Real-Time Visualization

04: Real-Time Computations on GPUs

Stefan Bruckner

Lectures

1. **Introduction:** Volume rendering basics, GPU architecture, OpenGL, CUDA, case studies, … (03.05.)
2. **Real-Time Volume Graphics 1:** GPU ray-casting, optimizations, memory management, OpenGL vs. CUDA, … (10.05.)
3. **Real-Time Volume Graphics 2:** Advanced illumination, filtering, derivatives, advanced transfer functions, … (17.05.)
4. **Real-Time Computations on GPUs:** Fluid simulation, level-set deformation, … (24.05.)
5. ~~**Volumetric Special Effects:** combining simulation and ray-casting, integration in game engine scenes, … (31.05.)~~
6. **Final event:** Project presentations (28.06.)

Fluid Simulation

Applications

- Scientific visualization (Water sewage system, dam construction)
- Medical simulation (Blood flow)
- Movie special effects (Finding Nemo, Pirates of Caribbean)
- Games (Half Life, Crysis)

Fluid Characteristics (1)

Basic properties

- Pressure, density, viscosity, surface tension

Different types of fluids:

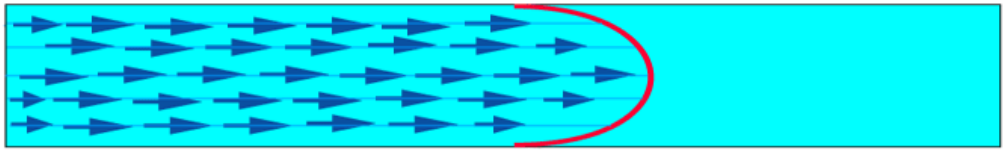
- Incompressible (divergence-free) fluids: no change in volume

- Compressible fluids: the fluid's volume can change significantly

- Viscous fluids: the fluid tends to resist a certain degree of deformation

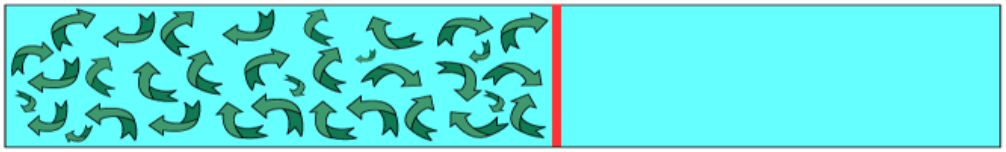
Fluid Characteristics (2)

Laminar flow



- Flow that has smooth behavior

Turbulent flow



- Flow that appears to have chaotic and random changes

Inviscid fluids

- Fluids which do not have resistance to shear stress

Newtonian fluids

- Constant viscosity – continue to flow, regardless of the force acting on it (e.g, water)

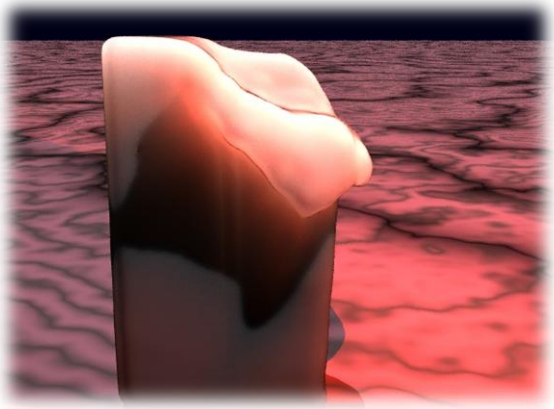
Fluid Characteristics (3)

Non-Newtonian fluids

- Fluids that have non-constant viscosity
- Typically dependent on shear rate or shear rate history

Phase Transition

- Fluids may change physical behavior under different environmental conditions (e.g., temperature, pressure)



Challenges

Modeling continuum fluids on discrete systems – it's all about approximations

- Topological variations
- Different kinds of behaviors with interacting subjects
- Numerical stabilities, accuracy and convergence issues

Performance

- Grid and temporal resolution

User interaction

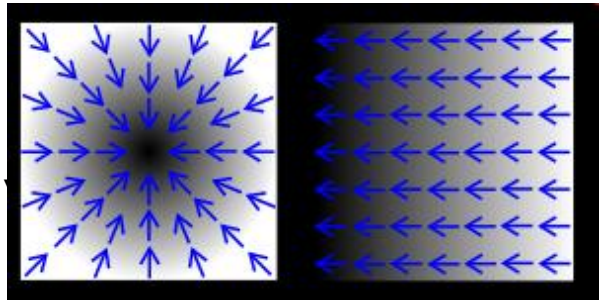
- Artistic control

Calculus Review (1)

Gradient (∇): A vector pointing in the direction of the greatest rate of increment

$$\nabla u = \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial z} \right)$$

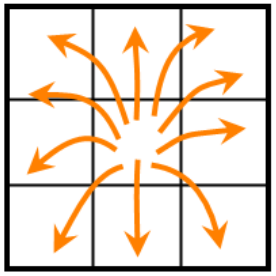
u can be a scalar or a vector



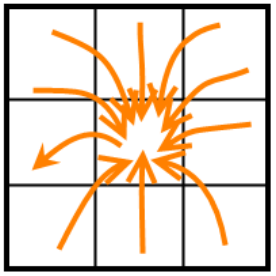
Divergence ($\nabla \cdot$): Measures how the vectors are converging or diverging at a given location (volume density of the outward flux)

$$\nabla \cdot \boldsymbol{u} = \frac{\partial \boldsymbol{u}}{\partial x} + \frac{\partial \boldsymbol{u}}{\partial y} + \frac{\partial \boldsymbol{u}}{\partial z}$$

u can only be a vector



Source, Div(**u**) > 0



Sink, Div(**u**) < 0

Calculus Review (2)

Laplacian (Δ or ∇^2): Divergence of the gradient

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}$$

u can be a scalar or a vector

Finite Difference: Derivative approximation

$$\frac{\partial u}{\partial x} = \frac{u_{i+1} - u_i}{x_{i+1} - x_i}$$

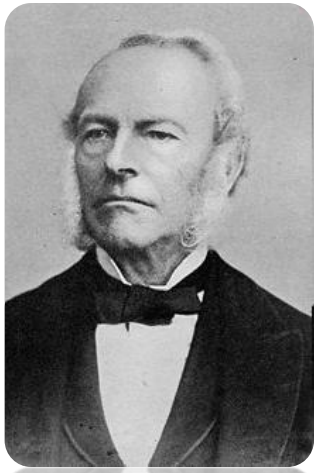
Navier-Stokes Equation

Momentum equation

$$\mathbf{u}_t = k\nabla^2\mathbf{u} - (\mathbf{u}\cdot\nabla)\mathbf{u} - \nabla p + \mathbf{f}$$

change in velocity | diffusion/ viscosity | advection | pressure | body forces

Claude-Louis Navier
(1785 - 1836)



George Gabriel Stokes
(1819 - 1903)

Incompressibility

$$\nabla\cdot\mathbf{u}=0$$

u: the velocity field

k: kinematic viscosity

Finite Difference Grids (1)

All values live on regular grids

- Need *scalar* and *vector* fields
- **Scalar fields:** Amount of smoke or dye
- **Vector fields:** Fluid velocity

Subtract adjacent quantities to approximate derivatives

Finite Difference Grids (2)

Use discretized forms of differential operators

Operator	Definition	Finite Difference Form
Gradient	$\nabla p = \left(\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y} \right)$	$\frac{p_{i+1,j} - p_{i-1,j}}{2\delta x}, \frac{p_{i,j+1} - p_{i,j-1}}{2\delta y}$
Divergence	$\nabla \cdot \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$	$\frac{u_{i+1,j} - u_{i-1,j}}{2\delta x} + \frac{v_{i,j+1} - v_{i,j-1}}{2\delta y}$
Laplacian	$\nabla^2 p = \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2}$	$\frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{(\delta x)^2} + \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{(\delta y)^2}$

Techniques Overview

Borrowed from CFD (Computational Fluid Dynamics)

Common techniques for solving Navier Stoke's equation:

- **Eulerian approach (grid-based)**
- **Lagrangian approach (particle-based)**
- Spectral method
- Lattice Boltzmann method

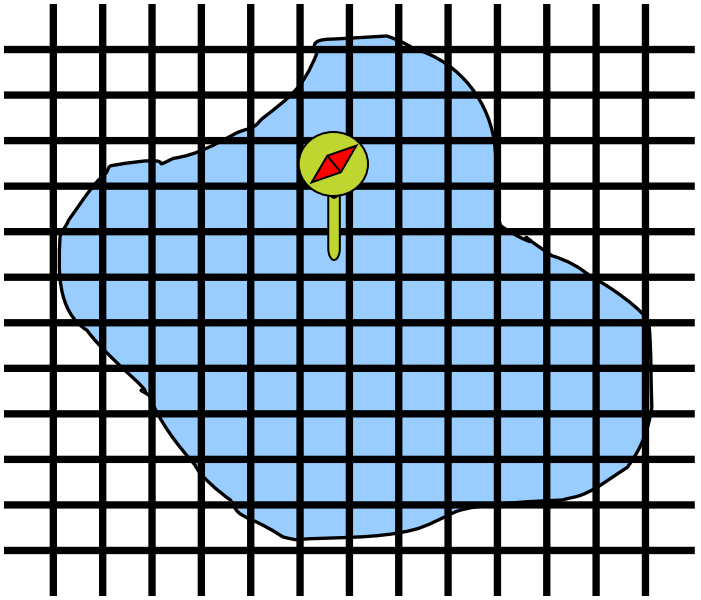
Eulerian Approach

Discretize the domain using finite differences

- Define scalar & vector fields on the grid
- Use the operator splitting technique to solve each term separately

Evaluation:

- Derivative approximation
- Adaptive time step/solver
- Memory usage & speed
- Grid artifact/resolution limitation



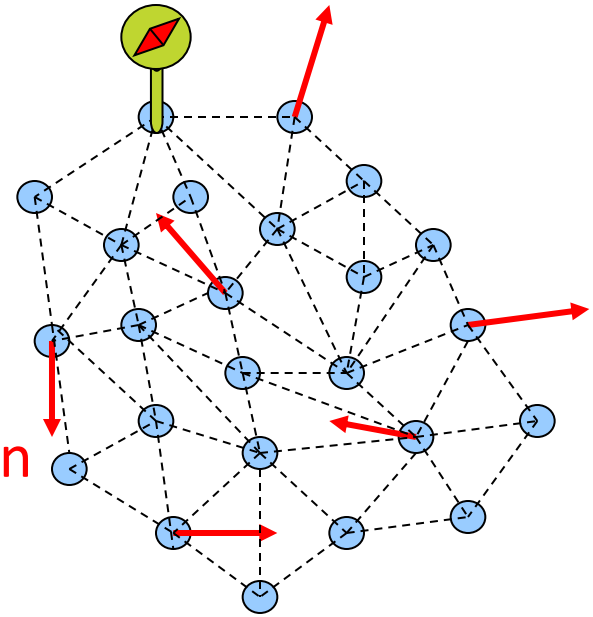
Lagrangian Approach

Treat the fluid as discrete particles

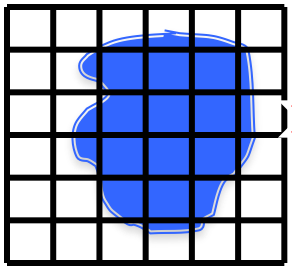
- Apply interaction forces (i.e. pressure/viscosity) according to certain pre-defined smoothing kernels

Evaluation:

- Mass / Momentum conservation
- More intuitive
- Fast, no linear system solving
- Connectivity information/surface reconstruction



Simulation Loop



scalar/vector fields defined on the grid

advection

body force

diffusion

pressure solve

$$\mathbf{u}_t = k\nabla^2\mathbf{u} - (\mathbf{u}\cdot\nabla)\mathbf{u} - \nabla p + \mathbf{f}$$

$$\nabla\cdot\mathbf{u}=0$$

Advection

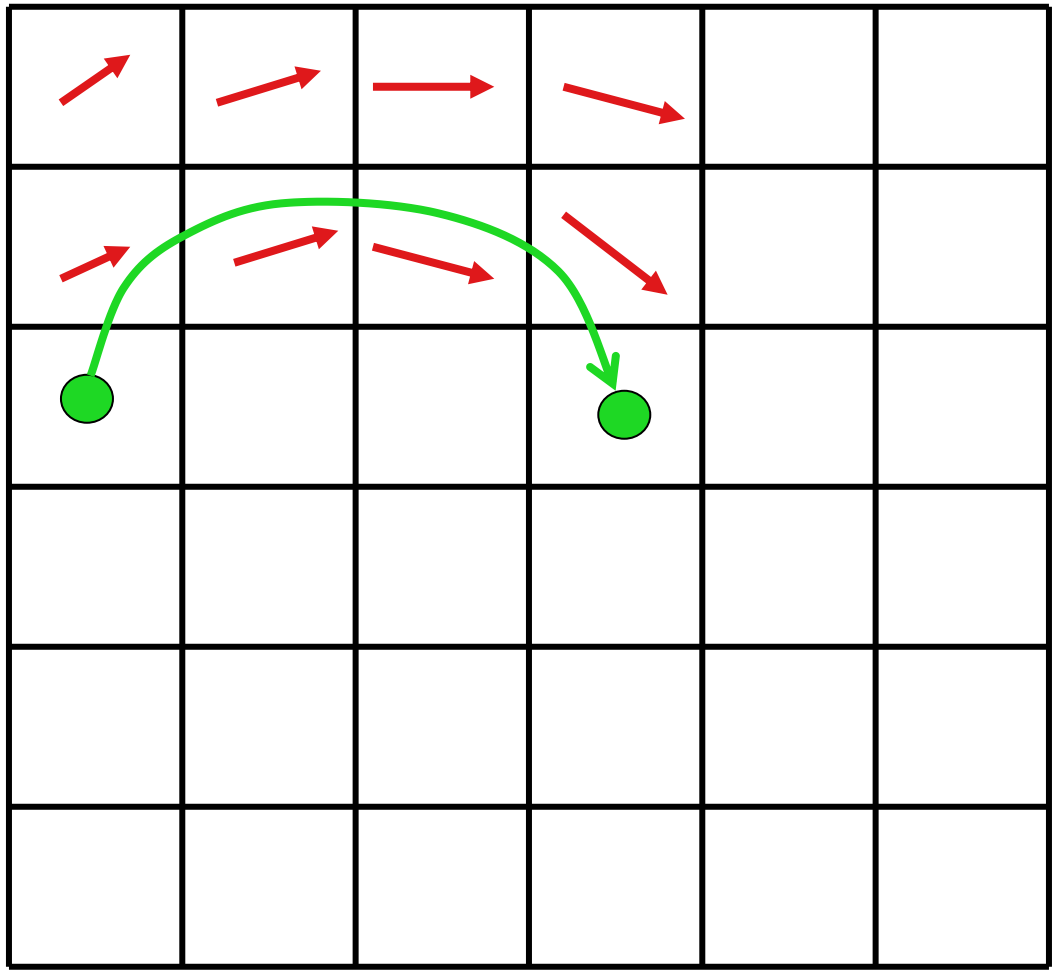
Sometimes called “convection” or “transport”

- Defines how a quantity moves with the underlying velocity field
- Ensures the conservation of momentum

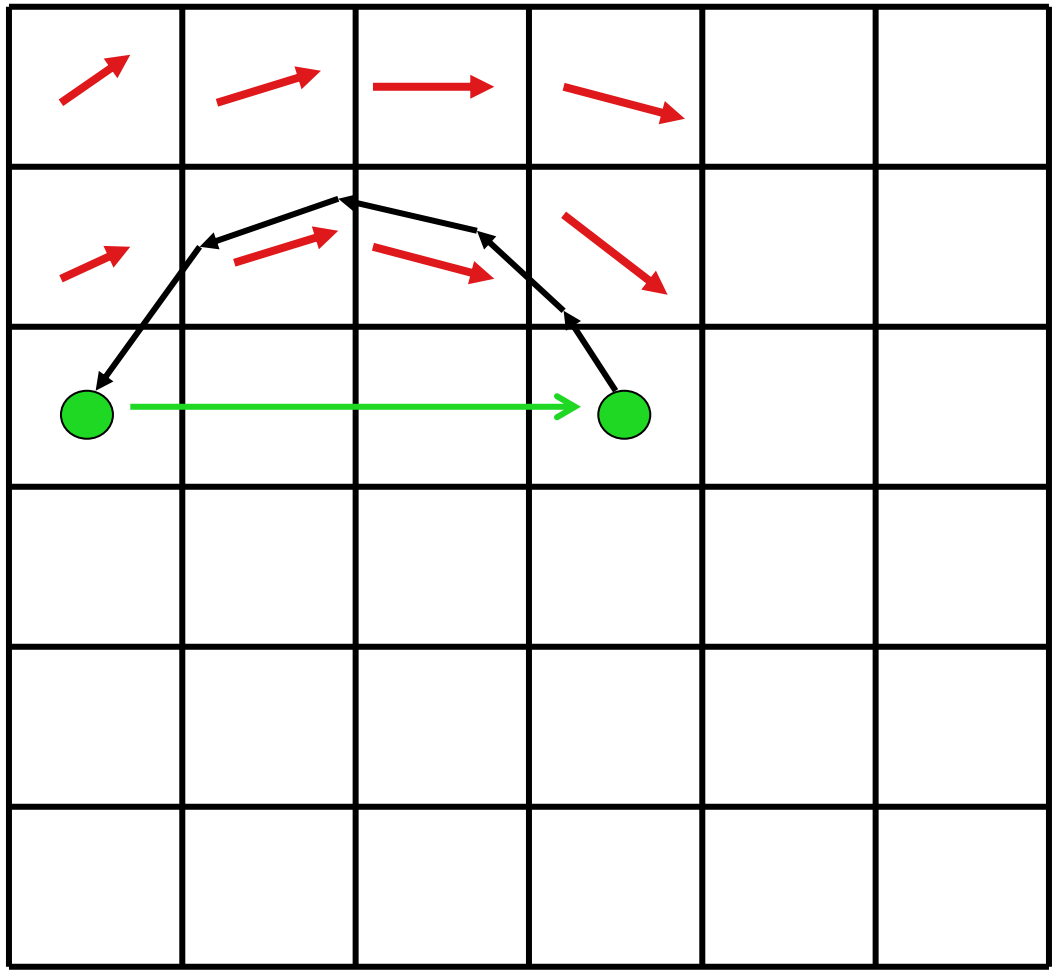
Approaches:

- Forward Euler (unstable)
- Semi-Lagrangian advection (stable for large time steps, but suffers from the dissipation issue)

Forward Euler Advection



Semi-Lagrangian Advection



BFECC (1)

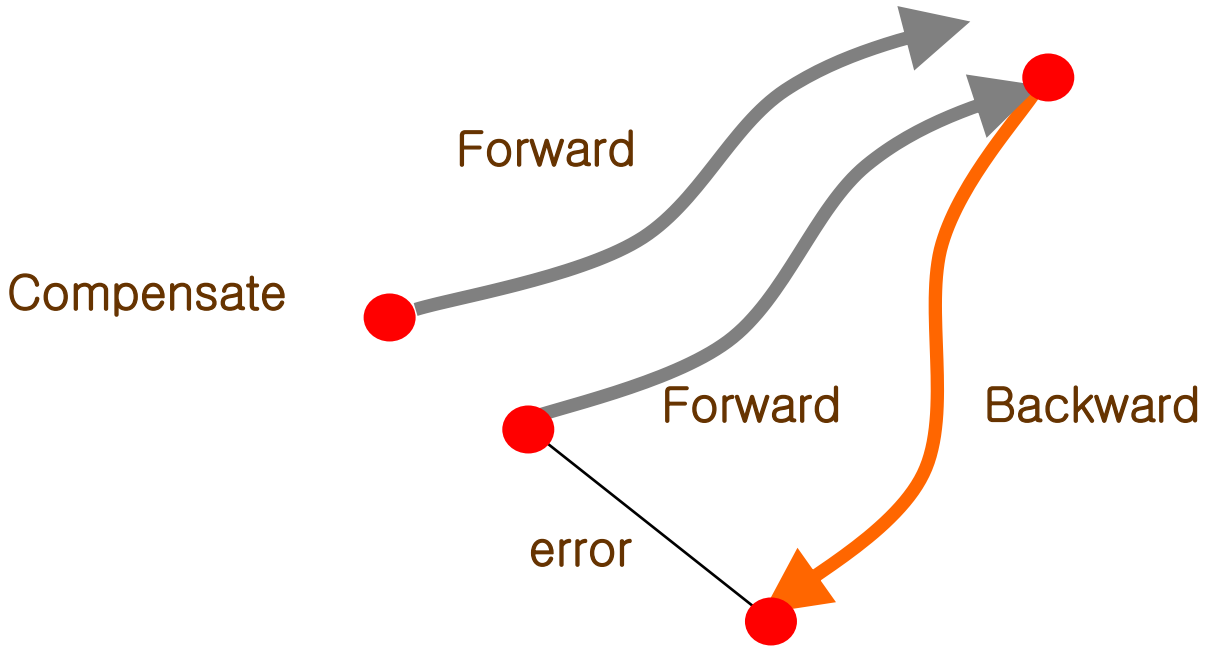
Forward schemes are unstable

Semi-Lagrangian approach leads to diffusion

Back and Forth Error Compensation and Correction (BFECC) [Moon 2005]

- Perform forward advection
- Do backward advection from new position
- Compute error and take correction step
- Do forward advection from corrected position

BFECC (2)



Scalar Field Advection

$$c_t = -(\mathbf{u} \cdot \nabla)c$$

Vector Field Advection

$$\mathbf{u}_t = -(\mathbf{u} \cdot \nabla)\mathbf{u}$$

For each dimension:

$$\mathrm{u}^{\mathrm{x}}{}_t = -(\mathbf{u} \cdot \nabla)\mathrm{u}^{\mathrm{x}}$$

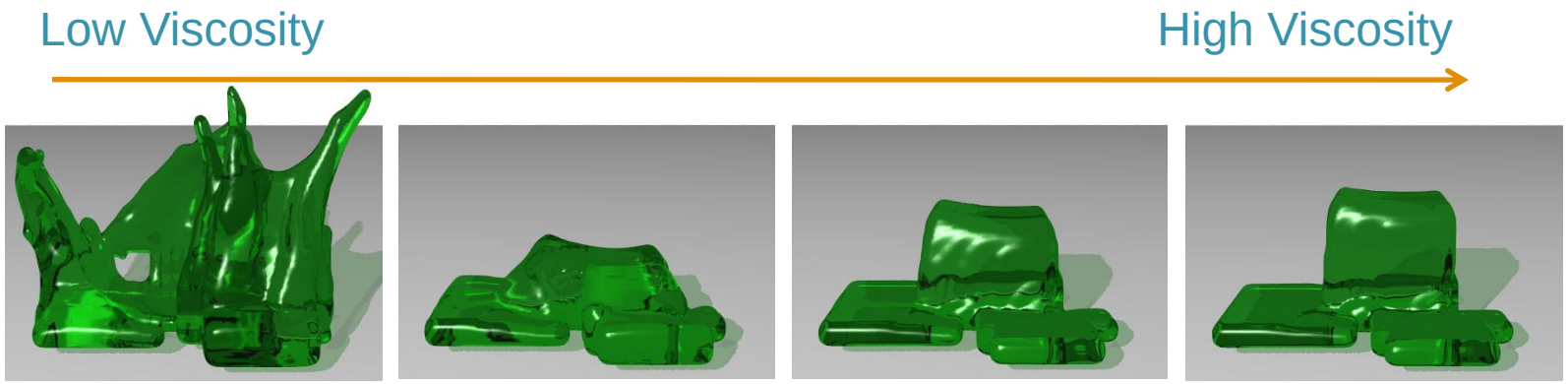
$$\mathrm{u}^{\mathrm{y}}{}_t = -(\mathbf{u} \cdot \nabla)\mathrm{u}^{\mathrm{y}}$$

$$\mathrm{u}^{\mathrm{z}}{}_t = -(\mathbf{u} \cdot \nabla)\mathrm{u}^{\mathrm{z}}$$

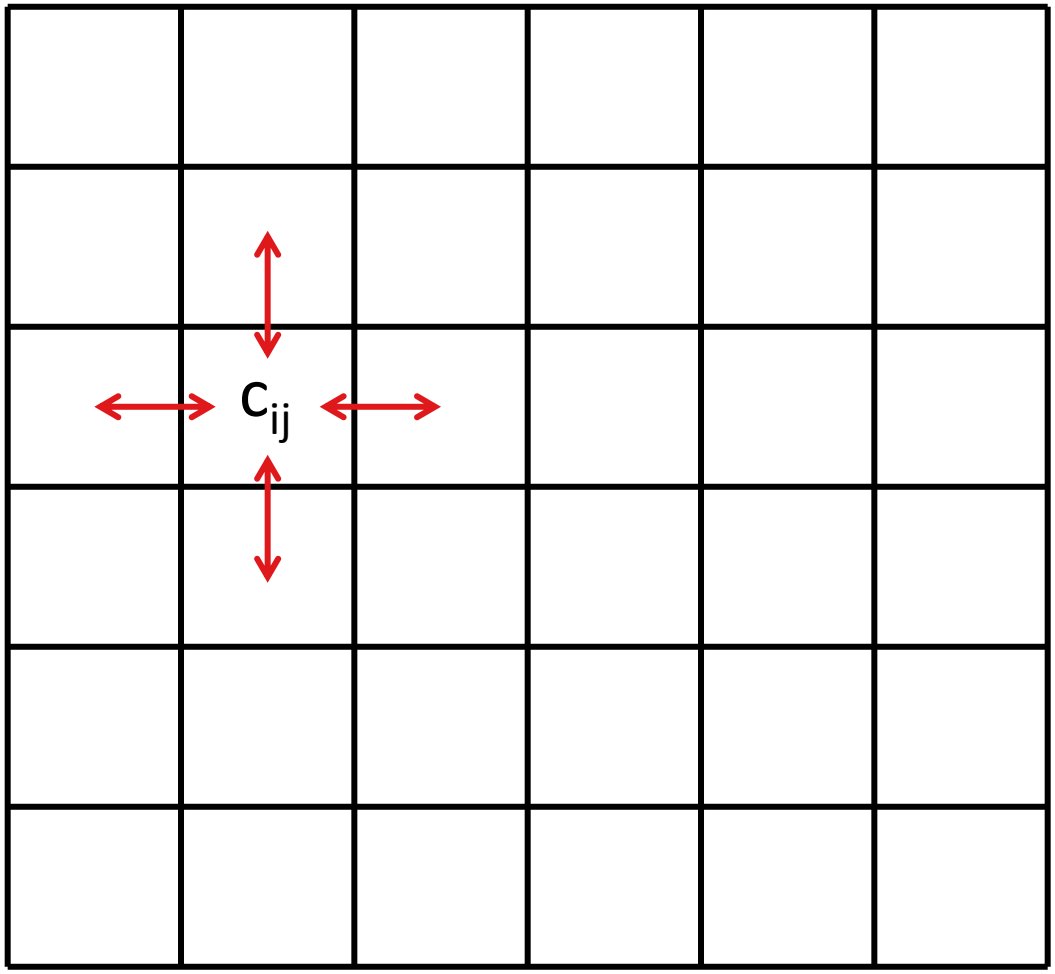
Diffusion (1)

Define how a quantity in a cell inter-changes with its neighbors

- Essentially results in a blurring process (i.e., dissipation to neighborhood)



Diffusion (2)

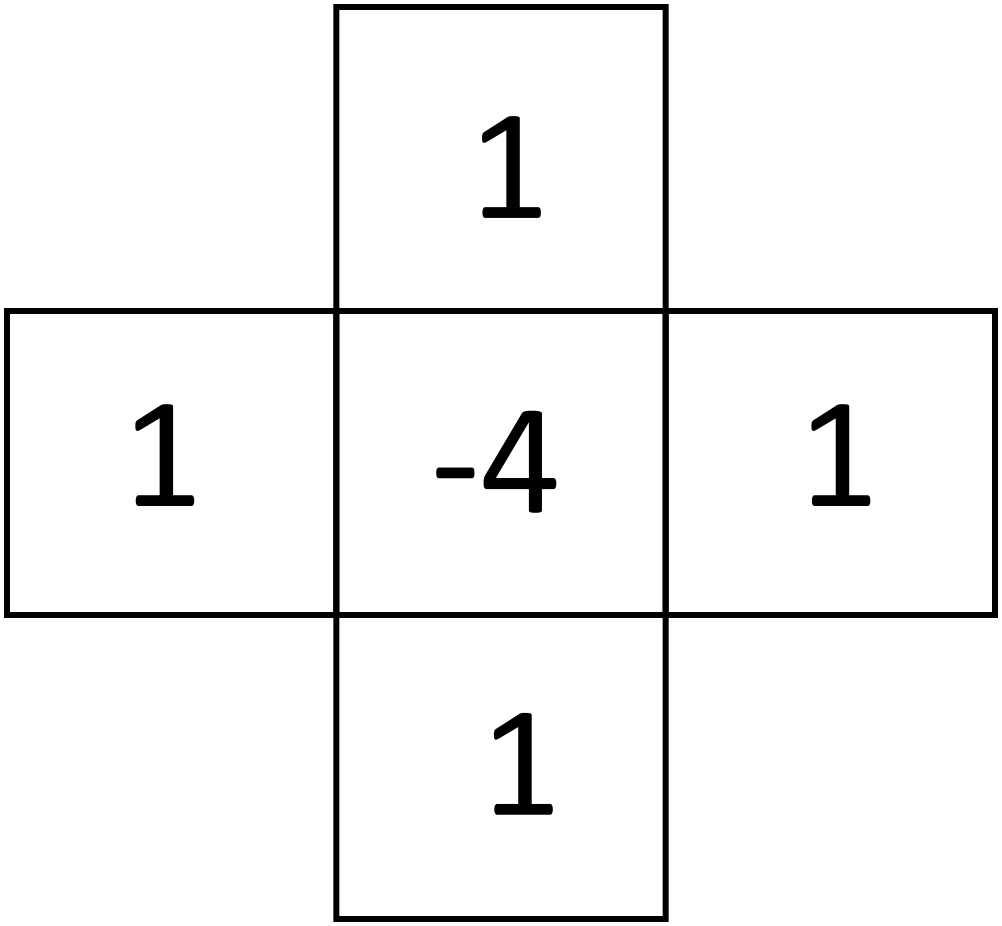


Scalar Field Diffusion

$$c_t = k\nabla^2 c$$

change in value

value relative to neighbors



$$c_{ij}^{new} = c_{ij} + k\,\Delta t\,(c_{i-1j} + c_{i+1j} + c_{ij-1} + c_{ij+1} - 4c_{ij})$$

Vector Field Diffusion

$$\mathbf{u}_t = k\nabla^2\mathbf{u}$$

viscosity

For each dimension

$$u^x{}_t = k\nabla^2 u^x$$

$$u^y{}_t = k\nabla^2 u^y$$

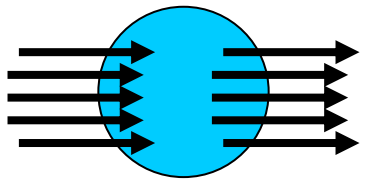
$$u^z{}_t = k\nabla^2 u^z$$

Pressure Solve (1)

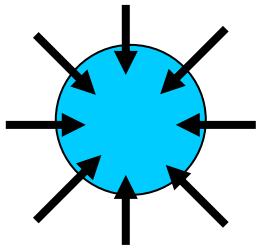
Sometimes called sometimes called “pressure projection”

Effects of pressure

- Keeps the fluid at constant volume (incompressible, conservation of mass)
- Makes sure the velocity field stays divergence-free



incompressible



compressible

Pressure Solve (2)

Helmholtz-Hodge Decomposition

- Vector field $\mathbf{w}$ can be decomposed into

$$\mathbf{w} = \mathbf{u} + \nabla p$$

  where $\mathbf{u}$ has zero divergence.

- We can decompose any vector field into a divergence-free vector field and the gradient of a scalar field
- Means that we can correct for divergence by subtracting ∇p from a vector field $\mathbf{w}$

Pressure Solve (3)

Helmholtz-Hodge Decomposition also leads to a method for computing the pressure field

$$\mathbf{w} = \mathbf{u} + \nabla p$$

Apply the divergence operator to both sides of the equation:

$$\nabla \cdot \mathbf{w} = \nabla \cdot \mathbf{u} + \nabla^2 p$$

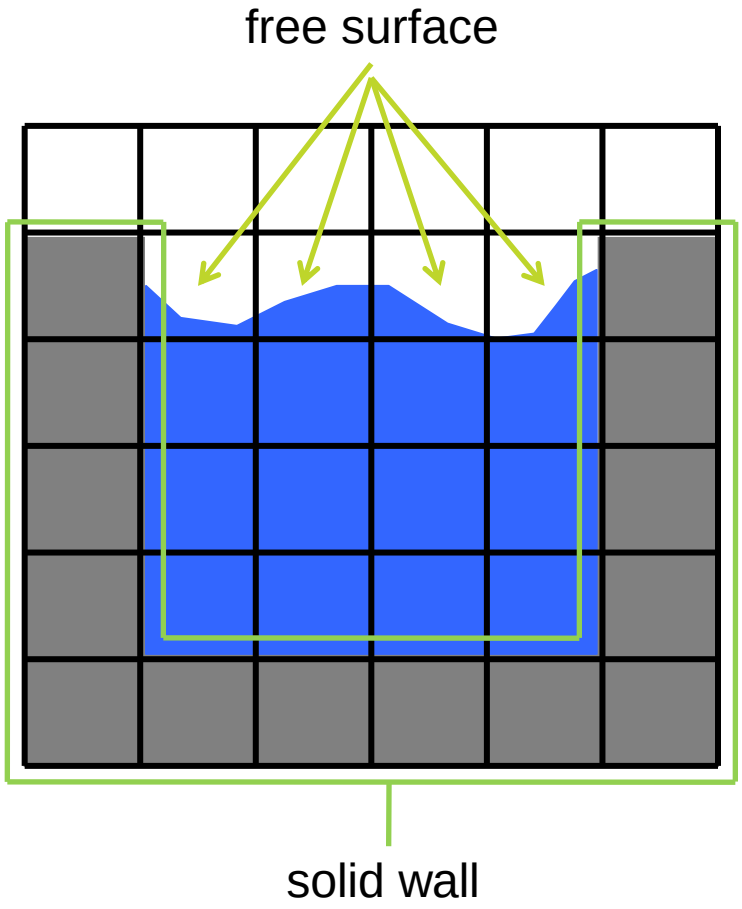
Under the assumption of incompressibility ($\nabla \cdot u = 0$) this becomes a Poisson equation:

$$\nabla^2 p = \nabla \cdot \mathbf{w}$$

Pressure Solve (4)

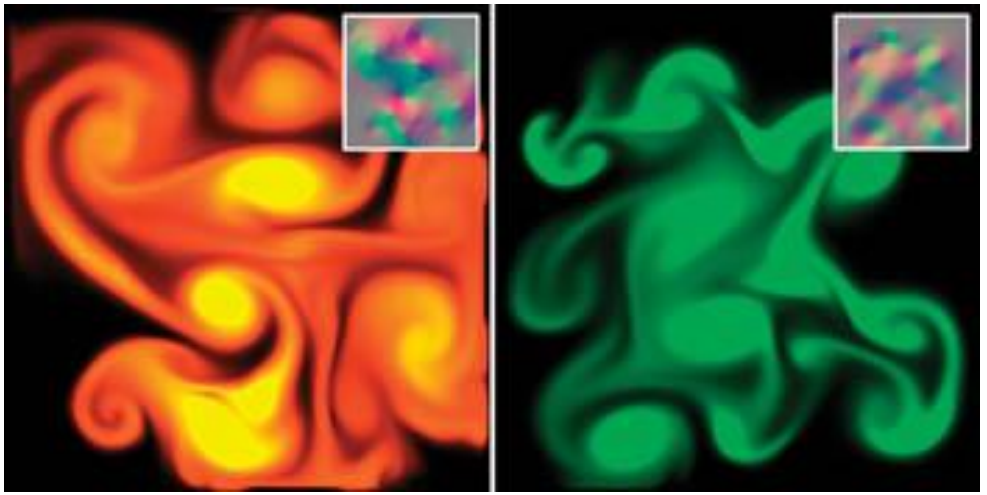
Boundary conditions

- **Periodic:** wrap around (fluid "lives" on a torus with the respective dimensionality) – can lead to very simple formulations
- **Solid wall:** The fluid can't penetrate the wall but can flow freely in tangential directions (Neumann BC)
- **Free surface:** The fluid can evolve freely (p = 0)



Fluid Simulation and Rendering

- Compute advection of fluid
  - (Incompressible) Navier-Stokes solvers
  - Lattice Boltzmann Method (LBM)
- Discretized domain; stored in 2D/3D textures
  - Velocity, pressure
  - Dye, smoke density, vorticity, …
- Updates in multi-passes
- Render current frame



Courtesy Mark Harris

Stable Fluids

Basic approach [Stam 2000]:

- Add forces: $\mathbf{w}_1 = \mathbf{u}(\mathbf{x}, t) + \mathbf{f}(\mathbf{x}, t)\Delta t$
- Advect: $\mathbf{w}_2 = \mathbf{w}_1(\mathbf{x} - \mathbf{w}_1\Delta t)$
- Diffuse: $(\mathbf{I} - \nu\Delta t\nabla^2)\mathbf{w}_3 = \mathbf{w}_2$
- Solve for pressure: $\nabla^2 p = \nabla \cdot \mathbf{w}_3$
- Subtract pressure gradient: $\mathbf{u}(\mathbf{x}, t + \Delta t) = \mathbf{w}_3 - \nabla p$

Add Forces

$$\mathbf{w}_1 = \mathbf{u}(\mathbf{x}, t) + \mathbf{f}(\mathbf{x}, t)\Delta t$$

Scale force by time step, add to velocity

- Simple approach: “splatting”
- Color of splat encodes direction and strength of force
- Add Gaussian splat to velocity texture

Advection

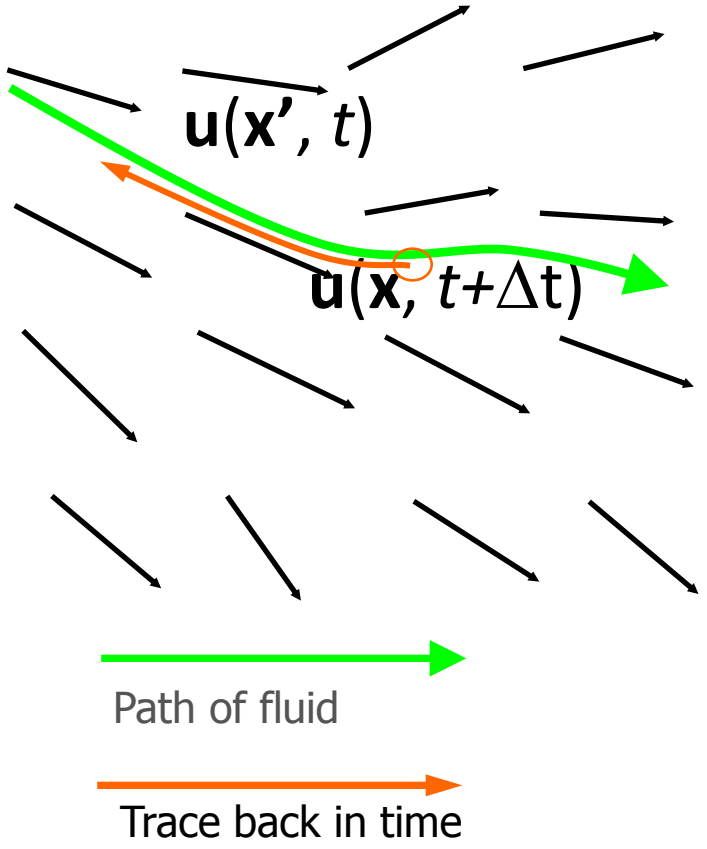
Advection: quantities in a fluid are carried along by its velocity

$$\mathbf{w}_2(\mathbf{x}) = \mathbf{w}_1(\mathbf{x} - \mathbf{w}_1 \Delta t)$$

Want velocity at position x at new time t + Δt

Follow velocity field back in time from x: (x - w_1Δt)

- Like tracing particles
- Simple in kernel or fragment program



Viscous Diffusion

$$(\mathbf{I} - \nu \Delta t \nabla^2)\mathbf{w}_3 = \mathbf{w}_2$$

Viscous fluid exerts drag on itself

- Causes diffusion of velocity

Implicit, discrete form of $\frac{\partial \mathbf{w}_2}{\partial t} = \nu \nabla^2 \mathbf{w}_2$

- Explicit form is unstable
- Requires solving of a Poisson equation (see also next step)

Poisson-Pressure Solution

Poisson Equation

- Discretize, solve using iterative solver (relaxation)
- Jacobi, Gauss-Seidel, Multigrid, etc.
  - Jacobi easy on GPU, the rest are trickier
- Boils down to repeated evaluation of:

$$p_{i,j}^{n+1} = \frac{1}{4}\left(p_{i+1,j}^{n} + p_{i-1,j}^{n} + p_{i,j+1}^{n} + p_{i,j-1}^{n} - \delta^2(\nabla \cdot \mathbf{w}_3)\right),$$

$$\nabla \cdot \mathbf{w}_3 = \frac{1}{2\delta}\left(u3_{i+1,j} - u3_{i-1,j} + v3_{i,j+1} - v3_{i,j-1}\right)$$

δ = grid spacing
u, v = components of $\mathbf{w}_3$
i, j = grid coordinates
n = solution iteration

Subtract Pressure Gradient

$$\mathbf{u}(\mathbf{x}, t + \Delta t) = \mathbf{w}_3 - \nabla p$$

Last computation of the time step

- **u** is now a divergence-free velocity field

Very simple operations

$$u_{i,j} = u3_{i,j} - \frac{1}{2}\delta\left(p_{i+1,j} - p_{i-1,j}\right),$$

$$v_{i,j} = v3_{i,j} - \frac{1}{2}\delta\left(p_{i,j+1} - p_{i,j-1}\right)$$

δ = grid spacing
u, v = components of $\mathbf{w}_3$
i, j = grid coordinates
n = solution iteration

Additional Quantities

Can advect/diffuse additional quantities along with velocity and pressure using essentially the same steps

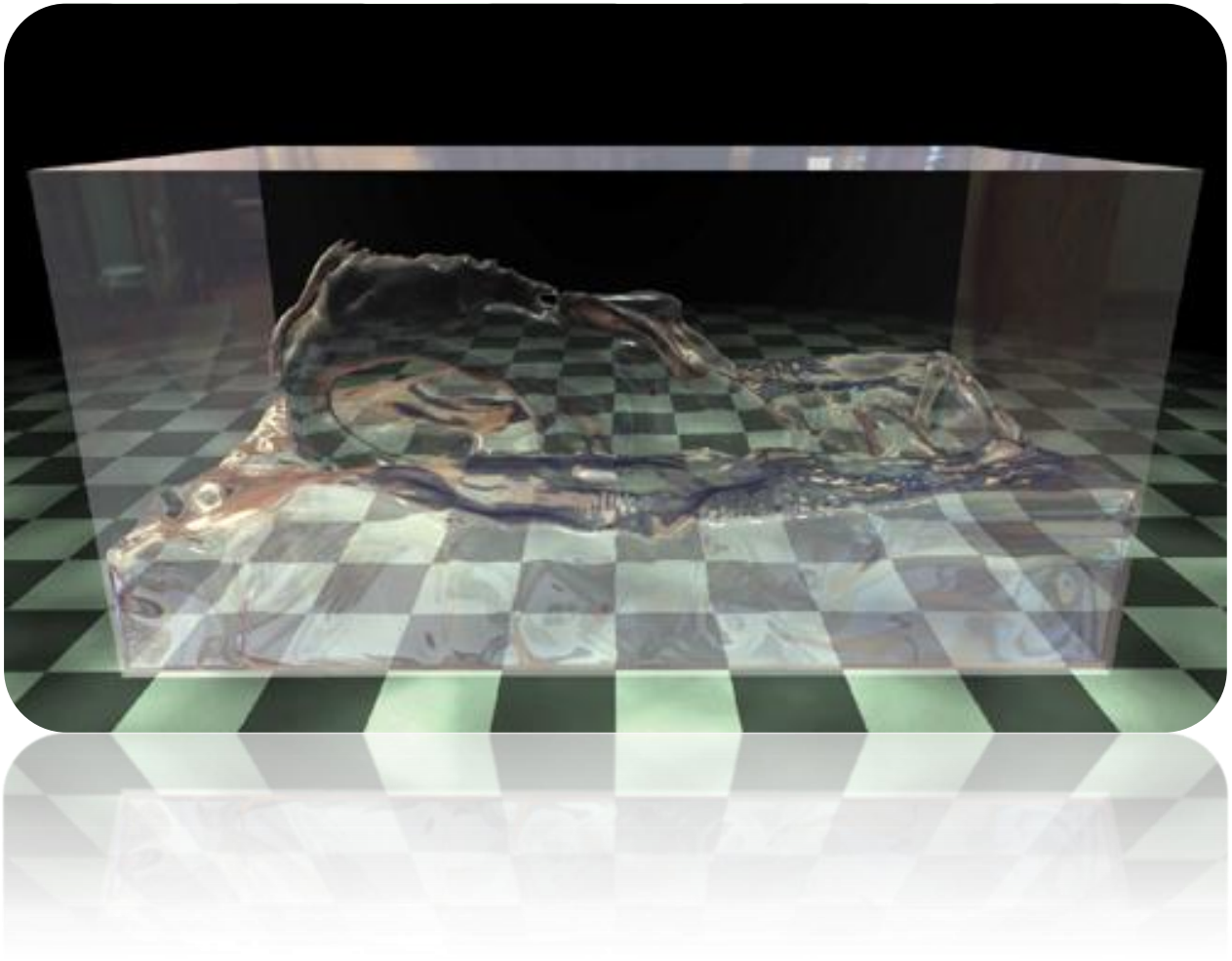
- Dye
- Density
- Temperature
- Texture coordinates

Fluid-Solid Interaction

Use voxelized geometry to specify boundary conditions

- GPU voxelization, inside-outside texture – non-watertight models can cause propblems
- For dynamic geometry, we also need to voxelity the solid's geometry
- See Keenan Krane in GPU Gems 3 for more details

Fluid Simulation Demo

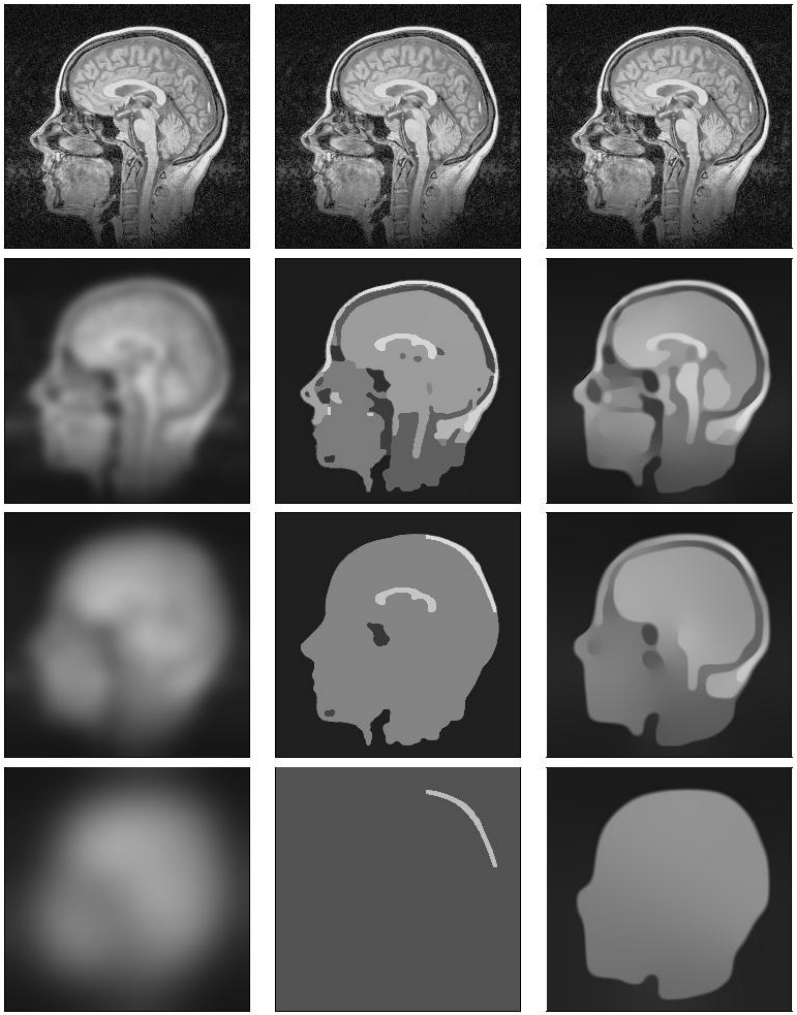


Diffusion Filtering (1)

Smoothing

- Linear isotropic
- Non-linear isotropic
- Non-linear anisotropic

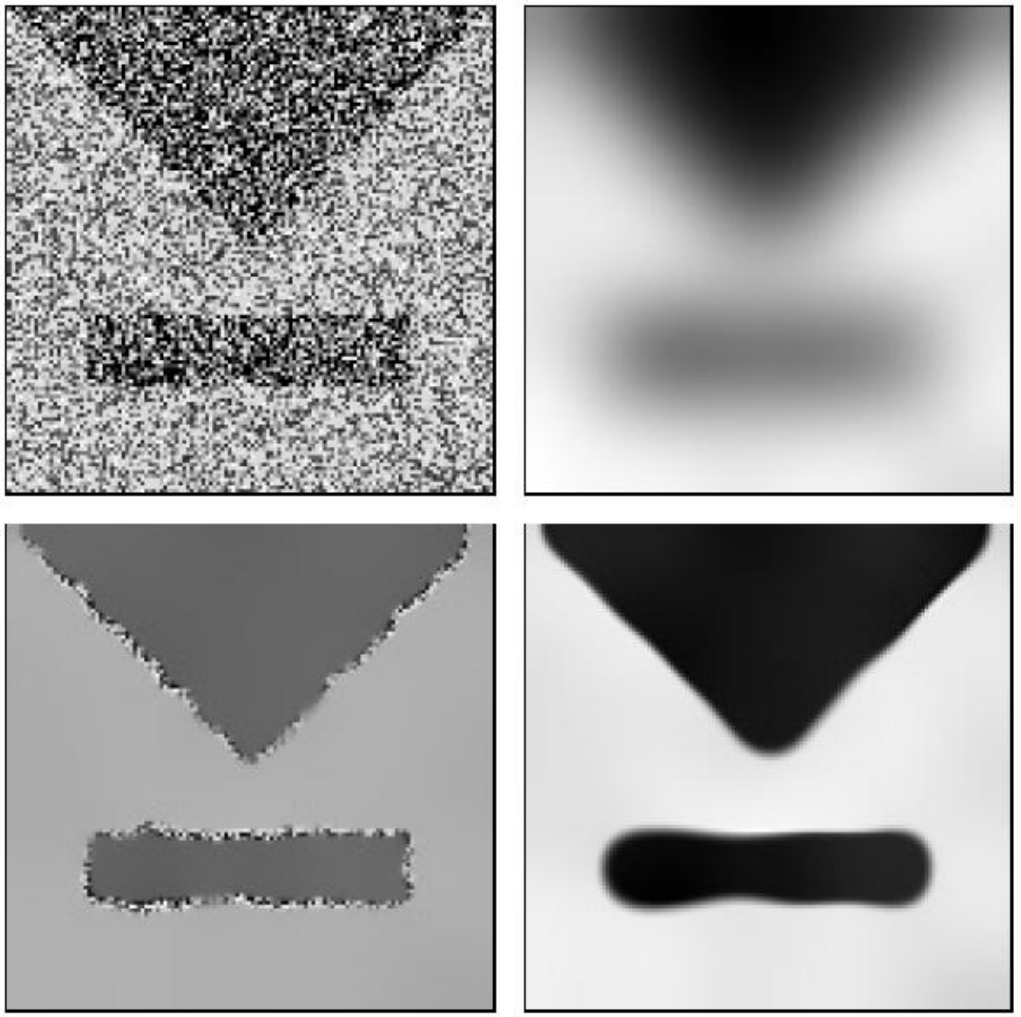
Scale space theory



Diffusion Filtering (2)

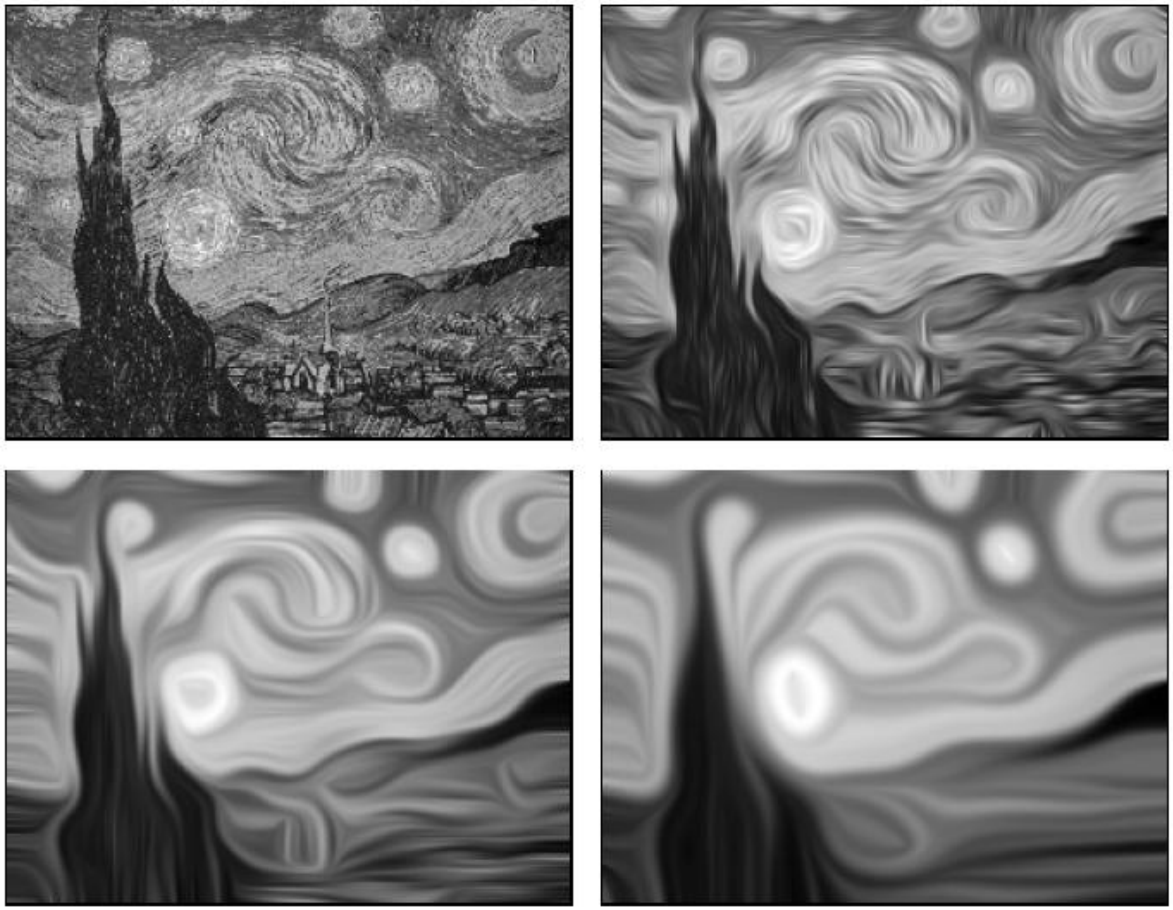
De-noising

- Original
- Linear isotropic
- Non-linear isotropic
- Non-linear anisotropic



Diffusion Filtering (3)

Coherence-enhancing diffusion



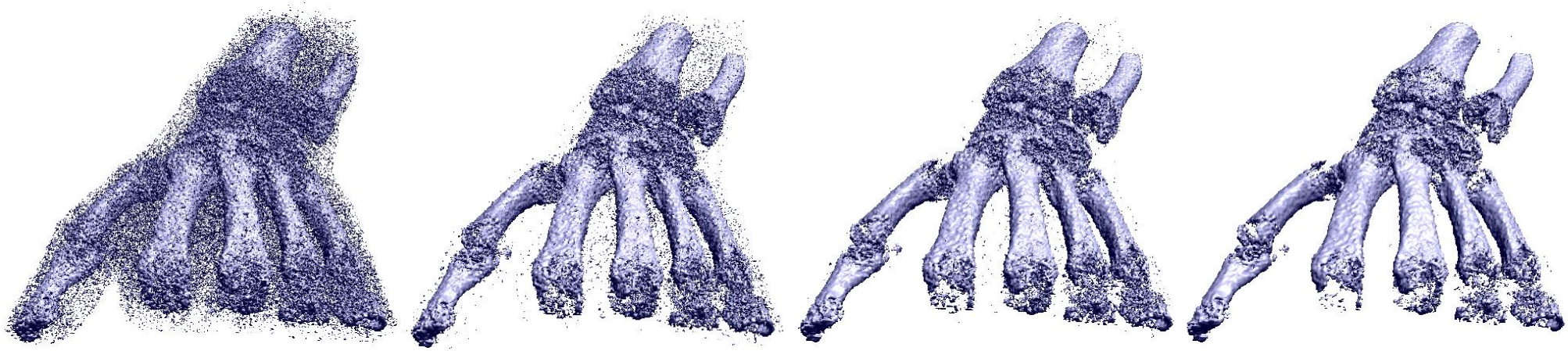
Diffusion Filtering of Volumes

Volume smoothing

- Nonlinear isotropic diffusion
- Edge-enhancing anisotropic diffusion

Diffusion-based segmentation

- Steer diffusion of seeds in segmentation mask from density volume



Diffusion

Equilibrates concentration differences without creating or destroying mass

$$\partial u_t = \mathrm{div}(D \cdot \nabla u)$$

D … diffusion tensor

- Isotropic: scalar function g() g controls intensity of flux
- Anisotropic: matrix g controls intensity and orientation of flux

∇u **…concentration gradient at position u(x,y,z)**

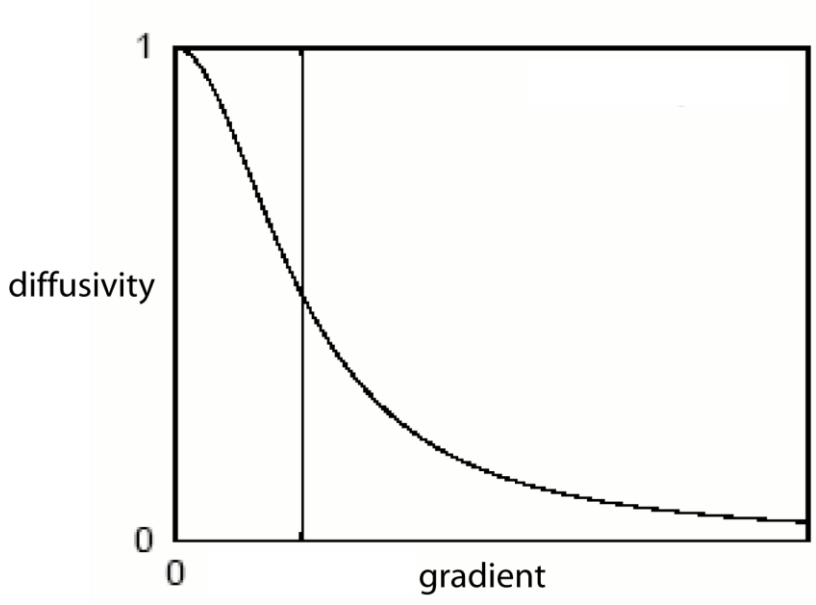
div …divergence operator

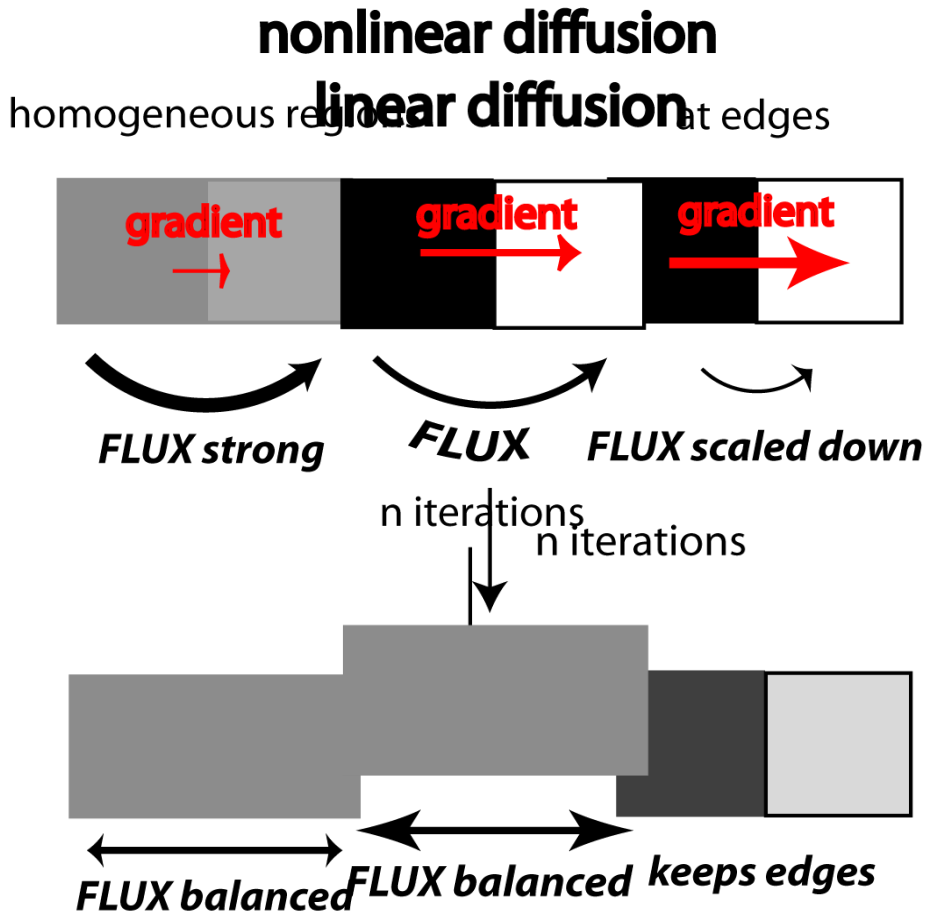
∂u_t **…modification of the volume at time step t**

Non-linear, Isotropic Diffusion

Feedback system: diffusivity function *g()*

$$\partial u_t = \mathrm{div}(g(\|\nabla u\|) \cdot \nabla u)$$



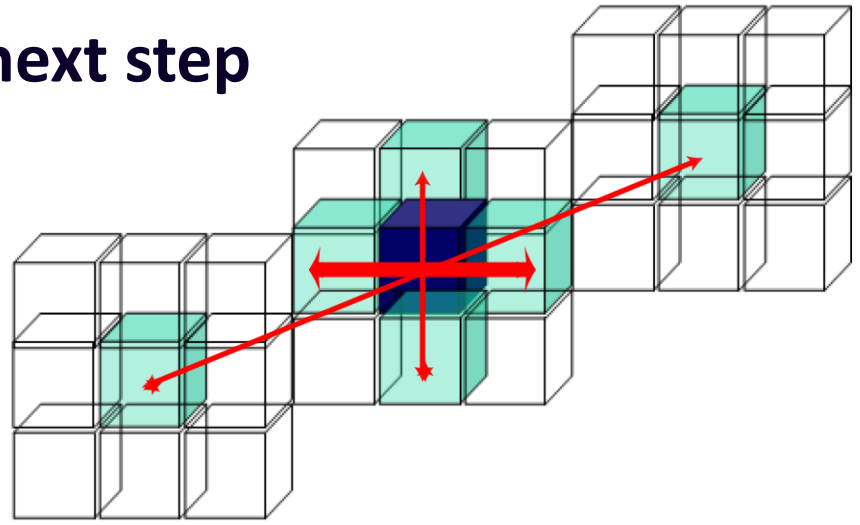


Discretization

For each position in the volume at time step *t* do:

- Approximate gradient and divergence with finite differences
- Calculate diffusivity *g()* to scale gradient
- Calculate for all directions and sum up
- Add to original density value

Use modified volume to calculate next step



Edge-Enhancing Anisotropic Diffusion (1)

$$\partial u_t = \mathrm{div}(D \cdot \nabla u)$$

Tensor *D*

- Tensor scales **and** rotates the flux
- The flux is no longer parallel to ∇u which maintains the location and strength of edges
- Preserves edges by smoothing parallel to edges instead across them

Edge-Enhancing Anisotropic Diffusion (2)

$$D = [v_1, v_2, v_3] \cdot \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \cdot \begin{bmatrix} v_1^T \\ v_2^T \\ v_3^T \end{bmatrix}$$

v_1, v_2, v_3 are the eigenvectors where

- $v_1 \| \nabla u_\sigma, v_2 \perp \nabla u_\sigma$ and $v_3 \perp \nabla u_\sigma$
- ∇u_σ gradient defined by a Gaussian kernel with width σ

$\lambda_1, \lambda_2, \lambda_3$ are the eigenvalues where

- $\lambda_2 = \lambda_3 = 1$ and λ_1 is one of the diffusivities *g()*

If $\lambda_1 = \lambda_2 = \lambda_3 = g()$ result is isotropic diffusion

Edge-Enhancing Anisotropic Diffusion (3)

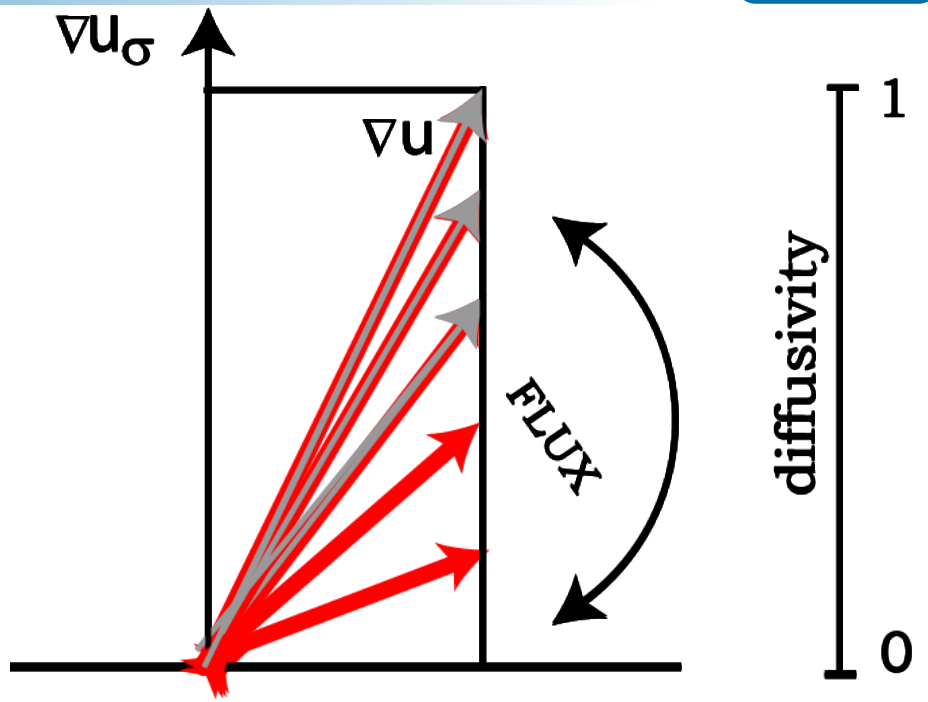
Homogeneous region:

- flux high

→flux proceeds along ∇u

Near edges:

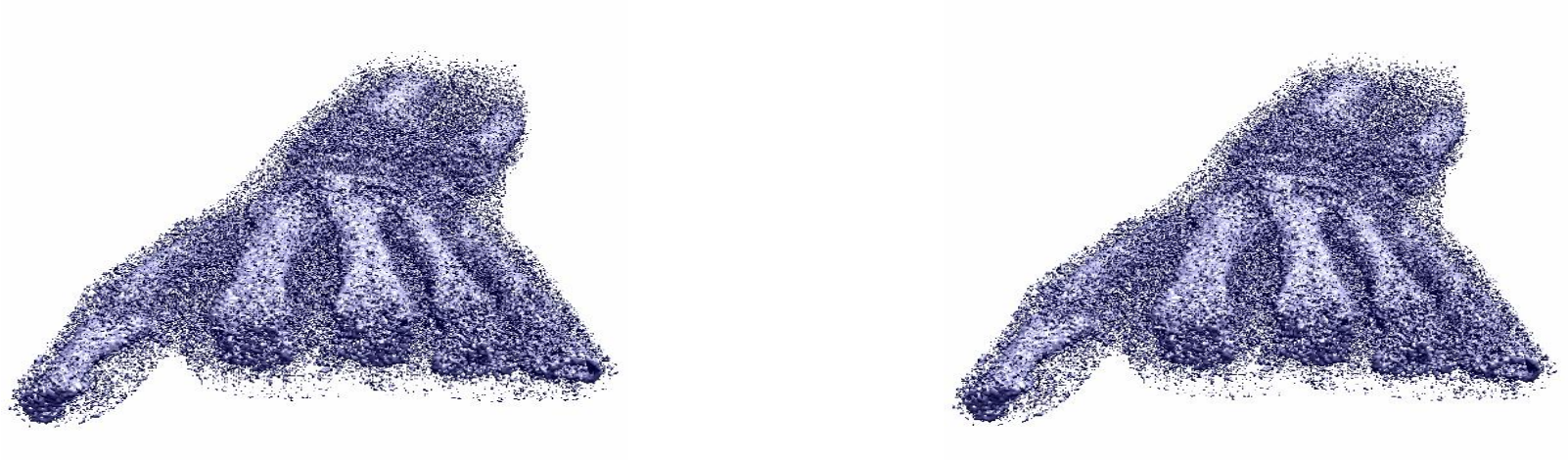
- ∇u scaled because of diffusivity
- flux decreases and rotates



∇u_σ has another orientation than the gradient ∇u at the same location from original density

→ **important for rotation of flux!**

Isotropic and Anisotropic Diffusion

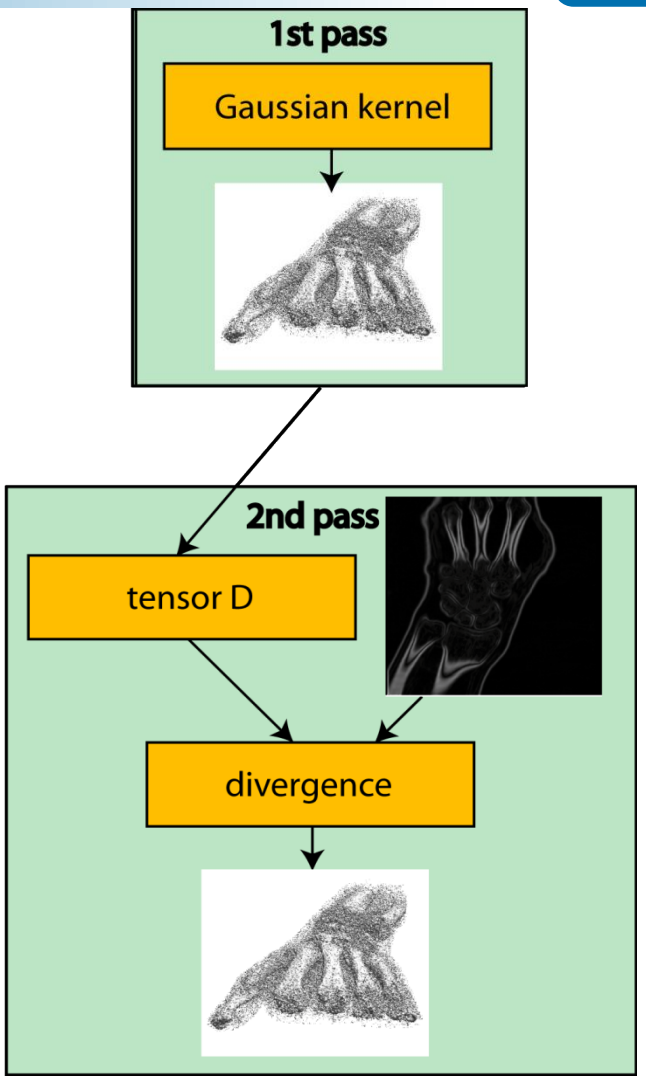


Algorithm (1)

For each slice of the volume:

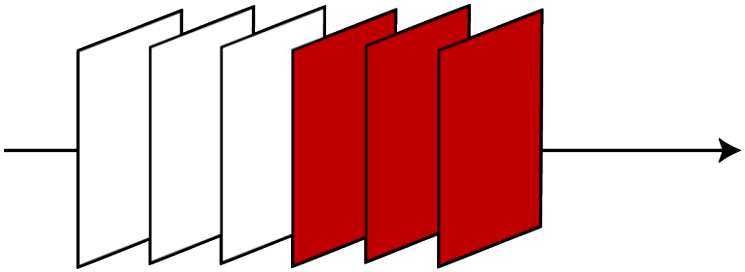
For each slice of the volume:

- 1st Pass:
  on-the-fly gradients calculated with Gaussian kernel
- 2nd Pass:
  calculate stencil to estimate divergence



Algorithm (2)

Smoothed gradients u_σ are calculated with Gaussian kernel



Gradient Slices:

- High values at edges (high intensity contrast)
- Low values in homogeneous regions

The more different $\nabla \mathrm{u}_\sigma$ are from $\nabla \mathrm{u}$ the more edge-stopping the function is

- gintensity and direction of flux controlled by Gaussian kernel

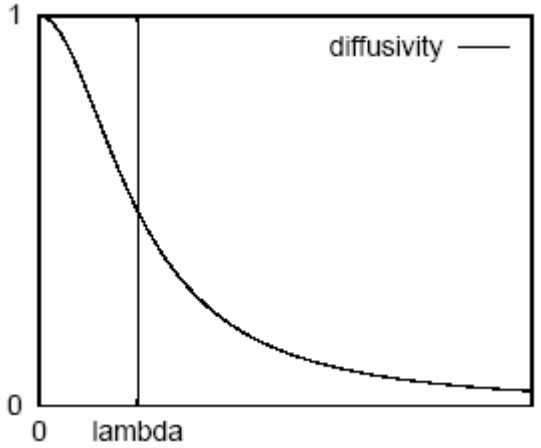
Diffusivity *g( )*

Perona-Malik: $$g(\| \nabla u \|) = e^{-(\|\nabla u\|^2 / \lambda^2)}$$

Tukey/Biweight: $$g(\| \nabla u \|) = \begin{cases} \frac{1}{2}\left[1 - \left(\frac{\| \nabla u \|}{\lambda}\right)^2\right]^2, & \| \nabla u \| \leq \lambda \\ 0 & otherwise \end{cases}$$

λ is the most critical value for image smoothing and segmentation.

- If λ too small for specific $\| \nabla u \|$ → small impact on volume
- If λ too high for specific $\| \nabla u \|$ → excessive blurring

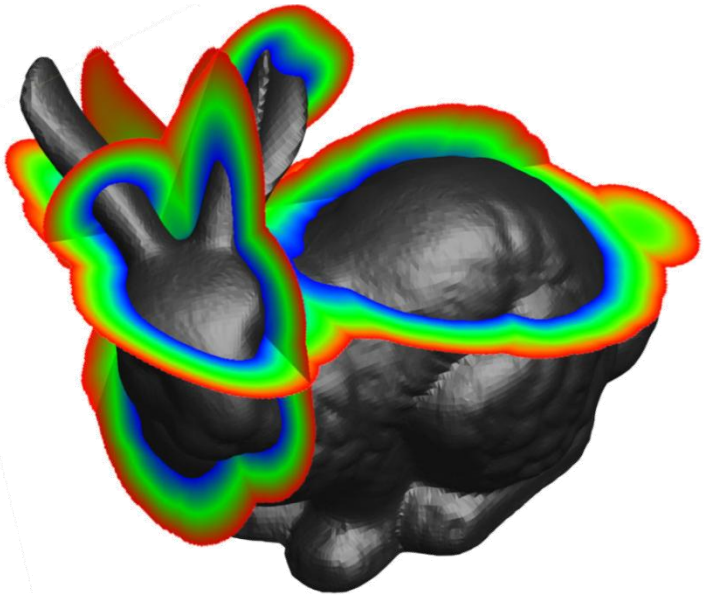


GPU Level-Sets (1)

- Additional volume: distance field

$$\phi : \mathbb{R}^3 \mapsto \mathbb{R}$$

$$S = \{\mathbf{x} | \phi(\mathbf{x}) = 0\}$$



- Solve PDE for every sample

$$\frac{\partial \phi}{\partial t} = F|\nabla \phi|$$

- Speed function F determines evolution/deformation

GPU Level-Sets (2)

- Hamilton-Jacobi equation (PDE)
- Discretized and solved for each point of grid
- $$\phi(t + \Delta t) = \phi(t) + \Delta t F |\nabla \phi|$$

$$\frac{\partial \phi}{\partial t} = F |\nabla \phi|$$

- Δt time step,
- $|\nabla \phi|$ gradient magnitude
- F speed function, determines deformation

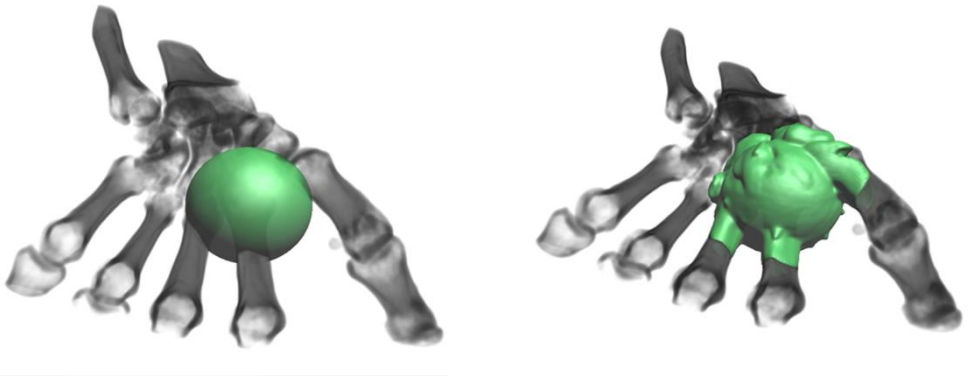
Volume Segmentation with Level-Sets

Motivation

- Interactive segmentation
- Simultaneous level-set computation and rendering
- Minimal memory use
- Compute in native format used by ray-caster

Both volumes can be bricked and use texture cache

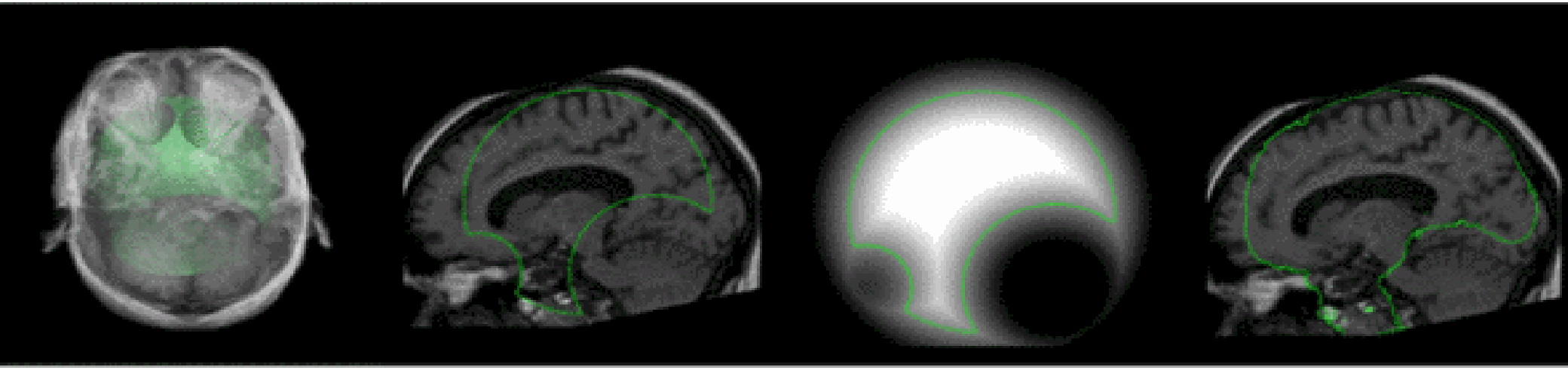
Render both volumes with correct intersection:



Volume Segmentation with Level-Sets

Motivation

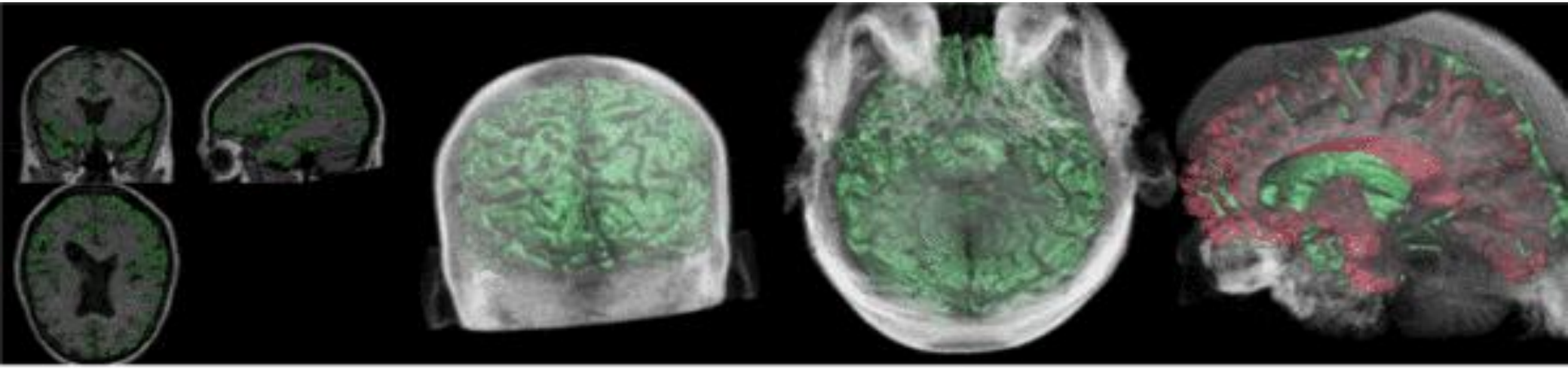
- Interactive segmentation
- Simultaneous level-set computation and rendering
- Minimal memory use (work directly on brick cache)
- Compute in native format used by ray-caster



Volume Segmentation with Level-Sets

Motivation

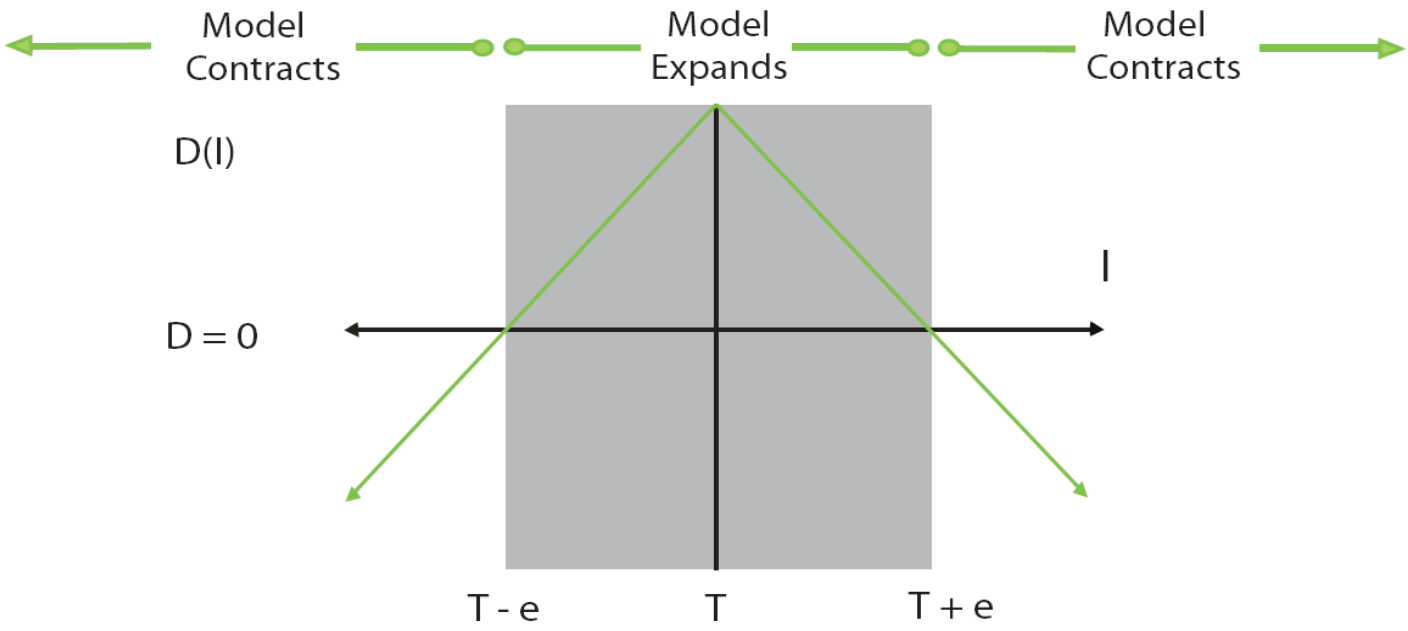
- Interactive segmentation
- Simultaneous level-set computation and rendering
- Minimal memory use (work directly on brick cache)
- Compute in native format used by ray-caster



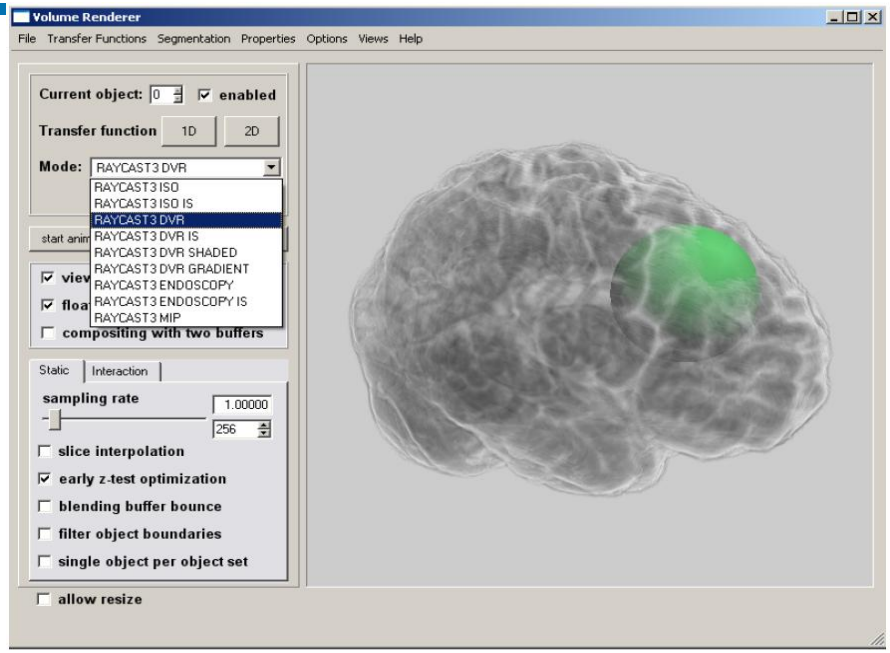
Segmentation: Density-Dependent Speed

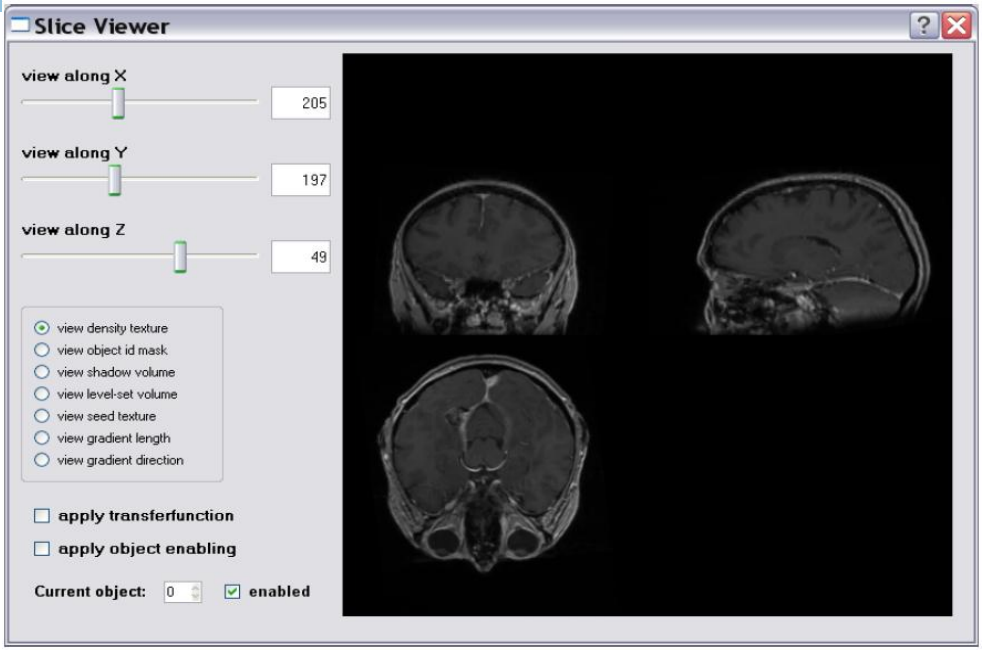
- Simple segmentation:

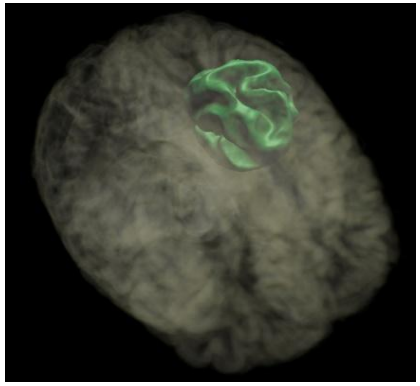
$$D = \epsilon - |I - T|$$

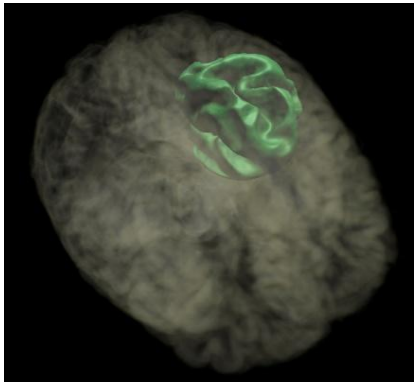


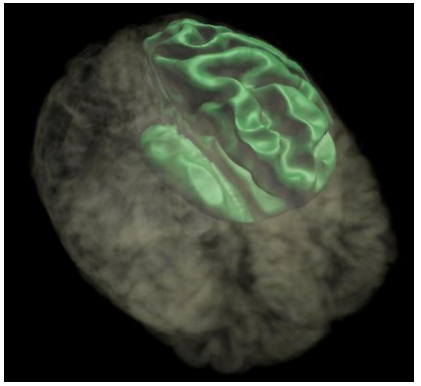
Example: Brain Segmentation

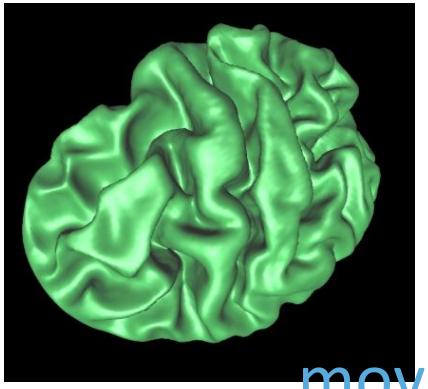












movie

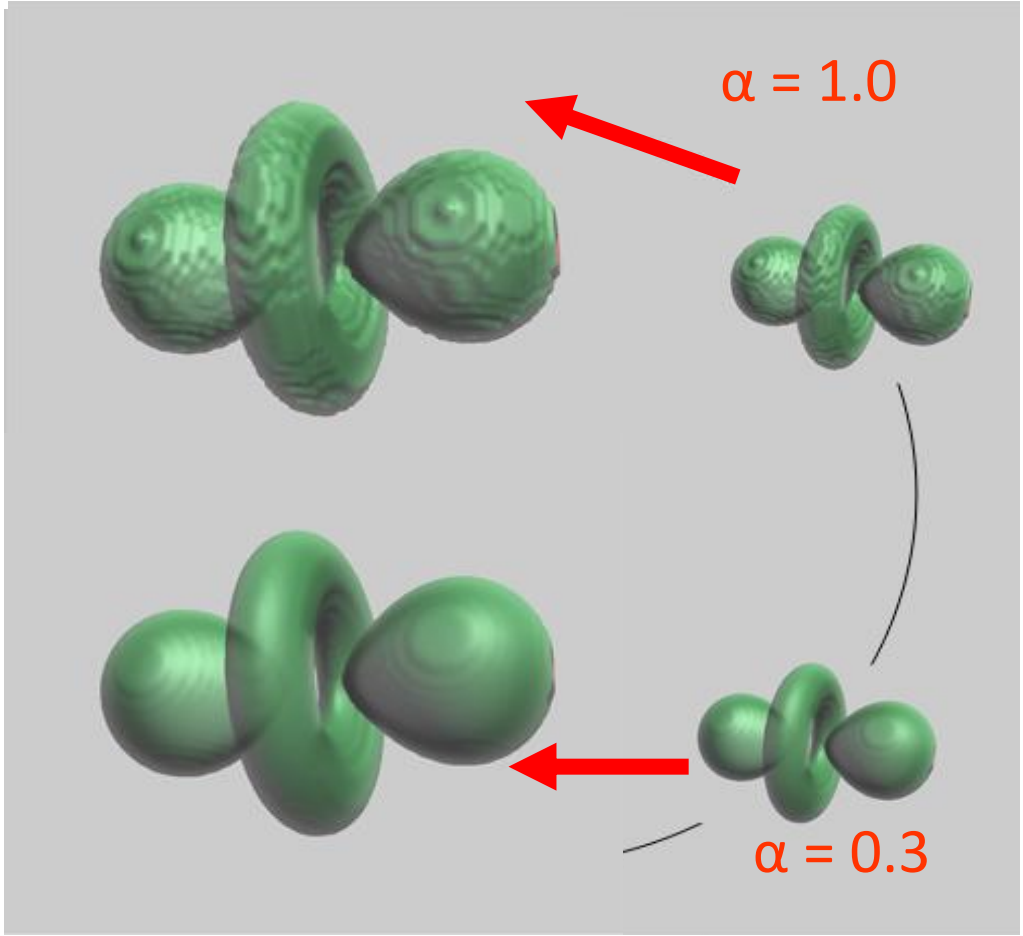
Plus Curvature Term

Density-dependent speed (D)

Mean curvature term (κ)

$$F = \alpha D + (1 - \alpha)\kappa$$

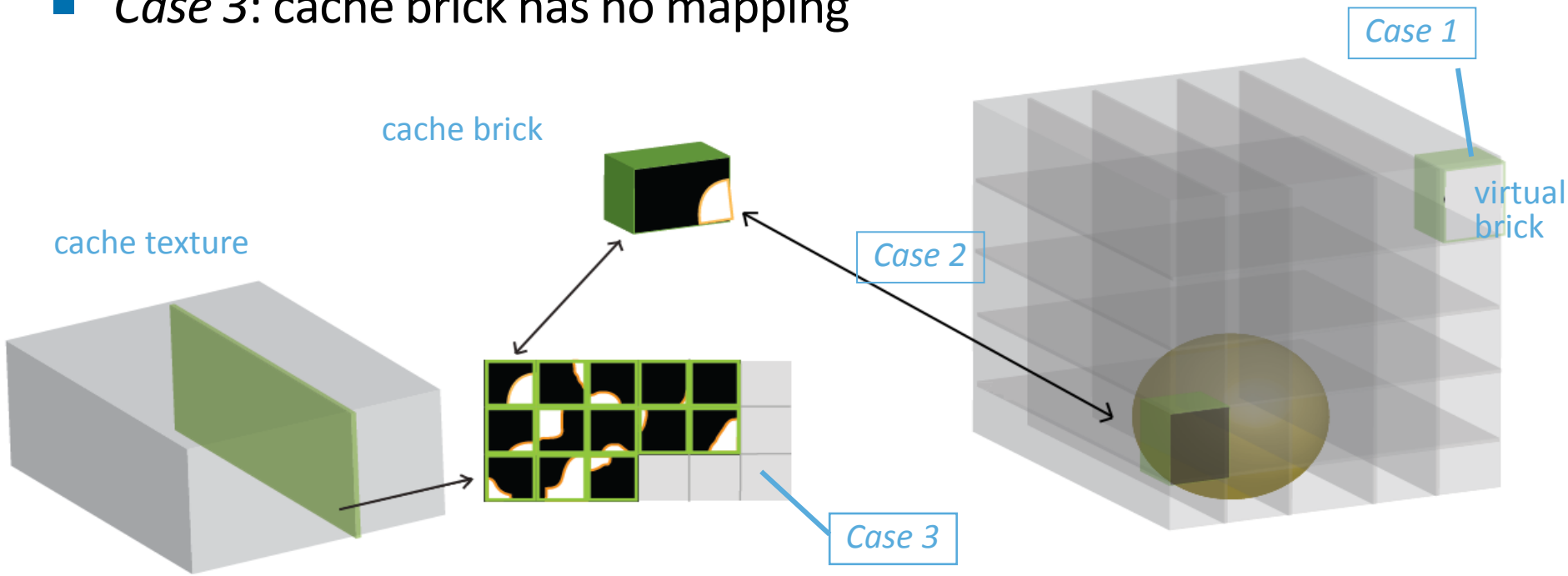
- Blend density term (D) and curvature term (κ)



Level-Set Cache

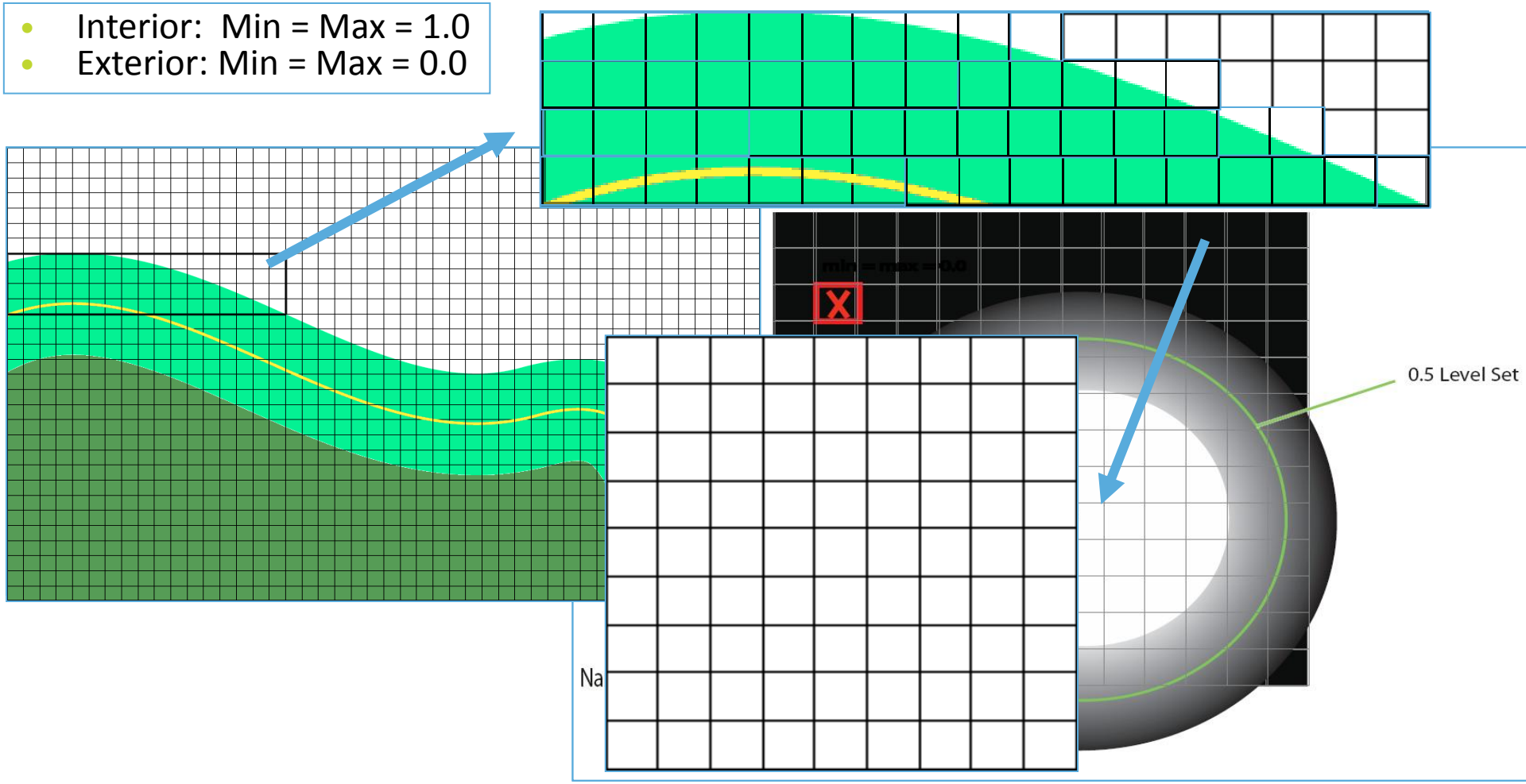
Sparse virtual volume for computation

- *Case 1*: virtual brick has no mapping
- *Case 2*: virtual brick maps to physical (cache) brick
- *Case 3*: cache brick has no mapping

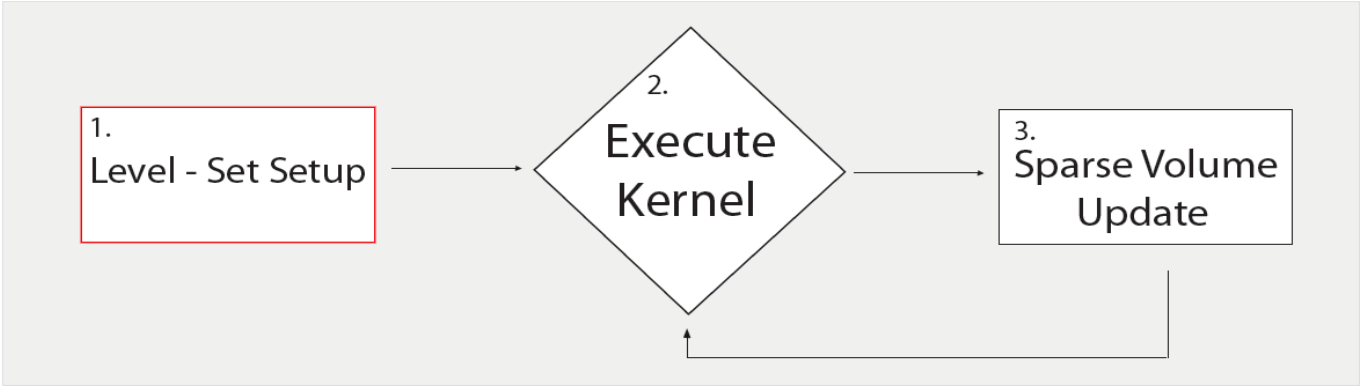


Narrow Band Culling and Packing

- Interior: Min = Max = 1.0
- Exterior: Min = Max = 0.0

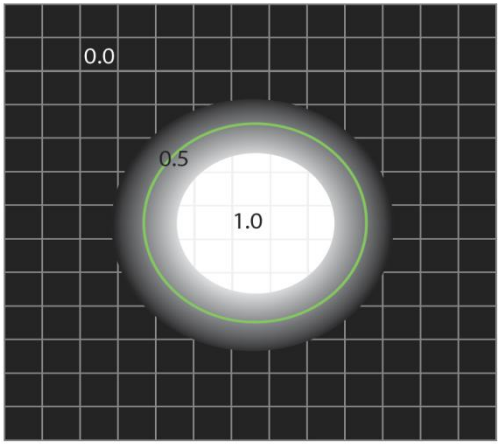


Level-Set Solver (1)

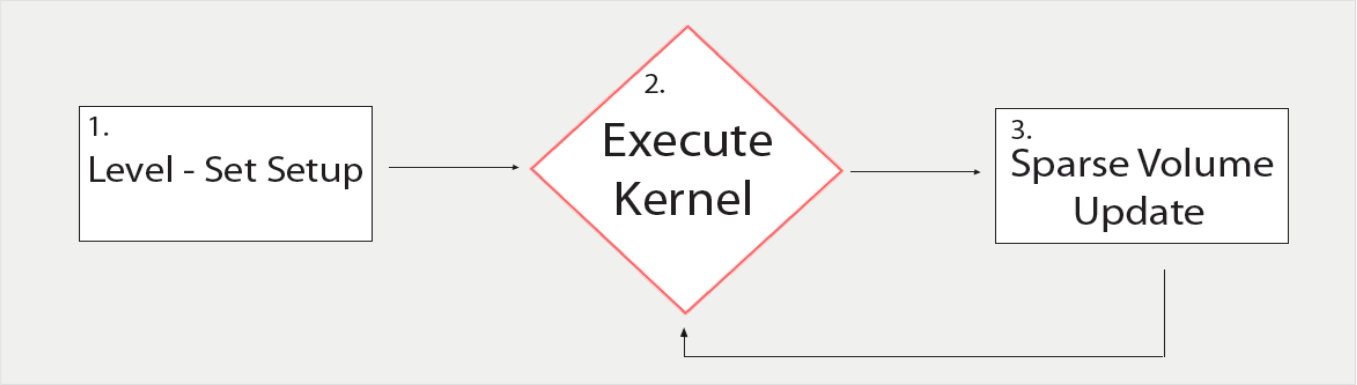


1. Setup

- Init signed distance field
  - Narrow band is [0.0, 1.0]
  - Perform initial culling
  - Allocate initial cache bricks
  - Compute clamped distances in active cache bricks



Level-Set Solver (2)



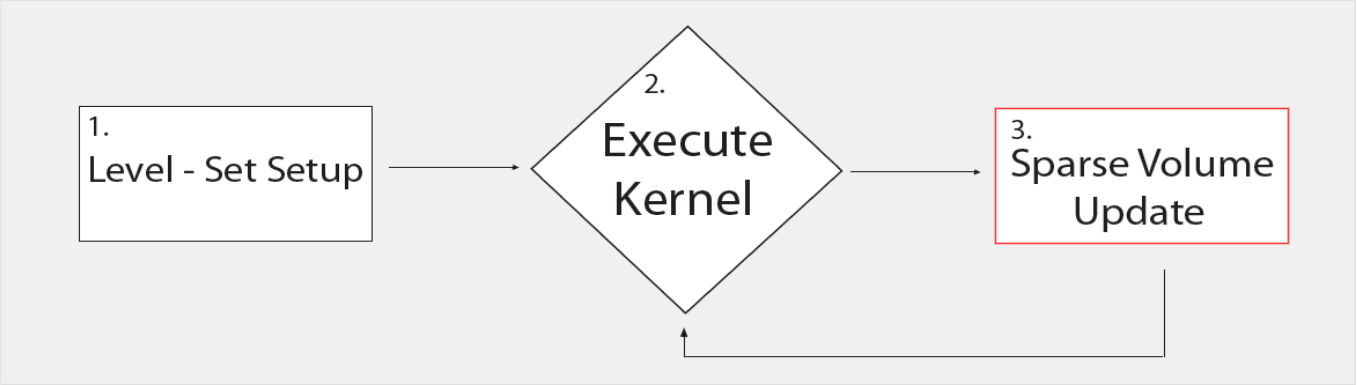
2. Level-set computation

- PDE solved in fragment program

$$\phi(t + \Delta t) = \phi(t) + \Delta t F |\nabla \phi|$$

- Need neighborhood for finite differences
- Upwinding and mean curvature computation

Level-Set Solver (3)

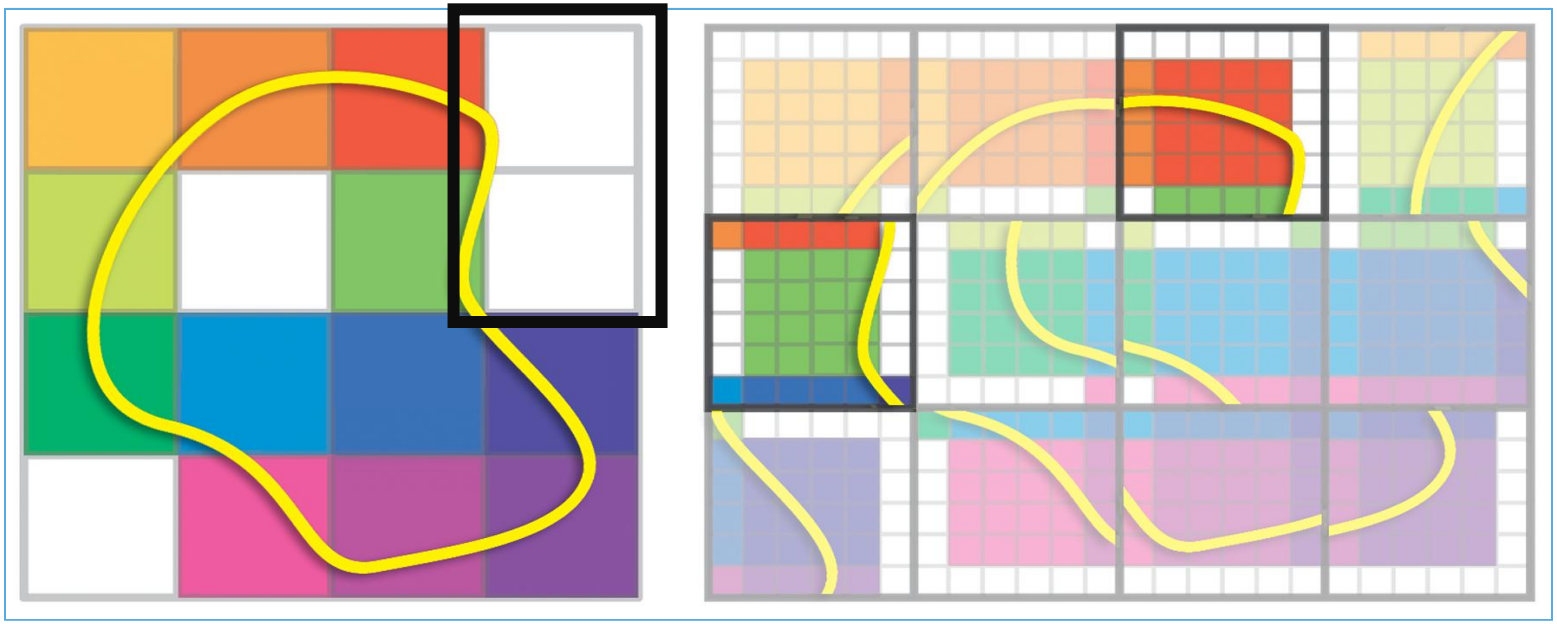


3. Sparse volume update

- GPU – CPU memory request
  - Determine brick activation/deactivation & encode
- CPU – GPU memory allocation
  - Decode brick activation/deactivation
  - Allocate/deallocate physical cache bricks

Neighbor Activation (1)

- Determine “leaking” of narrow band
- Min/max computation for six brick faces
- Encode in 6 bits and transmit to CPU



Thank You!

- Markus Hadwiger
- Michael Su
- Greg Turk
- Mark Harris
- Laura Fritz
- Oliver Klar
- Veronika Solteszova