Particle-based Fluid Simulation

Simon Green

February 2008

Overview

- Fluid Simulation Techniques
- A Brief History of Interactive Fluid Simulation
- CUDA particle simulation
- Spatial subdivision techniques
- Rendering methods
- Future

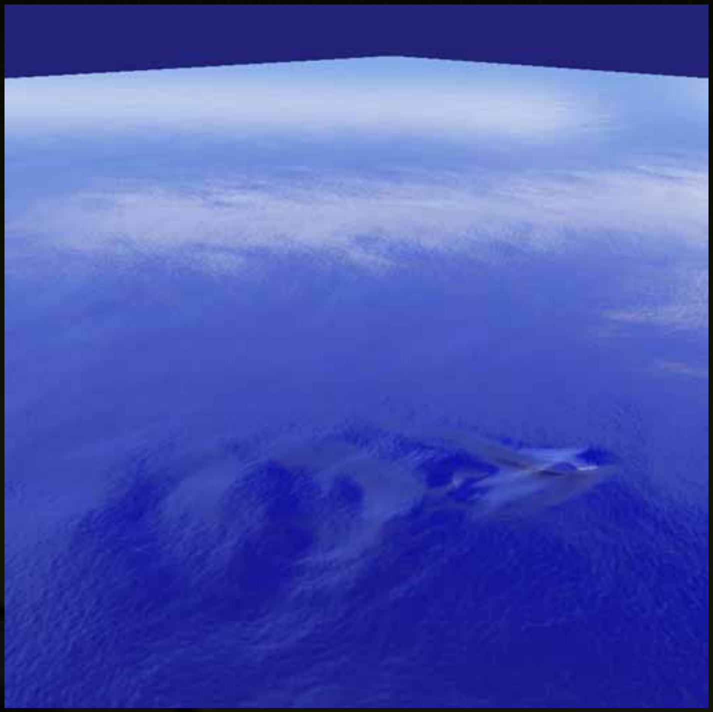
Fluid Simulation Techniques

- 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

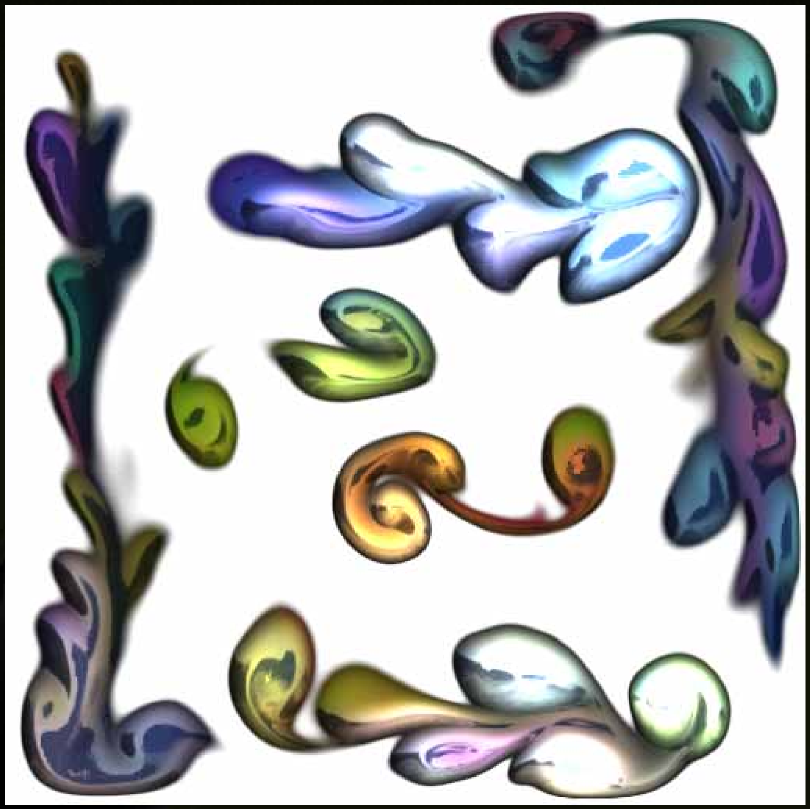
History: 2D Waves

- Old demo-scene trick
- 2D wave equation
- First implemented on GPU by Greg James, 2003
- GeForce 3
- Pixel shader 1.1
- Used register combiners and texture shader
- 8-bit fixed point



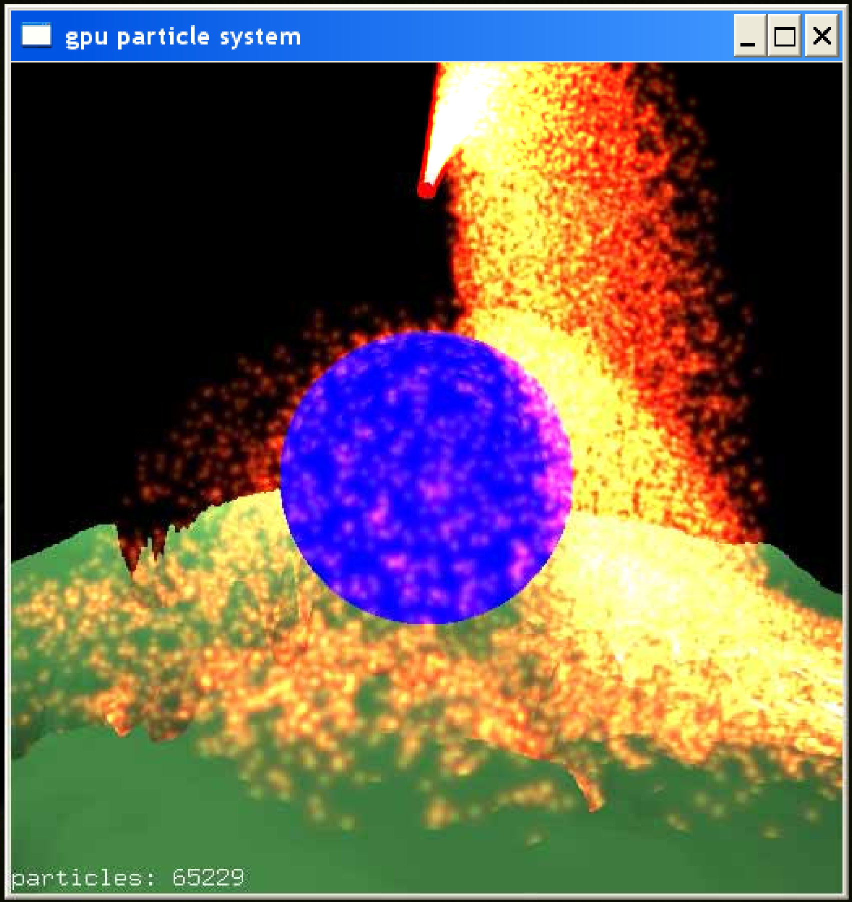
2D Fluid Flow

- Mark Harris, GPU Gems 1
- 2D Navier-Stokes solver
  - based on Jos Stam's Stable Fluids
- GeForce FX
- Used floating point textures
- Cg shaders
- Multiple passes to solve for pressure
- Texture interpolation used for advection step



GPU Particle Systems

- “Building a Million Particle System”, Lutz Latta, GDC 2004
- Position and velocity stored in FP textures
- Simple interactions with terrain height field and implicit shapes
- Emitters done on CPU
- Particles rendered using render-to-vertex array
- 1M particles at ~60fps



3D Fluids - Box of Smoke Demo

- G80 launch demo
- Written by Keenan Crane
- 3D Navier-Stokes solver
- Used tiled 2D textures
- Also tracked free surfaces for water
- BFECC advection scheme
- Ray cast rendering using pixel shader

NVIDIA SDK Smoke Demo

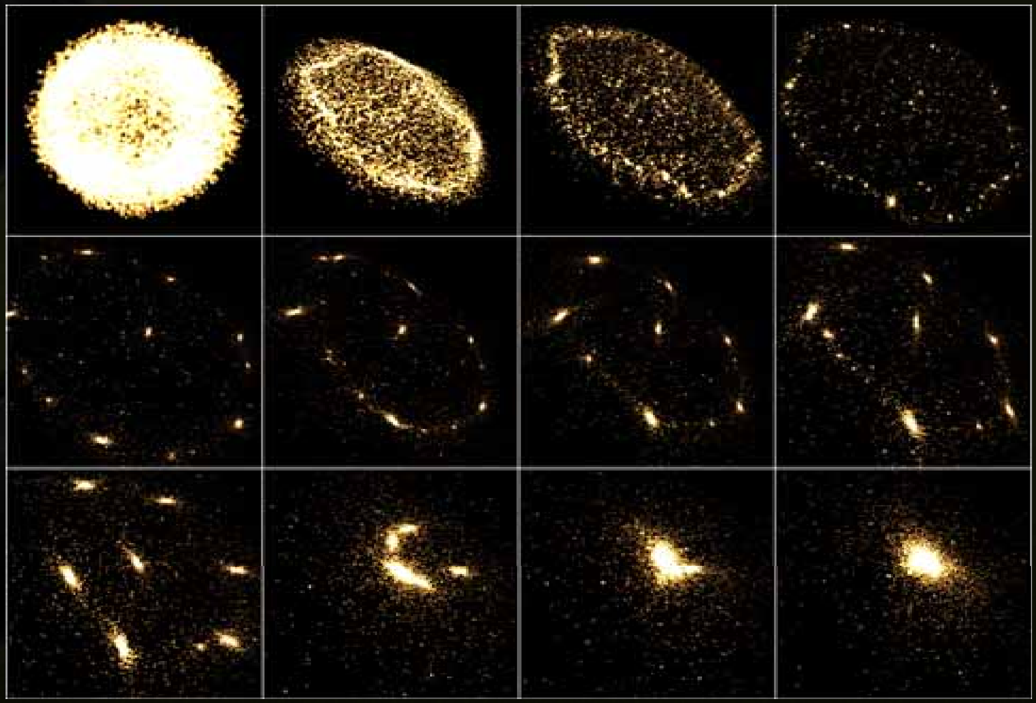
- 3D Navier-Stokes solver
- DirectX 10
- Used render to 3D texture
- Includes interaction with voxelized character

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 Systems

- ***Particle Systems: A Technique for Modeling a Class of Fuzzy Objects*, Reeves 1983**

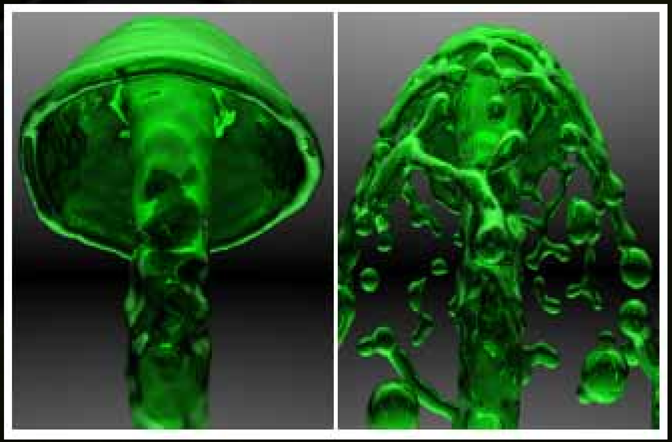
Particle-based Fluid Simulation

- **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**

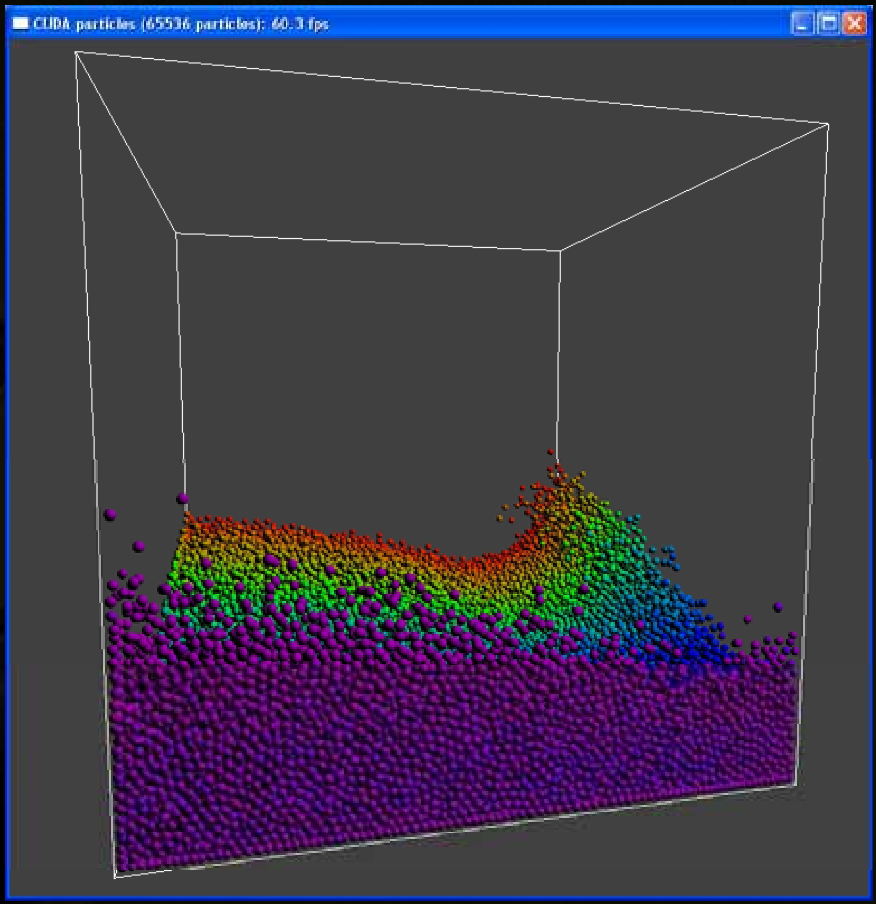
- **Particle-based Viscoelastic Fluid Simulation, Clavet et al, 2005**
- **1000 particles, 10fps**
- **20,000 particles, 2 secs / frame**



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



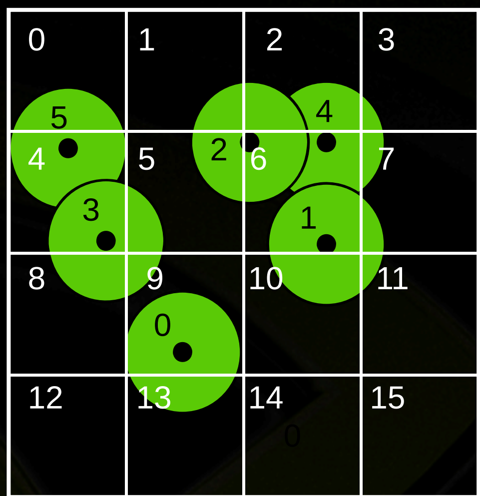
Uniform Grid

- 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: -

Spatial Hashing (Infinite Grid)

- For games, we don't want particles to be constrained to a finite grid
- Solution: use a fixed number of grid buckets, and store particles in buckets based on hash function of grid position
- Pro: Allows grid to be effectively infinite
- Con: Hash collisions (multiple positions hashing to same bucket) causes inefficiency
  - Choice of hash function can have big impact
- See: "*Optimized Spatial Hashing for Collision Detection of Deformable Objects*", Teschner et al.

Example Hash Function

```
__device__ uint calcGridHash(int3 gridPos)
{
     const uint p1 = 73856093;   // some large primes
     const uint p2 = 19349663;
     const uint p3 = 83492791;
     int n = p1*gridPos.x ^ p2*gridPos.y ^ p3*gridPos.z;
     n %= numBuckets;
    return n;
}
```

Smoothed Particle Hydrodynamics (SPH)

- Particle based fluid simulation technique
  - Originally developed for astrophysics simulations
- Interpolates fluid attributes over space using kernel functions
- For games, we can often get away with simpler simulations
  - combine soft particle collisions with attractive forces

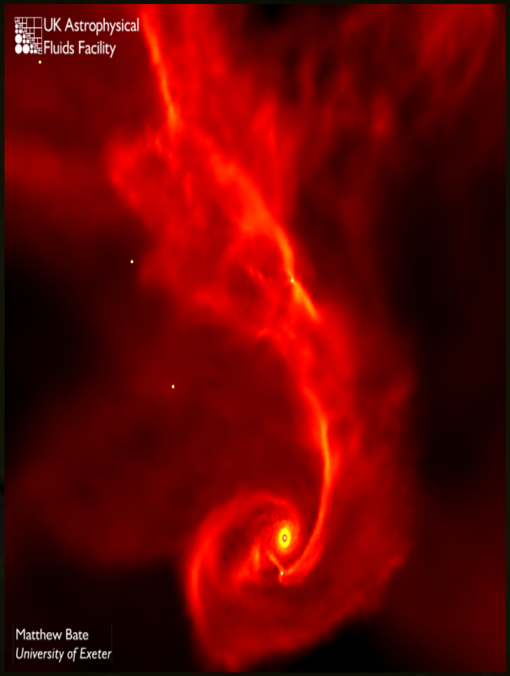


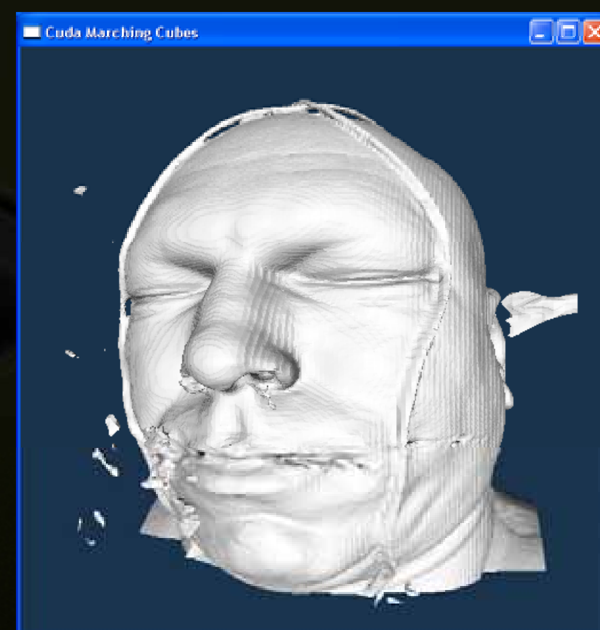
Image courtesy Matthew Bate

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)

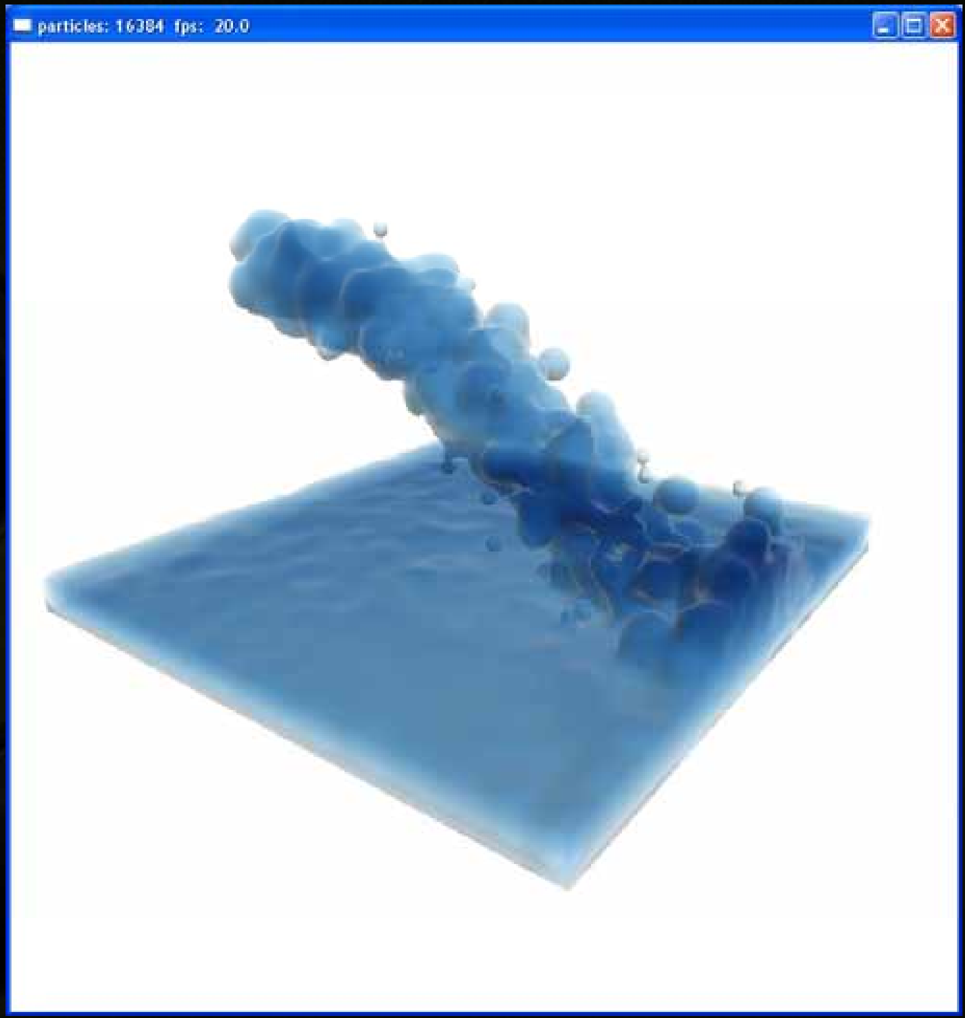
Marching Cubes

- Standard method for extracting isosurfaces from volume data
- CUDA marching cubes uses scan functions from CUDPP library for stream compaction
  - 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



Ray Marching

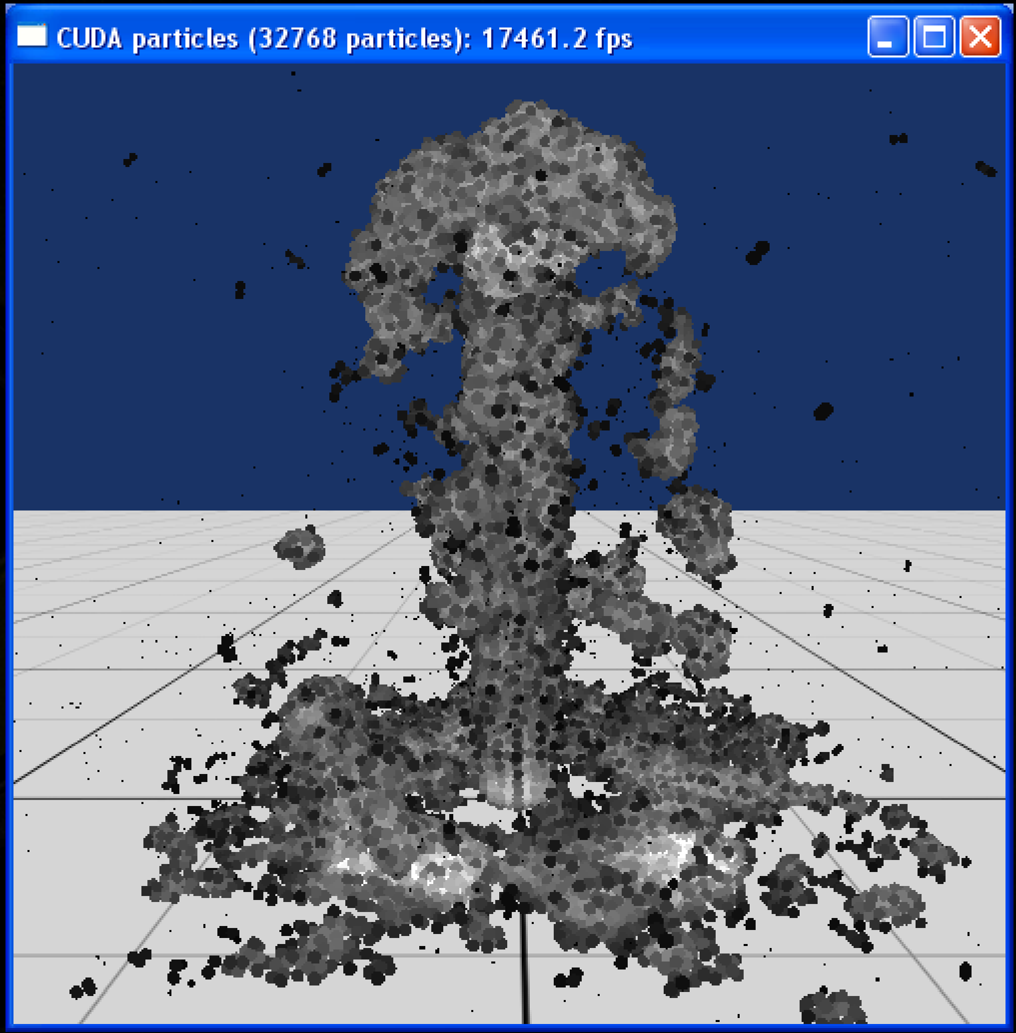
- Volume rendering technique
- Voxelize particles into 3D texture
  - Requires several passes for thickness
- Ray march through 3D texture in pixel shader
  - Can shade based on optical thickness
- Very fill intensive



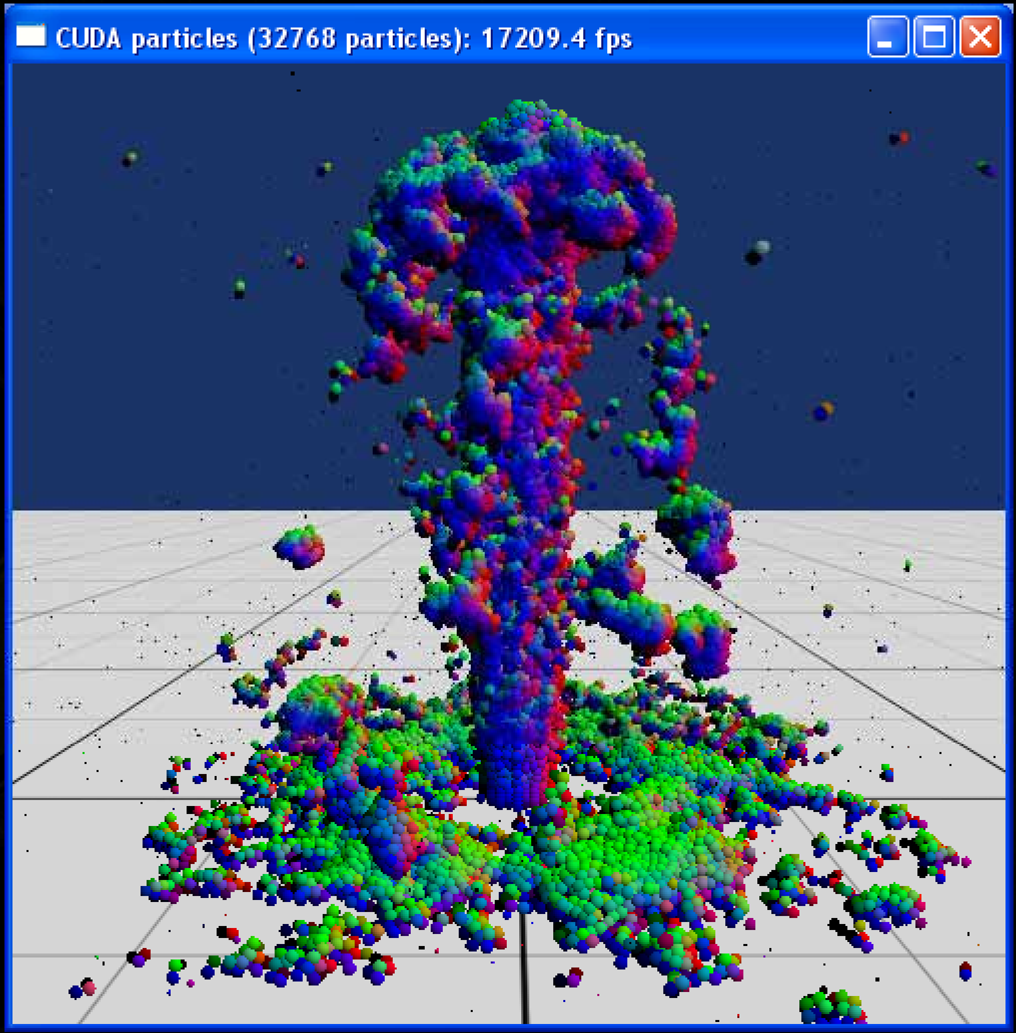
Density-based Shading

- 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

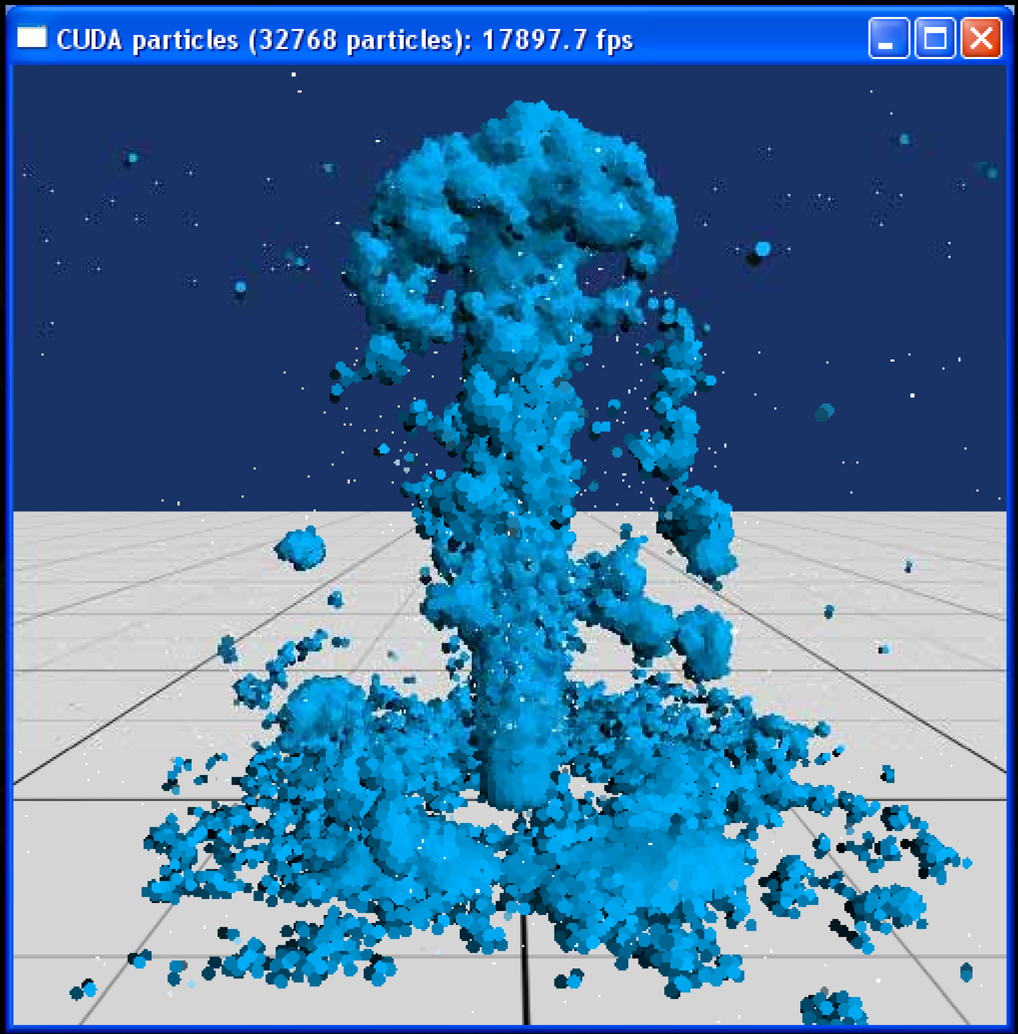
Particle Density



Particle Normal

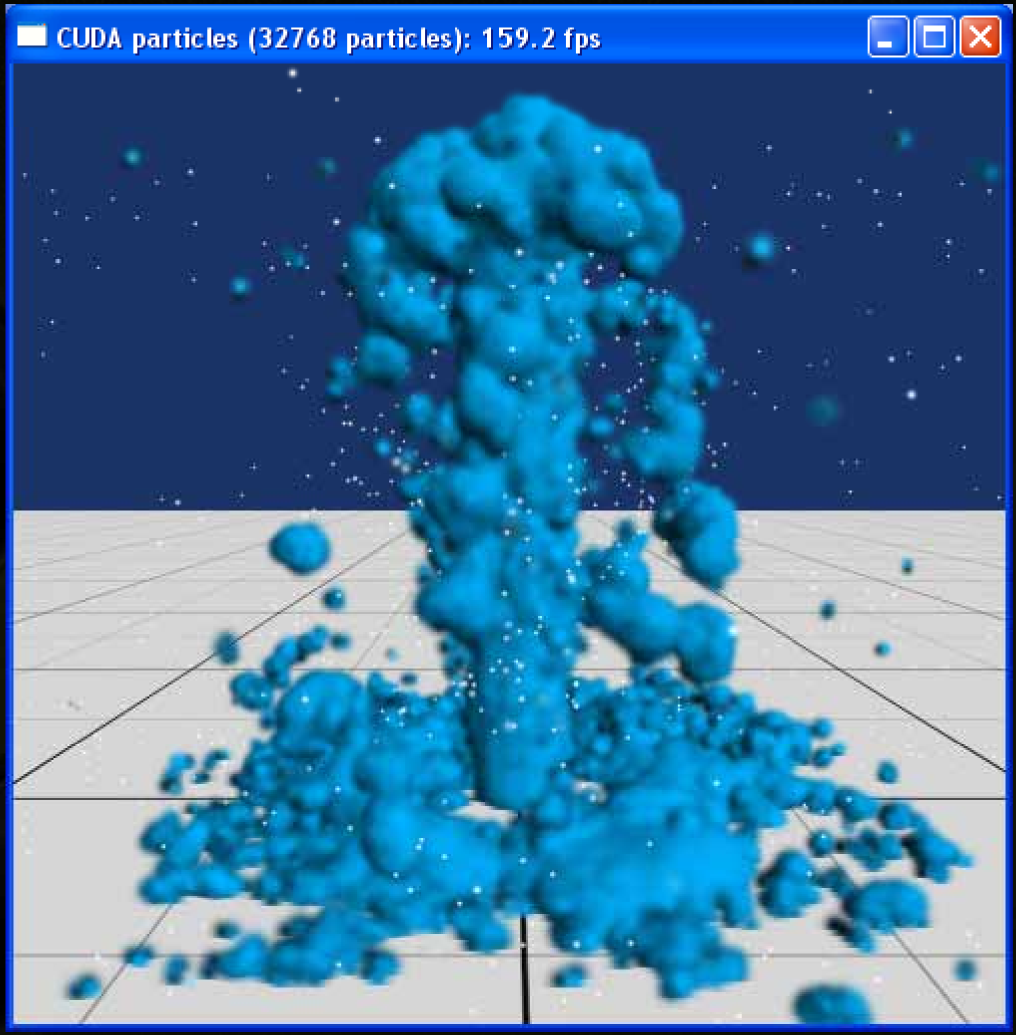


Flat Shaded Point Sprites



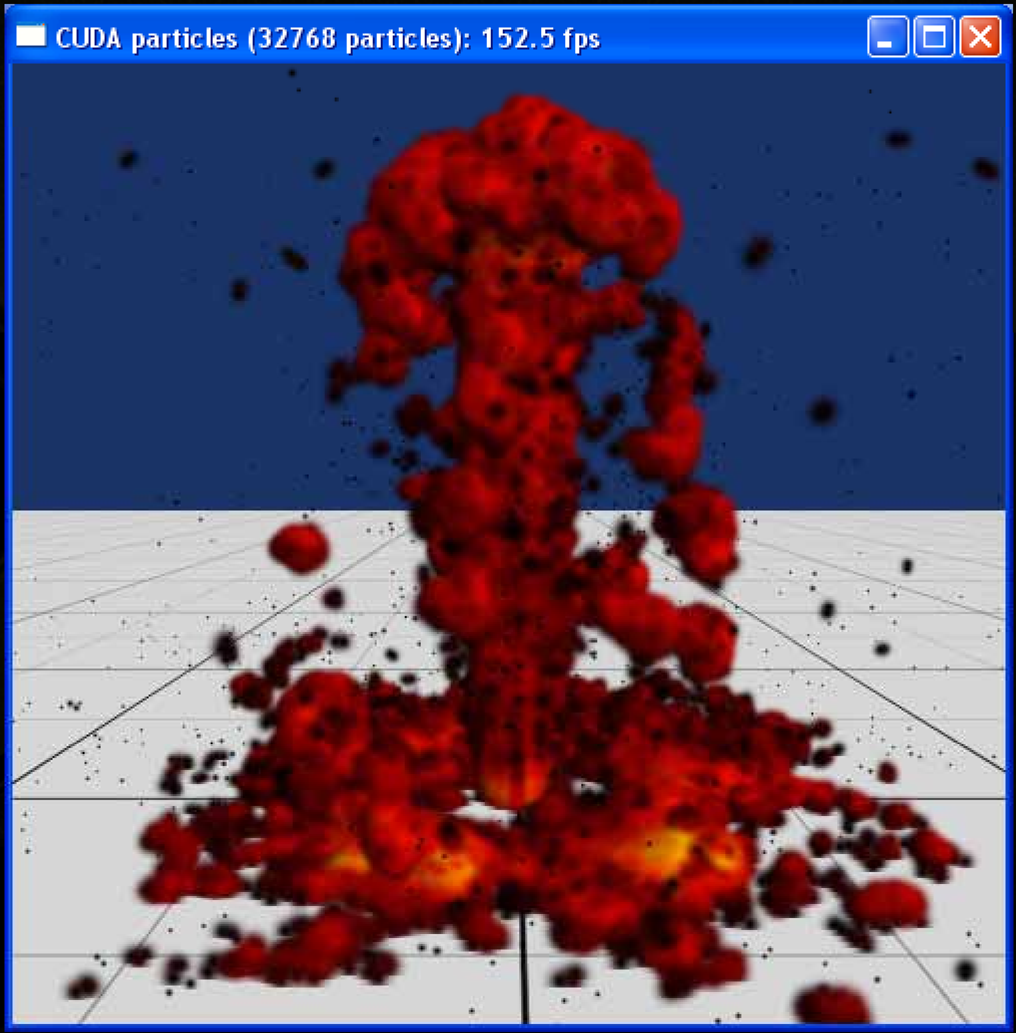
Blended Points Sprites (Splats)

- 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)

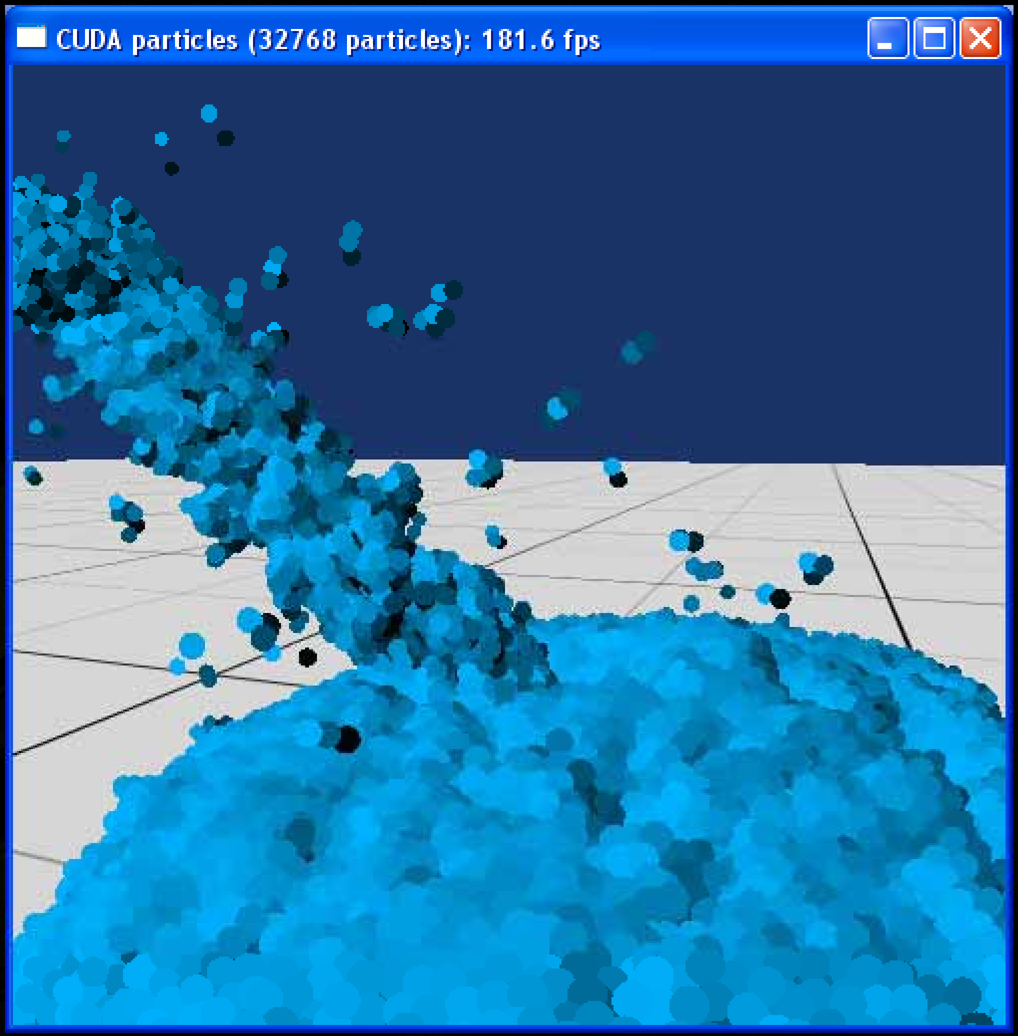
- **Modifies particle color based on density**



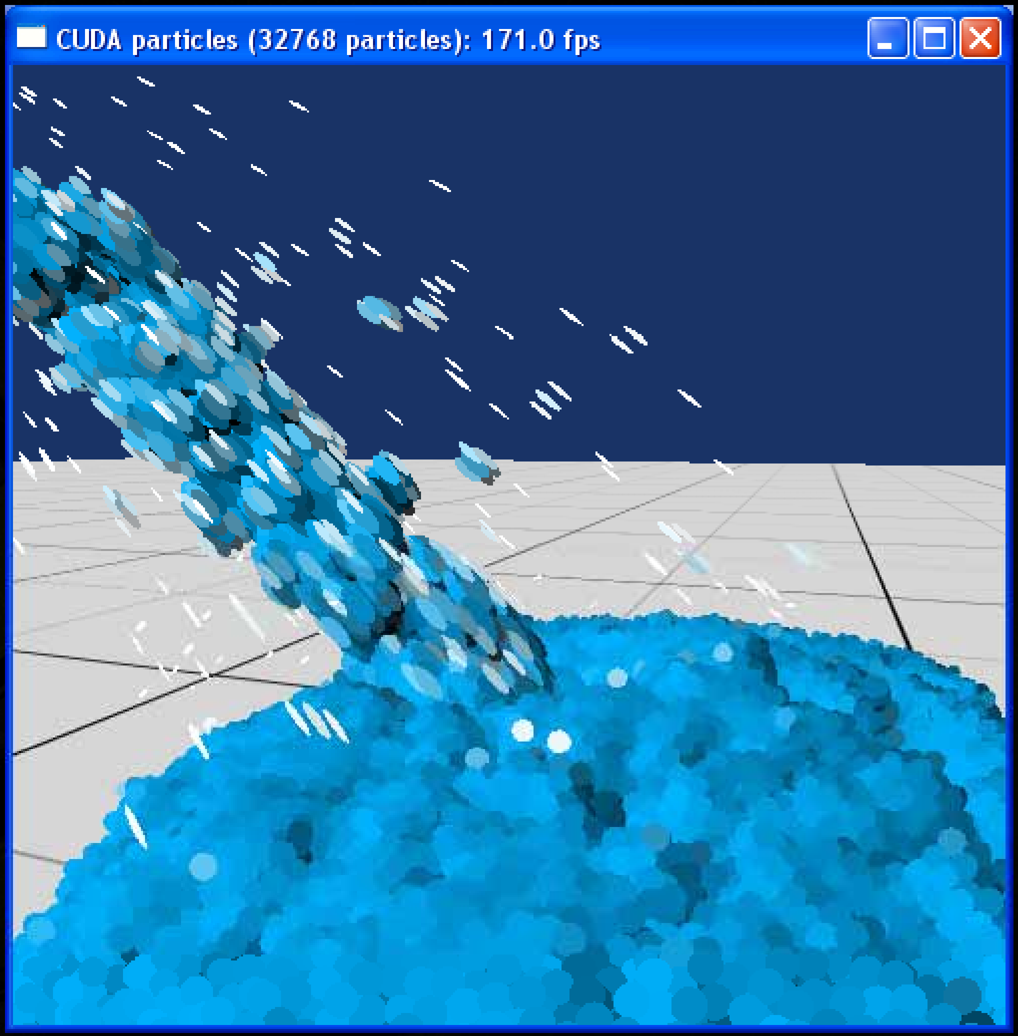
Motion Blur

- 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)

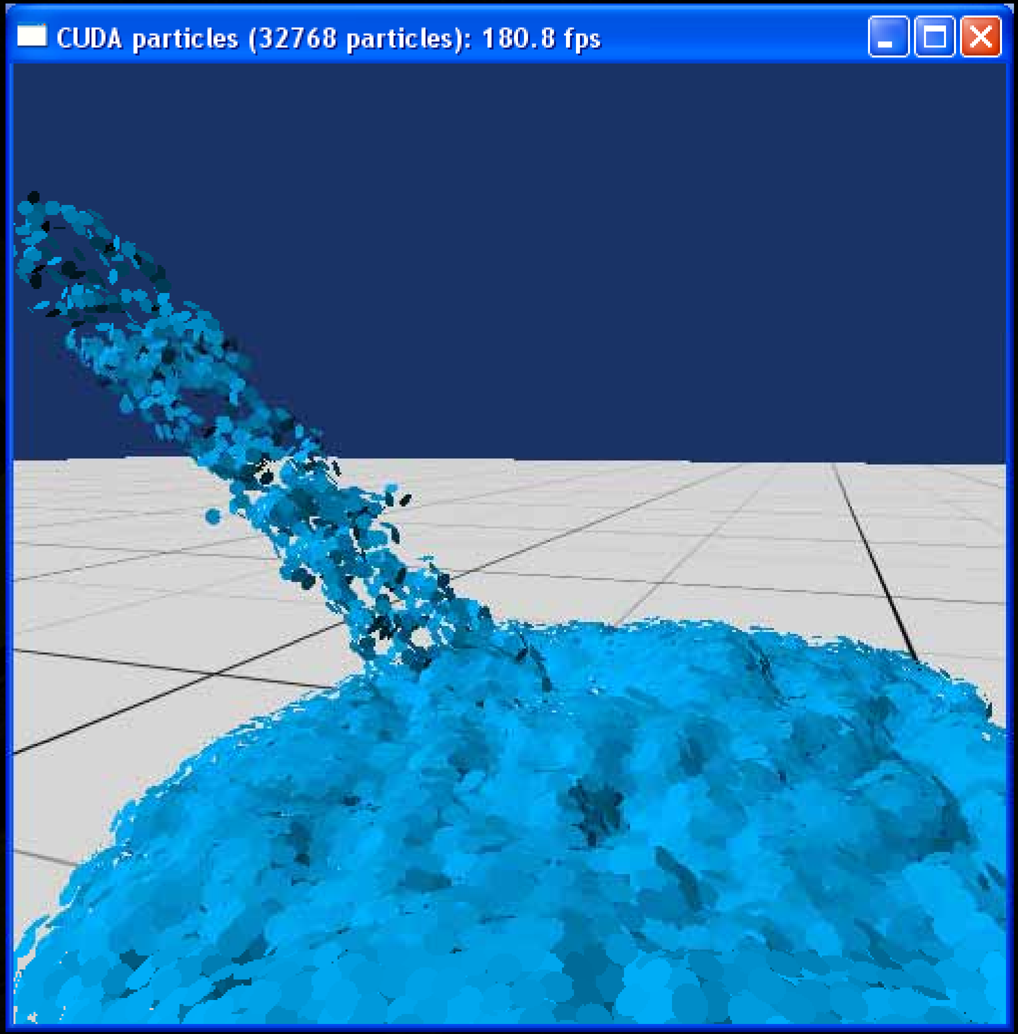
Spheres



Motion Blurred Spheres

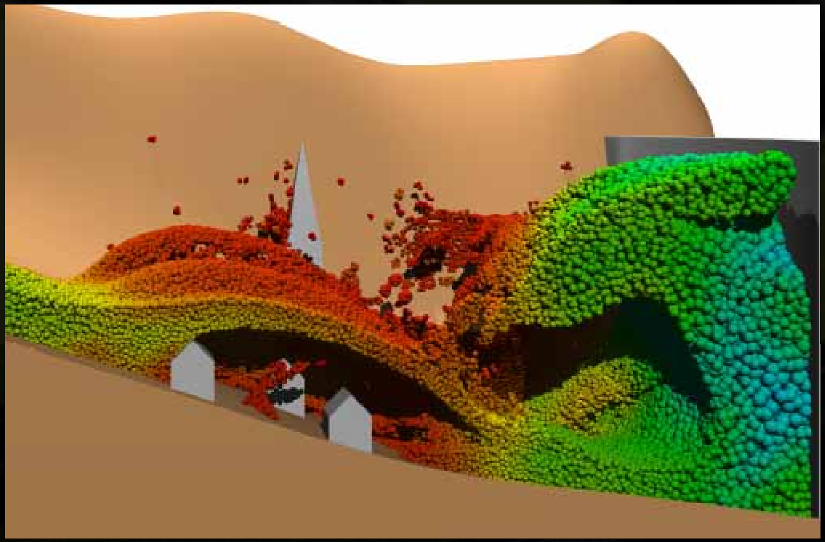


Oriented Discs

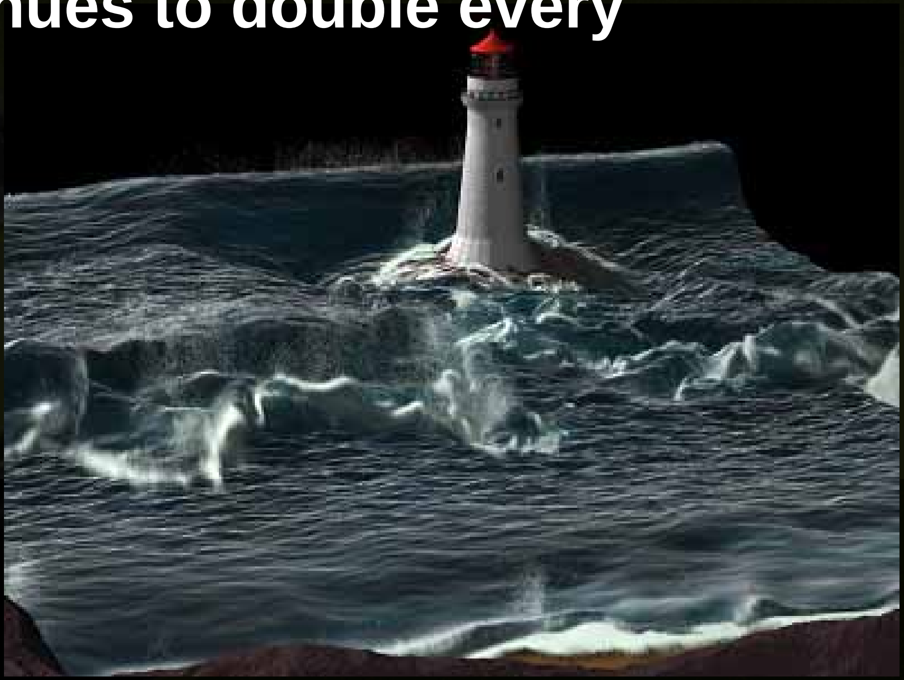


The Future

- **Practical game fluids will need to combine particle, height field, and grid techniques**
- **GPU performance continues to double every 12 months**



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*