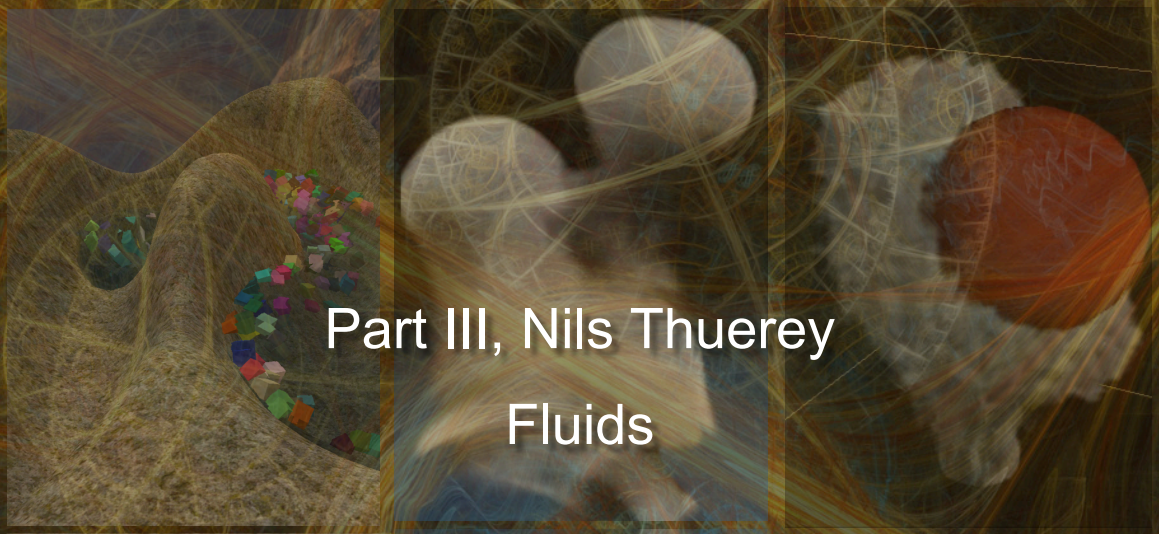
Real-time Physics



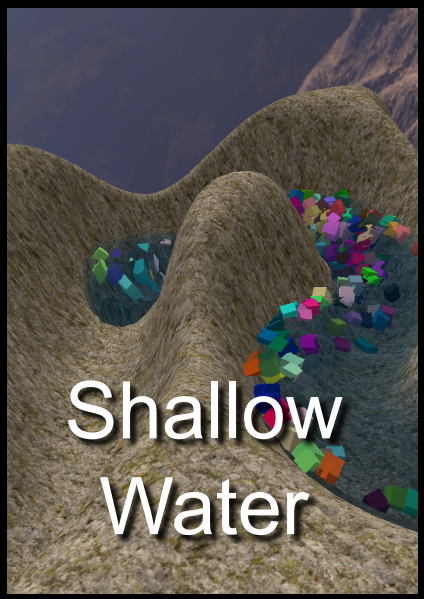
Part III, Nils Thuerey

Fluids

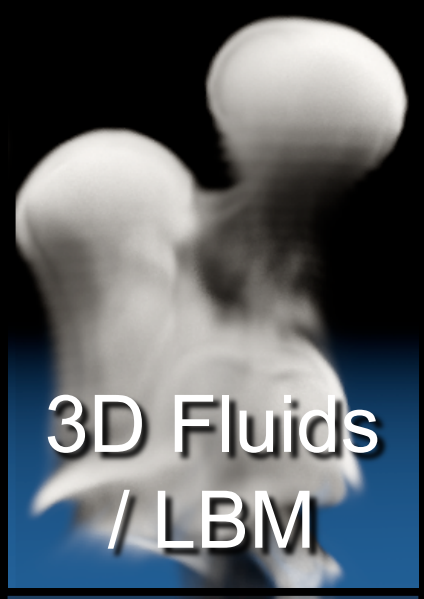
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

cgl
Computer Graphics Laboratory ETH Zurich

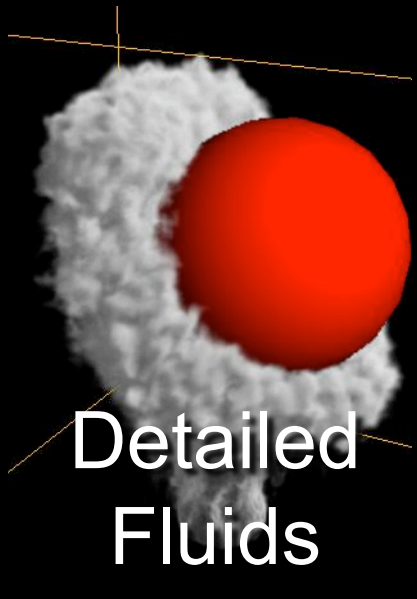
Overview



Shallow Water



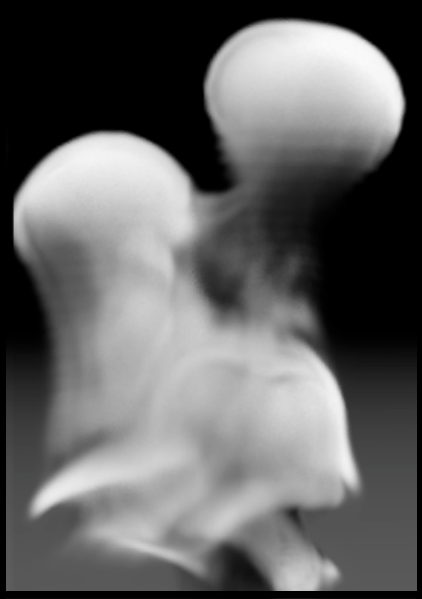
3D Fluids / LBM

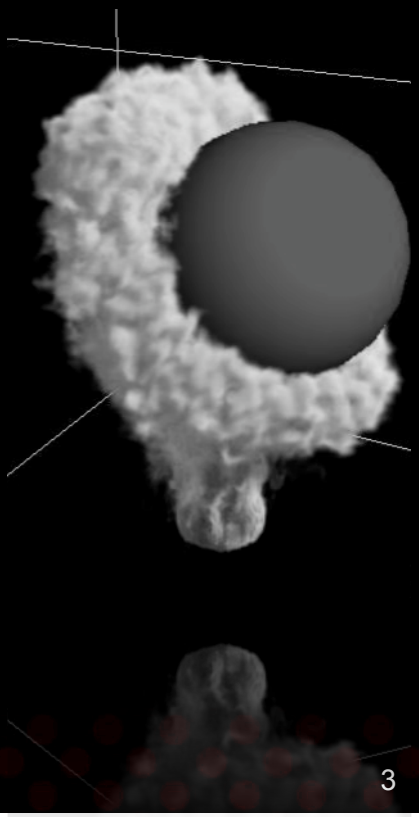


Detailed Fluids

Overview

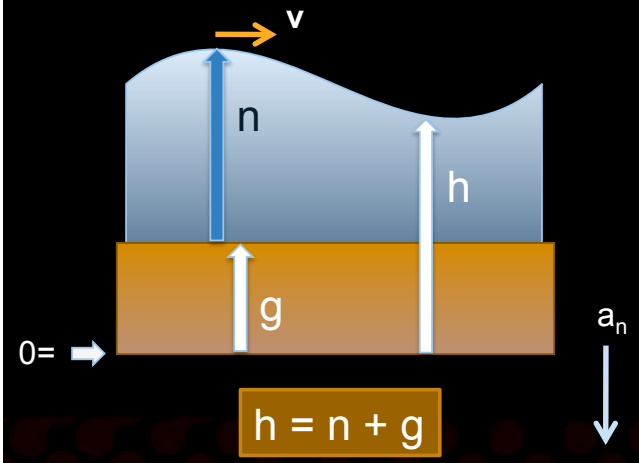
- Basic Solver
- Boundaries
- Rigid Bodies
- Terrain





Shallow Water Equations

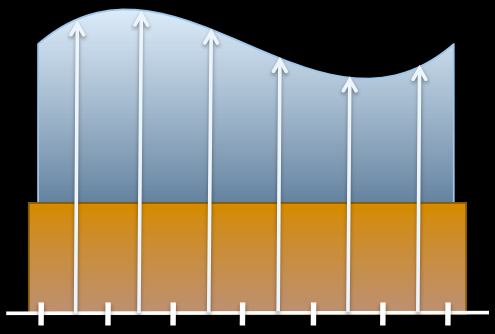
- Derived from the Navier-Stokes equations
- Simulate gravity waves on a fluid surface

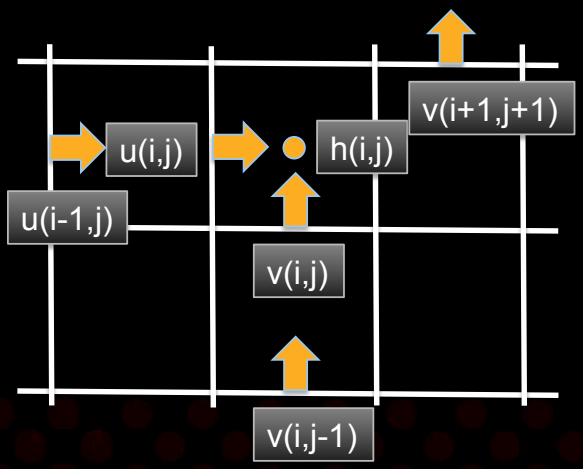


$$\frac{\partial \eta}{\partial t} + (\nabla \eta)\mathbf{v} = -\eta \nabla \cdot \mathbf{v}$$
$$\frac{\partial \mathbf{v}}{\partial t} + (\nabla \mathbf{v})\mathbf{v} = a_n \nabla h \,,$$

Discretization

- Staggered grid, fixed time step
- Compute derivatives with finite differences





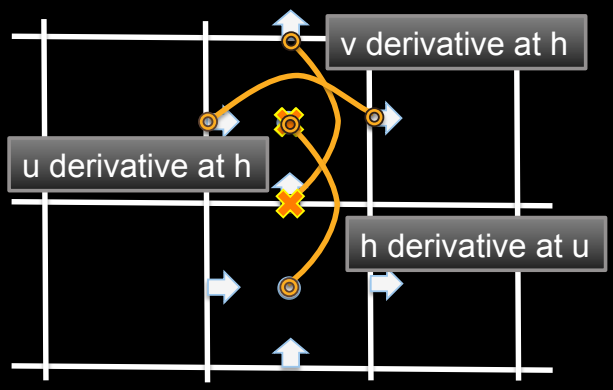
Basic algorithm

- Operator splitting
  - Compute advection
  - Then acceleration terms
- Algorithm
  - Advect *n*
  - Advect v1
  - Advect v2
  - Update height
  - h = n+g
  - Update velocities

$$\begin{aligned} \partial\eta/\partial t + (\nabla\eta)\mathbf{v} &= -\eta\nabla\cdot\mathbf{v} \\ \partial v_1/\partial t + (\nabla v_1)\mathbf{v} &= a_n\nabla h \\ \partial v_2/\partial t + (\nabla v_2)\mathbf{v} &= a_n\nabla h\,. \end{aligned}$$

Discretization

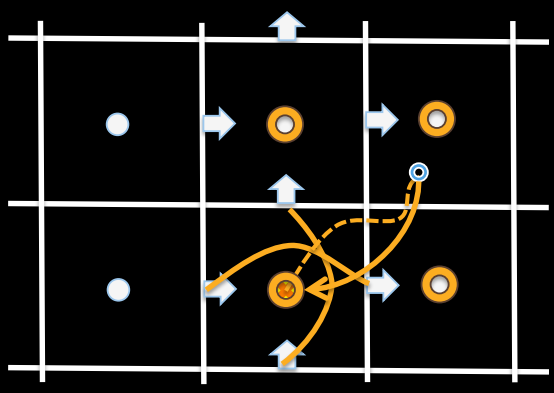
- Staggered grid
- Compute derivatives with finite differences



Advection

$$\partial\eta/\partial t + (\nabla\eta)\mathbf{v} = -\eta\nabla\cdot\mathbf{v}$$
$$\partial v_1/\partial t + (\nabla v_1)\mathbf{v} = a_n\nabla h$$
$$\partial v_2/\partial t + (\nabla v_2)\mathbf{v} = a_n\nabla h\,.$$

- Semi-Lagrange step, e.g. for h:
  - Interpolate velocities
  - Backtrace
  - Interpolate
  - Write
- Repeat 3x
  - *n*, v1, v2

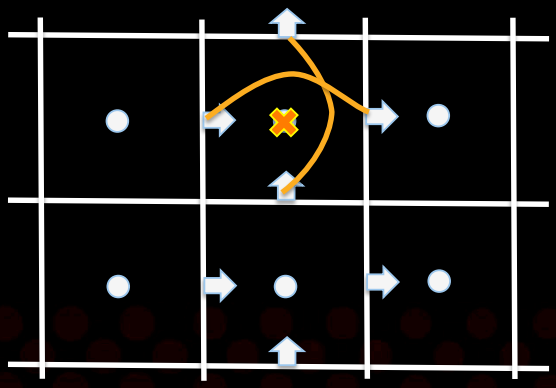


Accelerations

$$\partial\eta/\partial t + (\nabla\eta)\mathbf{v} = -\eta\nabla\cdot\mathbf{v}$$
$$\partial v_1/\partial t + (\nabla v_1)\mathbf{v} = a_n\nabla h$$
$$\partial v_2/\partial t + (\nabla v_2)\mathbf{v} = a_n\nabla h\,.$$

- Height update:
  - For all cells:

$$-\eta(i,j)\cdot\left(\frac{(v_1(i{+}1,j)-v_1(i,j))}{\Delta x}+\frac{(v_2(i,j{+}1)-v_2(i,j))}{\Delta x}\right)\Delta t$$

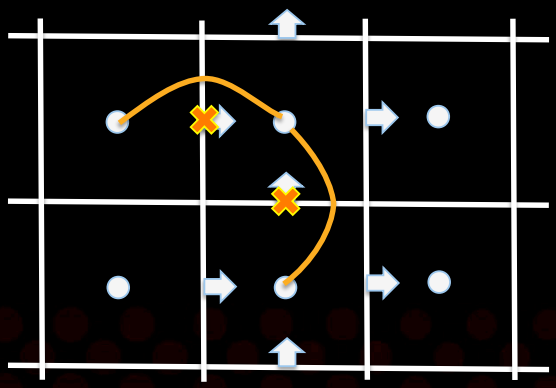


Accelerations

$$\partial\eta/\partial t + (\nabla\eta)\mathbf{v} = -\eta\nabla\cdot\mathbf{v}$$
$$\partial v_1/\partial t + (\nabla v_1)\mathbf{v} = a_n\nabla h$$
$$\partial v_2/\partial t + (\nabla v_2)\mathbf{v} = a_n\nabla h\,.$$

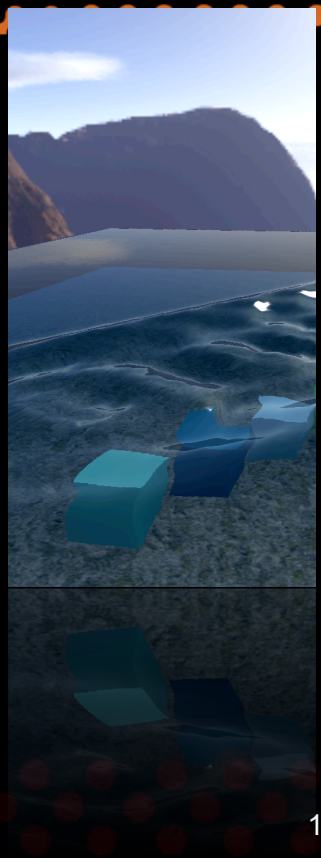
- Velocity update:
  - For all cells:
  - Here: a = gravity

$$v_1'(i,j) = a\left(\frac{h(i{-}1,j)-h(i,j)}{\Delta x}\right)\Delta t$$
$$v_2'(i,j) = a\left(\frac{h(i,j{-}1)-h(i,j)}{\Delta x}\right)\Delta t$$



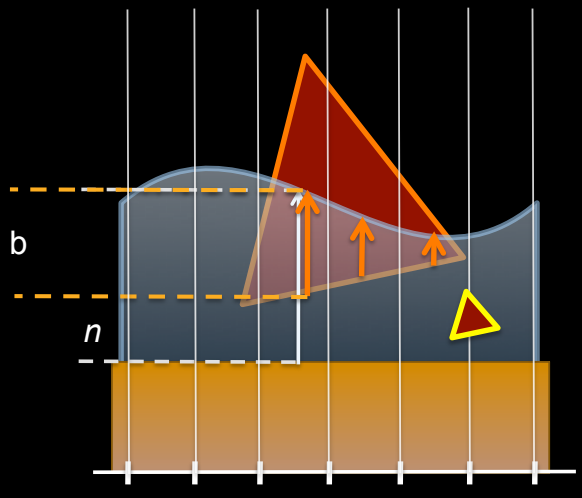
Discussion

- Stability:
  - Explicit method, not always stable
  - Stable: implicit update
  - In practice: clamp…
- Advection can be left out
- Sharper waves than with wave equation
- Example source code on the course page



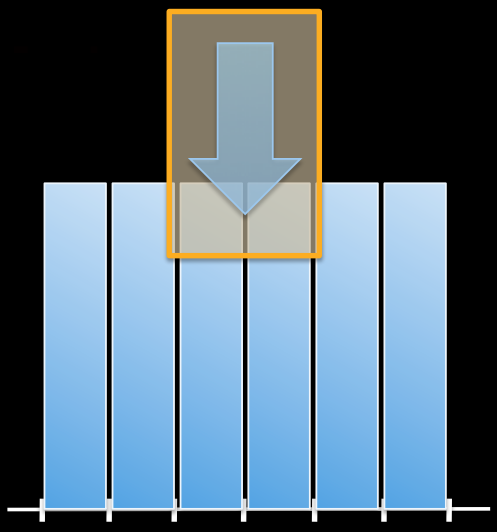
Rigid Body Coupling

- For each covered cell:
  - Immersion depths $b(\mathbf{x})$
  - Change over time
  - Distribute to NBs
  - Object → Fluid
  - Sum up forces
    - Relative fluid velocity
    - Buoyancy (from b)
- Volume preserving
- Small objects: project



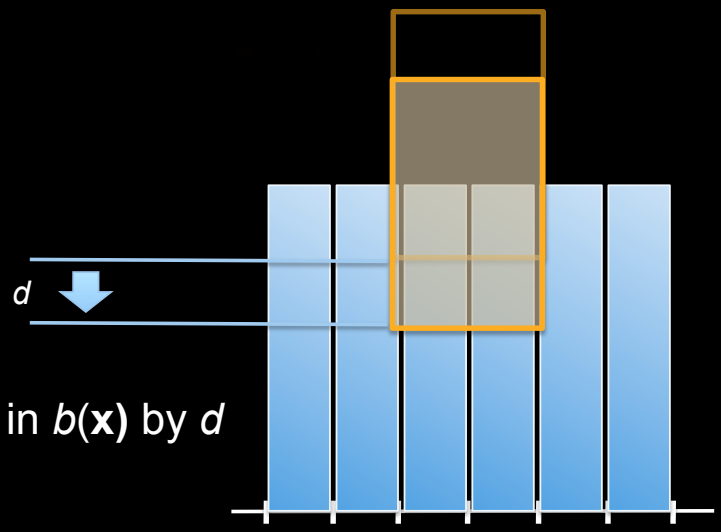
Rigid Body Example

Move down -
Increase volume



Time *t*

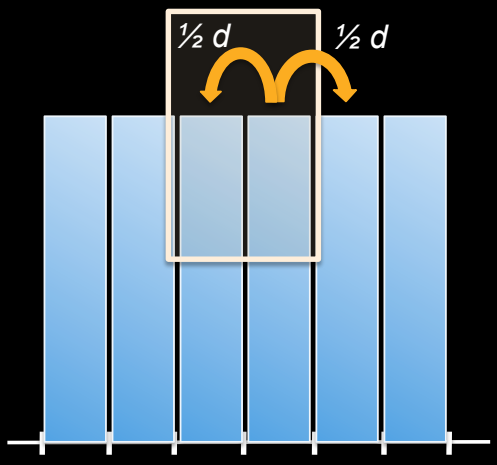
Rigid Body Example



Change in $b(\mathbf{x})$ by d

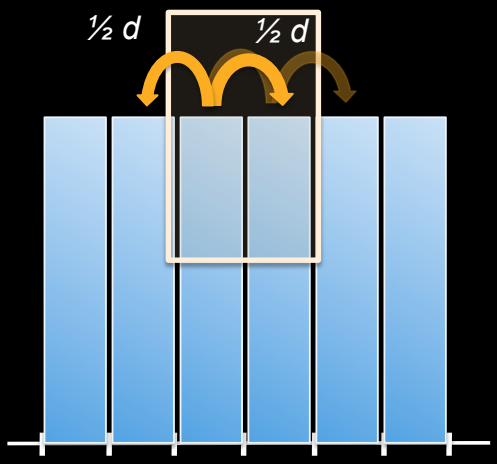
Time *t+1*

Rigid Body Example

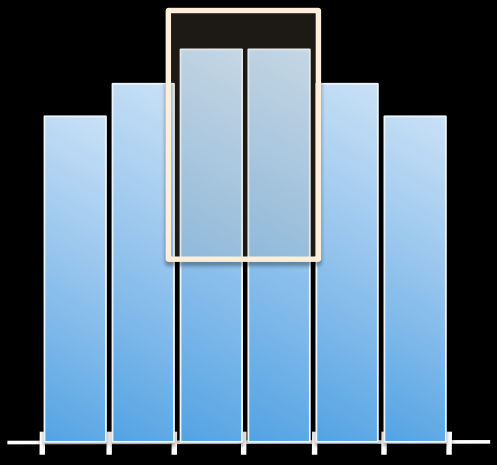


Distribute displacement

Rigid Body Example



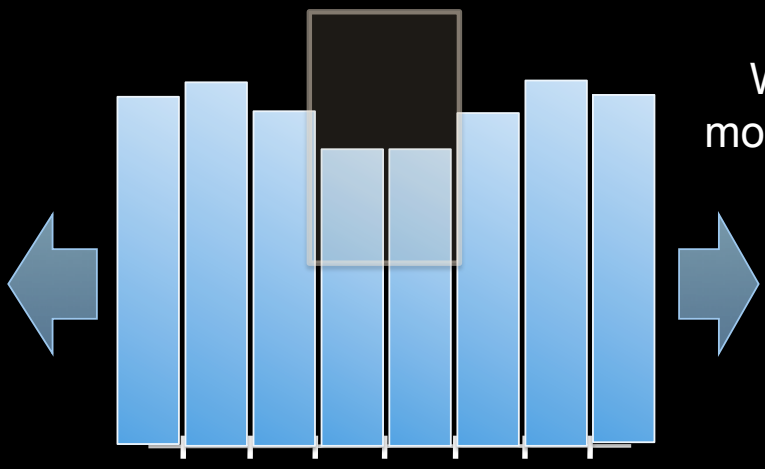
Rigid Body Example



Resulting surface

Time *t+1*

Rigid Body Example

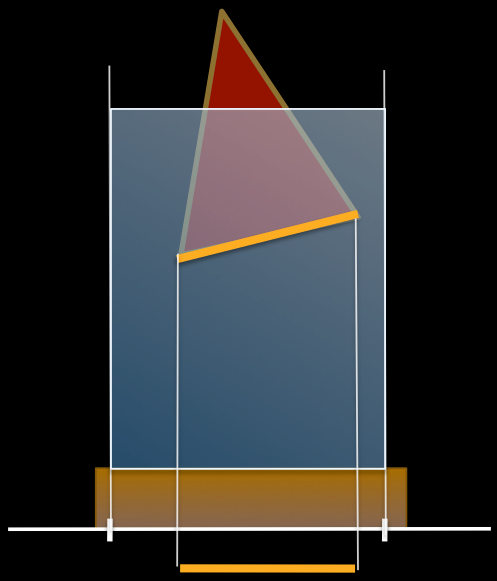


Wave will move outward

Time *t+1*

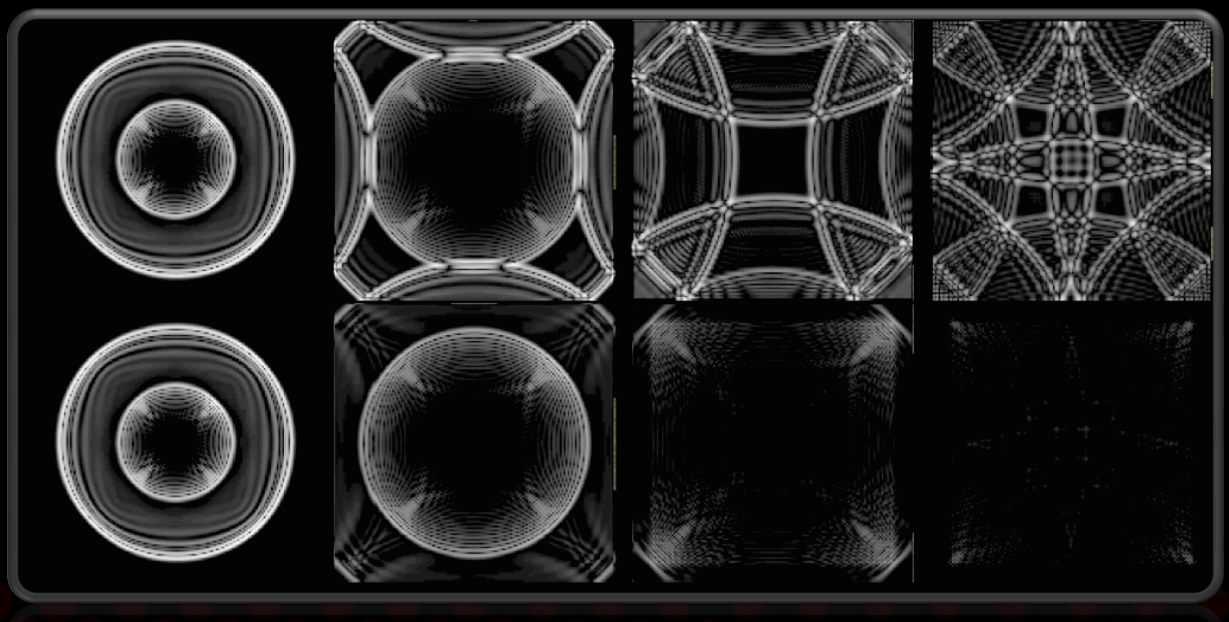
Rigid Body Coupling

- Volume preserving
- For small objects:
  - Project faces to grid
  - Again: compute forces based on displaced volume



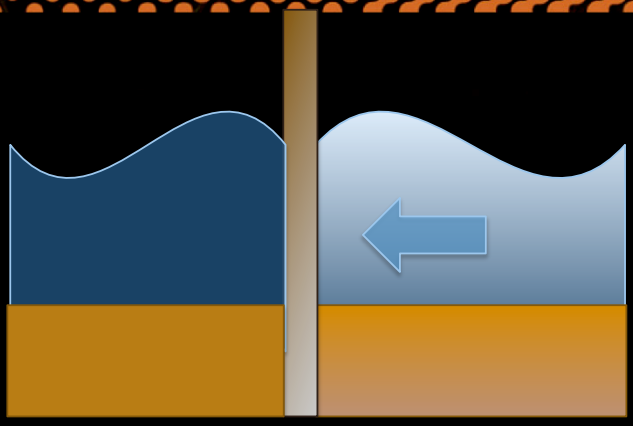
Boundaries

- Top-down example:



Boundaries

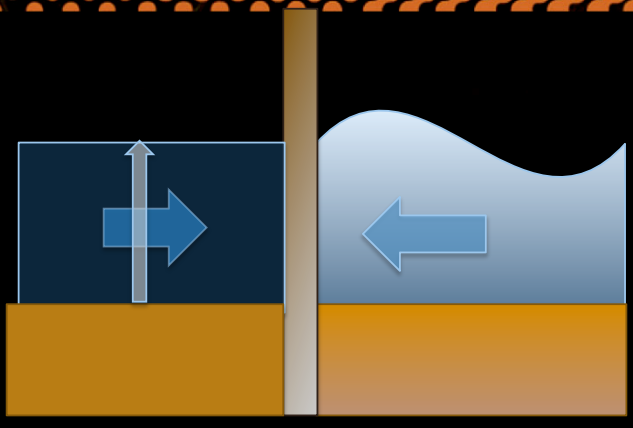
- Reflecting



Reflection:
Mirror Height

Boundaries

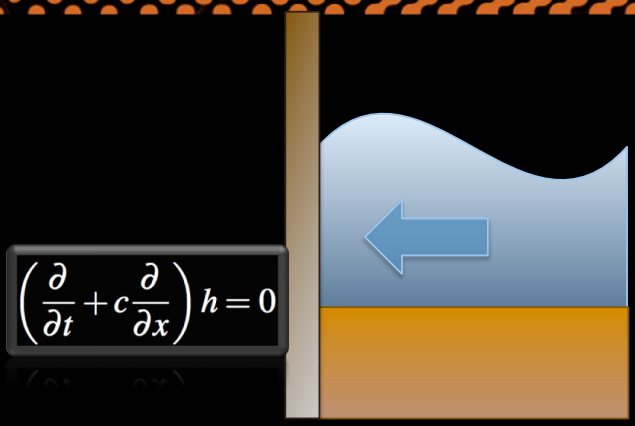
- Reflecting
- In/out flow



In/out flow:
Specify values

Boundaries

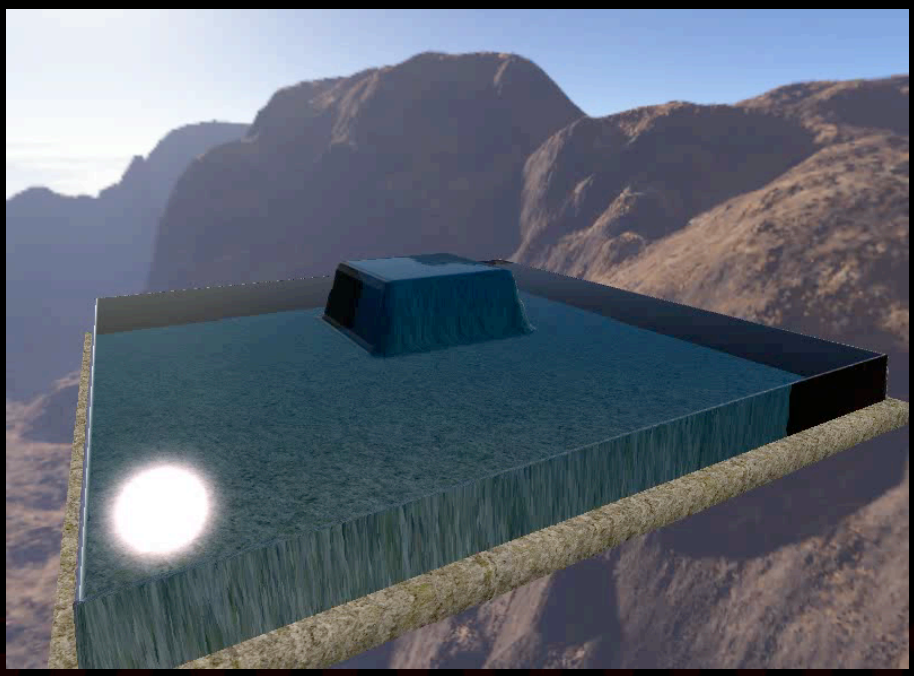
- Reflecting
- In/out flow
- Absorbing



Absorbing:
More complicated…

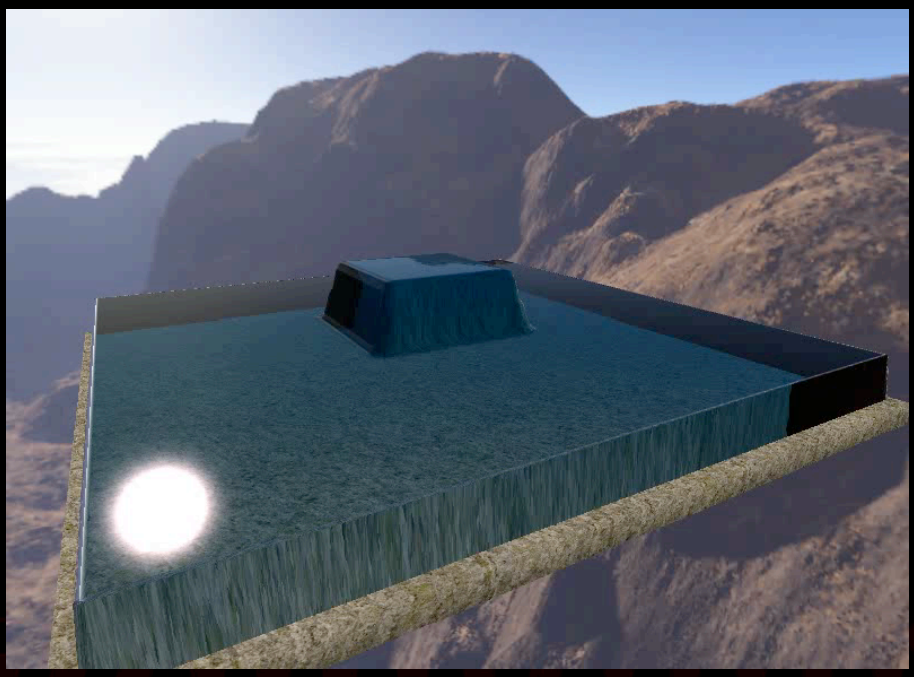
Boundary Examples

- Reflecting



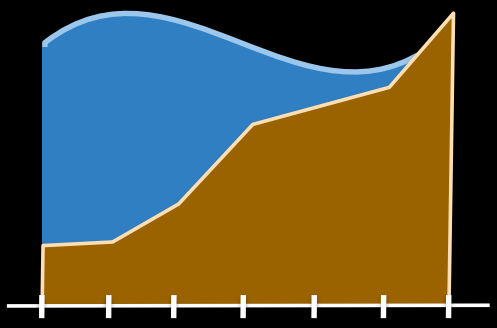
Boundary Examples

- Absorbing

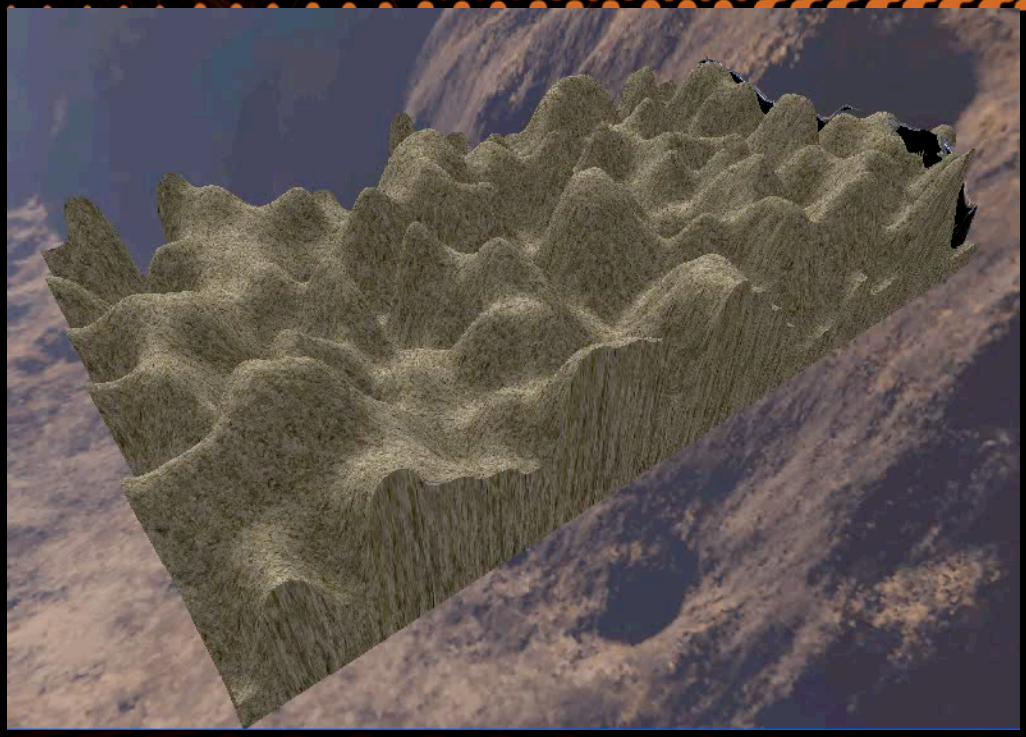


Free surfaces / terrain

- Track covered area
- Prevent negative depths
- For rendering
  - Compute surface position
  - Construct front line



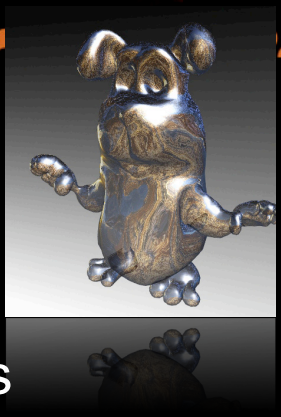
Free surfaces / terrain example

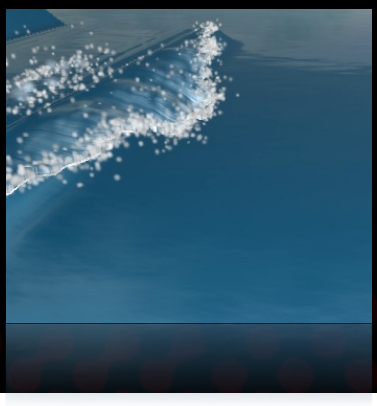


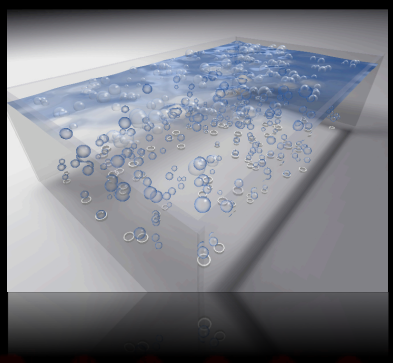
Terrain & rigid bodies example

Outlook

- 3D effects - bubbles
- Terrain – erosion
- SWS on meshes – fluid characters





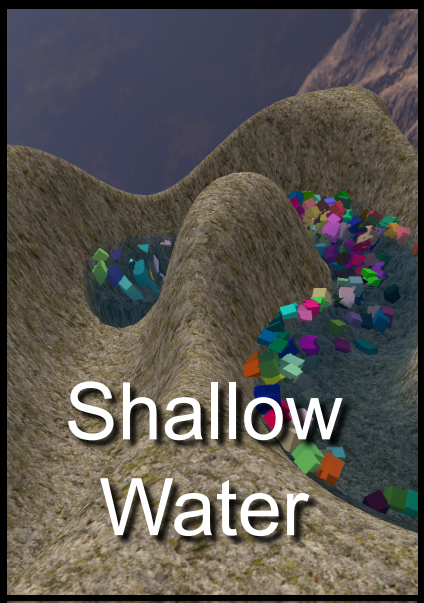


Shallow Water Simulations

- Thanks to: P. Hess, R. Angst, M. Mueller

- References (among others):
  - M. Kass &G. Miller: Rapid, Stable Fluid Dynamics for Computer Graphics, 1990
  - A. Layton & M. van der Panne: A Numerically Efficient and Stable Algorithm for Animating Water Waves, 2002
  - R. Bridson: Lecture on Animation Physics, 2005
  - X. Mei & P. Decaudin & B.-G. Hu: Fast Hydraulic Erosion Simulation and Visualization on GPU, 2007

Overview

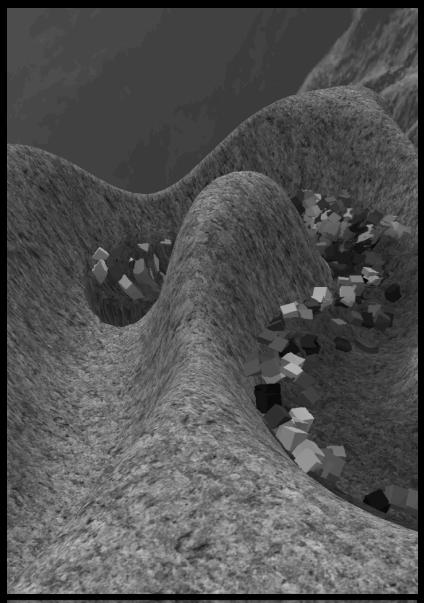


Shallow Water

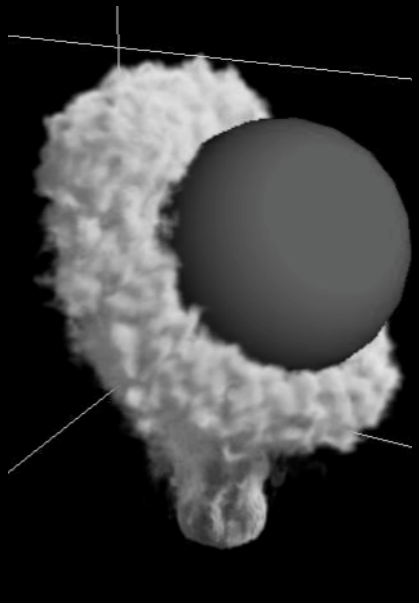
3D Fluids / LBM

Detailed Fluids

Overview



- Navier-Stokes
- LBM
- Implementation



3D Fluids

- Solve the full *Navier-Stokes* equations:

$$\underbrace{\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u}}_{\text{advection}} = \underbrace{-\frac{1}{\rho}\nabla p}_{\text{pressure}} + \underbrace{\nu \triangle \mathbf{u}}_{\text{viscosity}} + \mathbf{g}$$

(momentum equation)

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

(continuity equation)

3D Fluids

- Typical solver:
  - Advection (e.g. semi-Lagrange step)
  - Add forces (gravity, etc.)
  - Make divergence free (iterative solve)
- Full details in, e.g.:
  - *Bridson & Mueller;* SIGGRAPH course *"Fluid Simulation"*

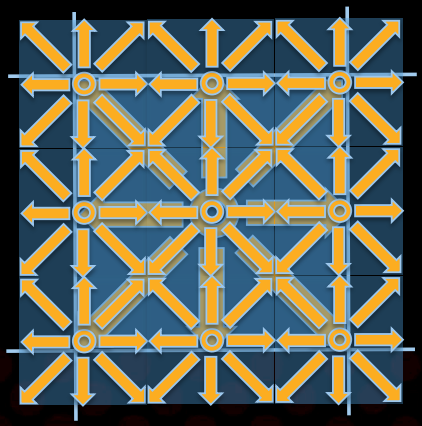
Lattice-Boltzmann Method (LBM)

- Alternative to standard solver
- Properties:
  - Solves the NS equations
  - Explicit time step (slightly compressible)
  - Grid based
  - Single loop
  - Highly parallelizable

L. Boltzmann
1844-1906

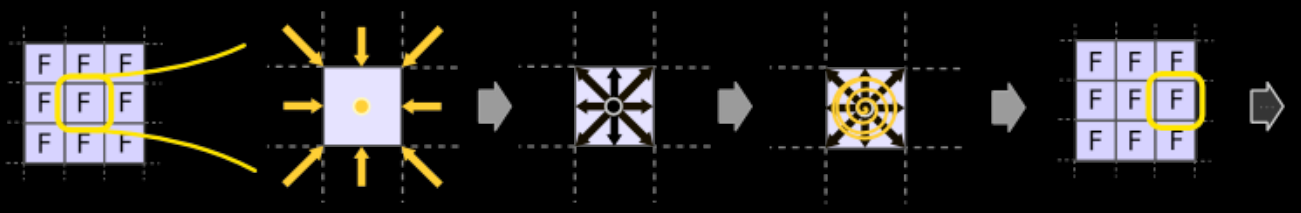
Setup

- Equi-distant Cartesian grid
- In 2D: 9 *distribution functions* (DFs) per cell
- = 9 floats, each represents fluid volume
- 9 fixed velocities …



…

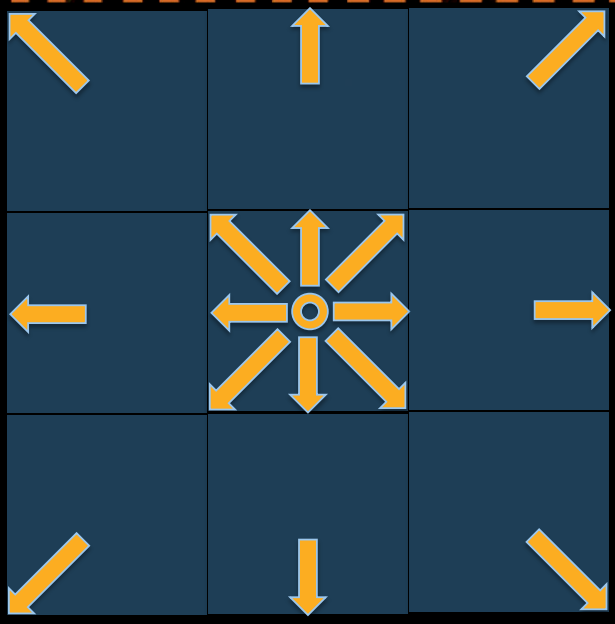
Overview – basic algorithm



- For all cells:
  - Stream – copy neighboring DFs
  - Collide – compute collisions among particles

Streaming

- Computes the advection
- Velocities are normalized
- Pure copying of values



Collision

- Accounts for all the dynamics…
- Compute
  - Density = sum of all DFs
  - Velocity = sum of all DFs x velocity vector
- Compute 9 equilibrium DFs
- Weight old DFs and equilibrium
  - Viscosity effects

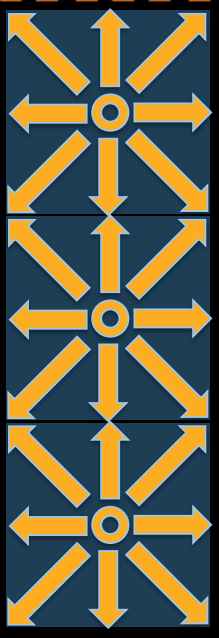
$$f_i^{eq} = w_i \left[\rho + 3\mathbf{e_i} \cdot \mathbf{u} - \frac{3}{2}\mathbf{u}^2 + \frac{9}{2}(\mathbf{e_i} \cdot \mathbf{u})^2 \right]$$

Implementation

- Allocate 2 grids
- Loop over grid
  - 9x copy DFs
  - 9x sum DFs for density & velocity
  - 9x compute equilibrium
  - 9x weight old & new values
  - Write to other grid…

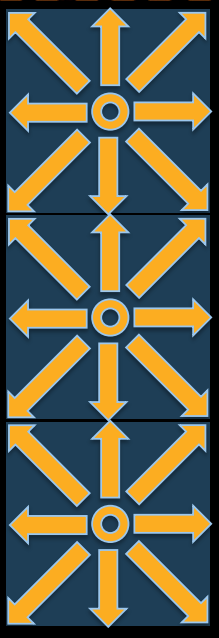
Discussion

- Solves the NS equations!
- 3D: 19 instead of 9 velocities
- Restriction
  - Time step: velocity < 1/3
  - Viscosity rel. high
- Low viscosity -> use turbulence model



Discussion

- Source code available:
  - Basic 3D solver
  - Main loop kernel: 42 lines of code
    - Nicely formatted
    - With comments
    - Including obstacle & acceleration handling
  - Support for Smagorinsky turbulence model
  - OpenMP support
  - … but not interactive



Lid driven cavity

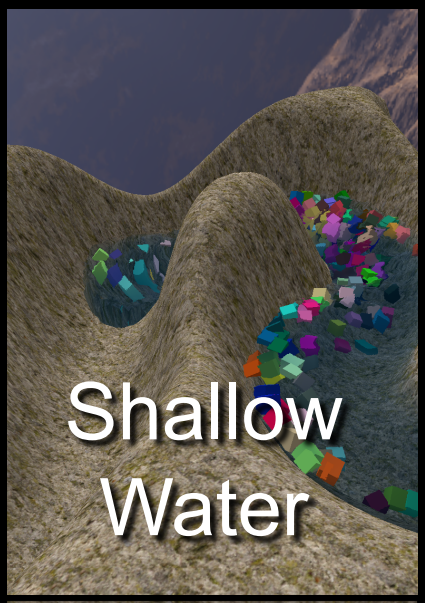
- Standard test setup:
  - Closed box with no-slip walls
  - Top “lid” is moving with fixed velocity

Extensions

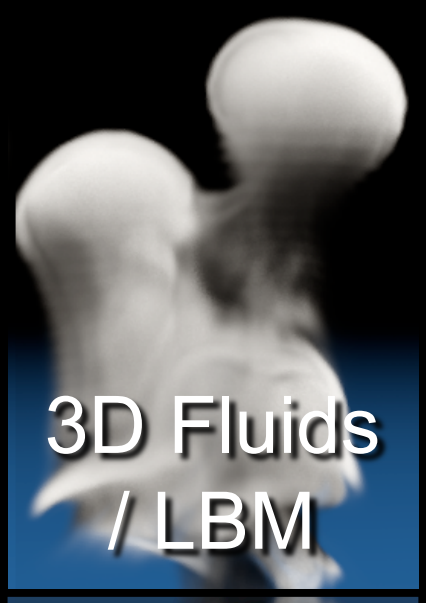
- Density advection
  - Add semi-Lagrange step
  - E.g. for smoke / fire

- Free surfaces
  - Add surface tracking
  - More complicated…
  - Try out in Blender!

Overview



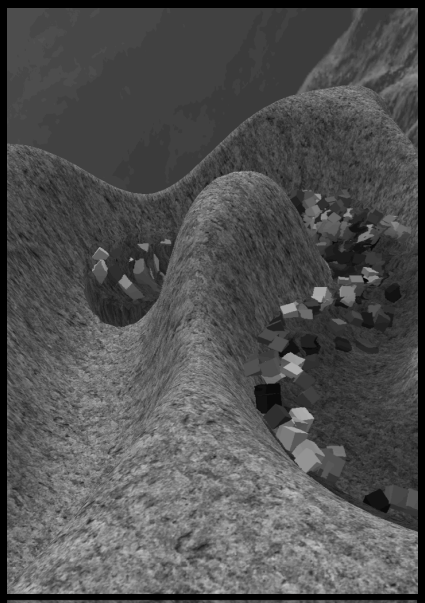
Shallow Water

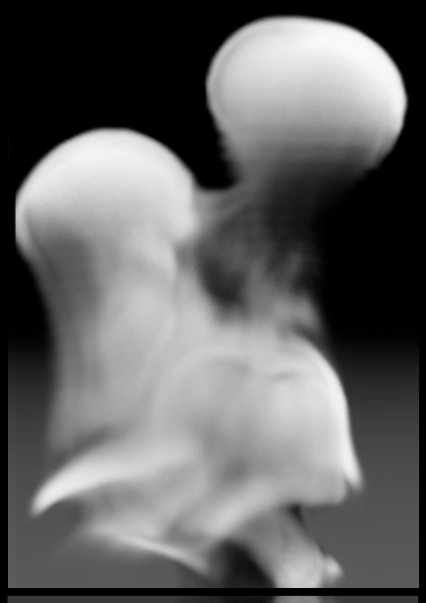


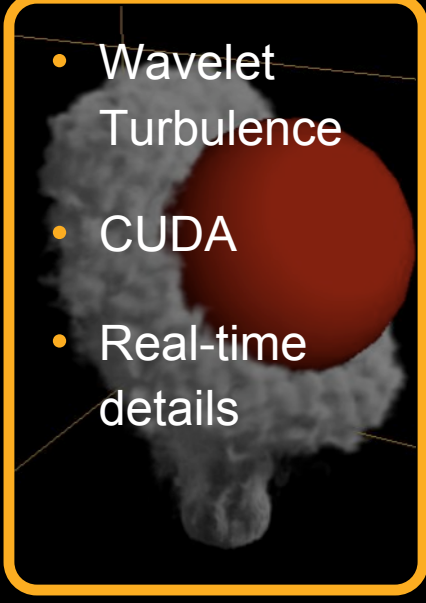
3D Fluids / LBM

Detailed Fluids

Overview



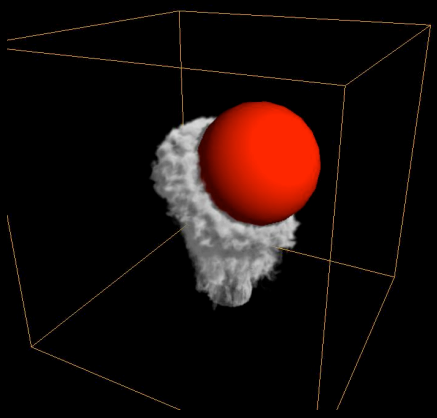




- Wavelet Turbulence
- CUDA
- Real-time details

Motivation

- Full 3D simulation expensive
  - Memory $O(n^3)$
  - Calculations $O(n^4)$
- Idea
  - Simulate as coarse as possible
  - Add detail for rendering
  - Here: wavelet turbulence

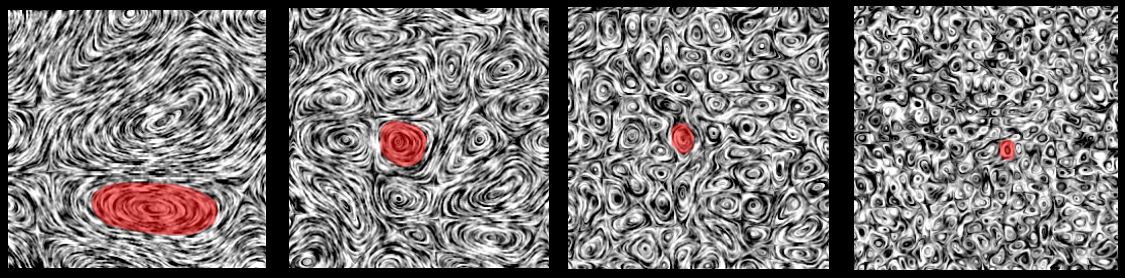


1) *R. Bridson. J. Hourihan and M. Nordenstam:* Curl-noise for procedural fluid flow. SIGGRAPH 2007

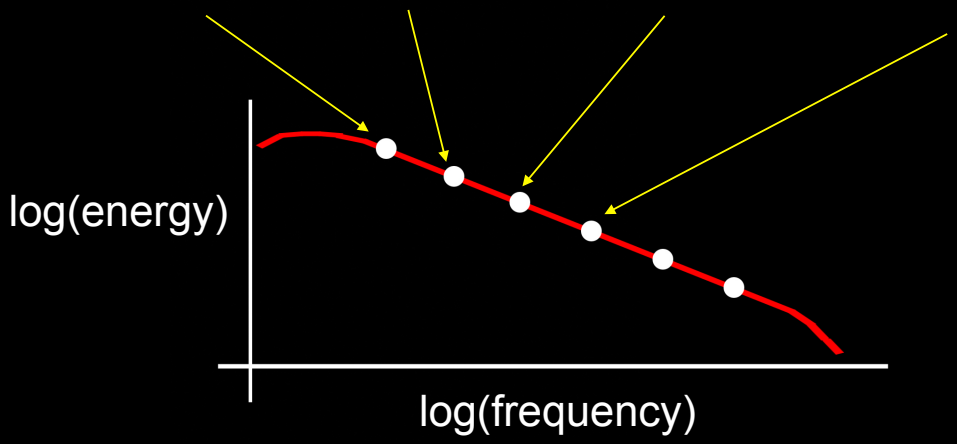
2) *A. Angelidis, F. Neyret, K. Singh and D. Nowrouzezahrai*: A controllable, fast and stable basis for vortex based smoke simulation. Proceedings of SCA 2006

3) *T. Kim, N. Thuerey, D. James, M. Gross*: Wavelet Turbulence for Fluid Simulation. SIGGRAPH 2008

Wavelet Turbulence

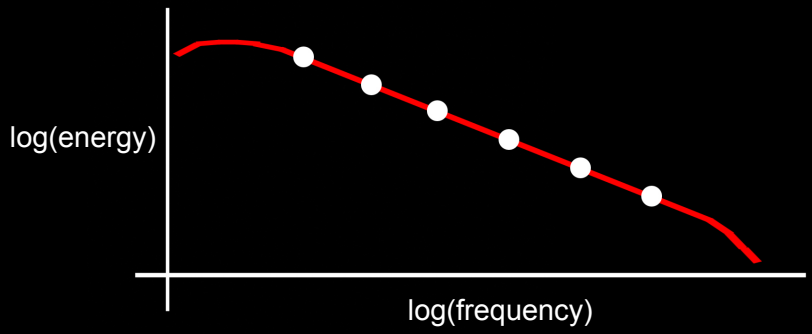


...

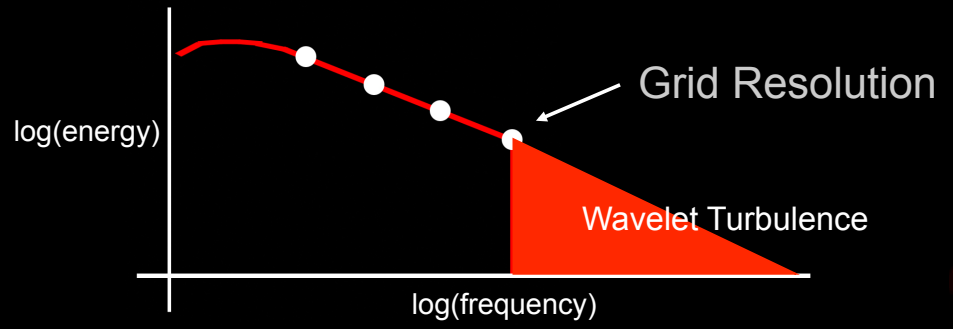


Wavelet Turbulence

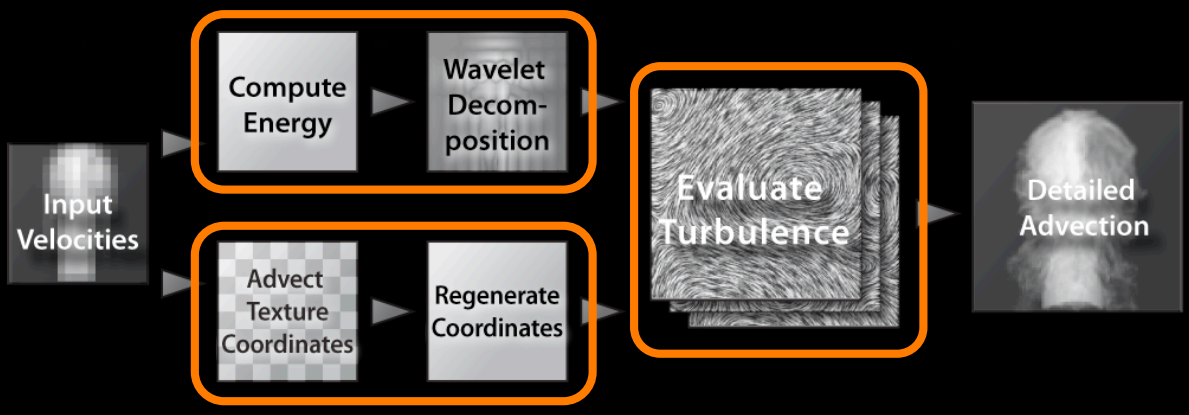
Ideal Result:



Simulation Result:

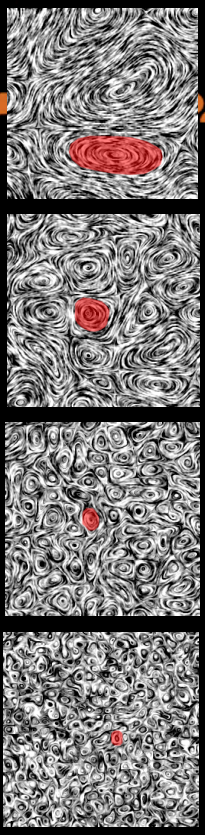


Wavelet Turbulence



- Wavelet decomposition for coupling
- Combined texture advection & regeneration
- Wavelet noise for synthesizing vortices

Turbulence Evaluation



- Main workload
- Not iterative, no dependencies
- Requires: position & look-up table

$$\mathbf{y}(\mathbf{x}) = \sum_{i=i_{\min}}^{i_{\max}} \mathbf{w}(2^i \mathbf{x}) 2^{-\frac{5}{6}(i-i_{\min})}$$

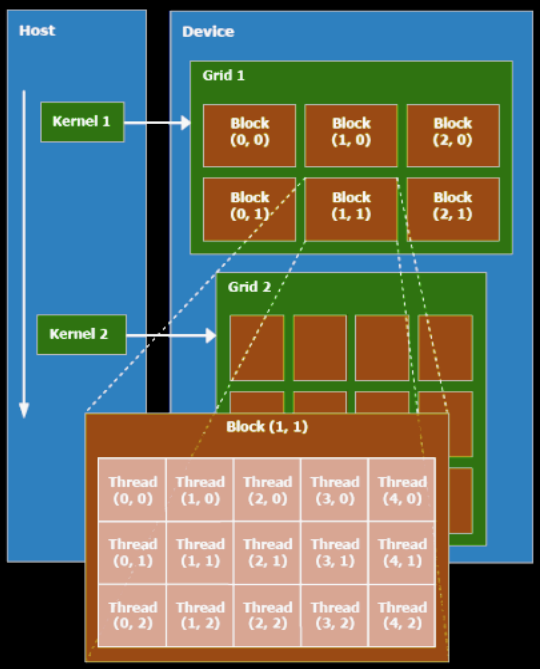
Here:

- Simulation:
  - Standard solver (1)
  - LBM (2)
  - SPH (3)
  - CPU / GPU / Co-processor
- GPU detail & rendering:
  - Get velocities
  - Evaluate turbulence
  - Advect high-res
  - Render

1) Real Time Simulation and Rendering of 3D Fluids; *K. Crane, I. Llamas, S. Tariq*; GPU Gems 3; 2007
2) Simulation with Complex Boundaries; *W. Li, Z. Fan, X. Wei, A. Kaufman*, GPU Gems 2; 2004
3) Particle-Based Fluid Simulation on the GPU; T. Amada, M. Imura, Y. Yasumuro, Y. Manabe, K. Chihara; 2004

Implementation with CUDA

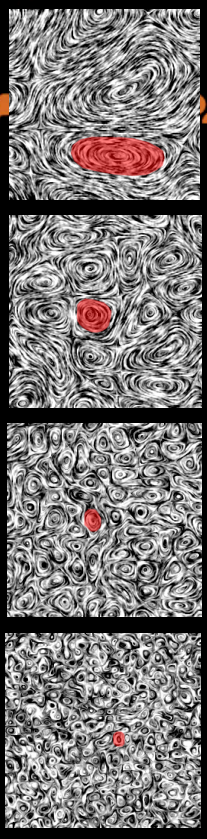
- Alternatives:
  - CTM, OpenCL, DirectX 11
- Approach:
  - High level language close to ANSI C
  - Kernel distributed to processing units
  - Massively parallel evaluation
  - Requires data transfer to / from GPU



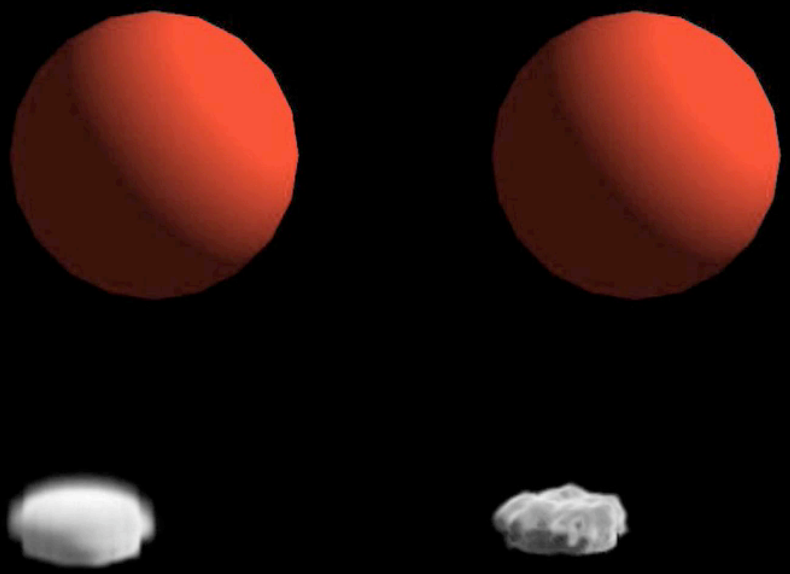
Turbulence Evaluation

- Implementation example:

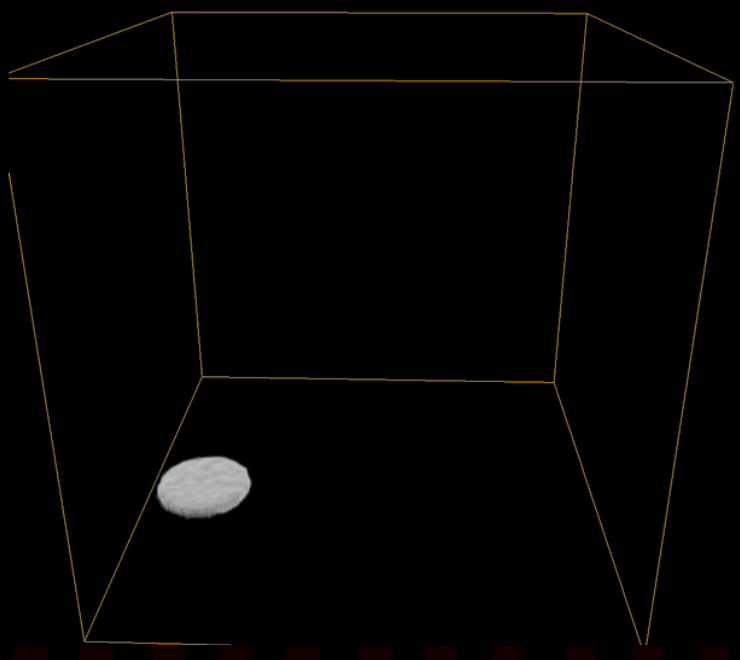
```
1)  texture< float4, 3, cudaReadModeElementType > tex_noise_ocave1;
2)  __constant__ float2 const_w0[ 3 ][ 32 ];
3)  __global__ void cuda_noise_inject(float* velocity, . . .) {
4)        const int i = threadIdx.x;
5)        const int j = blockIdx.x*blockDim.y+threadIdx.y;
6)        const int k = blockIdx.y*blockDim.z+threadIdx.z;
7)
8)        if (0 < i && i < sizeX - 1 &&
9)            0 < j && j < sizeY- 1  &&
10)           0 < k && k < sizeZ – 1 )  {
11)               . . .
12)     }
13) }
```



Sphere example – comparison

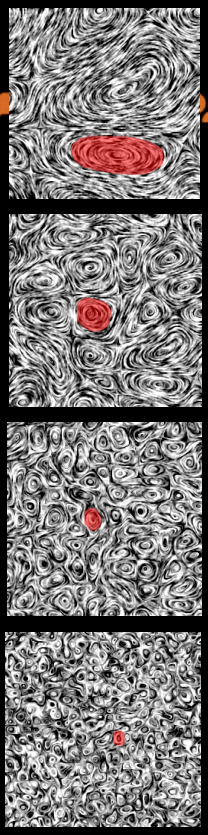


Moving obstacle example 1



Performance

- Final resolution 240^3
  - Turbulence evaluation: 3.8 FPS
- Final resolution 128^3
  - Overall: 8.9 FPS
  - 37% turbulence, 27% CPU-simulation, 17% rendering, 19% overhead
- Precomputations, 240^3
  - 11.7 FPS, with ¾ for advection

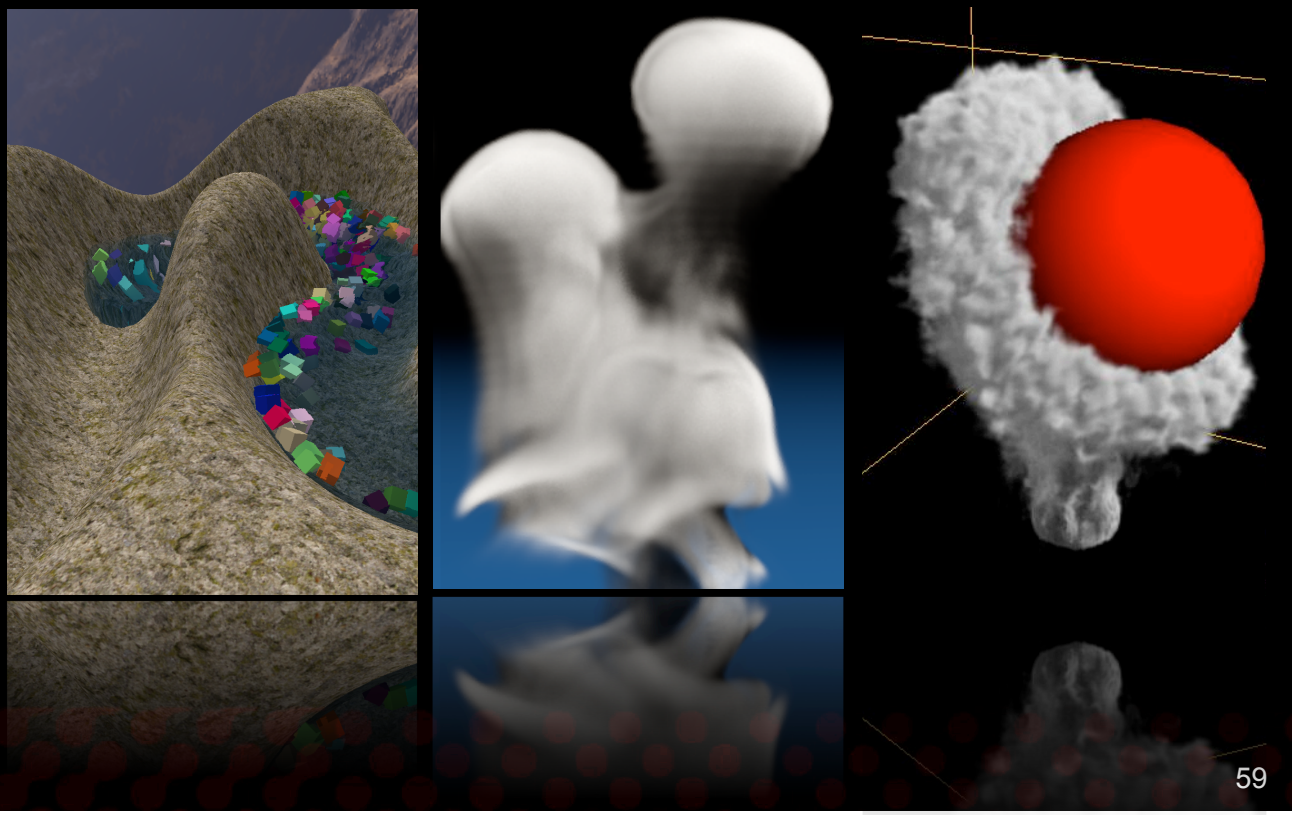


Wavelet Turbulence

- Thanks to: Ted Kim, B. Fierz
- References (among others):

1) *R. Cook, T. DeRose:* Wavelet Noise. SIGGRAPH 2005
2) *R. Bridson. J. Hourihan and M. Nordenstam:* Curl-noise for procedural fluid flow. SIGGRAPH 2007
3) 3) *T. Kim, N. Thuerey, D. James, M. Gross*: Wavelet Turbulence for Fluid Simulation. SIGGRAPH 2008

Conclusions



Conclusions

- Shallow water simulations
  - Interesting right now for real-time
- 3D Fluids / LBM
  - Possible efficiently in parallel settings
- Rendering / post-processing
  - Detailed simulations with wavelet turbulence

End

- Acknowledgements: AGEIA, NVIDIA, M. Mueller, P. Hess, R. Angst, T. Kim, B. Fierz

- Questions…?