CUDA Particle-based Fluid Simulation

Simon Green

Overview

- Fluid Simulation Techniques
- CUDA particle simulation
- Spatial subdivision techniques
- Rendering methods
- Future

Fluid Simulation Techniques

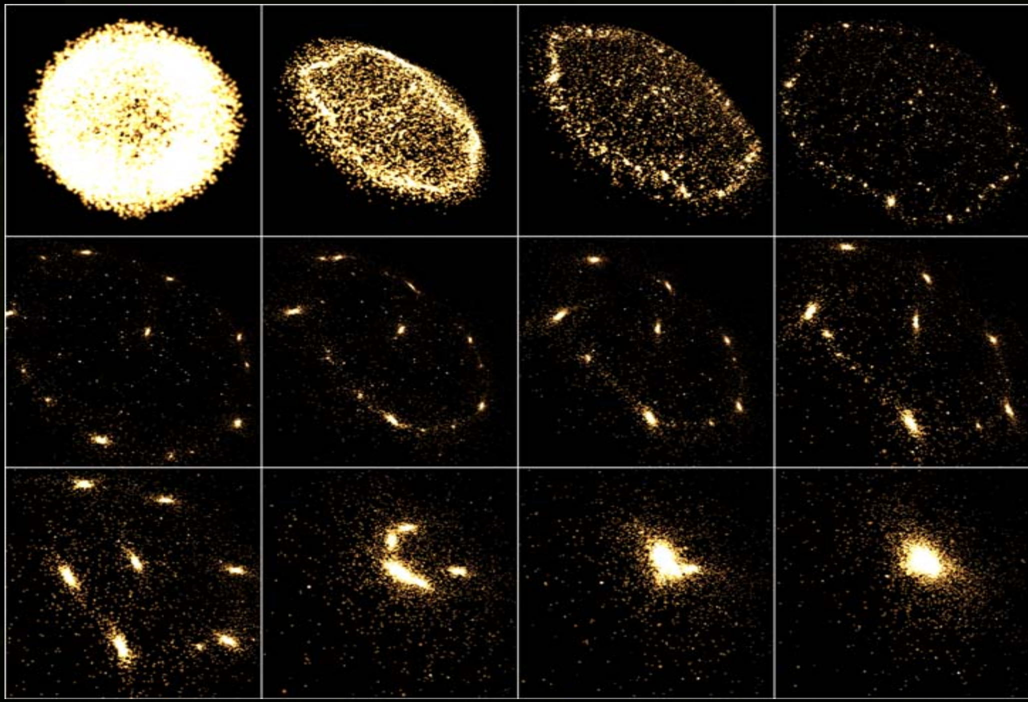
- Various approaches:
- Grid based (Eulerian)
  - Stable fluids
  - Particle level set
- Particle based (Lagrangian)
  - SPH (smoothed particle hydrodynamics)
  - MPS (Moving-Particle Semi-Implicit)
- Height field
  - FFT (Tessendorf)
  - Wave propagation – e.g. Kass and Miller

CUDA N-Body Demo

- Computes gravitational attraction between n bodies
- Computes all n^2 interactions
- Uses shared memory to reduce memory bandwidth

16K bodies @ 44 FPS
x 20 FLOPS / interaction
x $16K^2$ interactions / frame
= 240 GFLOP/s

GeForce 8800 GTX

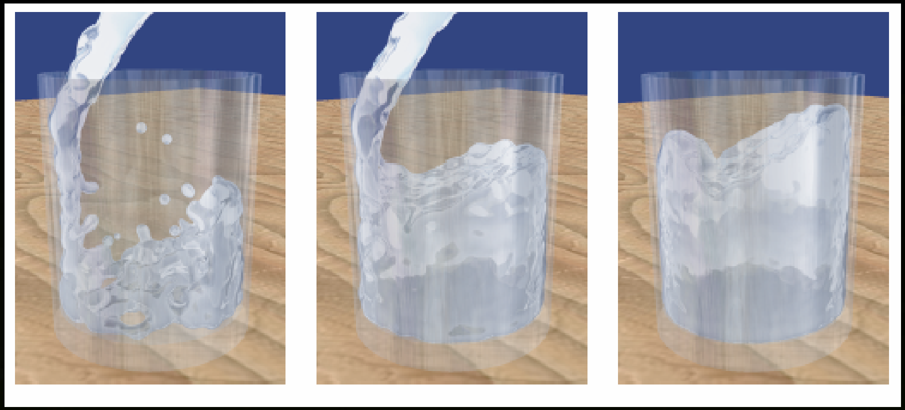


Particle-based Fluid Simulation

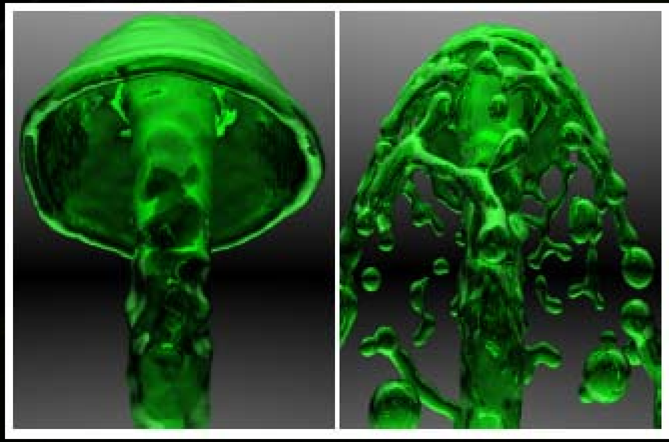
- Advantages
  - Conservation of mass is trivial
  - Easy to track free surface
  - Only performs computation where necessary
  - Not necessarily constrained to a finite grid
  - Easy to parallelize
- Disadvantages
  - Hard to extract smooth surface from particles
  - Requires large number of particles for realistic results

Particle Fluid Simulation Papers

- Particle-Based Fluid Simulation for Interactive Applications, M. Müller, 2003
- 3000 particles, 5fps



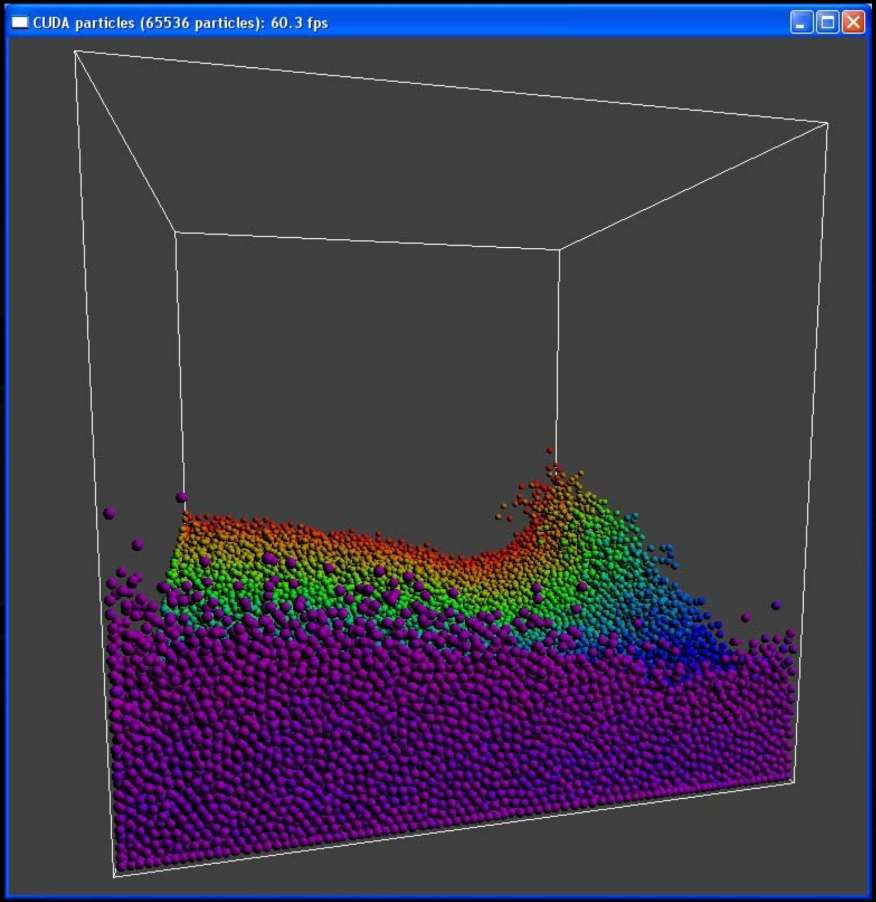
- Particle-based Viscoelastic Fluid Simulation, Clavet et al, 2005
- 1000 particles, 10fps
- 20,000 particles, 2 secs / frame



CUDA SDK Particles Demo

- Particles with simple collisions
- Uses uniform grid based on sorting
- Uses fast CUDA radix sort

- Current performance: >100 fps for 65K interacting particles on 8800 GT



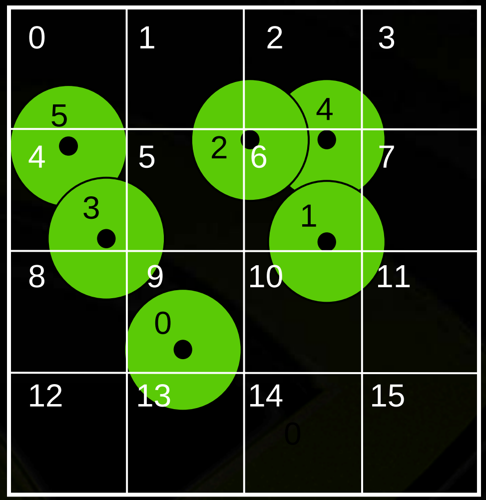
Uniform Grid

- Particle interaction requires finding neighbouring particles
- Exhaustive search requires n^2 comparisons
- Solution: use spatial subdivision structure
- Uniform grid is simplest possible subdivision
  - Divide world into cubical grid (cell size = particle size)
  - Put particles in cells
  - Only have to compare each particle with the particles in neighbouring cells
- Building data structures is hard on data parallel machines like the GPU
  - possible in OpenGL (using stencil routing technique)
  - easier using CUDA (fast sorting, scattered writes)

Uniform Grid using Sorting

- Grid is built from scratch each frame
  - Future work: incremental updates?
- Algorithm:
  - Compute which grid cell each particle falls in (based on center)
  - Calculate cell index
  - Sort particles based on cell index
  - Find start of each bucket in sorted list (store in array)
  - Process collisions by looking at 3x3x3 = 27 neighbouring grid cells of each particle
- Advantages
  - supports unlimited number of particles per grid cell
  - Sorting improves memory coherence during collisions

Example: Grid using Sorting



unsorted list
(cell id, particle id)

0: (9, 0)
1: (6, 1)
2: (6, 2)
3: (4, 3)
4: (6, 4)
5: (4, 5)

sorted by
cell id

0: (4, 3)
1: (4, 5)
2: (6, 1)
3: (6, 2)
4: (6, 4)
5: (9, 0)

cell start

0: -
1: -
2: -
3: -
4: 0
5: -
6: 2
7: -
8: -
9: 5
10: -
...
15: -

Fluid Rendering Methods

- 3D isosurface extraction (marching cubes)
- 2.5D isosurfaces (Ageia screen-space meshes)
- 3D texture ray marching (expensive)
- Image-space tricks (blur normals in screen space)

Marching Cubes

- Popular method for extracting isosurfaces from volume data
  - Lorensen and Cline (Siggraph 1987!)
  - Polygonizes a scalar field
  - Isosurface is surface where field == n
- Divides volume in cubical voxels
  - Outputs triangles based on field values at corners
  - Interpolates points along edges based on field values
  - Based on look-up tables

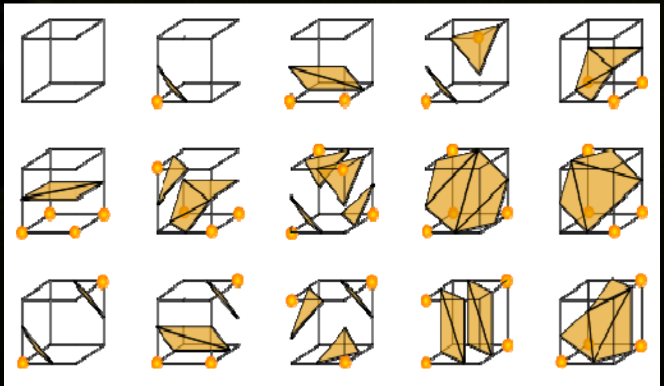
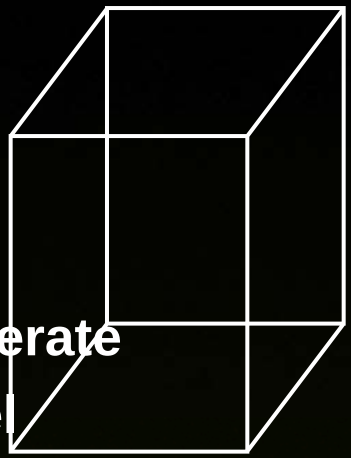


Image courtesy Wikipedia

Isosurface Extraction on the GPU

- Difficult on GPUs because of variable output
  - 0-5 triangles per voxel
- Implementations on previous hardware generations performed a lot of redundant computations
- Possible on DirectX 10 class hardware using geometry shader
  - Marching tetrahedrons matches hardware best (4 inputs)
- Can we also do this in CUDA?
  - Yes, using prefix sums (scan) for stream compaction
  - Uses CUDPP library (Harris et al)

CUDA Marching Cubes



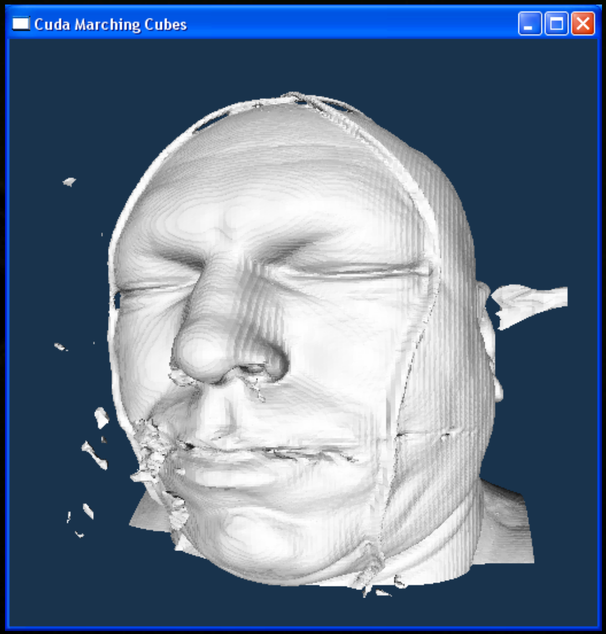
- Algorithm consists of several stages
  - tables are stored in 1D textures
- Execute *classifyVoxel* kernel
  - computes number of vertices voxel will generate
  - evaluates field at each corners of each voxel
  - one thread per voxel
  - writes *voxelOccupied* flag and *voxelVertices* to global memory
- Scan *voxelVertices* array
  - gives start address for vertex data for each voxel
- Read back total number of vertices from GPU to CPU
  - last element in scanned array

CUDA Marching Cubes (cont.)

- Scan *voxelOccupied* array
- Read back total number of occupied voxels from GPU to CPU
- Compact *voxelOccupied* array to get rid of empty voxels
- Execute generateTriangles kernel
  - runs only on occupied voxels
  - looks up field values again
  - generates triangles, using results of scan to write output to correct addresses
- Render geometry
  - using number of vertices from readback

Marching Cubes Performance

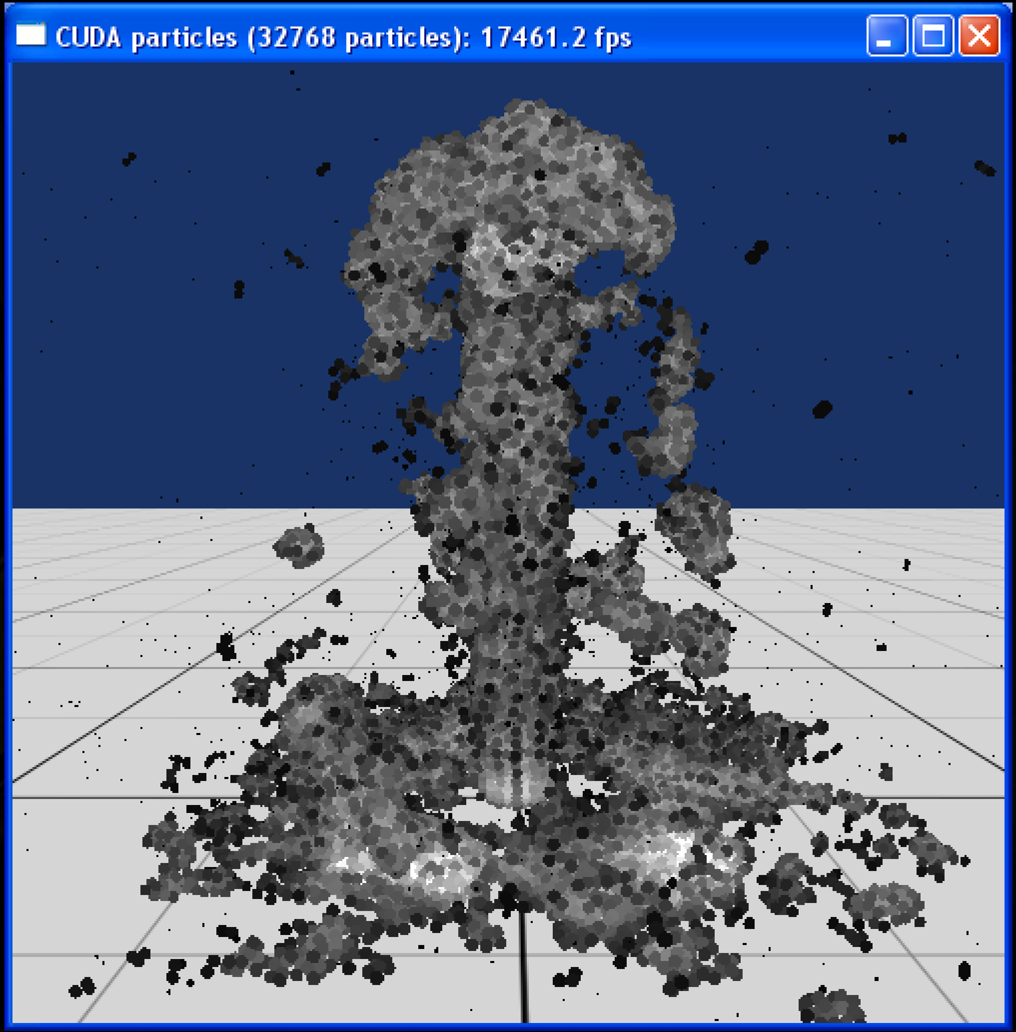
- Up to 8x faster than OpenGL geometry shader implementation using marching tetrahedra
- But still requires evaluating field function at every point in space
  - E.g. 128^3 = 2M points
  - Very expensive



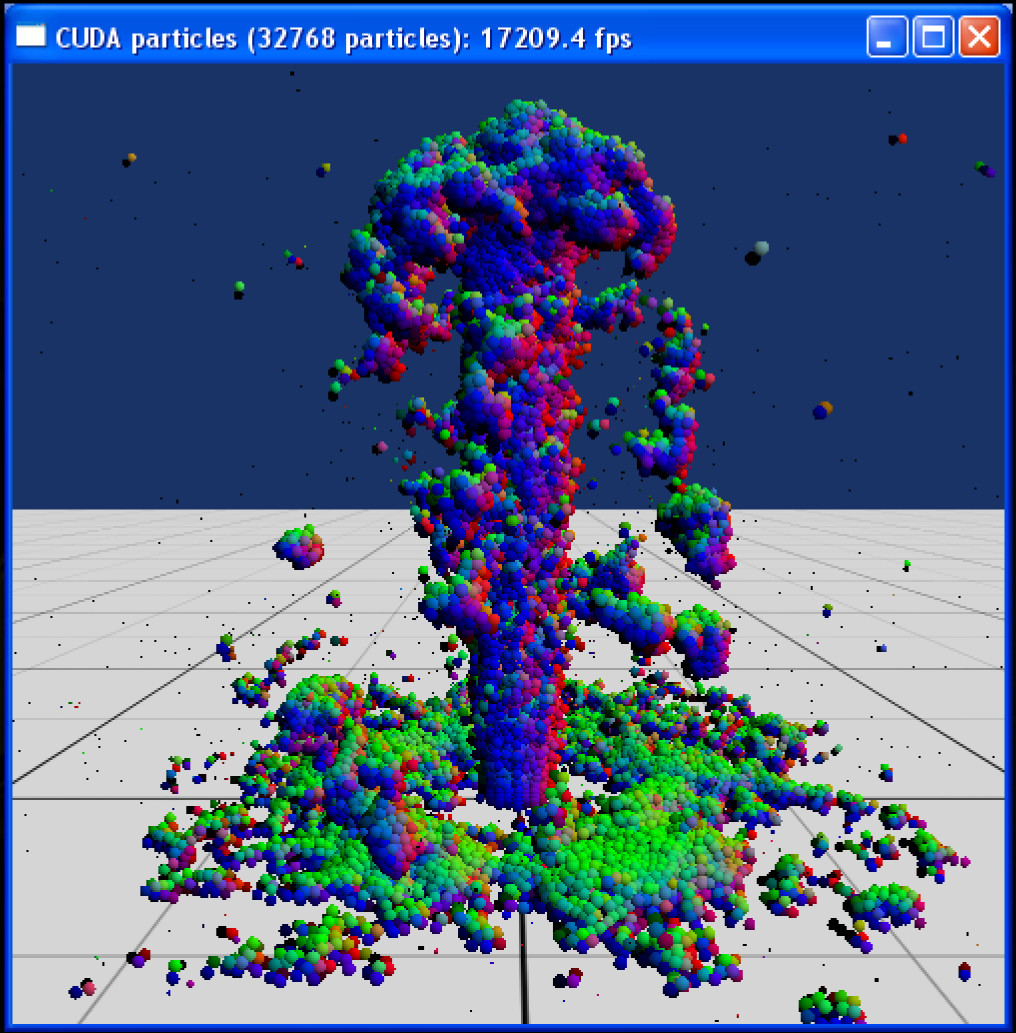
Density-based Shading

- Can calculate per-particle density and normal based on field function
  - SPH simulations often already have this data
  - Usually need to look at a larger neighbourhood (e.g. 5x5x5 cells) to get good results – expensive
- Can use density and normal for point sprite shading
- Normal only well defined when particles are close to each other
  - treat isolated particles separately – e.g. render as spray

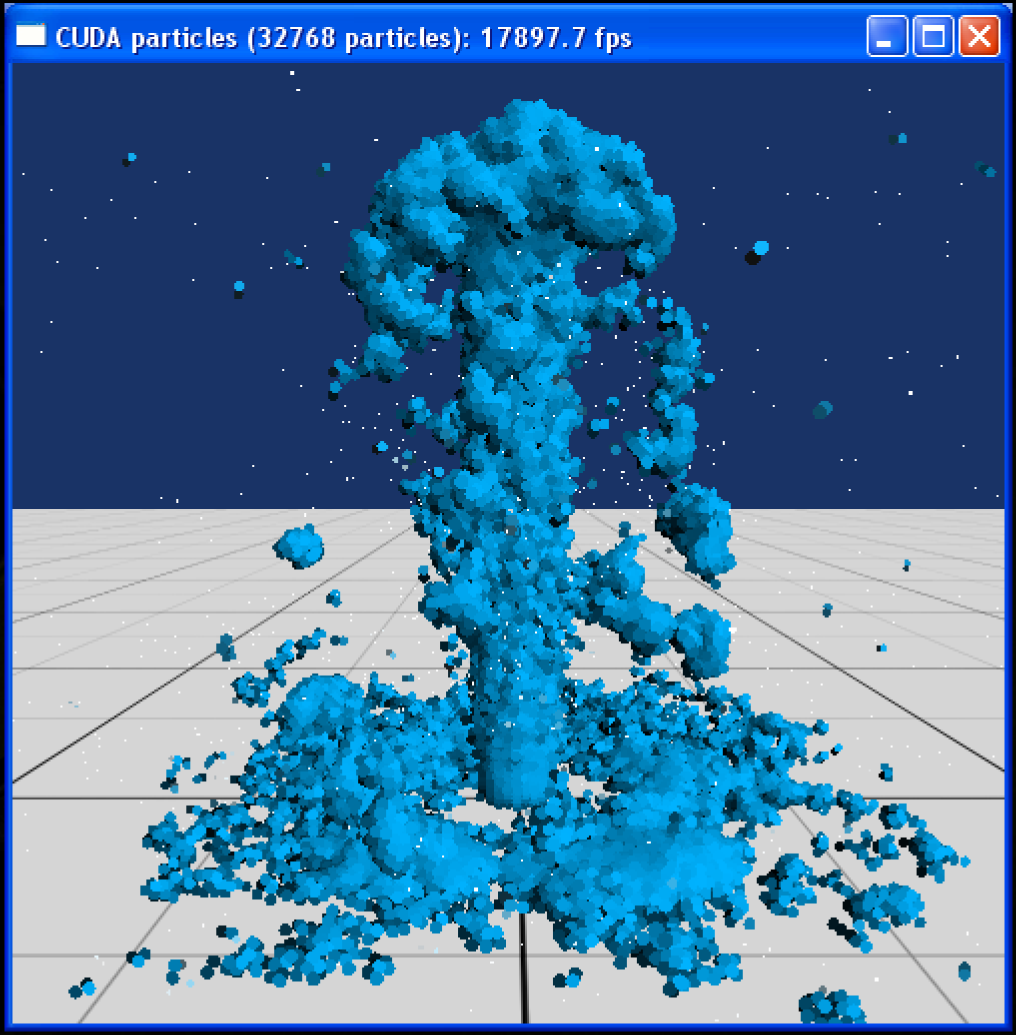
Particle Density



Particle Normal

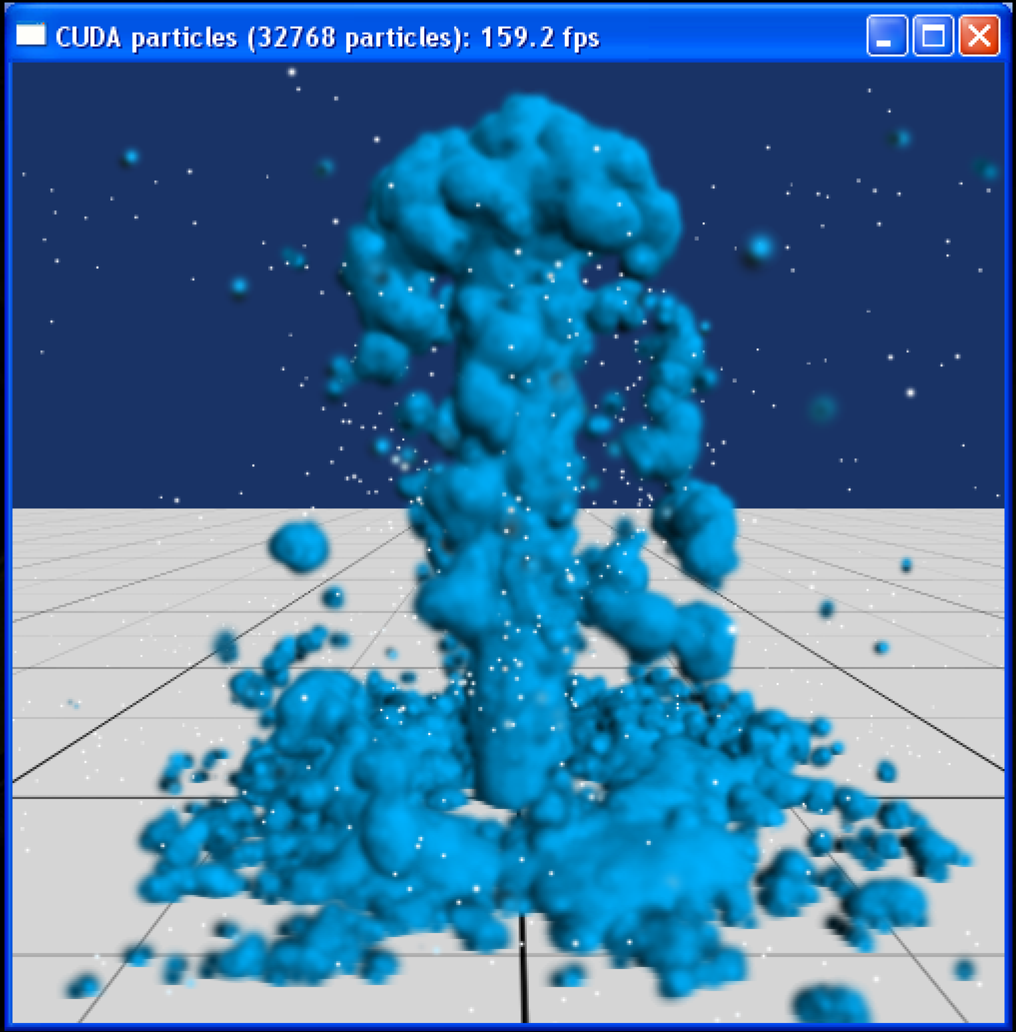


Flat Shaded Point Sprites



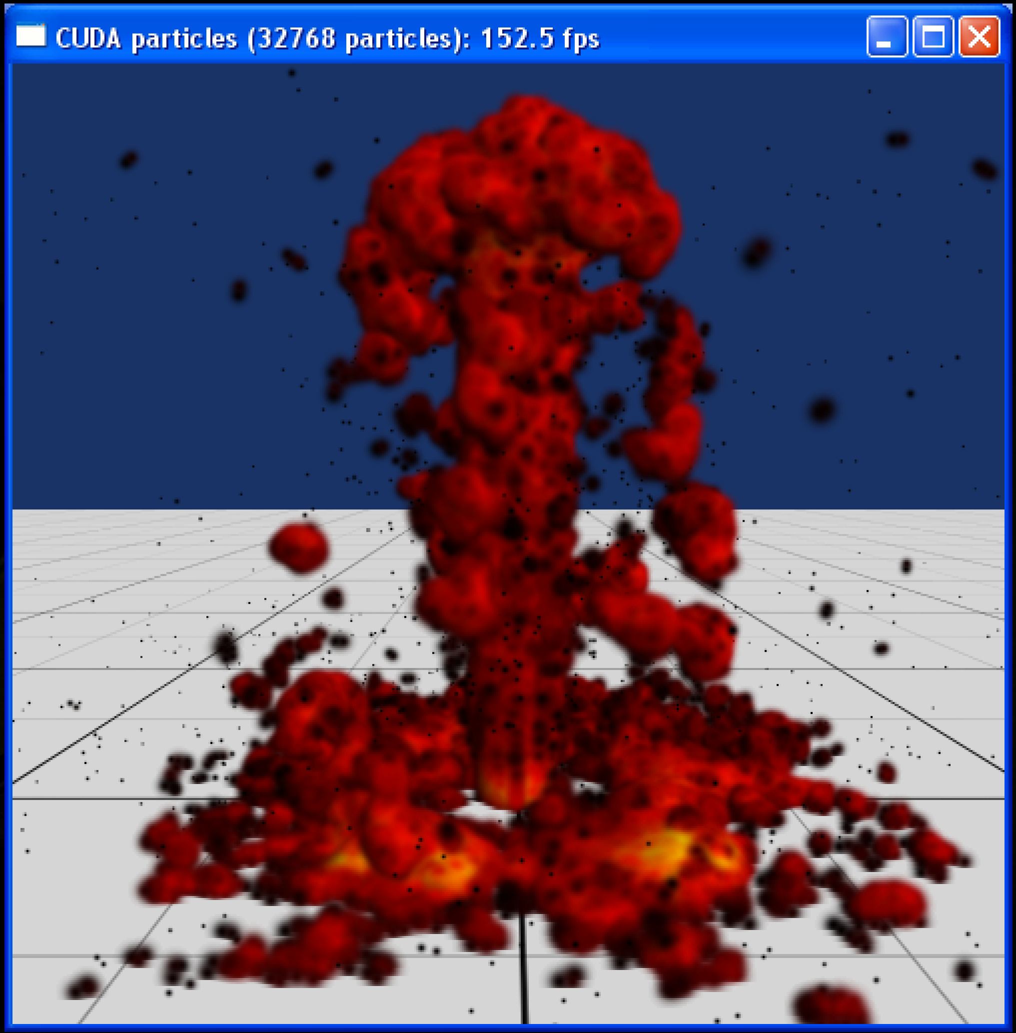
Blended Points Sprites (Splats)

- Scale up point size so they overlap
- Add alpha to points with Gaussian falloff
- Requires sorting from back to front
- Has effect of interpolating shading between points
- Fill-rate intensive, but interactive



Alternative Shading (Lava)

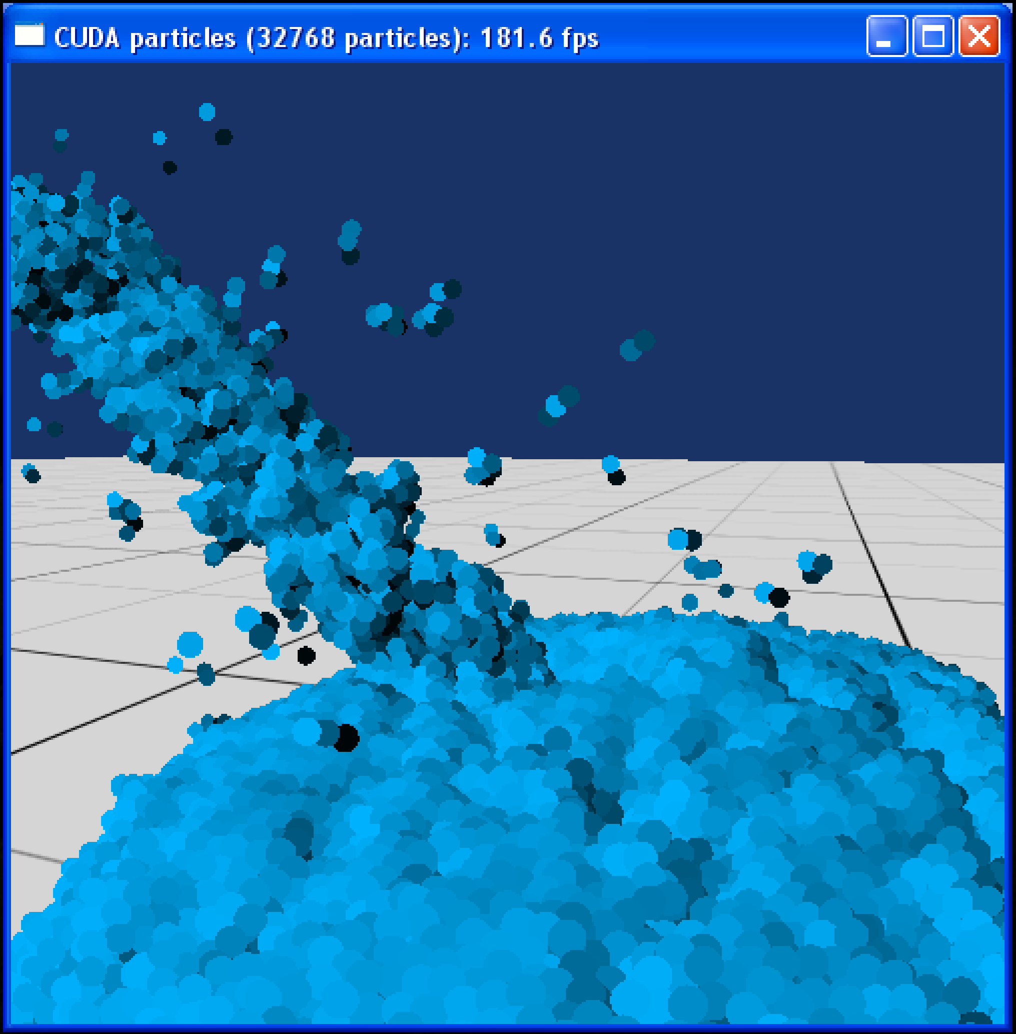
- **Modifies particle color based on density**



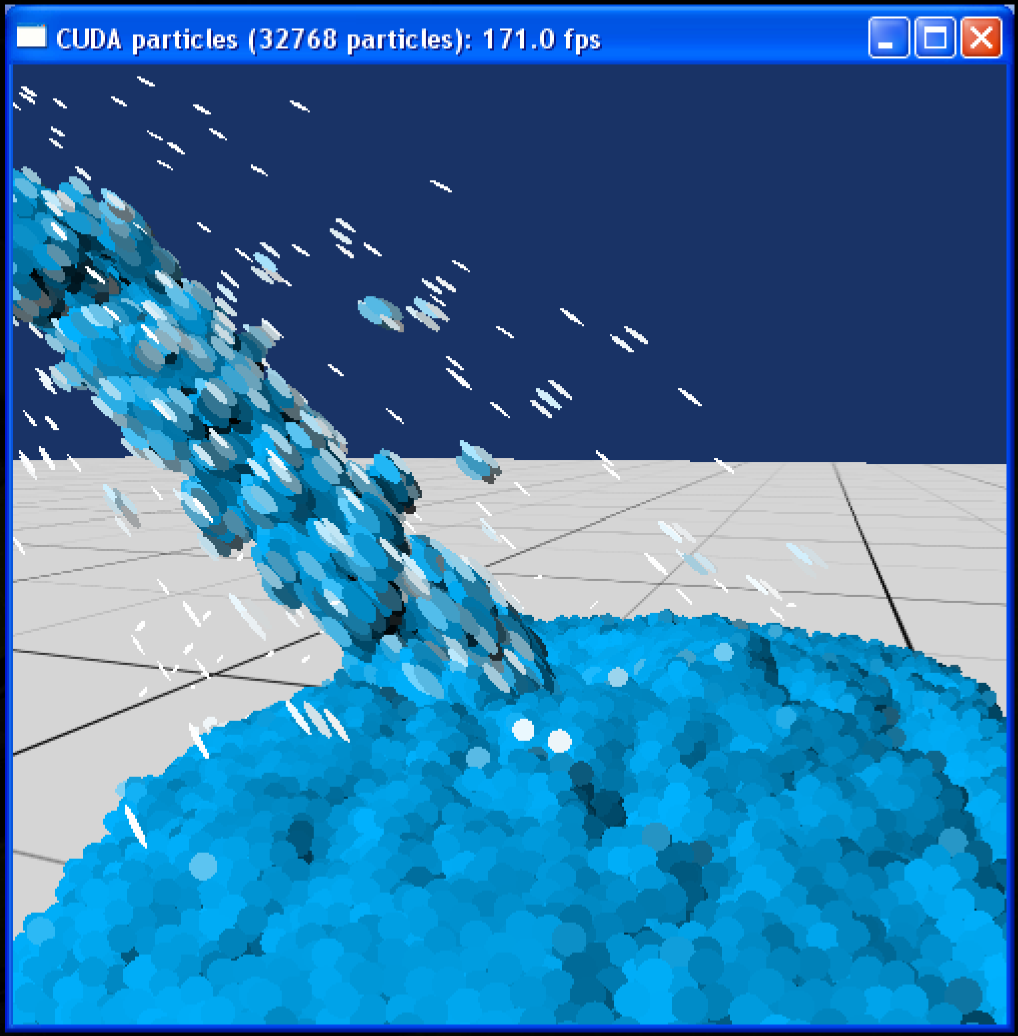
Motion Blur

- Create quads between previous and current particle position
  - Using geometry shader
- Try and orient quad towards view direction
- Improves look of rapidly moving fluids (eliminates gaps between particles)

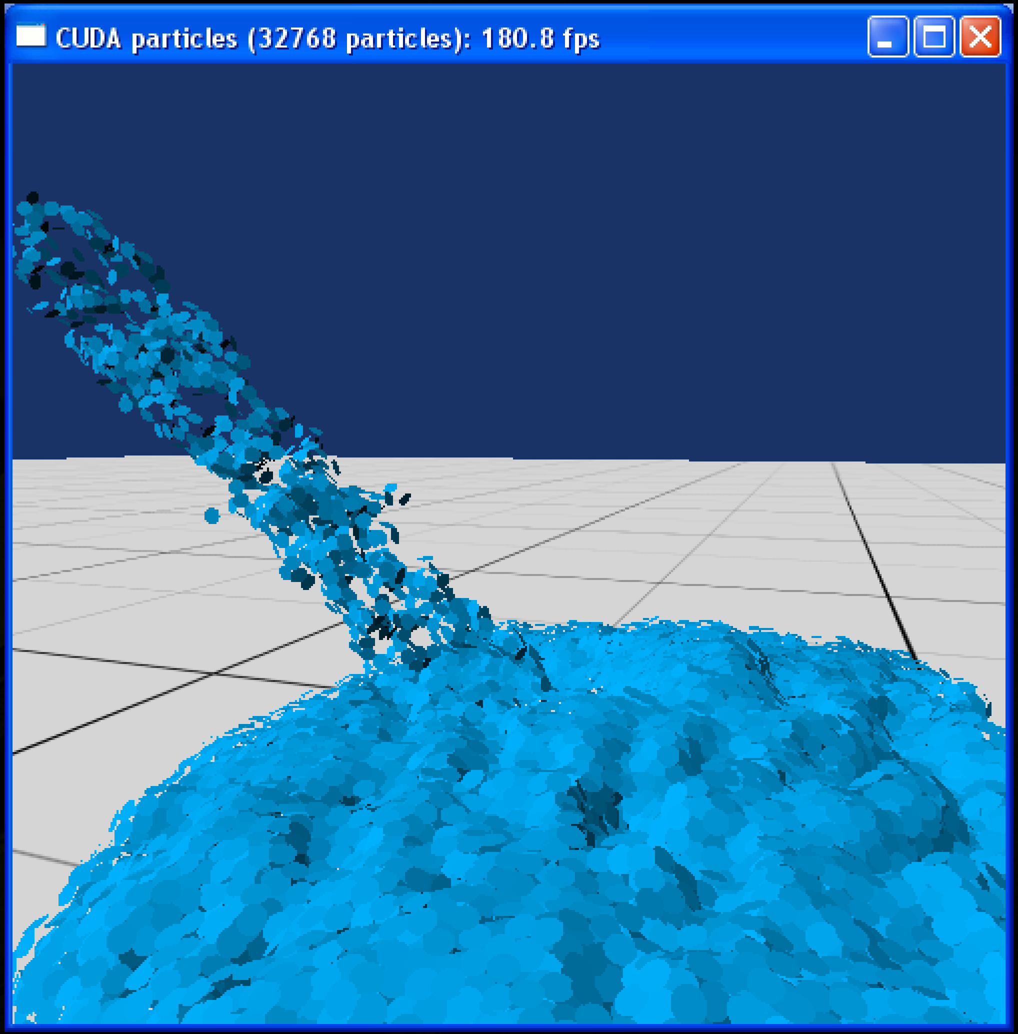
Spheres



Motion Blurred Spheres

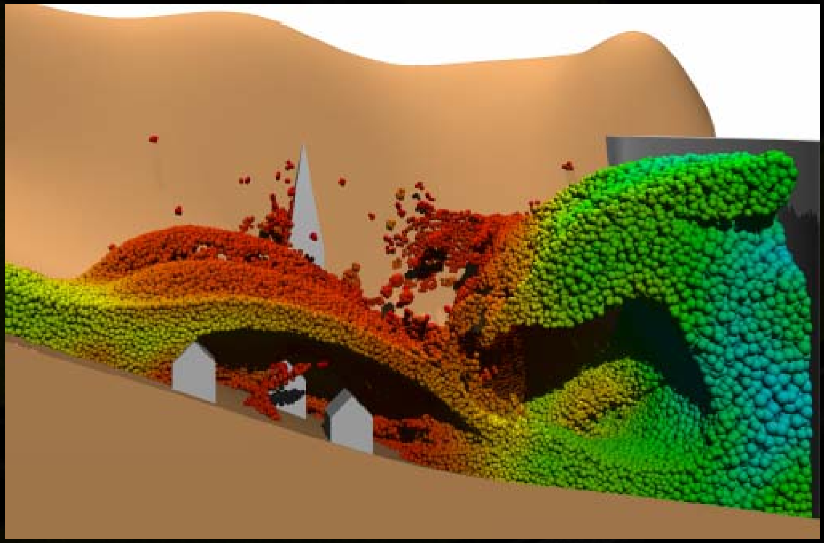


Oriented Discs



The Future

- Practical interactive fluids will need to combine particle, height field, and grid techniques
- GPU performance continues to double every 12 months – lots of room for improvement!



Adaptively Sampled Particle Fluids, Adams 2007

Two way coupled SPH and particle level set fluid simulation, *Losasso, F., Talton, J., Kwatra, N. and Fedkiw, R*

Questions?