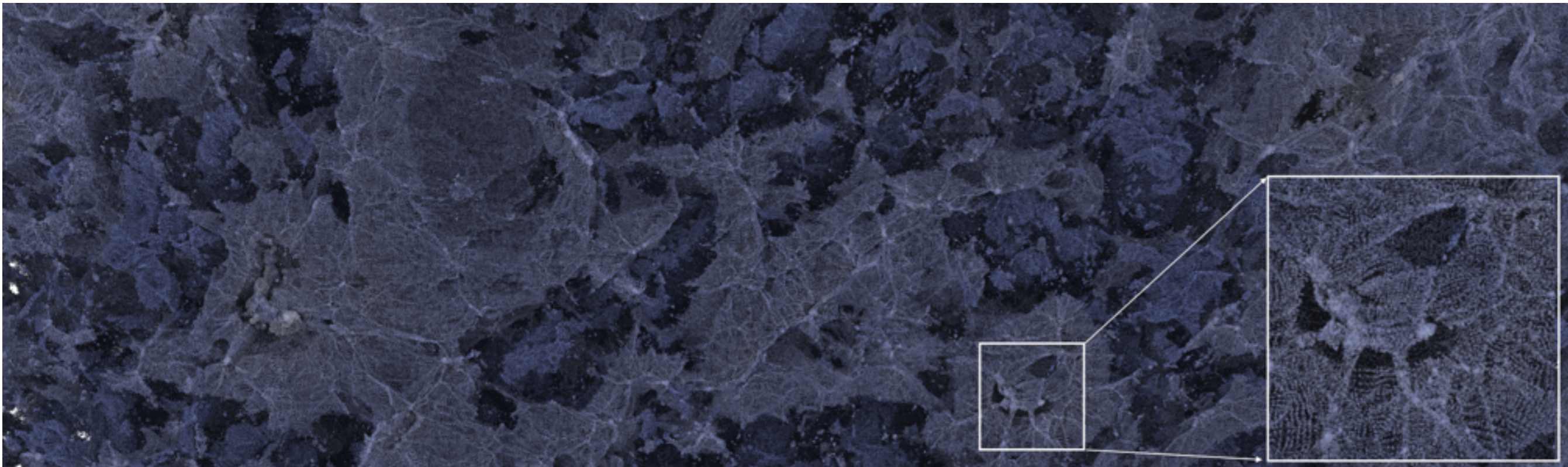


Supercomputing and Scientific Visualization



Aaron Knoll

Research Scientist, SCI Institute, University of Utah

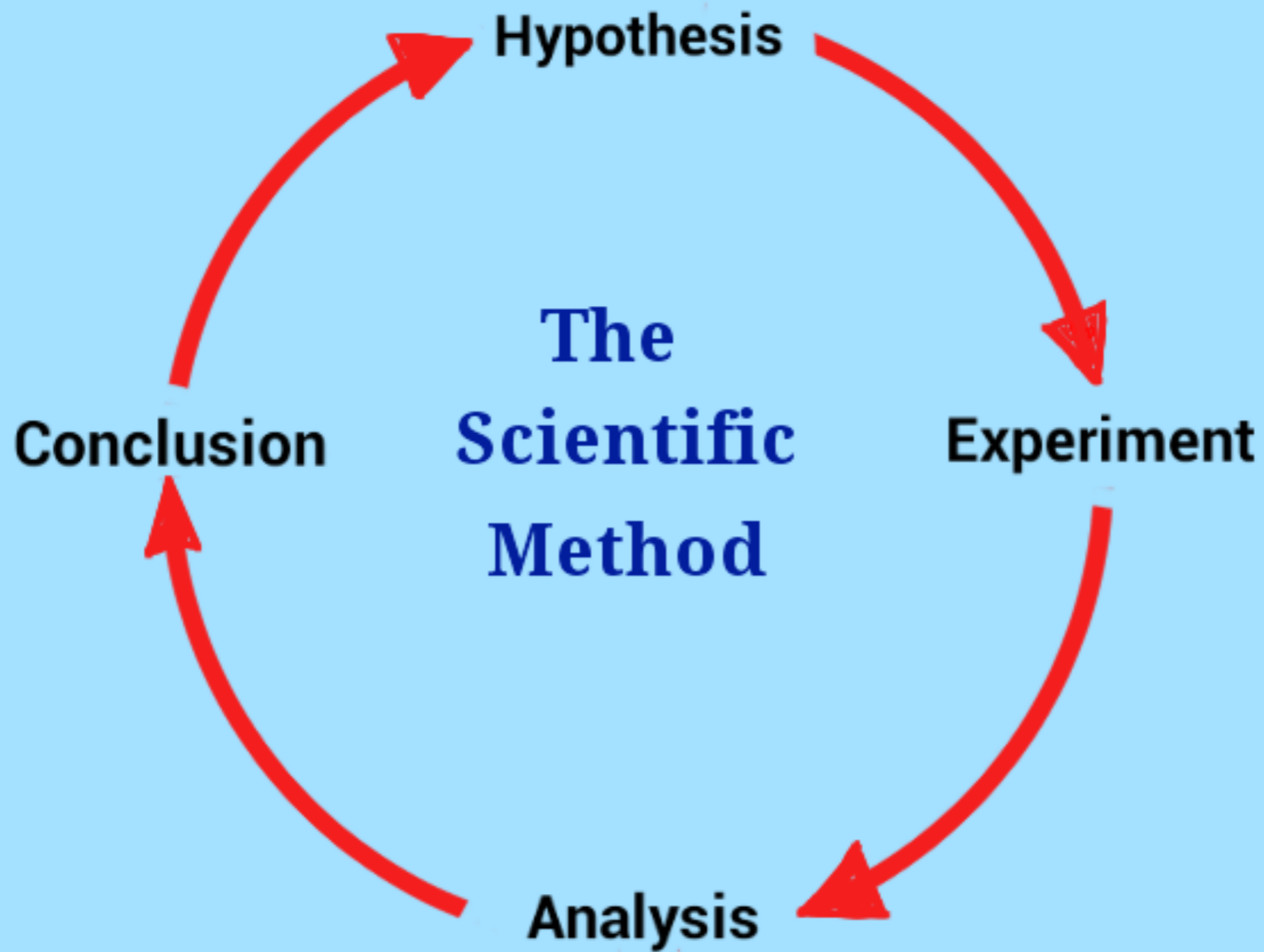
CS 1060, Explorations in Computer Science

4-21-16

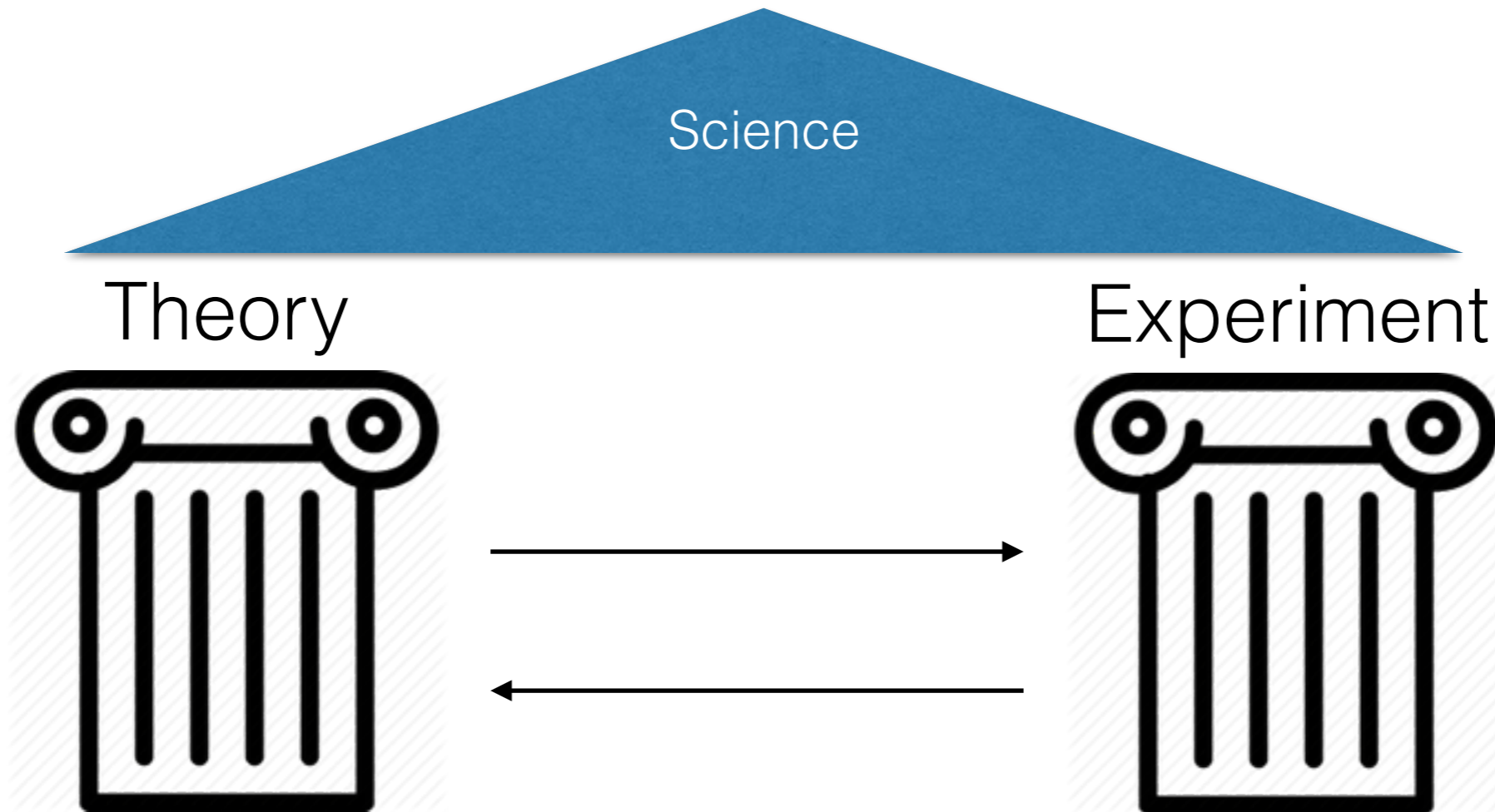
Roadmap

- Computing: the third pillar of the scientific method
- Visualization
- Supercomputing (and large-scale visualization)
- A fun example: volume rendering!

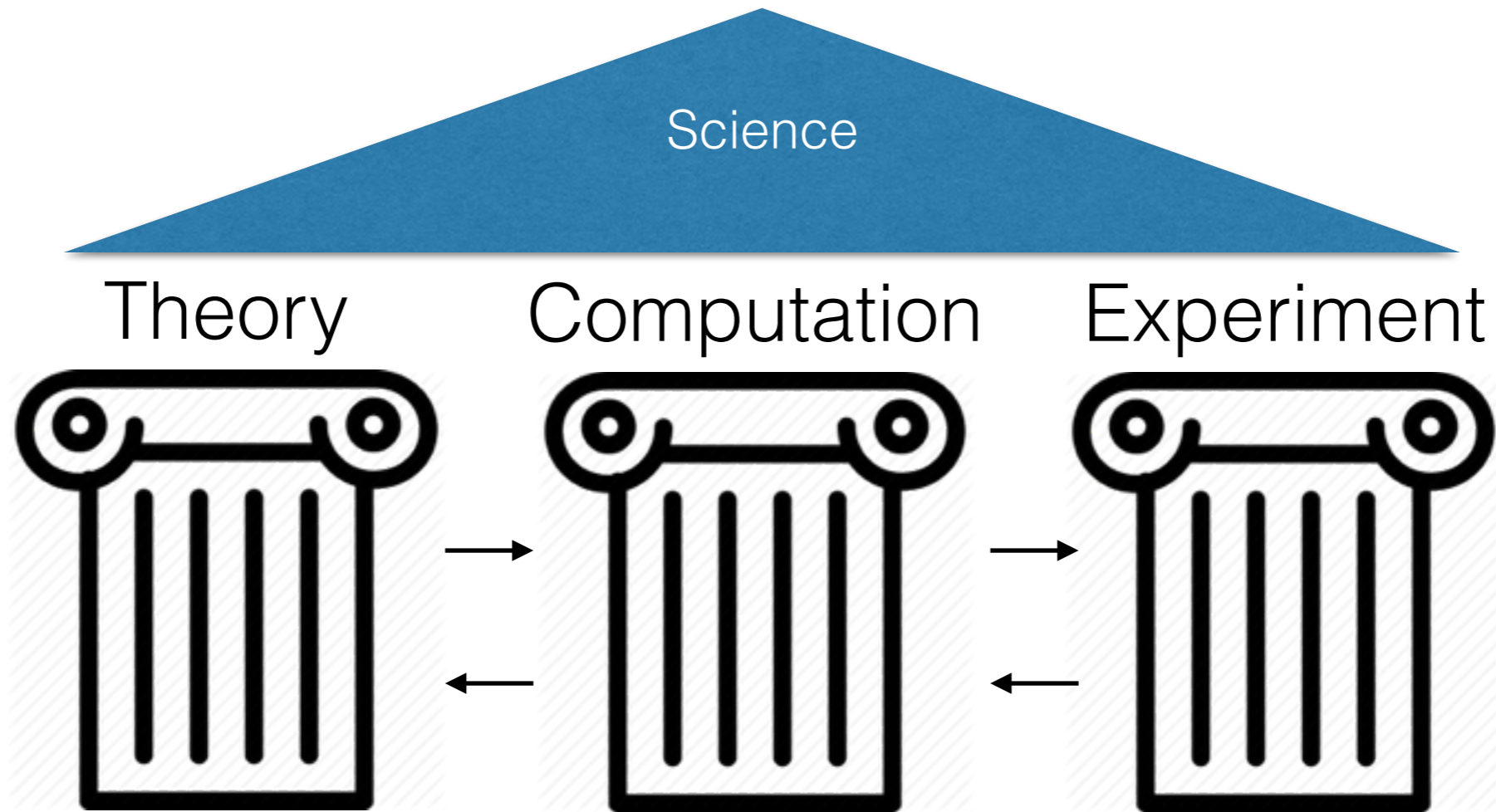
“The purpose of computing is insight not numbers.”
-- R. W. Hamming (1961)



Pillars of the scientific method



Pillars of the scientific method



Question:
How do we test the efficiency of an
airplane body?

Experiment

Milestones in Flight History

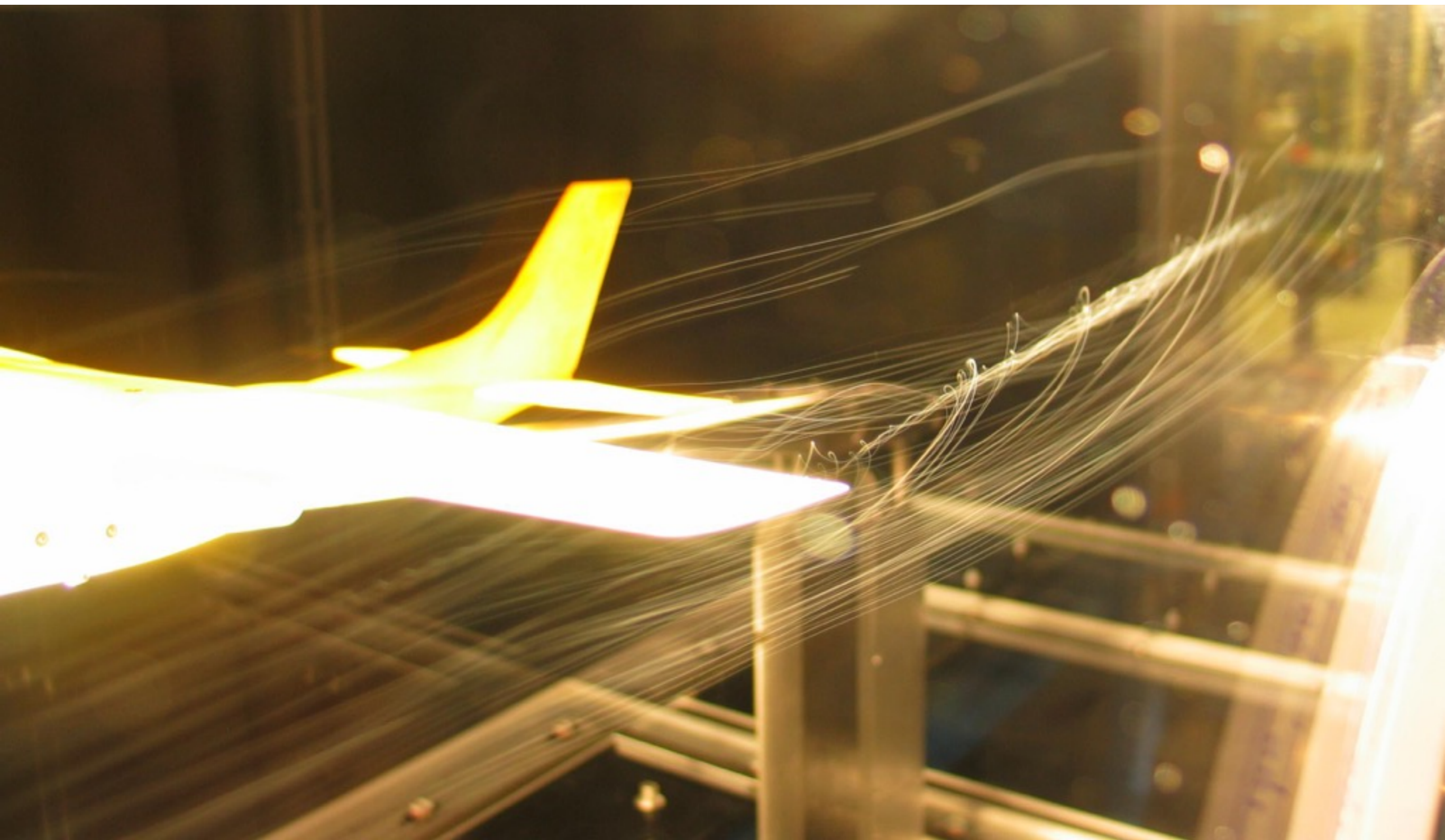
Dryden Flight Research Center



L-1011

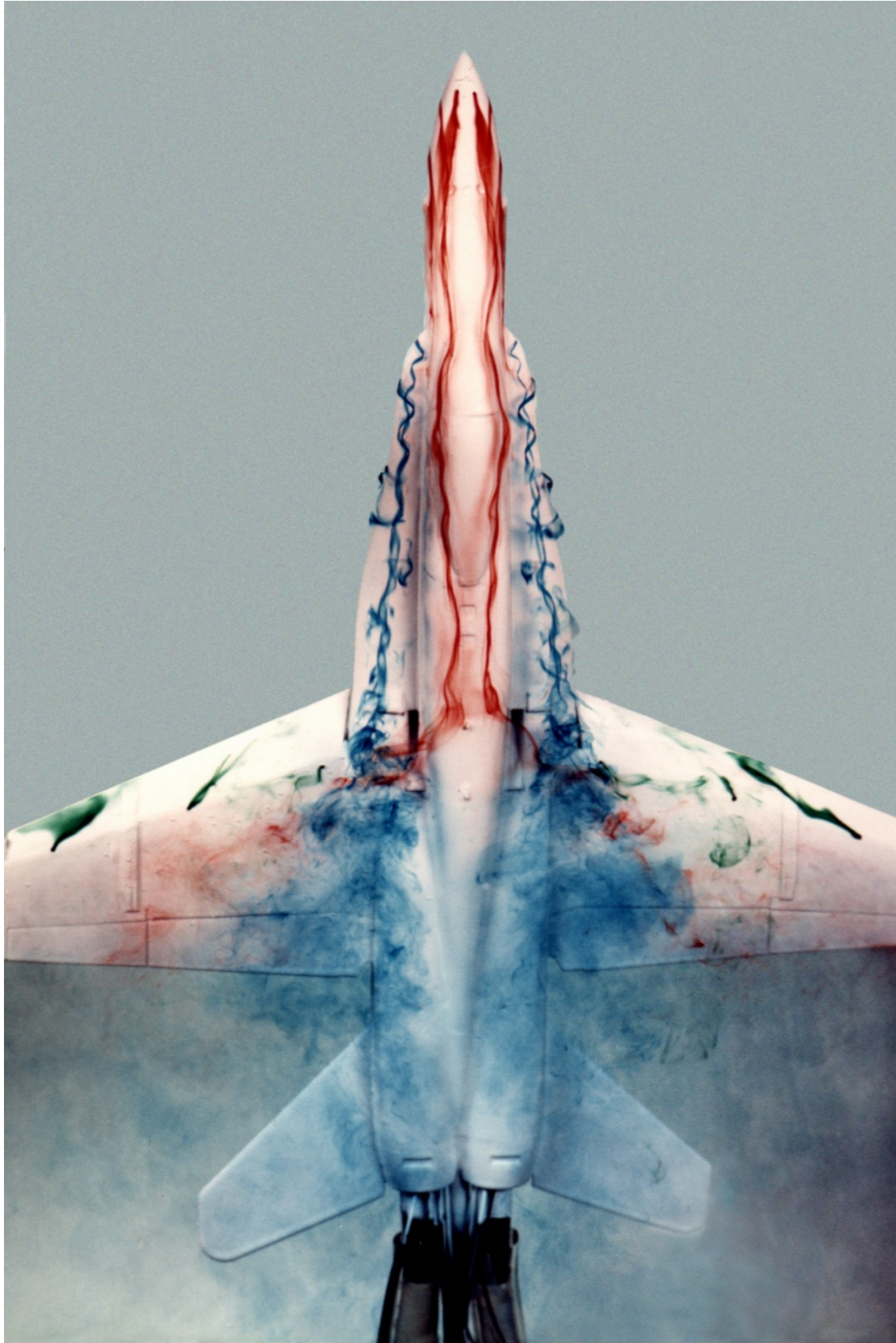
Airliner Wing Vortice Tests at Langley

Circa 1970s



A wind tunnel model of a Cessna 182 showing a wingtip vortex.
Tested in the RPI (Rensselaer Polytechnic Institute) Subsonic Wind Tunnel.
By Ben FrantzDale (2007).

Streaklines in Experimental Flow Vis



NASA Dryden Flight Research Center Photo Collection
<http://www.dfrc.nasa.gov/gallery/photo/index.html>
NASA Photo: ECN-33298-03 Date: 1985

1/48-scale model of an F-18 aircraft in Flow Visualization Facility (FVF)



Dryden Flight Research Center ECN 33298-47 Photographed 1985
F-18 water tunnel test in Flow Visualization Facility NASA/Dryden



Theory

Pijush K. Kundu **Ira M. Cohen** **David R. Dowling**

with contributions by P.S. Ayyaswamy and H.H. Hu



Fluid Mechanics

Fifth Edition



http://www3.nd.edu/~fthomas/Kundu_Fluid_Mechanics.pdf

Fluid Mechanics in One Slide



- **Navier-Stokes equations:** a set of PDE's modeling the behavior of fluids.
Example for compressible fluids:

$$\underbrace{\rho \left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right)}_1 = \underbrace{-\nabla p}_2 + \underbrace{\nabla \cdot (\mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)) - \frac{2}{3}\mu(\nabla \cdot \mathbf{u})\mathbf{I}}_3 + \underbrace{\mathbf{F}}_4$$

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0$$

Continuity equation

where \mathbf{u} is the fluid velocity, p is the fluid pressure, ρ is the fluid density, and μ is the viscosity.

- **Conservation of mass, momentum, energy** (relate to 2nd law of thermodynamics).
- **Viscosity** is the measure of the fluid's resistance to deformation, from shear or tensile stress. (A stress tensor with 9 degrees of freedom!)
- Flow can be **steady** (time derivative $\frac{\partial \rho}{\partial t} = 0$) or **unsteady** (or **transient**, i.e. high time derivative)
- Also **laminar** (flows in predictable, parallel layers) or **turbulent** (eddies, vortices, random chaos).
- **Reynolds number** indicates the turbulence of flow = inertial forces / viscous forces.

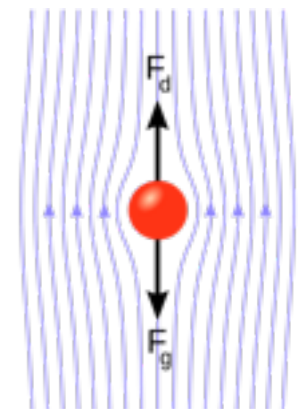
$$\text{Re} = \frac{\text{inertial forces}}{\text{viscous forces}} = \frac{\rho \mathbf{v} L}{\mu} = \frac{\mathbf{v} L}{\nu}$$

<https://www.comsol.com/multiphysics/navier-stokes-equations>

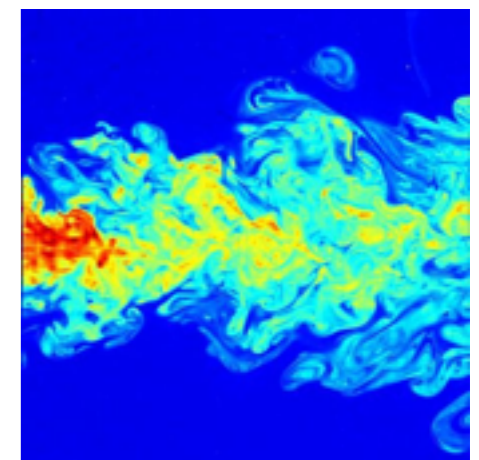
[https://en.wikipedia.org/wiki/Navier–Stokes equations](https://en.wikipedia.org/wiki/Navier–Stokes_equations)

https://en.wikipedia.org/wiki/Fluid_dynamics

https://en.wikipedia.org/wiki/Chaos_theory



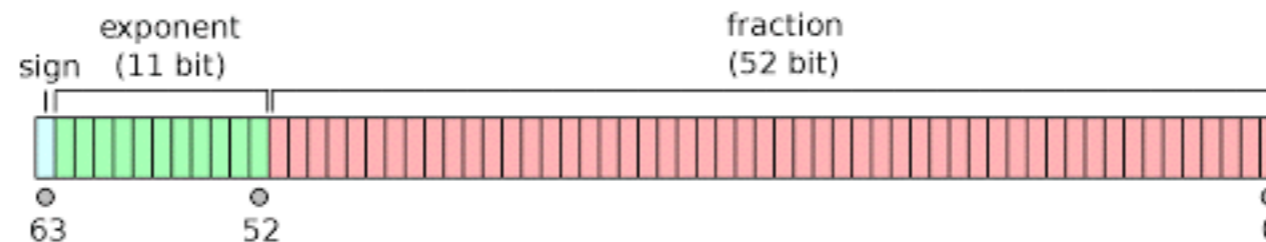
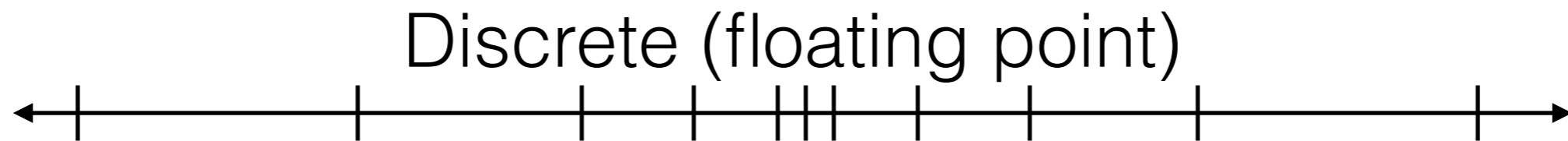
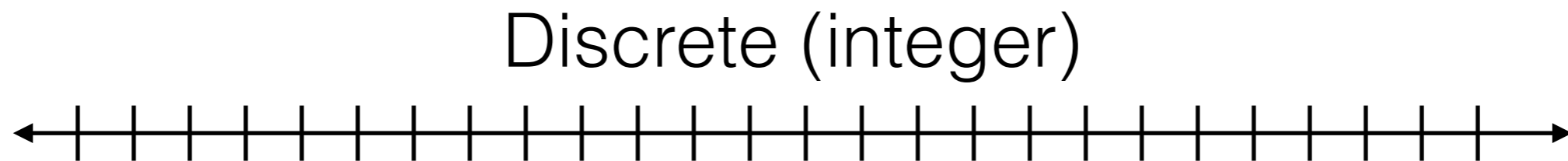
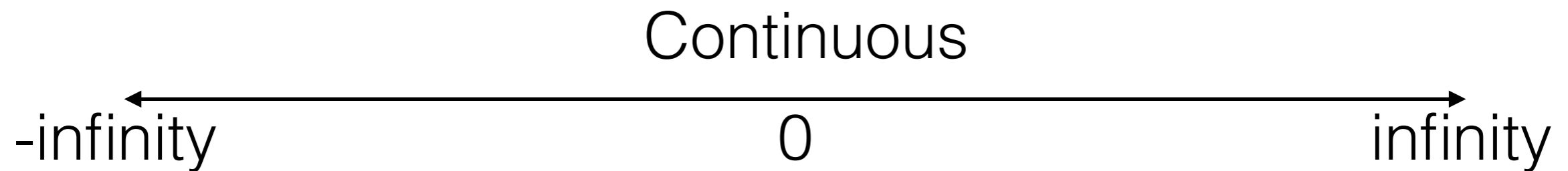
Laminar flow



Turbulent flow

Computation

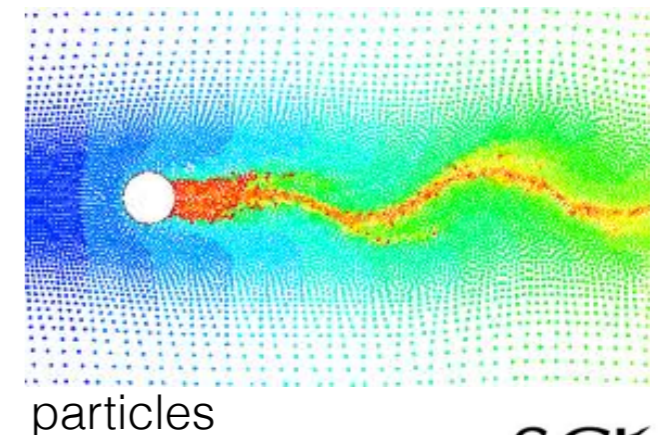
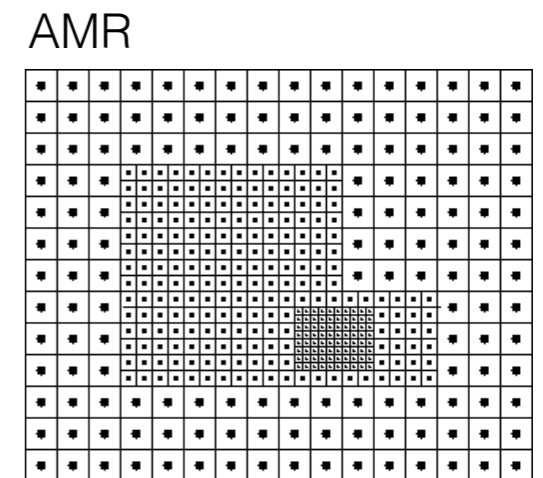
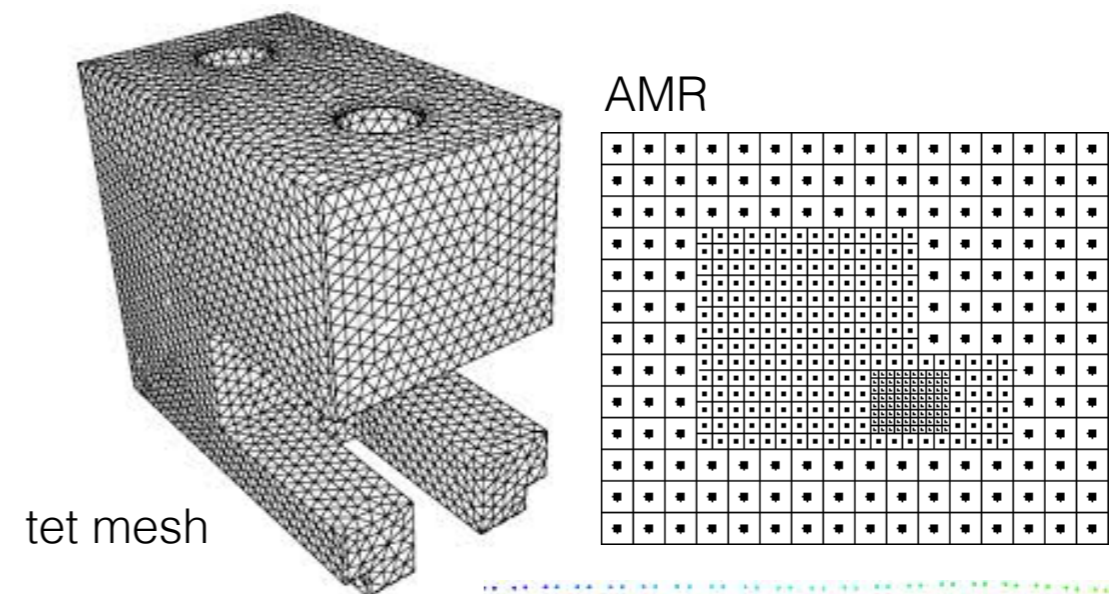
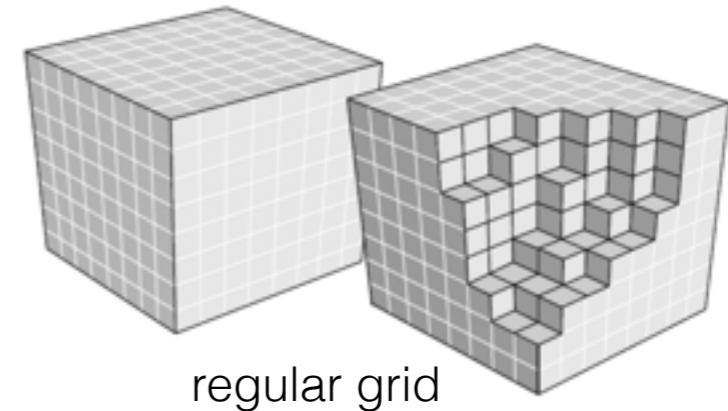
Representing numbers



https://en.wikipedia.org/wiki/IEEE_floating_point

From calculus to grids

- Lagrangian vs Eulerian
- Finite element method (FEM)
- Finite difference method (FDM)
- Finite volume method (FVM)
- Direct numerical simulation (DNS)
- Adaptive Mesh Refinement (AMR)
- Highly Recommended:
CS3200 : Introduction to Scientific Computing
<http://www.eng.utah.edu/~cs3200/>

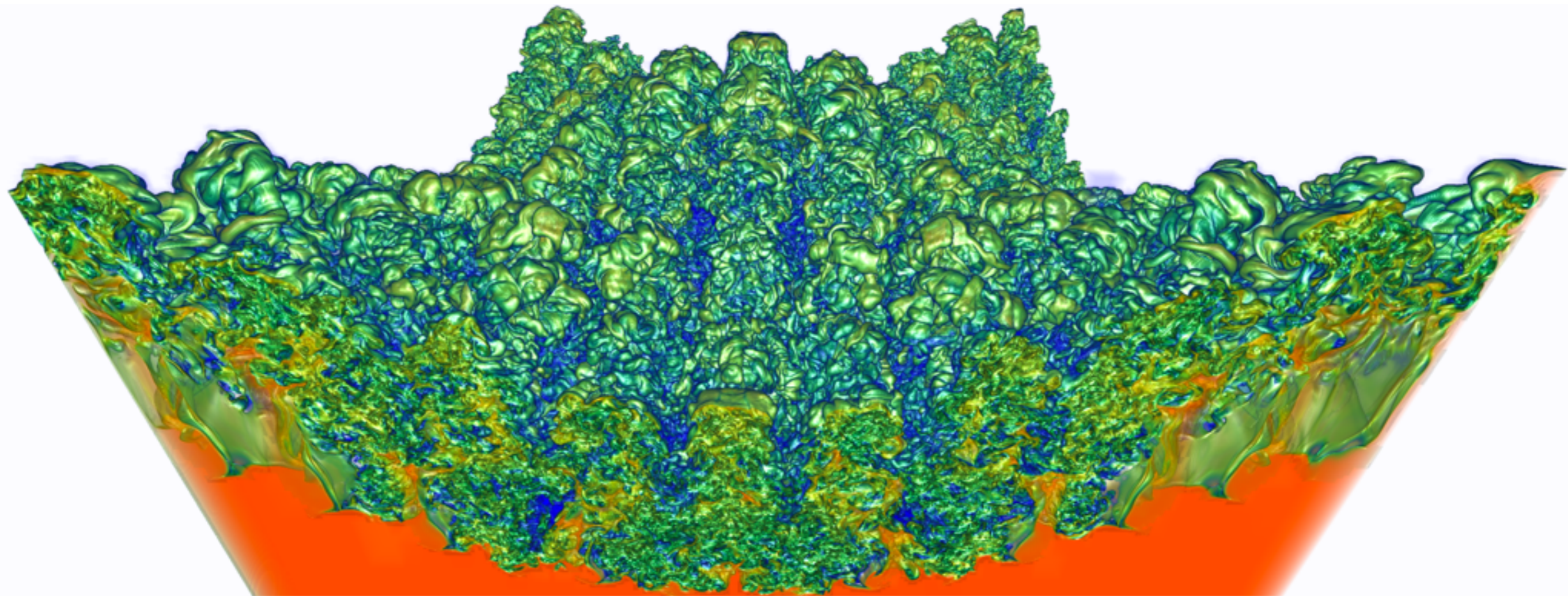


Solving a big matrix

$$\begin{bmatrix}
 N & \Sigma w & \Sigma x & \Sigma y & \Sigma x^2 & \Sigma wy & \Sigma xy & \Sigma wx & \Sigma w^2 & \Sigma y^2 & \Sigma x^3 & \Sigma w^3 & \Sigma y^3 \\
 \Sigma w & \Sigma w^2 & \Sigma wx & \Sigma wy & \Sigma wx^2 & \Sigma w^2 y & \Sigma wxy & \Sigma w^2 x & \Sigma w^3 & \Sigma wy^2 & \Sigma wx^3 & \Sigma w^4 & \Sigma wy^3 \\
 \Sigma x & \Sigma wx & \Sigma x^2 & \Sigma xy & \Sigma x^3 & \Sigma wxy & \Sigma x^2 y & \Sigma wx^2 & \Sigma w^2 x & \Sigma xy^2 & \Sigma x^4 & \Sigma xw^3 & \Sigma xy^3 \\
 \Sigma y & \Sigma wy & \Sigma xy & \Sigma y^2 & \Sigma x^2 y & \Sigma wy^2 & \Sigma xy^2 & \Sigma wxy & \Sigma w^2 y & \Sigma y^3 & \Sigma x^3 y & \Sigma w^3 y & \Sigma y^4 \\
 \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
 \Sigma x^2 & \Sigma wx^2 & \Sigma x^3 & \Sigma x^2 y & \Sigma x^4 & \Sigma wx^2 y & \Sigma x^3 y & \Sigma wx^3 & \Sigma w^2 x^2 & \Sigma x^2 y^2 & \Sigma x^5 & \Sigma w^3 x^2 & \Sigma x^2 y^3 \\
 \Sigma wy & \Sigma w^2 y & \Sigma wxy & \Sigma wy^2 & \Sigma wx^2 y & \Sigma w^2 y^2 & \Sigma wxy^2 & \Sigma w^2 xy & \Sigma w^3 y & \Sigma wy^3 & \Sigma wyx^3 & \Sigma w^4 y & \Sigma wy^4 \\
 \Sigma xy & \Sigma wxy & \Sigma x^2 y & \Sigma xy^2 & \Sigma x^3 y & \Sigma wxy^2 & \Sigma x^2 y^2 & \Sigma wx^2 y & \Sigma w^2 xy & \Sigma xy^3 & \Sigma x^4 y & \Sigma w^3 xy & \Sigma xy^4 \\
 \Sigma wx & \Sigma w^2 x & \Sigma wx^2 & \Sigma wxy & \Sigma wx^3 & \Sigma w^2 xy & \Sigma wx^2 y & \Sigma w^2 x^2 & \Sigma w^3 x & \Sigma wxy^2 & \Sigma wx^4 & \Sigma w^4 x & \Sigma wxy^3 \\
 \Sigma w^2 & \Sigma w^3 & \Sigma w^2 x & \Sigma w^2 y & \Sigma w^2 x^2 & \Sigma w^3 y & \Sigma w^2 xy & \Sigma w^3 x & \Sigma w^4 & \Sigma w^2 y^2 & \Sigma w^2 x^3 & \Sigma w^5 & \Sigma w^2 y^3 \\
 \Sigma y^2 & \Sigma wy^2 & \Sigma xy^2 & \Sigma y^3 & \Sigma x^2 y^2 & \Sigma wy^3 & \Sigma xy^3 & \Sigma wxy^2 & \Sigma w^2 y^2 & \Sigma y^4 & \Sigma x^3 y^2 & \Sigma w^3 y^2 & \Sigma y^5 \\
 \Sigma x^3 & \Sigma wx^3 & \Sigma x^4 & \Sigma x^3 y & \Sigma x^5 & \Sigma wx^3 y & \Sigma x^4 y & \Sigma wx^4 & \Sigma w^2 x^3 & \Sigma x^3 y^2 & \Sigma x^6 & \Sigma w^3 x^3 & \Sigma x^3 y^3 \\
 \Sigma w^3 & \Sigma w^4 & \Sigma w^3 x & \Sigma w^3 y & \Sigma w^3 x^2 & \Sigma w^4 y & \Sigma w^3 xy & \Sigma w^4 x & \Sigma w^5 & \Sigma w^3 y^2 & \Sigma w^3 x^3 & \Sigma w^6 & \Sigma w^3 y^3 \\
 \Sigma y^3 & \Sigma wy^3 & \Sigma xy^3 & \Sigma y^4 & \Sigma x^2 y^3 & \Sigma wy^4 & \Sigma xy^4 & \Sigma wxy^3 & \Sigma w^2 y^3 & \Sigma y^5 & \Sigma x^3 y^3 & \Sigma w^3 y^3 & \Sigma y^6
 \end{bmatrix}
 \cdot
 \begin{bmatrix}
 A \\
 B \\
 C \\
 D \\
 E \\
 F \\
 G \\
 H \\
 I \\
 J \\
 K \\
 L \\
 M
 \end{bmatrix}
 =
 \begin{bmatrix}
 \Sigma z \\
 \Sigma wz \\
 \Sigma xz \\
 \Sigma yz \\
 \Sigma x^2 z \\
 \Sigma wyz \\
 \Sigma xyz \\
 \Sigma wxz \\
 \Sigma w^2 z \\
 \Sigma y^2 z \\
 \Sigma x^3 z \\
 \Sigma w^3 z \\
 \Sigma y^3 z
 \end{bmatrix}$$

Highly recommended: Math 2270: Linear Algebra

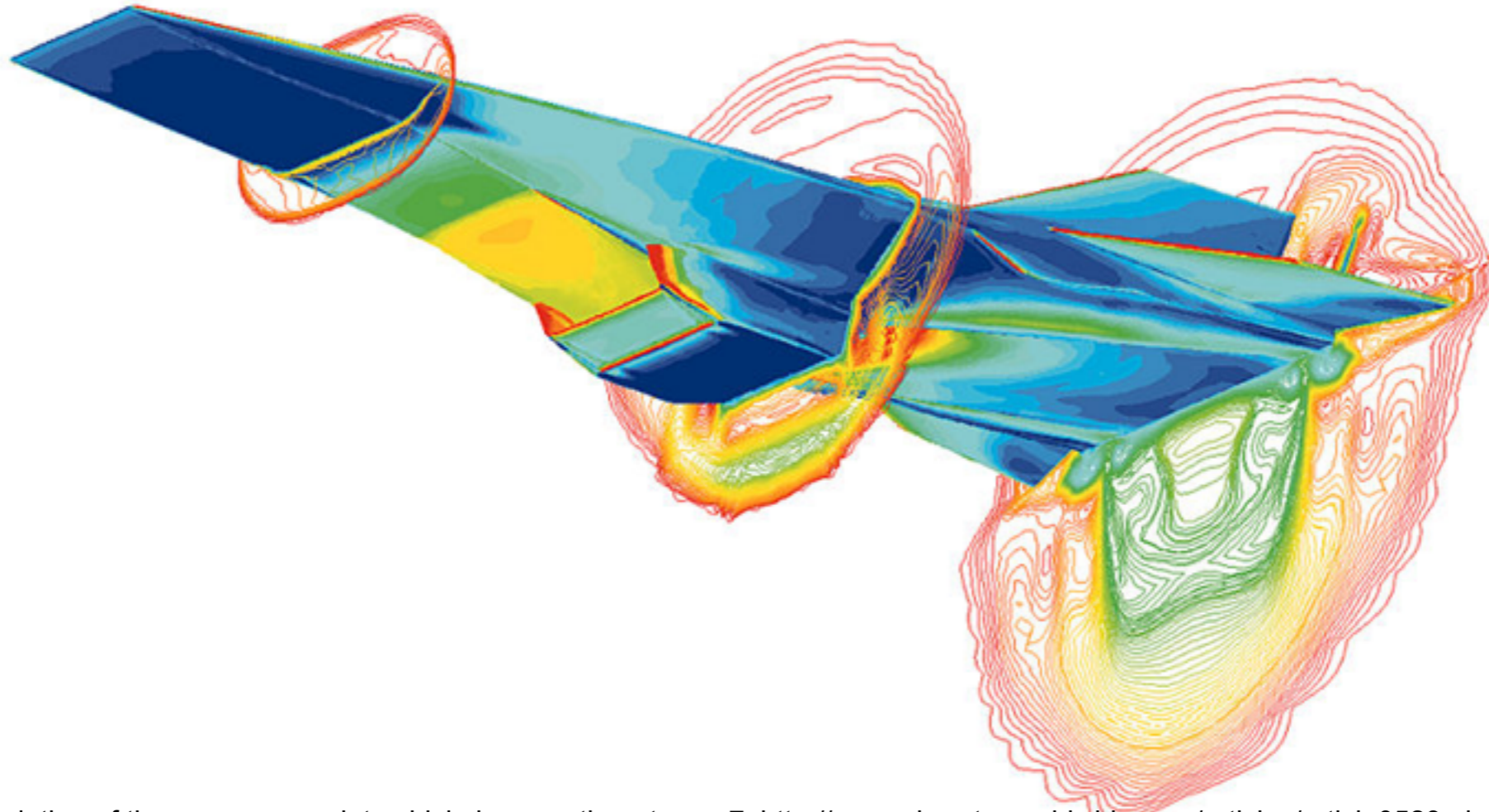
Richtmyer-Meshkov Instability



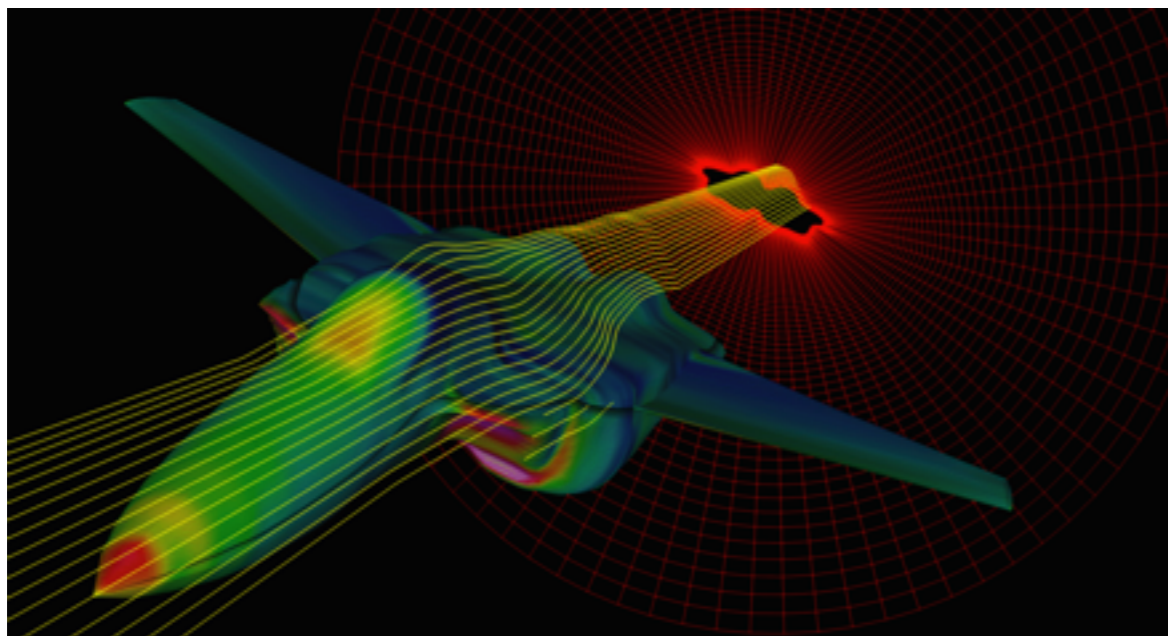
<http://swf.tubechop.com/tubechop.swf?vurl=2ZHPZP6njHU&start=64.27&end=109.43&cid=7907316> type="application/x-shockwave-flash" allowfullscreen="true" width="425" height="344"

2048x2048x1920 voxel Navier Stokes CFD simulation
Lawrence Livermore National Laboratory, circa 2003

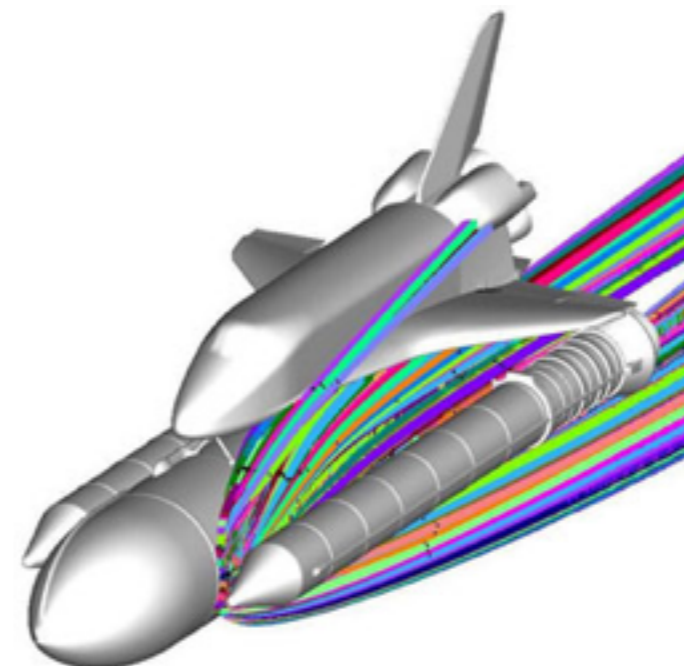
Computational Fluid Dynamics



A simulation of the [Hyper-X](http://www.airports-worldwide.com/articles/article0523.php) scramjet vehicle in operation at [Mach-7](http://www.airports-worldwide.com/articles/article0523.php). <http://www.airports-worldwide.com/articles/article0523.php>

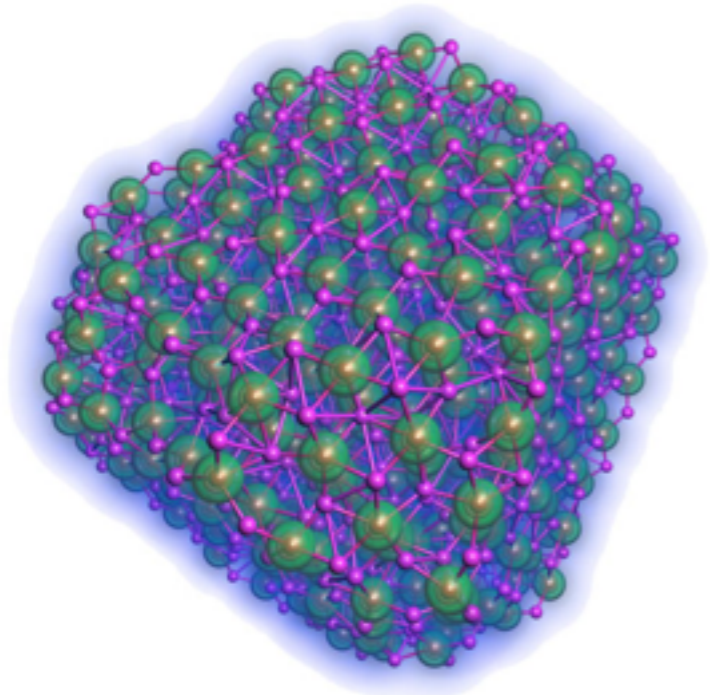


FAST, <http://www.openchannelfoundation.org>

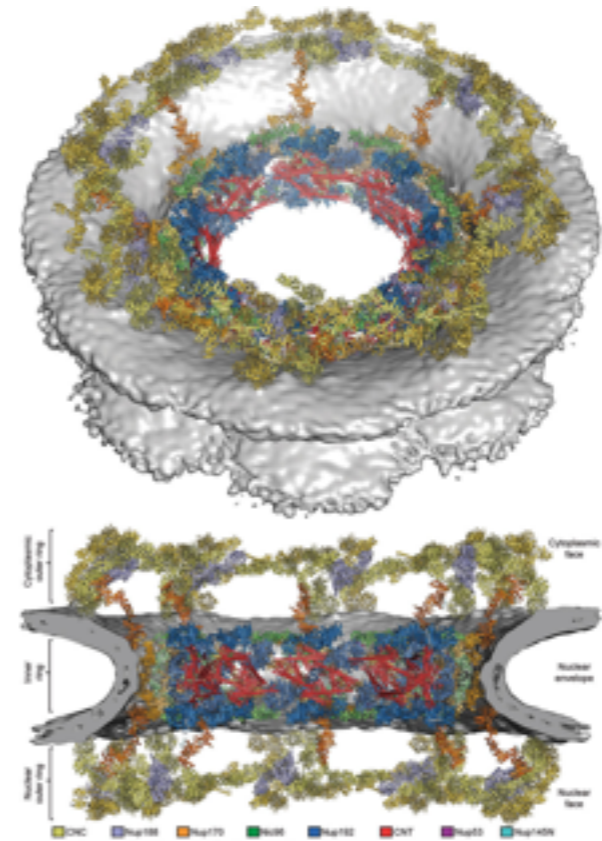


<http://www.cesc.zju.edu.cn/learningcenter.htm>

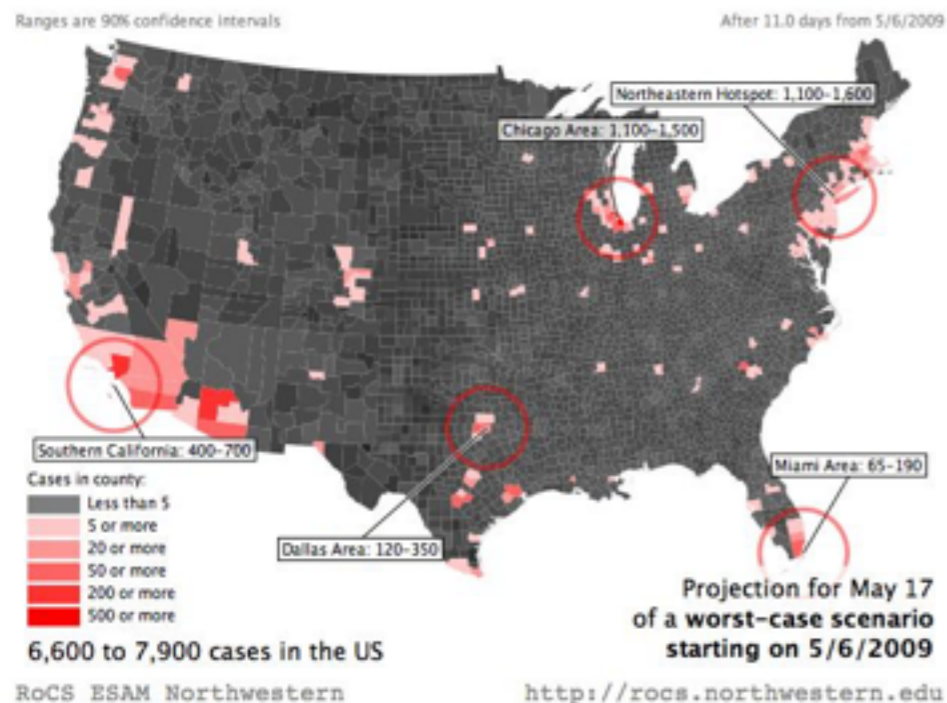
Other computing applications



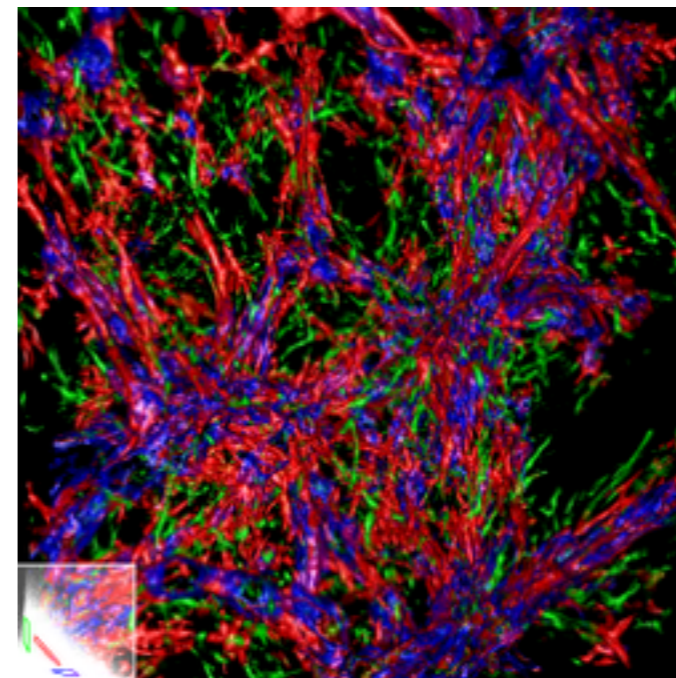
Materials Science



Biology



Epidemiology



Cosmology

Why is computing better?

Why is computing better?

- Pause, rewind, zoom in as much as you want
- Interrogate data anywhere
- Quantify *everything!*
- Simulate things you can't observe
- Change space scales: Angstroms to Megaparsecs
- Change time scales: femtoseconds to billions of years
- Validate, or test, a theory.



C-O - Pt7 Density Functional Theory computation
Julius Jellinek, Argonne National Laboratory

Why is experiment better?

Why is experiment better?

- It's *real*
- No code to write or debug!
- Bad theory? No problem!
- Real-time in situ visualization!
- Unlimited resolution

1 Planck length = 10^{-20} the length of 1 photon!

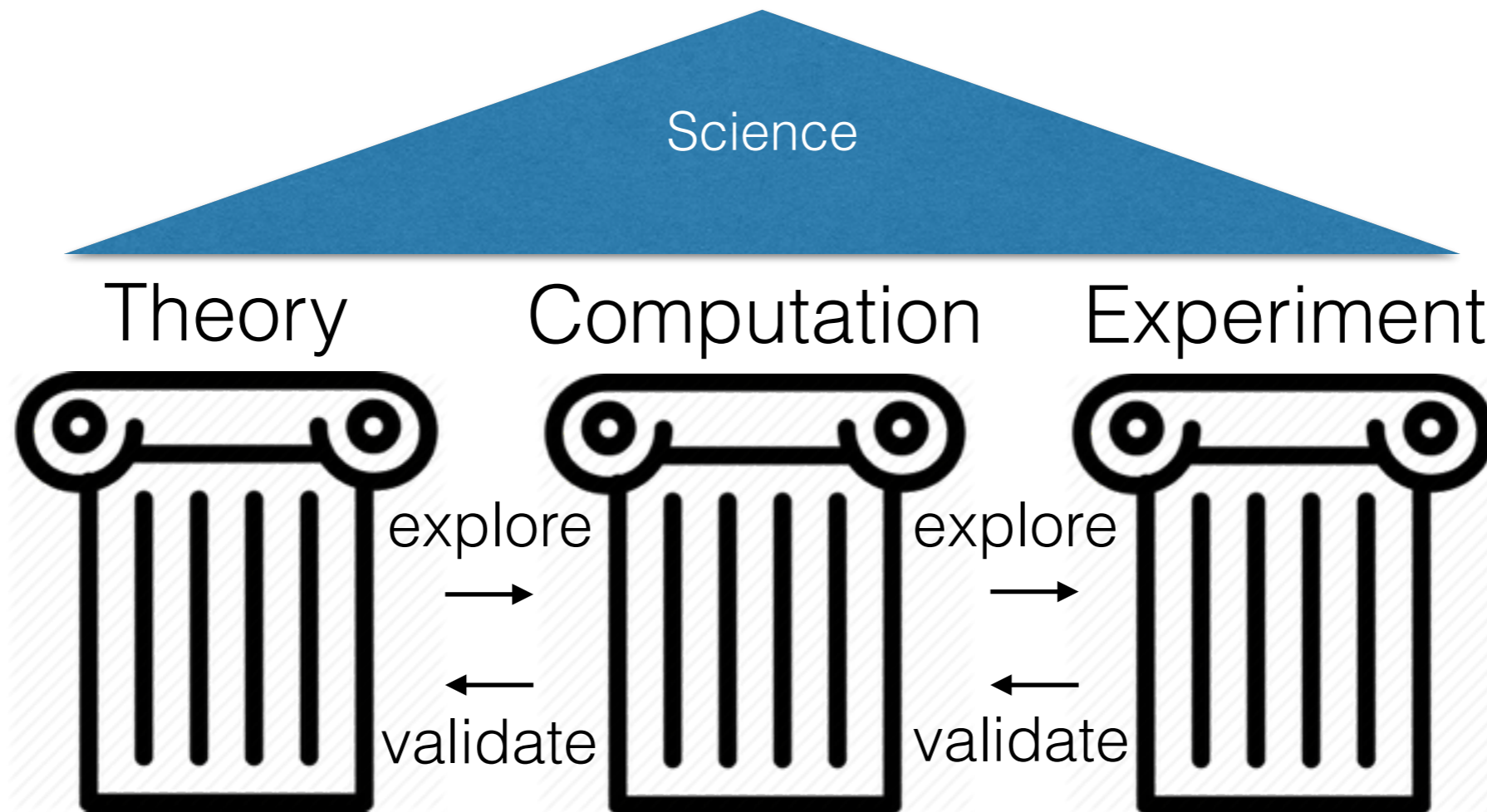
$$\ell_P = \sqrt{\frac{\hbar G}{c^3}} \approx 1.616\,199(97) \times 10^{-35} \text{ m}$$

where c is the [speed of light](#) in a vacuum, G is the [gravitational constant](#), and \hbar is the [reduced Planck constant](#).

The size of the Planck length can be visualized as follows: if a particle or dot about 0.1 mm in size (which is approximately the smallest the unaided human eye can see) were magnified in size to be as large as the [observable universe](#), then inside that universe-sized "dot", the Planck length would be roughly the size of an actual 0.1 mm dot. In other words, a 0.1 mm dot is halfway between the Planck length and the size of the observable universe on a [logarithmic scale](#).

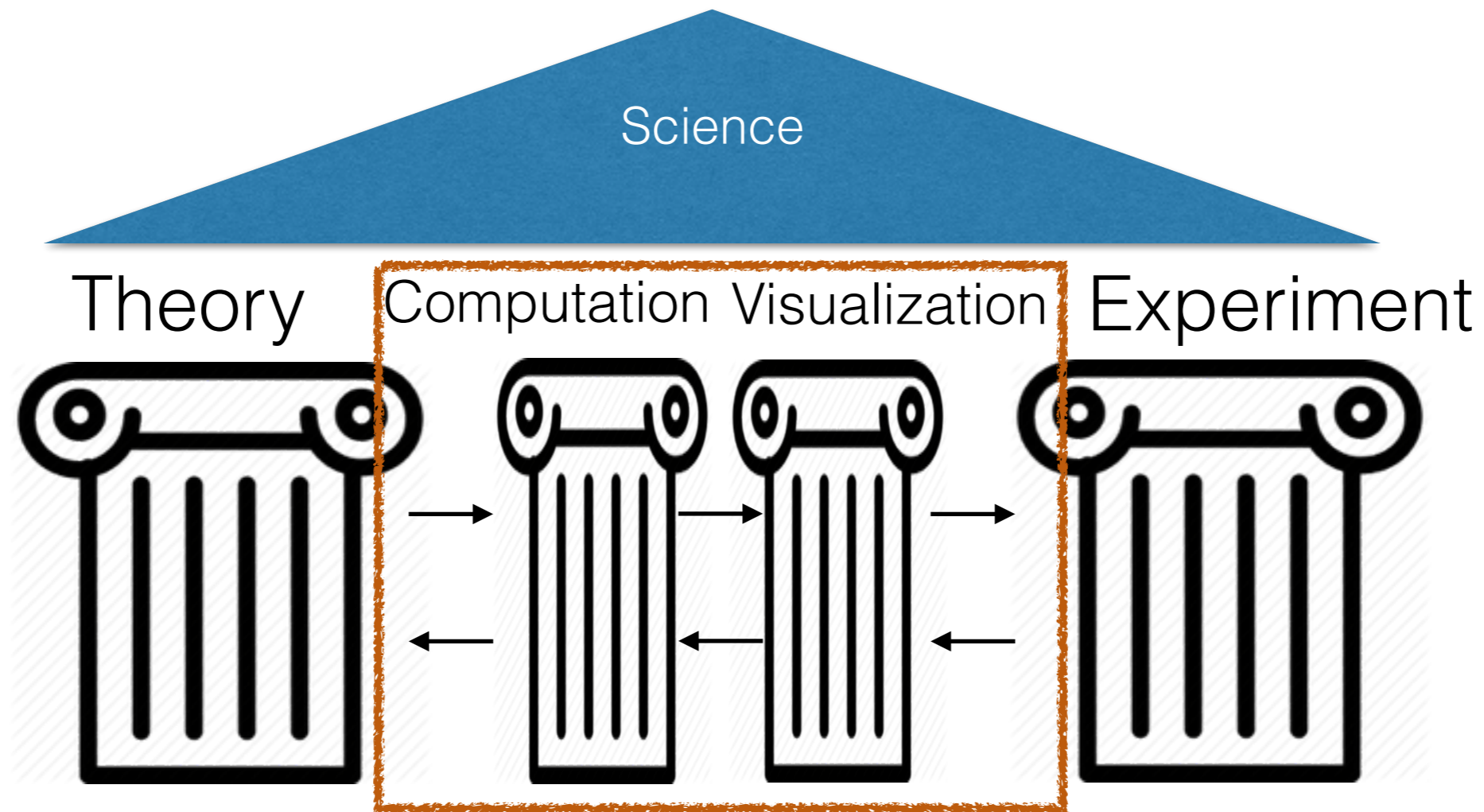
https://en.wikipedia.org/wiki/Planck_length

Theory, Computation and Experiment work best together.



Visualization

Computation is really *two* pillars



... we need to see and understand what we're computing!

<https://en.wikipedia.org/wiki/Visualization>

Visualization

From Wikipedia, the free encyclopedia

The term **visualization** may refer to:

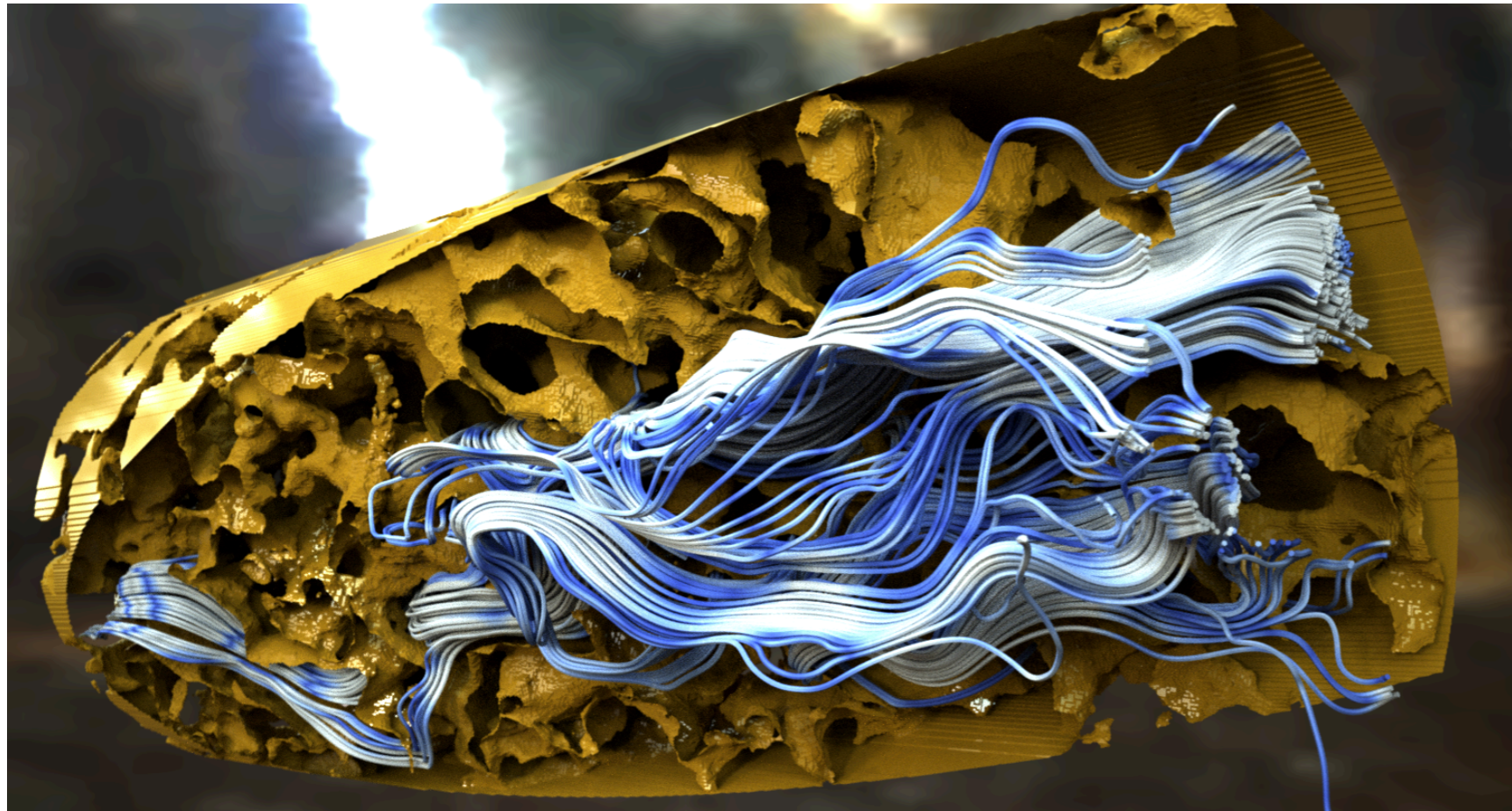
- [Mental image](#)
- [Creative visualization](#) (sports visualization)
 - [Motor imagery](#)
- [Flow visualization](#)
- [Geovisualization](#)
- [Illustration](#)
- [Information graphics](#), visual representations of information, data, or knowledge
- [Information visualization](#)
- [Interactive visualization](#)
- [Music visualization](#), a feature found in some media player software applications
- [Scientific visualization](#)
- [Security visualisation](#)
- [Software visualization](#)
- [Visualization \(computer graphics\)](#)
- [Visulation](#)
- [Guided imagery](#)

See also

- [All pages with titles containing *Visualization*](#)
- [List of graphical methods](#)
- [Image](#)
- [Mental image](#), as with imagination
- [Previsualization](#)
- [Spatial visualization ability](#), the ability to mentally manipulate 2-dimensional and 3-dimensional figures
- [Visual communication](#)
- [Visual perception](#)
- [Visual system](#)
- [Visual thinking](#)

“The purpose of computing is insight not numbers.”
-- R. W. Hamming (1961)

Illustration



Karst groundwater simulation visualized in OSPRay.

Visualization: Carson Brownlee (TACC), Data: Michael Sukop (Florida International University)

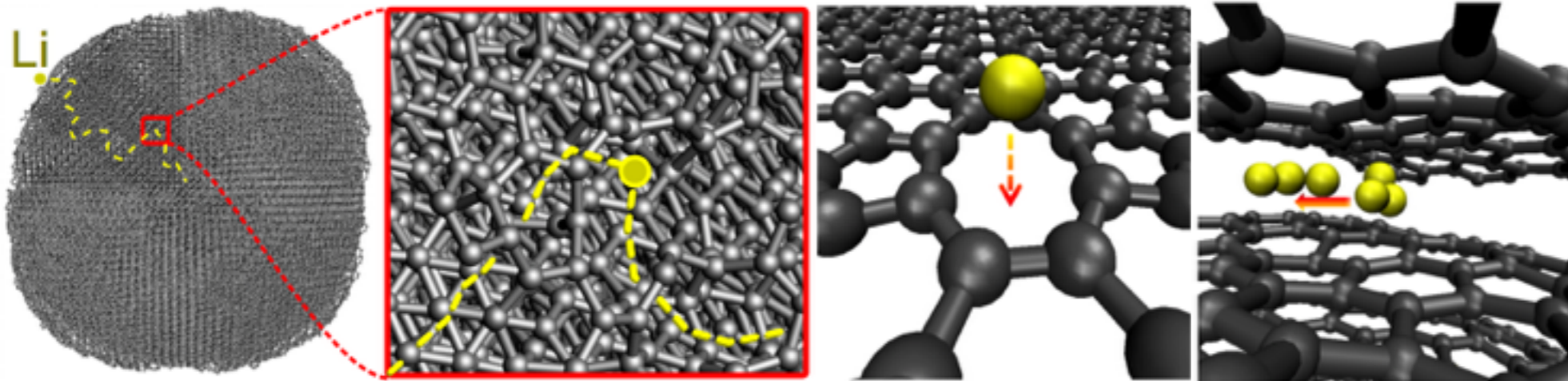


Rendering of a CO-Pt7 DFT computation in nanovol/vl3.

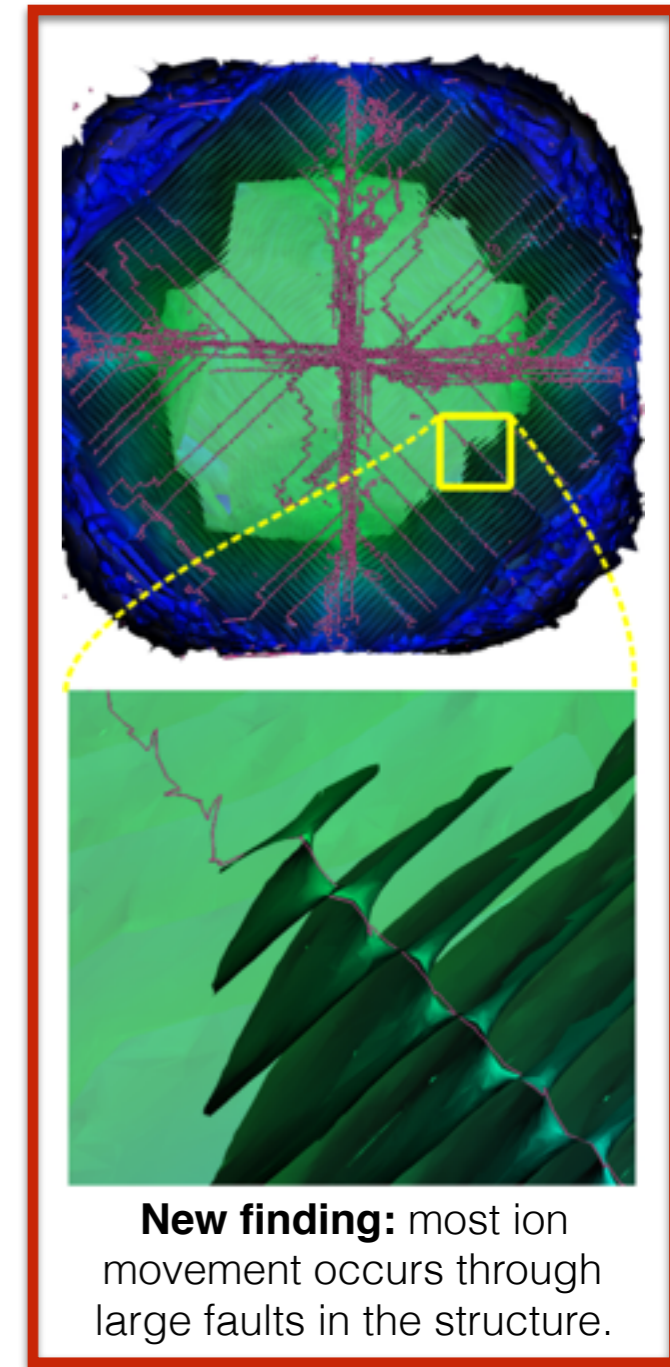
Data courtesy Aslihan Sumer and Julius Jellinek (Argonne National Laboratory)

Vis helps communicate science.

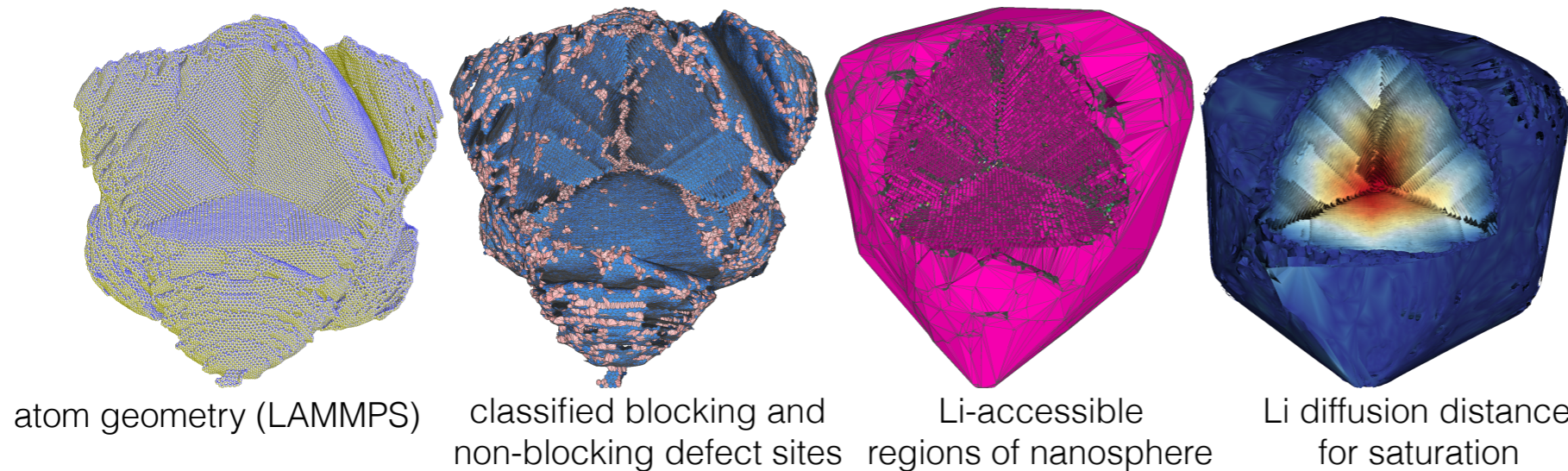
Analysis



Data: carbon nanosphere battery anode materials, Kah Chun Lau and Larry Curtiss (Argonne National Laboratory)



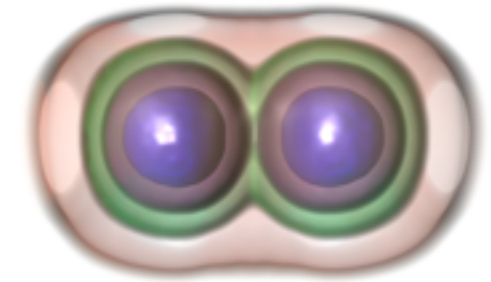
New finding: most ion movement occurs through large faults in the structure.



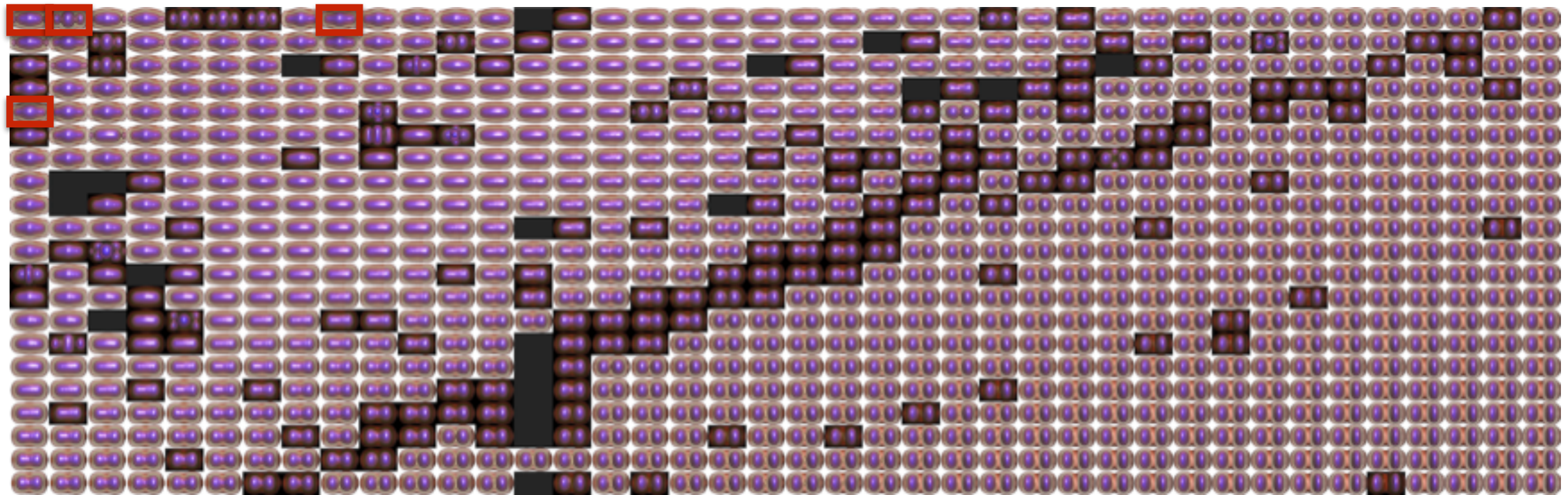
Theory: Morse-Smale analysis on a large-memory workstation finds ion diffusion pathways

Vis helps analyze computational results in-depth.

Validation



hexadecapole
moment Q40[b]



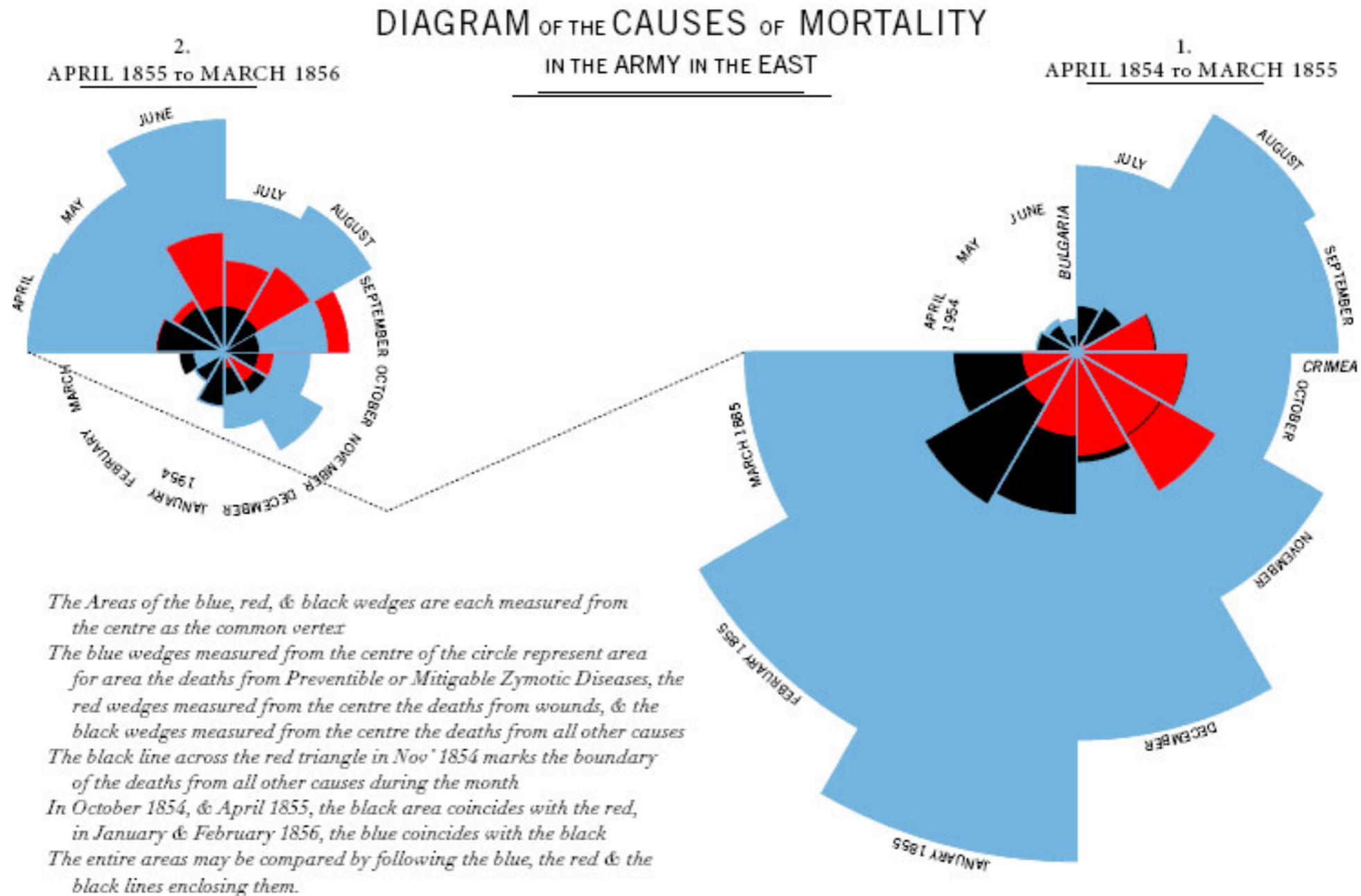
quadrupole moment Q20[b]

Parameter-space exploration of plutonium fission in nuclear DFT, Nicolas Schunck (LLNL) and Hai Ah Nam (ORNL)

White: computation succeeds (in theory)
Black: computation is known to diverge or fail.
Red: human inspection shows other failure cases.

Vis lets us debug our computational code.

Abstraction



Florence Nightengale Coxcombs: most deaths the Crimean War were due to poor sanitation in field hospitals!

Vis shows us what is important in data.

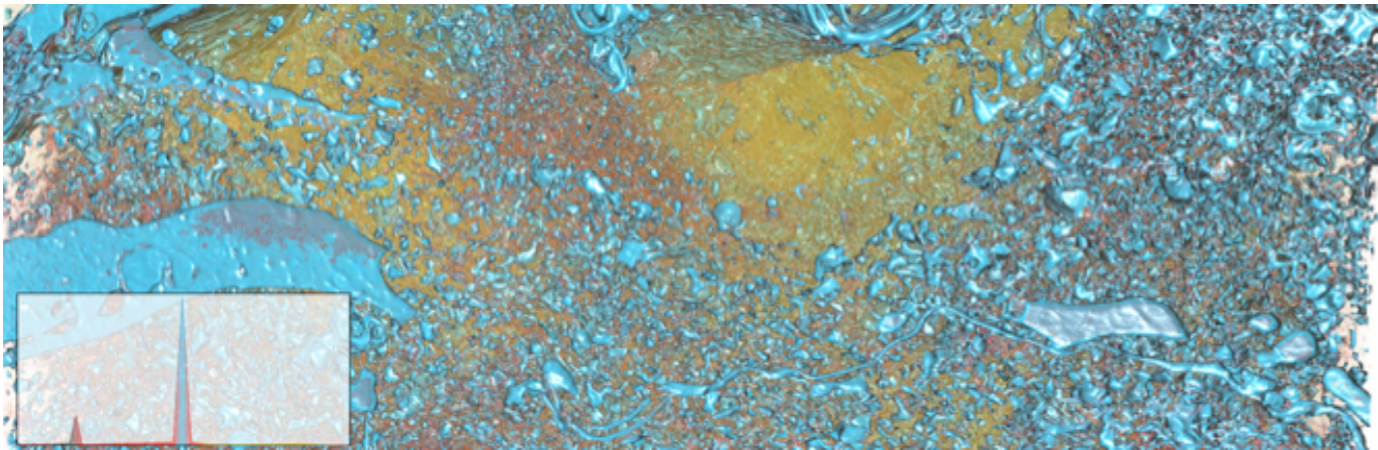
The “branches” of visualization (at the IEEE Visweek conference)

- Scientific Visualization
- Information Visualization
- Visual Analytics

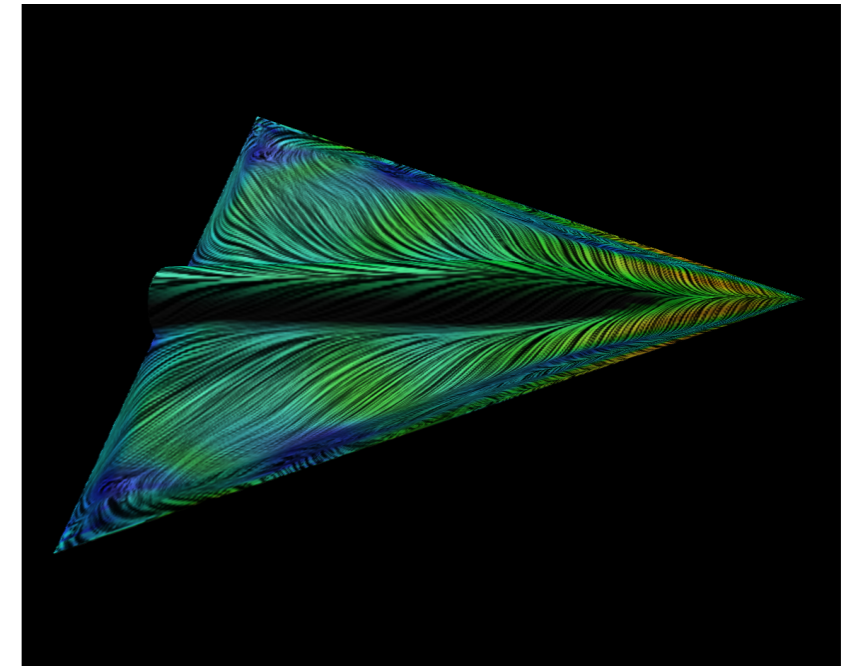
<http://ieevis.org>

Scientific visualization

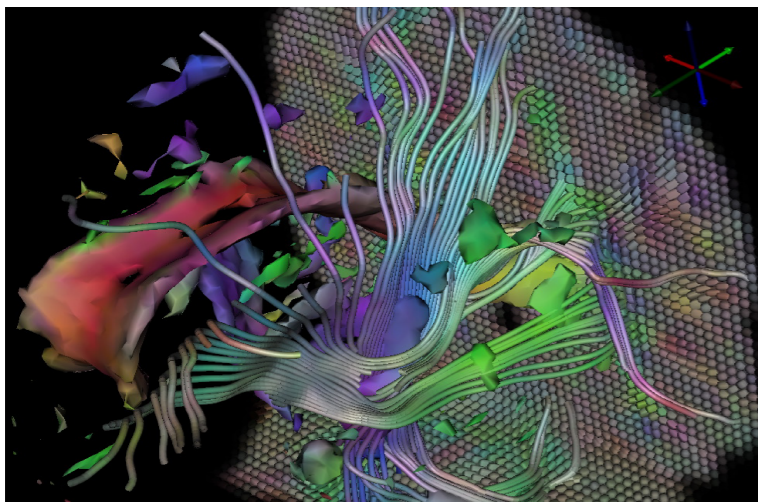
- Data have spatial context (usually from simulation or scan)
- Map spatial quantities to colors or geometry,
 $f(\text{space}, \text{time}) \rightarrow \text{rgba}$
- **2D or 3D graphics for visualization.**



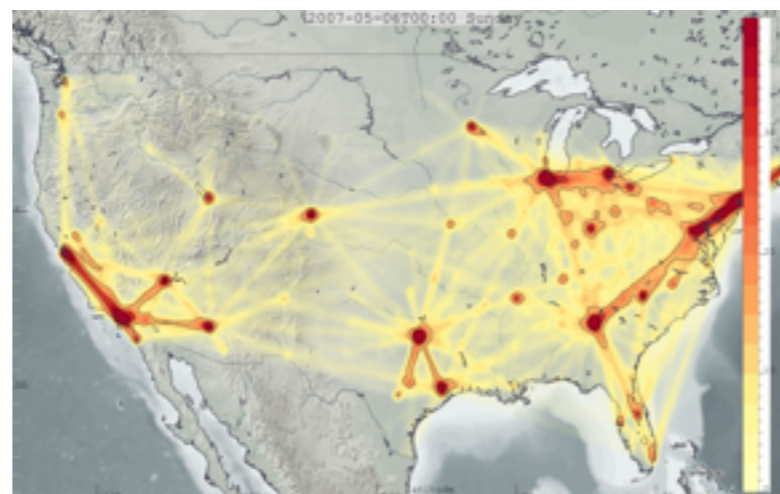
Volume rendering



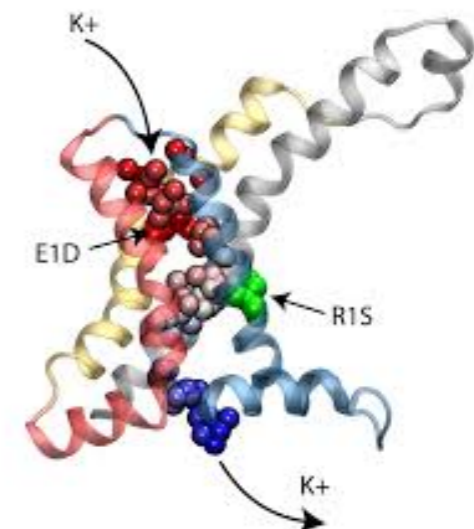
Flow visualization



Tensor field visualization



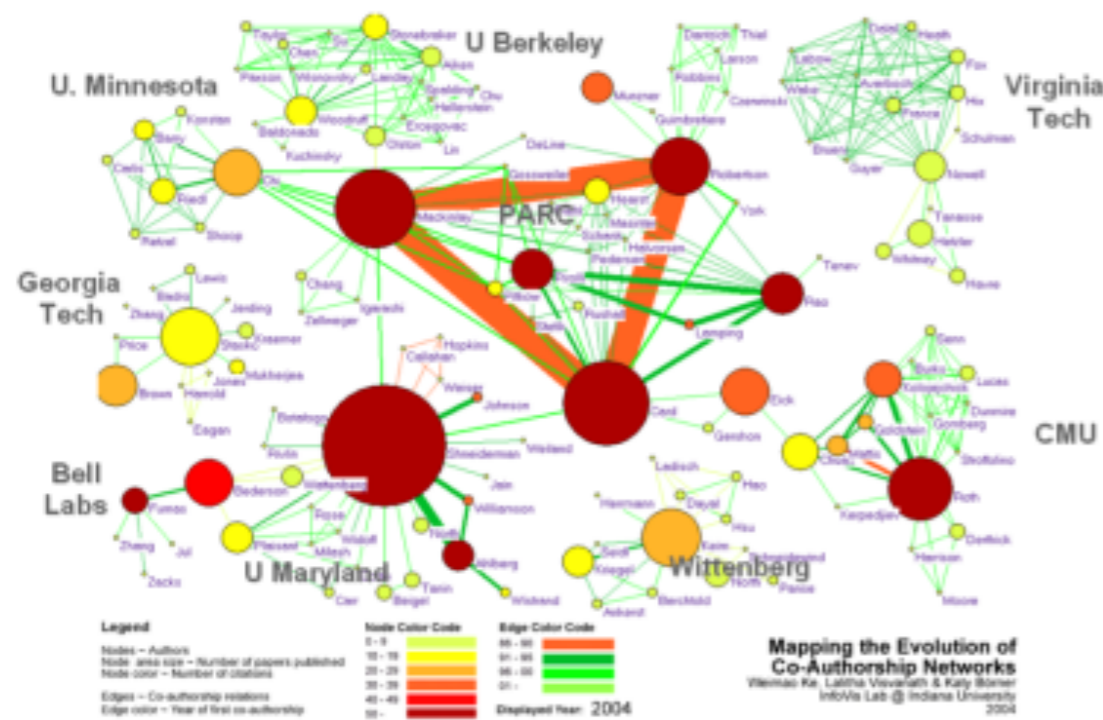
Map and GIS visualization



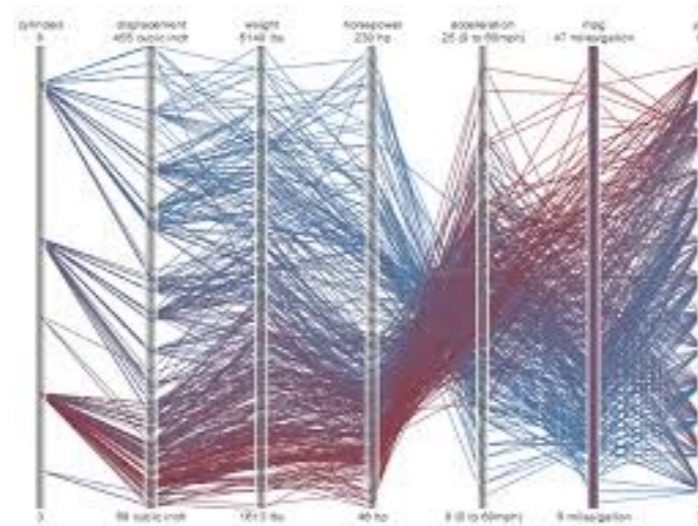
Molecular visualization

Information visualization

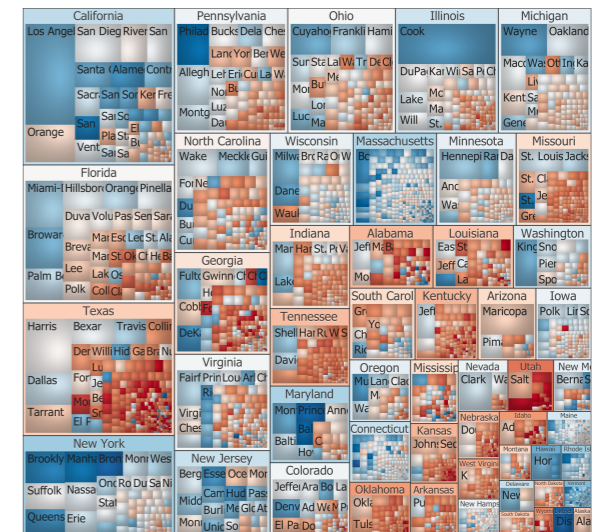
- Spatial position is secondary or non-existent.
- Illustrate relationships between abstract attributes.
- **Plots, charts, graphs, diagrams.**



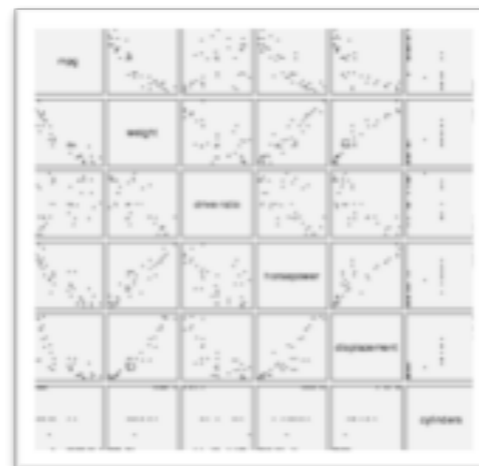
Graph and network visualization



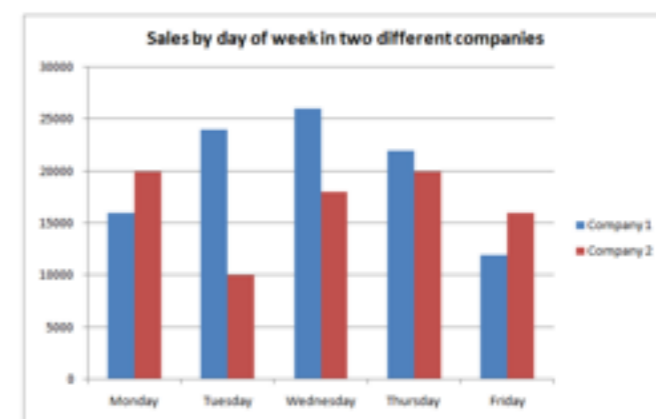
Parallel coordinates



Treemaps



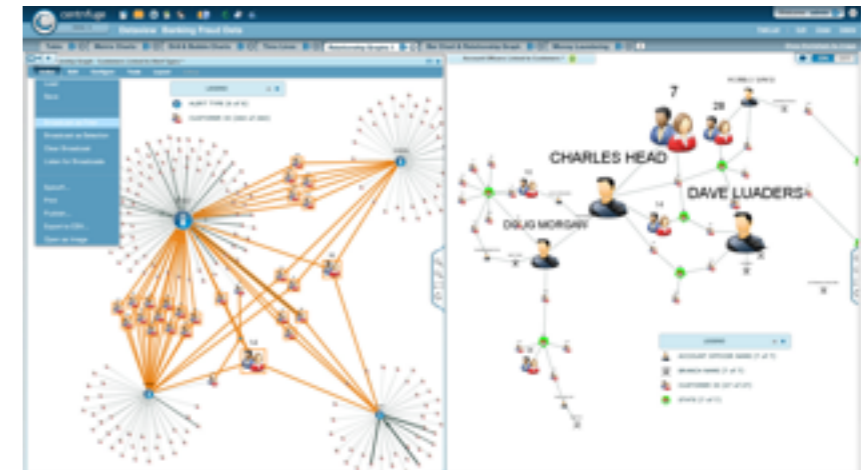
Scatterplots



Charts

Visual Analytics

- More about **interactive user interfaces** for data analysis.
- Uses techniques from both scientific visualization and information visualization, as well as statistics, perception, cognition.
- D3+Javascript, R, Matlab, information systems environments
- “Putting it all together”



Security visualization (Centrifuge)



Management Information Systems (SAS)



Genomics (Meyer et al. “Mizbee”)



So what *is* visualization?

- Visualization is how we choose to see, and therefore understand, otherwise abstract data.

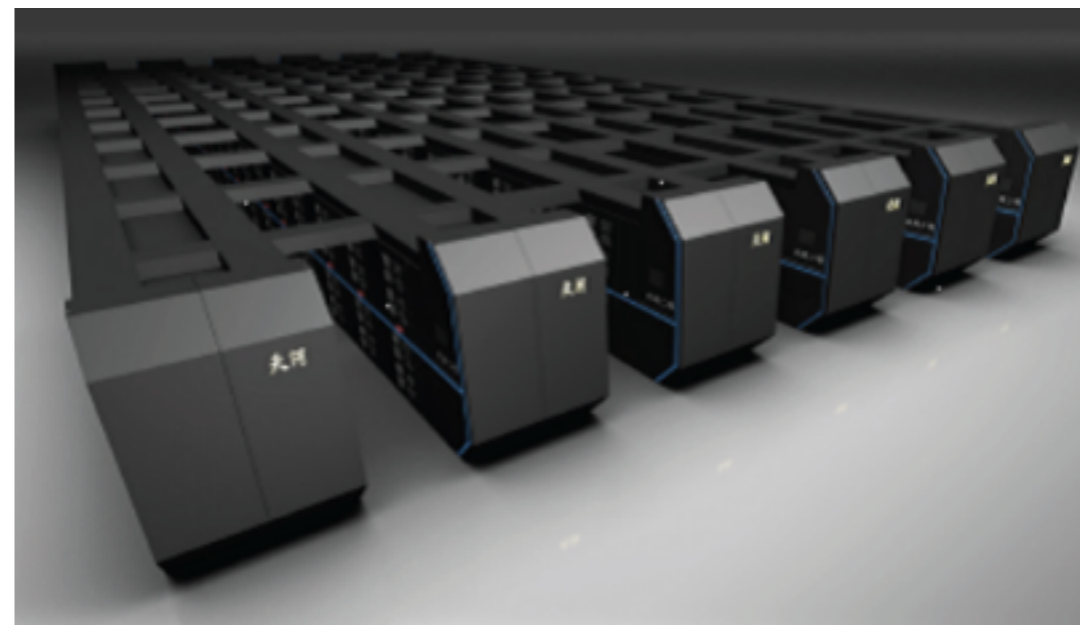
Supercomputing

More is better

- High performance computing (HPC)
- Distributed data-parallel computation
“Message Passing Interface”, MPI
- More computers
- More users
- More code
- Harder code
- Larger problems
- Tackle both the **largest scientific problems**,
and **lots of smaller ones** too!



Infiniband interconnect: up to 30,000 computers via MPI



Tianhe-2: up to 31 million cores!

“Supercomputers are time machines”

- Buddy Bland, Oak Ridge National Laboratory

"Titan has a peak performance of more than 27 petaflops – or 27 thousand trillion calculations per second (see [video](#), below). It can do more work in an hour than your personal computer can do in 20 years. “The simulations on the machine today are trying to predict what’s going to happen in the future,” Bland says. “So with a more powerful computer we can look farther into the future to predict what’s going to happen.”

<https://blogs.nvidia.com/blog/2012/11/14/why-the-worlds-fastest-computer-is-a-scientific-time-machine/>

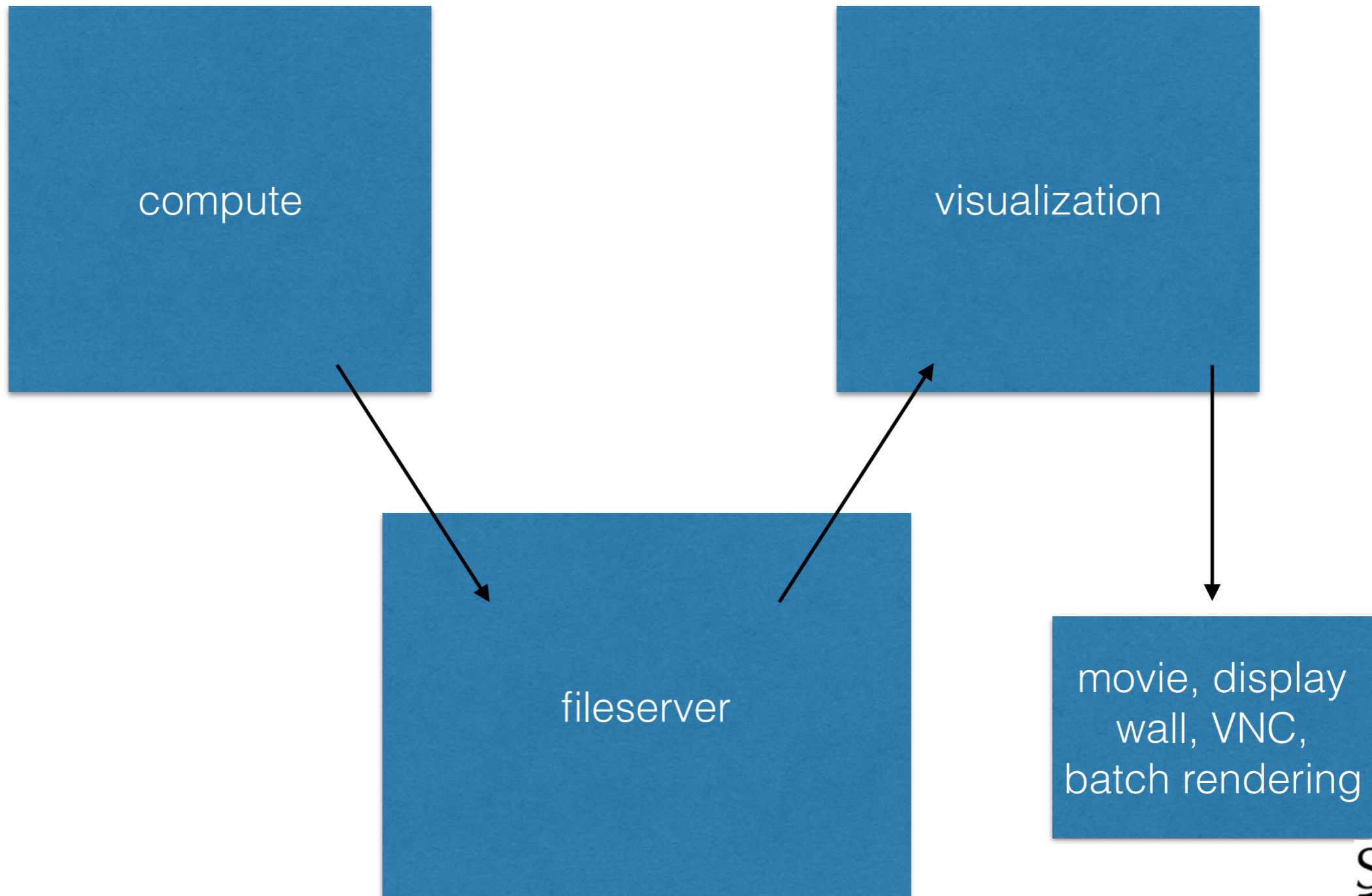
<http://www.top500.org>

RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)	RPEAK (TFLOP/S)	POWER (KW)
1	National Super Computer Center in Guangzhou China	Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 3151P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
5	DOE/SC/Argonne National Laboratory United States	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945
6	DOE/NNSA/LANL/SNL United States	Trinity - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Aries interconnect Cray Inc.	301,056	8,100.9	11,078.9	
7	Swiss National Supercomputing Centre (CSCS) Switzerland	Piz Daint - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115,984	6,271.0	7,788.9	2,325
8	HLRS - Höchstleistungsrechenzentrum Stuttgart Germany	Hazel Hen - Cray XC40, Xeon E5-2680v3 12C 2.5GHz, Aries interconnect Cray Inc.	185,088	5,640.2	7,403.5	
9	King Abdullah University of Science and Technology Saudi Arabia	Shaheen II - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Aries interconnect Cray Inc.	196,608	5,537.0	7,235.2	2,834
10	Texas Advanced Computing Center/Univ. of Texas United States	Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462,462	5,168.1	8,520.1	4,510

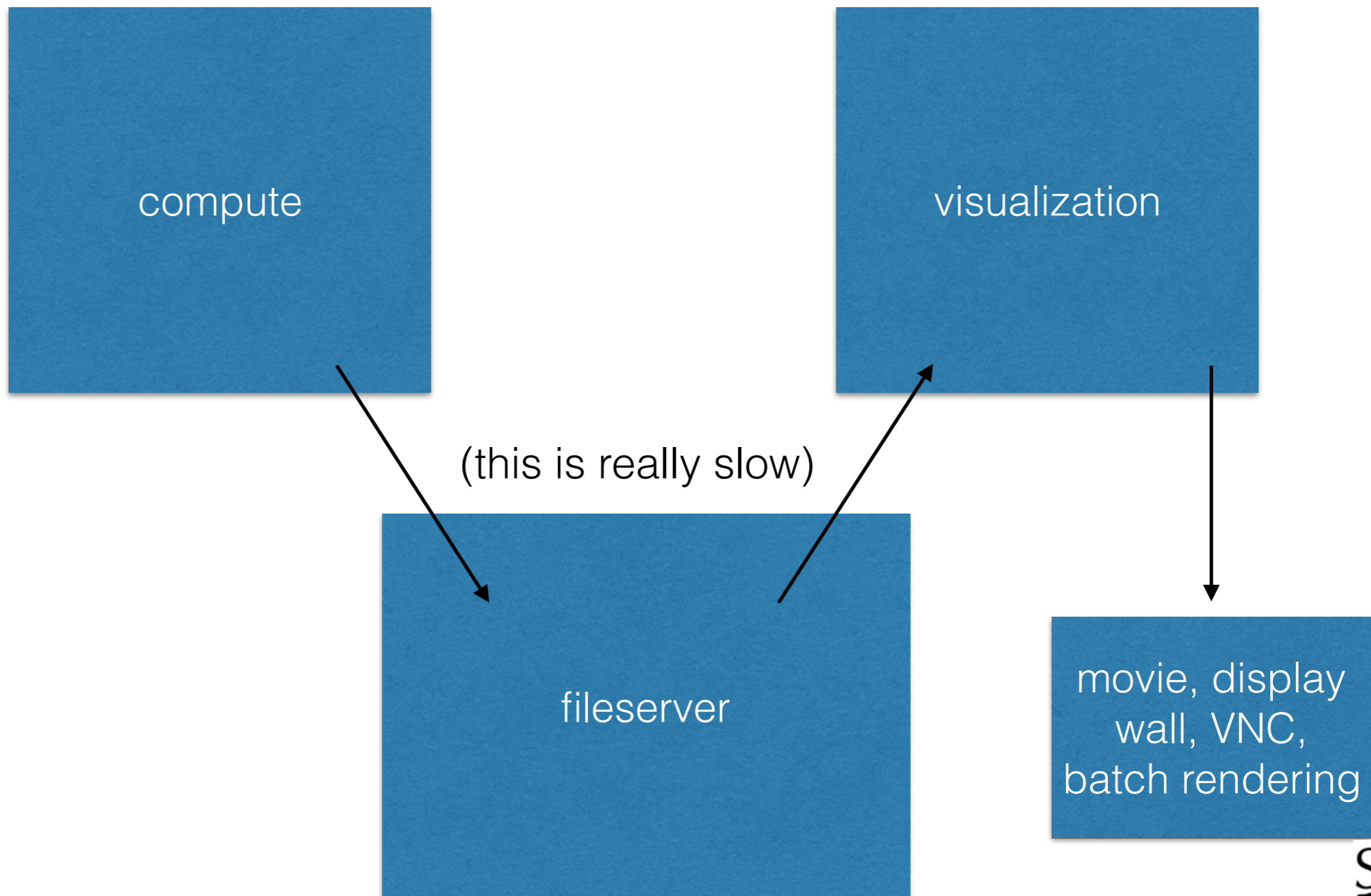
<http://www.top500.org>

RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)	RPEAK (TFLOP/S)	POWER (KW)
1	National Super Computer Center in Guangzhou China	Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 3151P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660
5	DOE/SC/Argonne National Laboratory United States	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945
6	DOE/NNSA/LANL/SNL United States	Trinity - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Aries interconnect Cray Inc.	301,056	8,100.9	11,078.9	
7	Swiss National Supercomputing Centre (CSCS) Switzerland	Piz Daint - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115,984	6,271.0	7,788.9	2,325
8	HLRS - Hochleistungsrechenzentrum Stuttgart Germany	Hazel Hen - Cray XC40, Xeon E5-2680v3 12C 2.5GHz, Aries interconnect Cray Inc.	185,088	5,640.2	7,403.5	
9	King Abdullah University of Science and Technology Saudi Arabia	Shaheen II - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Aries interconnect Cray Inc.	196,608	5,537.0	7,235.2	2,834
10	Texas Advanced Computing Center/Univ. of Texas United States	Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462,462	5,168.1	8,520.1	4,510

HPC



HPC



Texas Advanced Computing Center (TACC)



Stampede (9.6 PF peak supercomputer)

6400 Dell Poweredge nodes
2x 8-core Xeon E5-2650 2.7 GHz
32 GB RAM
Intel Xeon Phi SE10P, 8 GB RAM

128 GPU nodes: NVIDIA Tesla K20 GPU
4 Largemem nodes: 1 TB RAM, NVIDIA K20



Stampede LUSTRE filesystem

14 PB global, parallel filesystem
72 Dell R610 data servers
16 Dell R710 meta-data servers

Maverick (vis cluster)

132 nodes
2x 10-core Xeon E5-2680 2.8 GHz
256 GB RAM
NVIDIA Tesla K40 GPU, 12 GB RAM
Mellanox FDR InfiniBand



Stallion (328 Megapixel tiled display)

20 nodes, 80 screens, dual 6-core Intel
2667 W, NVIDIA Quadro K5000

Argonne Leadership Computing Facility (ALCF)



Mira (10 PF peak supercomputer)

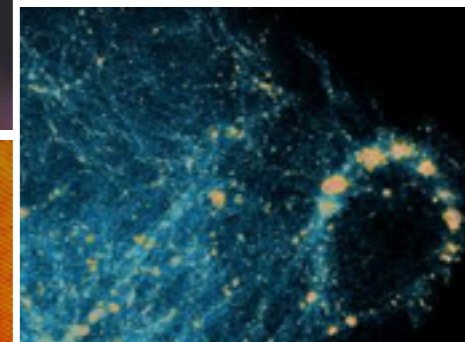
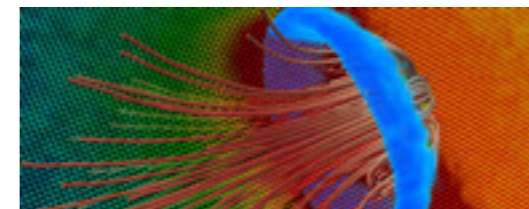
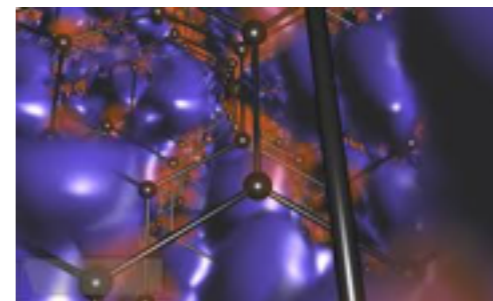
49,152 IBM BlueGene/Q nodes
16 1.6 GHz PowerPC A2 Cores, 16 GB RAM
5D torus interconnect



Cooley (vis cluster)

126 nodes
2x 6-core Xeon E5-2620 2.4 GHz
384 GB RAM
NVIDIA Tesla K80 GPU, 24 GB RAM

Mira filesystem
24 PB IBM GPFS (global parallel filesystem)

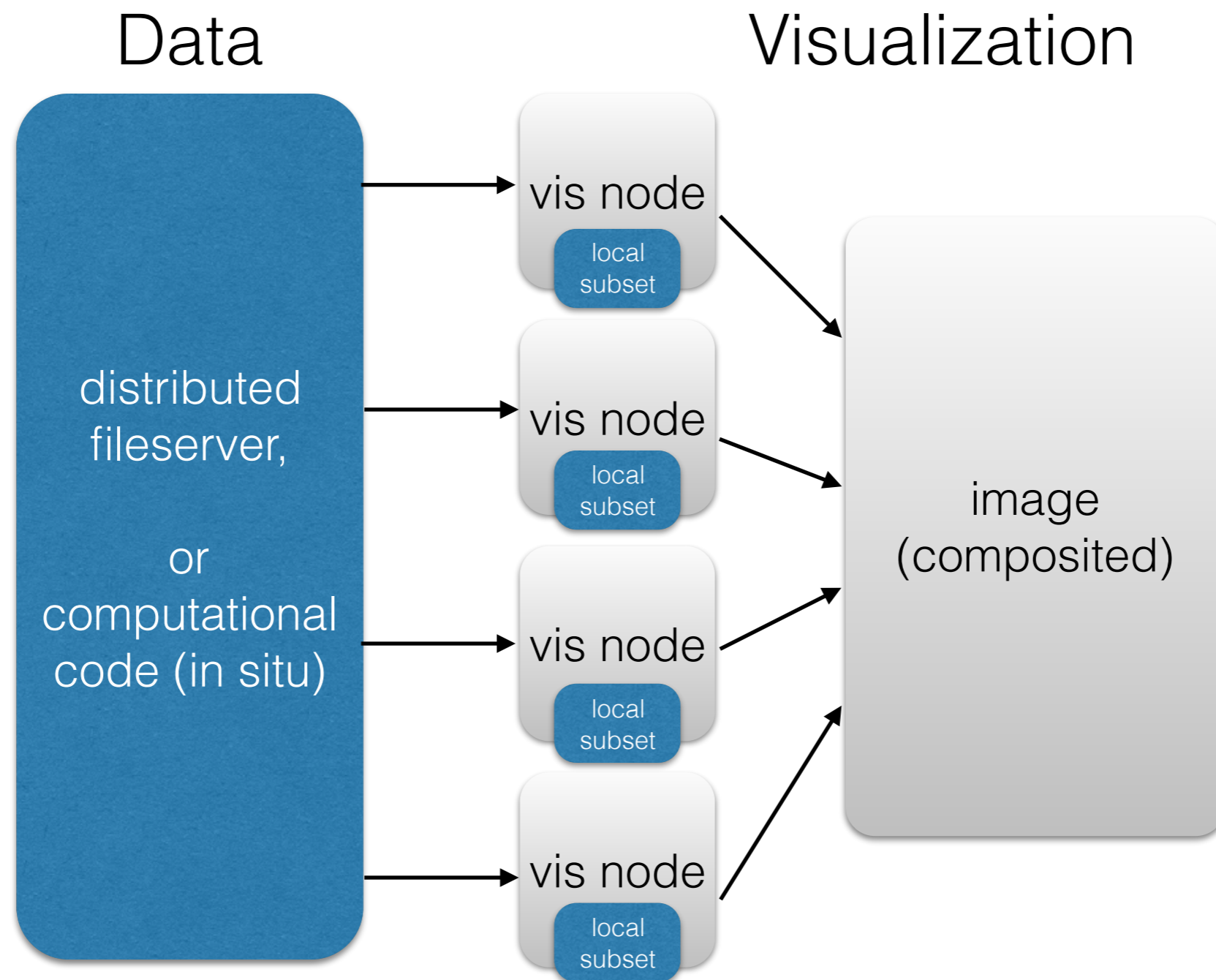


parallel batch rendering software
ParaView, VisIt, vl3, nanovol

HPC + visualization

- One of the challenges of scientific visualization is to solve the largest-scale scientific problems coming out of supercomputing.

Data parallel visualization



ParaView

- <http://www.paraview.org>
- Built on top of the Visualization Toolkit (VTK) <http://www.vtk.org>
 - full data model for 2D, 3D, unstructured mesh data
- Data-parallel reading, filtering, and compositing (rendering)
- great for unstructured and 2.5D (i.e. climate, weather) data

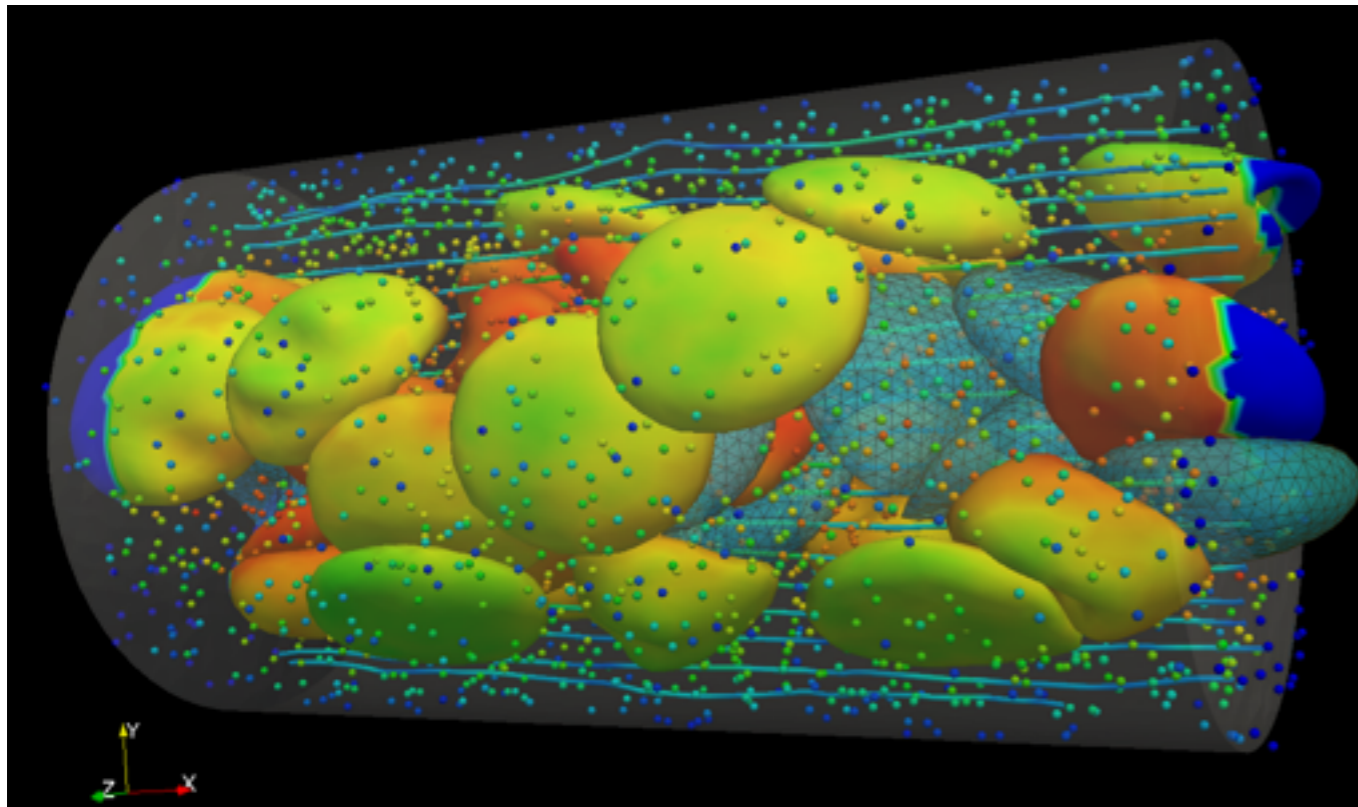


Image courtesy Joe Insley, ANL

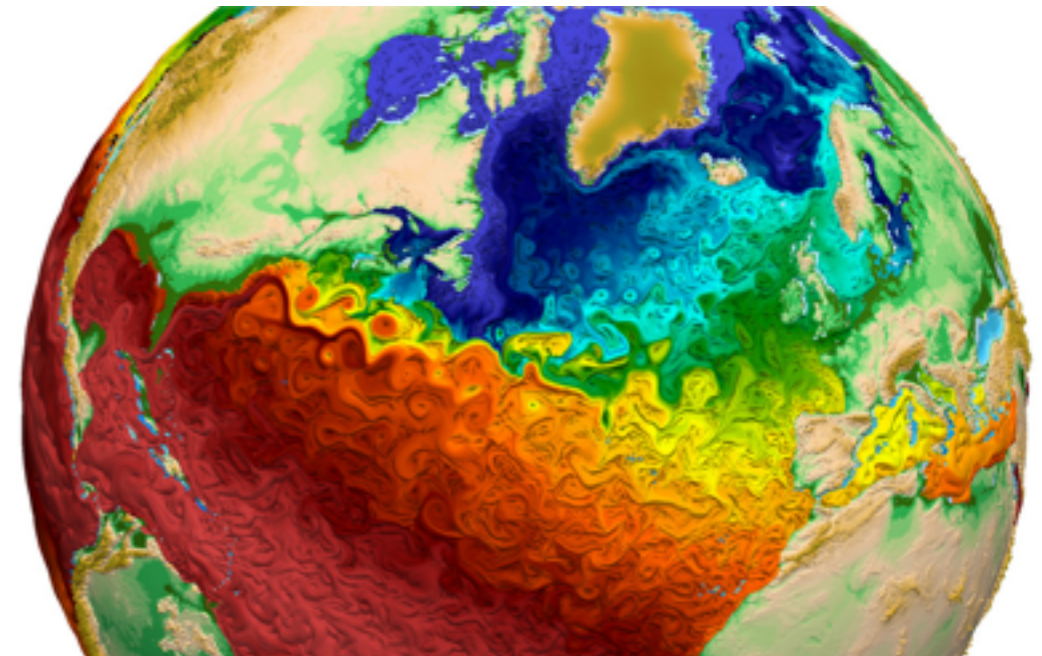
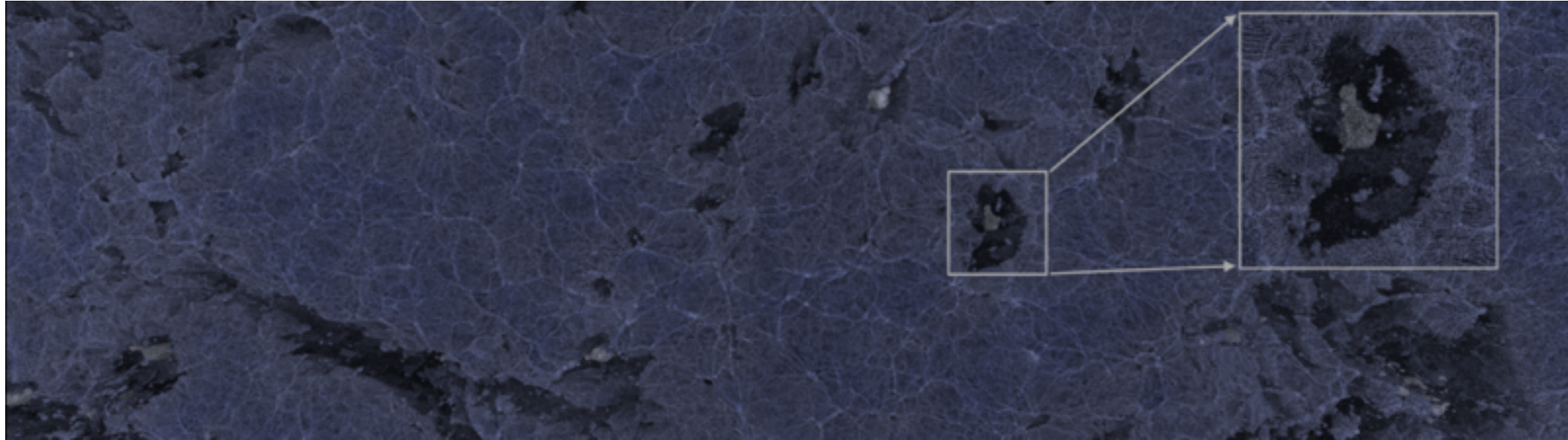


Image courtesy LANL

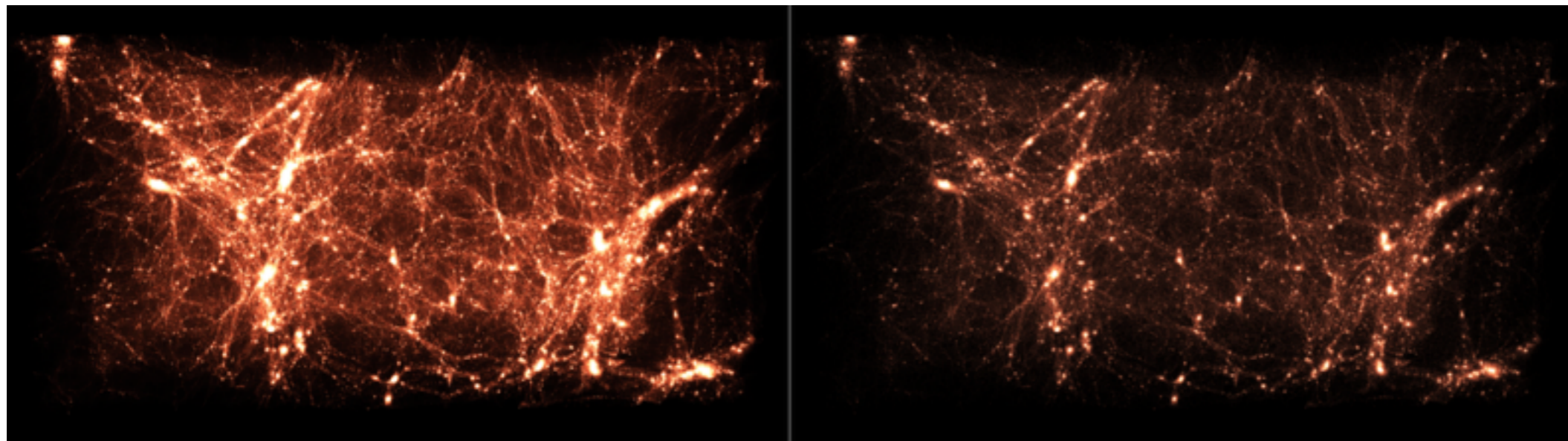
Two ways to visualize 30 billion particles!

1. One huge 72-core CPU, 3 TB of shared memory, ray tracing



I Wald, A Knoll, G Johnson, W Usher, M E Papka, V Pascucci. "CPU Ray Tracing Large Particle Data with Balanced P-k-d Trees", IEEE Vis 2015
30 billion particle (450 GB) subset of a PM3D simulation, ray traced with ambient occlusion
6 FPS (72-core 2.5 GHz Xeon E7-8890 v3).

2. 128-GPU cluster, 1 TB distributed memory, rasterization



30 billion particles: ~20 MRays/s

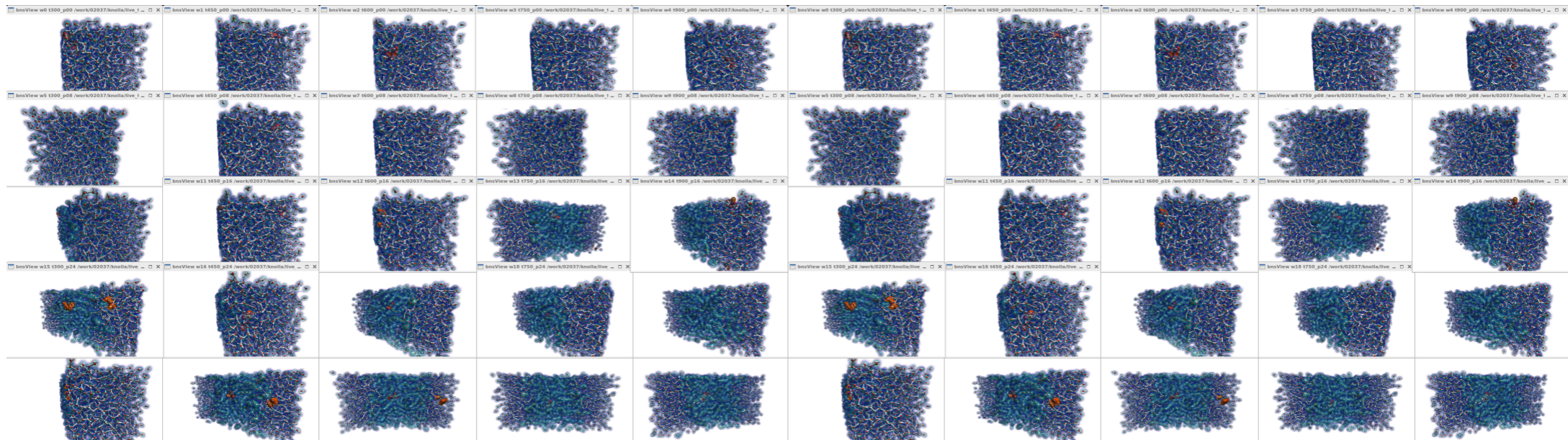
3 billion particles: ~200 MRays/sec

S. Rizzi, M. Hereld, J. Insley, M. Papka, V. Vishwanath. "Large-Scale Parallel Vis. of Particle-Based Simulations using Point Sprites and LOD".
EGPGV 2015

In situ visualization



Running on 32 Xeon Phi's on Stampede, streaming 1 gigapixel live at ~2 fps to the Stallion display wall.



50 simulation+visualization ensembles on the Cherry Creek Phi cluster at SC13

You can do this on GPU's too!

Or, purely in-core on compute CPUs.

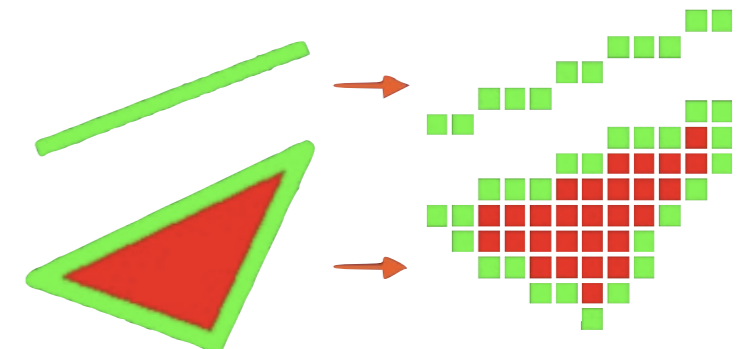
A visualization algorithm: volume rendering

Two ways of doing 3D computer graphics

- Computer graphics is the process of converting a 3D scene (model) into a 2D image (frame buffer), via a camera model.
- Principally, there are two ways of doing this:

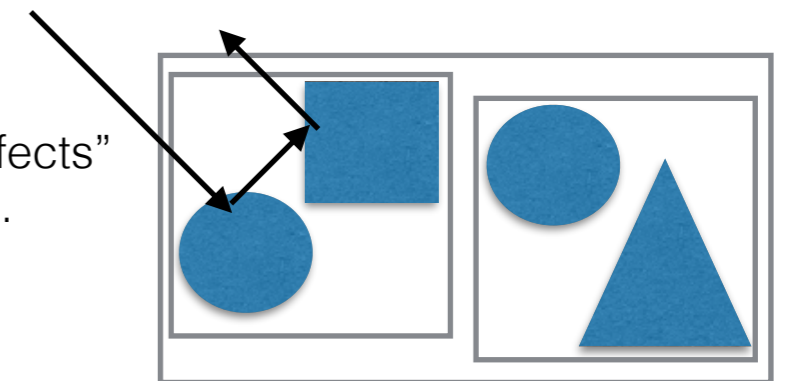
- **Rasterization**

- “project the scene, sort and shade textured fragments, and shade”
- The camera transforms the primitives.
- 4x4 matrix multiplication, Z-buffer algorithm, scan conversion (“projected space”)
- *Cost: $O(N)$ — “really fast for small data, really slow for large data”*
- **Everything has to be a triangle**
- APIs: OpenGL / WebGL / three.js, DirectX, Vulkan



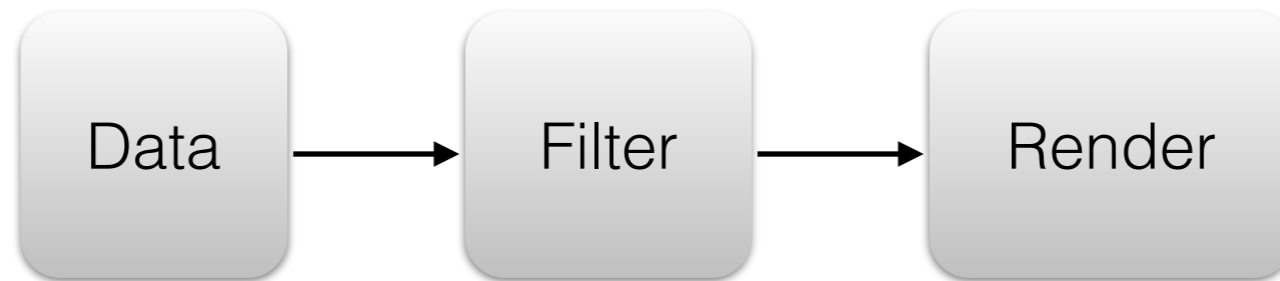
- **Ray tracing**

- Ray casting: “generate rays, search the scene for what the ray hits, and shade”
- Ray tracing: “keep bouncing rays off objects to get reflections, refractions, other effects”
- Camera defines the ray; primitives remain in native 3D coordinates (“world space”).
- *Cost: $O(P \log N)$ — “slow for small data, fast for large data.”*
- **Can support non-triangle data — spheres, cylinders, volume data**
- APIs: Intel Embree & OSPRay, NVIDIA OptiX & IndeX, write your own!

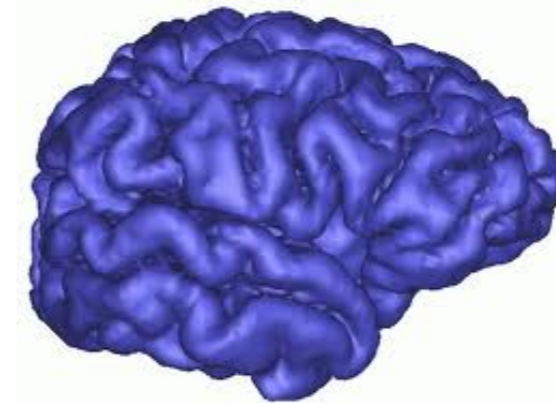
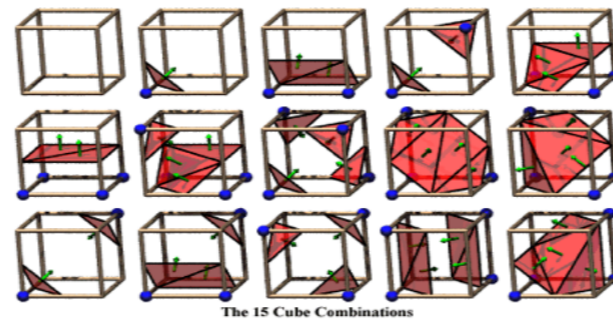


- Volume rendering can be implemented either via ray tracing (sampling along the ray) or rasterization (with textured triangle proxy geometry)

Rasterization route



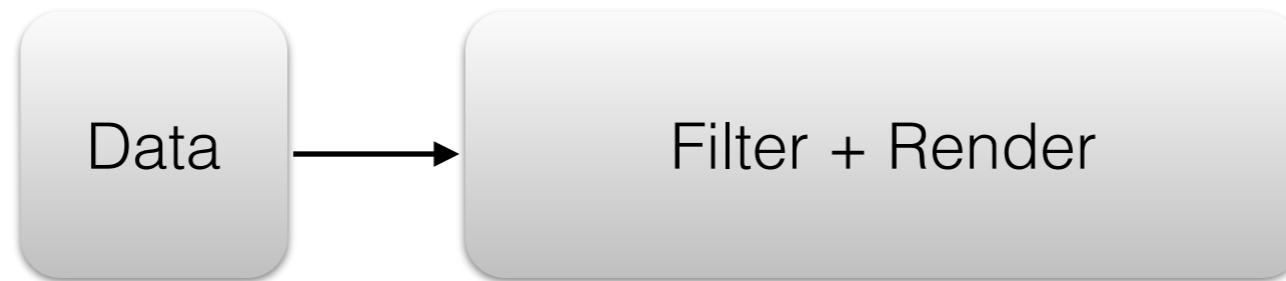
0	4	8	0
4	14	9	0
6	11	1	0
2	1	0	0



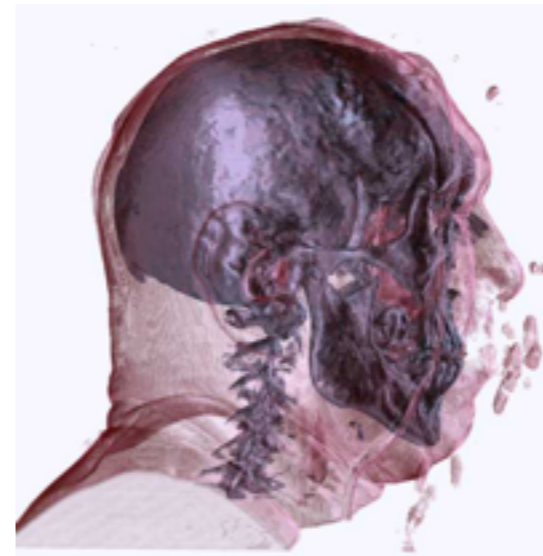
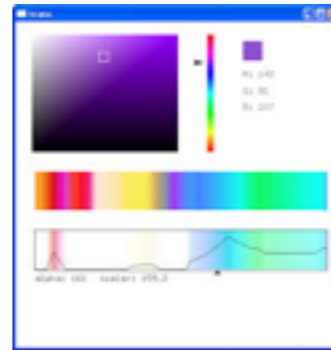
Rasterization route



Direct route

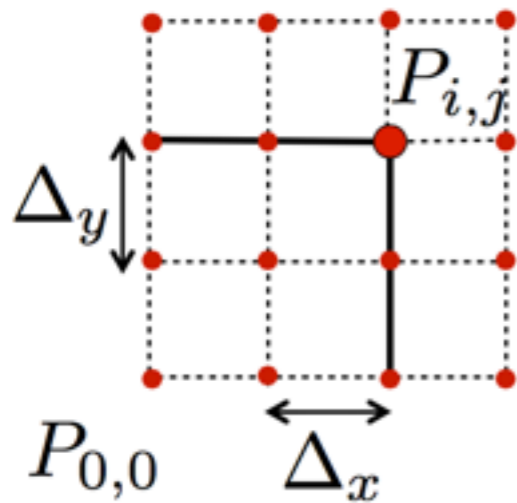


0	4	8	0
4	14	9	0
6	11	1	0
2	1	0	0

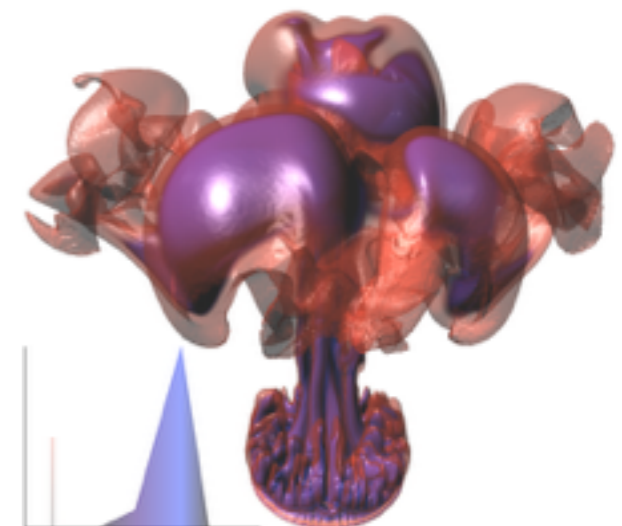
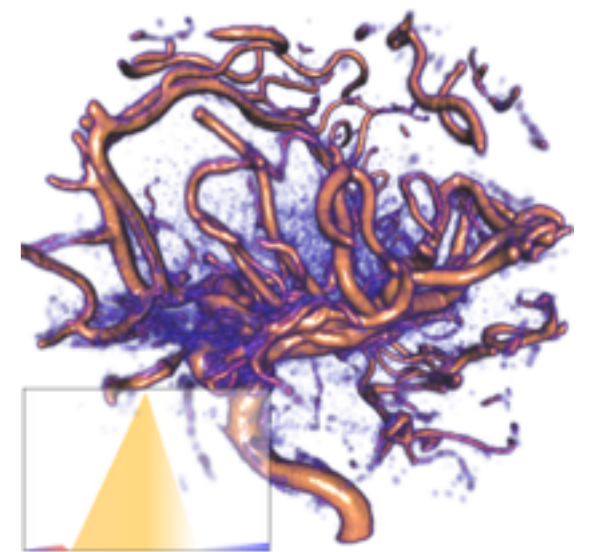
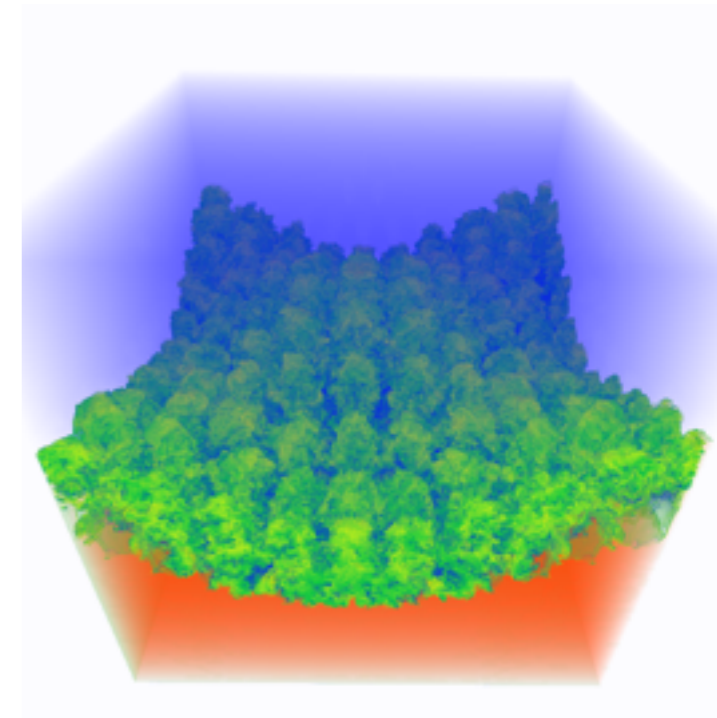


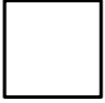
Volume data

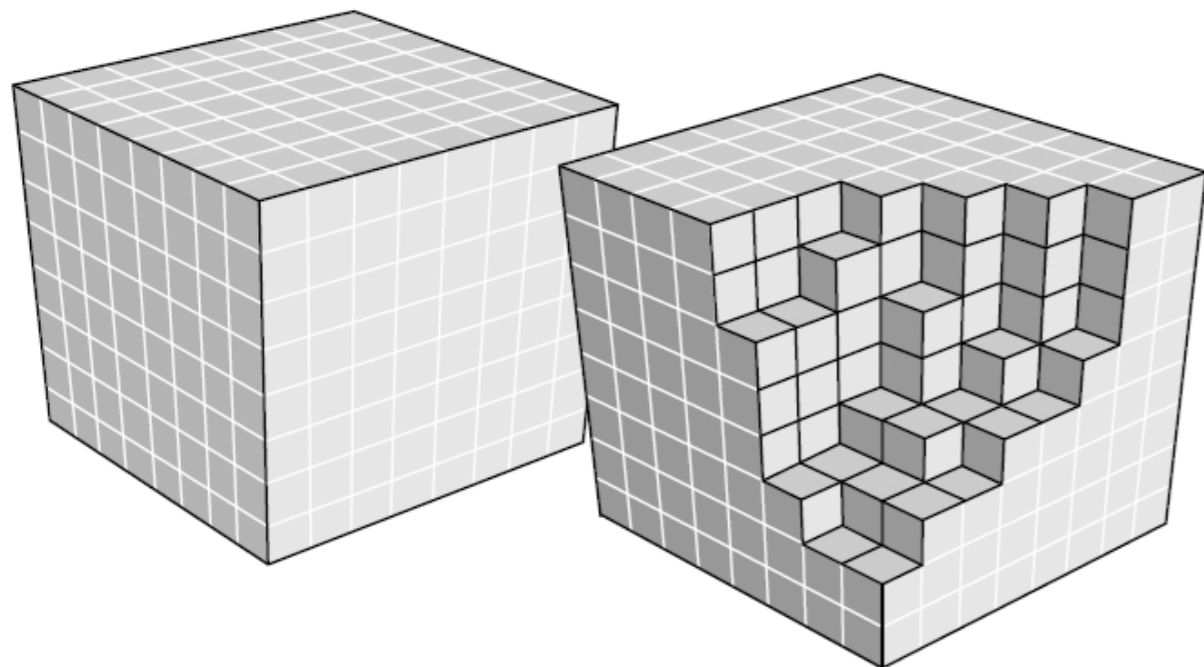
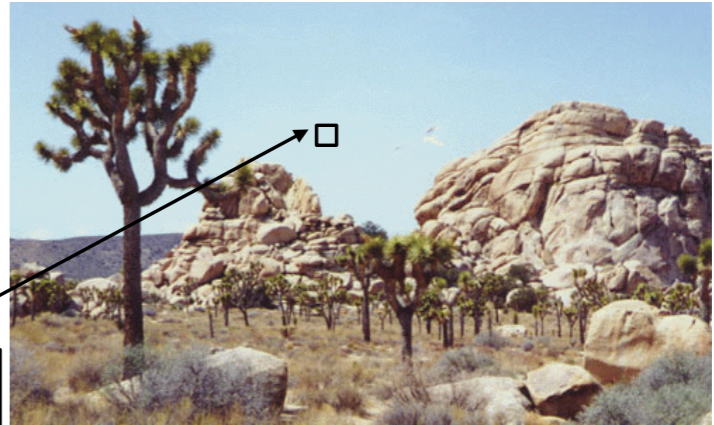
- Usually, a uniform 3D grid of *voxels* —
 - voxels are 3D equivalent of 2D pixels.
- Some “color map”, or “transfer function”, to give each voxel a color



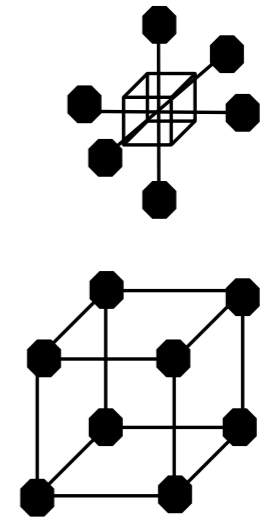
$$P_{i,j,k} = P_{0,0} + i\Delta_x\vec{e}_x + j\Delta_y\vec{e}_y$$



- Representation of scalar 3D data set $\Omega \in R^3 \rightarrow R$
- Analogy: pixel (picture element) 
- Voxel (volume element), with two interpretations:
 - Values between grid points are resampled by interpolation

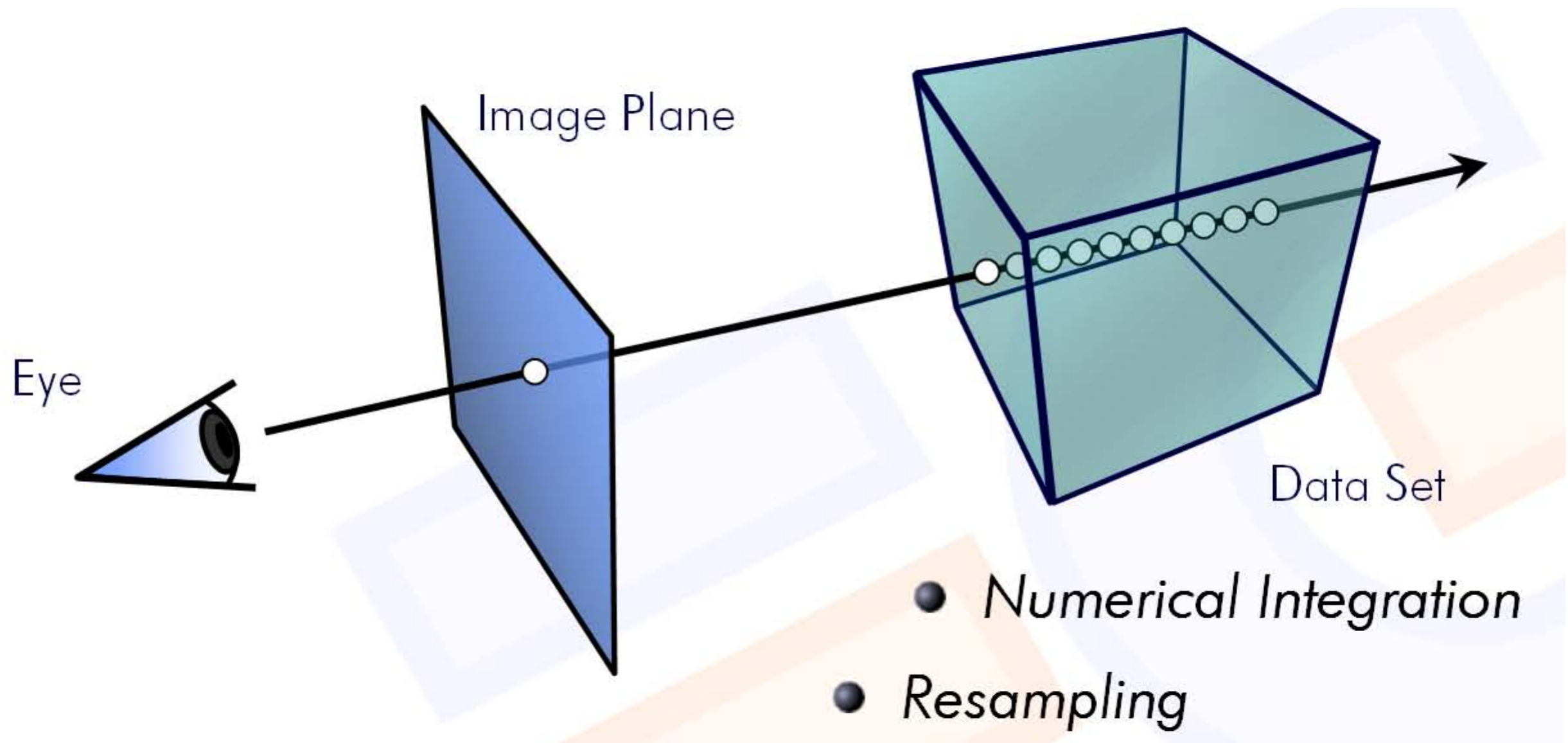


- Collection of voxels
- Uniform grid



You can **sampleVolume()** anywhere and it will give you a value.
i.e. pretend it is “continuous.”

The basic algorithm



The basic algorithm

Given:

float2 pixel.xy, float3 volumeBoxMin, float3 volumeBoxMax

1. generate ray origin, direction from pixel
- 2a. find tStart, tEnd where ray intersects volume box
- 2b. accumulate the color along the samples (step 2b)
3. set the pixel value as that color

The basic algorithm

Given:

float2 pixel, float3 volumeBoxMin, float3 volumeBoxMax, Volume volume

1. generate ray origin, direction from pixel

2a. find tStart, tEnd where ray intersects volume box

2b. accumulate the color along the samples (step 2b)

3. set the pixel value as that color

The basic algorithm

Given:

float2 pixel, float3 volumeBoxMin, float3 volumeBoxMax, Volume volume

1. generate ray origin, direction from pixel

2a. find tStart, tEnd where ray intersects volume box

2b. accumulate the color along the samples (step 2b)

3. set the pixel value as that color

The basic algorithm

Given:

float2 pixel, float3 volumeBoxMin, float3 volumeBoxMax, Volume volume

1. generate ray origin, direction from pixel
- 2a. find tStart, tEnd where ray intersects volume box
- 2b. accumulate the color along the samples (step 2b)**
3. set the pixel value as that color

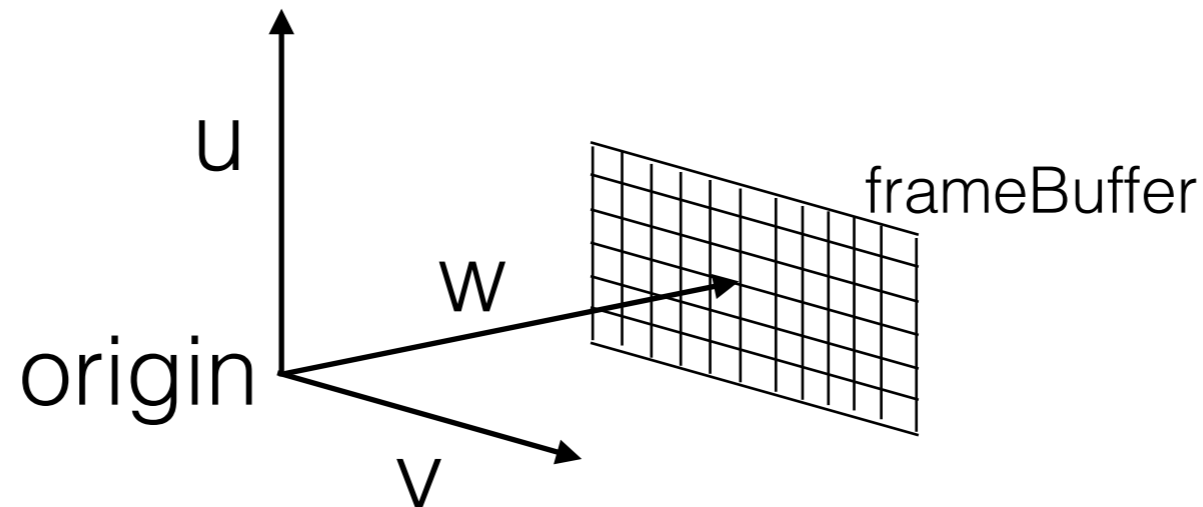
The basic algorithm

Given:

float2 pixel, float3 volumeBoxMin, float3 volumeBoxMax, Volume volume

1. generate ray origin, direction from pixel
- 2a. find tStart, tEnd where ray intersects volume box
- 2b. accumulate the color along the samples (step 2b)
- 3. set the pixel value as that color**

Generate a ray



- **Camera setup (per frame, on the CPU):**

```
float3 u,v,w;  
w = normalize(lookAt - origin);           //a bit of graphics math to give you a ray  
u = cross(up, w);  
v = cross(w,u);  
u = normalize(u);  
v = normalize(v);
```

```
float tanThetaOver2 = tanf(fovy * .5 * PI / 180);  
float aspect = width / height;
```

```
float3 framebuffer_u = u * tanThetaOver2;  
float3 framebuffer_v = v * tanThetaOver2 / aspect;
```

- **Ray generation (per pixel, e.g. in a fragment shader on the GPU, or task per pixel):**

```
float3 rayOrigin; //this is just the position of the eye  
float3 rayDirection = w + (framebuffer_u * pixelPos.x) + (framebuffer_v * pixelPos.y);
```


Ray-box intersection

Given:

float2 pixel.xy, float3 volumeBoxMin, float3 volumeBoxMax

1. generate ray origin, direction from pixel

2a. find tStart, tEnd where ray intersects volume box

```
float3 dt0 = volumeBoxMin-rayOrigin / rayDirection;  
float3 dt1 = (volumeBoxMax - origin) / rayDirection;
```

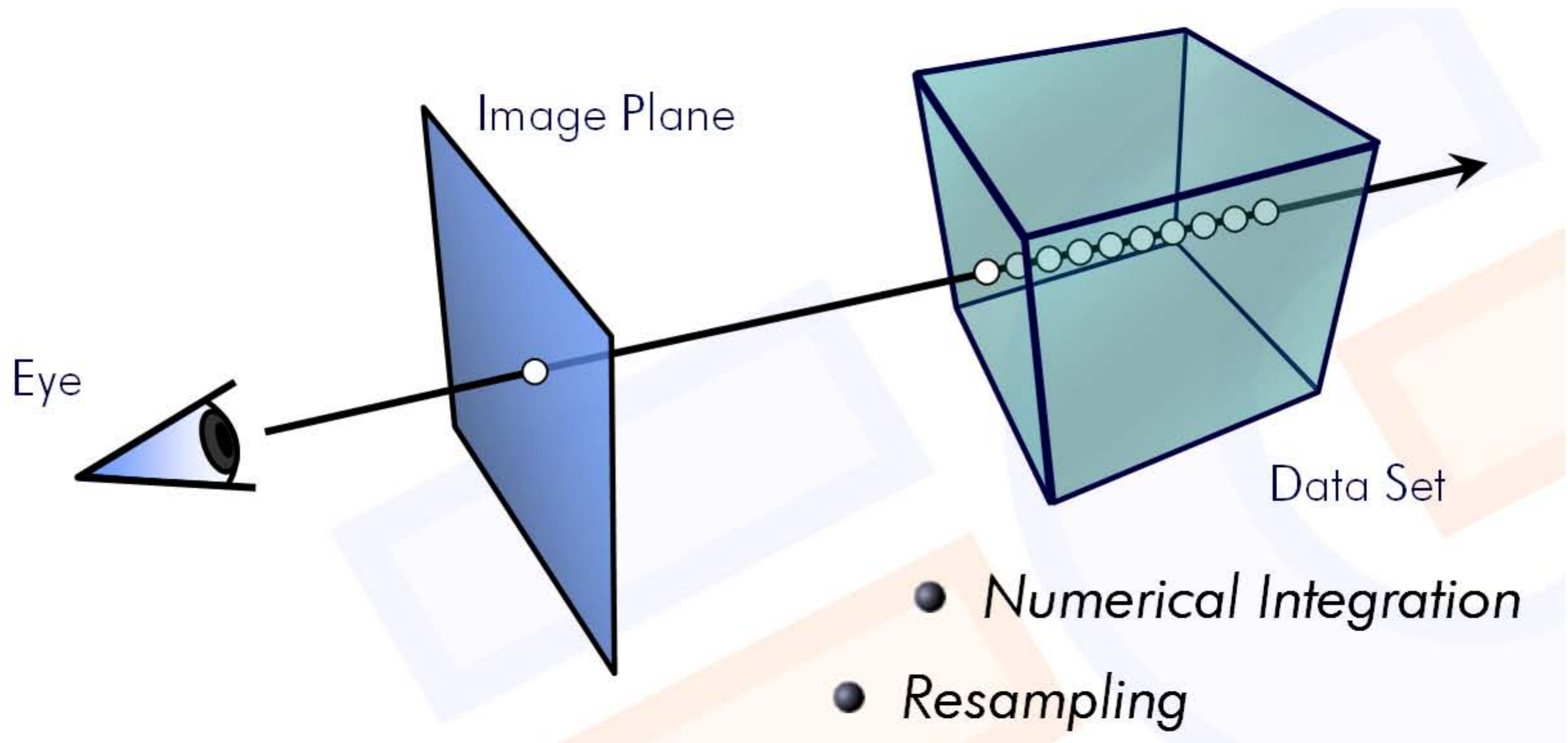
```
float3 tmin = min(dt0, dt1);  
float3 tmax = max(dt0, dt1);
```

```
float tStart = max(tmin.x, max(tmin.y, tmin.z));  
float tEnd = min(tmax.x, min(tmax.y, tmax.z));
```

2b. accumulate the color along the samples (step 2b)

3. set the pixel value as that color

Sample along the ray and accumulate color



The basic algorithm

Given:

float2 pixel, float3 volumeBoxMin, float3 volumeBoxMax

1. generate ray origin, direction from pixel

2a. find tStart, tEnd where ray intersects volume box

2b. accumulate the color along the samples (step 2b)

```
                // r g b a  
Color accumColor = (0, 0, 0, 0); //red, green, blue, alpha (transparency)
```

```
for(t = tStart, t <= tEnd, t += deltaT)
```

```
    Point p = origin + t * direction;
```

```
    Sample s = sampleVolume( p );
```

```
    Color c = colorMap ( s );
```

```
    float alpha_1msa = c.a * (1 - color.a);
```

```
    accumColor.rgb += c.rgb * alpha_1msa; //blend
```

```
    accumColor.a += alpha_1msa;
```

```
    shade(accumColor); //if you want to add lighting effects
```

3. set the pixel value as that color

The basic algorithm

Given:

float2 pixel.xy, float3 volumeBoxMin, float3 volumeBoxMax

1. generate ray origin, direction from pixel
- 2a. find tStart, tEnd where ray intersects volume box
- 2b. accumulate the color along the samples (step 2b)

3. set the pixel value as that color

```
pixel.color = accumColor;
```


The basic algorithm

Given:

float2 pixel, float3 volumeBoxMin, float3 volumeBoxMax, Volume volume

- 1. generate ray origin, direction from pixel**
- 2a. find tStart, tEnd where ray intersects volume box**
- 2b. accumulate the color along the samples (step 2b)**
- 3. set the pixel value as that color**

Demo — as time permits!

Thanks!

Recommended classes if you like this:

CS5630: Data Visualization

CS5600: Intro to Computer Graphics