Simulation Part 2 + Guest Lecture

Slides by Sarah Jobalia and Kevin Li, with modifications by Kate Baker

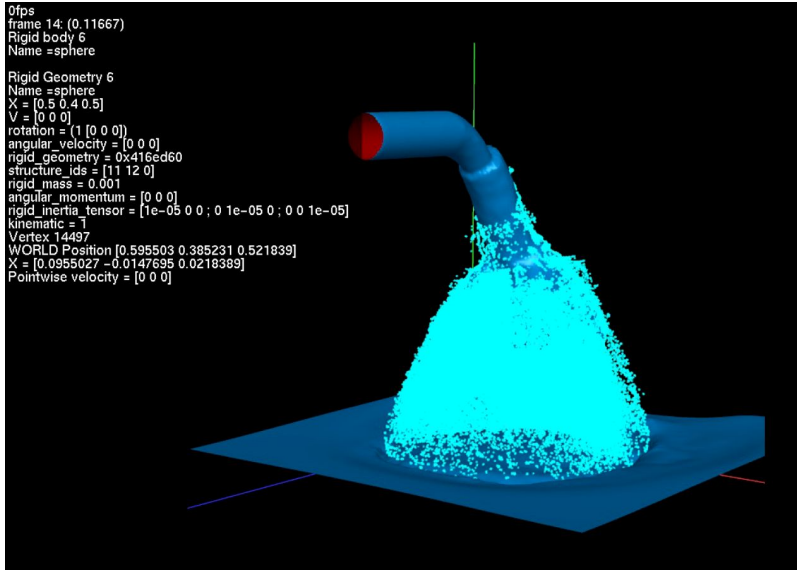
Lecture Outline

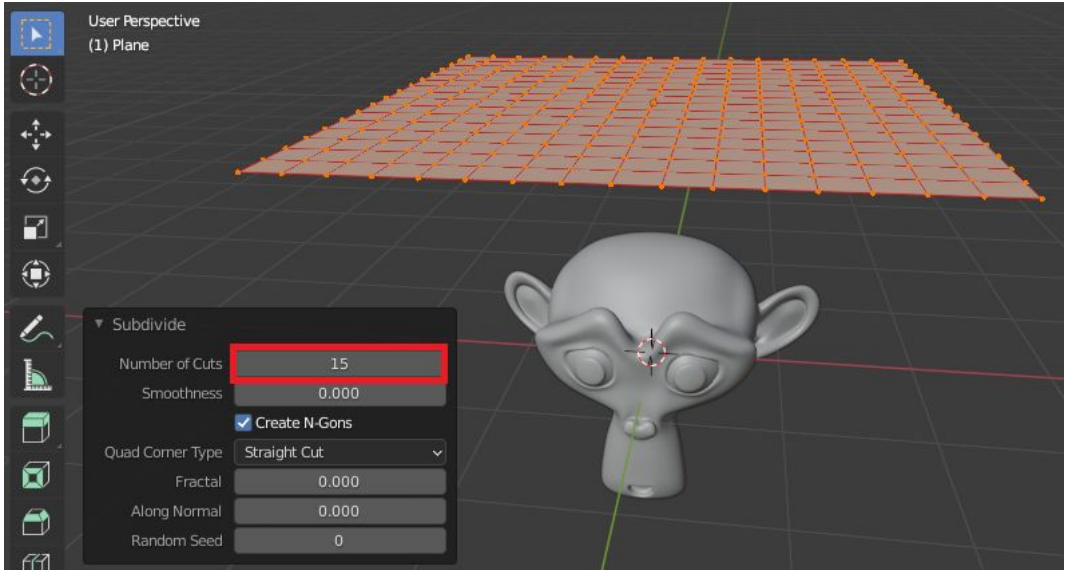
- Participating media simulation (advanced)
- Guest speaker!

Lecture Outline

- Participating media simulation (advanced)
- Guest speaker!

Simulation


0fps
frame 14: (0.11667)
Rigid body 6
Name =sphere
Rigid Geometry 6
Name =sphere
X = [0.5 0.4 0.5]
V = [0 0 0]
rotation = (1 [0 0 0])
angular_velocity = [0 0 0]
rigid_geometry = 0x416ed60
structure_ids = [11 12 0]
rigid_mass = 0.001
angular_momentum = [0 0 0]
rigid_inertia_tensor = [1e-05 0 0 ; 0 1e-05 0 ; 0 0 1e-05]
kinematic = 1
Vertex 14497
WORLD Position [0.595503 0.385231 0.521839]
X = [0.0955027 -0.0147695 0.0218389]
Pointwise velocity = [0 0 0]

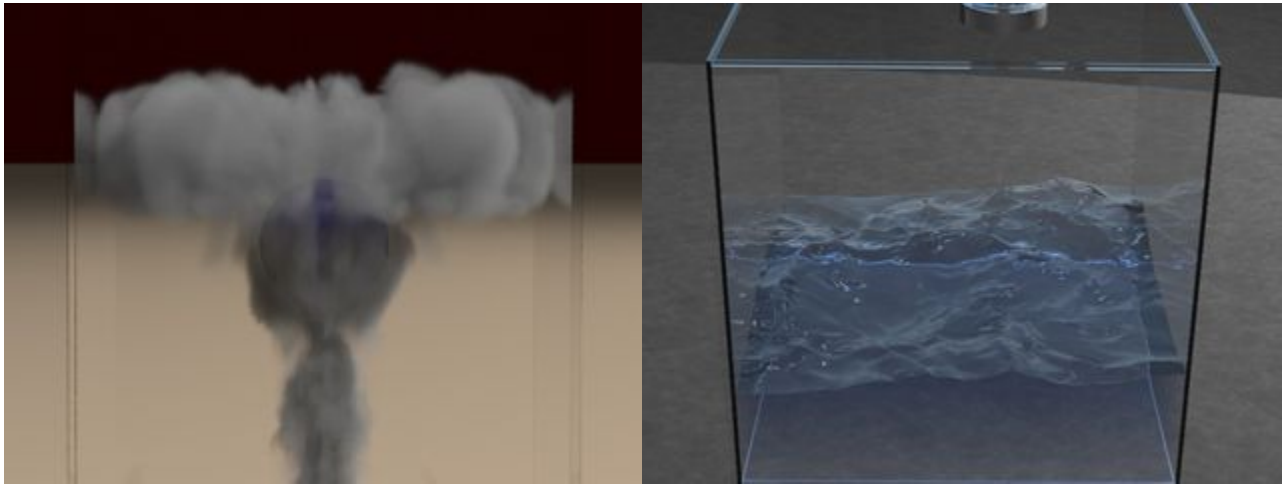

User Perspective
(1) Plane
Subdivide
Number of Cuts
15
Smoothness
0.000
Create N-Gons
Quad Corner Type
Straight Cut
Fractal
0.000
Along Normal
0.000
Random Seed
0

Volumetric Rendering

- How to model **participating media** (dust/fog/clouds/smoke,etc)?

Volumetric Rendering

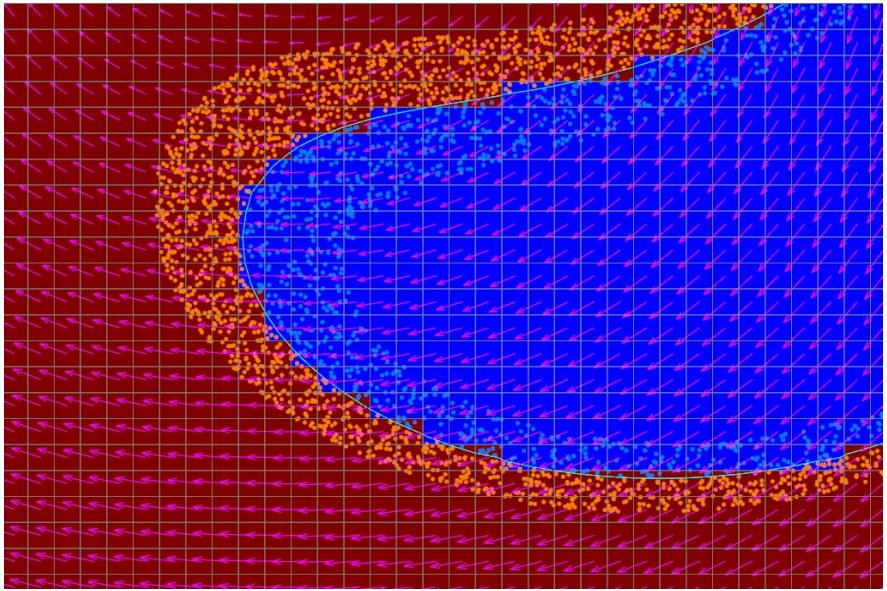
- How to model **participating media** (dust/fog/clouds/smoke,etc)?
- Two part process:
  - **1) Simulate the media** similar to how we similar fluids
  - **2) Ray trace the media** similar to how we handle transmissions



Recall: Fluid Simulation

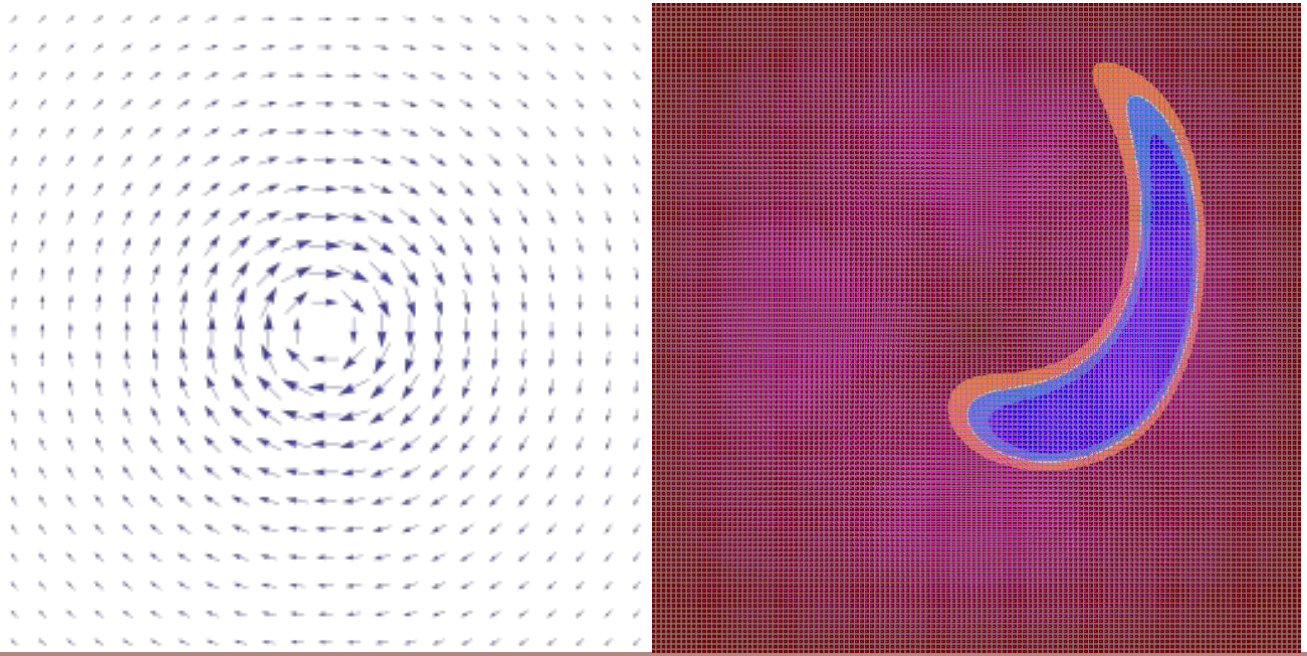
- We track the ~~**particles**~~ **quantities** in space using a fluid domain
- In 2D we use a rectangular grid, in 3D we use a bounding volume

- 2D Example –
- Each grid cell stores velocity vectors along edges
- Essentially, the domain stores a velocity field
- These velocities are used to move the ~~**particles**~~ **quantities** in the cell from state to state



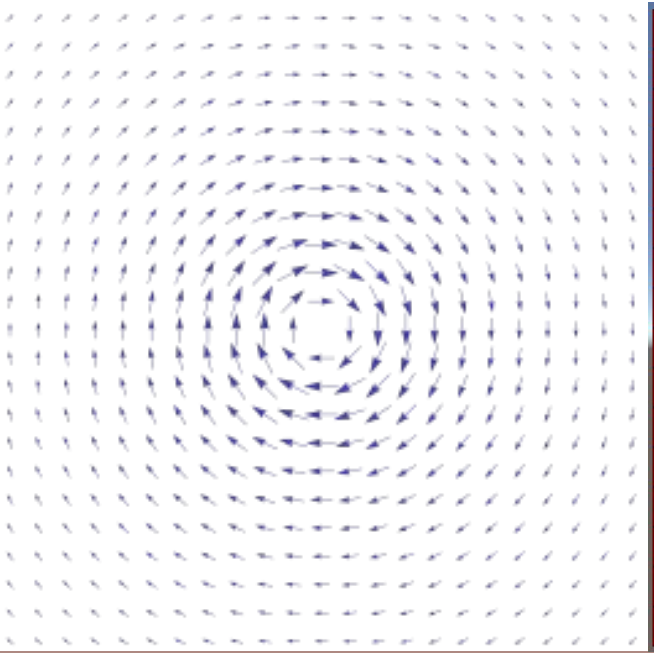
Fields in Calculus and Physics

- Storing quantities on each grid cell gives us a **field**, aka a lookup function that tells us what the quantity is at a point in space



Fields in Calculus and Physics

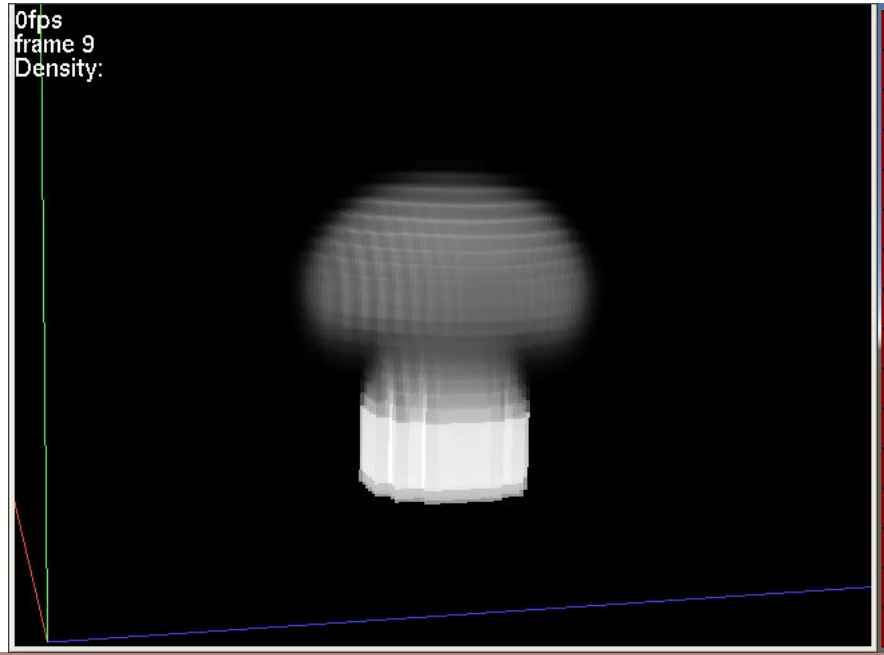
- **Vector fields** store vectors at each grid cell (e.g. velocity fields)
- **Scalar fields** store scalars at each grid cell (e.g. distance fields)



6.6	5.9	5.3	4.7	4.3	3.9	3.6	3.5	3.5	3.6	3.9	4.3	4.7	5.3	5.9	6.6
5.9	5.2	4.5	3.9	3.4	3.0	2.7	2.5	2.5	2.7	3.0	3.4	3.9	4.5	5.2	5.9
5.3	4.5	3.8	3.1	2.5	2.0	1.7	1.5	1.5	1.7	2.0	2.5	3.1	3.8	4.5	5.3
4.7	3.9	3.1	2.4	1.7	1.1	0.7	0.5	0.5	0.7	1.1	1.7	2.4	3.1	3.9	4.7
4.3	3.4	2.5	1.7	0.9	0.3	-0.2	-0.5	-0.5	-0.2	0.3	0.9	1.7	2.5	3.4	4.3
3.9	3.0	2.0	1.1	0.3	-0.5	-1.1	-1.5	-1.5	-1.1	-0.5	0.3	1.1	2.0	3.0	3.9
3.6	2.7	1.7	0.7	-0.2	-1.1	-1.9	-2.4	-2.4	-1.9	-1.1	-0.2	0.7	1.7	2.7	3.6
3.5	2.5	1.5	0.5	-0.5	-1.5	-2.4	-3.3	-3.3	-2.4	-1.5	-0.5	0.5	1.5	2.5	3.5
3.5	2.5	1.5	0.5	-0.5	-1.5	-2.4	-3.3	-3.3	-2.4	-1.5	-0.5	0.5	1.5	2.5	3.5
3.6	2.7	1.7	0.7	-0.2	-1.1	-1.9	-2.4	-2.4	-1.9	-1.1	-0.2	0.7	1.7	2.7	3.6
3.9	3.0	2.0	1.1	0.3	-0.5	-1.1	-1.5	-1.5	-1.1	-0.5	0.3	1.1	2.0	3.0	3.9
4.3	3.4	2.5	1.7	0.9	0.3	-0.2	-0.5	-0.5	-0.2	0.3	0.9	1.7	2.5	3.4	4.3
4.7	3.9	3.1	2.4	1.7	1.1	0.7	0.5	0.5	0.7	1.1	1.7	2.4	3.1	3.9	4.7
5.3	4.5	3.8	3.1	2.5	2.0	1.7	1.5	1.5	1.7	2.0	2.5	3.1	3.8	4.5	5.3
5.9	5.2	4.5	3.9	3.4	3.0	2.7	2.5	2.5	2.7	3.0	3.4	3.9	4.5	5.2	5.9
6.6	5.9	5.3	4.7	4.3	3.9	3.6	3.5	3.5	3.6	3.9	4.3	4.7	5.3	5.9	6.6

Fields in Calculus and Physics

- Participating media like **smoke** and **fog** have **scalar density fields**
- Smoke also involves tracking a **scalar temperature field**



6.6	5.9	5.3	4.7	4.3	3.9	3.6	3.5	3.5	3.6	3.9	4.3	4.7	5.3	5.9	6.6
5.9	5.2	4.5	3.9	3.4	3.0	2.7	2.5	2.5	2.7	3.0	3.4	3.9	4.5	5.2	5.9
5.3	4.5	3.8	3.1	2.5	2.0	1.7	1.5	1.5	1.7	2.0	2.5	3.1	3.8	4.5	5.3
4.7	3.9	3.1	2.4	1.7	1.1	0.7	0.5	0.5	0.7	1.1	1.7	2.4	3.1	3.9	4.7
4.3	3.4	2.5	1.7	0.9	0.3	-0.2	-0.5	-0.5	-0.2	0.3	0.9	1.7	2.5	3.4	4.3
3.9	3.0	2.0	1.1	0.3	-0.5	-1.1	-1.5	-1.5	-1.1	-0.5	0.3	1.1	2.0	3.0	3.9
3.6	2.7	1.7	0.7	-0.2	-1.1	-1.9	-2.4	-2.4	-1.9	-1.1	-0.2	0.7	1.7	2.7	3.6
3.5	2.5	1.5	0.5	-0.5	-1.5	-2.4	-3.3	-3.3	-2.4	-1.5	-0.5	0.5	1.5	2.5	3.5
3.5	2.5	1.5	0.5	-0.5	-1.5	-2.4	-3.3	-3.3	-2.4	-1.5	-0.5	0.5	1.5	2.5	3.5
3.6	2.7	1.7	0.7	-0.2	-1.1	-1.9	-2.4	-2.4	-1.9	-1.1	-0.2	0.7	1.7	2.7	3.6
3.9	3.0	2.0	1.1	0.3	-0.5	-1.1	-1.5	-1.5	-1.1	-0.5	0.3	1.1	2.0	3.0	3.9
4.3	3.4	2.5	1.7	0.9	0.3	-0.2	-0.5	-0.5	-0.2	0.3	0.9	1.7	2.5	3.4	4.3
4.7	3.9	3.1	2.4	1.7	1.1	0.7	0.5	0.5	0.7	1.1	1.7	2.4	3.1	3.9	4.7
5.3	4.5	3.8	3.1	2.5	2.0	1.7	1.5	1.5	1.7	2.0	2.5	3.1	3.8	4.5	5.3
5.9	5.2	4.5	3.9	3.4	3.0	2.7	2.5	2.5	2.7	3.0	3.4	3.9	4.5	5.2	5.9
6.6	5.9	5.3	4.7	4.3	3.9	3.6	3.5	3.5	3.6	3.9	4.3	4.7	5.3	5.9	6.6

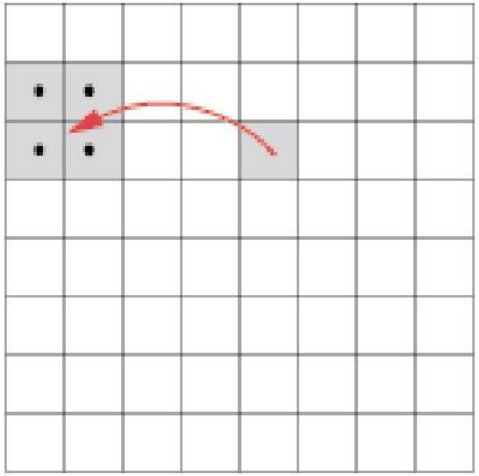
Fields in Calculus and Physics

- The **evolution of a field** over time **due to a velocity field** can be thought of as moving a particle in a cell due to the velocity

- Recall the equation of motion for a point given its initial position and velocity, and a time step size:

$$q(t + \Delta t) = q(t) + (\Delta t)v(t)$$

$$q_{k+1} = q_k + (\Delta t)v_k$$



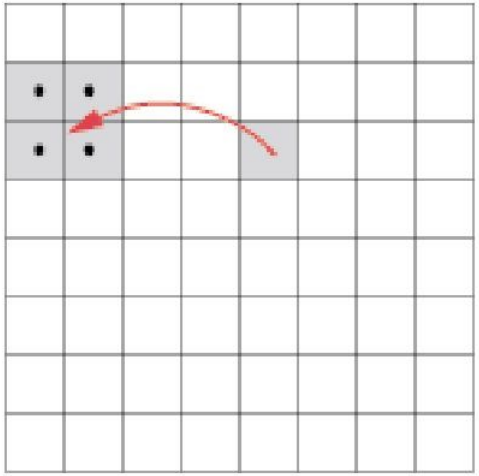
Fields in Calculus and Physics

- The **evolution of a field** over time **due to a velocity field** can be thought of as moving a particle in a cell due to the velocity

- Naive Approach: **Treat the grid cell as if it were a point** via its center, and **compute where it moves** to based on the velocity vector that was stored at the cell:

$$q_{k+1} = q_k + (\Delta t) v_k$$

- After computing where the grid cell moves to, **store its quantity in the new location**



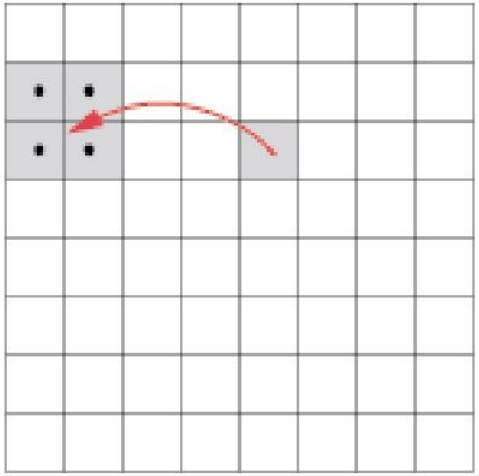
Fields in Calculus and Physics

- The **evolution of a field** over time **due to a velocity field** can be thought of as moving a particle in a cell due to the velocity

- Naive Approach:

$$q_{k+1} = q_k + (\Delta t) v_k$$

- Problems:
  - Uses **Explicit Euler**, which recall can be unstable (“blow up”) for big time steps
  - What if the new location of the cell isn’t exactly a grid cell (e.g. see diagram)?

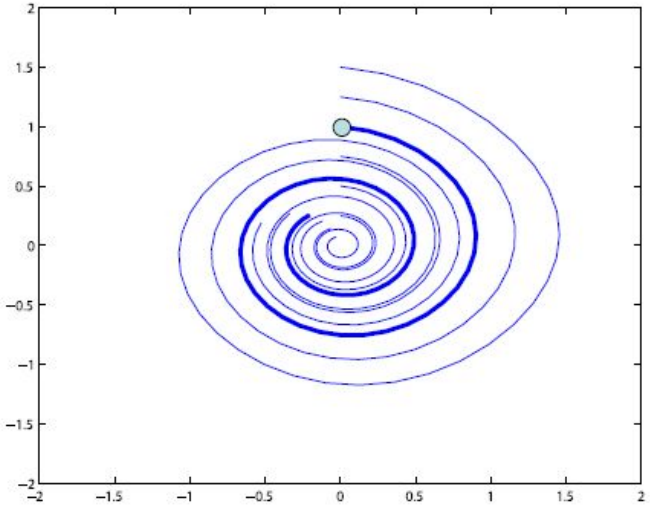


Alternatively: Implicit Euler

- Consider **Implicit Euler** (aka Backward Euler):

$$q_{k+1} = q_k + (\Delta t) v_{k+1}$$

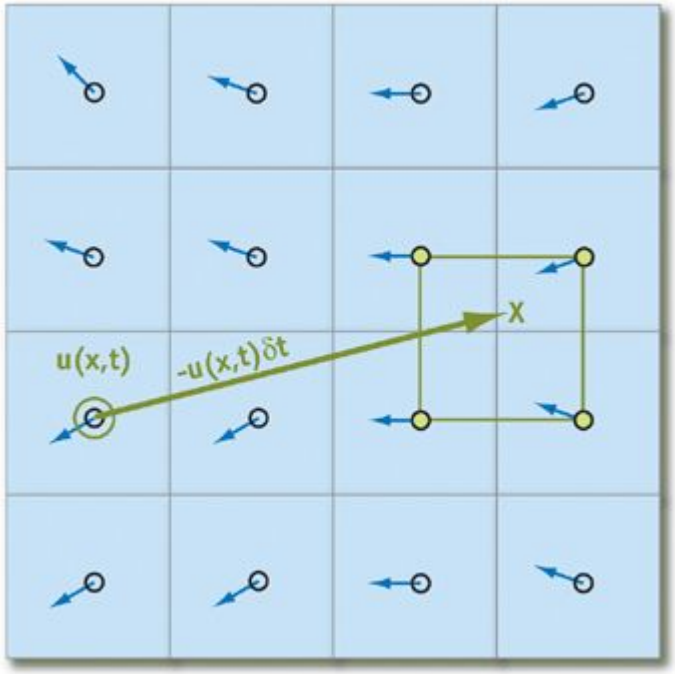
- Can do analysis to see this is stable for ALL time steps!



- **What if we use Implicit Euler for our advection process?**

Implicit Euler for Advection

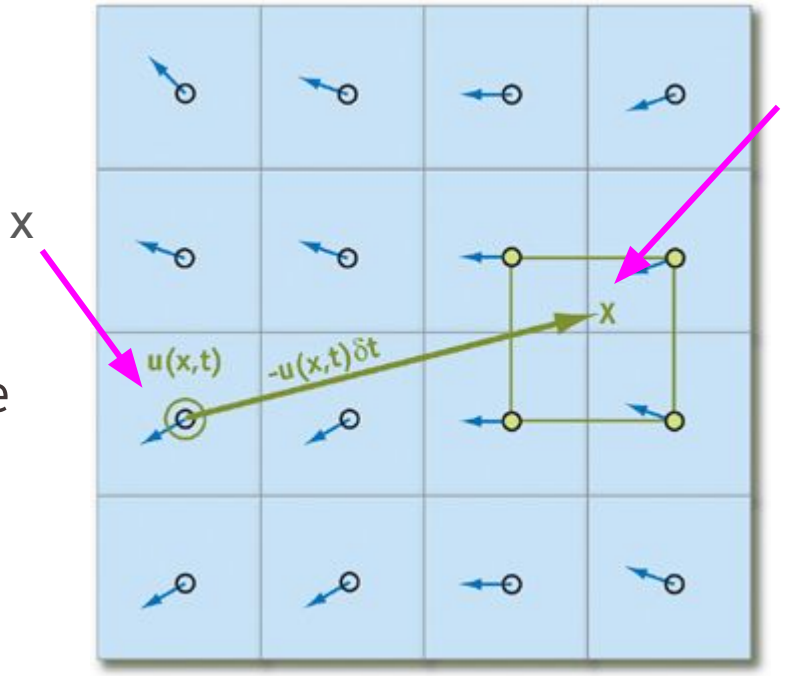
- Use the velocity stored at the grid cell to **trace it “back in time”** to its last position (hence “Backward” Euler)



Implicit Euler for Advection

- Use the velocity stored at the grid cell to **trace it "back in time"** to its last position (hence "Backward" Euler)

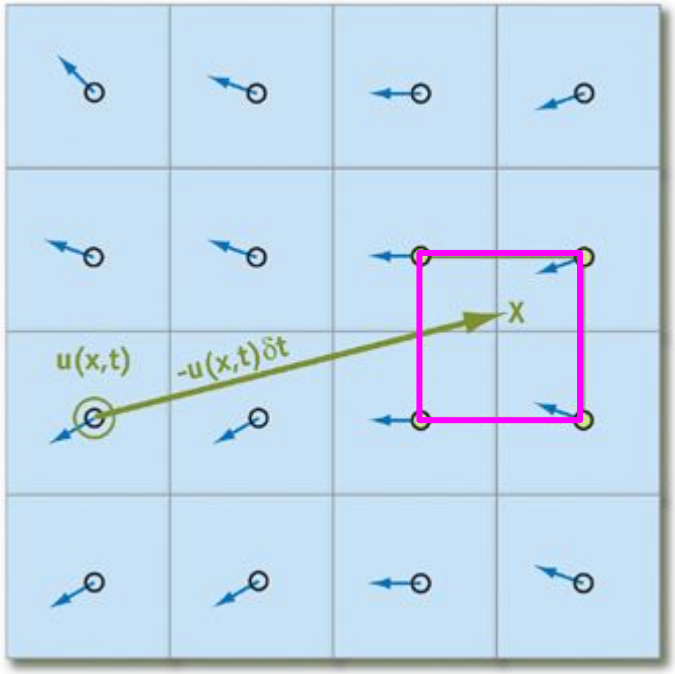
- Implicit Euler Advection Process:
  - Trace grid cell center x back in time to get position X



Implicit Euler for Advection

- Use the velocity stored at the grid cell to **trace it "back in time"** to its last position (hence "Backward" Euler)

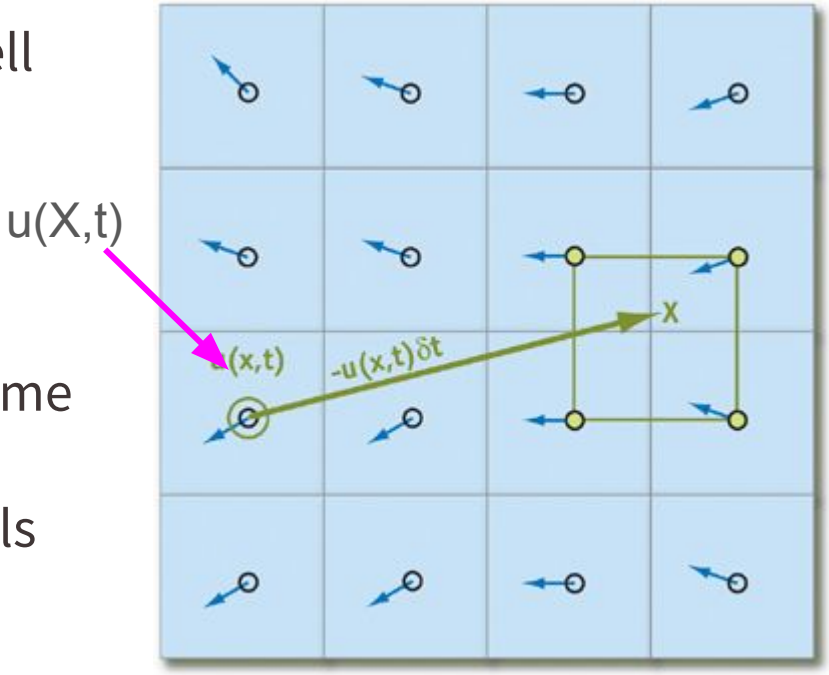
- Implicit Euler Advection Process:
  - Trace grid cell center x back in time to get position X
  - Interpolate surrounding grid cells around X to get quantity at X



Implicit Euler for Advection

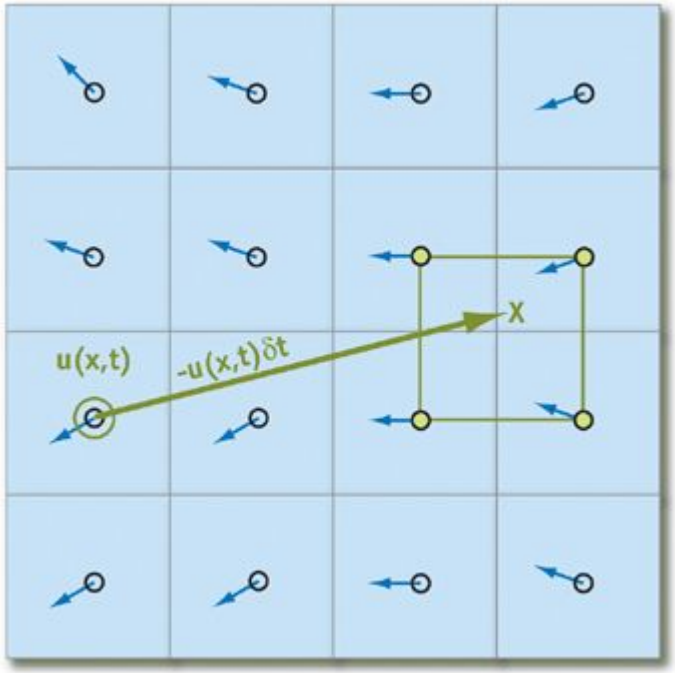
- Use the velocity stored at the grid cell to **trace it "back in time"** to its last position (hence "Backward" Euler)

- Implicit Euler Advection Process:
  - Trace grid cell center x back in time to get position X
  - Interpolate surrounding grid cells around X to get quantity at X
  - Store computed quantity at the grid cell for its new value



Implicit Euler for Advection

- Implicit Euler Advection Process:
  - Trace grid cell center x back in time to get position X
  - Interpolate surrounding grid cells around X to get quantity at X
  - Store computed quantity at the grid cell for its new value
- This **only works with implicit** (not explicit) euler because you already know the past points and can then interpolate



Advecting Velocity with Velocity

- We can advect any field (distance, density, temperature, etc. fields) with the velocity field
- Why not also advect the velocity field with itself?

- Actually done in the **Navier-Stokes equations for incompressible flow** when updating the state of a fluid with velocity field u:

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u - \frac{1}{\rho}\nabla p + \nu \nabla^2 u + F$$

$$\nabla \cdot u = 0$$

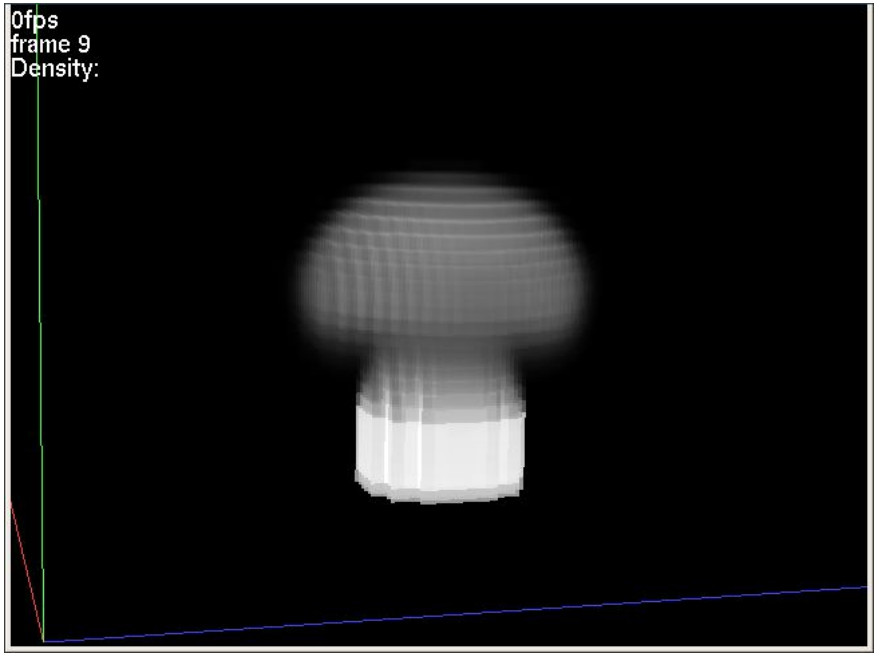
The "Advection Operator"

Can read more on Navier-Stokes from this Nvidia paper

Ray Tracing Density Fields

- Participating media like **smoke** and **fog** have **scalar density fields**

- Step 1: Simulate the media and update its (density) fields

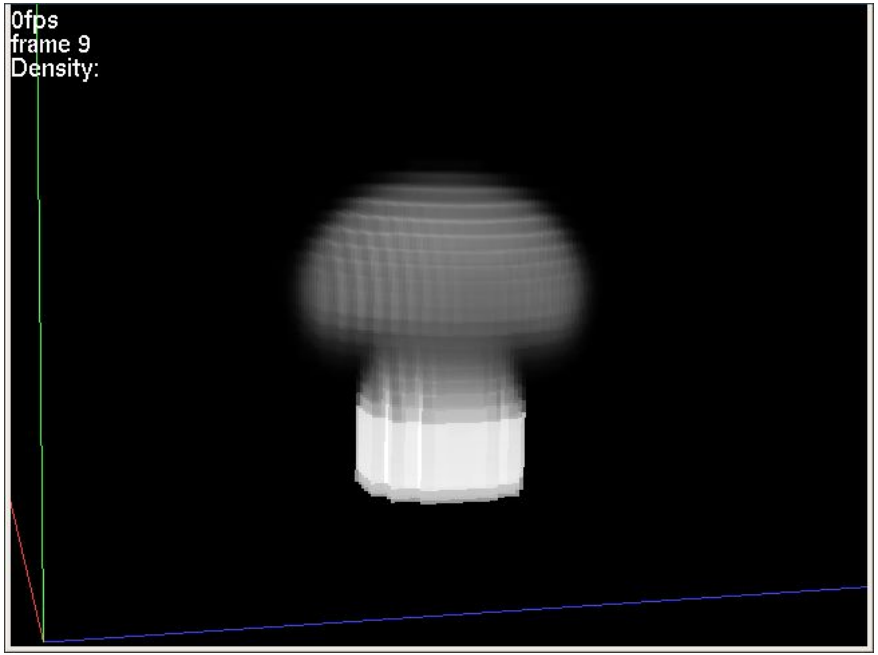


Ray Tracing Density Fields

- Participating media like **smoke** and **fog** have **scalar density fields**

- Step 1: Simulate the media and update its (density) fields

- Step 2: Take into account the density field as we shoot rays through the domain



Absorption

- Step 2: Take into account the density field as we shoot rays through the domain
- As light travels through participating media, it can be **absorbed** and converted into non-visible energy, e.g. heat

Absorption

- Step 2: Take into account the density field as we shoot rays through the domain
- As light travels through participating media, it can be **absorbed** and converted into non-visible energy, e.g. heat

- Computer science parallel: as light moves a distance dx along a ray, a fraction (absorption coefficient σ_a) of its radiance is lost/absorbed:

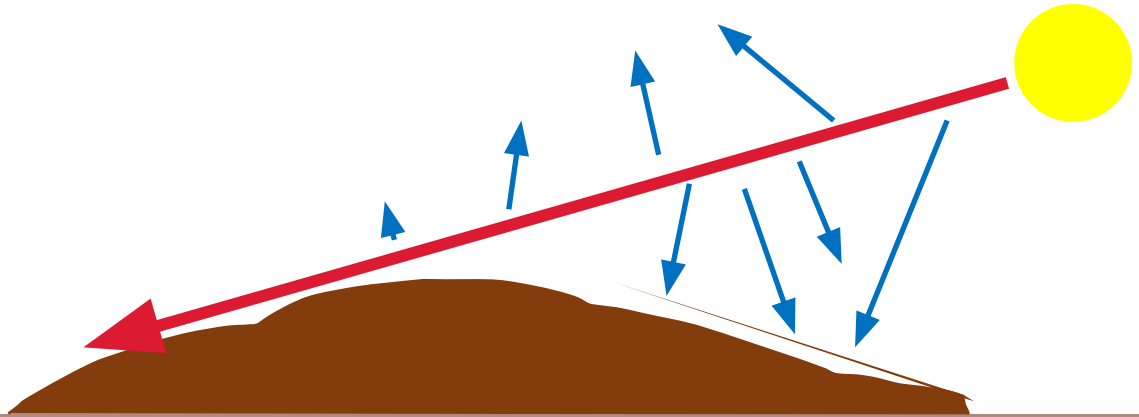
$$dL(x,\omega) = -\sigma_a(x)L(x,\omega)dx$$

Based on Density Field

Out-Scattering

- As light travels through participating media, it can be **out-scattered** to different direction(s)
- Computer science parallel: as light moves a distance dx along a ray, a fraction (scattering coefficient σ_s) of its radiance is lost/scattered

$$dL(x,\omega) = -\sigma_s(x)L(x,\omega)dx$$



Attenuation

- Combine the absorption and scattering coefficients to get the **attenuation coefficient**:

$$c(x) = \sigma_a(x) + \sigma_s(x)$$

$$dL(x, \omega) = -c(x)L(x, \omega)dx$$

Attenuation

- Combine the absorption and scattering coefficients to get the **attenuation coefficient**:

$$c(x) = \sigma_a(x) + \sigma_s(x)$$

$$dL(x, \omega) = -c(x)L(x, \omega)dx$$

- Integrate both sides… we get **Beer's Law**!

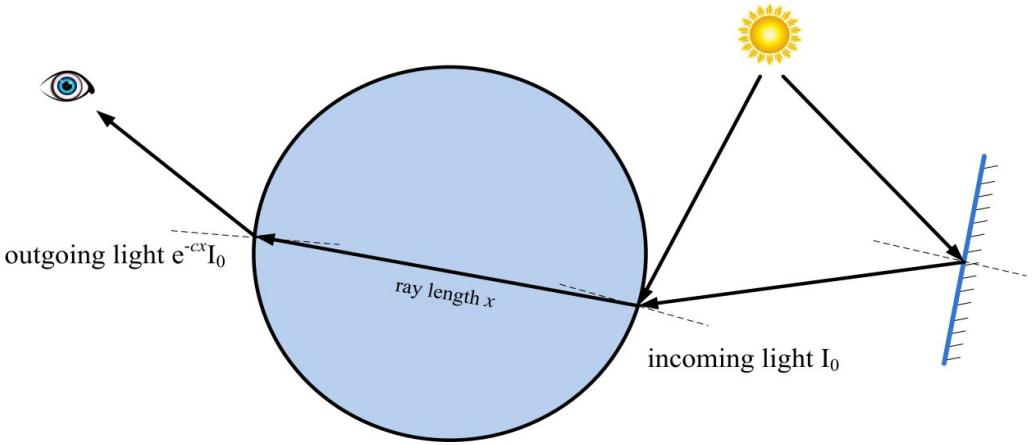
$$L_{atten} = Le^{-cx}$$

Attenuation

- Recall **Beer's Law** tells us how much light is lost as we go through a medium, e.g. going from air through a transparent material

$$L_{atten} = Le^{-cx}$$

- Light enters a medium along a ray and travels through a distance x
- The **falloff** (fraction of light lost) is e^(-cx)



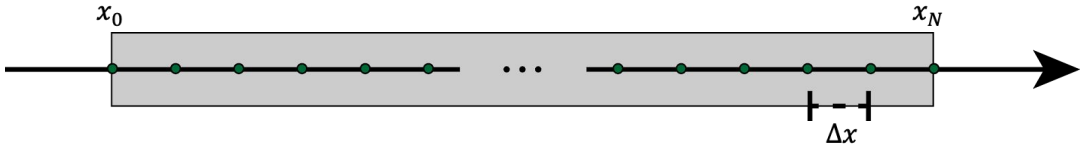
Heterogeneous Beer's Law

- For homogenous media, the attenuation coefficients are constant throughout (e.g. the glass from HW3)
- For participating media (smoke, fog, etc), our density field has changing quantities throughout our grid domain
- Must use **heterogeneous Beer's Law** to compute falloff for light along the ray at every tiny step as we ray trace:

Heterogeneous Beer's Law

- For homogenous media, the attenuation coefficients are constant throughout (e.g. the glass from HW3)
- For participating media (smoke, fog, etc), our density field has changing quantities throughout our grid domain
- Must use **heterogeneous Beer's Law** to compute falloff for light along the ray at every tiny step as we ray trace:

$$Le^{-c(\frac{x_0+x_1}{2})\Delta x}e^{-c(\frac{x_1+x_2}{2})\Delta x}...e^{-c(\frac{x_{N-1}+x_N}{2})\Delta x}$$



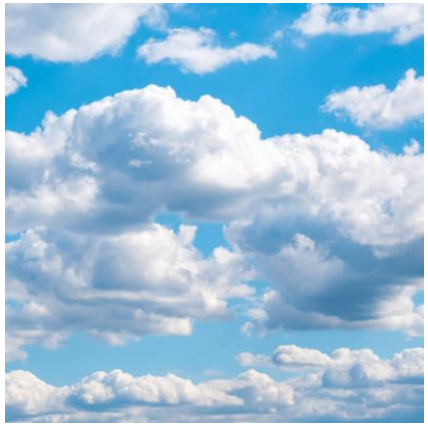
Attenuation of Shadow and Camera Rays

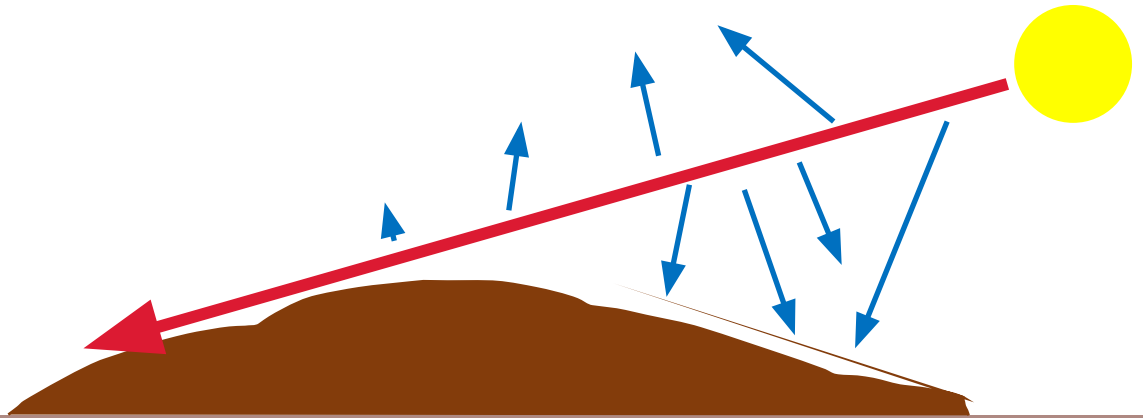
- Attenuation causes participating media like smoke to cast a shadow
- Light gets lost as it goes through the smoke density field, becoming darker

- Attenuation also obscures objects in smoke due to lost light

In-Scattering

- As light travels through participating media, it can be **out-scattered** to different direction(s)
- The out-scattered light can be **in-scattered** to our eyes/camera
- Example: the blue components of sunlight out-scattered by the atmosphere become in-scattered into our eye to make the sky blue

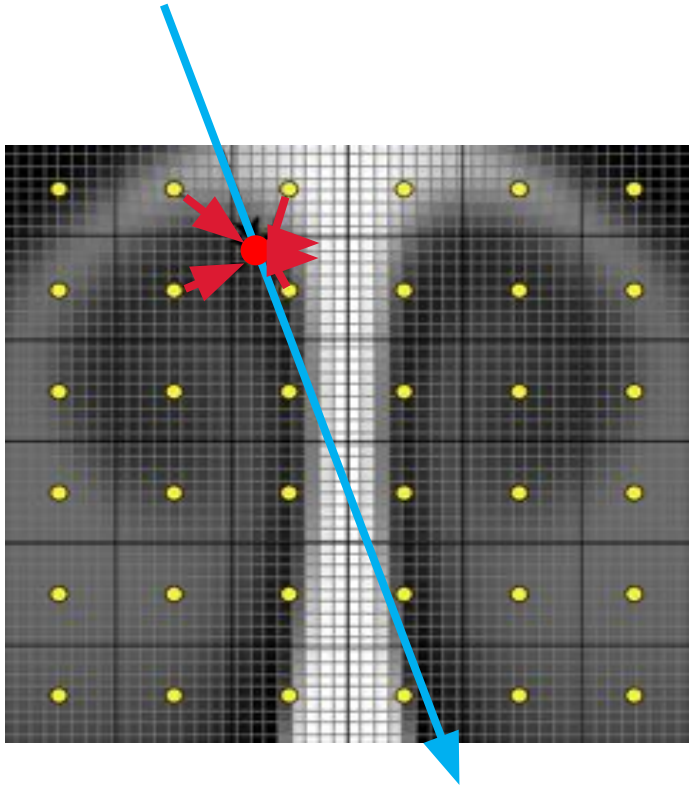




Volumetric Light Map

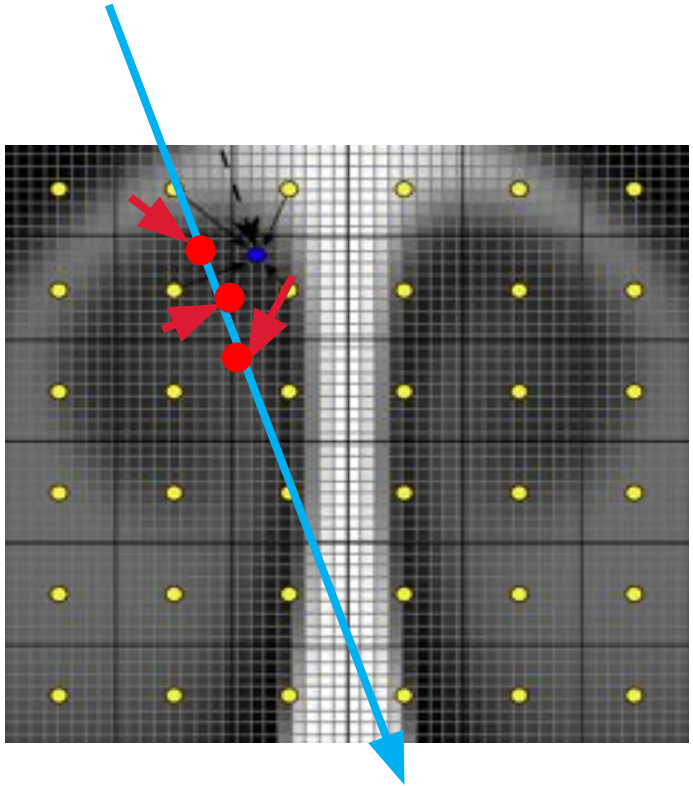
- Model in-scattering when ray tracing with a **volumetric light map**
- **Precompute** one **for each light**

- For each volumetric light map:
  - At each grid cell of our domain, cast a ray to the light
  - Compute how much light gets to that grid cell through the density field
  - Store that radiance at the cell



Volumetric Light Map

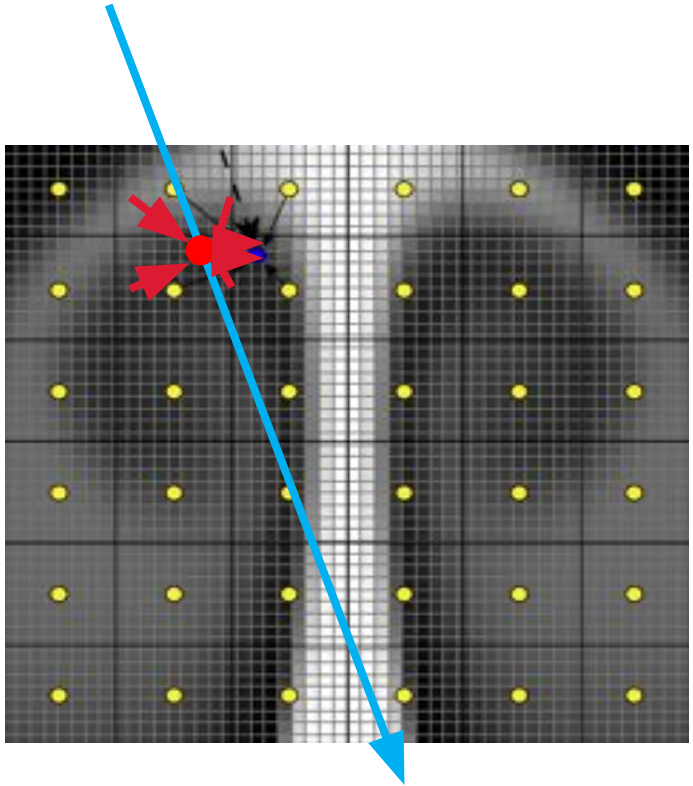
- As we ray trace, interpolate radiance values in nearby grid cells to see how much light is available for in-scattering



Volumetric Light Map

- As we ray trace, interpolate radiance values in nearby grid cells to see how much light is available for in-scattering

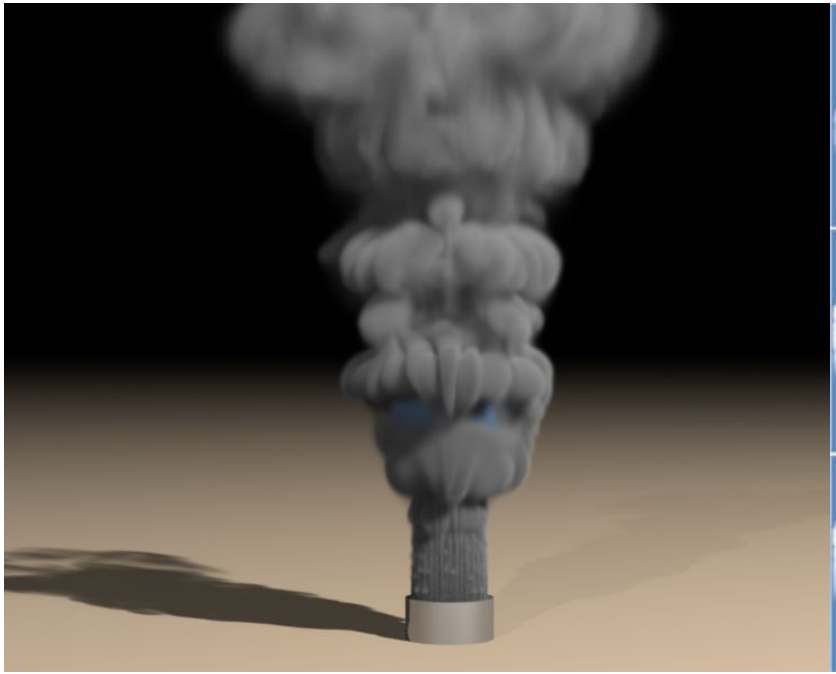
- Use a **phase function** that depends on the direction of our ray and the light to compute the probability of adding the interpolated light as in-scattered light

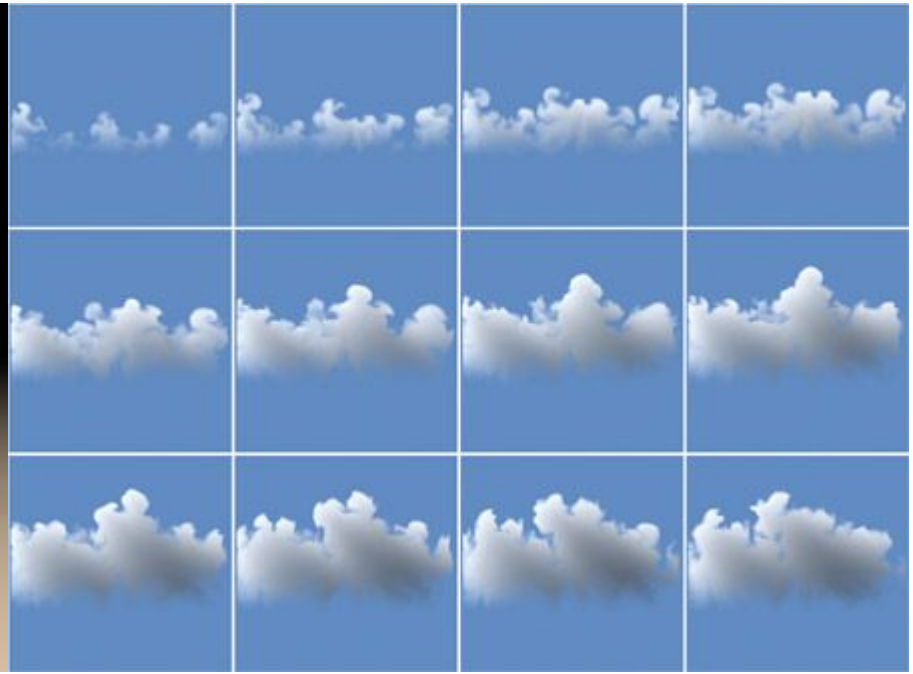


Phase Functions

- As we ray trace, interpolate radiance values in nearby grid cells to see how much light is available for in-scattering
- Use a **phase function** that depends on the direction of our ray and the light to compute the probability of adding the interpolated light as in-scattered light
- Example phase functions of angle between our ray and the light:
  - Rayleigh (for atmosphere): $$p(\theta) = \frac{3}{8}(1 + \cos^2\theta)$$
  - Henyey-Greenstein (more general): $$p(\theta) = \frac{\frac{1}{4\pi}(1 - g^2)}{(1 + g^2 - 2g\cos\theta)^{1.5}}$$

Volumetric Rendering





Lecture Outline

- Participating media simulation (advanced)
- Guest speaker!

GUEST LECTURE

Scientific visualization.

Turning simulation data into something you can see.

Matt Larsen · Luminary

Introduction to Computer Graphics

01

What scientific visualization is, and who needs it.

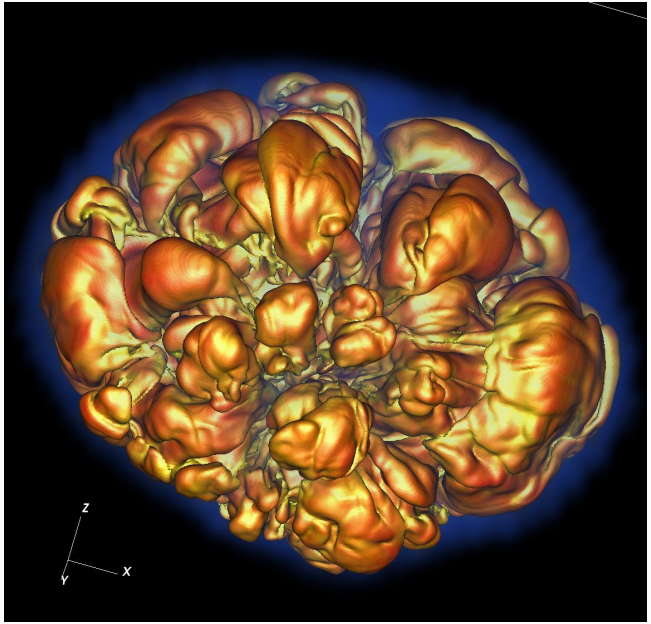
Simulation produces numbers. People need pictures.

- An interdisciplinary corner of computer graphics, aimed at 3D physical phenomena.
- Input is a discretized field over a mesh — not triangles handed to you by an artist.

– Scalars: temperature, pressure, density

– Vectors: velocity, momentum, magnetic field

- Considered a branch of computer science that is a subset of computer graphics

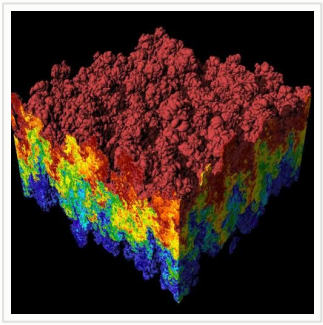


Isosurface, temperature field, supernova simulation

Three reasons anyone opens a visualization tool.

Debugging

Something in the run is wrong. The picture tells you where, and usually faster than the log does.

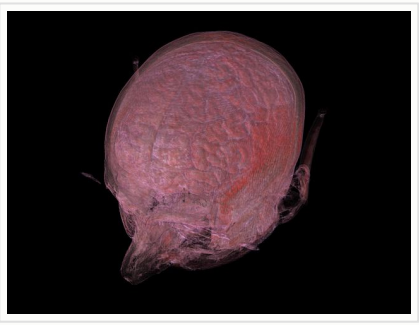


Analysis

The run is fine and now you need a number, a trend, a mechanism you can defend.

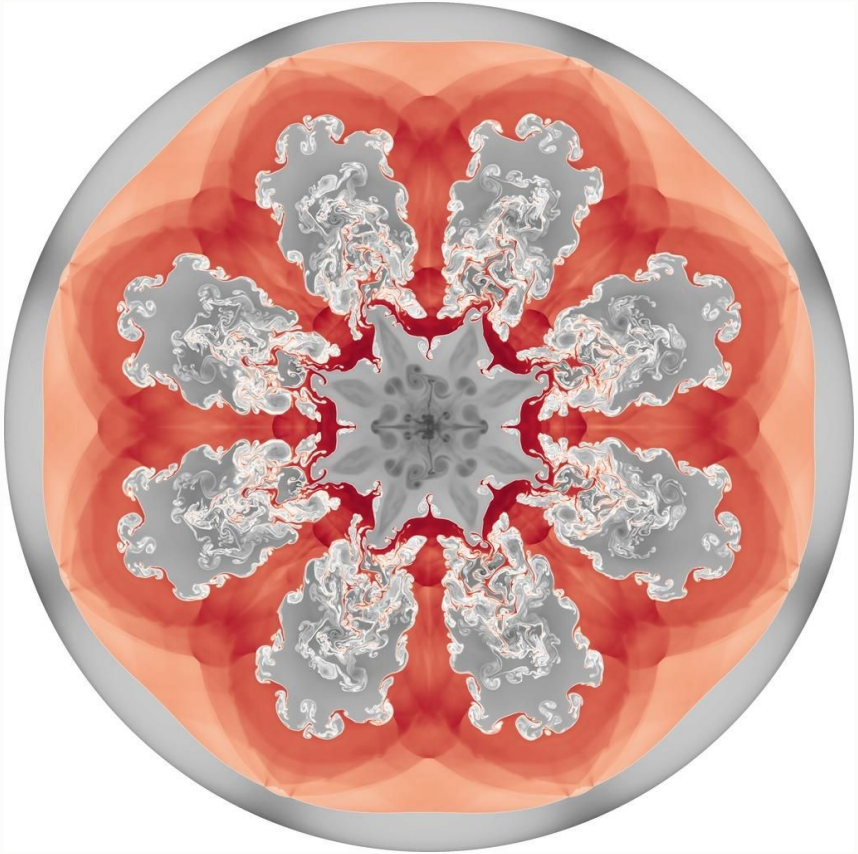
Communication

A reviewer, a customer, or a program manager has to believe the result.



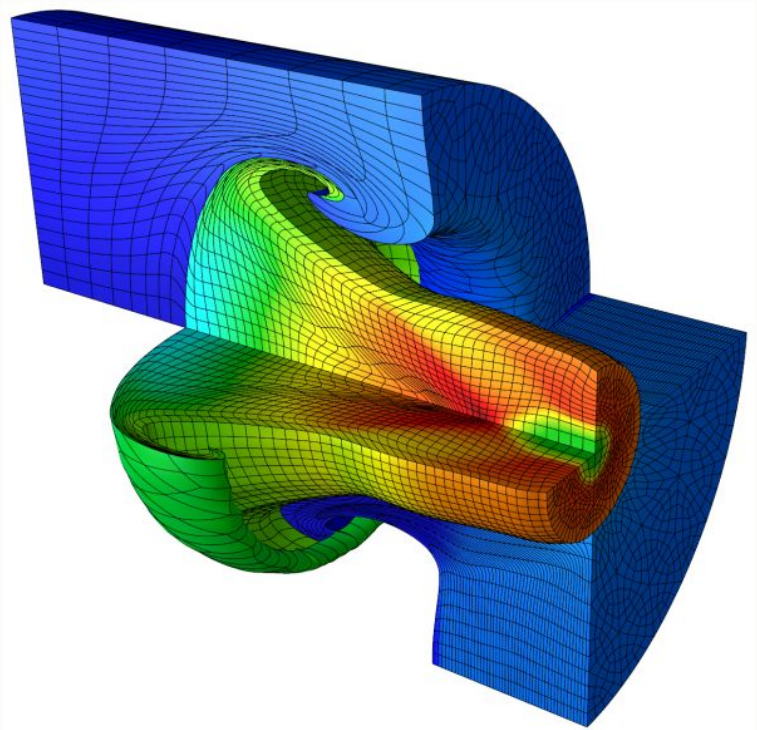
Who actually does this?

- National labs (DOE)
  - NNSA: LLNL, LANL, SNL
  - Office of Science: LBL, ANL,...
- Industry
  - Any sort of product development
- Academia
  - Research



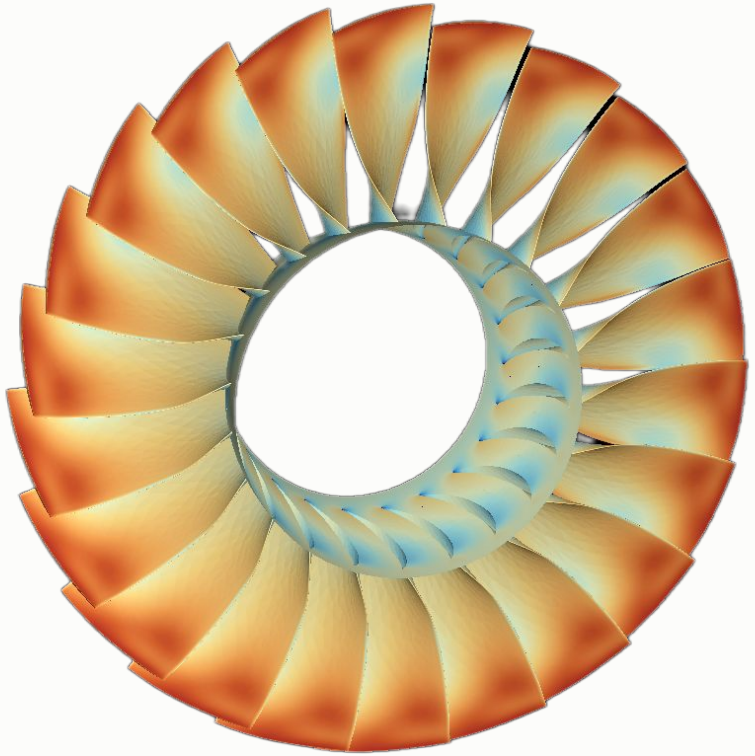
Almost every physical science has a simulation code.

- Hydrodynamics (many flavors), CFD, N-body
- Seismic, radiation transport, finite element
- Particle accelerators, climate, geology
- Astrophysics, materials science, molecular dynamics, …



Even CFD alone has many applications

- Aerospace
- Automotive
- Turbomachinery
- Defense
- Dishwashers
- Pizza ovens



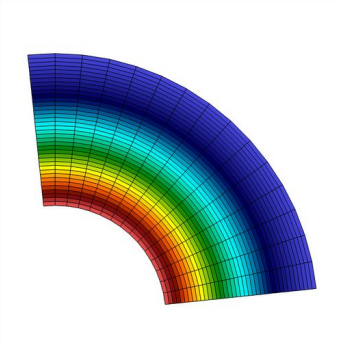
The mesh is the data structure: regular grids

- Uniform — spacing is constant; store an origin and a delta.
- Rectilinear — per-axis coordinate arrays; vertices are the cross product.

– X = {0,1,2,3}, Y = {2,3,5,6}

– Cheap to store, trivial to index, and extremely common

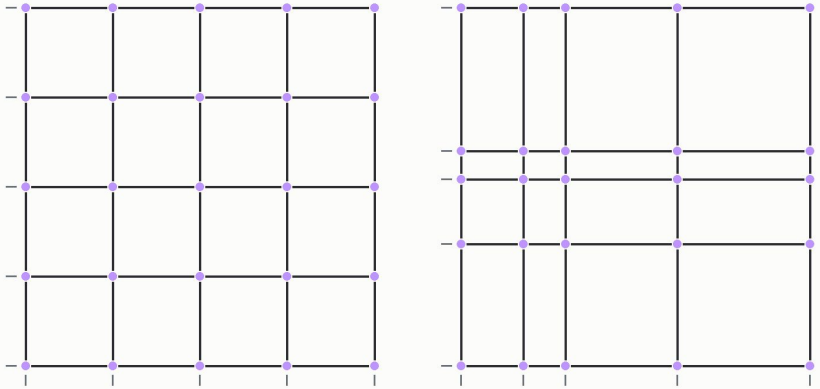
- Curvilinear — unstructured coordinates.



CURVILINEAR

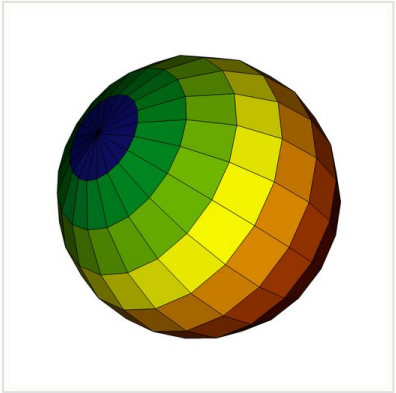
UNIFORM

RECTILINEAR

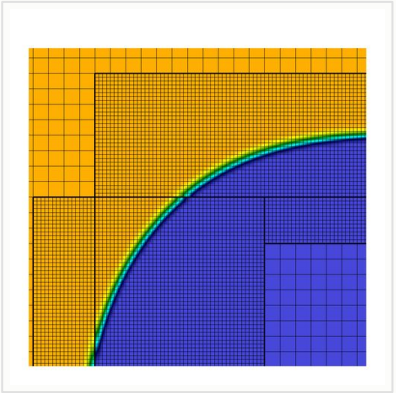


same vertex count — only the spacing rule differs

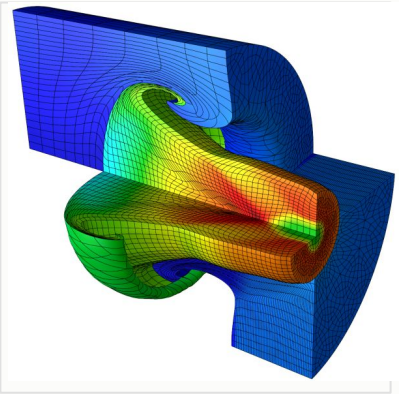
More advanced mesh flavors



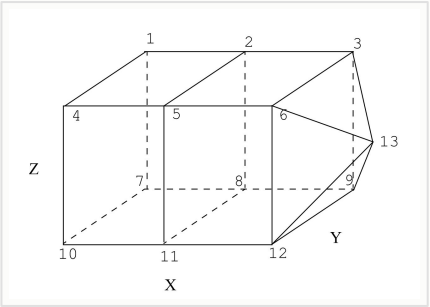
UNSTRUCTURED / POLYHEDRAL



ADAPTIVE MESH REFINEMENT



High Order Mesh



CELL CONNECTIVITY

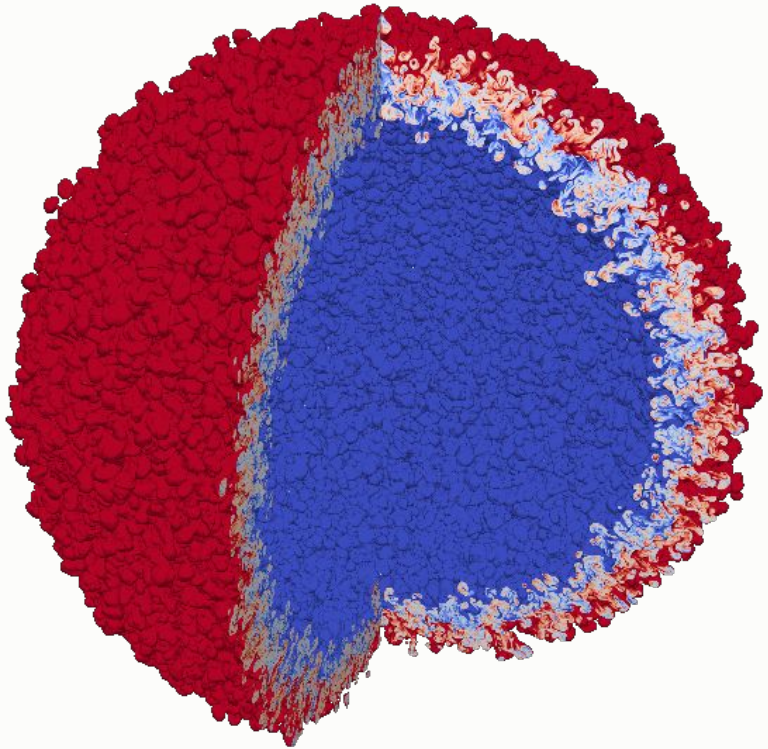
- Explicit connectivity: a point list plus a cell list. Flexible, and far more expensive per cell.
- High order — cells carry curved basis functions, so a "linear" renderer quietly gets it wrong.

How large is the data?

- Day-to-day workloads: 1–64 nodes. This is most of the actual science.
- Hero runs: the whole machine, scheduled months ahead, run once.
- Either way the data is distributed, so the visualization is too — MPI, one rank per piece.

– Every algorithm that follows has to work without seeing the whole mesh

- Data size is challenging
- Data partitioning is not great for visualization



The Top 500

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**LineShine** - LingKun, LX2 304C 1.55GHz, LingQi, Kylin OS, Shenzhen Cloud Computing Center Co., Ltd. National Supercomputing Centre in Shenzhen (NSCS) China	13,789,440	2,198.40	2,735.82	42,220
2	**El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	11,340,000	1,809.00	2,821.10	29,685
3	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607
4	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
5	**JUPITER Booster** - BullSequana XH3000, GH Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, RedHat Enterprise Linux, Bull EuroHPC/FZJ Germany	4,801,344	1,000.00	1,226.28	15,794

02

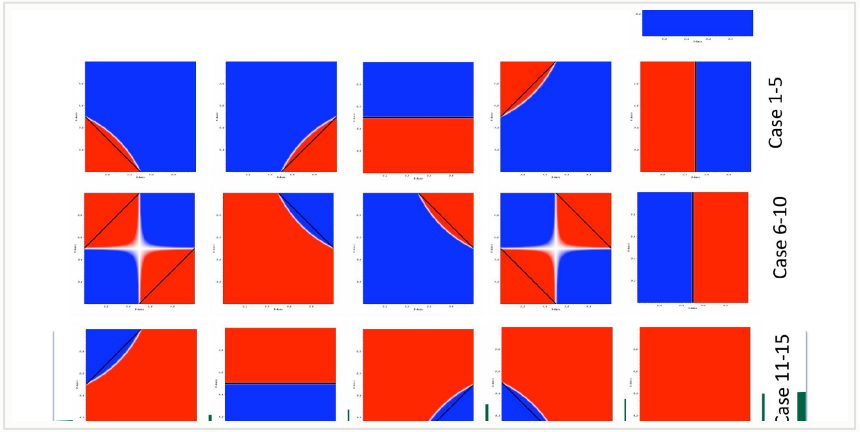
A short tour of the algorithm zoo.

Isosurfacing: draw where the field equals a value.

- Pick an isovalue. Find the surface where F = that value.
- Per cell, the sign pattern at the corners determines the topology.

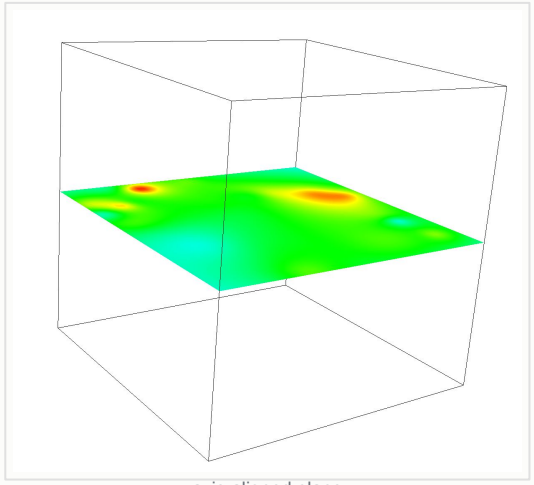
– 2D: 16 cases. 3D: 256, reduced to 15 by symmetry

- Precompute every case into a lookup table, then it is one table read per cell.
- Marching cubes, 1987 — still the workhorse.

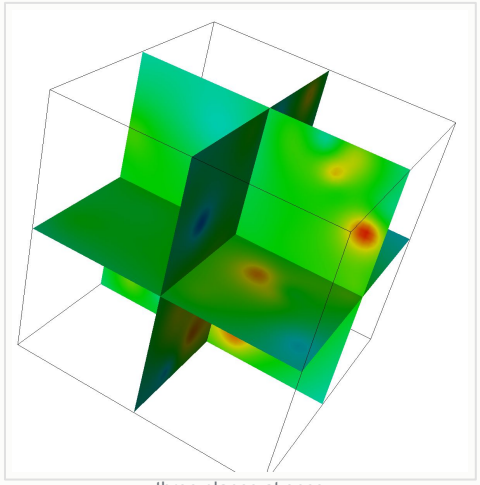


The 16 two-dimensional cases

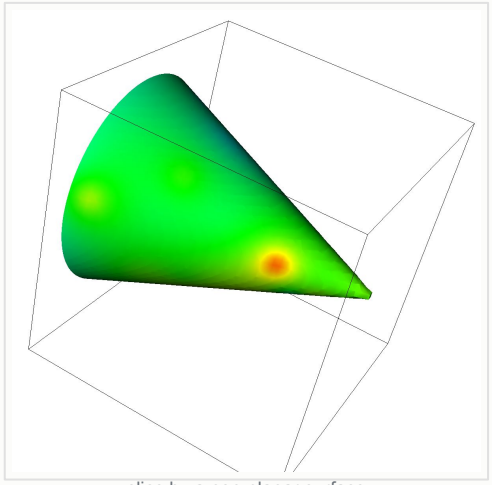
Slicing: trade a dimension for clarity.



axis-aligned plane



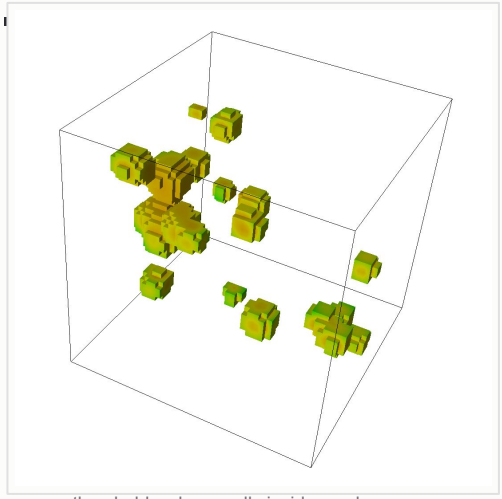
three planes at once



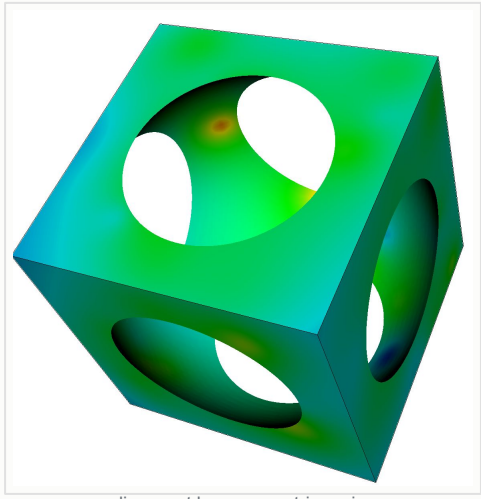
slice by a non-planar surface

Implemented with a signed distance function: evaluate it at every point, then contour it at zero. Which means slicing is isosurfacing wearing a different hat.

Thresholding and clipping: throw away what you do not need.



threshold — keep cells inside a value range



clip — cut by a geometric region

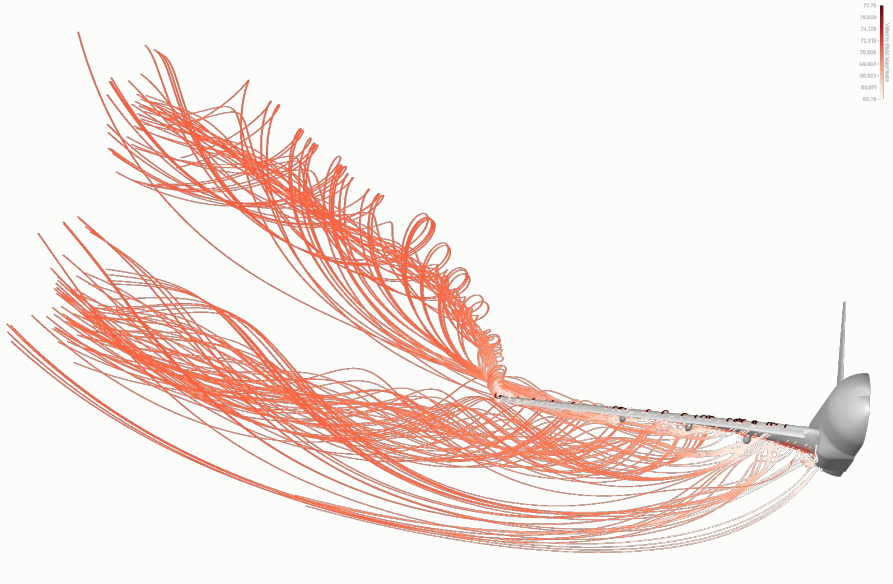
Threshold is a query on values; clip is a query on space. Both shrink the problem before the expensive stage.

Streamlines: follow the velocity field.

- Integrate a massless particle through the vector field.

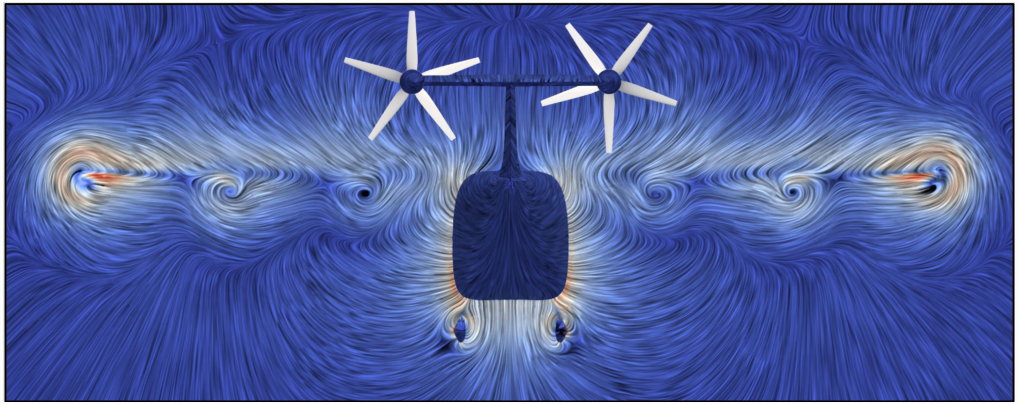
– Euler is cheap and wrong; RK4 is the default

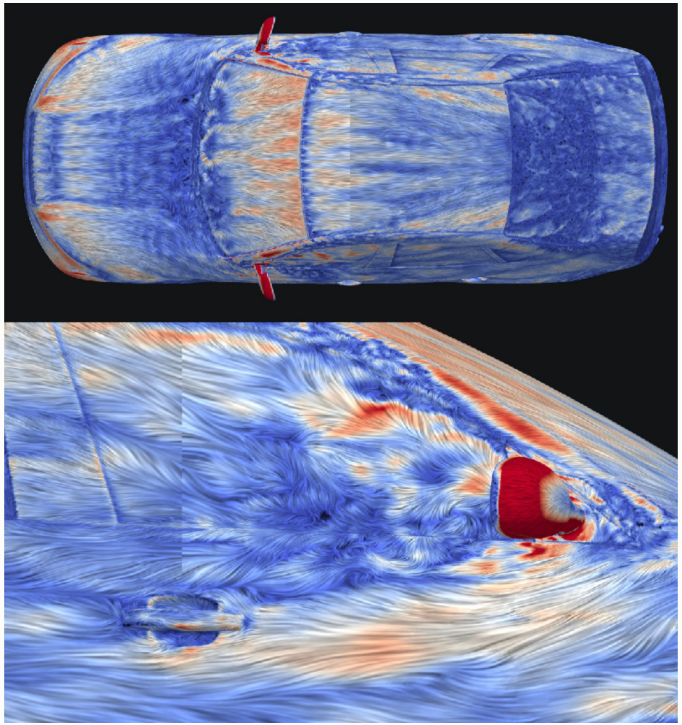
- Steady field → streamline. Time-varying field → pathline.
- Seed placement is the whole game: bad seeds, useless picture.
- Sweep a seed curve instead of a point and you get a stream surface.



Line integral convolution.

- Smear a noise texture along the local flow direction.
- Shows direction everywhere on a surface at once — no seeds to place.
- Cheap enough to run as a fragment shader, but that depends on image resolution



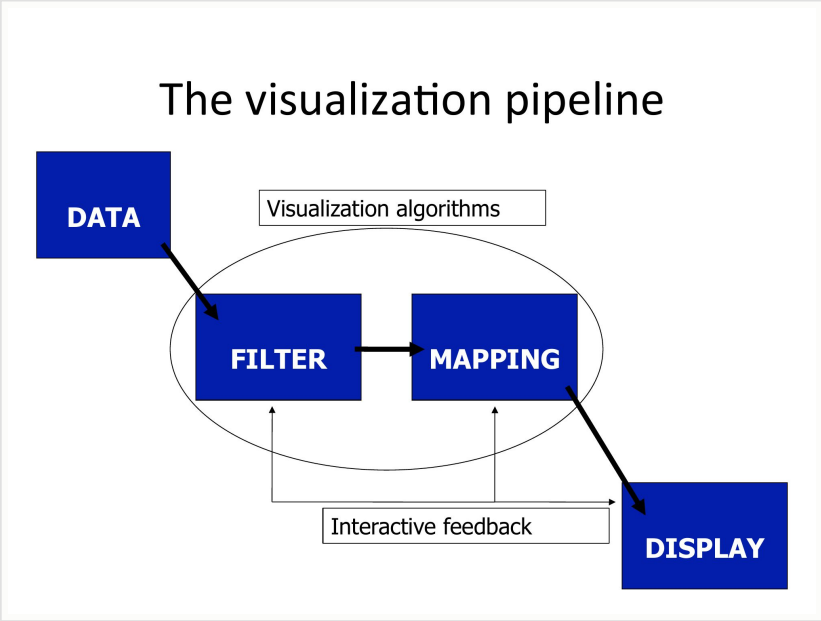


03

How visualization systems are put together.

Sources, filters, sinks.

- Source — produces data: a file reader, or a synthetic generator.
- Filter — takes data in, gives data out: slice, threshold, contour.
- Sink — consumes and does not return: a renderer, a writer.
- Execution is demand-driven: the sink pulls, an update pass walks upstream, then data flows down.

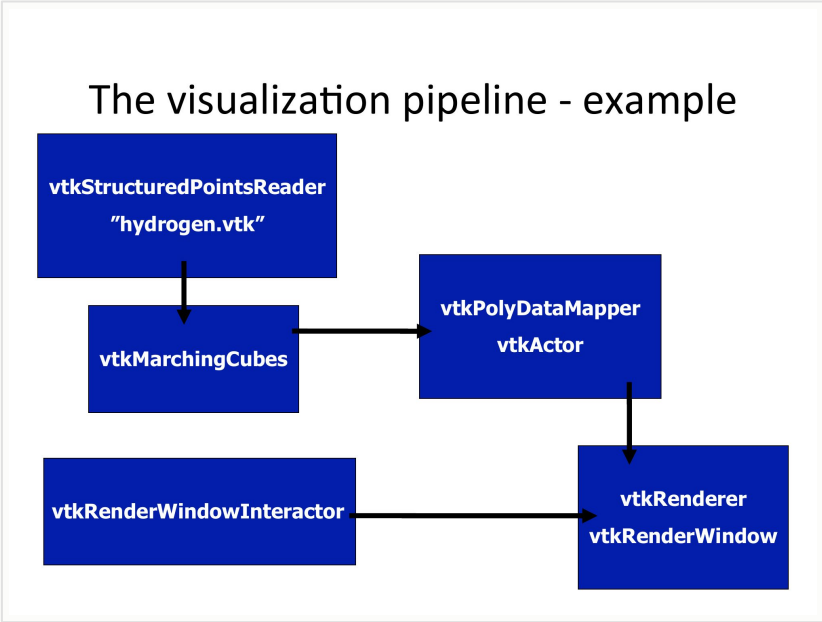


the canonical visualization pipeline

It is a graph, not a line.

- Filters fan out and rejoin — a DAG, not a chain.
- One read feeding a contour, a slice, and an outline is three branches on one source.
- Buys you caching and lazy re-execution; costs you memory.

– Reference counting keeps intermediate results from piling up



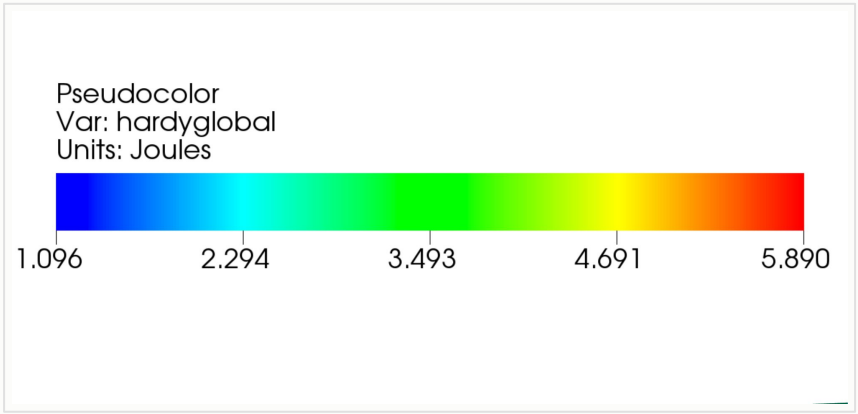
a concrete filter DAG

04

Color Maps

A color map is a function from scalar to color.

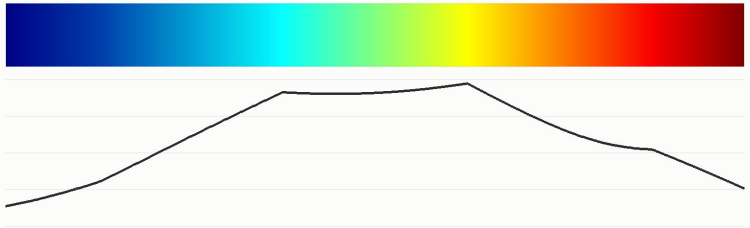
- Pick a range, map every value into it, draw a legend.
- The legend is not optional — without it the image is an illustration, not a measurement.
- Clamp or not at the ends? That decision changes what the reader concludes.



scalar field with legend

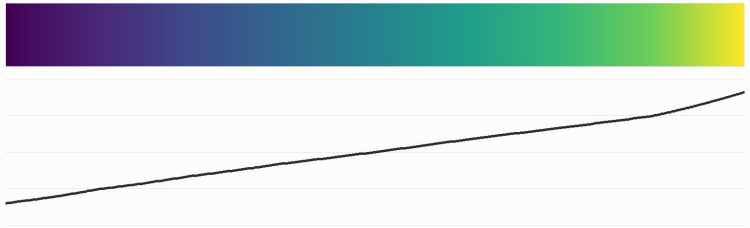
Rainbow is the default, and the default is wrong.

RAINBOW / JET



lightness L* along the ramp

PERCEPTUALLY LINEAR



lightness L* along the ramp

- Jet reverses perceptual lightness 114 times across 255 steps. A linear ramp reverses 6.
- So it invents edges where the data is smooth, and hides real gradients in the green.

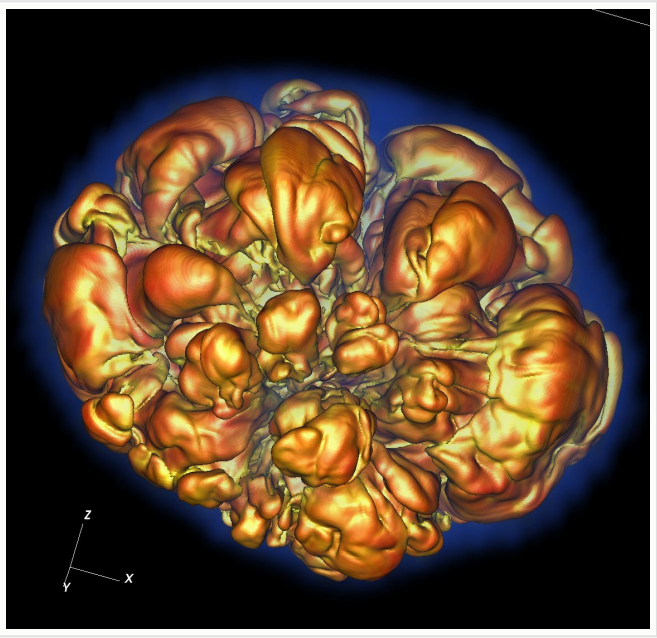
Borland & Taylor, "Rainbow Color Map (Still) Considered Harmful", IEEE CG&A 27(2), 2007

05

Rendering: surfaces, and then volumes.

Two ways to turn a field into pixels.

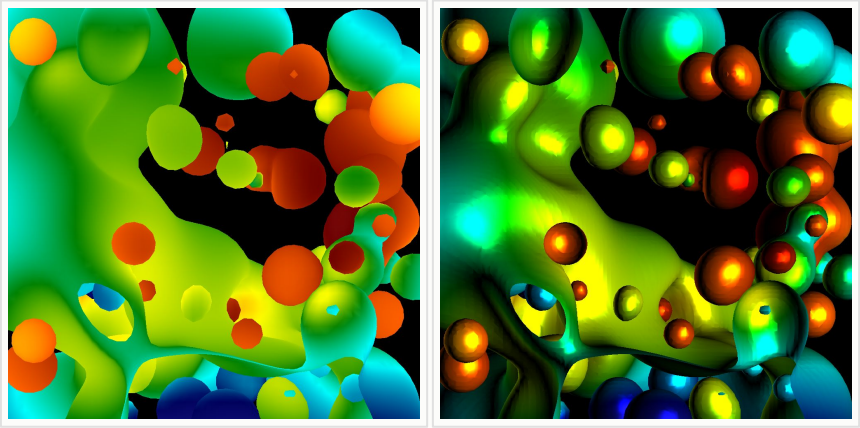
- Surface rendering — extract geometry first (external faces), then render it. This is ordinary graphics, and the GPU already likes it.
- Volume rendering — never build geometry. Integrate along rays through the field itself.
- Surfaces answer "where is the boundary". Volumes answer "what is inside".



both at once: surface plus volume

Surface shading is the part you already know.

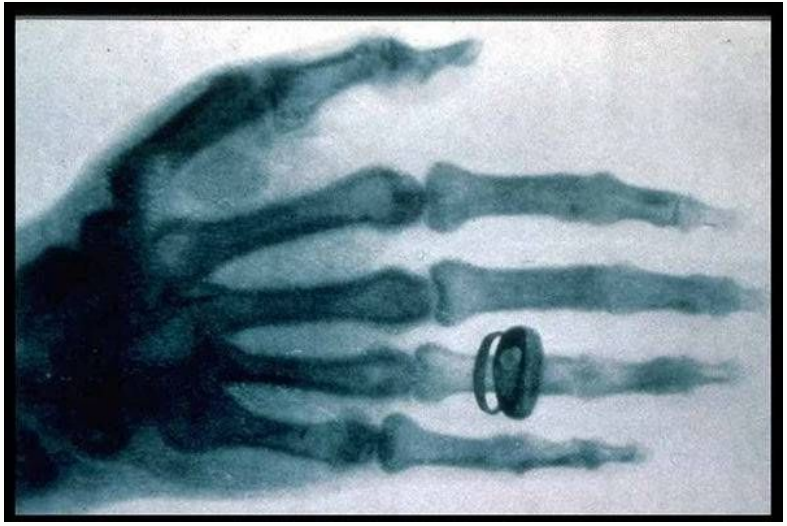
- Triangles come out of marching cubes with a normal per vertex.
- From there it is the lighting model from your graphics course.
- LIC drops in as a fragment shader over the extracted surface.



normals and reflection on extracted geometry

Volume rendering starts with an X-ray.

- An X-ray is a line integral through a density field, printed on film.
- No surface was extracted. Every point along the ray contributed.
- That is the whole idea — now do it with a transfer function and a camera you control.



Röntgen-era radiograph

Ray casting, end to end.

- For every pixel on the screen:

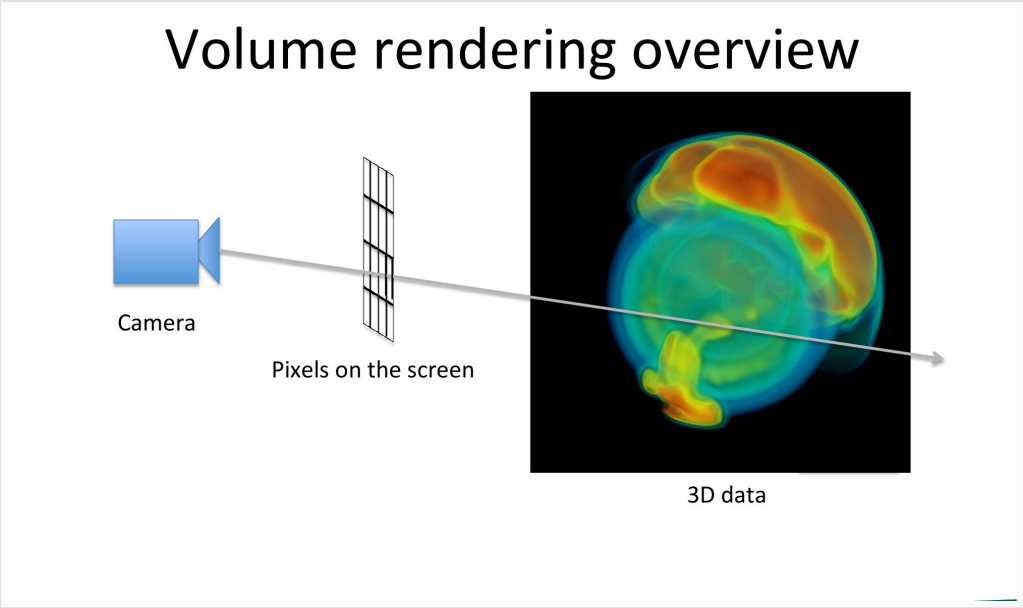
– find the ray for that pixel

– intersect the volume with the ray

– compute a color from the intersection

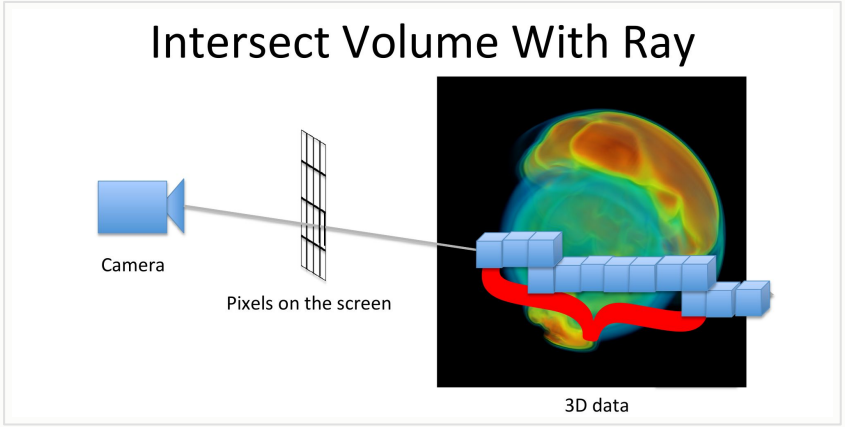
– assign that color to the pixel

- Steps 1 and 4 are easy. Steps 2 and 3 are the lecture.

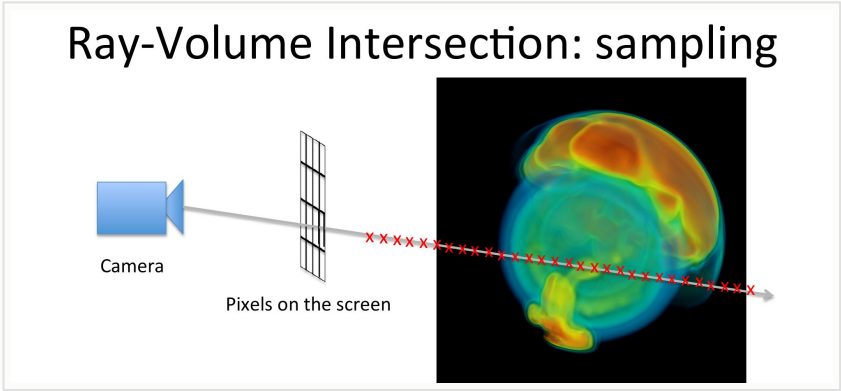


camera, image plane, volume

Intersect, then sample along the ray.



the ray crosses a run of cells

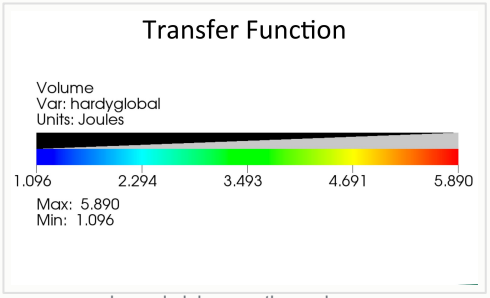


sample at a fixed step

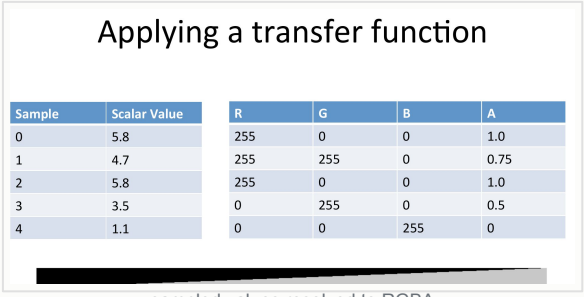
Sampling turns an integral into a sum. Step size is the quality-versus-cost dial, and picking it badly is the most common way a volume rendering lies to you.

The transfer function assigns color and opacity.

- Pseudocoloring needs a color per value. Volume rendering also needs an opacity per value.
- Opacity is what decides which material you can see through.
- It is the primary interaction in every volume renderer, and it is fiddly by nature.



color and alpha over the scalar range



sampled values resolved to RGBA

Composite the samples into one color.

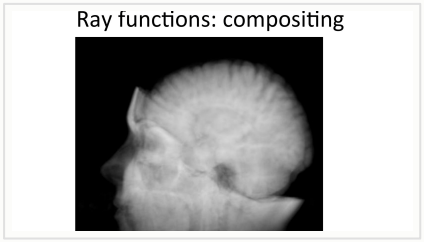
- Walk the samples front to back, accumulating color weighted by remaining transparency.
- Opacity is defined per unit distance, so it must be corrected when the step size changes.

– Halve the step and you double the number of contributions

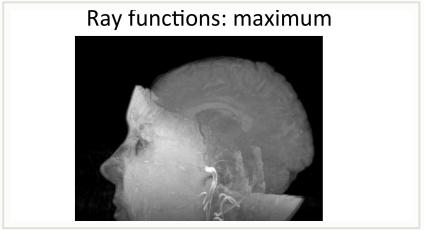
- Once accumulated opacity is near one, stop — early ray termination.

an opacity profile along one ray

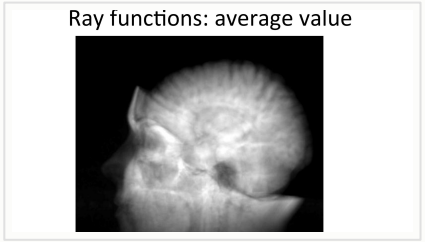
Compositing is one choice among several.



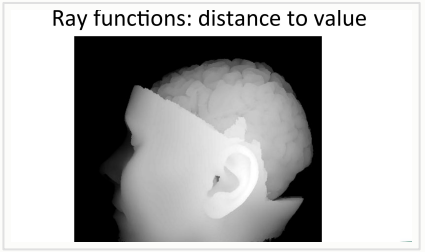
COMPOSITE



MAXIMUM



AVERAGE



DISTANCE TO VALUE

Same data, same rays, different reduction. Maximum intensity projection is standard practice in medical imaging.

Images from the VTK book

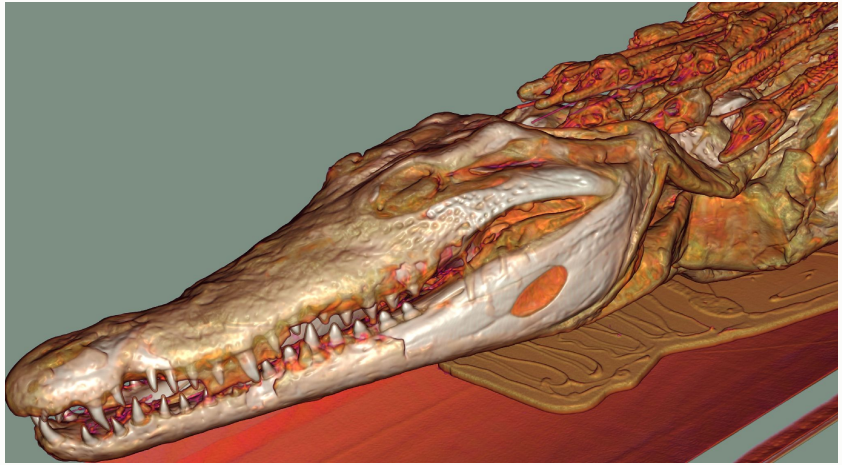
Lighting a volume: there is no surface to shade.

- Shading needs a normal. A volume has no surface to take one from.
- Use the gradient of the scalar field as the normal.

– Central differences on the six neighbours

– Gradient magnitude conveniently indicates how surface-like a point is

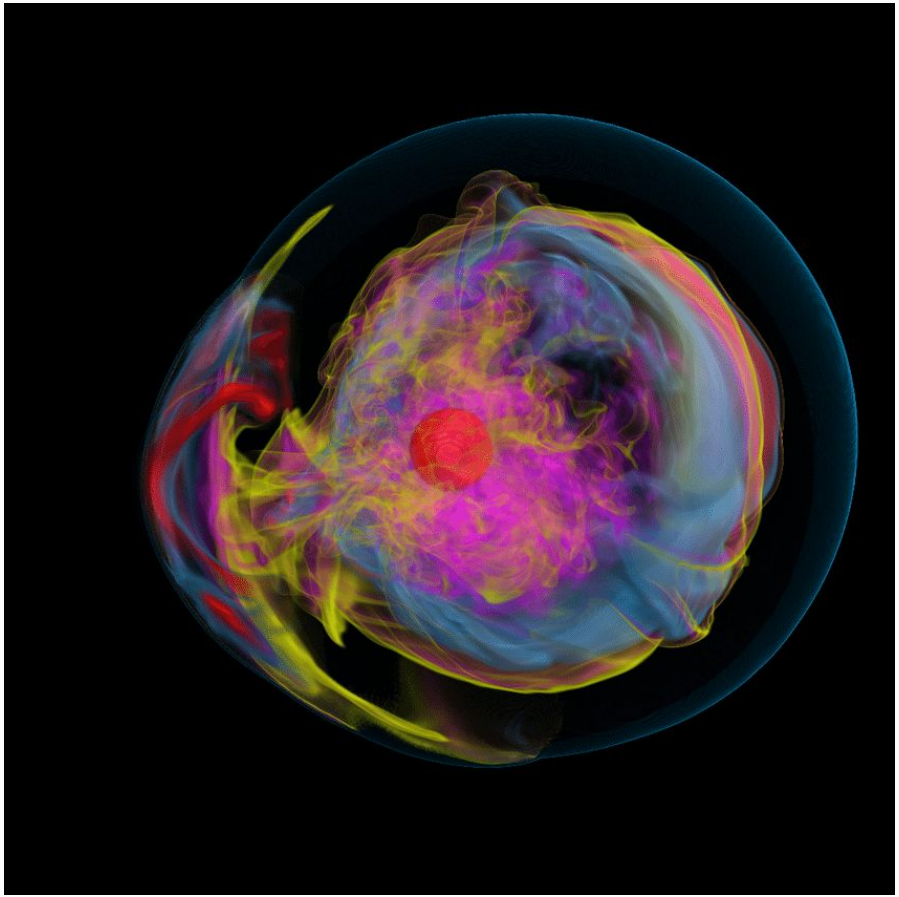
- Then run the same lighting equation as before.



gradient-based normals

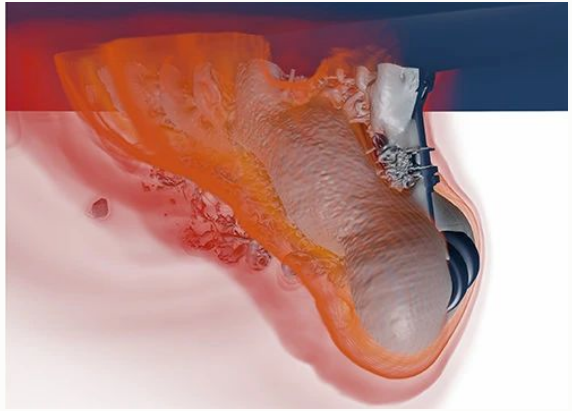
Making it fast.

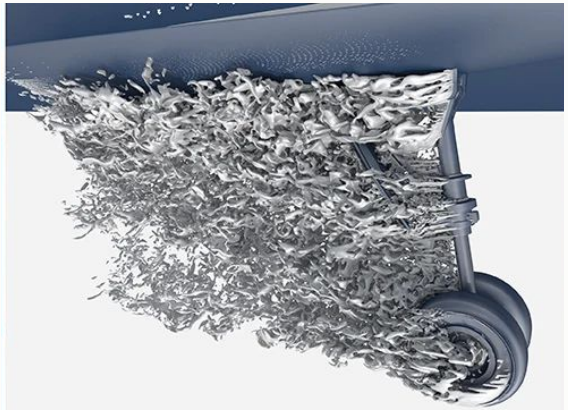
- Most of a volume is empty, and sampling empty space costs the same as sampling data.
- Build a hierarchy over the field, store per-node value ranges, and skip whole subtrees the transfer function maps to zero. Octrees, BVH, kd-trees
- Adaptive sampling: large steps in the boring regions, small steps near features.
- Early ray termination pairs with all of it.



Isosurfaces without the isosurface.

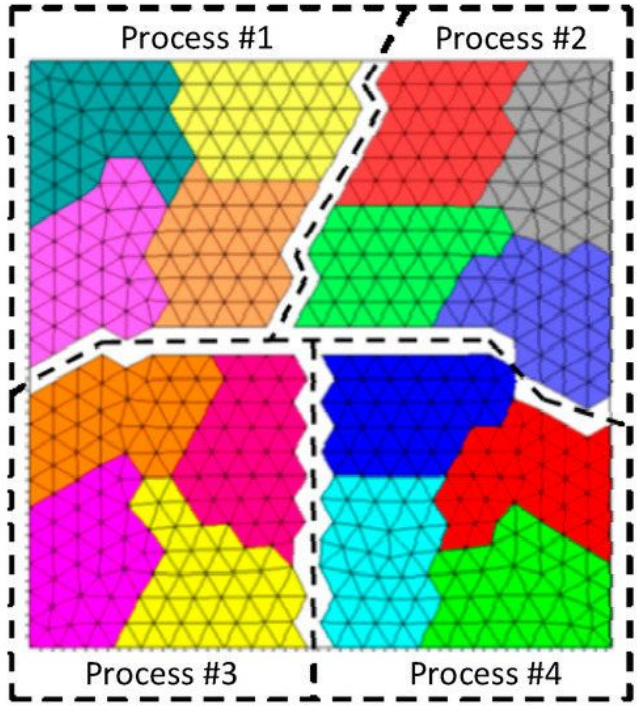
- You do not have to extract triangles to see an isosurface.
- March along the ray watching for the sign change in F − isovalue, then refine the crossing.
- No mesh generated, no memory spent, and the surface is exact at pixel resolution.
- The normal comes from the gradient — same trick as volume shading.





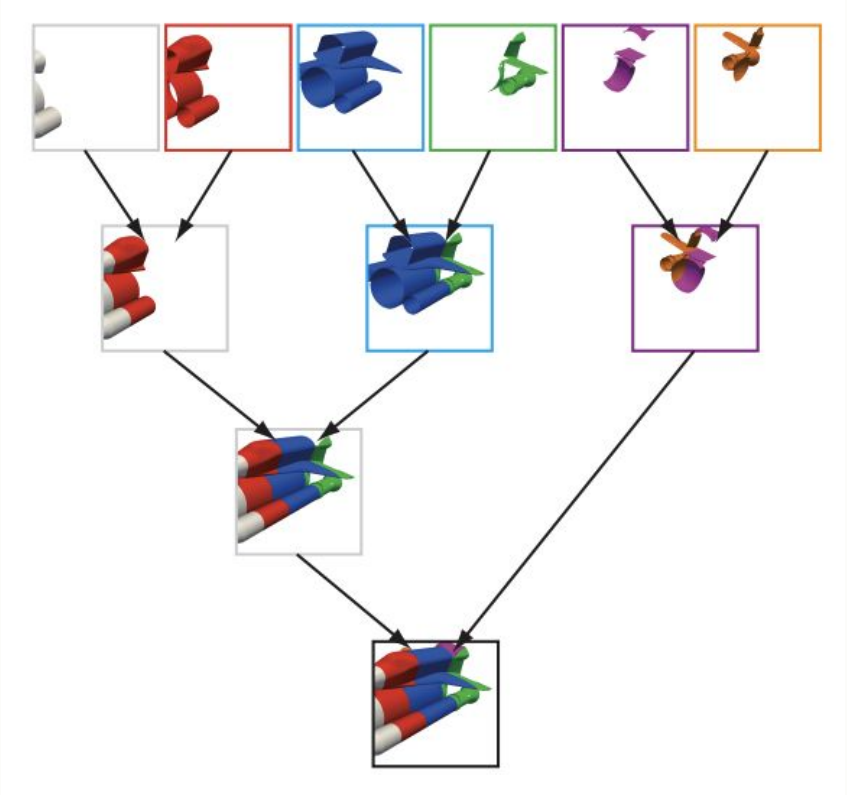
One image, many ranks: parallel compositing.

- Each rank owns a piece of the mesh, so each renders a partial image with depth or accumulated opacity.
- Those partials have to be merged in the correct order to make one picture.
- Three options
  - Sort first (primitives)
  - Sort middle (primitives)
  - Sort last (fragments)



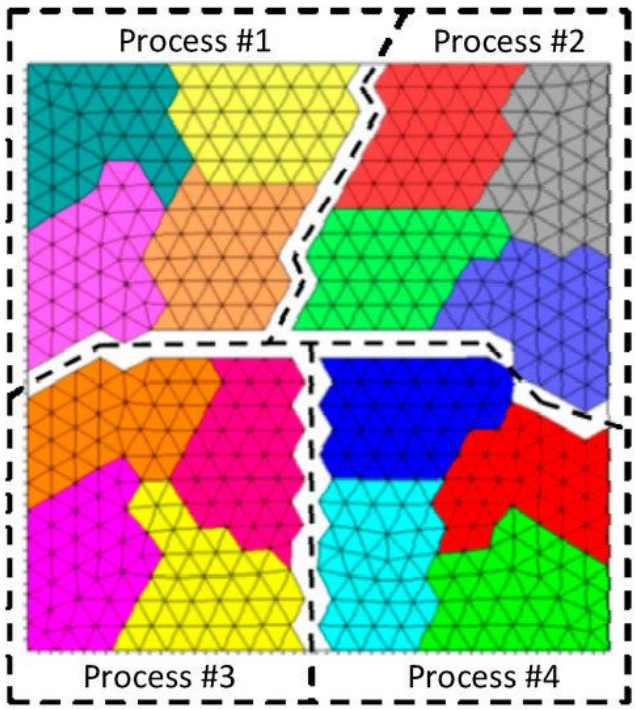
Sort last compositing

- Multiple strategies
  - Direct send
  - Binary swap
  - Radix-k
- Cost scales with pixels and ranks, not with cells — which is why it survives at full machine scale.



What about ray tracing?

- Distributed memory ray tracing is complicated
- Strategies
  - Scene duplication and render by tiles (not HPC)
  - Ray passing (complicated)
  - Ray Queue Cycling (simple)



WRAP UP

It is graphics, with the burden of being right.

- A mesh and a field, not an artist's triangles.
- A handful of filters, composed into a graph.
- Color and opacity as measurement, not decoration.
- Rays through data instead of geometry — then make it fast.