GPU Fluid Simulation

(Smoothed-particle Hydrodynamics)

Tina Truong Denis Schmidt Pablo Delgado Krämer

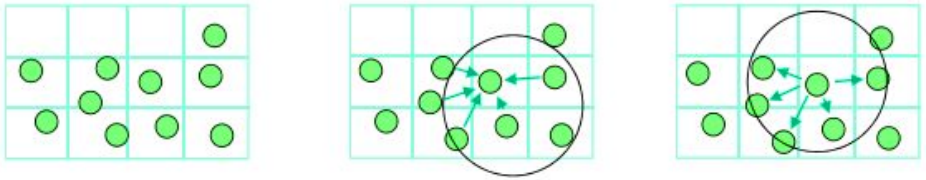
Smoothed-particle Hydrodynamics (SPH)

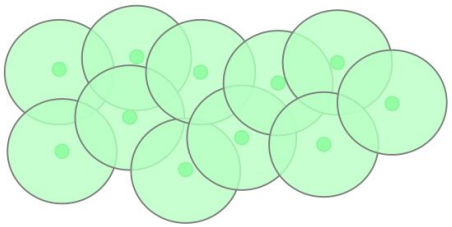
"Particle-Based Fluid Simulation for Interactive Applications" [Müller et al. 2003]

Mass & Momentum Conservation equations (simplified Navier-Stokes)

$$\frac{D\rho}{Dt} = 0 \qquad \frac{D\mathbf{U}}{Dt} = -\frac{1}{\rho}\nabla P + \nu\nabla^2\mathbf{U} + \mathbf{g}$$

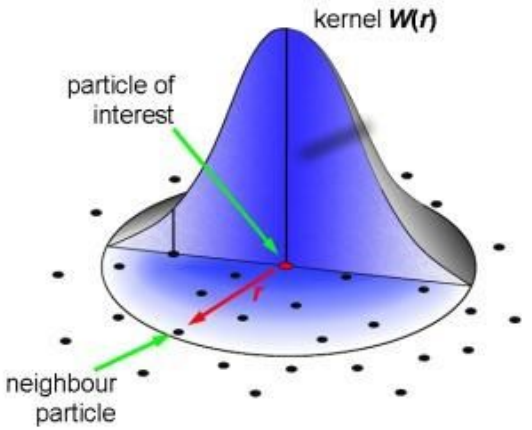
Density & Forces





Smoothing Kernels

- symmetric weighting functions
- used in neighborhood search



$$A_S(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h)$$

GPU Implementation I

"Smoothed Particle Hydrodynamics on GPUs"
[Harada et al. 2007]

$$\rho(\mathbf{x}) = \sum_j m_j W(\mathbf{x} - \mathbf{x}_j)$$

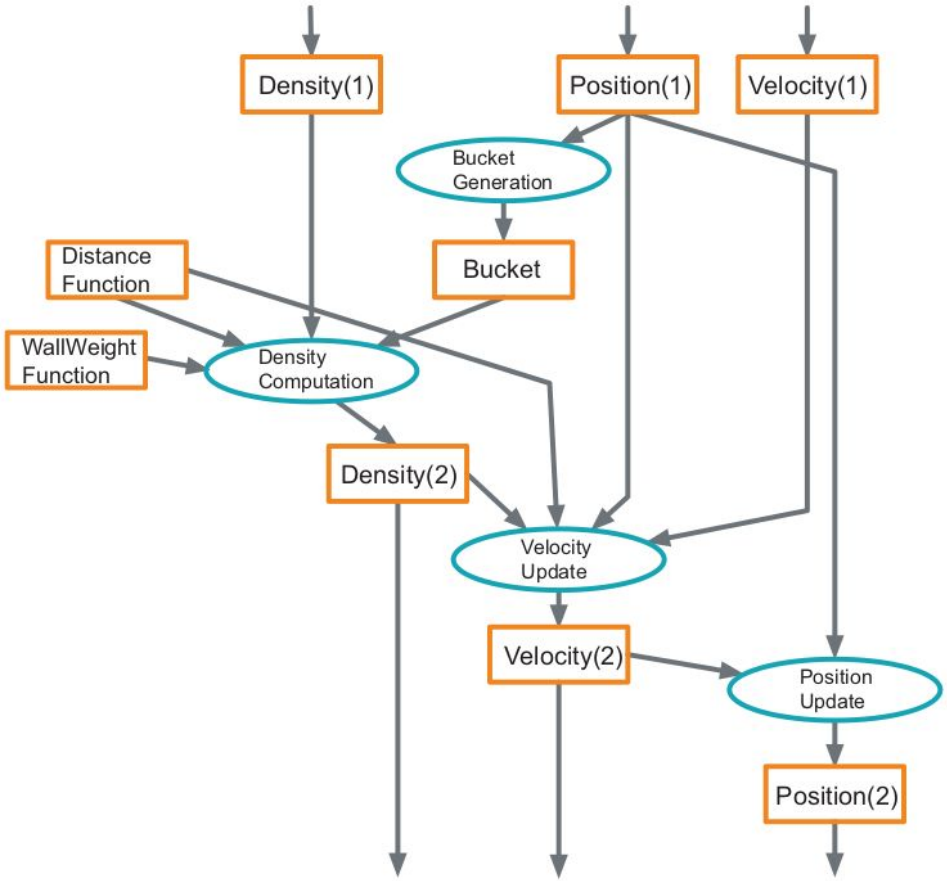
$$p = p_0 + k(\rho - \rho_0)$$

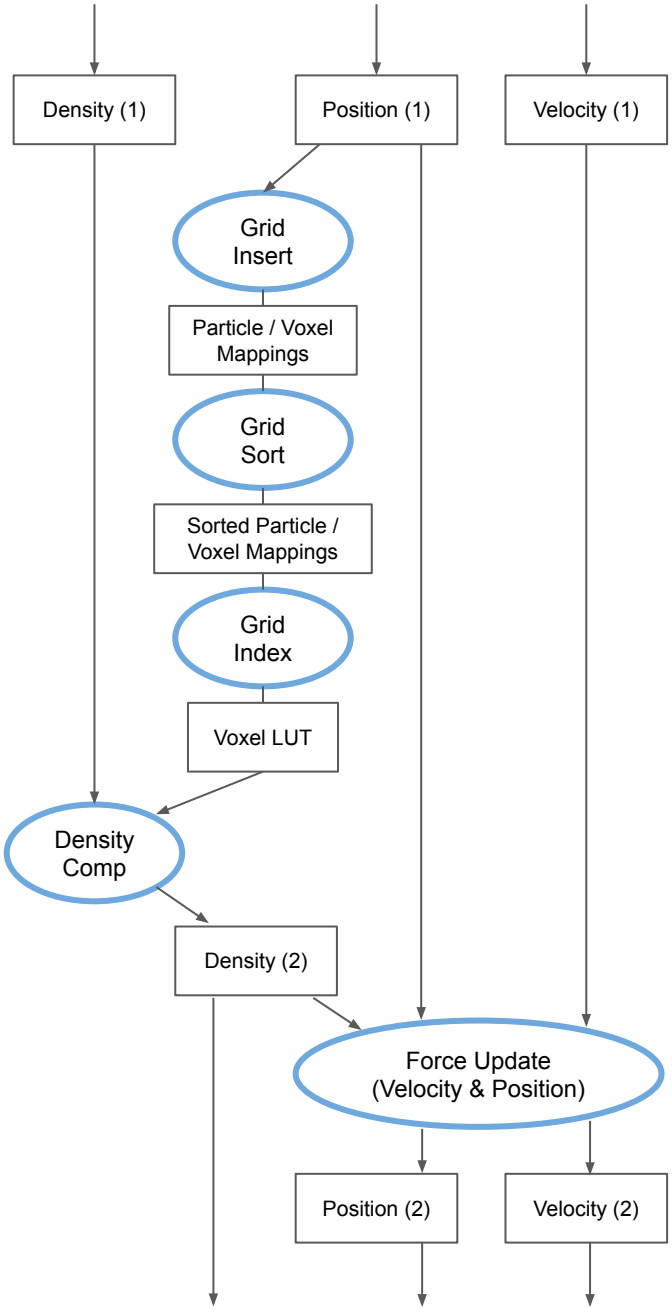
$$\mathbf{F}_i^{press} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W_{press}(\mathbf{r}_{ij})$$

$$\mathbf{F}_i^{vis} = \nu \sum_j m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla W_{vis}(\mathbf{r}_{ij})$$

$$\nabla W_{vis}(\mathbf{r}) = \frac{45}{\pi r_e^6}(r_e - |\mathbf{r}|)$$

$$W(\mathbf{r}) = \frac{315}{64\pi r_e^9}(r_e^2 - |\mathbf{r}|^2)^3 \qquad \nabla W_{press}(\mathbf{r}) = \frac{45}{\pi r_e^6}(r_e - |\mathbf{r}|)^3 \frac{\mathbf{r}}{|\mathbf{r}|}$$





GPU Implementation II

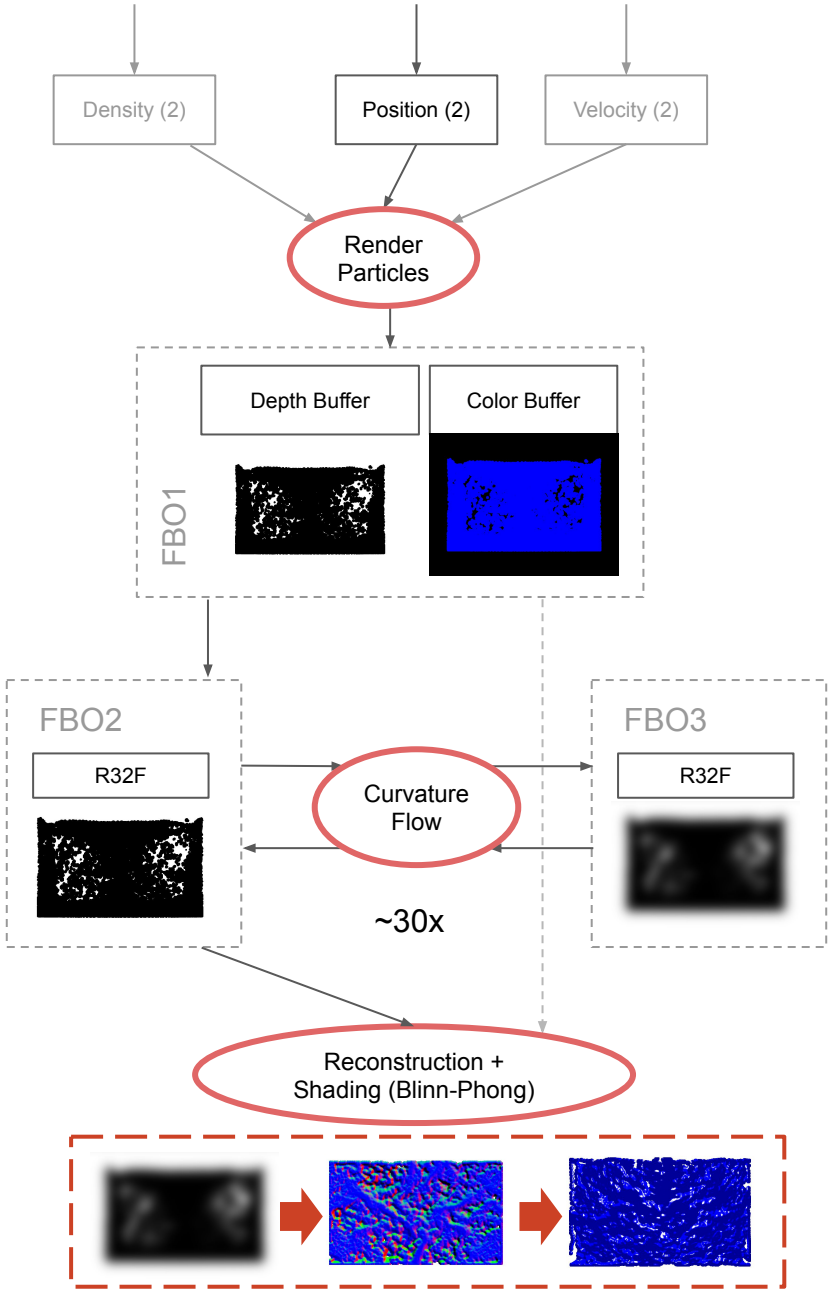
Our Pipeline

1.0 Grid Insert
Fill buffer with <ParticleID, VoxelID> pairs
1.1 Grid Sort
Sort buffer by VoxelID using *bitonic mergesort*
1.2 Grid Index
Find voxel offsets using *parallel binary search*

2.0 Density Computation
Search neighborhood, count and weight particles

3.0 Force Update
Neighborhood search
Calculate pressure from density
Calculate viscosity
Update velocity
Update position

4.0 Rendering
Render screen-space spheres
Fill GBuffer (depth, color)
Apply *curvature flow* (~30 times) on depth
(fragment culling using OBB)
Reconstruct position and normal
Blinn-Phong shading



GPU Implementation III

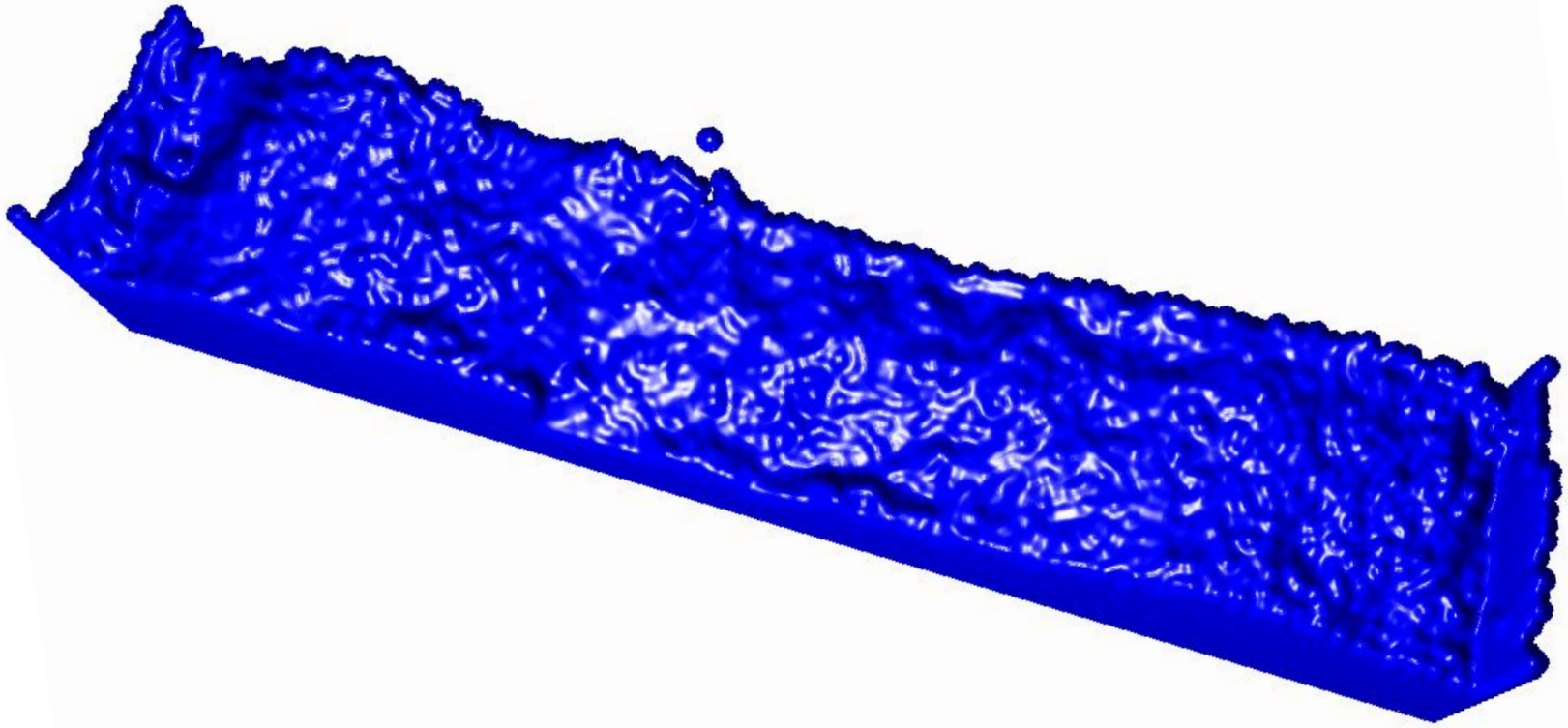
Our Pipeline

1.0 Grid Insert
Fill buffer with <ParticleID, VoxelID> pairs
1.1 Grid Sort
Sort buffer by VoxelID using *bitonic mergesort*
1.2 Grid Index
Find voxel offsets using *parallel binary search*

2.0 Density Computation
Search neighborhood, count and weight particles

3.0 Force Update
Neighborhood search
Calculate pressure from density
Calculate viscosity
Update velocity
Update position

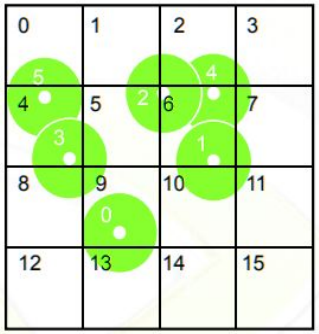
4.0 Rendering
Render screen-space spheres
Fill GBuffer (depth, color)
Apply *curvature flow* (~30 times) on depth
(fragment culling using OBB)
Reconstruct position and normal
Blinn-Phong shading



Thank you for listening!

Bonus Slides I

Uniform Grid Construction



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)

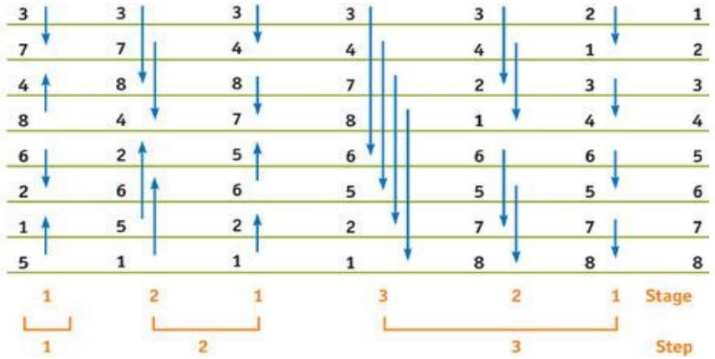
cell start

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

NVIDIA

Bitonic Mergesort

- Input: 2^k elements
- Parallel sorting
- Better alternatives:
  - Radixsort
  - Counting Sort

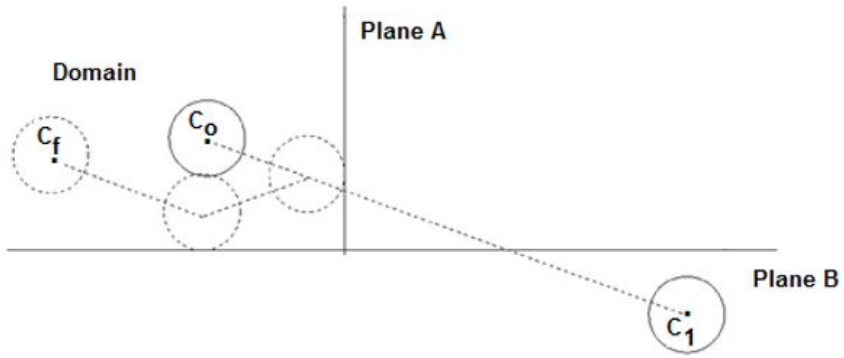


Bonus Slides II

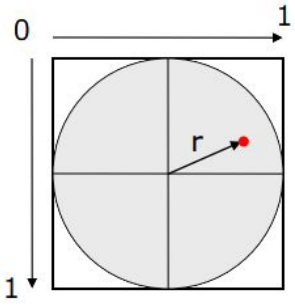
Boundary Condition

Our approach:

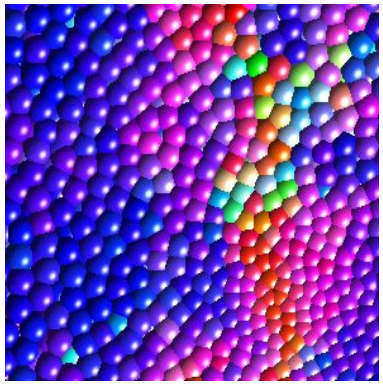
- Bounce off walls
  (reflect velocity)
- Corner-case: corners
- Walls don't have density contribution
  - "virtual" particles are a better solution

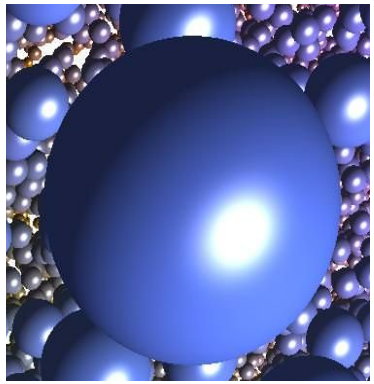


Particle Rendering



- as GL_POINTS
- Screen-space spheres
  (fragment shader)
- Deferred shading
- OBB
- Curvature flow
- Blinn-Phong





```
float3 N;
N.xy = texCoord*2.0-1.0;
float r2 = dot(N.xy, N.xy);
if (r2 > 1.0)
    discard;
N.z = sqrt(1.0 - r2);
```