实时可交互流体模拟
Interactive Fluid Simulation in Real Time

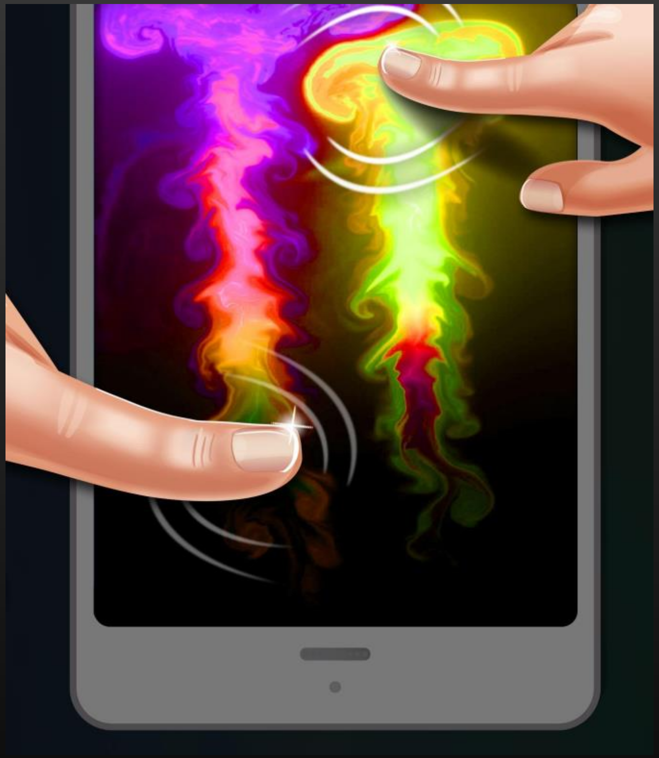
Lecturer：Yu Zhang
Instructor：Xiaosheng Li

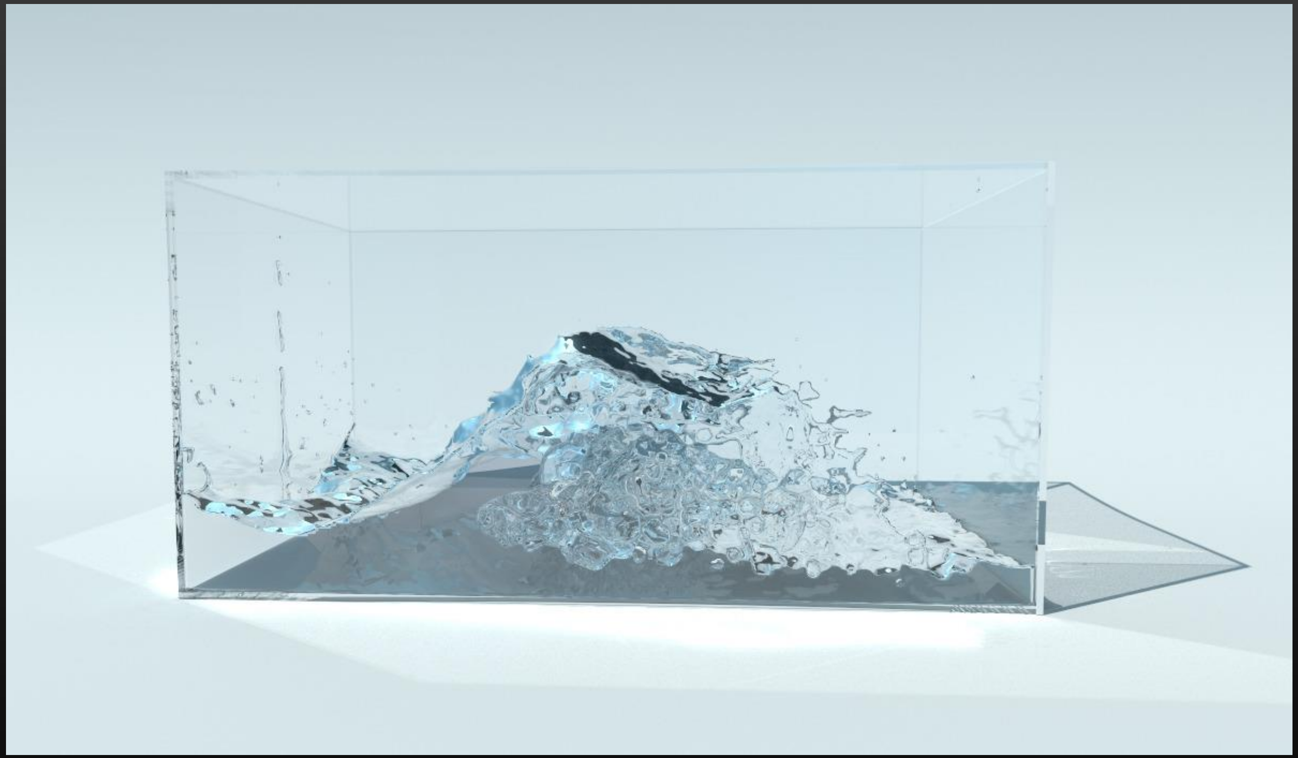
Contents

- Motivation
- Background
- SPH
- PBF
- Rendering
- Mobile
- Conclusion

Motivation

Motivation





Motivation

Aim

- Interactive fluid simulation on mobile in real time

Method

- Solving N-S Equation: SPH & PBF
- Rendering: Marching Cubes & SSFR
- Fully on GPU

Background

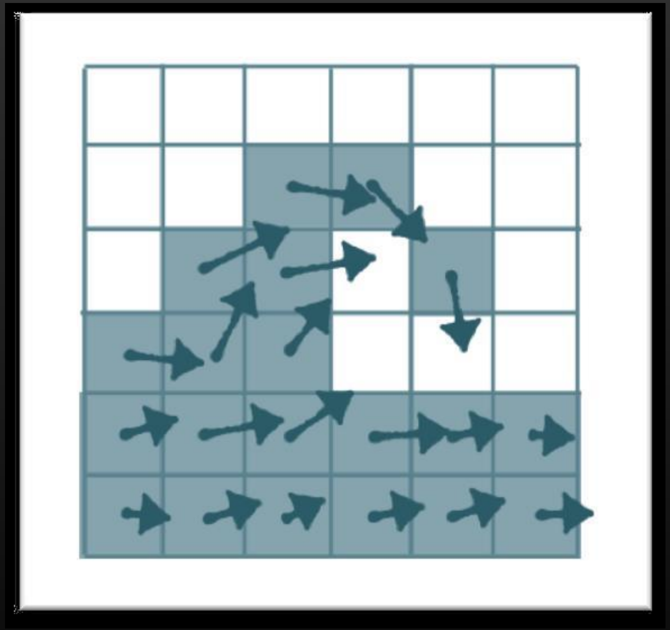
Navier-Stokes Equation

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

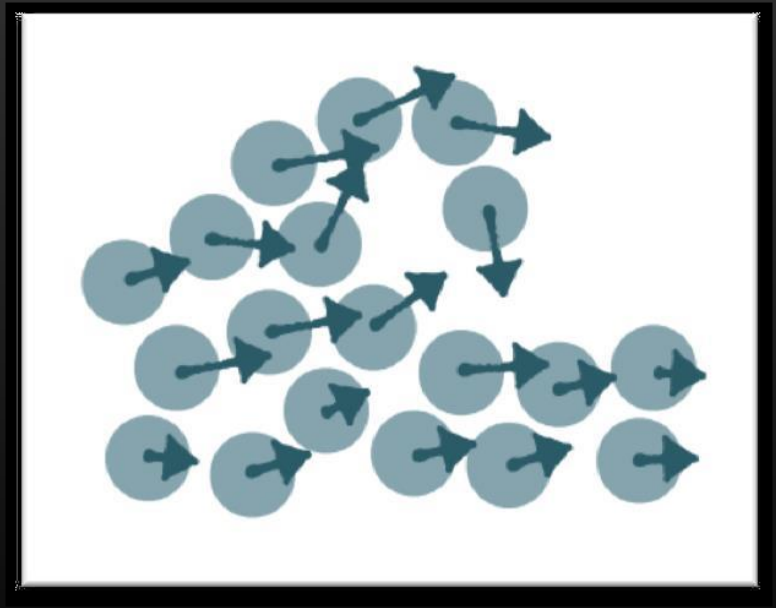
$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0$$

Solving N-S Equation

Eulerian (grids)



Lagrangian (particles)



Fluid Simulation

Scale

- Small: Particle
- Medium: Particle/Grid
- Large: Wave simulation

Solving N-S Equation

Grid

- Advantages
  - Easy to approximate: finite difference
- Disadvantages
  - Mass non-conservation leads to dissipation: sticky & mass losing
  - Massive grids when need details

Particle

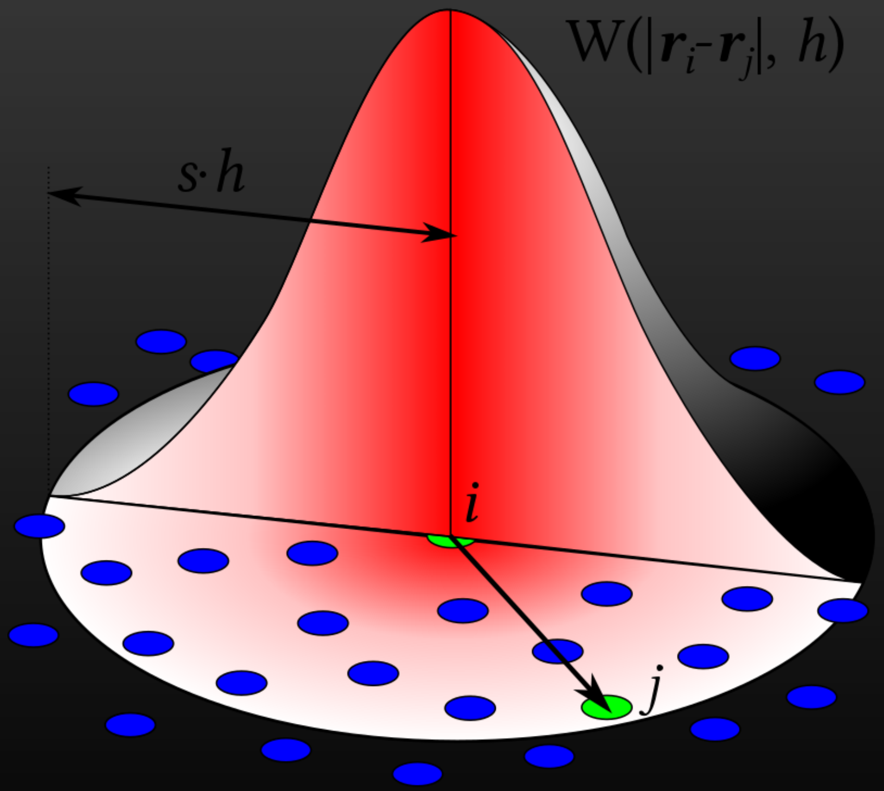
- Advantages
  - Mass conservation
  - Better to simulate details
  - Convenient to couple with other physical simulations
- Disadvantages
  - Neighbor searching

SPH

Smooth Particle Hydrodynamics

SPH Approximation

$$F_i = \sum_j F_j \frac{m_j}{\rho_j} W(\vec{r_i} - \vec{r}_j, h)$$



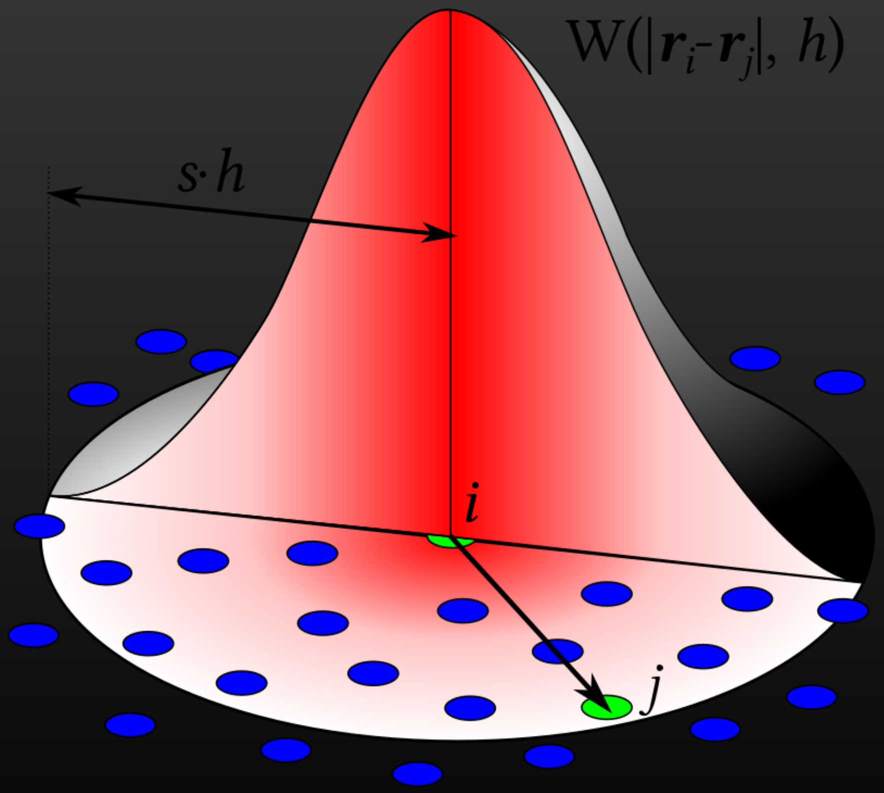
SPH Approximation

Example

$$\rho_i = \sum_j m_j W(\vec{r}_i - \vec{r}_j, h)$$

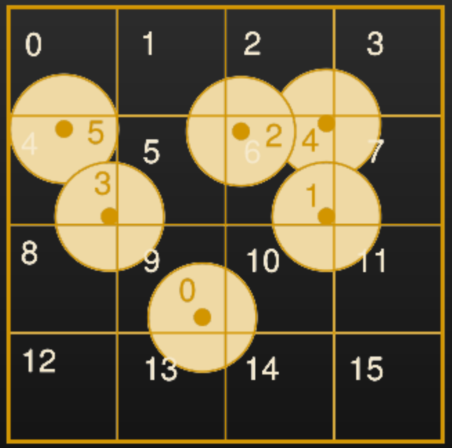
$$\vec{f}_i^{pressure} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(\vec{r}_i - \vec{r}_j, h)$$

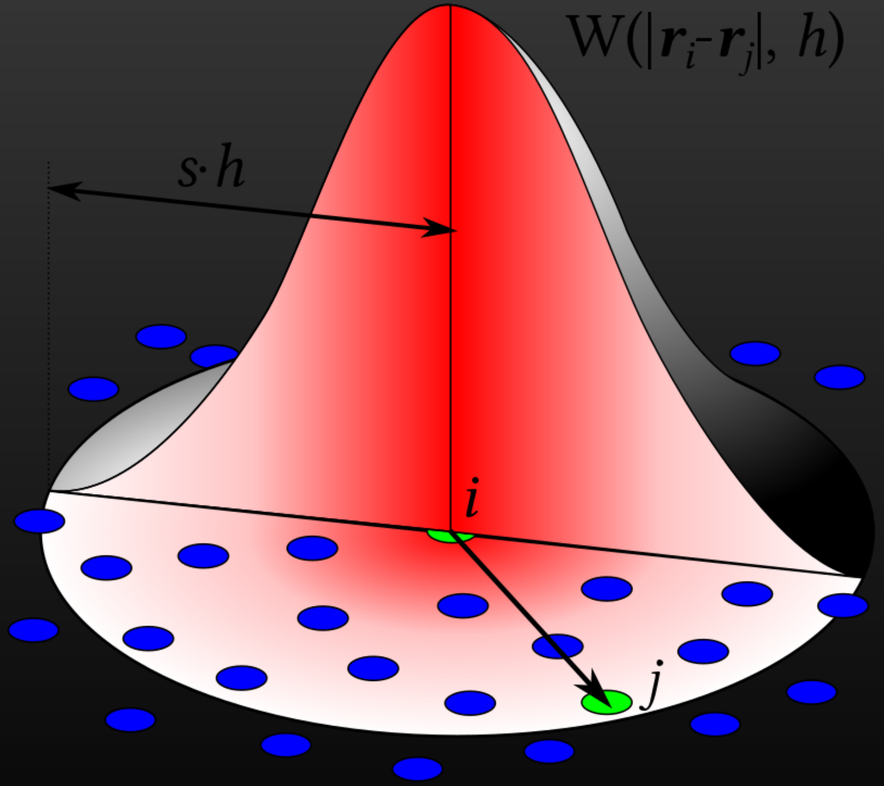
$$\vec{f}_i^{viscosity} = \mu \sum_j m_j \frac{\vec{v}_j - \vec{v}_i}{\rho_j} \nabla^2 W(\vec{r}_i - \vec{r}_j, h)$$



Neighbor Searching

Space partition





Neighbor Searching

Space partition

- List

```
for(int k = 0; k < 27; ++k) {
    int gridIndex = particles[i].gridIndex + spaceCellIndexOffset[k];
    float density = 0;
    for(int j = spaceCellParticleHead[gridIndex]; j != -1; j = particles[j].indexInCell) {
        float3 x = particles[i].position - particles[j].position;
        density += W_poly6(x);
    }
    particles[i].density = density;
}
```

Neighbor Searching

Space partition

- Sorting

```
for(int k = 0; k < 27; ++k) {
    int gridIndex = particles[i].gridIndex + spaceCellIndexOffset[k];
    int head = cellStartIndex[gridIndex];
    int tail = cellStartIndex[gridIndex + 1];
    float density = 0;
    for(int j = head; j < tail; ++j){
        float3 x = particles[i].position - particles[j].position;
        density += W_poly6(x);
    }
    particles[i].density = density;
}
```

Neighbor Searching

Space partition

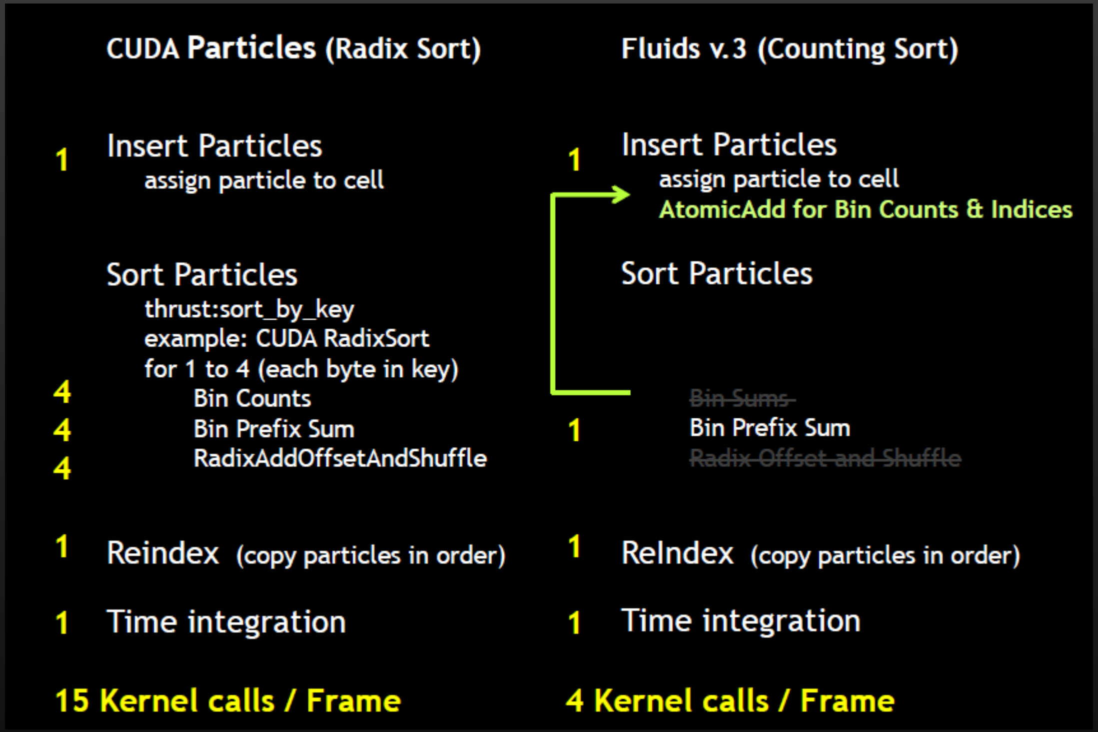
- List ×
- Sorting √

	Locality	Performance (125k particles)	
		Pre-Processing	Neighbor Accessing
List	Very bad!	0.6ms	88ms
Sorting	Good	1.4ms	10ms

Neighbor Searching

Space partition

- Sorting
  - Radix sorting
  - Counting sorting √



Hoetzlein, R. C. "Fast fixed-radius nearest neighbors: interactive million-particle fluids." GPU Technology Conference. Vol. 18. 2014.

Neighbor Searching

Prefix Sum

```
[numthreads(SPH_GROUP_SIZE, 1, 1)]
void updateSpaceCellIndex (uint groupIdx : SV_GroupIndex, uint3 groupId : SV_GroupID) {
    uint i = groupIdx + groupId.x * SPH_GROUP_SIZE;
    if(i < 0 || i >= particleNum) return;
    particles[i].gridIndex = spaceCellFlatIndex(particles[i].position);
    int start;
    InterlockedAdd(spaceCellParticleCountTemp[particles[i].gridIndex], 1, start);
    particles[i].indexInCell = start;
}
```

Neighbor Searching

Prefix Sum

```
[numthreads(SPACE_GROUP_SIZE, 1, 1)]
void updateArrayPrefixSumInGroup (uint groupIdx : SV_GroupIndex, uint3 groupId : SV_GroupID) {
    uint gi = groupIdx + groupId.x * SPACE_GROUP_SIZE;
    uint li = groupIdx;
    for(uint p = 0; (1<<p) < SPACE_GROUP_SIZE; ++p) {
        uint PP = (1<<p);
        if((p&1)==0) {
            uint temp = spaceCellParticleCountTemp[gi];
            temp += (li>=PP?spaceCellParticleCountTemp[gi-PP]:0);
            spaceCellParticleCount[gi] = temp;
        } else {
            uint temp = spaceCellParticleCount[gi];
            temp += (li>=PP?spaceCellParticleCount[gi-PP]:0);
            spaceCellParticleCountTemp[gi] = temp;
        }
        AllMemoryBarrierWithGroupSync();
    }
}
```

Neighbor Searching

Prefix Sum

```
[numthreads(SPACE_GROUP_SIZE, 1, 1)]
void updateArrayPrefixSumBetweenGroup (uint groupIdx : SV_GroupIndex, uint3 groupId : SV_GroupID) {
    uint groupNum = (spaceCellNum + SPACE_GROUP_SIZE - 1) / SPACE_GROUP_SIZE;
    uint li = groupIdx;
    for(uint j = 1; j < groupNum; ++j) {
        spaceCellParticleCount[j*SPACE_GROUP_SIZE + li] += spaceCellParticleCount[j*SPACE_GROUP_SIZE-1];
        AllMemoryBarrierWithGroupSync();
    }
}
```

Neighbor Searching

Optimization in GPU Sorting

- Locality
- Atomic operations
- Global/Shared memory
- Bank conflicts

SPH Algorithm

1. Neighbor searching
2. Update density and pressure
3. Update forces
4. Update velocity and position
5. Collision handling

Problems

$$\rho_i = \sum_j m_j W(\vec{r}_i - \vec{r}_j, h)$$

- Density problem
  - Single particle mass is set very large to get expected density
- Adjust parameters to achieve expected simulation effect
  - Mass
  - Volume
  - Time Step
  - Smooth Radius (h)

PBF

Position Based Fluids

Position Based Dynamics (PBD)

Example

$$C(\vec{p}) = 0$$

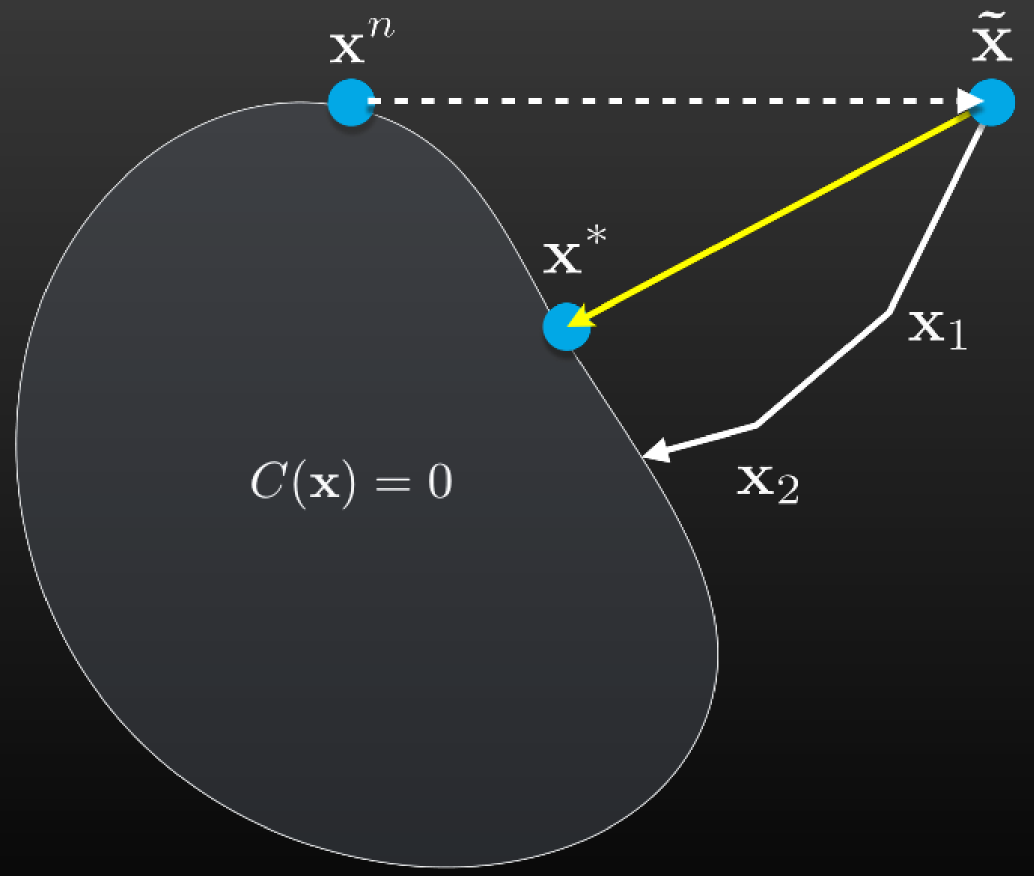
$$C_{circle}(\vec{p}) = |\vec{p} - \vec{c}| - r$$

$$C_{stretch}(\vec{p}_1, \vec{p}_2) = |\vec{p}_1 - \vec{p}_2| - d$$

Position Based Dynamics (PBD)

$$C(\vec{p} + \Delta\vec{p}) \cong$$
$$C(\vec{p}) + \nabla_{\vec{p}}\, C(\vec{p}) \cdot \Delta\vec{p} = 0$$

$$\Delta\vec{p} = \lambda \nabla_{\vec{p}} C(\vec{p})$$



Position Based Dynamics (PBD)

$$\Delta\vec{p} = -\frac{C(\vec{p})}{\left|\nabla_{\vec{p}}C(\vec{p})\right|^2}\nabla_{\vec{p}}C(\vec{p})$$

Position Based Fluids (PBF)

- SPH + PBD

$$\rho_i = \sum_j m_j W(r_i - r_j, h)$$

$$C_i(r_1, r_2, \ldots, r_n) = \frac{\rho_i}{\rho_0} - 1$$

Solving Constraints

- Direct Method ×
- Iterative Method √
  - Gauss-Seidel method
  - Jacobi method √

	Single Thread	**Multiple Threads**
Gauss-Seidel	Converges fast	×
Jacobi	Converges slow	√

PBF Algorithm

1. Neighbor searching
2. Predict position
3. Jacobi iteration
   1. Neighbor searching
   2. Calculate lambda
   3. Correct Position
4. Update velocity
5. XSPH

Demo

Video

- SPH *vs.* PBF (125k particles)

Comparison

	Parameters	**Stability**	**Performance**	**Scalability**
SPH	Fake parameters, time consuming to adjust parameters	Not stable, sensitive to parameters	Adjust parameters to optimize performance	Not easy to couple with other physical simulation
PBF	Real physical parameters	Stable. Increase iterations	Reduce iterations	Convenient to couple with others physical simulation due to the feature of PBD (PhysX, Flex)

Demo

Video

- Real Time Simulation (PBF, 6 iterations, 16k particles, 60fps)

Current Performance

PBF, 6 iterations, compute shader in Unity, GTX 750Ti

Particles	27k	64k	125k	216k
FPS (No Rendering)	75	45	25	15

Future Optimization

Locality

Atomic operations

Global/Shared memory

Bank conflicts

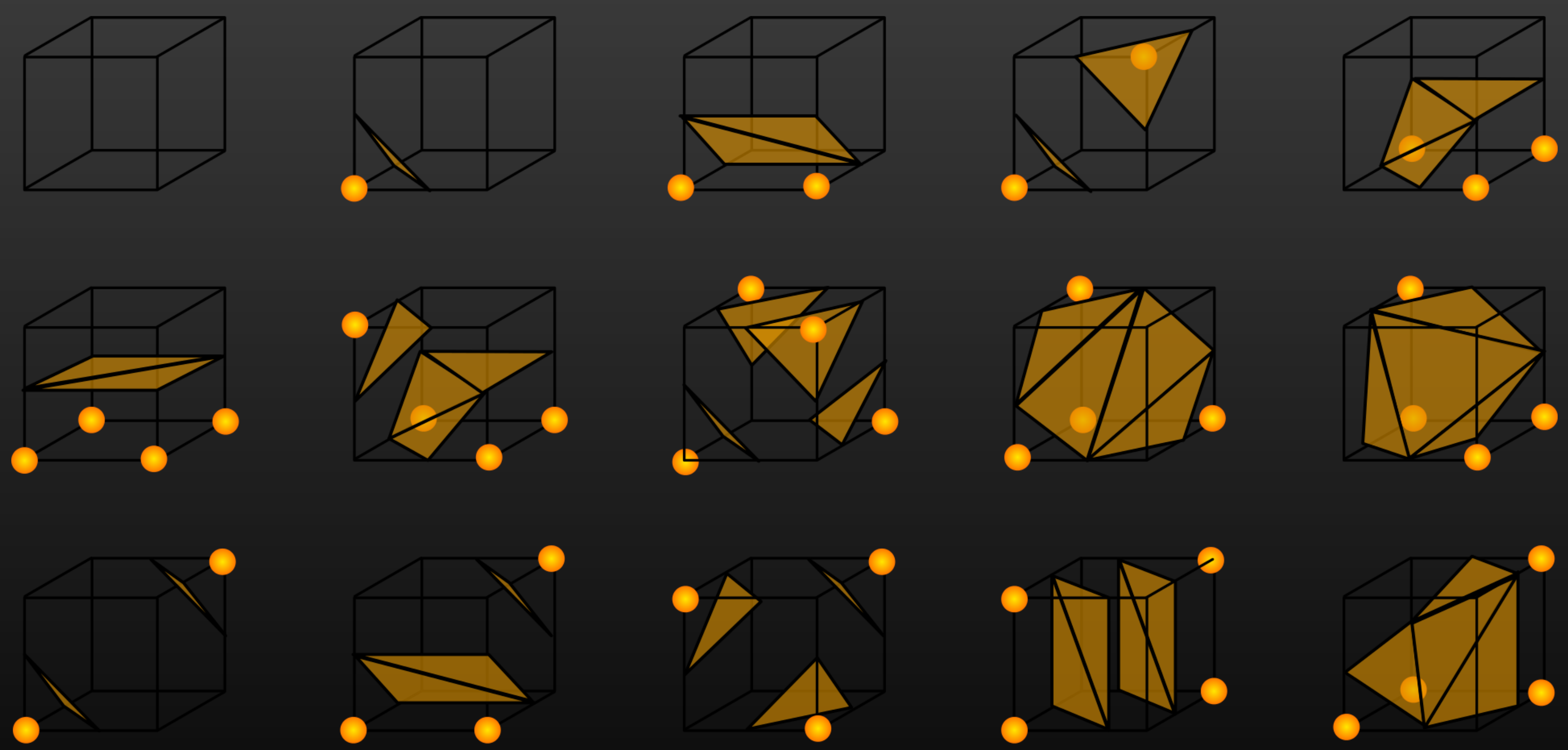
Other GPU optimizations

Rendering

Rendering

- Marching Cubes (1987)
- Screen Space Fluid Rendering (SSFR, 2009)

Marching Cubes

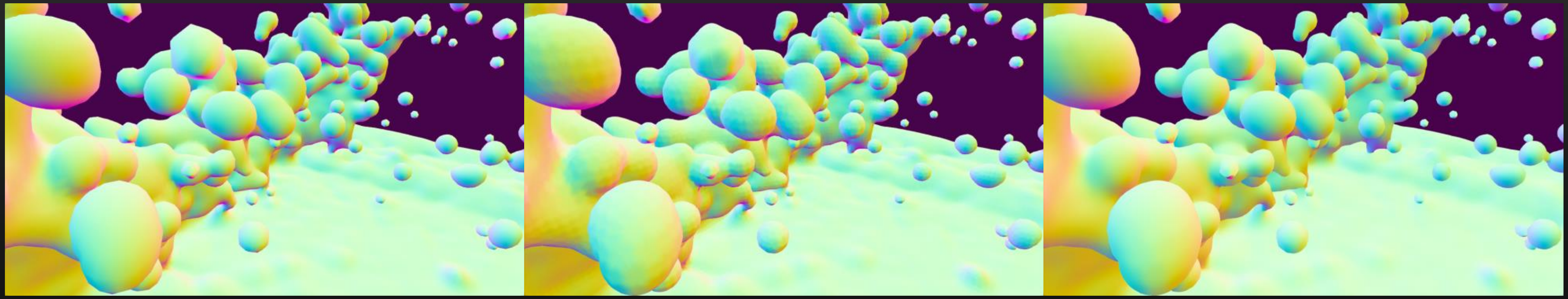


https://graphics.stanford.edu/~mdfisher/MarchingCubes.html

Problems

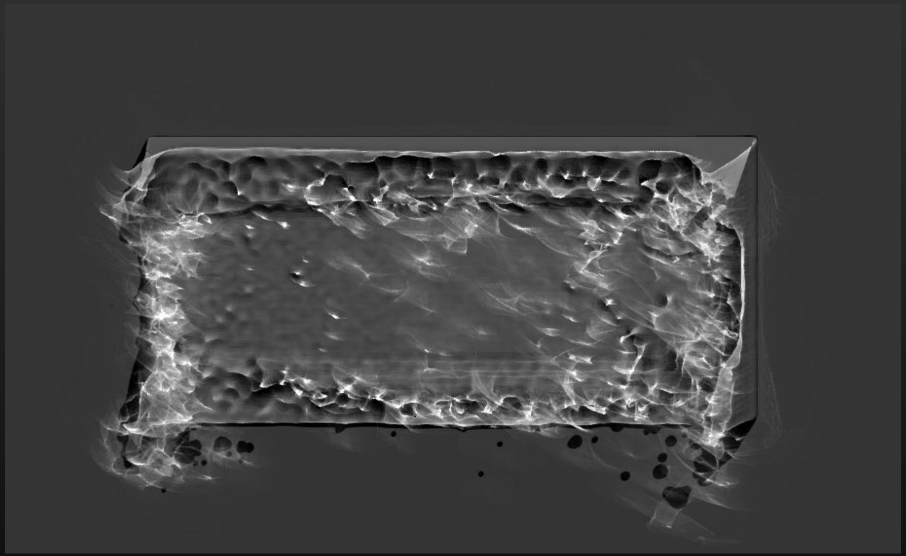
Optimization

- Triangles subdivision: PN triangles
- Normal smoothing: Bilateral filtering



Caustics

Geometry method

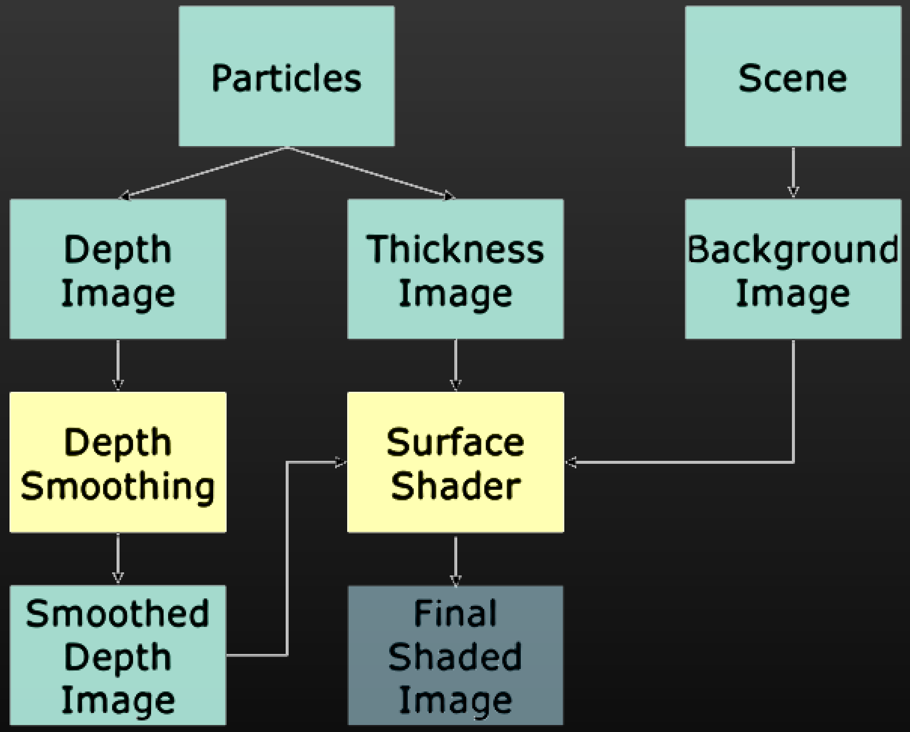


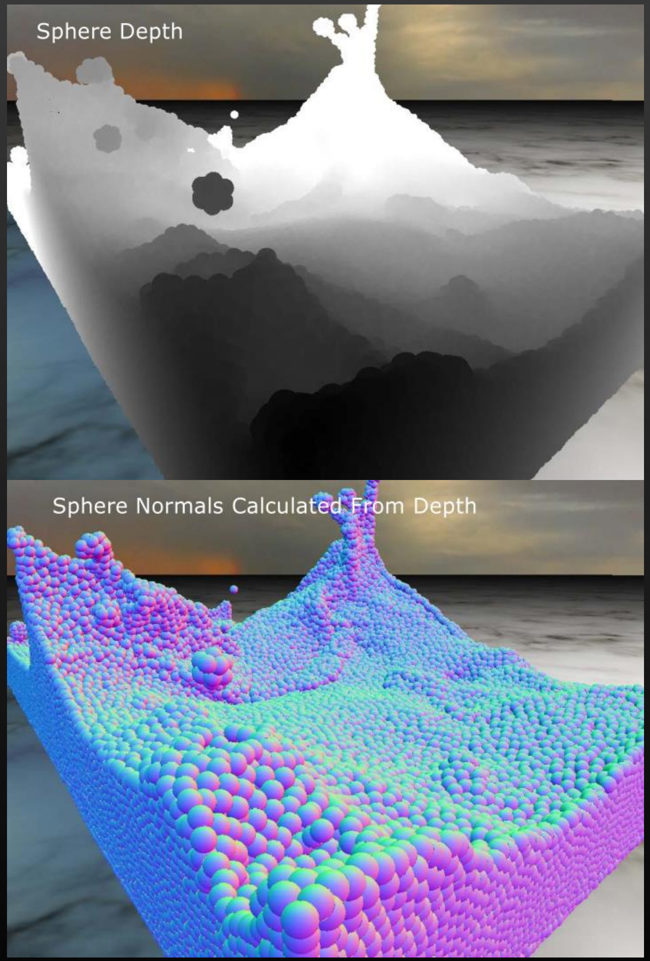
Demo

Video

- MC Final Rendering
- MC Smoothing

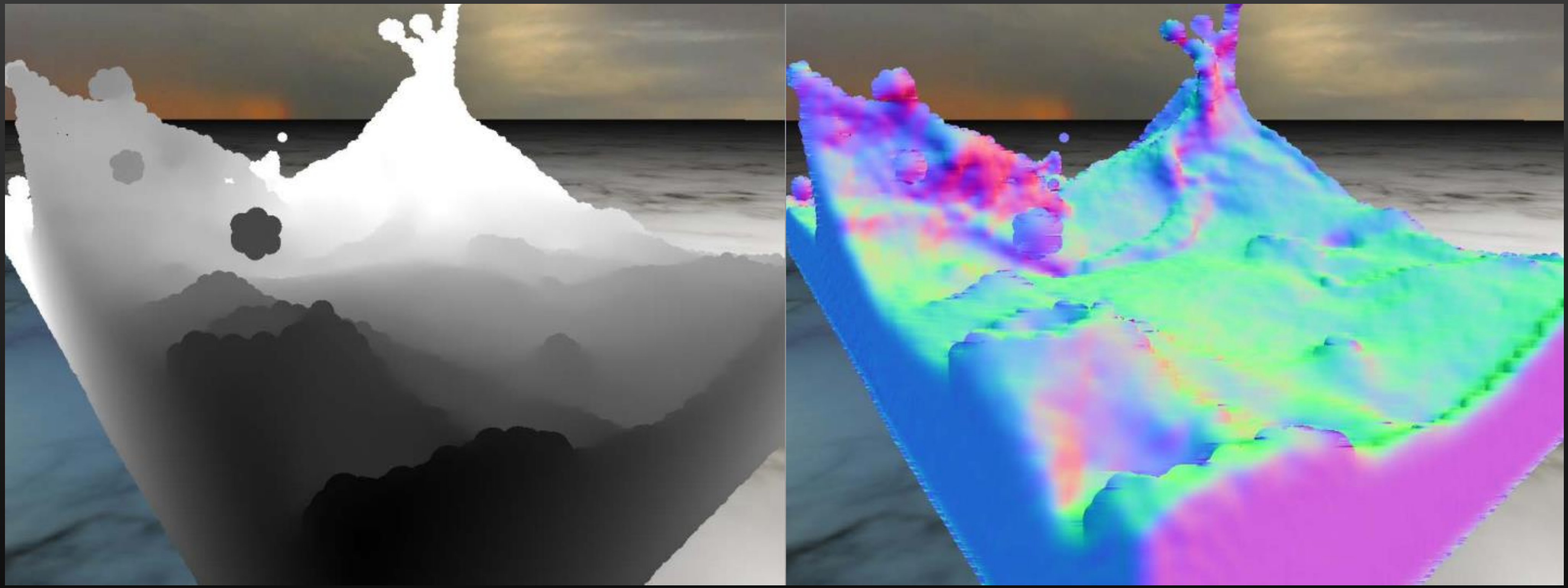
Screen Space Fluid Rendering (SSFR)



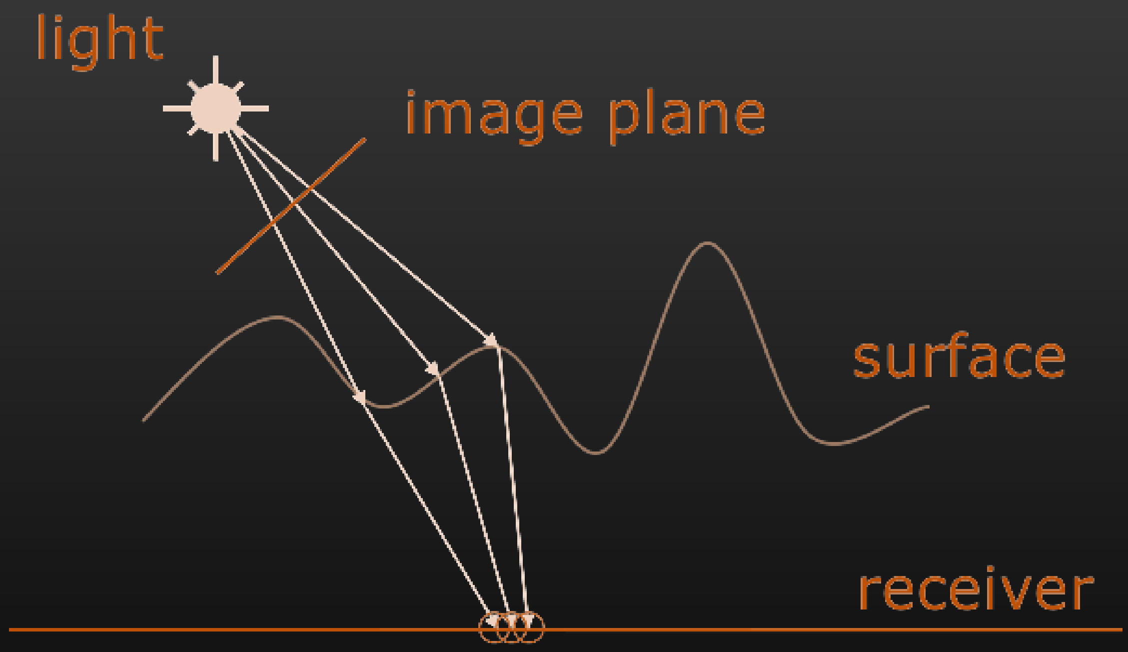


http://developer.download.nvidia.com/presentations/2010/gdc/Direct3D_Effects.pdf

Normal Smoothing



Caustics



Demo

Video

- MC *vs.* SSFR

Comparison

	Performance	Visual Effect	Memory	Parameters
Marching Cubes	Decrease heavily with grid resolution	Depend on grid resolution	Increase with grid resolution	Easy
SSFR	Depend on the number of pixels	Good when viewpoint is far, obvious artifact when view is near	Several Textures	Need tricks

Small scale (<5k / Near) : Marching Cubes

Large scale (>10k / Far) : SSFR

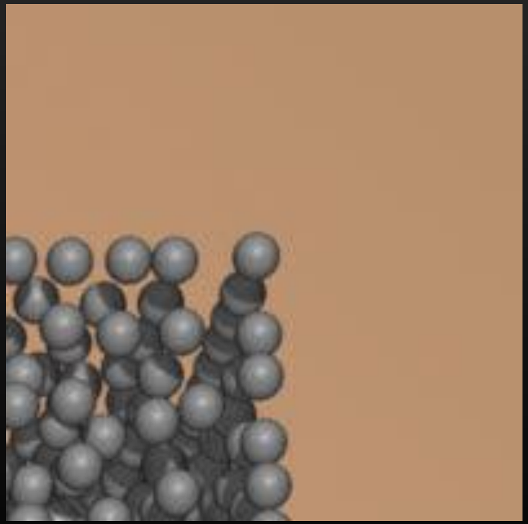
Optimization

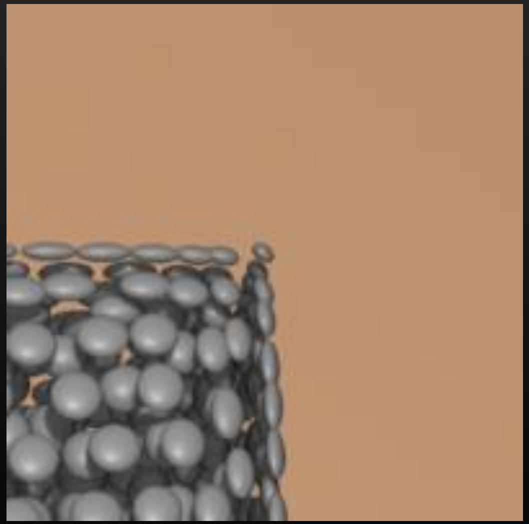
Performance

- Caustics must die
- Bilateral filtering two directions separately (approximation)

Visual Effect

- Blurring Radius view invariant
- Anisotropic particles





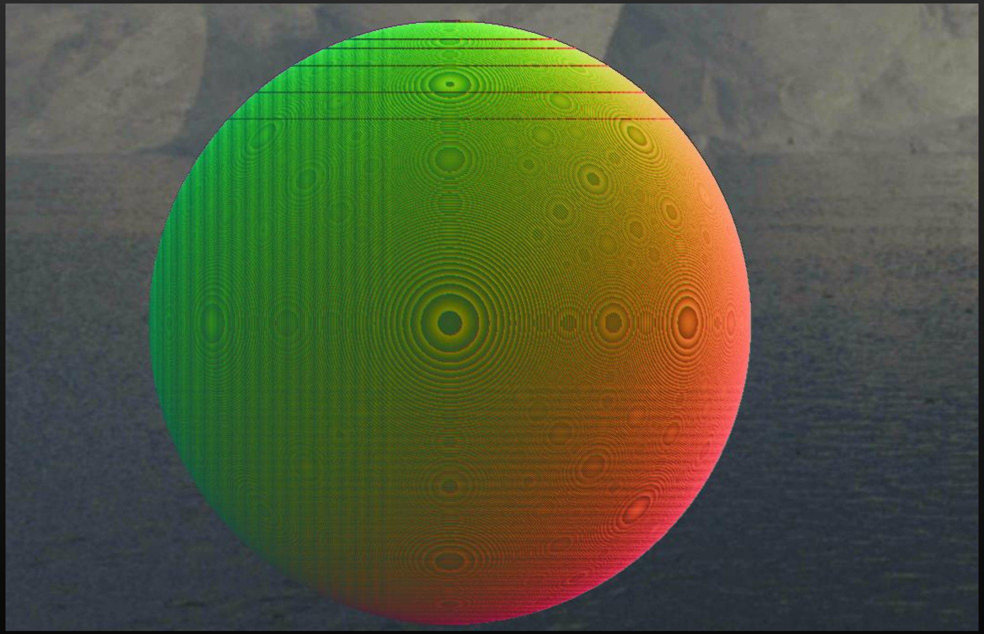
Mobile

Android, Xiaomi6

Qualcomm Adreno 540, Vulkan 1.0.61

Problems

- Geometry shader : Generate triangles in compute shader (Point Sprites)
- Texture formats / MRT supported : Use alternative format
- Precision (restore normal in SSFR) : `highp`



Current Performance

- PBF, 2 iterations, Marching Cubes 80×40×40
- Android, Xiaomi6
- Qualcomm Adreno 540, Vulkan 1.0.61

	1k particles	**5k particles**
With rendering	30fps	25fps
Without rendering	60fps	40fps

Future Optimization

Sorting

Optimization methods on mobile

References

Stam, Jos. "Stable fluids." Acm Transactions on Graphics1999(1999):121--128.

Liu G R , Liu M B . Smoothed Particle Hydrodynamics (SPH): an Overview and Recent Developments[J]. Archives of Computational Methods in Engineering, 2010, 17(1):25-76.

Müller, Matthias, David Charypar, and Markus Gross. "Particle-based fluid simulation for interactive applications." Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation. Eurographics Association, 2003.

Solenthaler, Barbara, and Renato Pajarola. "Predictive-corrective incompressible SPH." ACM transactions on graphics (TOG). Vol. 28. No. 3. ACM, 2009.

Müller, Matthias, et al. "Position based dynamics." Journal of Visual Communication and Image Representation 18.2 (2007): 109-118.
Macklin, Miles, and Matthias Müller. "Position based fluids." ACM Transactions on Graphics (TOG) 32.4 (2013): 104.

Harris, Mark, Shubhabrata Sengupta, and John D. Owens. "Parallel prefix sum (scan) with CUDA." GPU gems 3.39 (2007): 851-876.
van der Laan, Wladimir J., Simon Green, and Miguel Sainz. "Screen space fluid rendering with curvature flow." Proceedings of the 2009 symposium on Interactive 3D graphics and games. ACM, 2009.

Yu, Jihun, and Greg Turk. "Reconstructing surfaces of particle-based fluids using anisotropic kernels." ACM Transactions on Graphics (TOG) 32.1 (2013): 5.

Green, Simon. "Cuda particles." NVIDIA whitepaper 2.3.2 (2008): 1.

Hoetzlein, R. C. "Fast fixed-radius nearest neighbors: interactive million-particle fluids." GPU Technology Conference. Vol. 18. 2014.

软件学院. 不可压平滑粒子流体动力学算法GPU并行加速及其应用研究[J]. 清华大学, 2013.