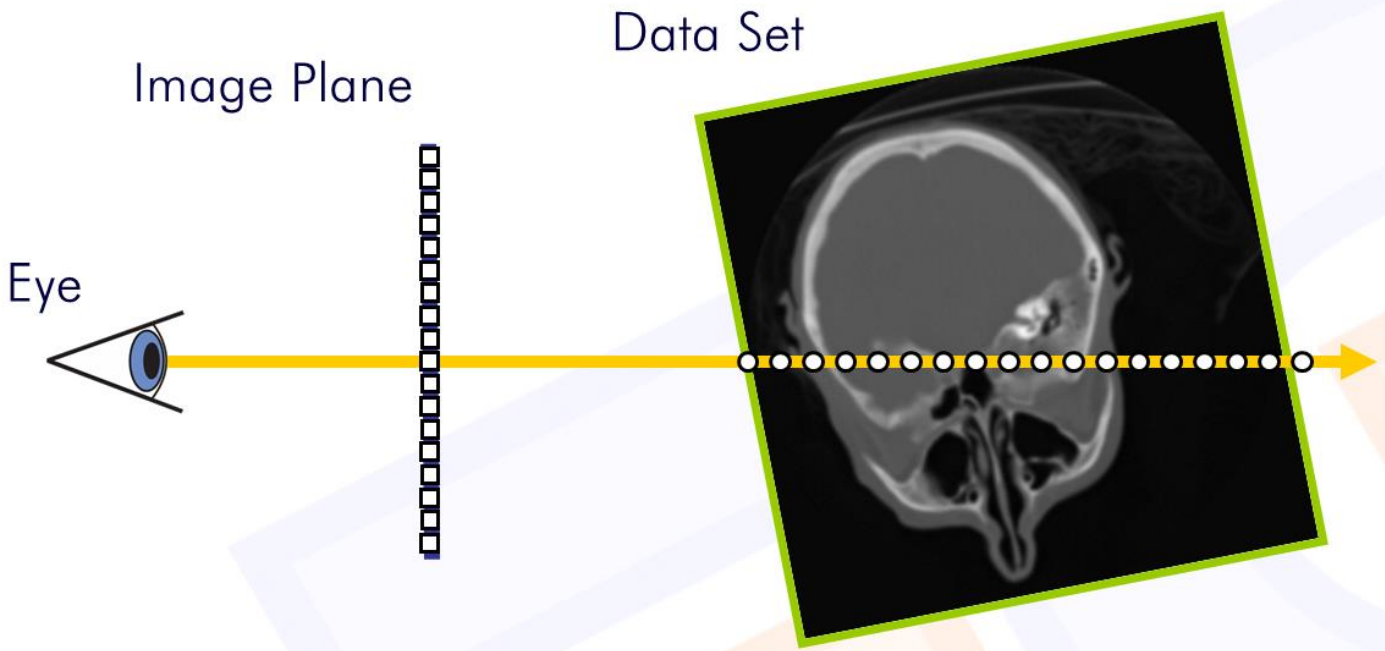


Raycasting Concept



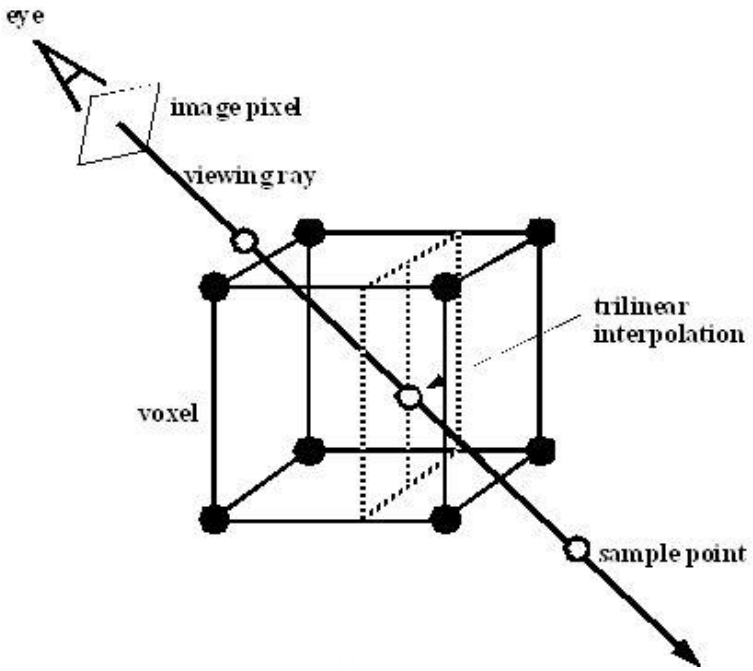
- *Numerical Integration*
- *Resampling*

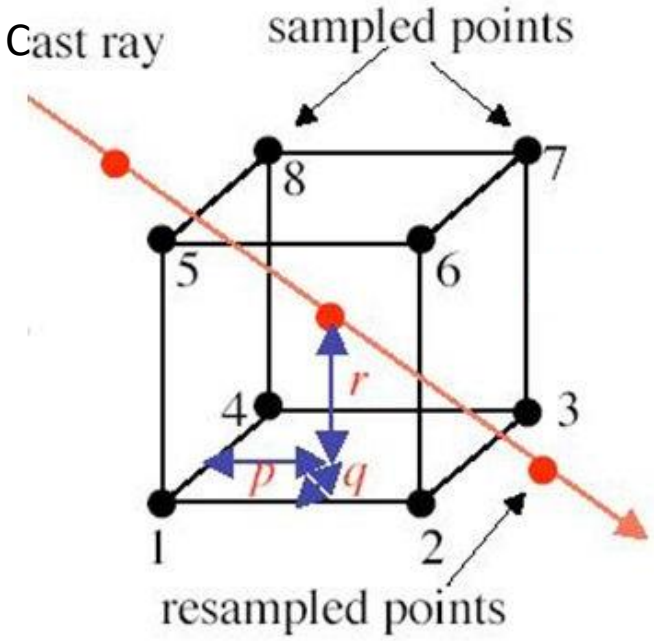
Sampling Along the Ray



Estimate sample values via *interpolation*

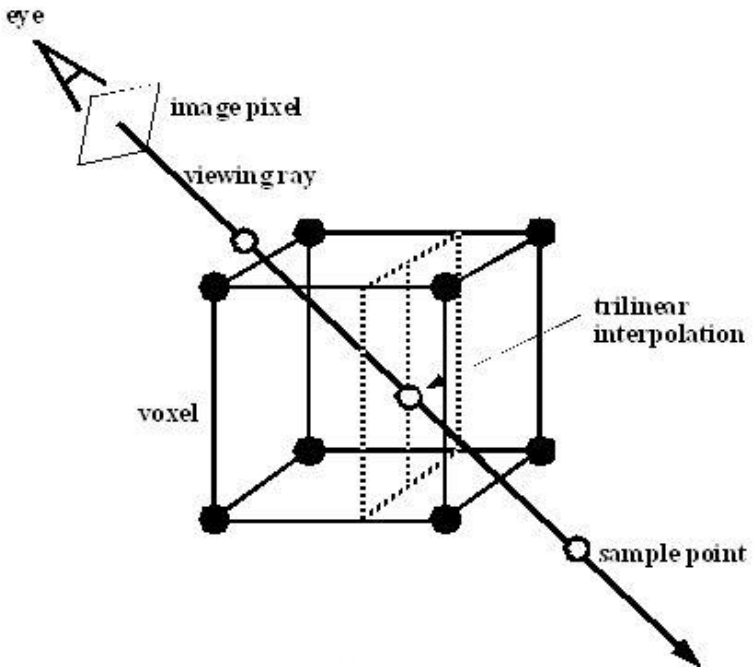
Sampling via Trilinear Interpolation

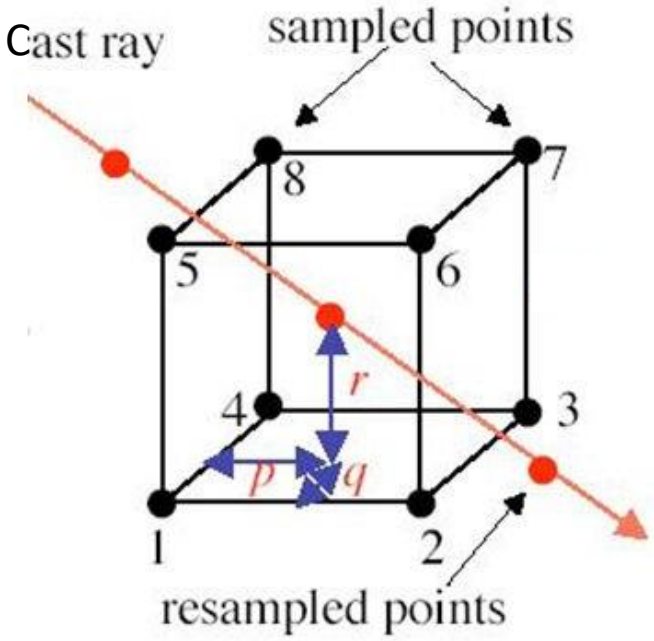




$$f_v = f_1(1-p)(1-q)(1-r) + f_2(p)(1-q)(1-r) + f_3(p)(q)(1-r) + f_4(1-p)(q)(1-r) + f_5(1-p)(1-q)(r) + f_6(p)(1-q)(r) + f_7(p)(q)(r) + f_8(1-p)(q)(r)$$

Sampling via Trilinear Interpolation

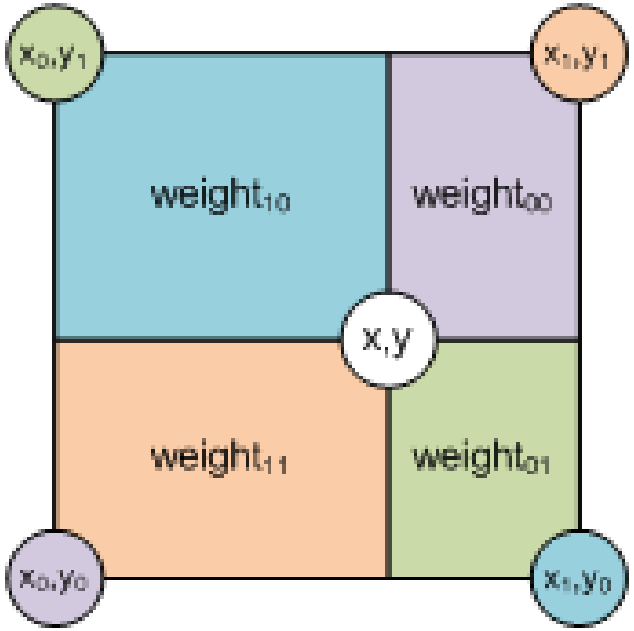




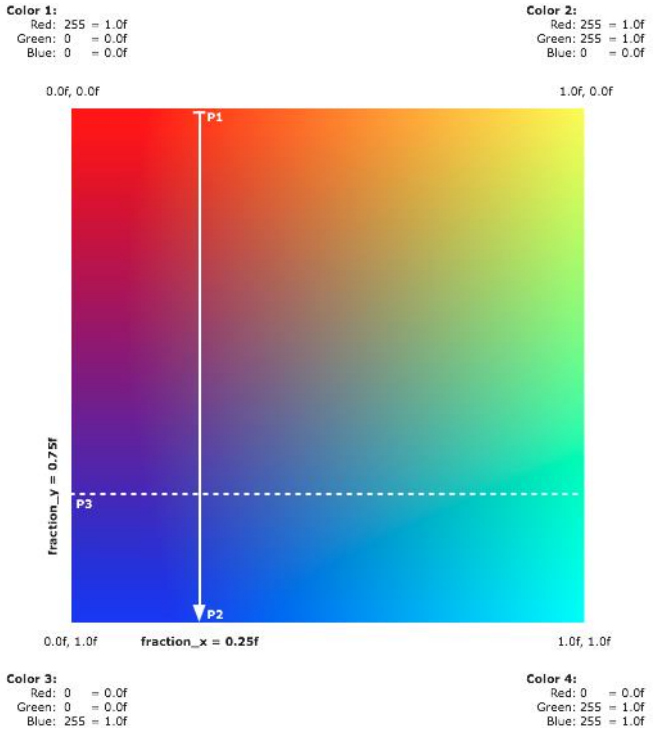
$$f_v = f_1(1-p)(1-q)(1-r) + f_2(p)(1-q)(1-r) + f_3(p)(q)(1-r) + f_4(1-p)(q)(1-r) + f_5(1-p)(1-q)(r) + f_6(p)(1-q)(r) + f_7(p)(q)(r) + f_8(1-p)(q)(r)$$

What Does This Exactly Mean?

Here is what it looks like in 2D for bi-linear interpolation



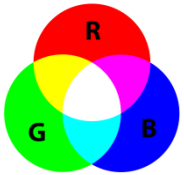
weights



interpolation result within one cell

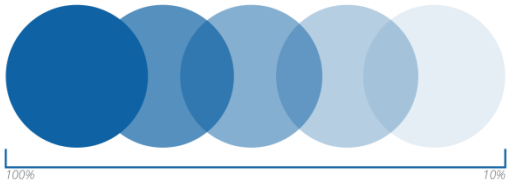
Transparency and Opacity

We learned about RGB



There is one more channel – opacity (A)

- gives RGBA color
- opacity (A) = 1 – transparency (T)
- range [0.0 … 1.0]



Opacity (A) multiplied by RGB creates a weighting effect

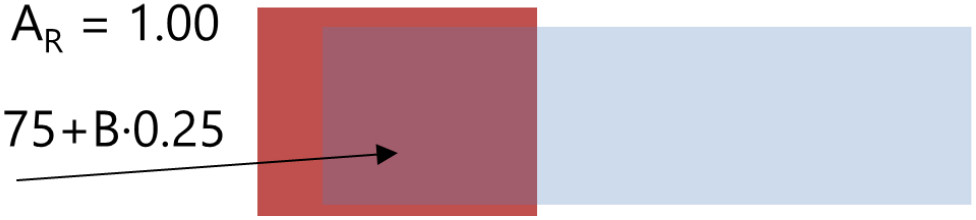
Opacity and Color Blending

$$C_{mix} = C_{back} A_{back} (1 - A_{front}) + C_{front} A_{front}$$

$$C_{mix} = C_R A_R (1 - A_B) + C_B A_B$$

$T_R = 0.00$, $A_R = 1.00$

$C = R \cdot 0.75 + B \cdot 0.25$

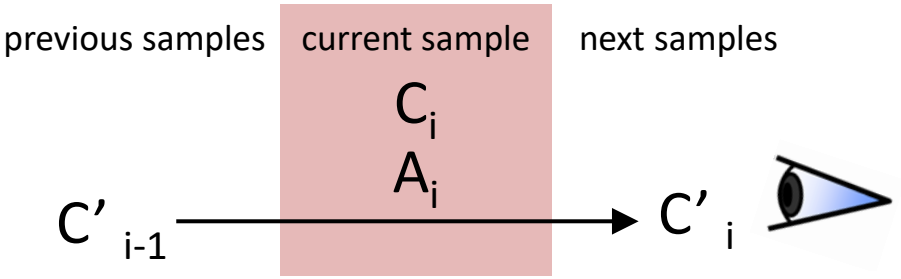


$T_B = 0.75$

$A_B = 0.25$

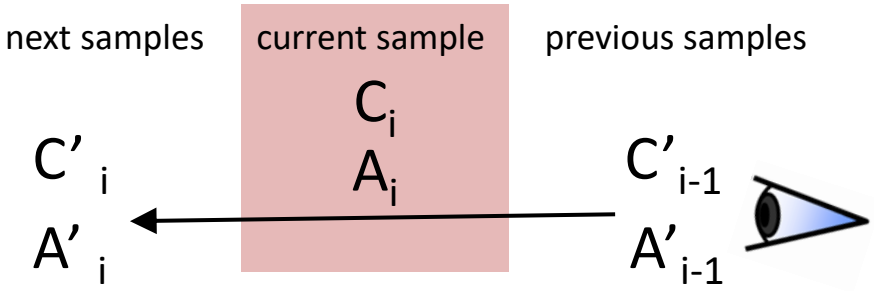
Compositing – Merging the Samples

Back-to-front rendering



$$C_i' = C_i A_i + (1 - A_i) C_{i-1}'$$

Front-to-back rendering



$$C_i' = C_{i-1}' + (1 - A_{i-1}') C_i A_i$$

$$A_i' = A_{i-1}' + (1 - A_{i-1}') A_i$$

A: Opacity = 1- Transparency = 1 - T

C: Color

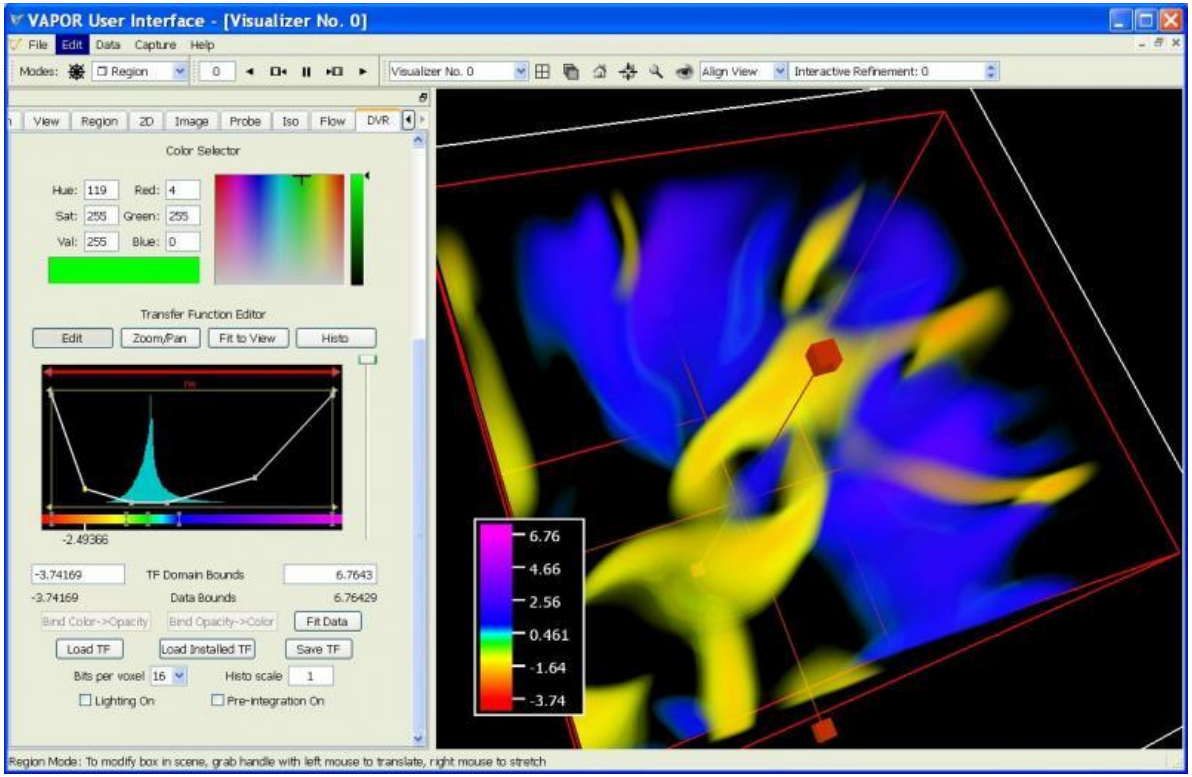
Transfer Function

Determines what color & opacity a sample value should have

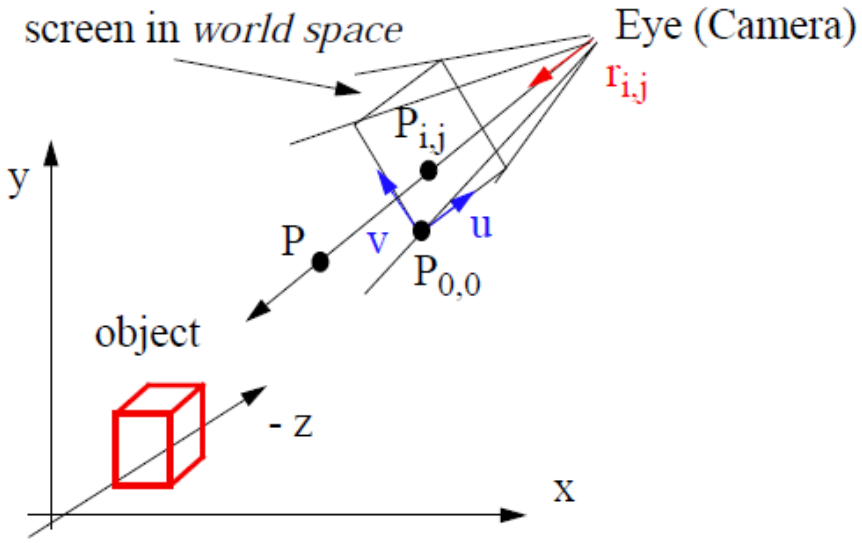
- input: an interpolated density value
- output: a color and opacity (RGBA)

rendering ↓

transfer function →



Raycasting Specifics



A ray is specified by:

- eye position (Eye)
- screen pixel location $P_{i,j}$

→ ray direction vector ($r_{i,j}$) of unit length

$$r_{i,\,j} = \frac{P_{i,\,j} - Eye}{\left| P_{i,\,j} - Eye \right|}$$

A point P on a ray is given by:

$$P = Eye + t \cdot r_{i,\,j}$$

t: parametric variable

Spacing of pixels on image plane:

$$\Delta i = \frac{W}{Ni - 1} \qquad \Delta j = \frac{H}{Nj - 1}$$

Ni, Nj: image dims. in pixels

Image-order projection:

- scan the image row by row, column by column:

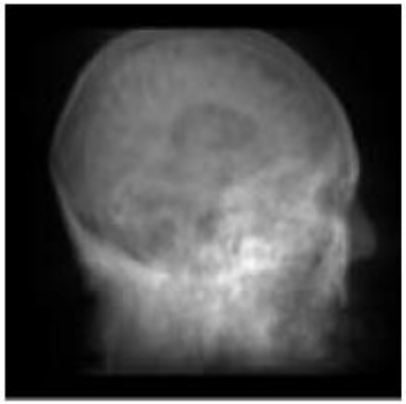
$$P_{i,\,j} = P_{0,\,0} + i \cdot v \cdot \Delta j + j \cdot u \cdot \Delta i$$

- $P_{i,\,j}$: Location of image pixel (i, j) in world space

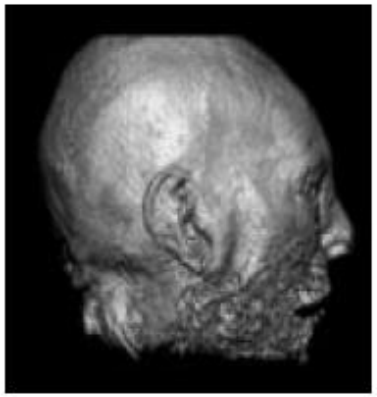
$0 \le i < Ni \quad 0 \le j < Nj$

- $P_{0,\,0}$: image (=screen) origin in world space
- u, v, n: orthonormal image plane vectors ($n = v \times u$)

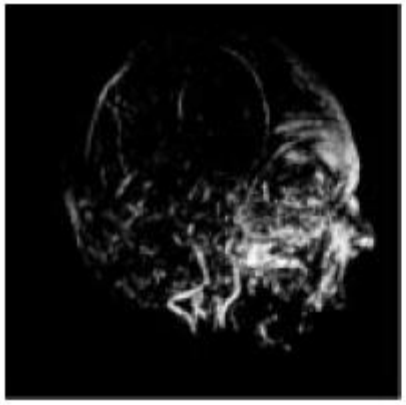
Volume Rendering Modes



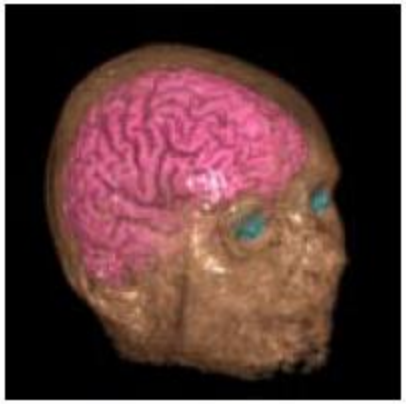
X-ray:
rays sum volume cor
tributions along their
linear paths



Iso-surface:
rays look for the object
surfaces, defined by a cer-
tain volume value



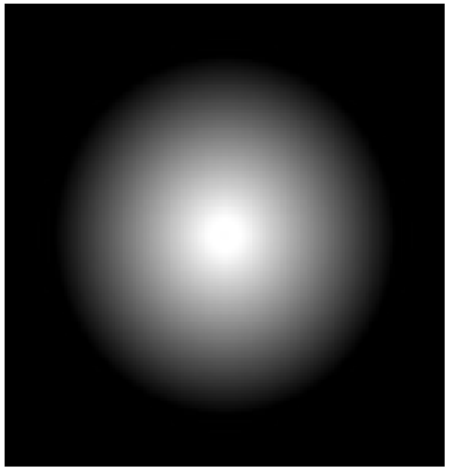
Maximum Intensity Pro
jection (MIP):
a pixel value stores th
largest volume value
along its ray



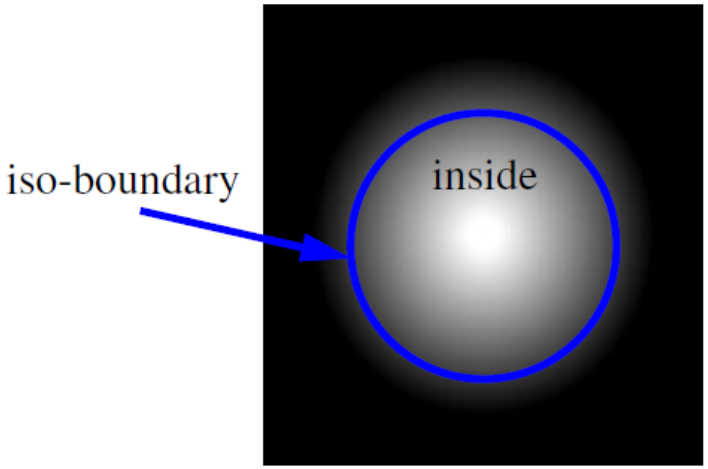
Full volume rendering:
rays *composite* volume
contributions along their
linear paths

Iso-Surface Rendering

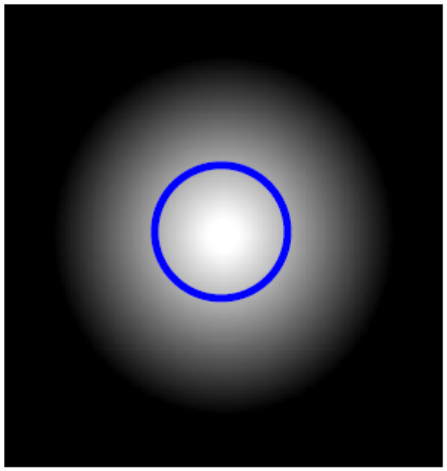
- A closed surface separates 'outside' from 'inside' (Jordan theorem)
- In iso-surface rendering we say that all voxels with values > some threshold are 'inside', and the others are 'outside'
- The boundary between 'outside' and 'inside' is the *iso-surface*
- All voxels near the iso-surface have a value close to the *iso-threshold* or *iso-value*
- Example:



cross-section of a smooth sphere

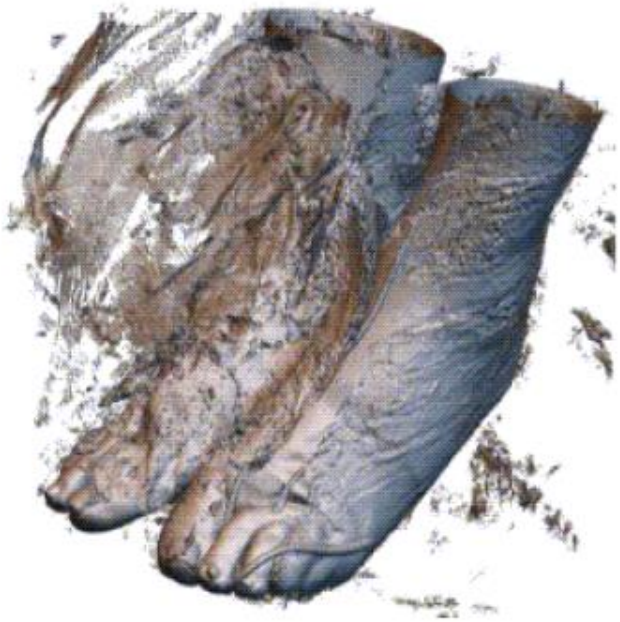


iso-value = 50
will render a large sphere



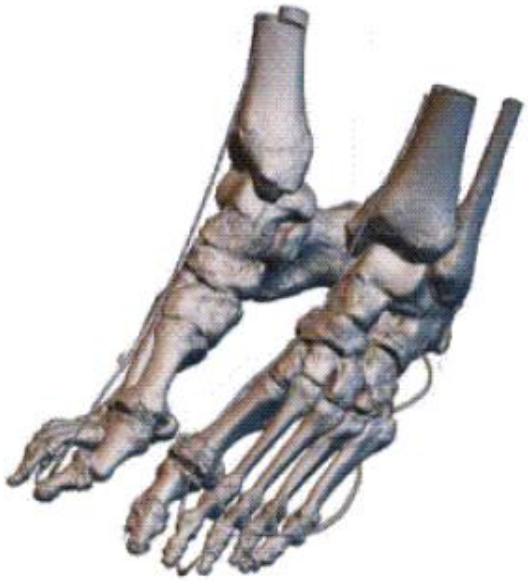
iso-value = 200
will render a small sphere

Iso-Surface Rendering



iso-value = 30

iso-value = 80

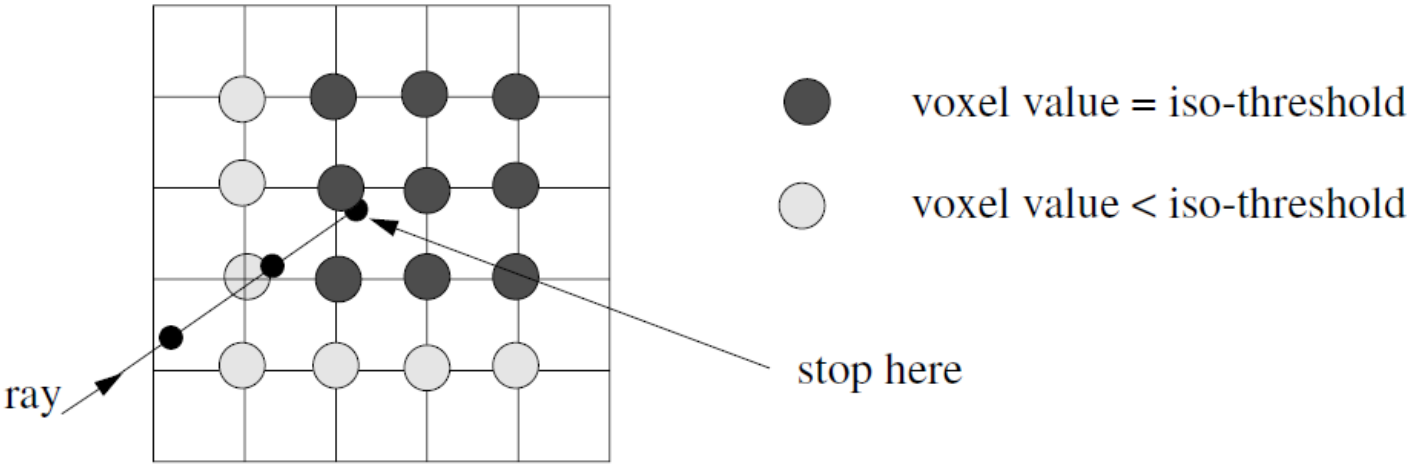


iso-value = 200

Iso-Surface Rendering – Details

- To render an iso-surface we cast the rays as usual...

  but we stop, once we have interpolated a value iso-threshold

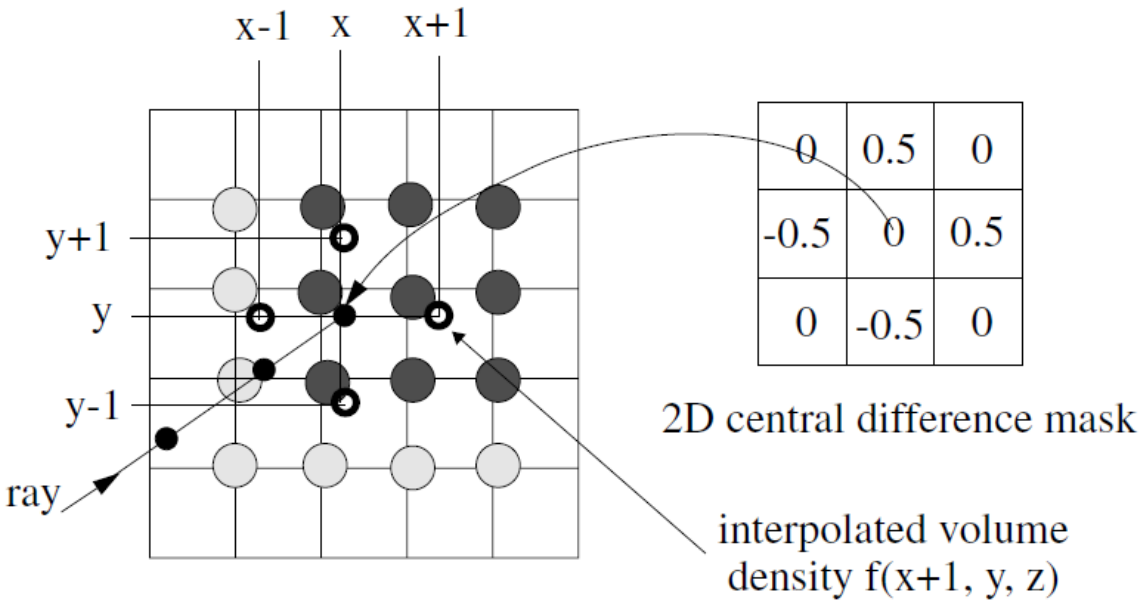


- We would like to illuminate (shade) the iso-surface based on its orientation to the light source
- Recall that we need a normal vector for shading
- The normal vector N is the local gradient, normalized

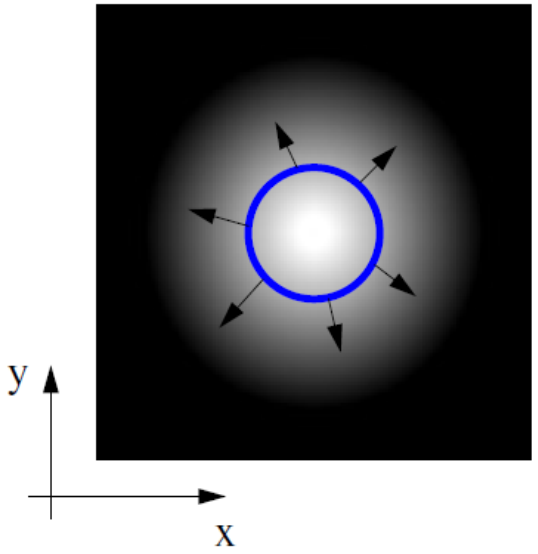
The Gradient Vector

- The gradient vector $\mathbf{g}=(g_x, g_y, g_z)^T$ at the sample position (x, y, z) is usually computed via central-differencing (for example, g_x is the volume density gradient in the x-direction):

$$g_x = \frac{f(x-1, y, z) - f(x+1, y, z)}{2} \qquad g_y = \frac{f(x, y-1, z) - f(x, y+1, z)}{2} \qquad g_z = \frac{f(x, y, z-1) - f(x, y, z+1)}{2}$$



the x and y component of the gradient vector for the smooth sphere

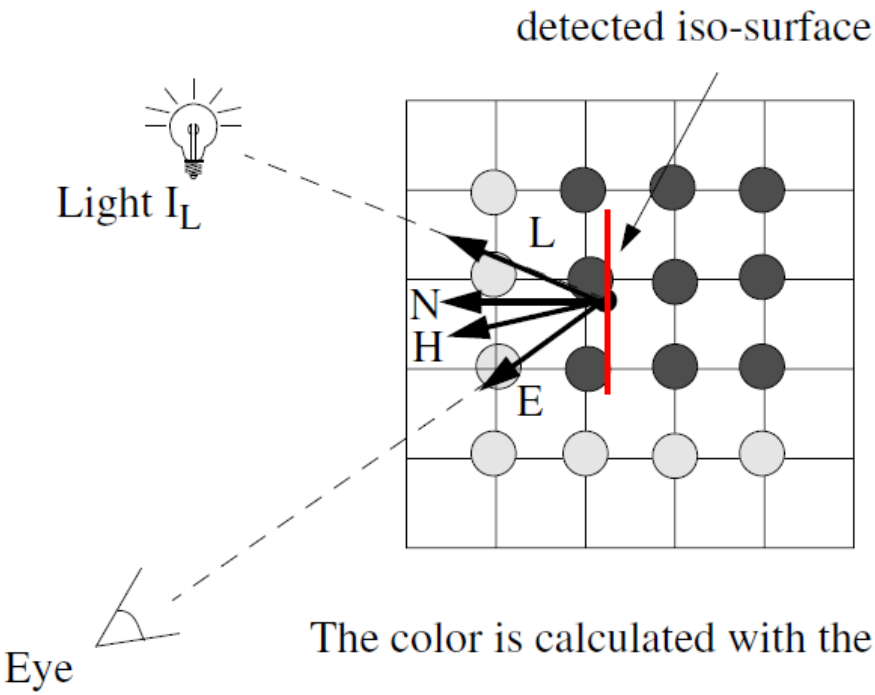


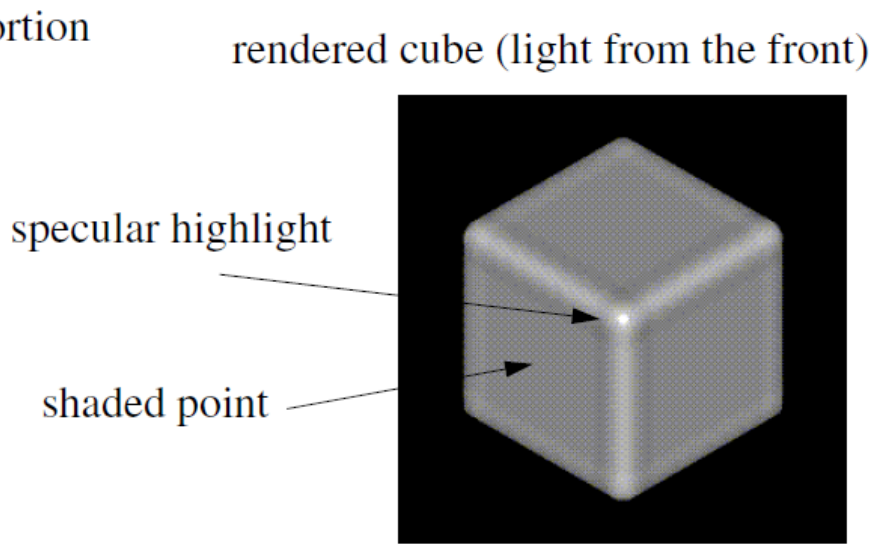
Shading the Iso-Surface

- The normal vector is the *normalized* gradient vector g

  N = g / |g| (normal vector always has unit length)

- Once the normal vector has been calculated we shade the iso-surface at the sample point
- The color so obtained is then written to the pixel that is due to the ray





The color is calculated with the standard shading equation:

$$C = C_{obj} (k_a I_A + k_d I_L N \cdot L) + k_s I_L (H \cdot N)^{ns}$$

C_{obj} is obtained by indexing the color transfer function with the interpolated sample value

Full Volume Rendering

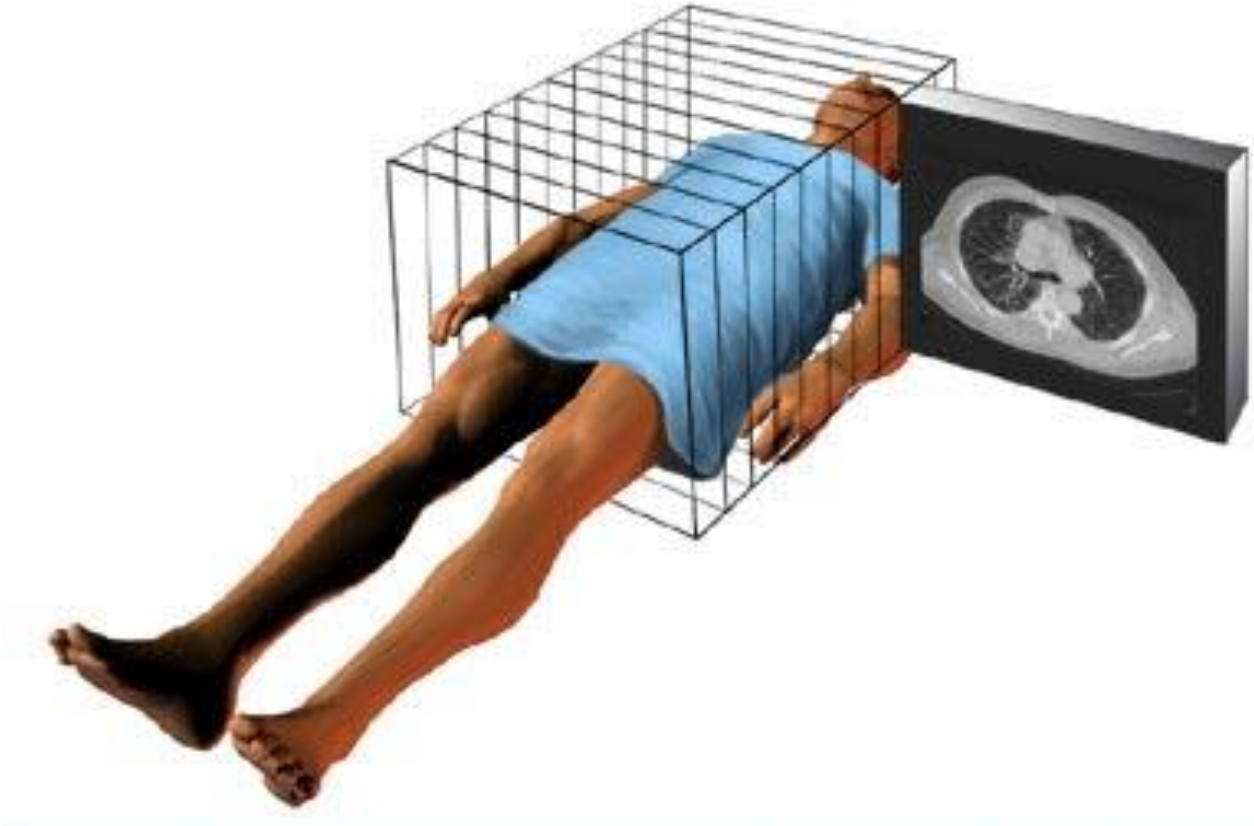
When hitting a surface set A< 1.0

- ray marches on
- inner structures can be seen

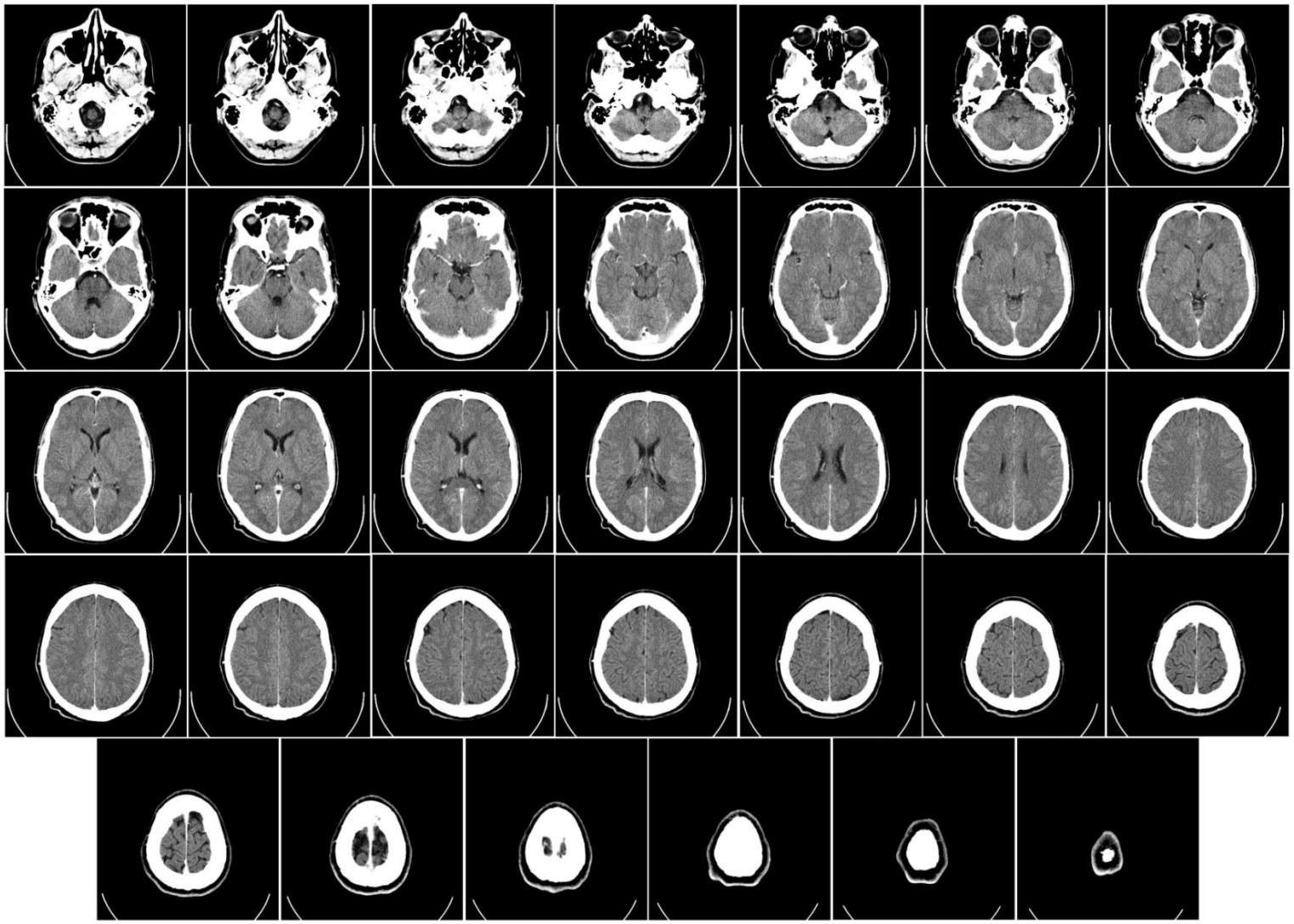
Volume Data Generation

Often obtained by scanning

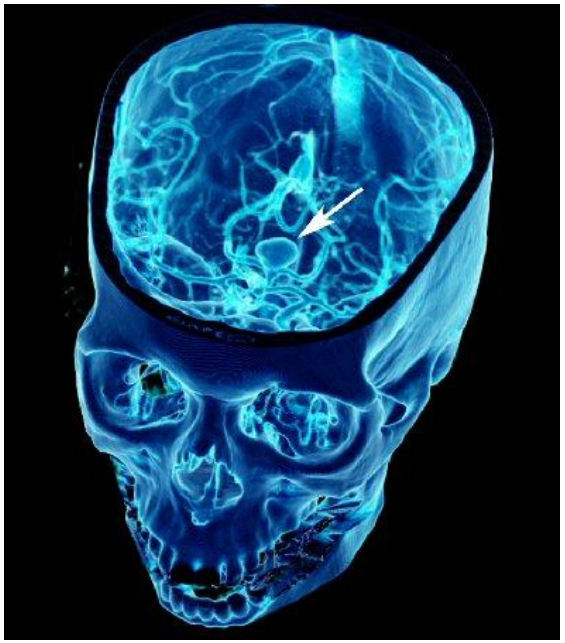
- for example, X-ray CT



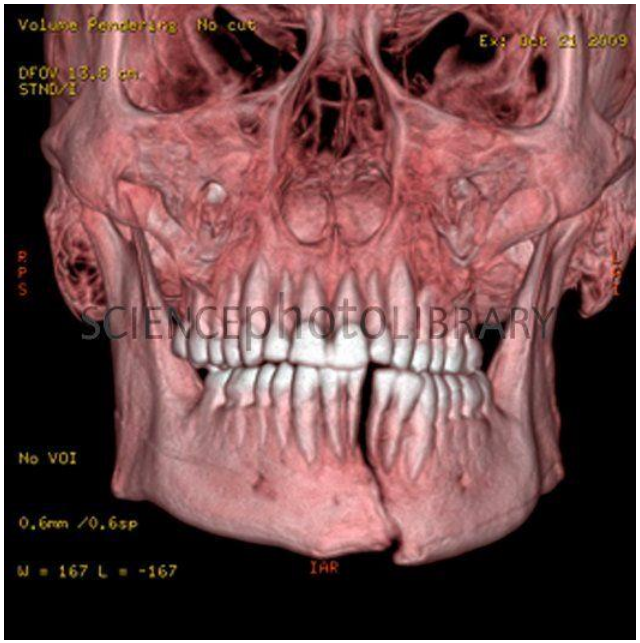
Volume Data – 2D Slice View



Volume Data – 3D Rendered View



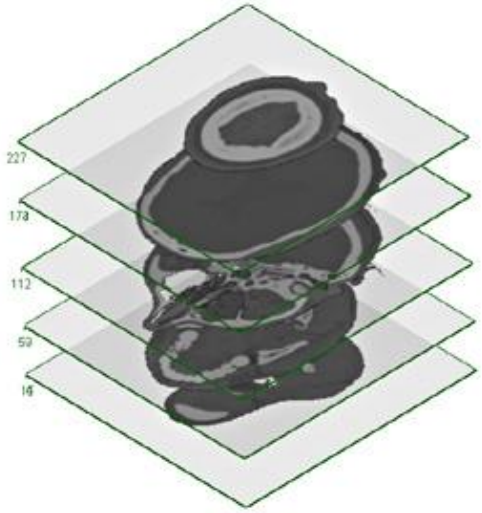
aneurism



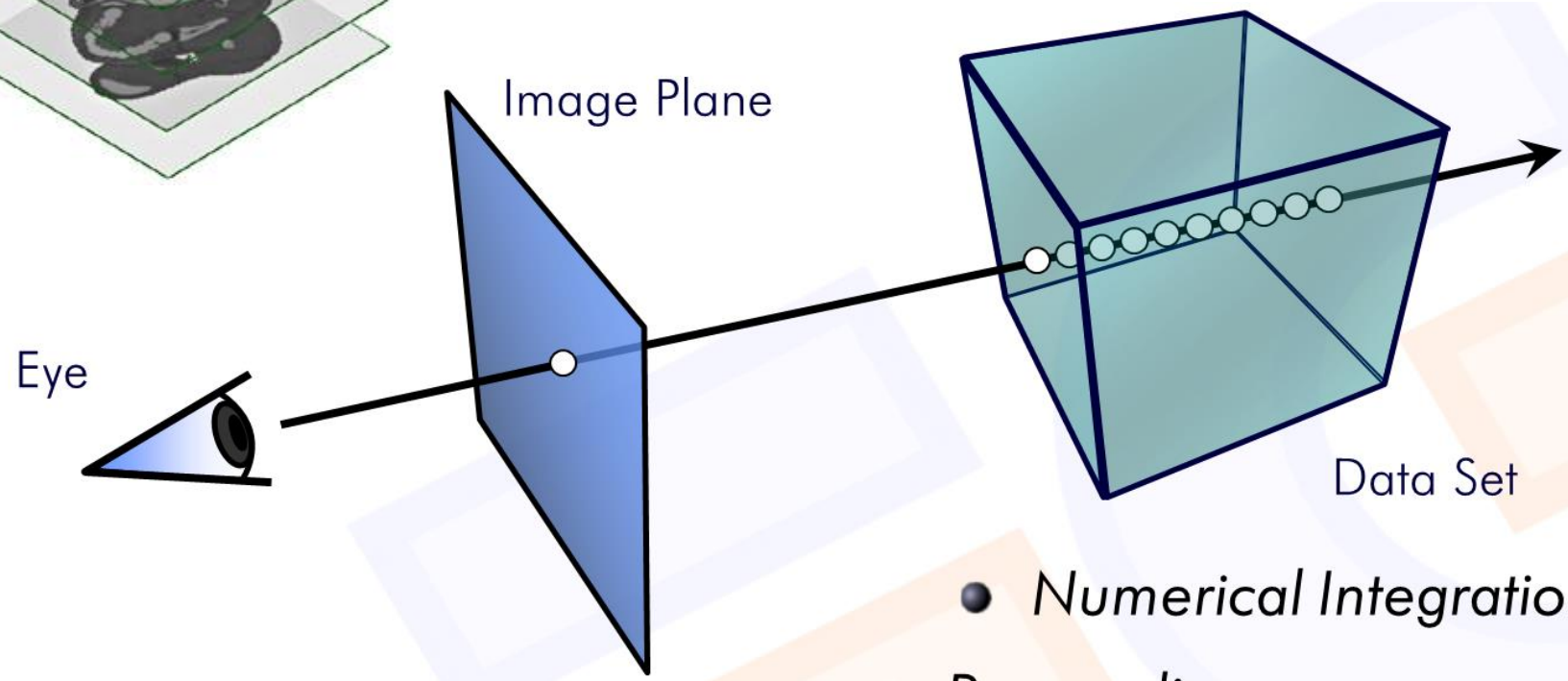
broken jaw

carotid arteries

Which do you prefer: 2D or 3D

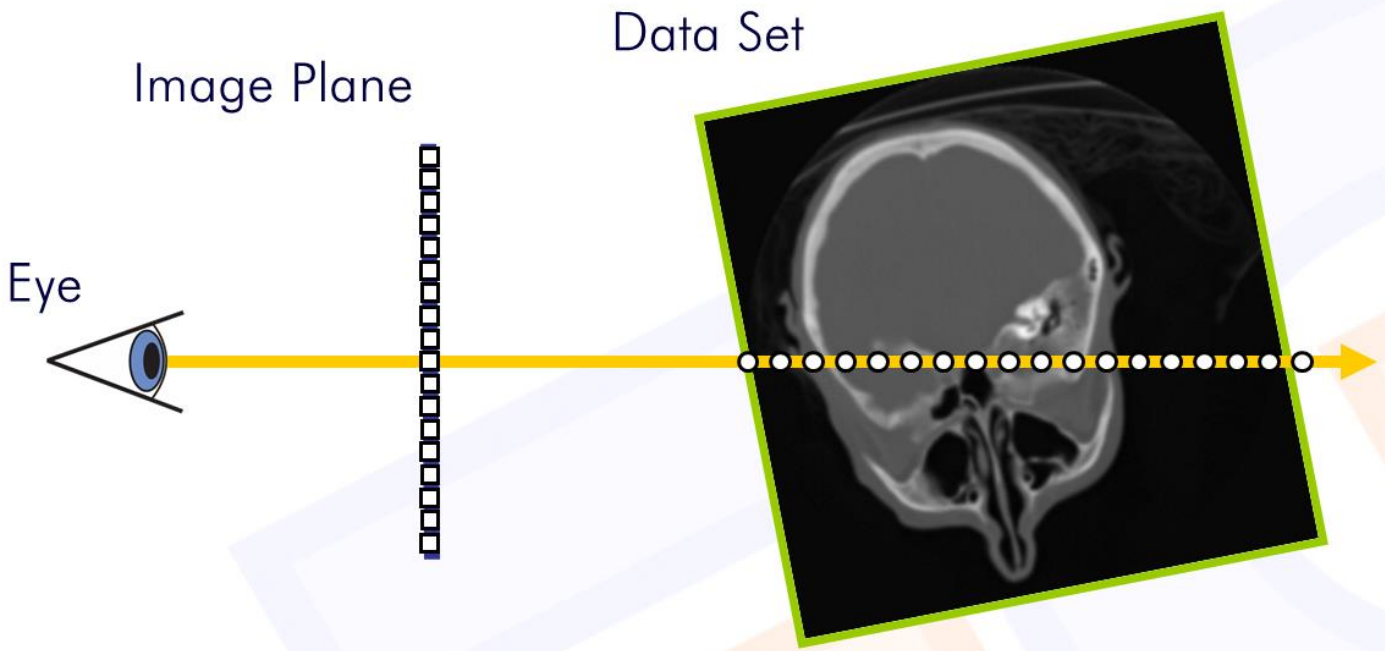


Raycasting Concept



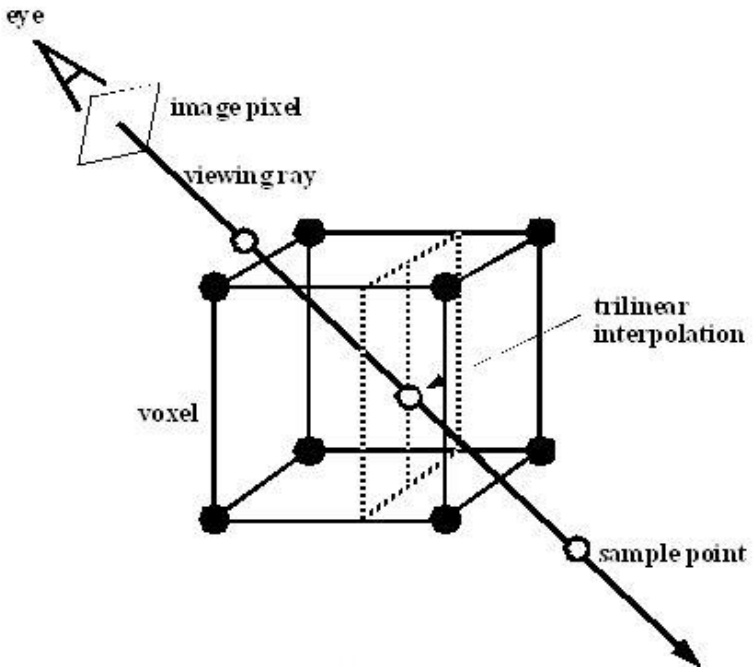
- *Numerical Integration*
- *Resampling*

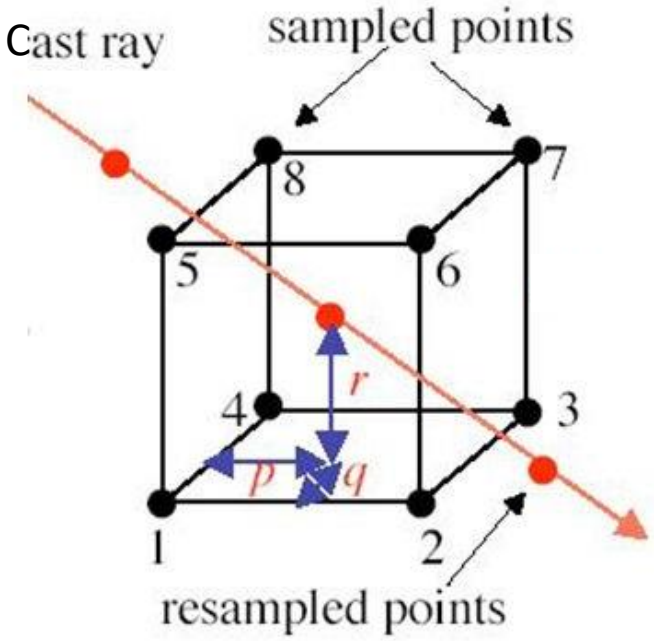
Sampling Along the Ray



Estimate sample values via *interpolation*

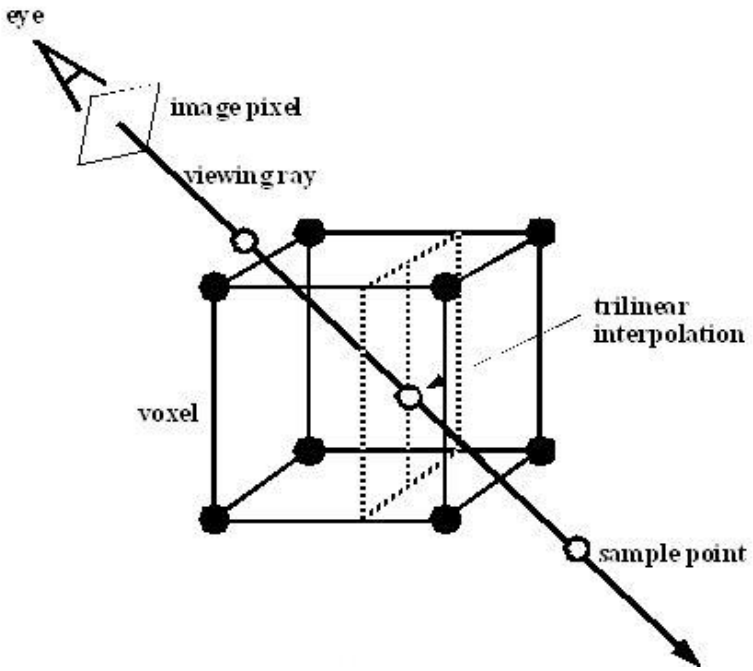
Sampling via Trilinear Interpolation

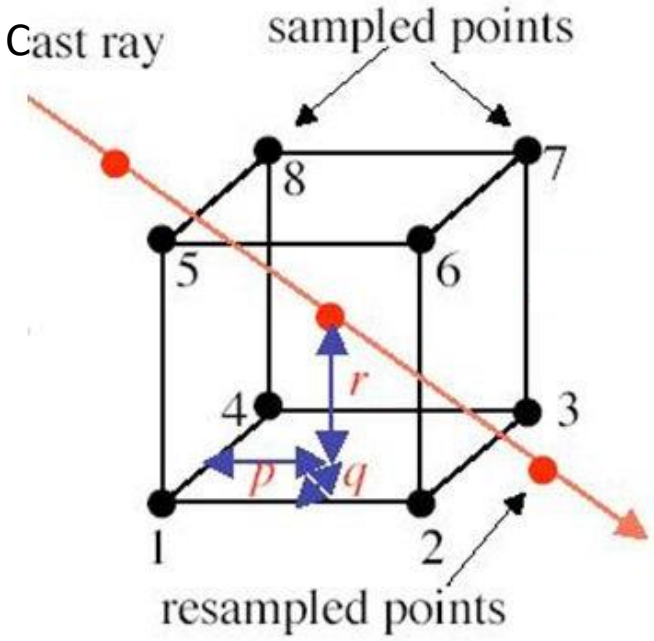




$$f_v = f_1(1-p)(1-q)(1-r) + f_2(p)(1-q)(1-r) + f_3(p)(q)(1-r) + f_4(1-p)(q)(1-r) + f_5(1-p)(1-q)(r) + f_6(p)(1-q)(r) + f_7(p)(q)(r) + f_8(1-p)(q)(r)$$

Sampling via Trilinear Interpolation

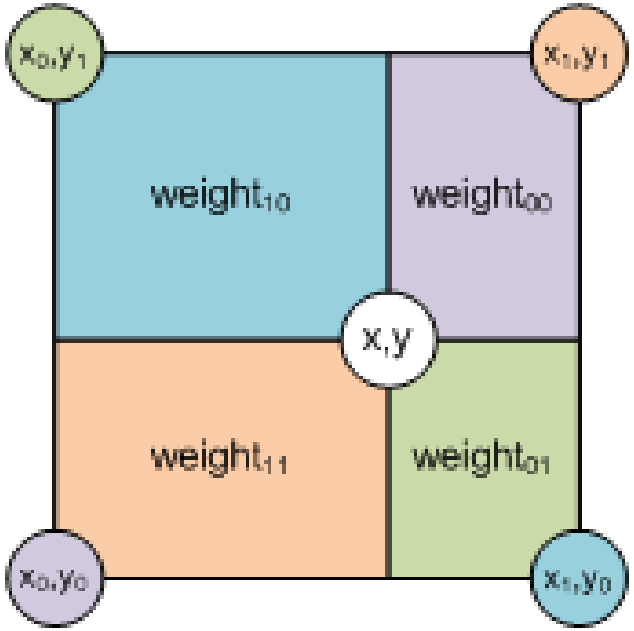




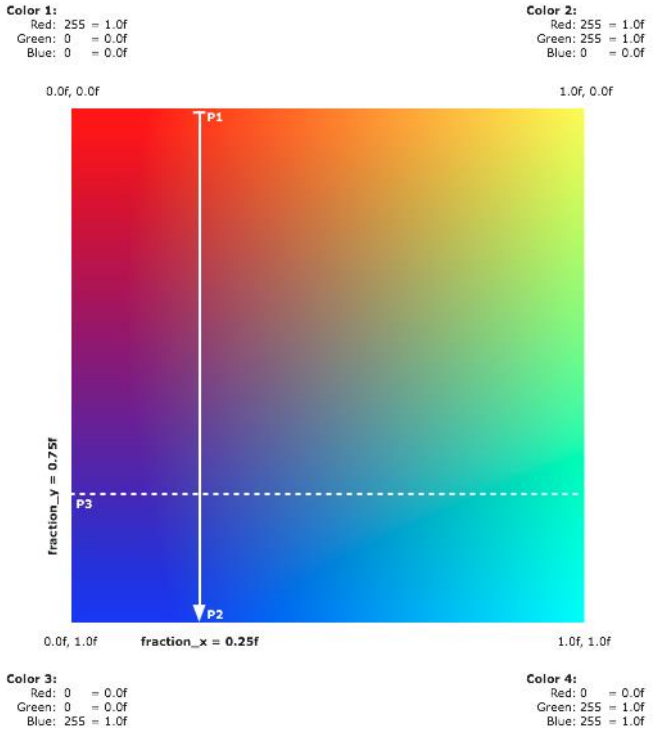
$$f_v = f_1(1-p)(1-q)(1-r) + f_2(p)(1-q)(1-r) + f_3(p)(q)(1-r) + f_4(1-p)(q)(1-r) + f_5(1-p)(1-q)(r) + f_6(p)(1-q)(r) + f_7(p)(q)(r) + f_8(1-p)(q)(r)$$

What Does This Exactly Mean?

Here is what it looks like in 2D for bi-linear interpolation



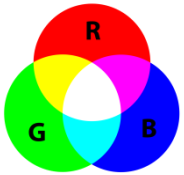
weights



interpolation result within one cell

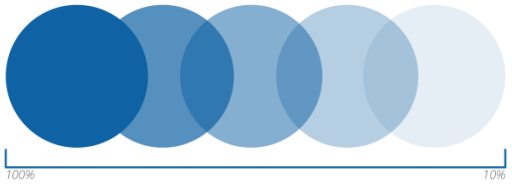
Transparency and Opacity

We learned about RGB



There is one more channel – opacity (A)

- gives RGBA color
- opacity (A) = 1 – transparency (T)
- range [0.0 … 1.0]



Opacity (A) multiplied by RGB creates a weighting effect

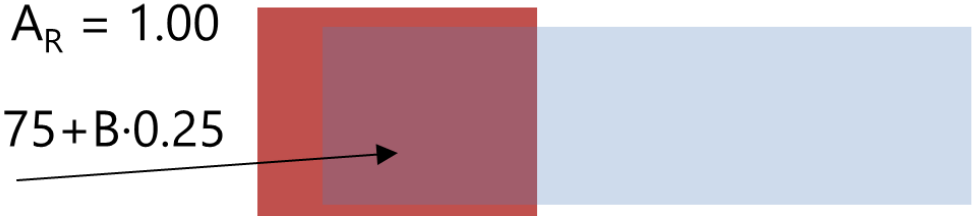
Opacity and Color Blending

$$C_{mix} = C_{back} A_{back} (1 - A_{front}) + C_{front} A_{front}$$

$$C_{mix} = C_R A_R (1 - A_B) + C_B A_B$$

$T_R = 0.00$, $A_R = 1.00$

$C = R \cdot 0.75 + B \cdot 0.25$

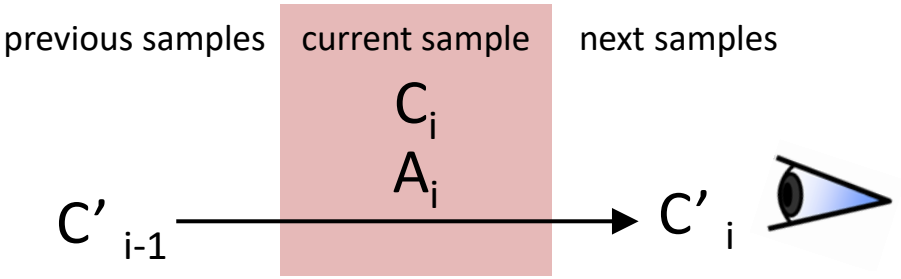


$T_B = 0.75$

$A_B = 0.25$

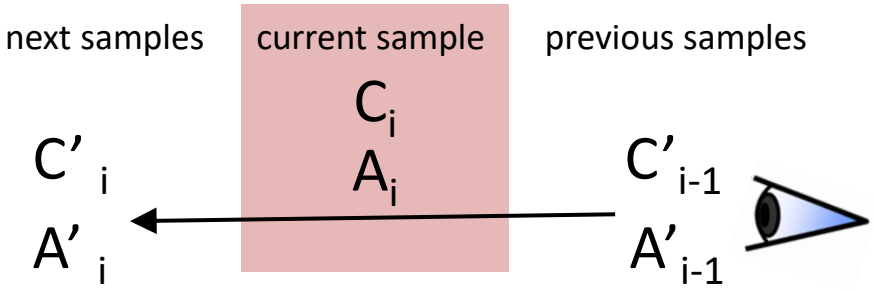
Compositing – Merging the Samples

Back-to-front rendering



$$C_i' = C_i A_i + (1 - A_i) C_{i-1}'$$

Front-to-back rendering



$$C_i' = C_{i-1}' + (1 - A_{i-1}') C_i A_i$$

$$A_i' = A_{i-1}' + (1 - A_{i-1}') A_i$$

A: Opacity = 1- Transparency = 1 - T

C: Color

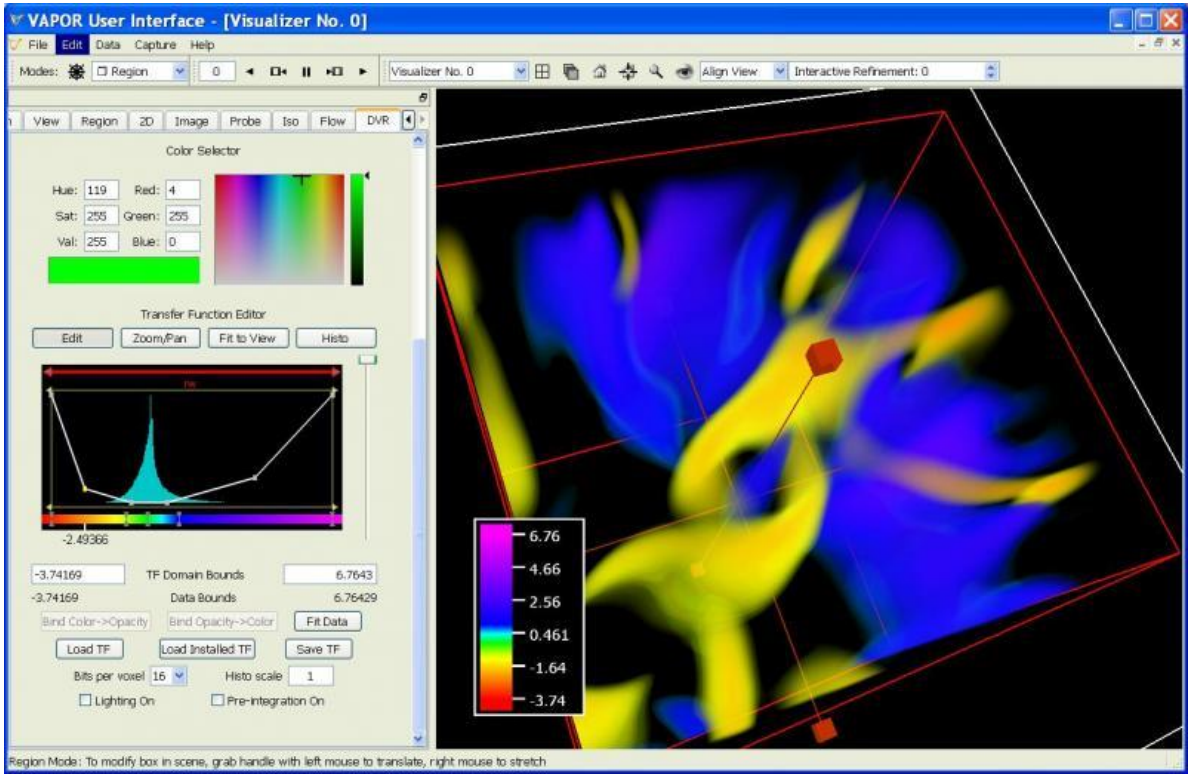
Transfer Function

Determines what color & opacity a sample value should have

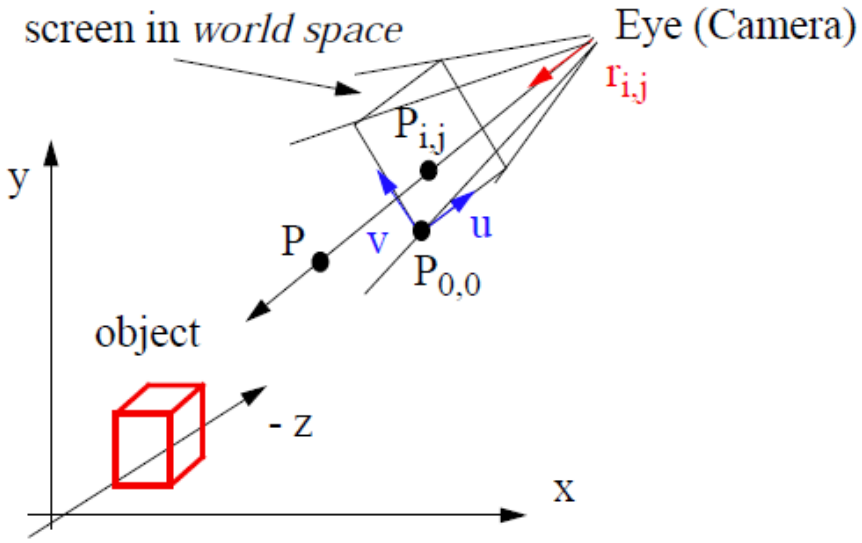
- input: an interpolated density value
- output: a color and opacity (RGBA)

rendering ↓

transfer function →



Raycasting Specifics



A ray is specified by:

- eye position (Eye)
- screen pixel location $P_{i,j}$

→ ray direction vector ($r_{i,j}$) of unit length

$$r_{i,\,j} = \frac{P_{i,\,j} - Eye}{\left| P_{i,\,j} - Eye \right|}$$

A point P on a ray is given by:

$$P = Eye + t \cdot r_{i,\,j}$$

t: parametric variable

Spacing of pixels on image plane:

$$\Delta i = \frac{W}{Ni - 1} \qquad \Delta j = \frac{H}{Nj - 1}$$

Ni, Nj: image dims. in pixels

Image-order projection:

- scan the image row by row, column by column:

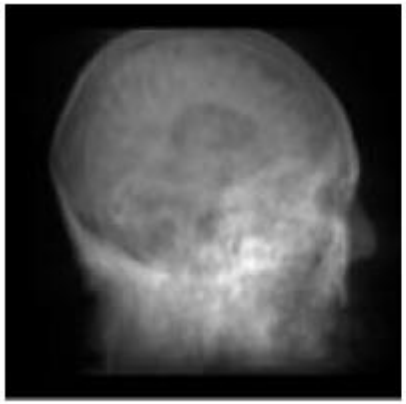
$$P_{i,\,j} = P_{0,\,0} + i \cdot v \cdot \Delta j + j \cdot u \cdot \Delta i$$

- $P_{i,\,j}$: Location of image pixel (i, j) in world space

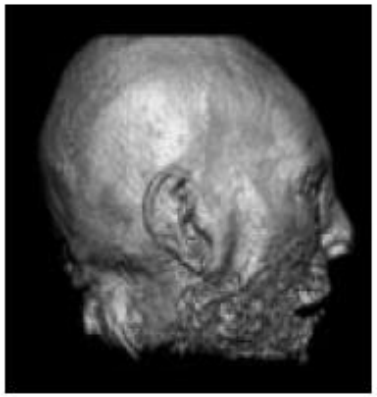
$0 \le i < Ni \quad 0 \le j < Nj$

- $P_{0,\,0}$: image (=screen) origin in world space
- u, v, n: orthonormal image plane vectors ($n = v \times u$)

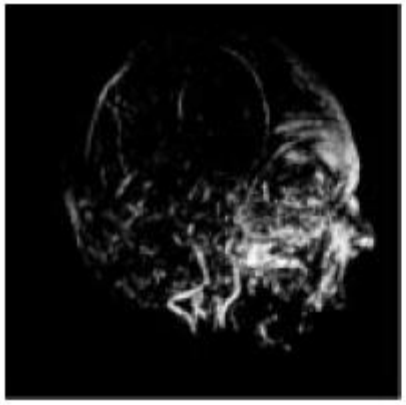
Volume Rendering Modes



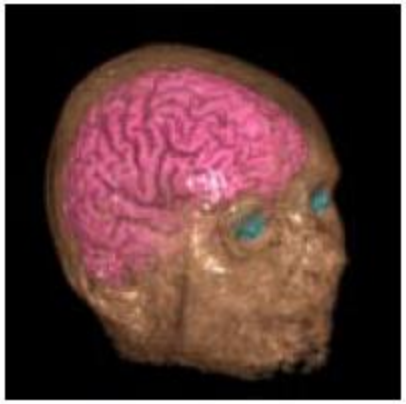
X-ray:
rays sum volume cor
tributions along their
linear paths



Iso-surface:
rays look for the object
surfaces, defined by a cer-
tain volume value



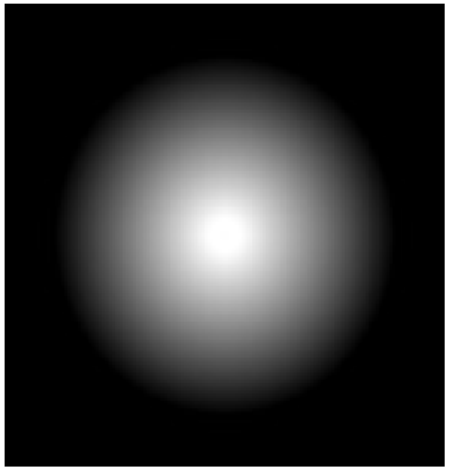
Maximum Intensity Pro
jection (MIP):
a pixel value stores th
largest volume value
along its ray



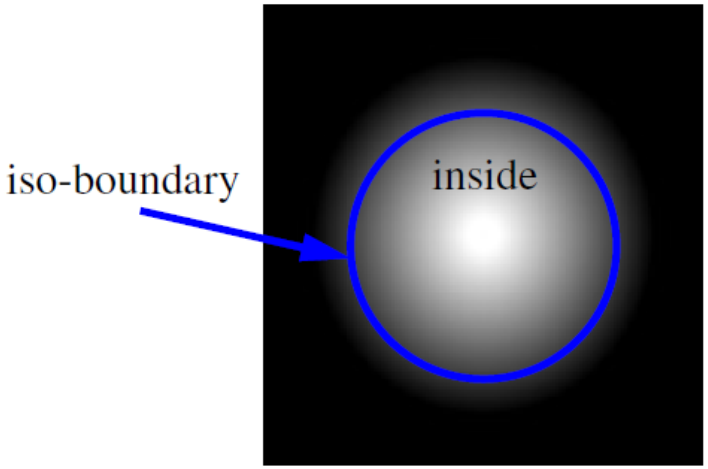
Full volume rendering:
rays *composite* volume
contributions along their
linear paths

Iso-Surface Rendering

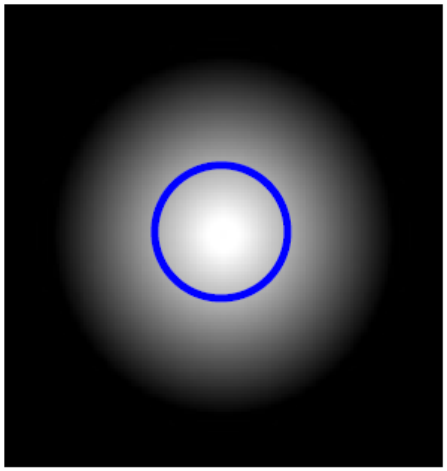
- A closed surface separates 'outside' from 'inside' (Jordan theorem)
- In iso-surface rendering we say that all voxels with values > some threshold are 'inside', and the others are 'outside'
- The boundary between 'outside' and 'inside' is the *iso-surface*
- All voxels near the iso-surface have a value close to the *iso-threshold* or *iso-value*
- Example:



cross-section of a smooth sphere

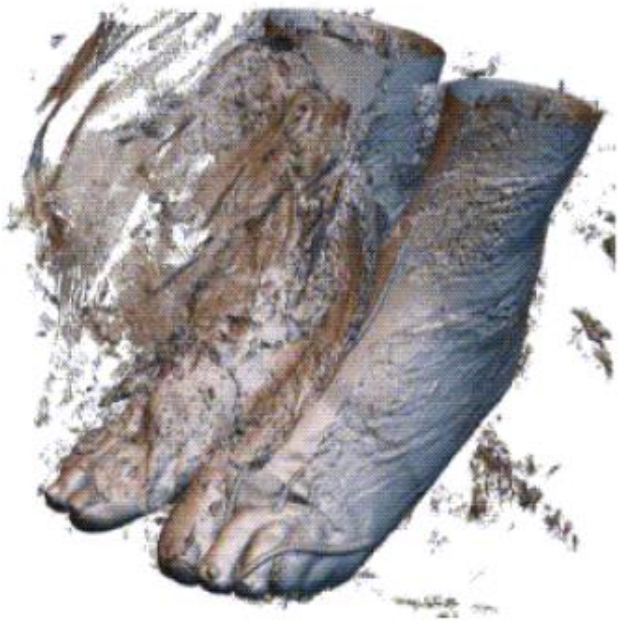


iso-value = 50
will render a large sphere



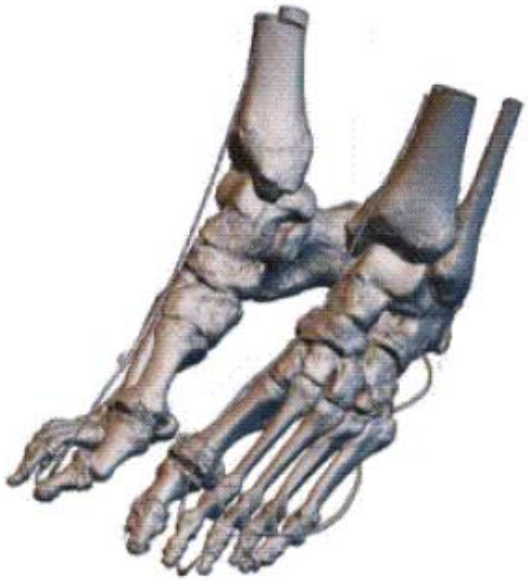
iso-value = 200
will render a small sphere

Iso-Surface Rendering



iso-value = 30

iso-value = 80

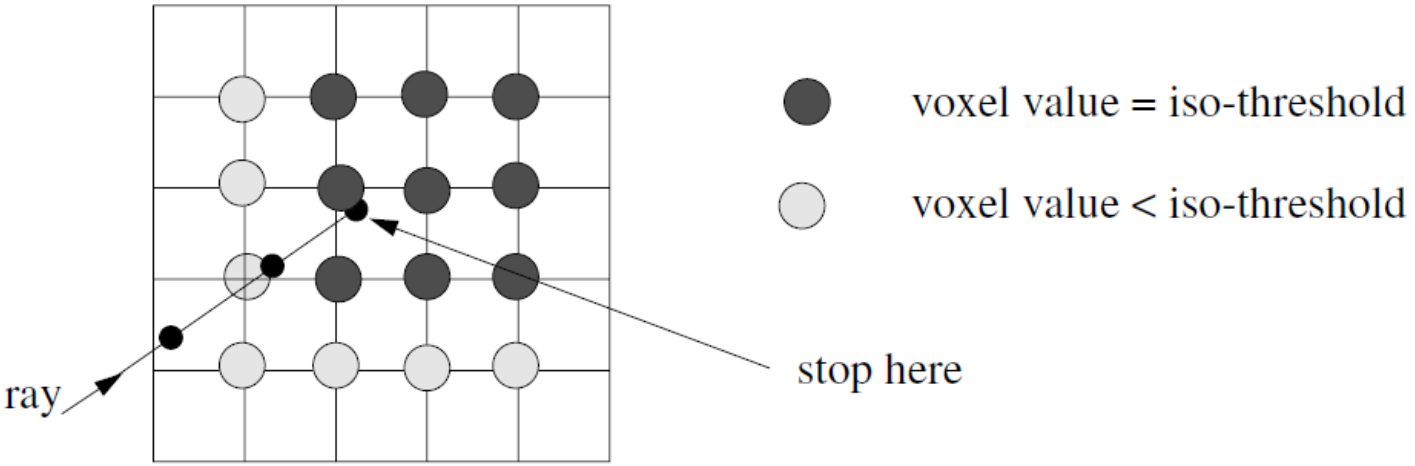


iso-value = 200

Iso-Surface Rendering – Details

- To render an iso-surface we cast the rays as usual...

  but we stop, once we have interpolated a value iso-threshold

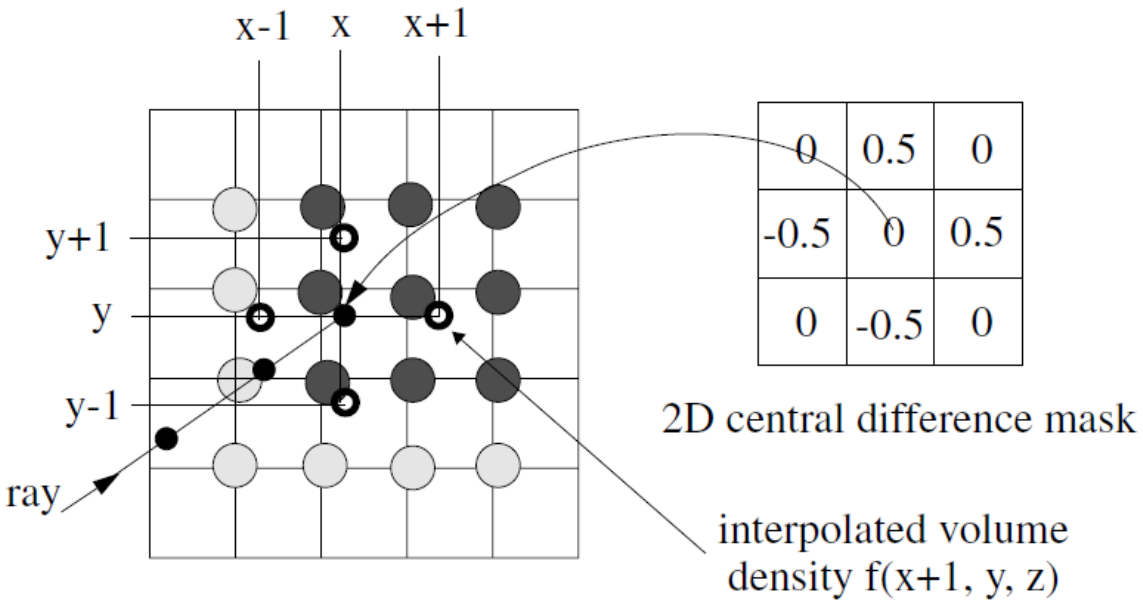


- We would like to illuminate (shade) the iso-surface based on its orientation to the light source
- Recall that we need a normal vector for shading
- The normal vector N is the local gradient, normalized

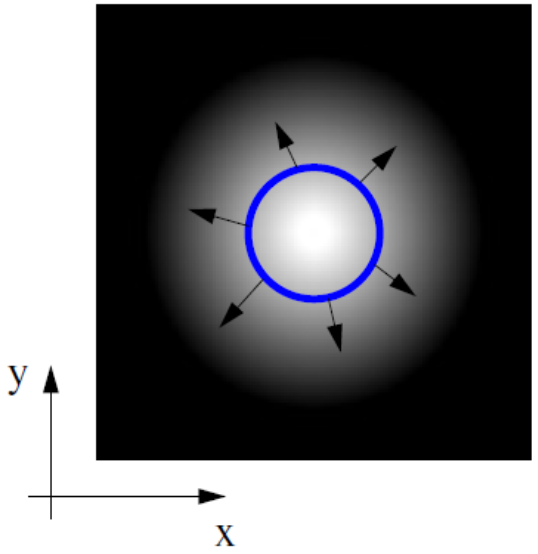
The Gradient Vector

- The gradient vector $\mathbf{g}=(g_x, g_y, g_z)^T$ at the sample position (x, y, z) is usually computed via central-differencing (for example, g_x is the volume density gradient in the x-direction):

$$g_x = \frac{f(x-1, y, z) - f(x+1, y, z)}{2} \qquad g_y = \frac{f(x, y-1, z) - f(x, y+1, z)}{2} \qquad g_z = \frac{f(x, y, z-1) - f(x, y, z+1)}{2}$$



the x and y component of the gradient vector for the smooth sphere

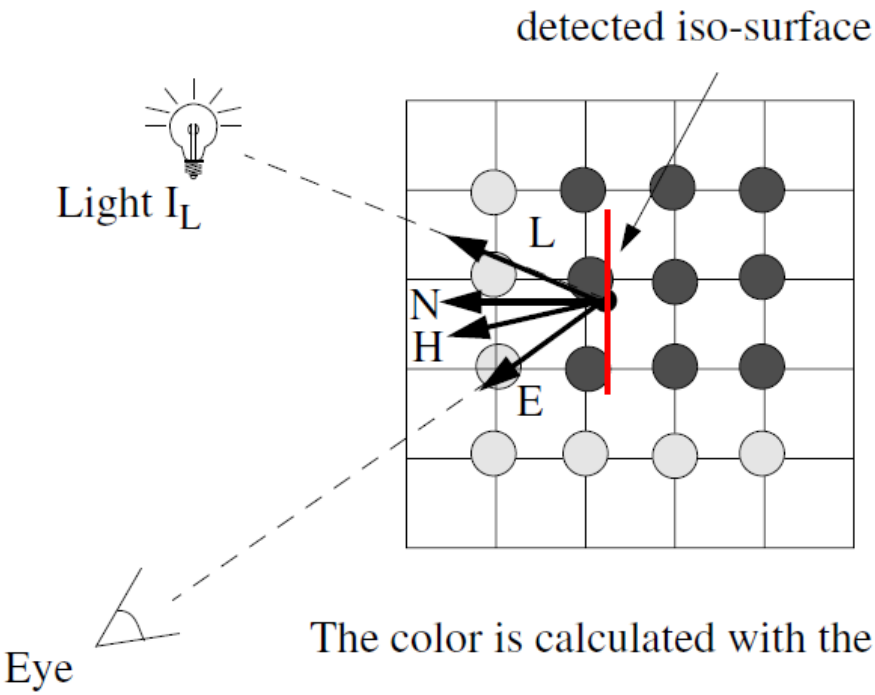


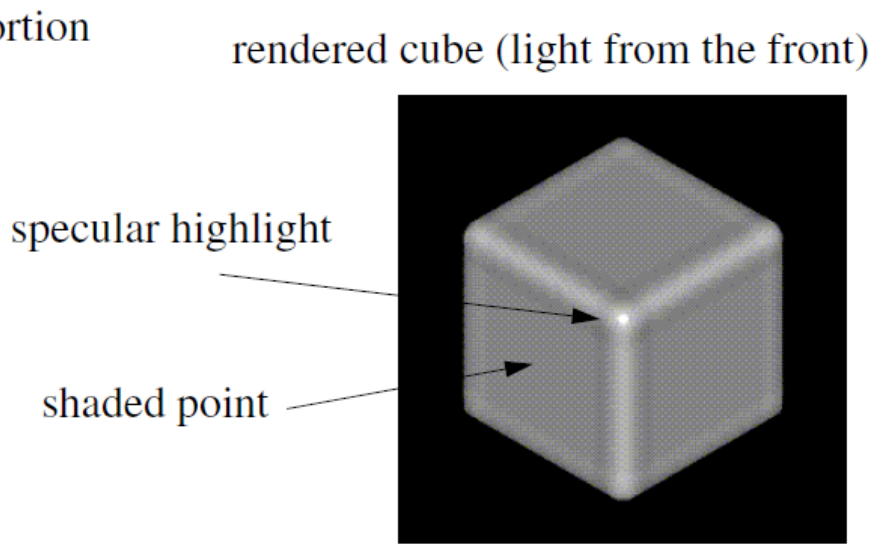
Shading the Iso-Surface

- The normal vector is the *normalized* gradient vector g

  N = g / |g| (normal vector always has unit length)

- Once the normal vector has been calculated we shade the iso-surface at the sample point
- The color so obtained is then written to the pixel that is due to the ray





The color is calculated with the standard shading equation:

$$C = C_{obj} (k_a I_A + k_d I_L N \cdot L) + k_s I_L (H \cdot N)^{ns}$$

C_{obj} is obtained by indexing the color transfer function with the interpolated sample value

Full Volume Rendering

When hitting a surface set A< 1.0

- ray marches on
- inner structures can be seen

What is It?

10 petaFLOPS Titan supercomputer (released in 2012)

- 1 petaFLOP = 10^{15} floating point ops per second

18,688 AMD Opteron 6274 16-core CPUs

18,688 Nvidia Tesla K20X GPUs

Even Faster Now…

Summit supercomputer (2018, #1 worldwide, Oak Ridge Nat'l Lab)

- 200 petaFLOPS (2x the top speed of TaihuLight, previous #1)
- 4,608 compute servers (each with two 22-core IBM Power9 processors and six NVidia Tesla V100 GPUs)

What Does it Do?

Compute, compute, compute

Examples:

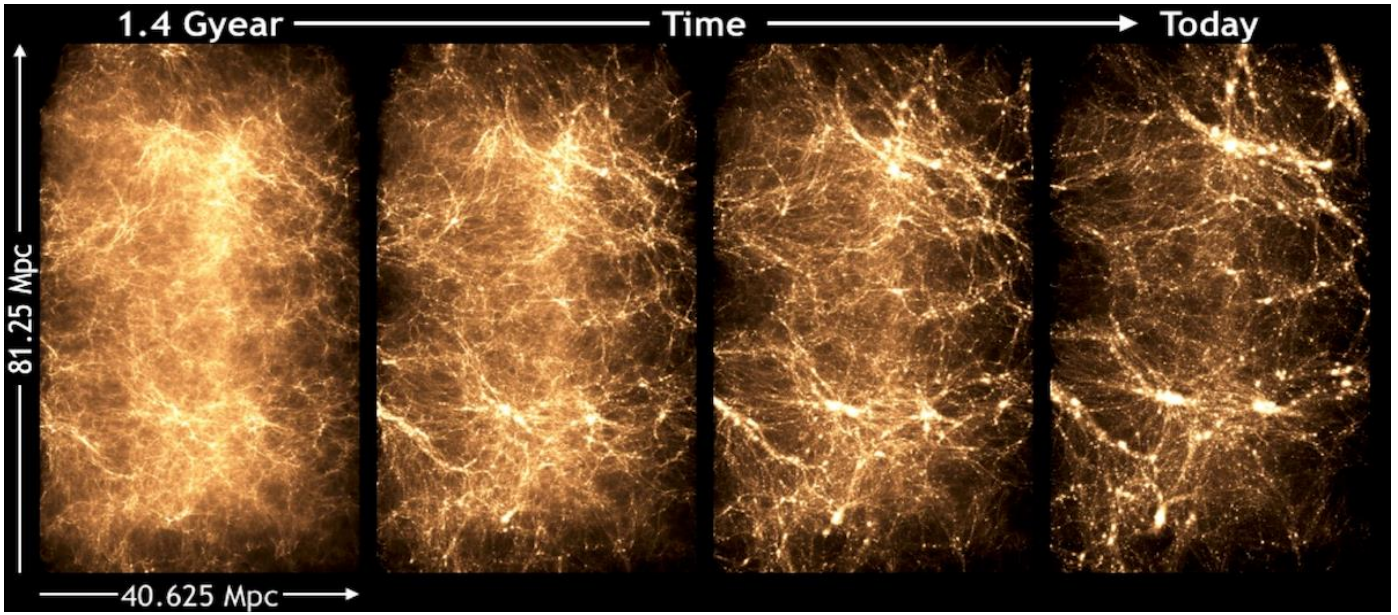
- **S3D:** models the molecular physics of combustion, aims to improve the efficiency of diesel and biofuel engines
- **Denovo:** simulates nuclear reactions with the aim of improving the efficiency and reducing the waste of nuclear reactors
- **WL-LSMS:** simulates the interactions between electrons and atoms in magnetic materials at temperatures other than absolute zero
- **Bonsai:** simulates the Milky Way Galaxy on a star by star basis, with 200 billion stars
- **Non-Equilibrium Radiation Diffusion (NRDF):** plots non-charged particles through supernovae with potential applications in laser fusion, fluid dynamics, medical imaging, nuclear reactors, energy storage and combustion

What Does it Output

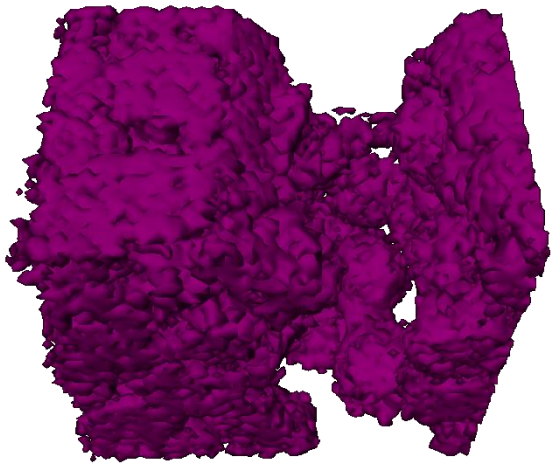
Numbers, lots of them

- Titan's I/O subsystem is capable of pushing around 240 GB/s of data
- that's a lot to visualize

Example: a visualization of the Q Continuum simulation for cosmology

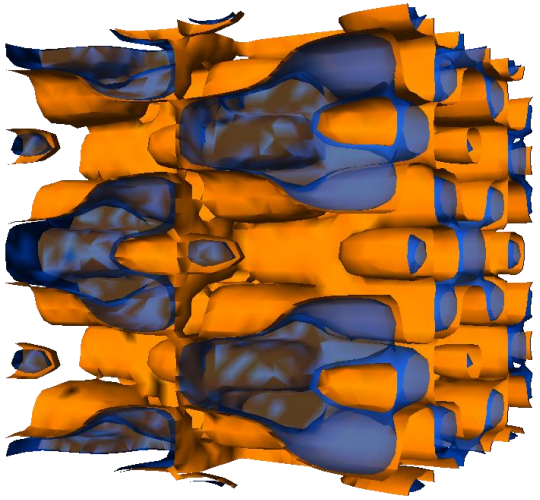


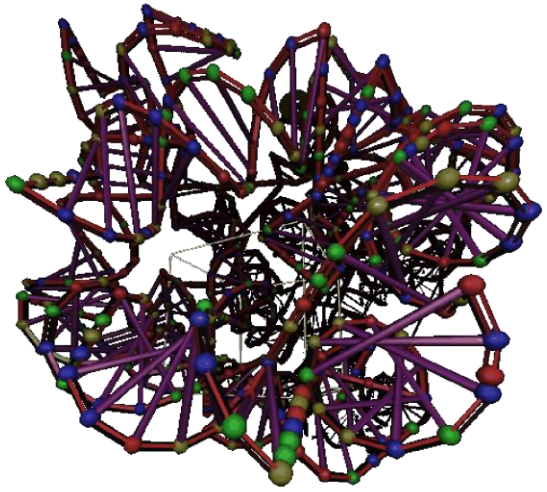
More Examples



Nuclear, Quantum, and Molecular Modeling

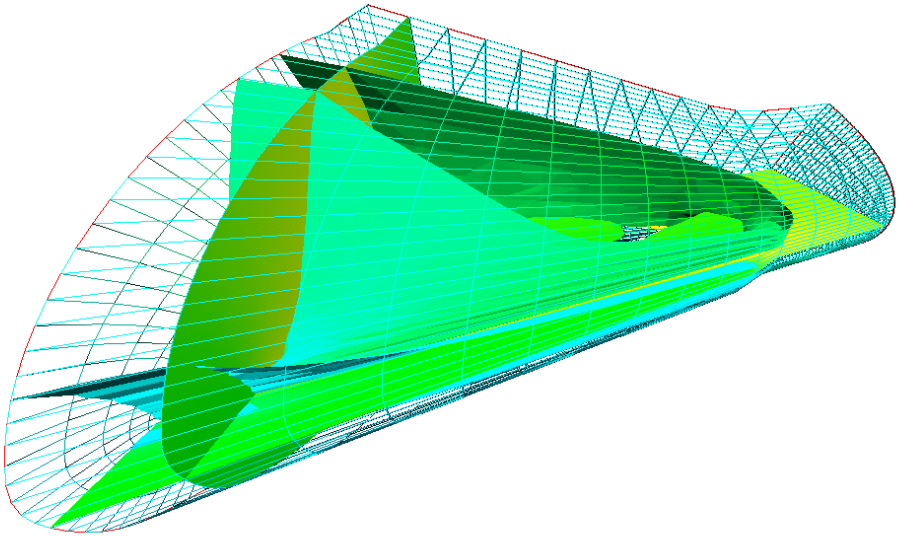
Structures, Fluids and Fields



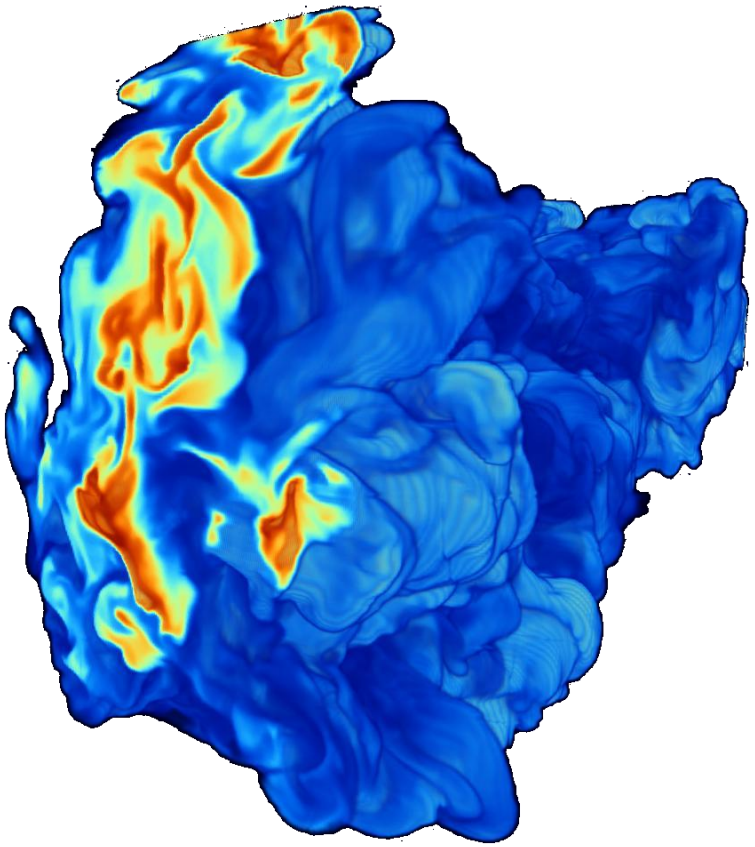


Advanced Imaging and Data Management

More Examples



Surface Rendering with VTK
(The Visualization Toolkit)

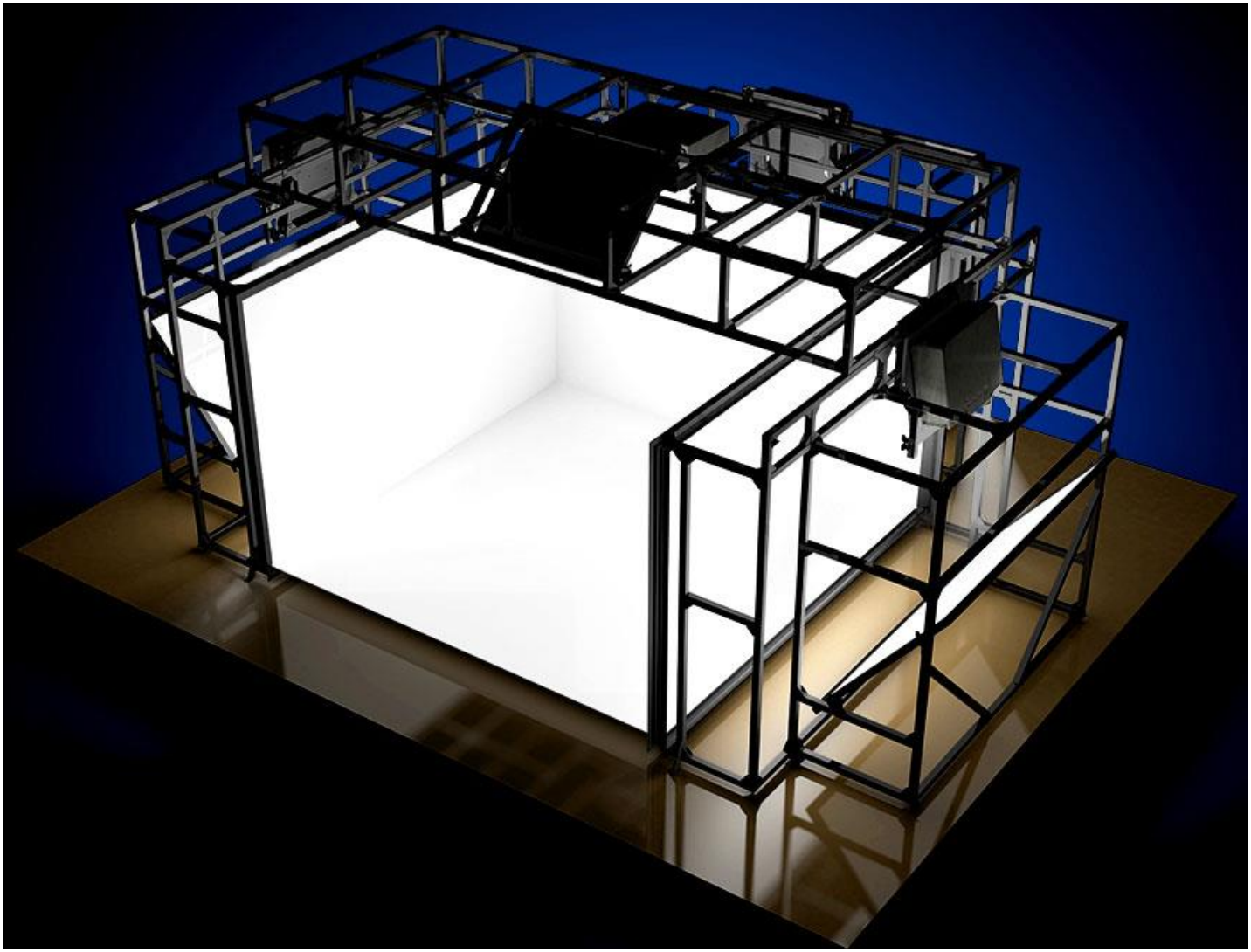


Volume Rendering

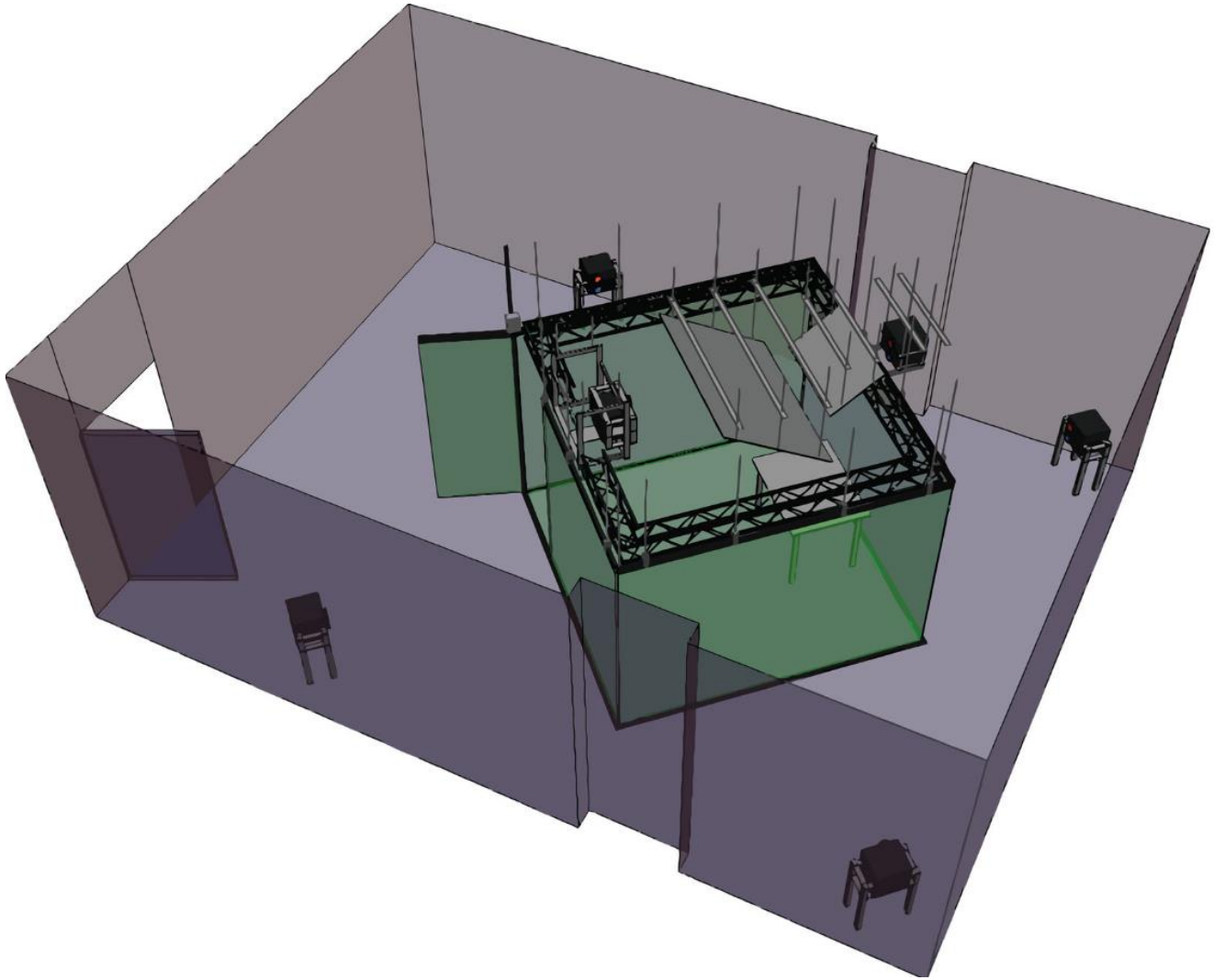
Where to Visualize All This?

Display Wall

CAVE =
Cave Automatic Virtual Environment

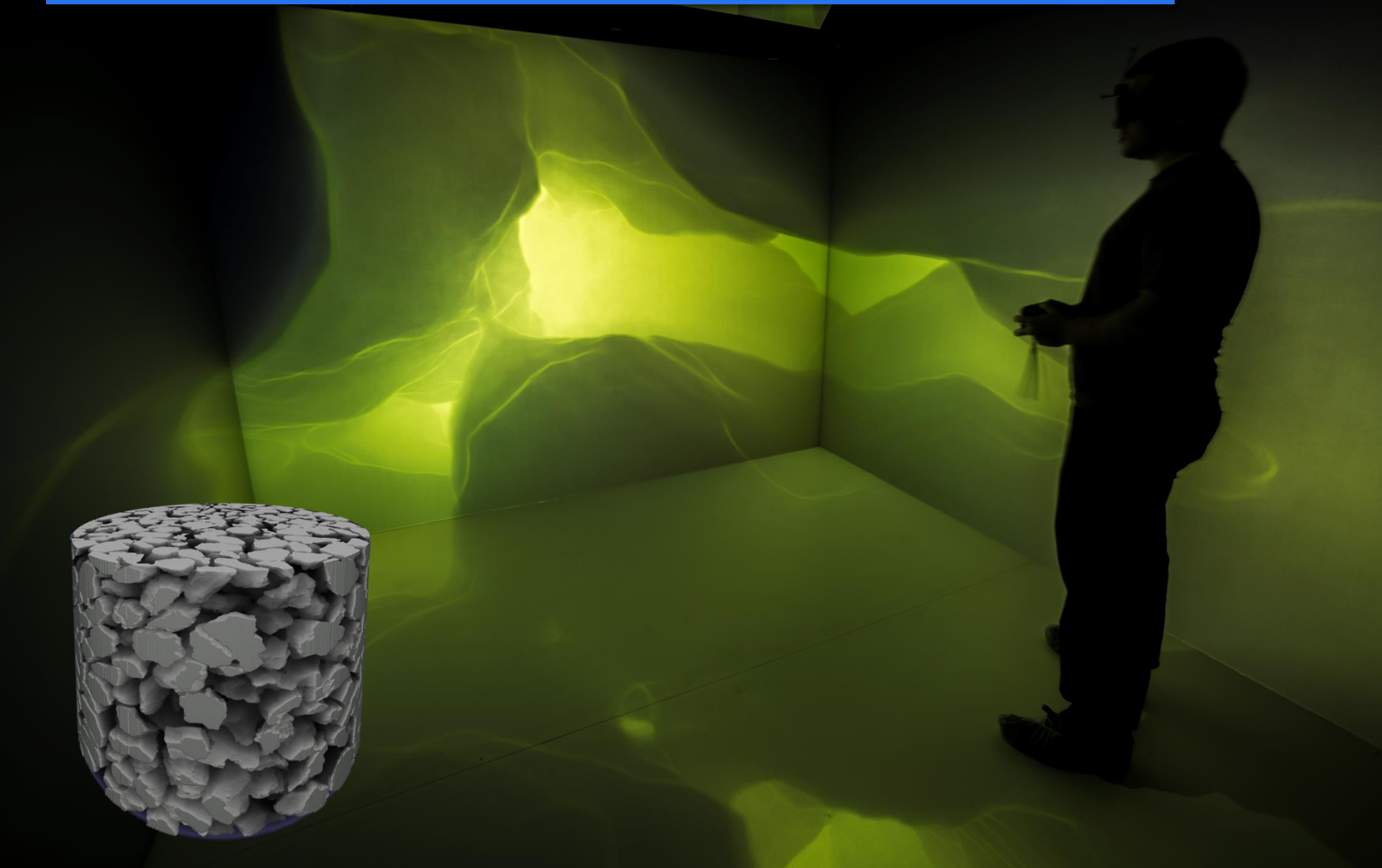


The Stony Brook Immersive Cabin

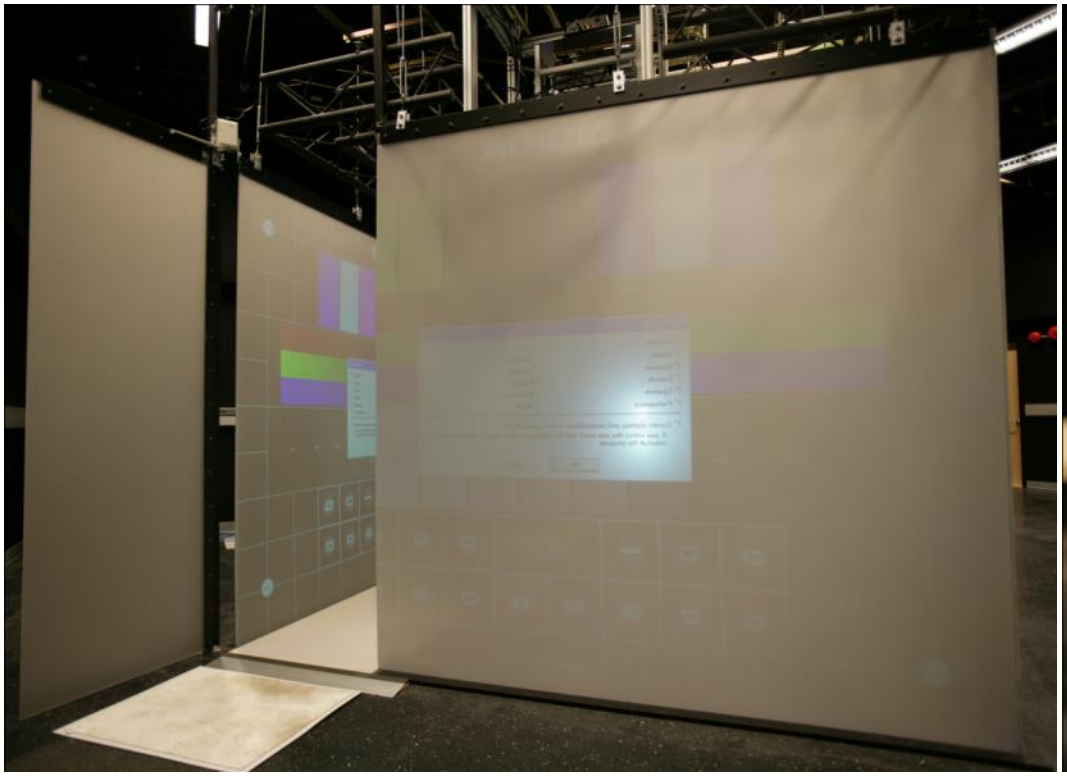


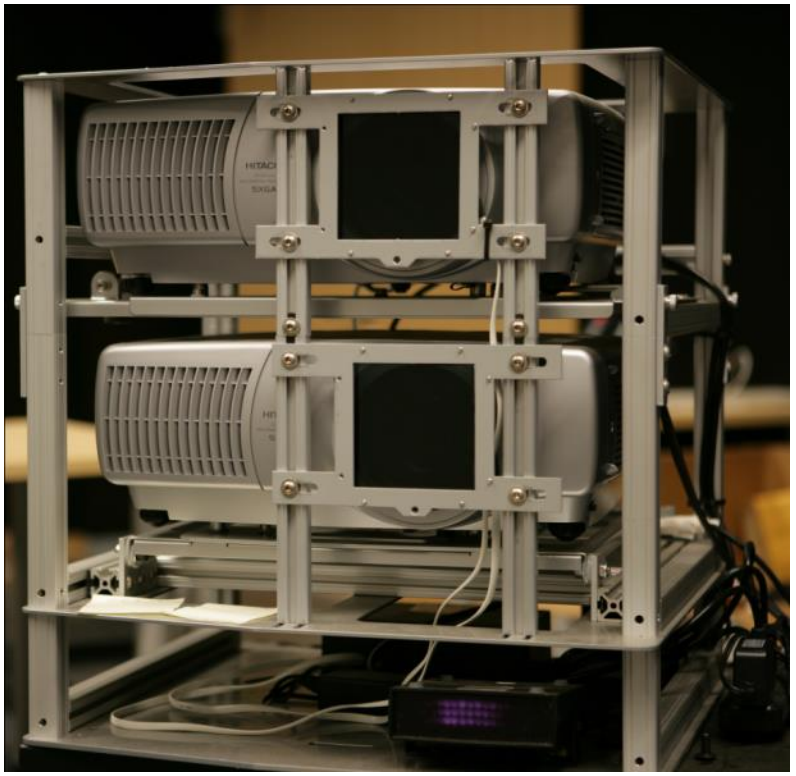
Inside The Immersive Cabin

Microtomography
(BNL, soil sample)



The Stony Brook Immersive Cabin





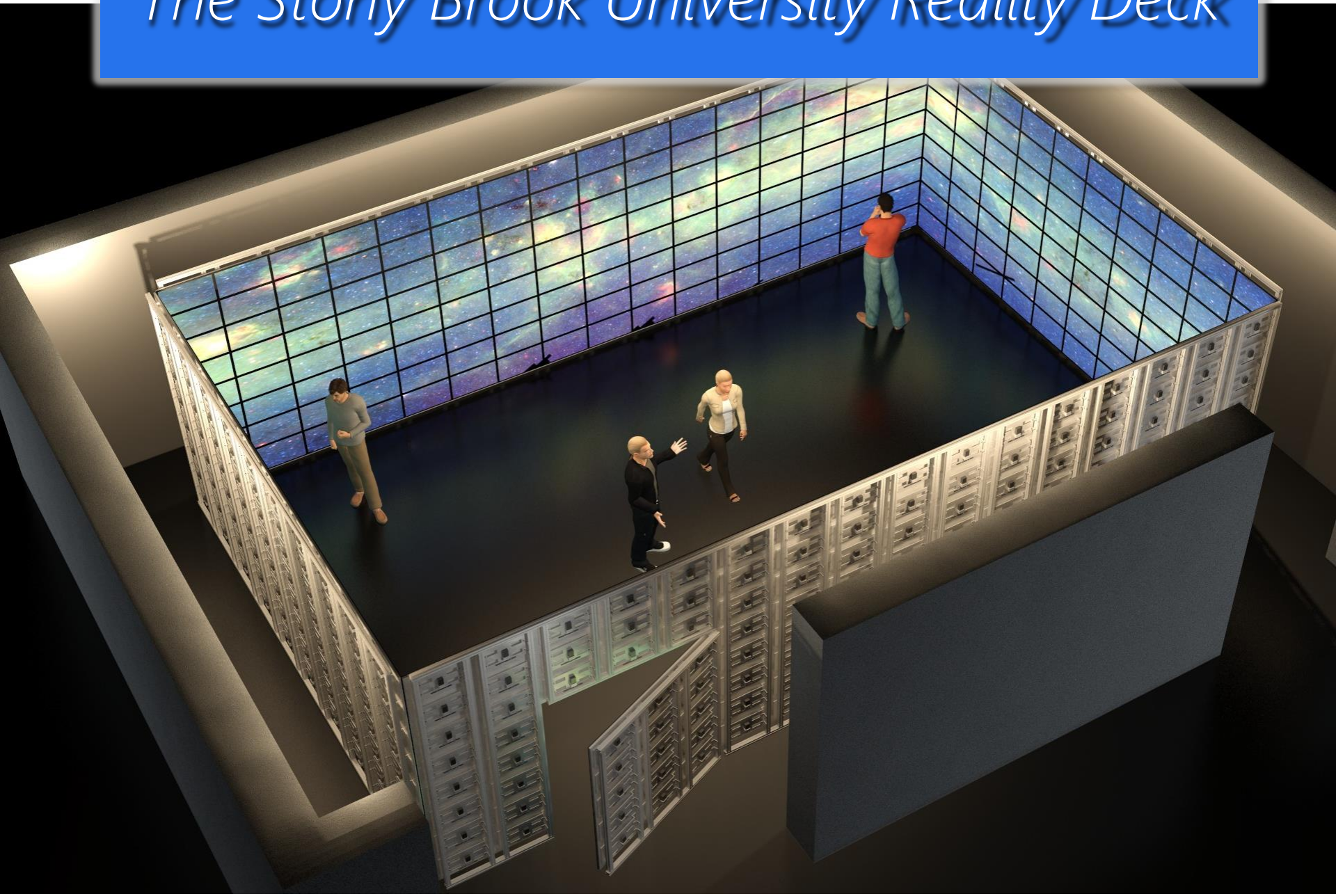
Projector based system

- 5 walls, 12′×12′ footprint, 8′ tall
- difficult to scale up to Giga-pixel range

Can We Get Bigger?

(yes we can)

The Stony Brook University Reality Deck



The Reality Deck – Under the Hood

Visualization

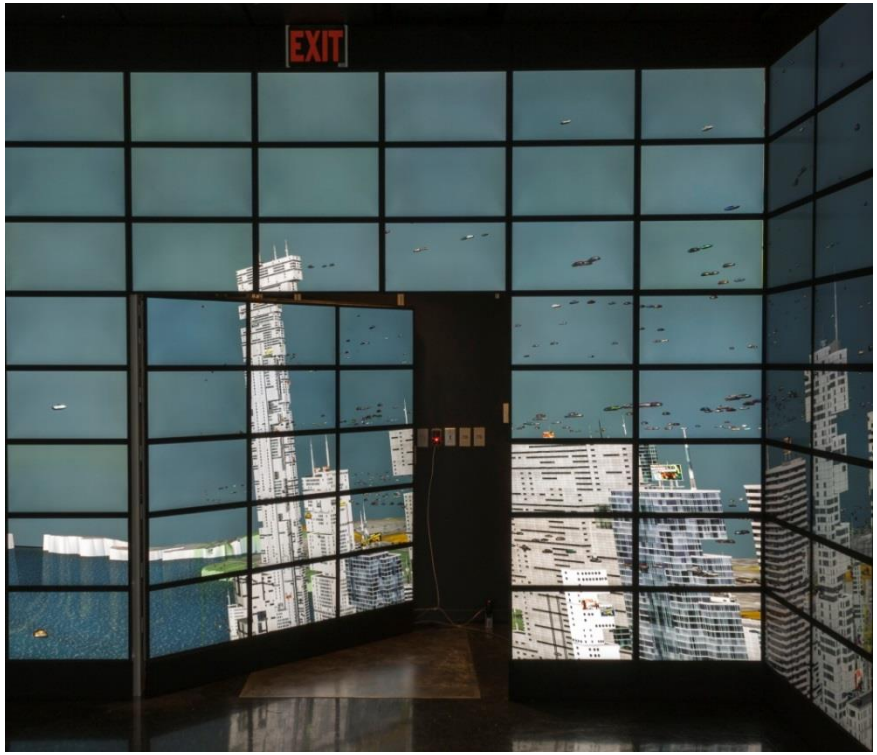
- 30′×40′×11′ environment
- 416 UQXGA LCD Displays
  - 2,560×1,440 resolution over 50’-100’ DisplayPort cables
  - fast response time, wide viewing angles, good dynamic range
- 20-node GPU cluster, each node equipped with:
  - 2× six-core CPUs, 48 GB Ram
  - 4× AMD FirePro V9800 with 4GB Ram and 6 DisplayPort outputs each
  - AMD S400 hardware video synchronization card
  - 40Gb Infiniband adapter
  - 1TB storage
- In total:
  - 1,533,542,400 pixels (1.5 Gigapixel) over 6 miles of DisplayPort cables
  - 240 CPU cores: 2.3 TFLOPs peak performance, 20 TB distributed memory
  - 80 GPUs: 220 TFLOPs peak performance, 320 GB distributed memory

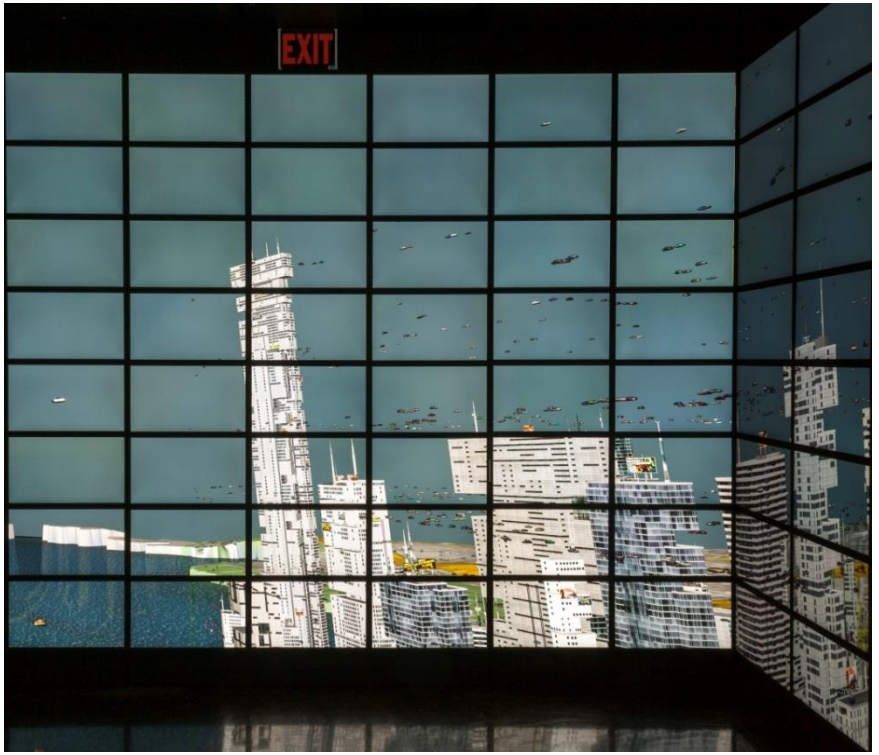
Automatic Door

3×5 section of displays

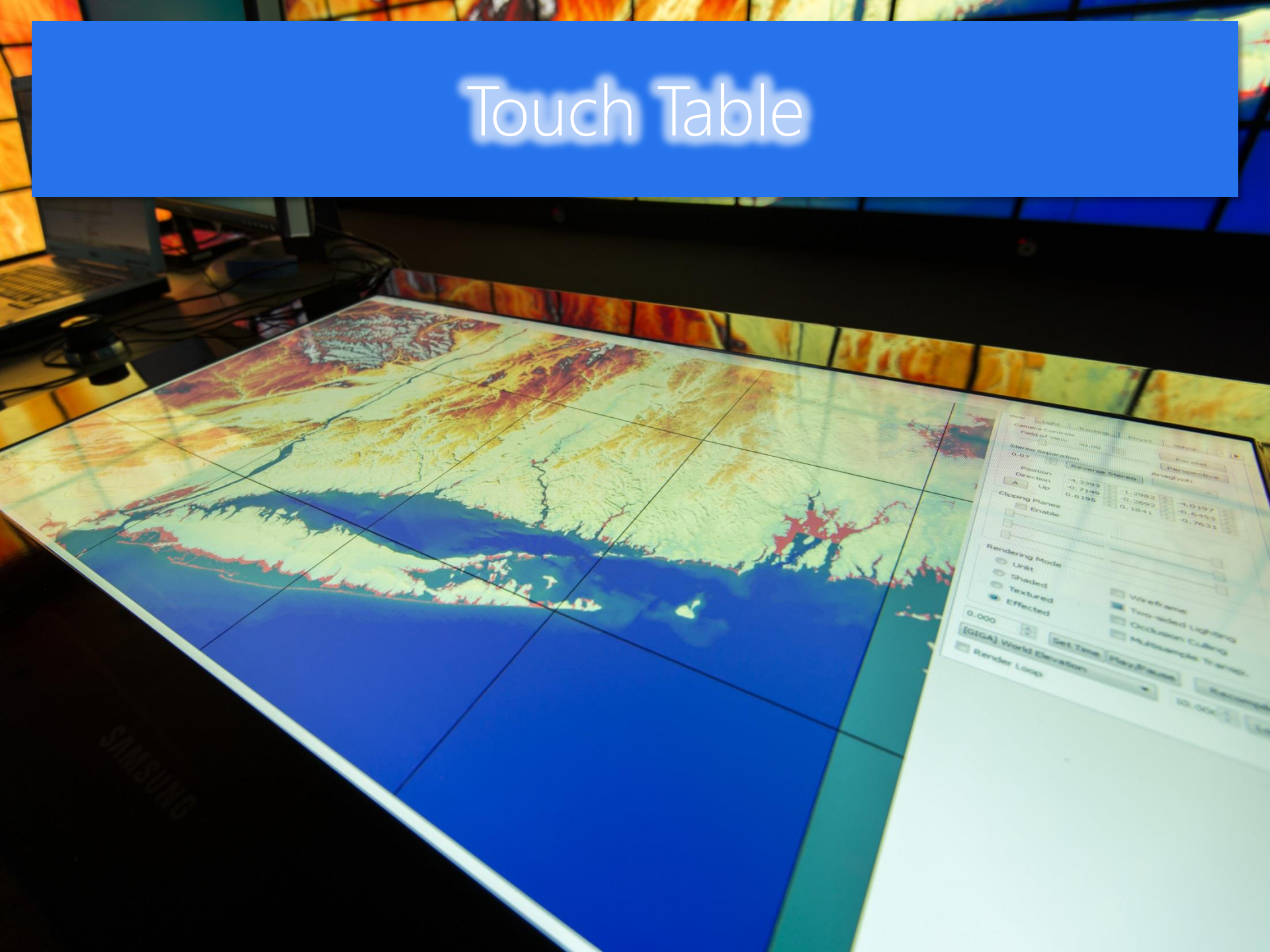
Visually indistinguishable from rest of the display

- allows for a fully enclosed visualization environment



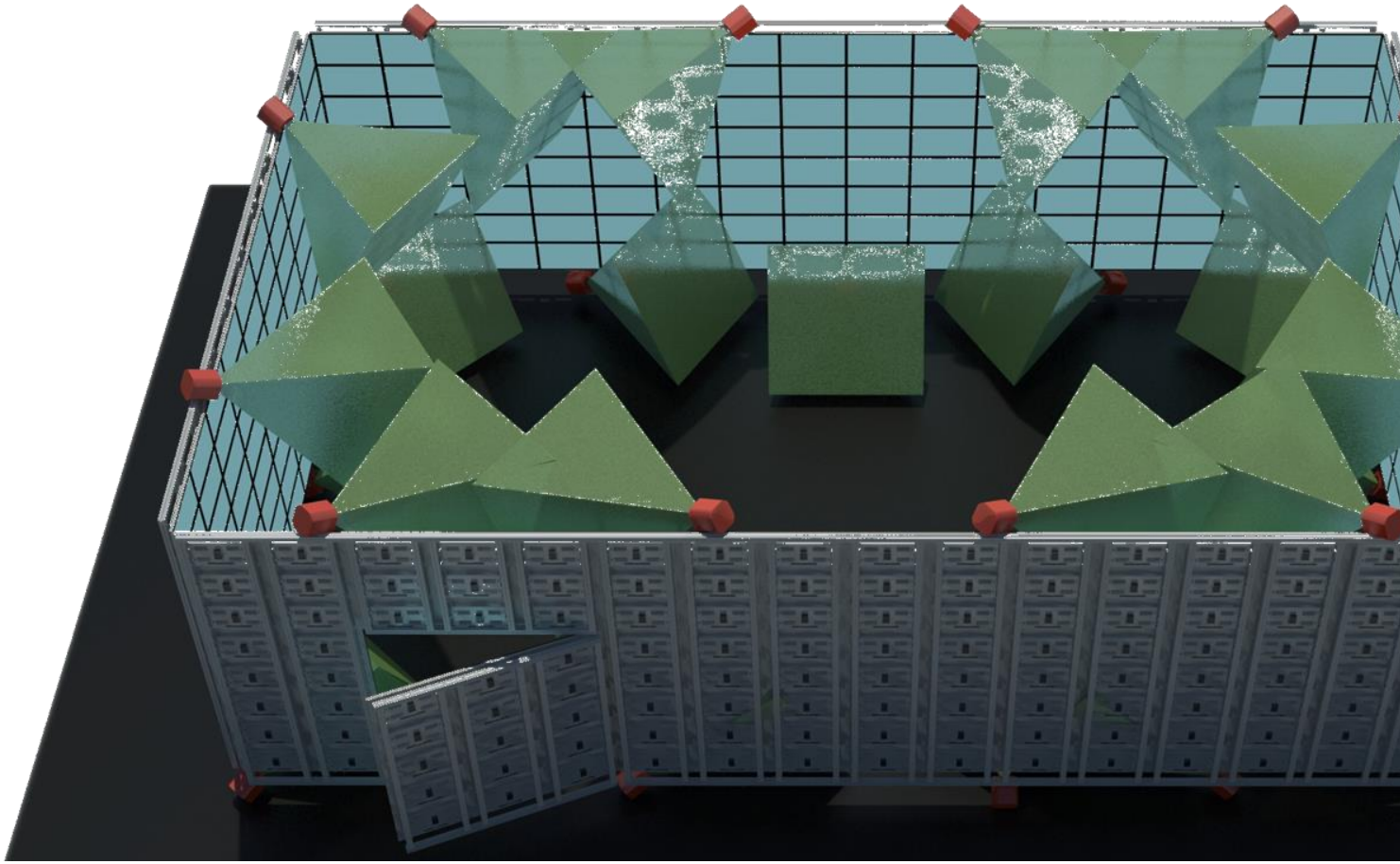


Touch Table



Reality Deck Tracking System

24-camera infrared optical system from OptiTrack



Reality Deck Sound System

24.4 channel professional-grade system

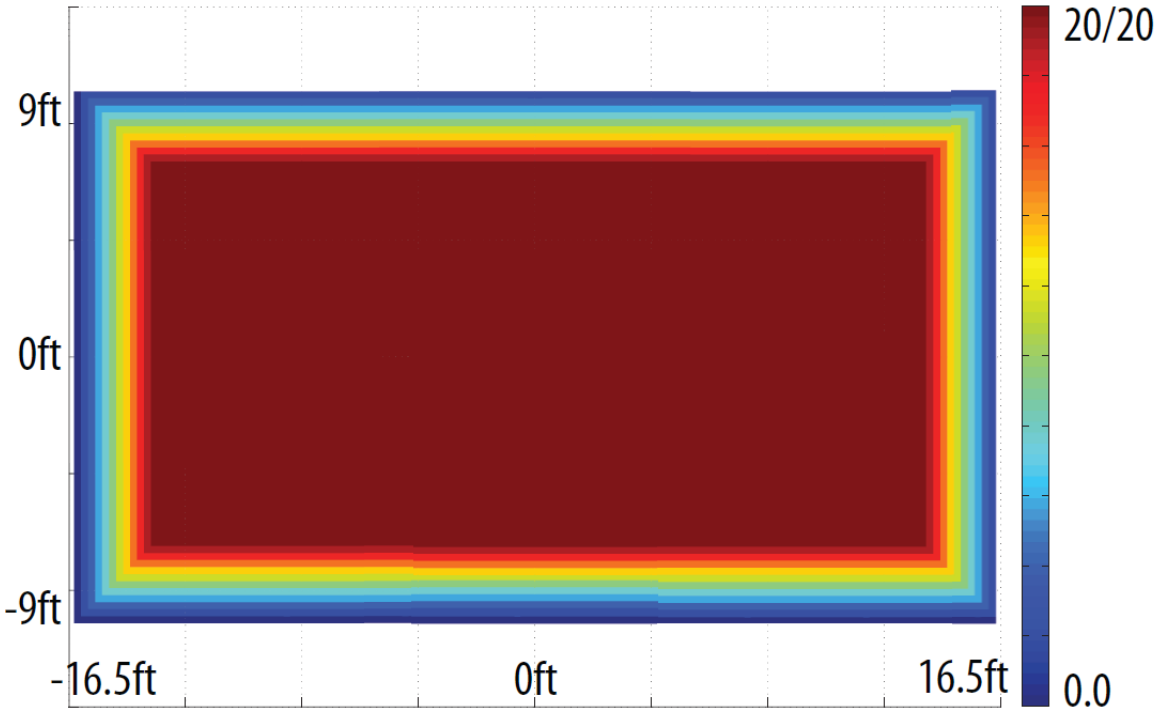
Positional audio with real-time ambisonics

- using the Rapture3D OpenAL driver

Uniformly High Visual Acuity

User can make visual queries at an instant

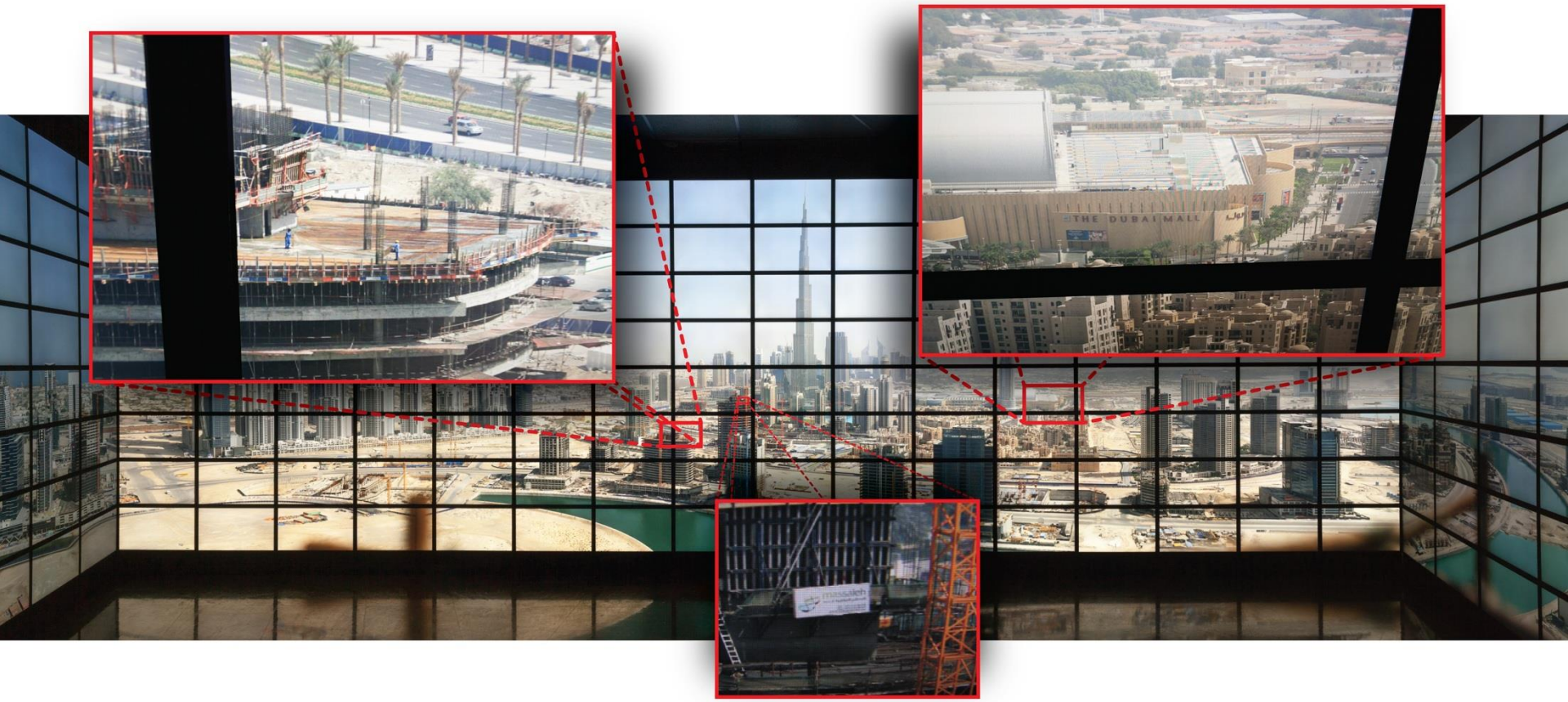
- walk up to obtain more detail
- just like in real life – hence the Reality Deck
- 20/20 visual acuity at 1.5’-2’ away



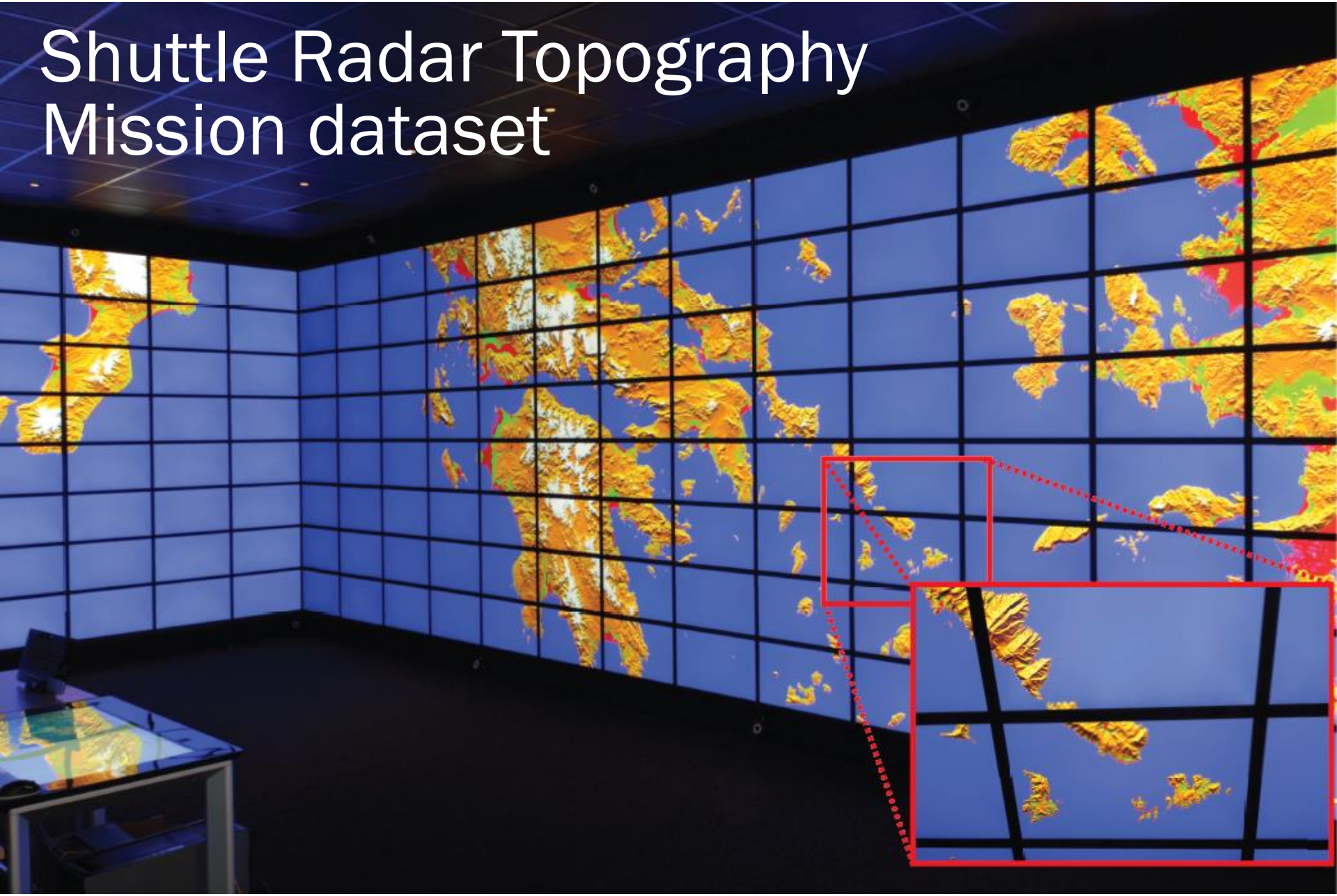
Gigapixel Visualization

Dubai dataset

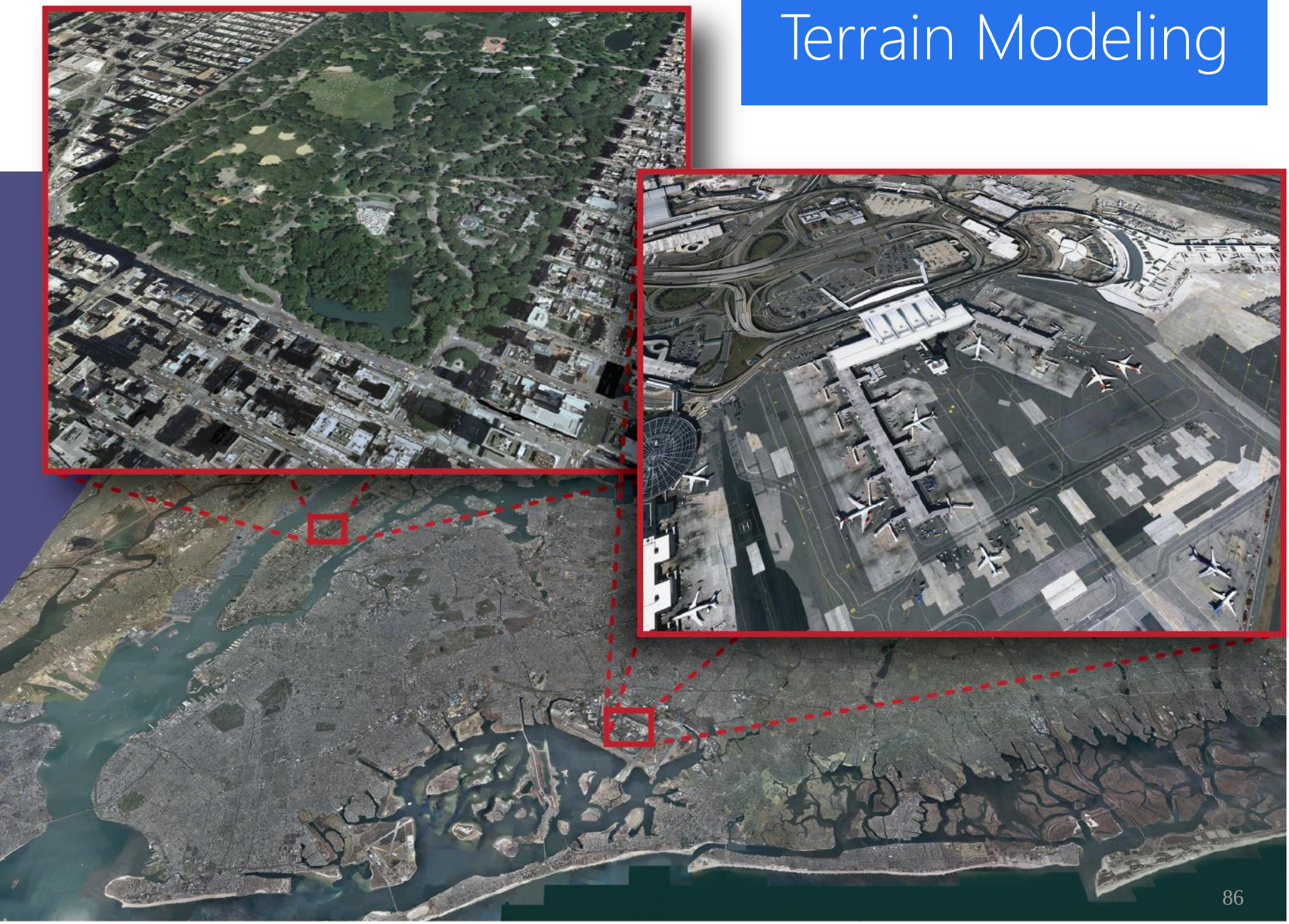
- 45 Gigapixels, 180° field of view



Shuttle Radar Topography Mission dataset



Terrain Modeling



3D Relief Map

Sea level simulation

Protein Visualization

Reality Deck

Scientific Simulation

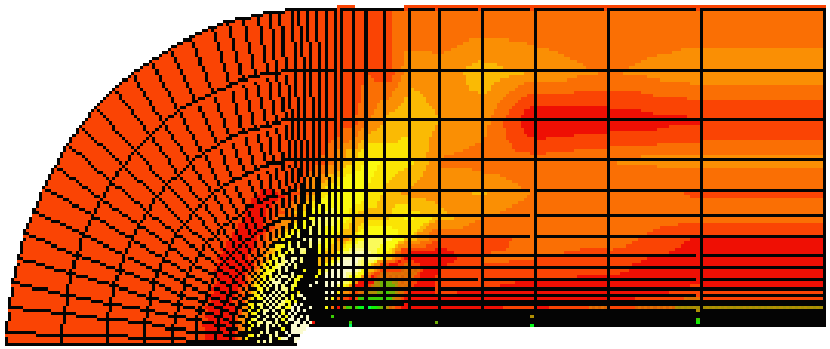
Say, you want to simulate the airflow around an airplane wing

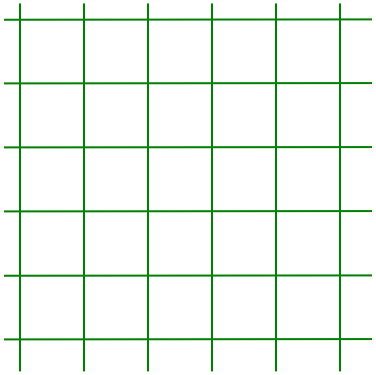
- where is the flow most interesting?

- right, close to the surface

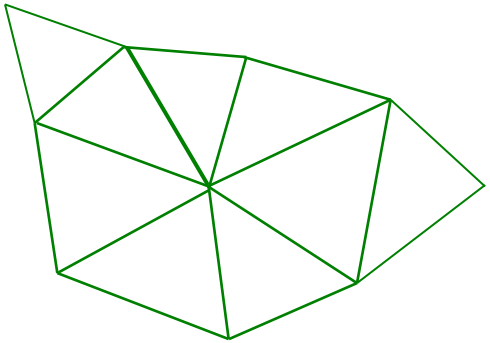
Simulation lattice

Make the simulation lattice densest along the surface





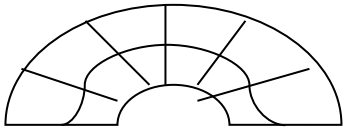
Regular → irregular grids



Grids

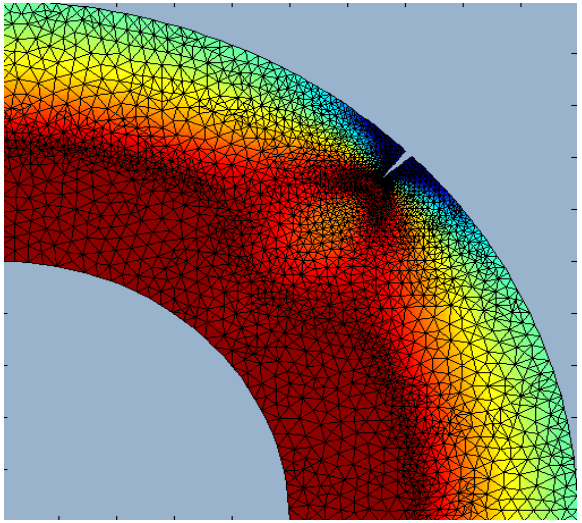
Structured grid

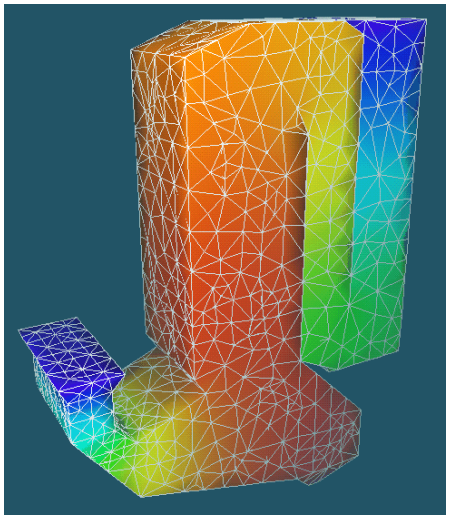
- more or less a bent regular grid

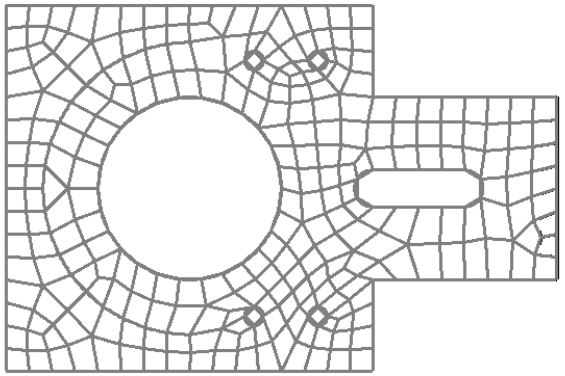


Unstructured grid

- collection of vertices, edges, faces and cells whose connectivity information must be explicitly stored





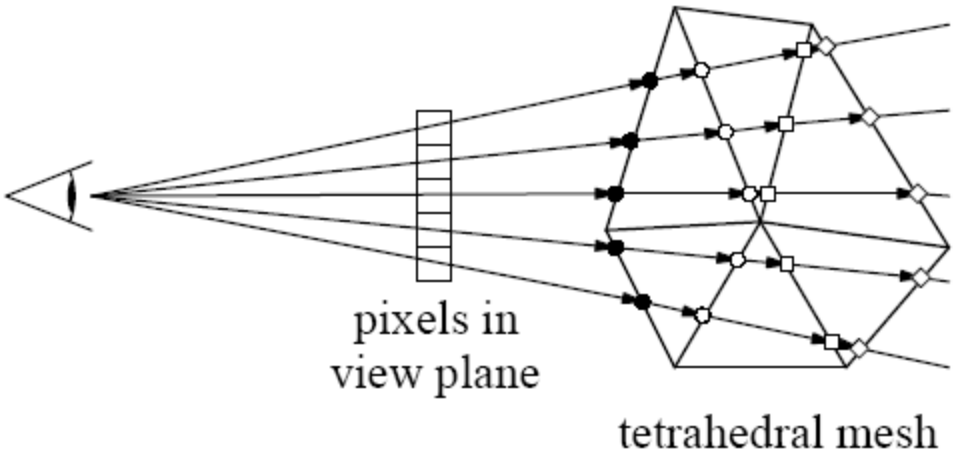


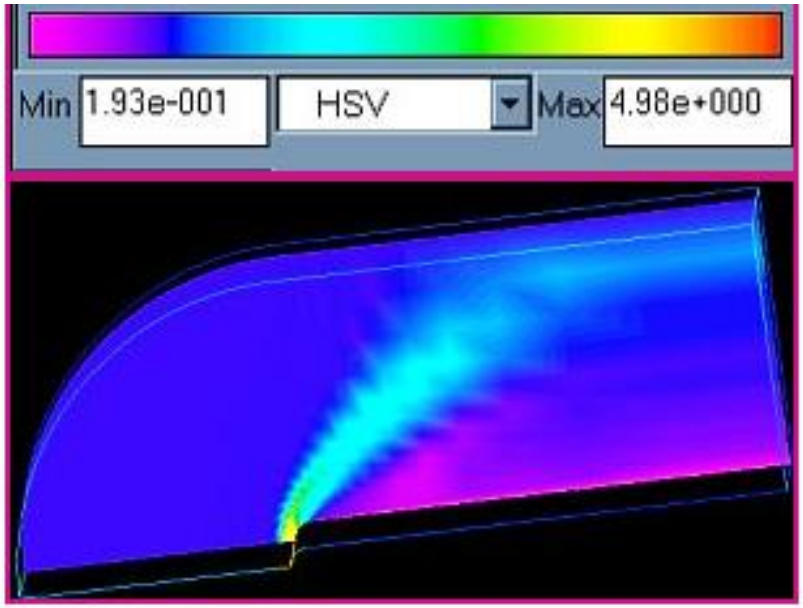
The Bluntfin Dataset

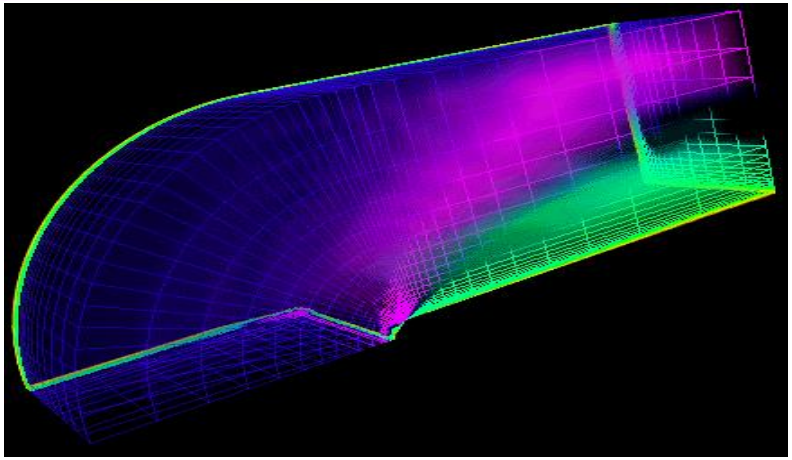
Mapping flow strength to color

Rendering by cell traversal

- go from cell to cell
- composite colors and opacities

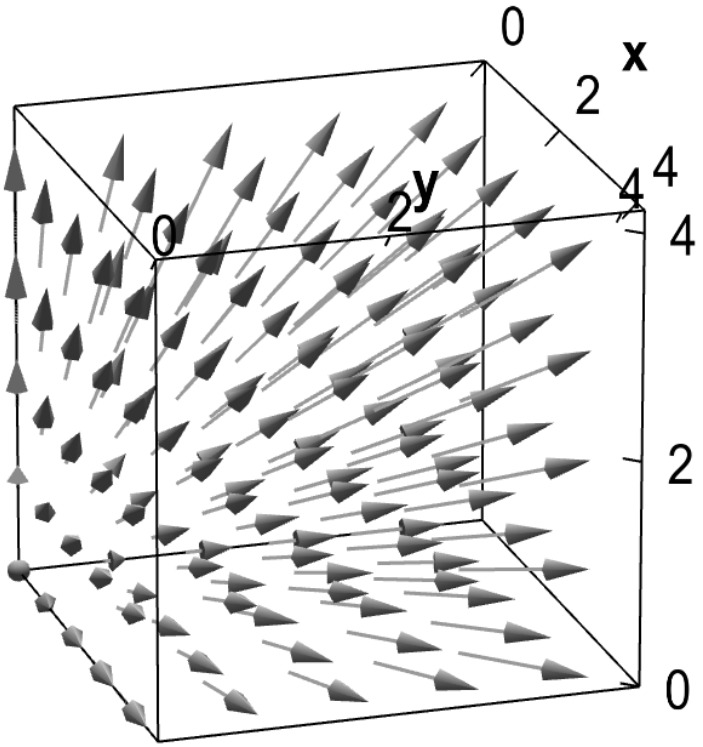


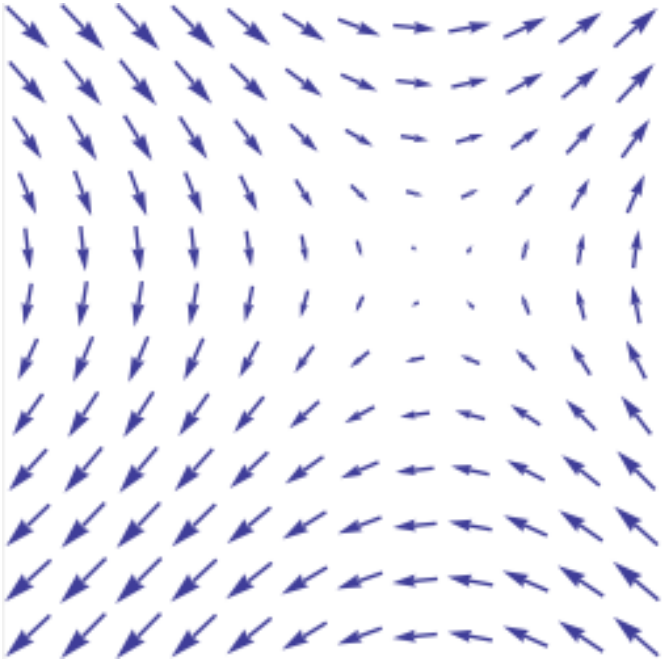




Flow Visualization

Also called vector field visualization

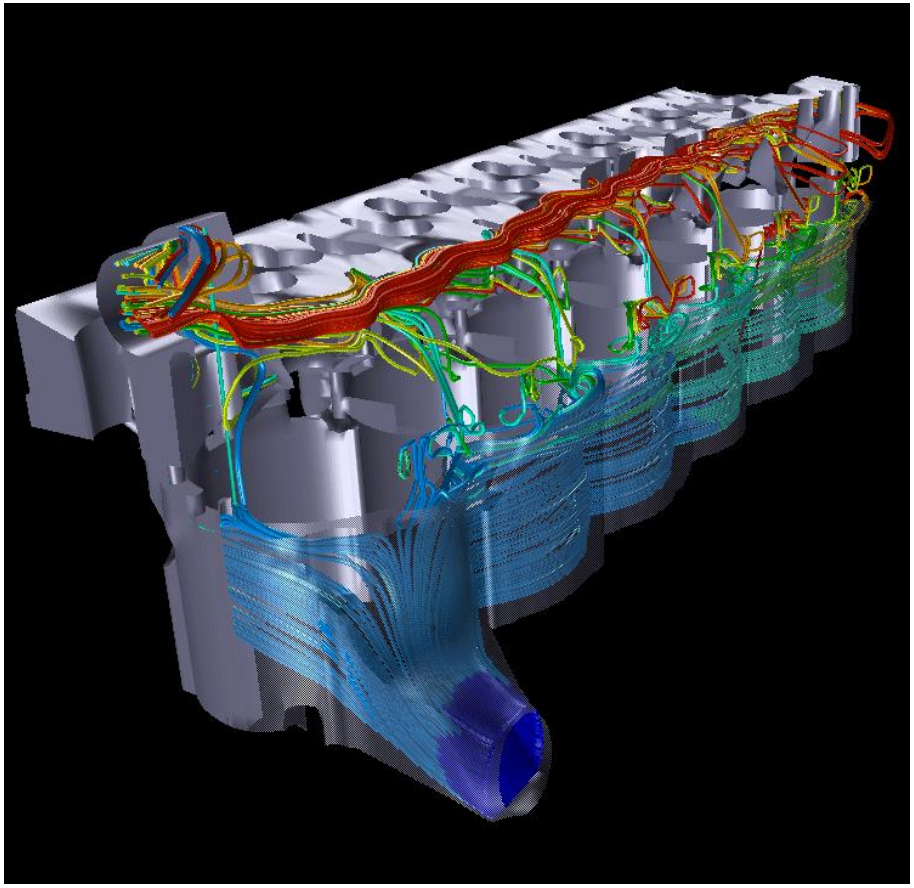




Stream Lines

Perform an integration through the vector field

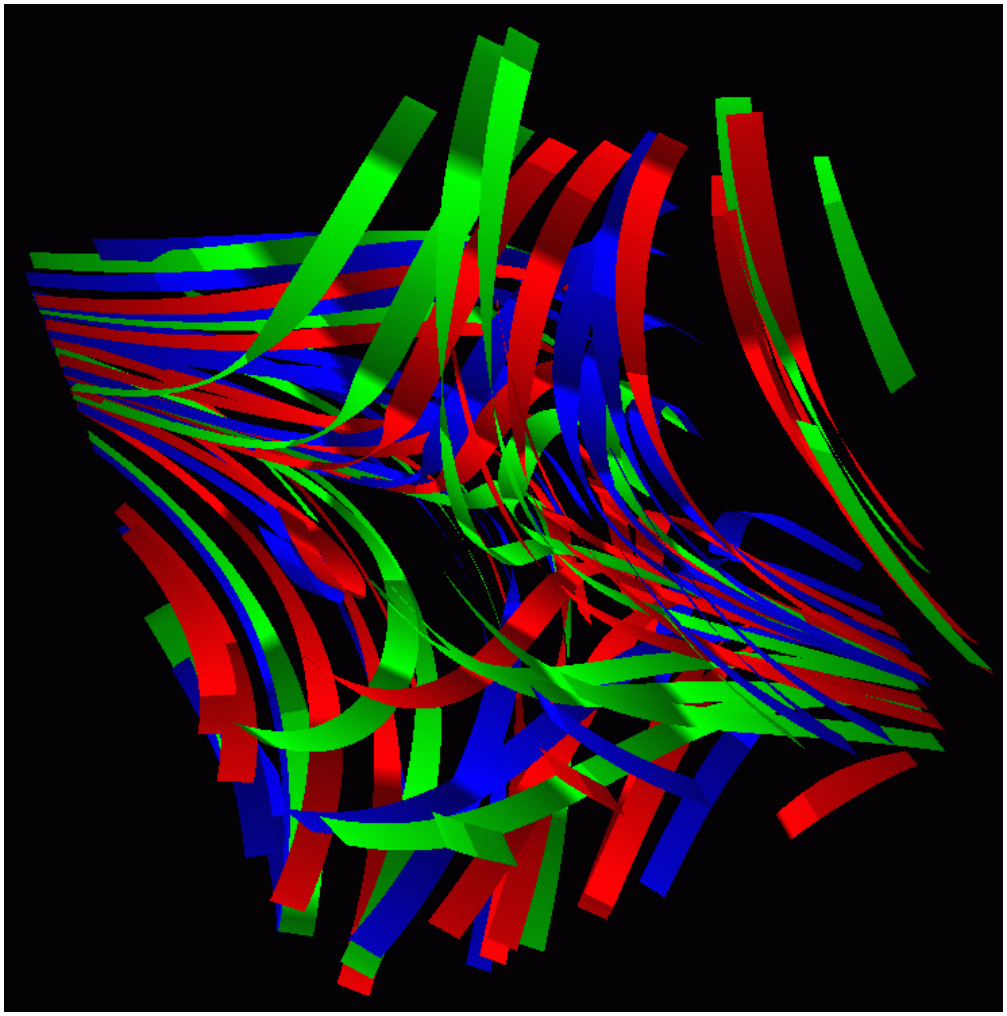
- color maps to temperature



Stream Ribbons

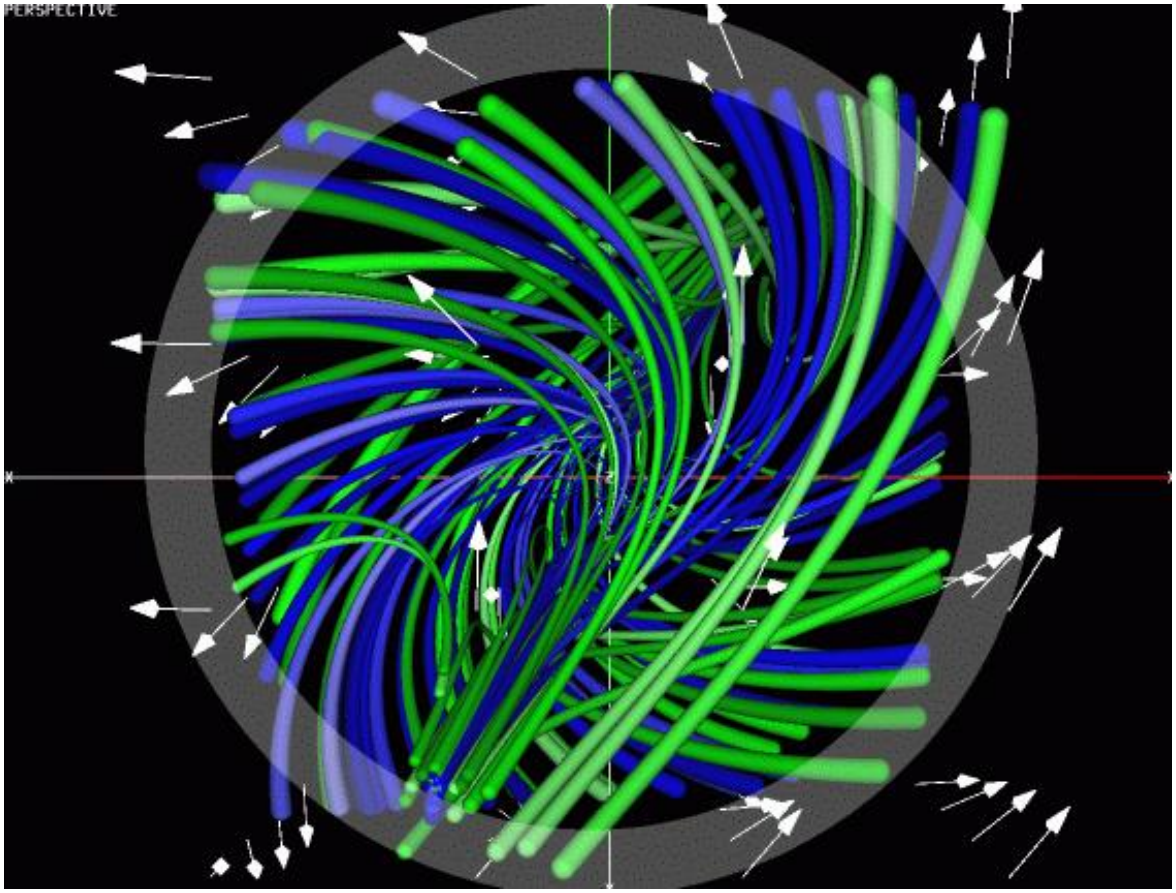
Connect two streamlines

- the center streamline gives direction, the other two indicate the twisting



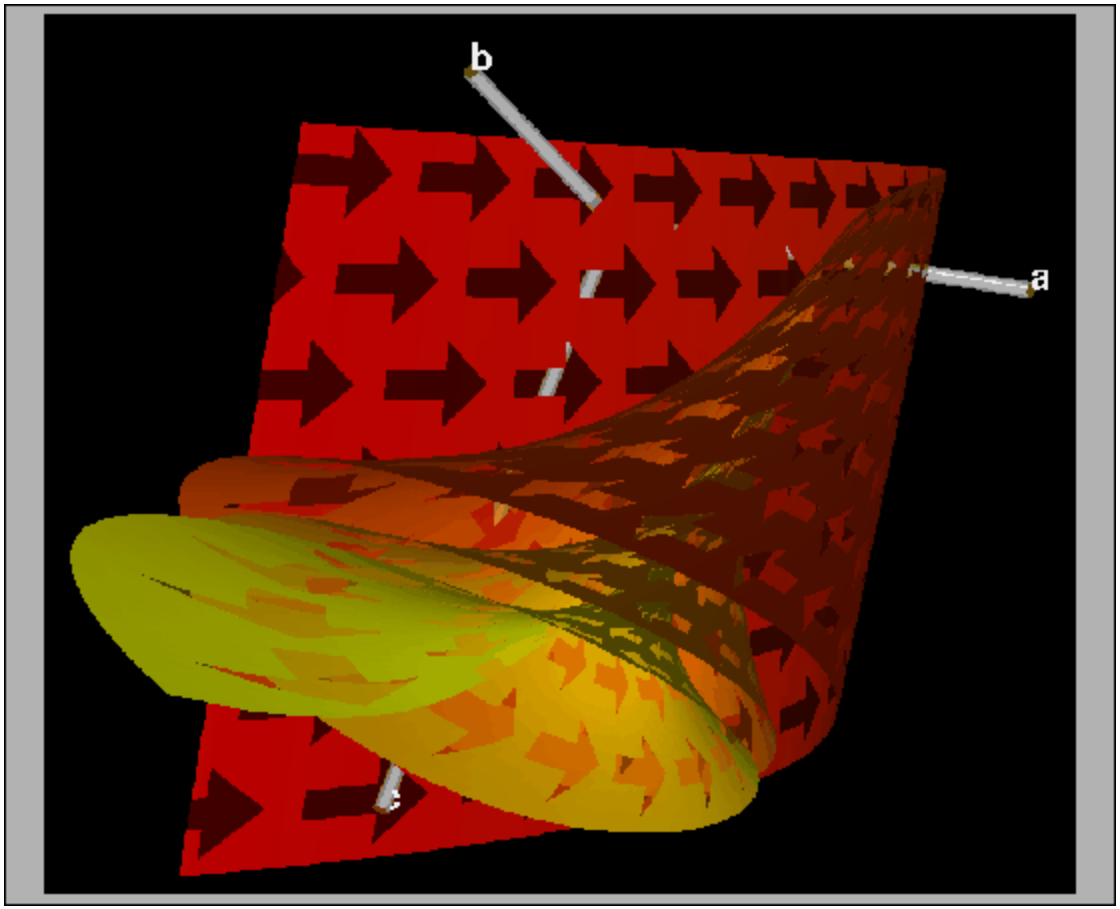
Stream Tubes

Connect three or more streamlines



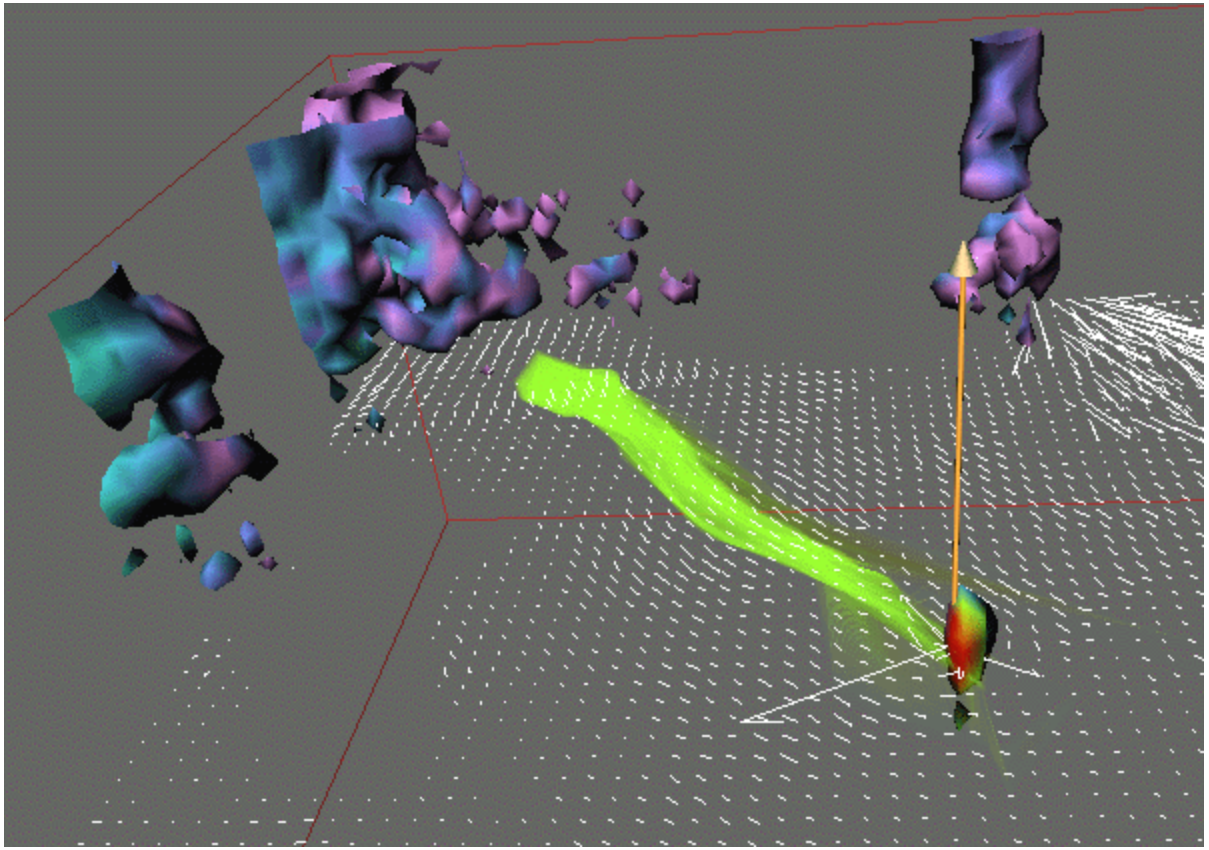
Stream Surfaces

Sweep a line segment through the vector field



Stream Balls

Smoke is injected into the flow field and compresses/expands due to the vector field

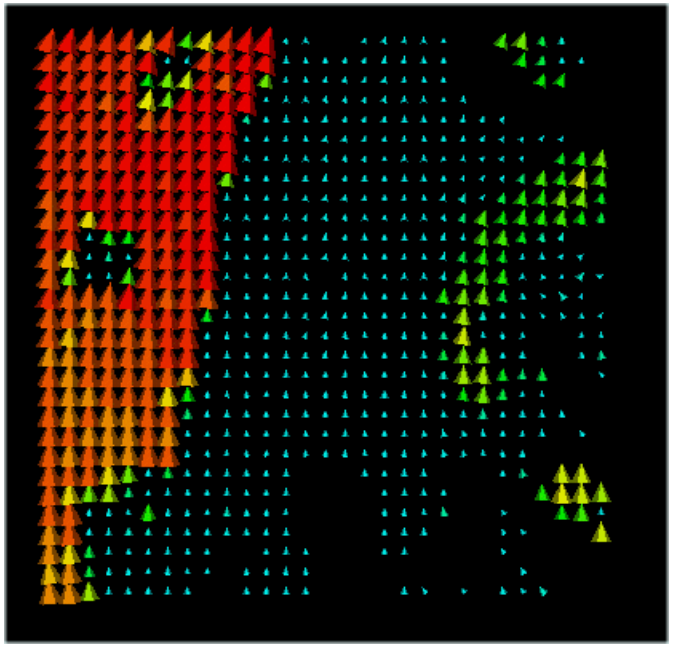


Global Techniques

Seek to give a more global view of the vector field

Hedgehogs

- oriented lines spread over the volume, indicating the orientation and magnitude of the flow
- do not show directional information

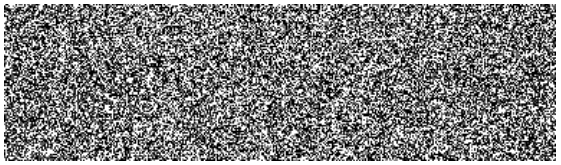


Glyphs, arrows

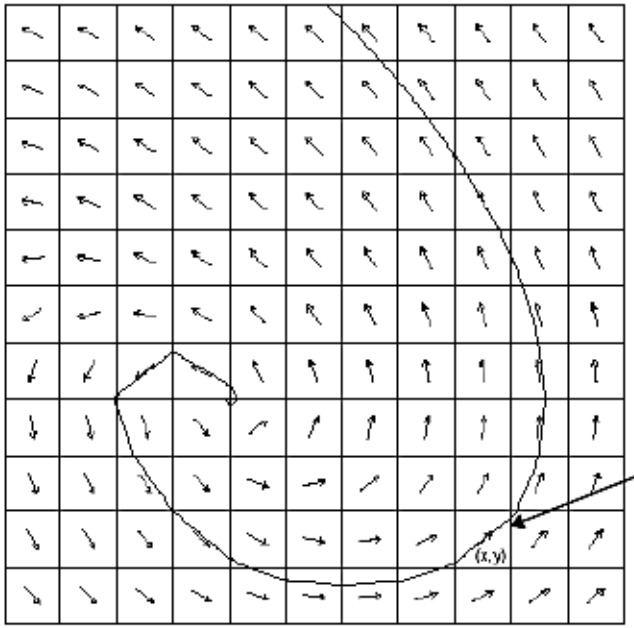
- icons that show directions, but tend to clutter the display

Line Integral Convolution (LIC)

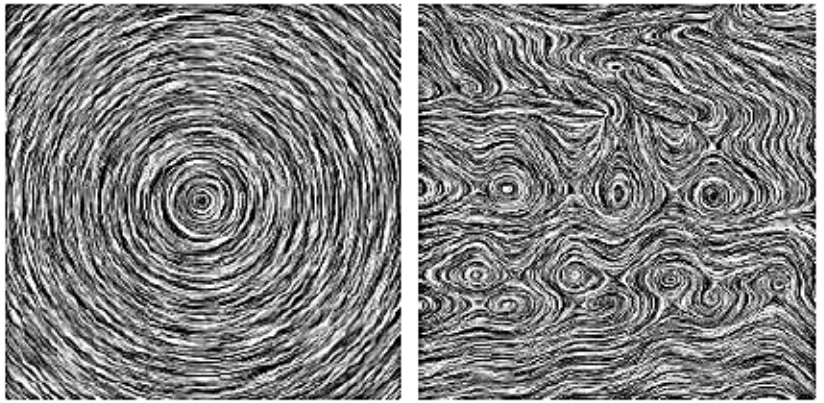
- Input:
  - a 2D vector field
  - an image that will be "smeared" according to the stream lines described by the vector field



salt+pepper noise



input vector field



output image = line-integrated white noise image

stream line

For each ouput pixel (x, y)

Follow the stream line forward for some distance Δs

Multiply each pixel value by a 1D filter kernel and add

Follow the stream line backward for some distance Δs

Multiply each pixel value by a 1D filter kernel and add

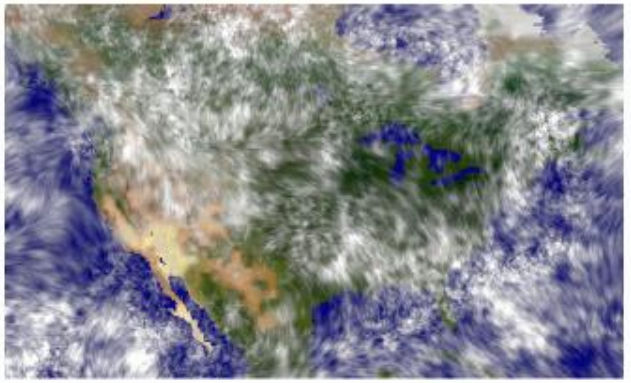
Follow the stream line backward for some distance Ds

filter aligned with the stream line

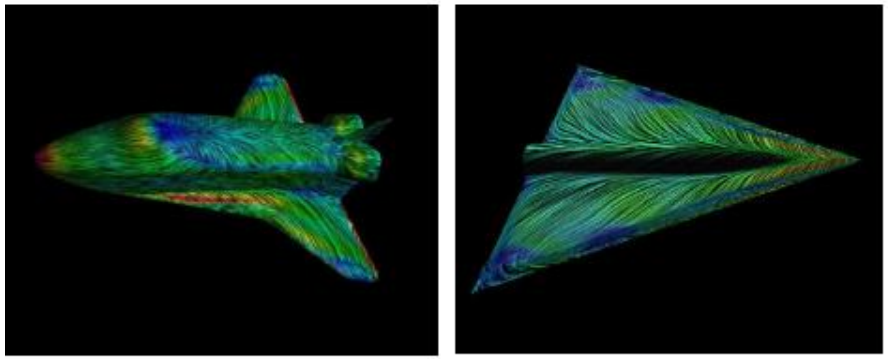
Line Integral Convolution (LIC)

a flower image with different vector fields

a simple motion vector field over the hand



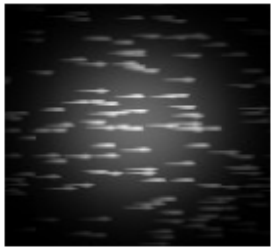
using vector magnitude to determine Δs



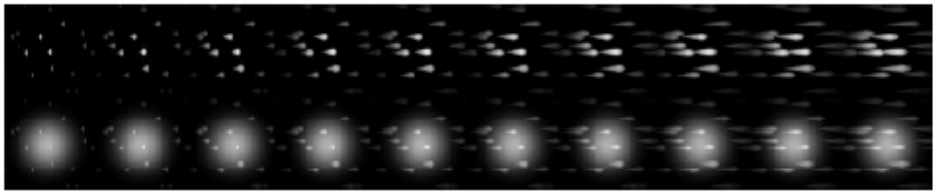
mapping LIC onto an object surface

Textured Splats

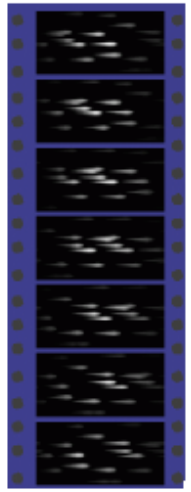
- Embed flow field vector icons into a splat
  - this enables smooth blending of neighboring icons



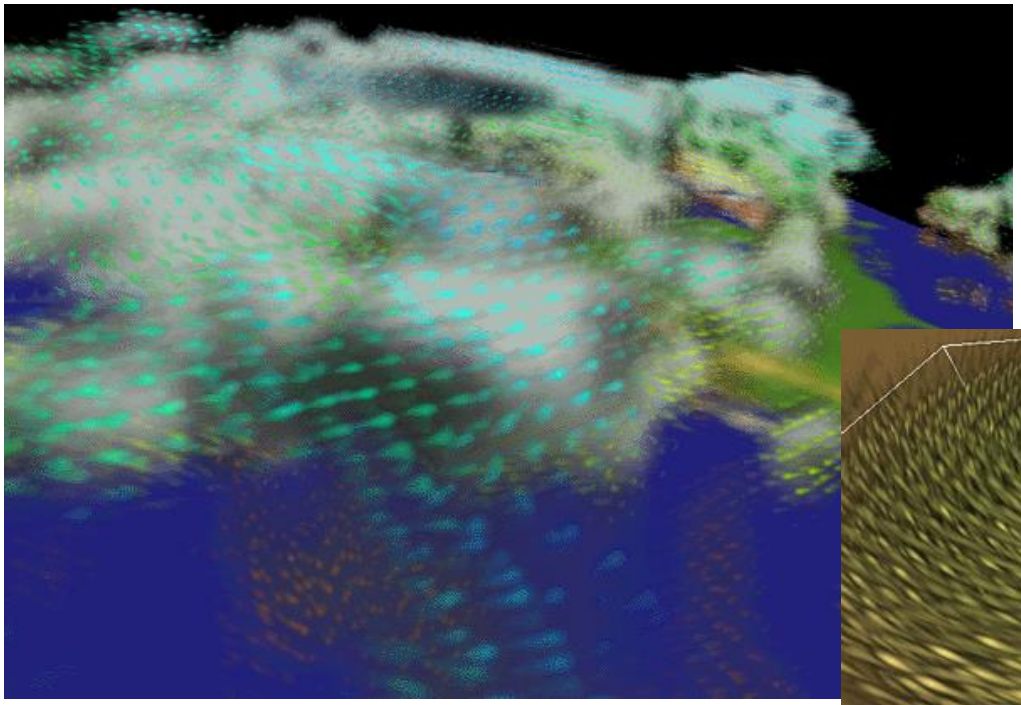
- Create a table of texture splats with varying icon distribution (to prevent regular patters)
- For a given location, select a random splat and rotate corresponding to the flow field direction
- Since the flow field is 3D, the component of the vectors that is parallel to the screen varies
- Need to provide splats that accommodate for vector foreshortening when the flow heads towards us

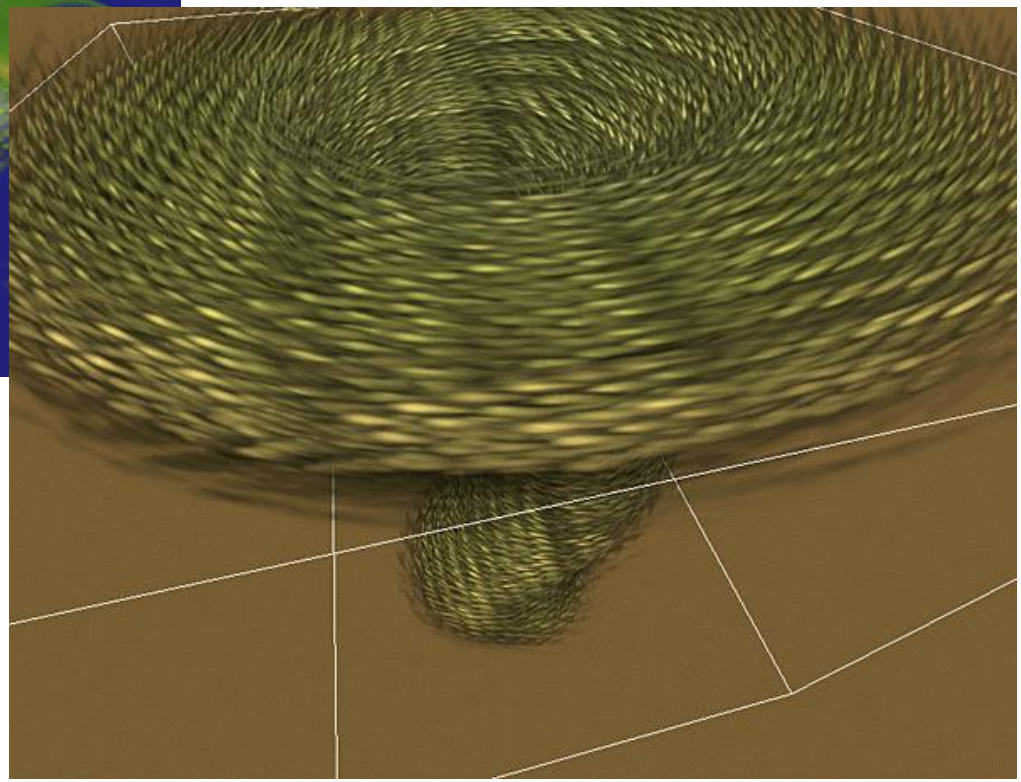


- Animated display
  - store a splat table with vector icons that are cyclically shifted from left to right
  - cycle through this table when picking splats to update the animated display



Textured Splats Examples





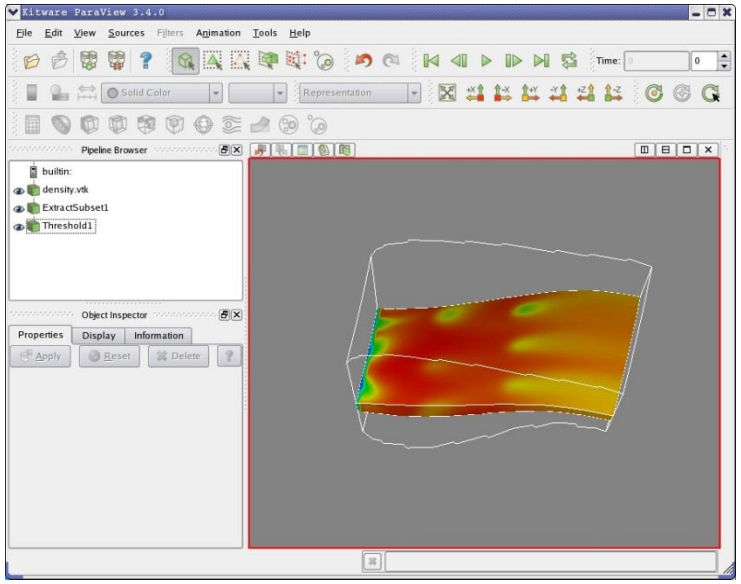
Popular Software & Libraries

VTK

- The Visualization Toolkit library
- developed by Kitware

Paraview

- built on top of VTK
- open-source
- multi-platform
- developed by
  Sandia & Los Alamos National Labs



VisIt

- open source
- developed by Lawrence Livermore National Lab