CUDA Fluid Simulation in NVIDIA PhysX

Mark Harris

NVIDIA Developer Technology

NVIDIA

SIGGRAPHASIA2008

NEW HORIZONS

NVIDIA PhysX

- Game physics API and engine
  - Uses GPU and CPU

- Rigid Bodies
  - Destruction, cluttered environments, ragdolls, vehicles
- Deformables
  - Clothing, meaty chunks, organic creatures
- Fluids (as well as simple particle systems)
  - Fluid emitting weapons, debris effects
  - Focus of this talk

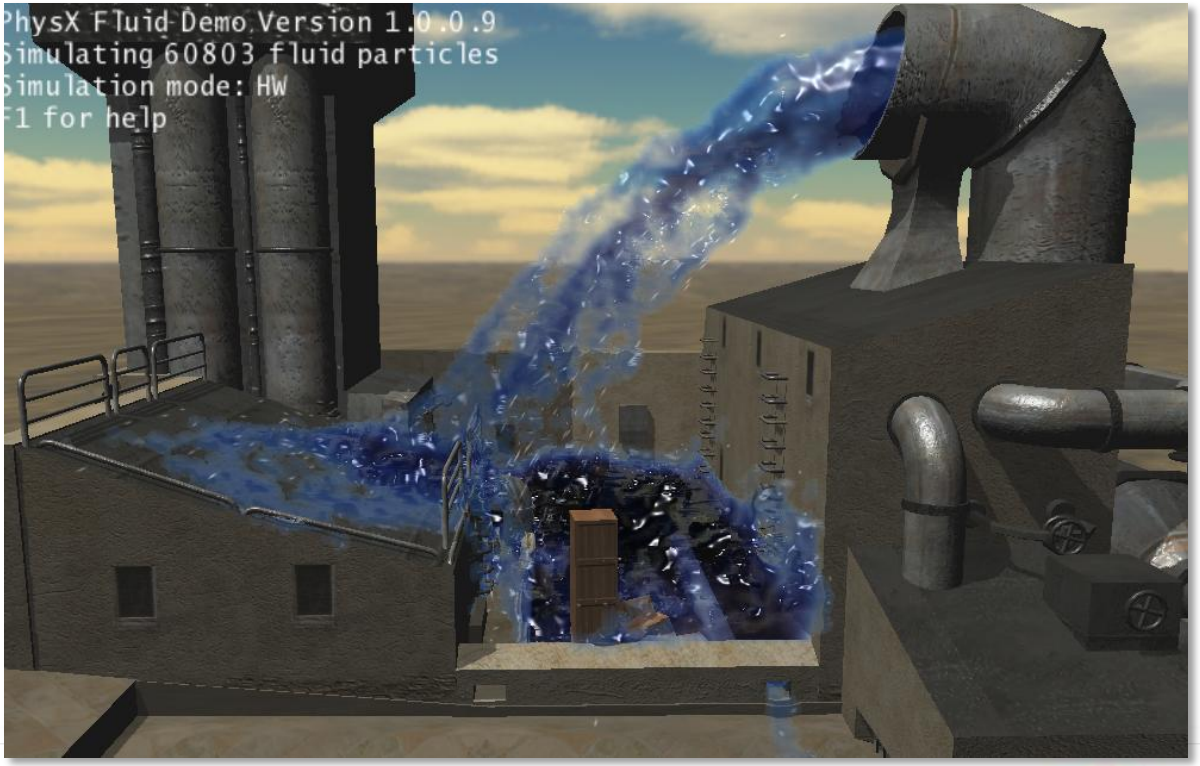
Fluid feature requirements

- Simulate a fluid that…
  - Doesn't pass through rigid objects
  - Conserves volume
    - so puddles and splashes form when objects are hit
  - Can move and be moved by rigid bodies
    - Can push objects and objects can float in bodies of water
  - Can flow anywhere in a large environment
    - Not contained to a small box
- Also: multiple independent fluids per scene
  - Efficient parallel multi-fluid simulation

NVIDIA PhysX Fluid Demo

- Available for download on the web:

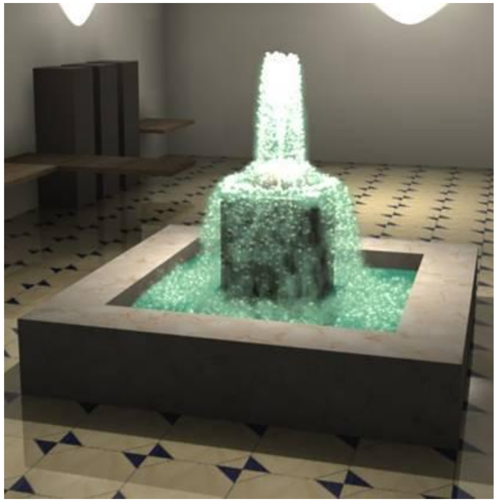
http://www.nvidia.com/content/graphicsplus/us/download.asp

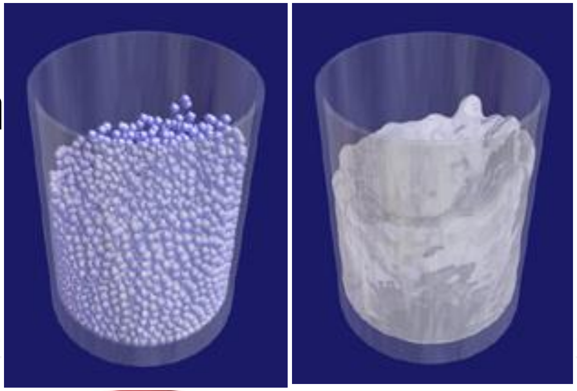


SIGGRAPH ASIA 2008
NEW HORIZONS

Particle-Based Fluids

- Particle systems are simple and fast
- Without particle-particle interactions
  - Can use for spray, splashing, leaves, debris, sparks, etc.
- With particle-particle interactions
  - Can use for volumetric fluid simulation

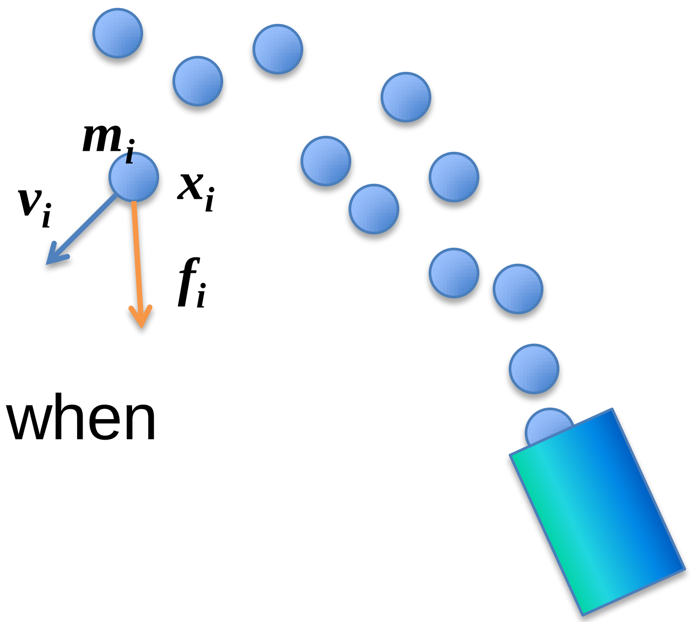




Simple Particle Systems

- Particles store mass, position, velocity, age, etc.
- Integrate: $d/dt\, x_i = v_i$

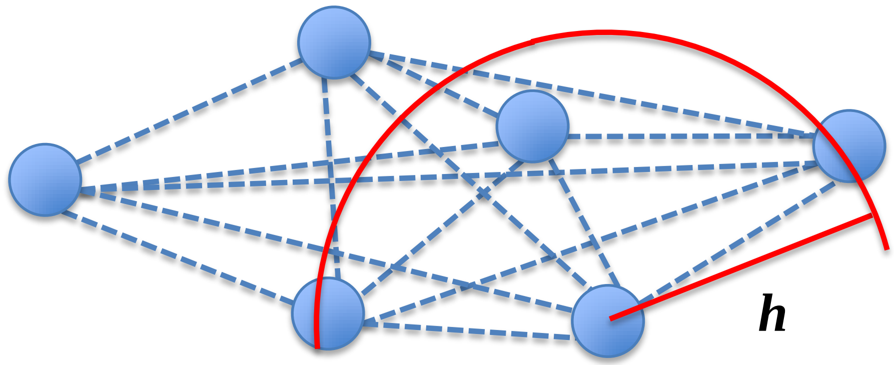
  $d/dt\, v_i = f_i / m_i$



- Generated by emitters, deleted when age > lifetime

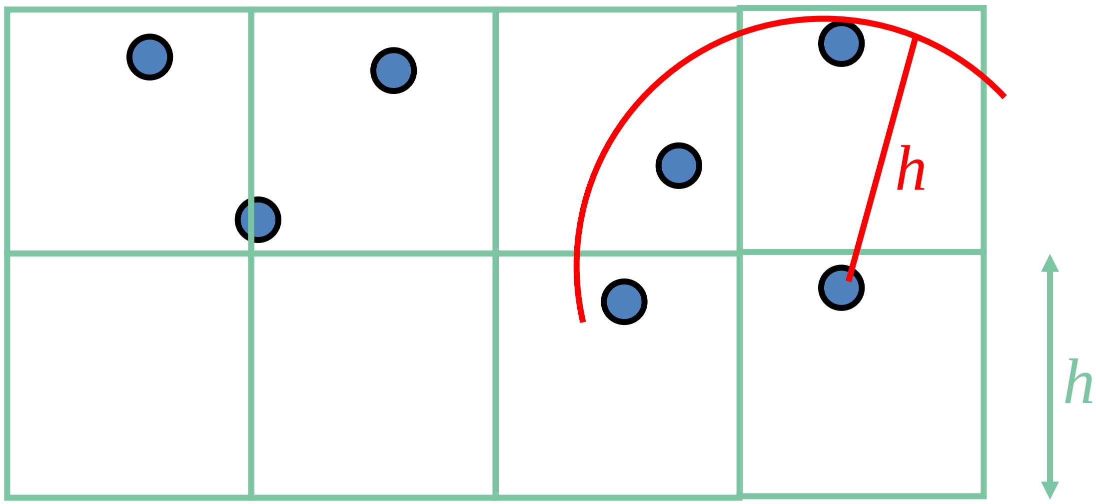
Particle-Particle Interaction

- Fluid simulation with particles requires inter-particle forces



- $O(n^2)$ potential computations for n particles!
- Reduce to linear complexity $O(n)$ by defining interaction cutoff distance h

Spatial Hashing



- Fill particles into grid with spacing h
- Only search potential neighbors in adjacent cells
- Map cells [i,j,k] into 1D array via hash function h(i,j,k)
  - [Teschner03]

Navier-Stokes Equations

$$\rho\left(\frac{\partial v}{\partial t}+v\cdot\nabla v\right)=\rho g-\nabla p+\mu\nabla^2 v$$

Gravity

Viscosity

Inertia: velocity advects velocity

Pressure gradient: fluid moves from high pressure to low

$$\nabla\cdot v=0$$

Conservation of mass: guaranteed by particles

Equation of Motion

$$\rho\left(\frac{\partial v}{\partial t} + v \cdot \nabla v\right) = \rho g - \nabla p + \mu \nabla^2 v$$

- Because particles follow the fluid we have:

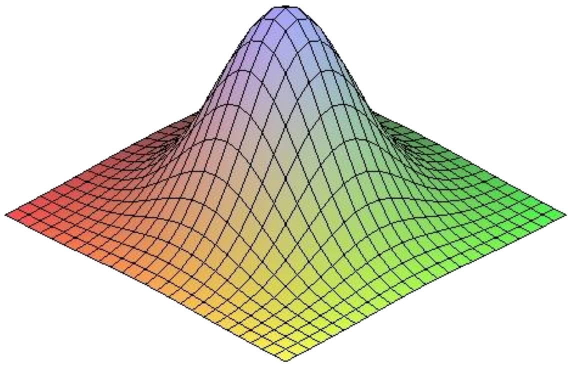
$$\frac{Dv}{Dt} = \left(\frac{\partial v}{\partial t} + v \cdot \nabla v\right) = \frac{dv_i}{dt} = a_i$$

- Thus acceleration of particle ***i*** is just $\boldsymbol{a_i} = \boldsymbol{f_i} / \boldsymbol{\rho_i}$
  - where $\boldsymbol{f_i}$ is the body force evaluated at $\boldsymbol{x_i}$

Solving the Navier-Stokes Equations

- How do we formulate fluids on particles?
  - We need to evaluate continuous fields (e.g. $v(x)$)
  - Only have v_i sampled on particles

- Basic idea:
  - Particles induce smooth local fields
  - Global field is sum of local fields

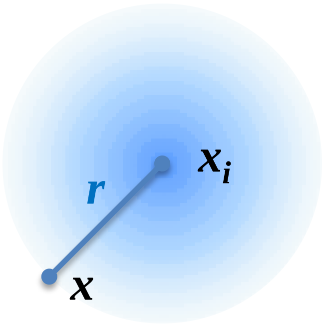


- **S**moothed **P**article **H**ydrodynamics

SPH

- Invented for the simulation of stars [Monaghan 92]
- Our implementation based on [Müller 03]

- Use scalar kernel function $W(r)$
  - $W_i(x) = W(|x - x_i|)$
  - Normalized: $\iiint W_i(x)dx = 1$

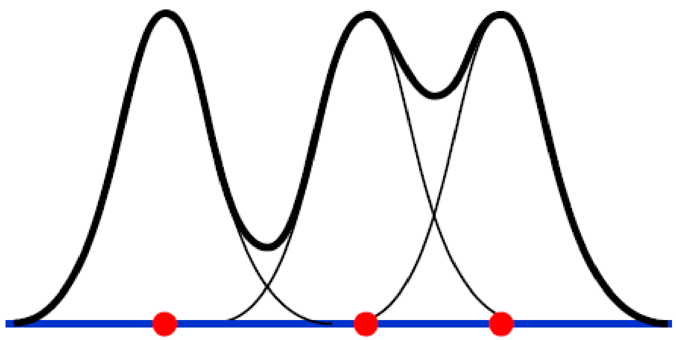


- Example [Müller 03]: $W(r,h) = \frac{315}{64\pi h^9}(h^2 - r^2)^3, 0 \le r \le h$

Density Computation

- Global Density Field

$$\rho(x) = \sum_j m_j W(x - x_j)$$



- Density of each particle

$$\rho_i = \rho(x_i)$$

- Mass conservation guaranteed if W is normalized

$$\int \rho(x) = \sum_j \left(m_j \int W(x - x_j)\,dx \right) = \sum_j m_j$$

Pressure

- The pressure term yields:

$$f_i^{\text{pressure}} = -\nabla p(x_i) = -\sum_j \frac{m_j}{\rho_j} \nabla W(x_i - x_j)$$

- Symmetrize: (SPH problem: action ≠ reaction)

$$f_i^{\text{pressure}} = -\nabla p(x_i) = -\sum_j \frac{m_j}{\rho_j} \frac{p_i + p_j}{2} \nabla W(x_i - x_j),$$

$$\text{where } p_i = k(\rho_i - \rho_0)$$

- k is gas constant (stiffness), ρ_0 is desired rest density
  - Other state laws are possible [Becker 07]

Remaining forces

- External force, e.g. gravity:

$$f_i^{\text{external}} = \rho_i g$$

- Viscosity (symmetrized)

$$f_i^{\text{viscosity}} = \mu \sum_j m_j \frac{v_j - v_i}{\rho_j} \nabla^2 W(x_i - x_j)$$

SPH algorithm

- Build spatial data structure on particles
  - To enable fast neighbor finding

- For each particle
  - find neighbors and compute density
- For each particle
  - find neighbors, compute force and update velocity
- For each particle
  - Find neighboring triangles, check collision, update pos

Parallel SPH algorithm

- Build spatial data structure on particles (**in parallel**)
  - To enable fast neighbor finding
- For all particles **in parallel**
  - find neighbors and compute density
- For all particles **in parallel**
  - find neighbors, compute force and update velocity
- For all particles **in parallel**
  - Find neighboring triangles, check collision, update pos

GPU Algorithm Goals

- For best GPU performance, we need
  - Thousands of threads to cover memory latency
  - Minimal communication/sync between threads
  - Similar amount of work per thread
  - High arithmetic intensity

SPH Algorithm properties

- Explicit integration, accept some compressibility
  - Each particle independent so can be given own thread
  - No synchronization or r/w comms between threads
- Particles have many neighbors
  - High arithmetic intensity
- Variable number of neighbors
  - Threads take different amounts of time
  - However there is coherence in workload
- Pair computation quite complex
  - Register-intensive

Spatial Data Structures

- Must enable sparse fluids in large environments
- Must quickly find all neighbors of each particle
  - Within a given radius

- Options
  - KD-tree
  - Octree
  - Hashed uniform grid

Hashed Uniform Grid

- Abstract, infinite 3D grid mapped to concrete, finite grid
  - Concrete grid = 64*64*64 uniform grid
- Abstract (x,y,z) maps to concrete (x%64,y%64, z%64).

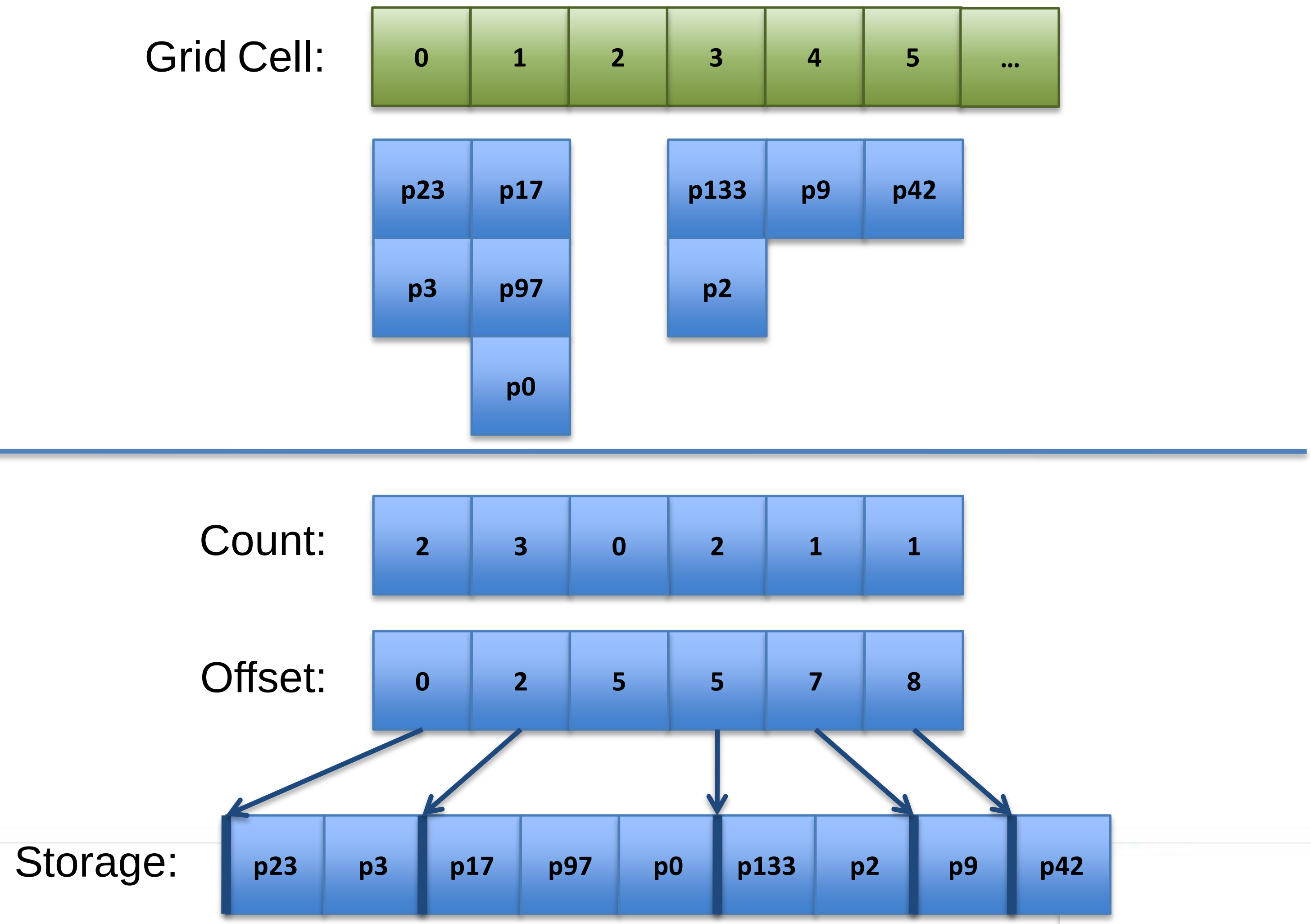
- Neighbor finding is fast (assuming no collisions)
  - Abstract neighbor cell ($x+dx$,$y+dy$,$z+dz$) maps to concrete (($x+dx$)%64,($y+dy$)%64,($z+dz$)%64)

- Like hashing, but with an intuitive hash function
- Can support multiple fluids by storing fluid ID in hash

Building the Hashed Grid

- For all particles in parallel
  - Compute hash key (cell index) from position
- Sort particle (key, particleID) pairs by hash key
  - Use a fast GPU radix sort
  - Use sorted order to re-order particle pos, vel arrays
    - Sorted arrays ensure coherent memory access
- Store starting (sorted) particle index for each grid cell
  - Some waste for large, sparse grids

Hashed Grid: Array of Lists



Radix Sort

- Fast CUDA radix sort for variable-length keys

- Advantage: many fluids can be sorted at once
  - Put fluid ID in MSBs, particle ID in LSBs
  - With 64^3 grid, 4 fluids = 20-bit key, 64 fluids = 24-bit key

- "Designing Efficient Sorting Algorithms for Manycore GPUs". N. Satish, M. Harris, and M. Garland. To appear in Proceedings of *IEEE International Parallel and Distributed Processing Symposium* 2009.

Mapping SPH to hardware

- Each particle is assigned a thread
- Thread queries grid to find its neighbors
- Due to virtual grid, some “neighbors” may be far away
  - SPH smoothing will ensure they don’t have an effect
  - Can early exit density/pressure calc if dist is too large
  - Hasn’t been a performance issue yet for PhysX

Computing Density and Force

- For each particle position x use grid to iterate over neighbor particles j and compute density:

$$\rho(x) = \sum_j m_j W(x - x_j)$$

- For each particle position x use grid to iterate over neighbor particles j and compute force:

$$f_i^{\text{pressure}} = -\sum_j \frac{m_j}{\rho_j} \frac{p_i + p_i}{2} \nabla W(x_i - x_j), \text{where } p_i = k(\rho_i - \rho_0)$$

- Other forces:
  - viscosity computed with pressure, gravity is trivial

Collision

- Dynamic shape collisions
  - Test against dynamic boxes, capsules and convexes
  - Compute dynamic collision restitution
- Static shape collisions
  - Test against static boxes, capsules, and convexes
  - Compute static collision restitution
- Triangle Meshes (e.g. terrain) collisions
  - Find collisions of particles with static meshes
  - Compute static collision restitution

Parallelizing Collision Detection

- Host broad phase generates collision *work units*:
  - Particle "packets" + potentially colliding objects (PCO)
- One CUDA thread block per work unit, thread per particle
  - Test (continuous) collision with each PCO
  - Uniform per-thread work
- Convex shapes slightly trickier due to variable # of planes
  - But still uniform work across threads in a work unit

Fluid Rendering

Screen Space Mesh Rendering

- Similar to [Müller 2007], without explicit meshing

- Setup:
  - Render scene to color texture
  - Render particles to depth texture as sphere point sprites
  - Render particle *thickness* to thickness texture with additive blending
  - Smooth depth texture to avoid “particle appearance”
  - Compute surface normal texture

Final rendering

- Use color, depth and thickness buffers

$$C_{out} = a(1 - F(n \cdot v)) + bF(n \cdot v) + k_s (n \cdot h)^{\alpha}$$

Fresnel Refraction | Fresnel Reflection | Phong Highlight

- With $a = \mathrm{lerp}(C_{fluid}, S(x + \beta \mathbf{n}_x, y + \beta \mathbf{n}_y), e^{-T(x,y)})$,

$$\beta = \gamma \mathrm{T}(\mathrm{x}, \mathrm{y})$$

- S = scene color texture; T(x, y) = thickness texture; **n** = normal texture computed from smoothed depth

Smoothing the depth buffer

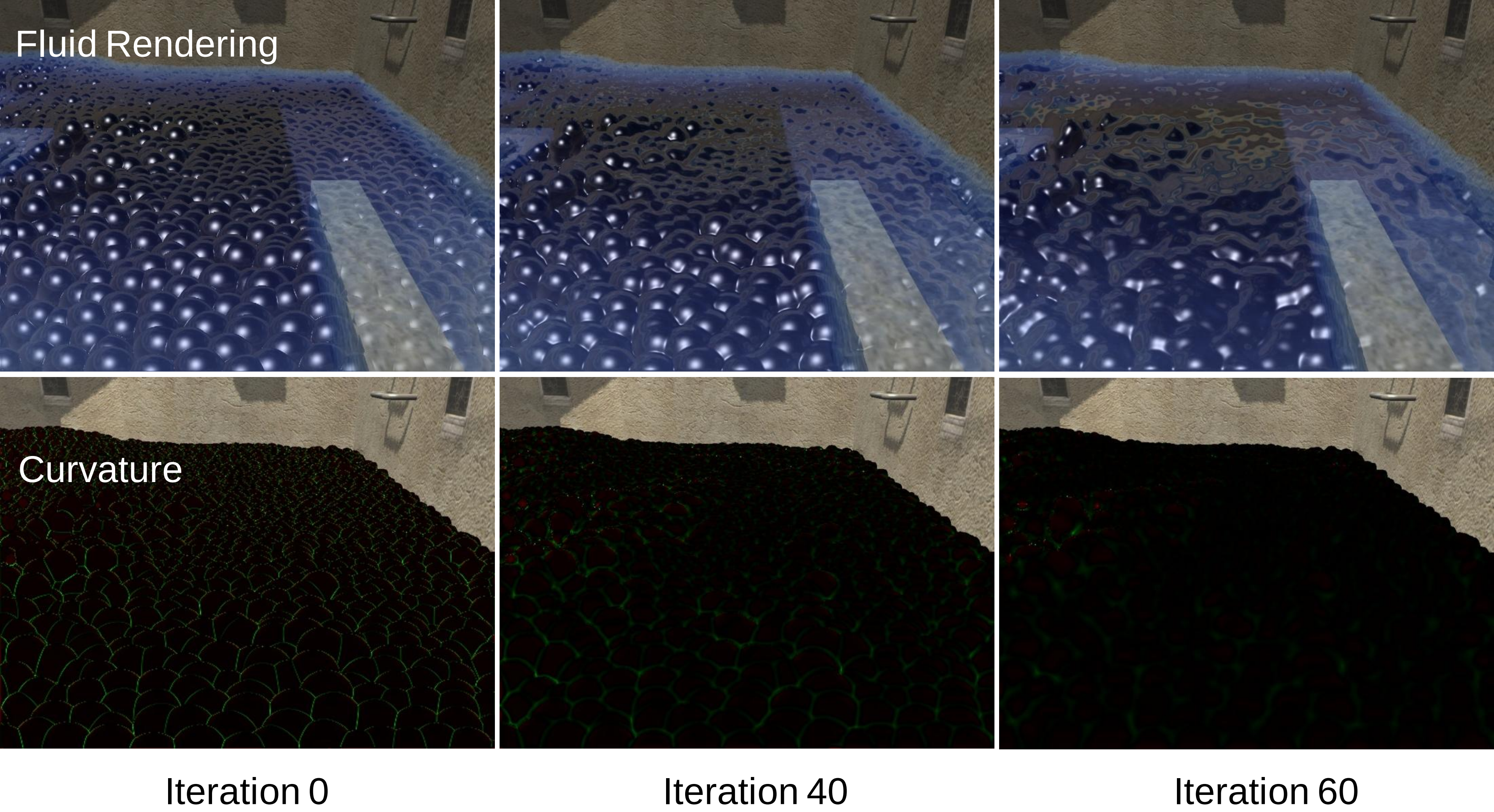
- Initially tried Gaussian and Bilateral Gaussian filters
  - Gaussian kernels produce leaking
  - Bilateral filters are non-separable (expensive)

- Released PhysX fluid demo uses Gaussian blur
  - Fluid looks slightly bumpy, like “tapioca”

- Look at the problem a different way:
  - Need to smooth out sudden changes in curvature
  - Seems natural, since this is what surface tension does

Screen-space Curvature Flow

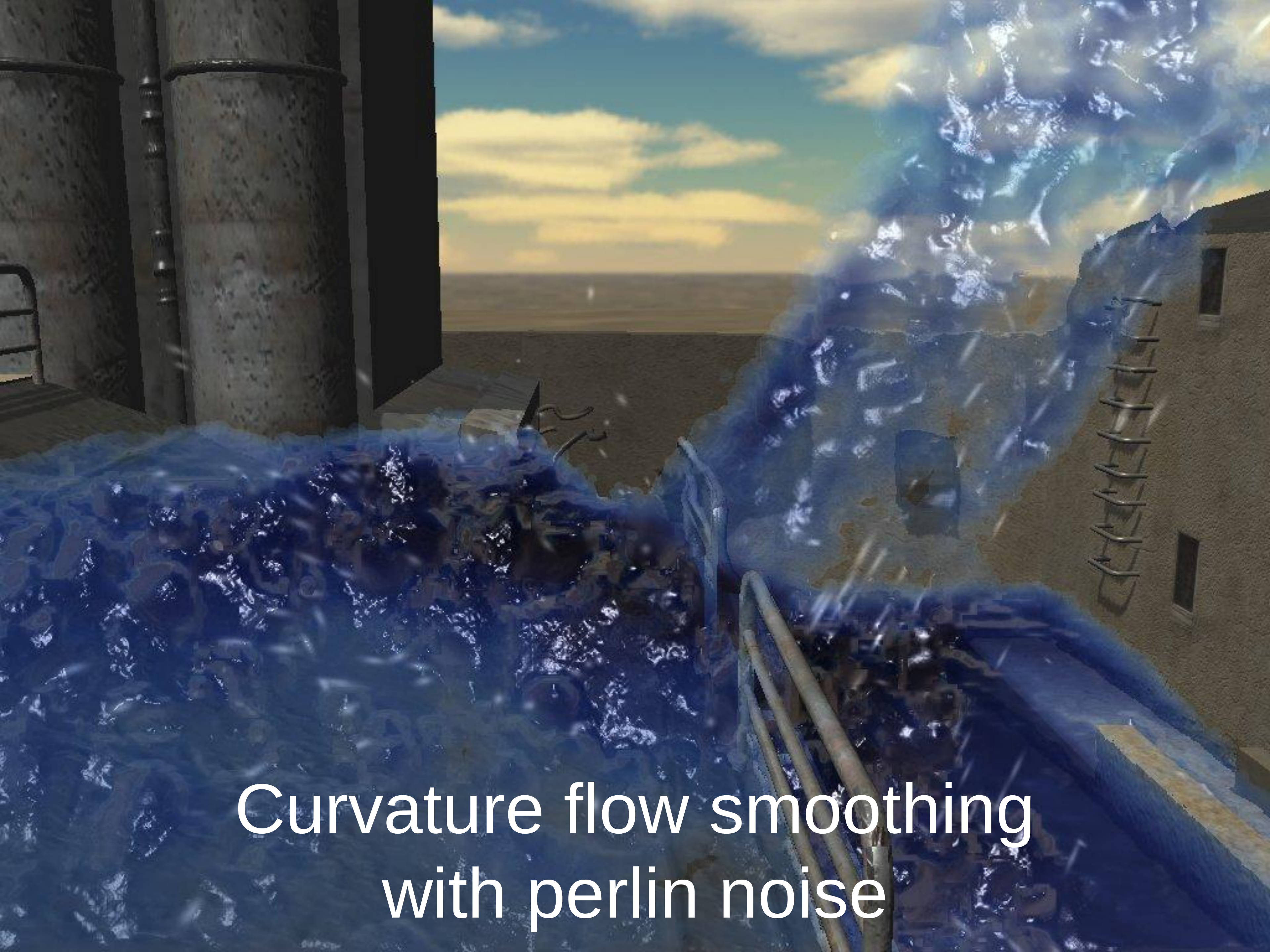
- *Curvature flow* [Malladi and Sethian 1995]
  - Evolve surface along normal direction with speed determined by mean surface curvature
- *Screen-space curvature flow* moves surface only in *Z*
  - Hence *screen-space*, since it modifies depth only.
  - Works well in practice
- "Screen Space Fluid Rendering with Curvature Flow". Wladimir Van der Laan, Simon Green, and Miguel Sainz. To appear in proceedings of I3D 2009.



•Iterative screen space curvature flow smoothing


Gaussian-based smoothing

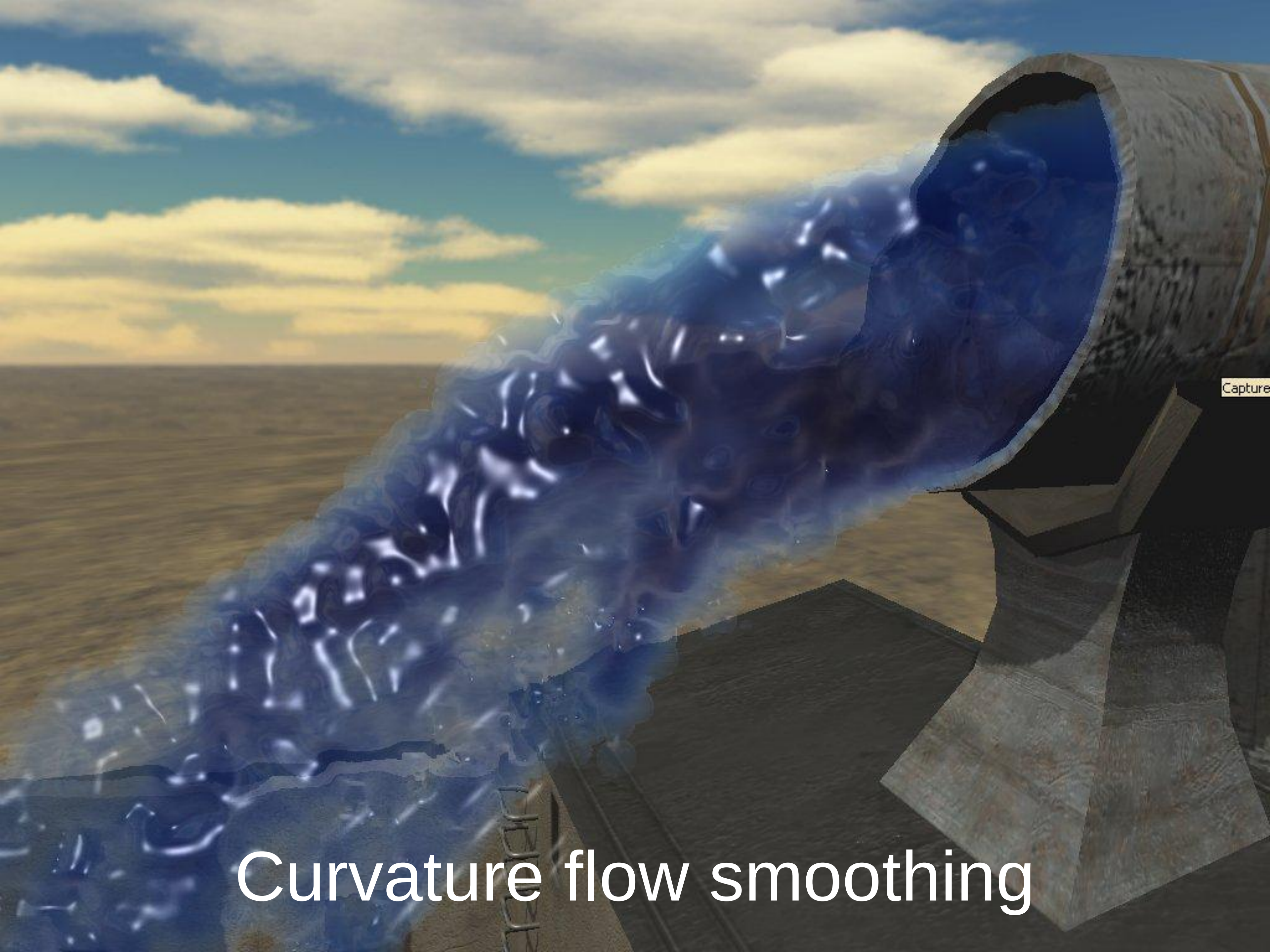

Curvature flow smoothing

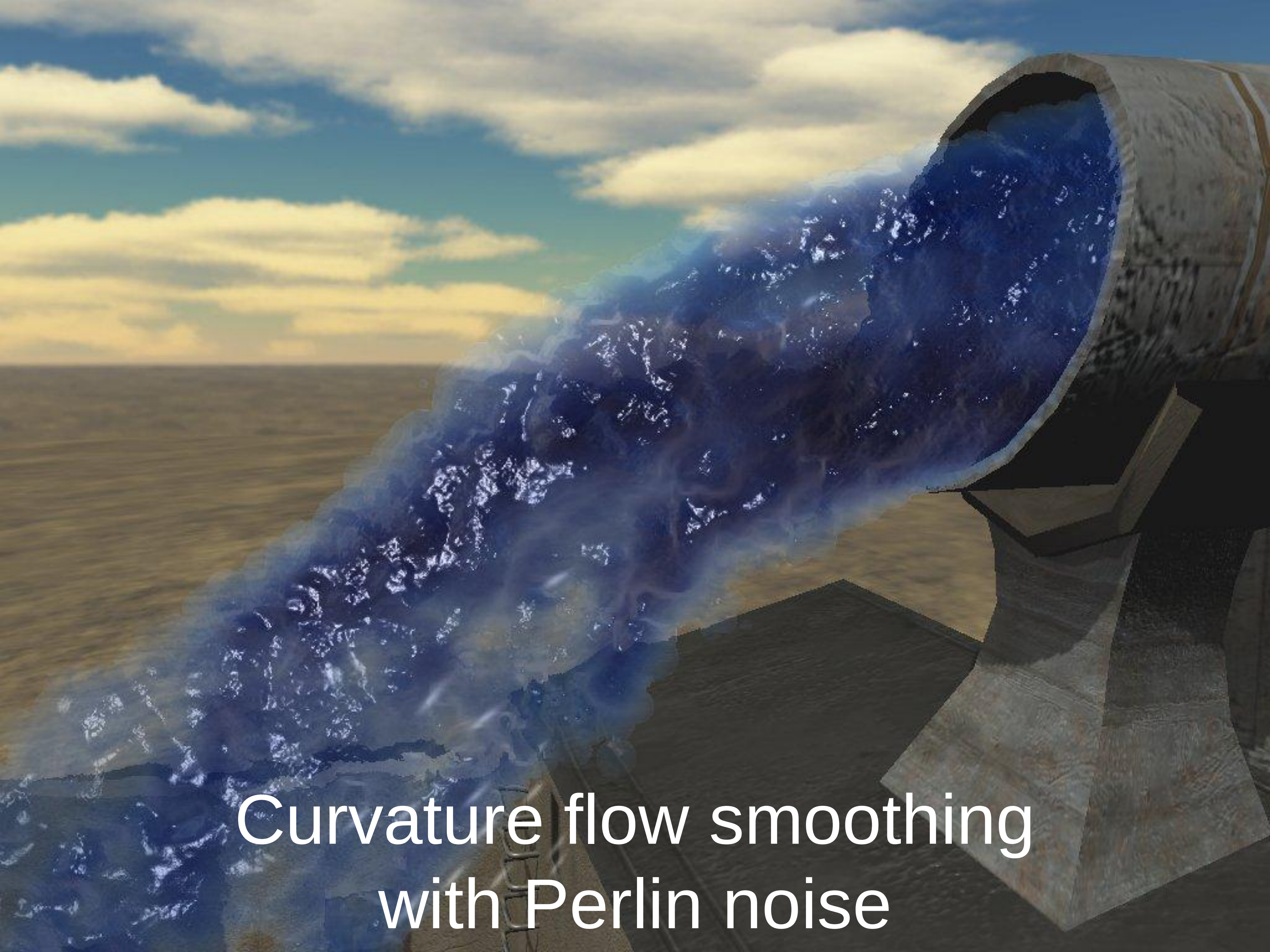

Curvature flow smoothing
with perlin noise


Capture a window
Gaussian-based smoothing


Curvature flow smoothing

Gaussian-based smoothing


Curvature flow smoothing


Curvature flow smoothing
with Perlin noise

Acknowledgements

- PhysX Fluids project contributors (direct and indirect)
  - NVIDIA PhysX team
    - Simon Schirm: fluid software architect
  - NVIDIA Developer Technology team
  - NVIDIA CUDA Software team
  - NVIDIA Research

- Richard Tonge, Simon Green, Miguel Sainz, Matthias Müller-Fischer
  - slides and content

Questions?

mharris@nvidia.com

References

- [Becker 07] M. Becker and M. Teschner. “Weakly compressible SPH for free surface flows”. SCA 07
- [Monaghan 92] J.J. Monaghan. “Smoothed Particle Hydrodynamics”. Annual Review of Astronomy and Astrophysics, 30:543-574, 1992
- [Müller 03] Müller, M., et al.“Particle-based Fluid Simulation for Interactive Applications”. SCA 03
- [Müller 07] Müller, M., Schirm, S., and Duthaler, S. “Screen-space Meshes”. SCA 07

References

- [Satish 09] Satish, N., Harris, M., and Garland, M. "Designing Efficient Sorting Algorithms for Manycore GPUs". Proceedings of *IEEE International Parallel and Distributed Processing Symposium* (IPDPS) 2009 (to appear).
- [Teschner03] M. Teschner et al., "Optimized Spatial Hashing for Collision Detection of Deformable Objects", VMV 03
- [van der Laan 09] van der Laan, W., Green, S., and Sainz, M. "Screen Space Fluid Rendering with Curvature Flow". I3D 2009 (to appear).