PARTICLE SIMULATIONS ON THE GPU

Summary by Øystein Krog based on presented articles for TDT 24 Fall 2009

Instructor: Anne C. Elster

"Particle-Based Fluid Simulation for Interactive Applications", Matthias Muller, David Charypar and Markus Gross, Eurogrpahics/Siggraph 2003, D. Breeen, M. Lin Editors

"Smoothed Particle Hydrodynamics on GPU", Takahiro Harada et. al,

PARTICLE SIMULATIONS ON THE GPU

- Why are particle simulations so interesting for the GPU?
  - Relatively easy to parallelize
  - BUT, not trivial because (for our needs) we need to handle particle interactions

Some interesting implementations that use particle simulations:

  - HOOMD
  - Highly Optimized Object-oriented Molecular Dynamics
  - Uses CUDA
  - Open Source
  - Fast, one GPU can compete with small clusters (~36 cores)

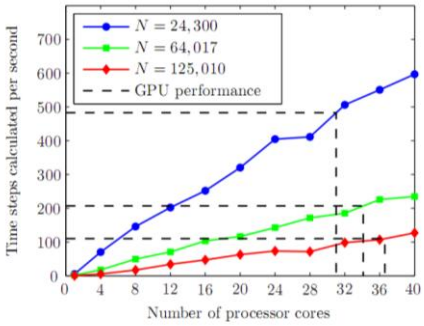


Fig. 8. Benchmark of LAMMPS simulating the Lennard-Jones liquid model for various system sizes on Lightning. The dotted lines mark the performance of the GPU running the same simulation.

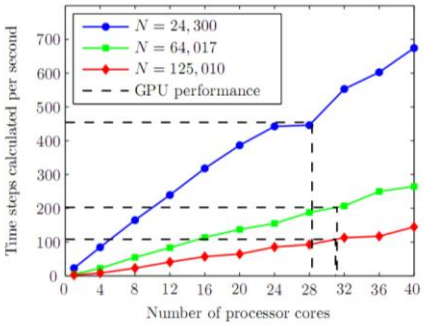


Fig. 9. Benchmark of LAMMPS simulating the polymer model for various system sizes on Lightning. The dotted lines mark the performance of the GPU running the same simulation.

- PhysX
  - NVidia
  - Real-time physics engine
  - Particles used in particular in the fluid simulation
  - Uses SPH – Smoothed Particle Hydrodynamics
  - Why SPH?
    - Need to handle collisions/interaction with rigid objects
    - Conserves volume
    - Support for large volumes (i.e. sparse fluid in large environment)
    - Support multiple independent fluids per scene
    - Fast and efficient

SMOOTHED PARTICLE HYDRODYNAMICS

SOURCES:

- Particle-based fluid simulation for interactive applications - Matthias Müller
- Smoothed Particle Hydrodynamics on GPUs - Takahiro Harada
- Particle based Fluid Simulation - Simon Green/NVidia (slides from NVidia presentation at GDC08)

WHAT IS SPH?

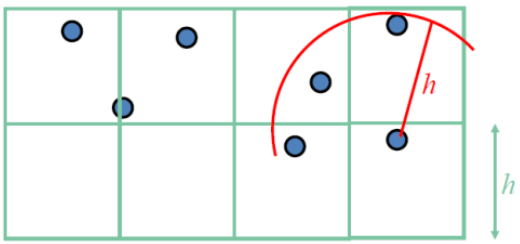
- Particle system with inter-particle forces/interactions
  - Interactions gives O(n^2) computational complexity…
  - Define interaction cutoff distance
    - Less complexity
    - Easier to parallelize
- First developed for use in astrophysics
  - Simulation of stars / galaxies
- Different from a lot of other fluid simulation methods (CFD methods)
  - Mesh-free (does not consider any "surfaces")
  - Interpolation method for particle systems
- Each particle is affected only by the particles in a specific radius around it (cutoff distance)
  - Uses a "radial symmetrical smoothing kernel" to define the cutoff

WHY USE SPH?

- Advantages:
  - Very fast/efficient compared to other fluid simulation methods.
  - Guarantees conservation of mass (particles themselves are the mass) (if weighted functions are normalized)
  - Computes pressure from weighted contributions of neighboring particles (instead of solving linear systems of equations)
- Disadvantages
  - Still have to reconstruct a surface to render (marching cubes, splatting etc)
  - Need a lot of particles for realistic simulation

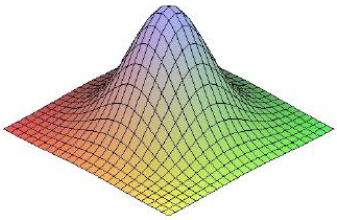
HOW DO WE EFFICIENTLY PARALLELIZE THIS?

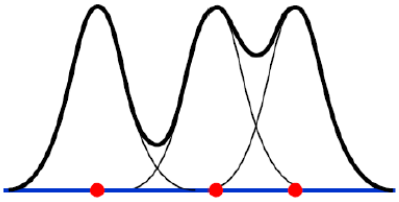
- Use a spatial subdivision data structure
  - **A uniform grid**
  - Fill particles into grid with spacing h
  - Only search potential neighbors in adjacent cells.
  - More on this later…



HOW DO THE PARTICLES INTERACT?

- We have local smoothing fields that sum together into a global field





- Kernel function
  - Specifies how we smooth "between" particles
  - Very important for behavior and stability

THE EQUATIONS OF SPH

- Equations from Mueller unless otherwise noted.
- The particle at location r is calculated as a weighted sum of neighboring particles
  $(m_j, r_j, \rho_j, A_j$ are mass, position, density and field quantity of particle j)

$$(1) \quad A_s(r) = \sum_j m_j \frac{A_j}{\rho_j} W(r - r_j, h)$$

(h is radius of support)

- In the discretized functions, W(r, h) is the smoothing kernel.
- If the kernel is
  - Even
    - W(r, h) = W(-r,h))
  - and Normalized
    - $(2) \int W(r)dr = 1$
  - the kernel is of second order accuracy

- The governing equations (mass and momentum conservation) are simplified Navier-Stokes

$$(6) \quad \frac{\partial \rho}{\partial t} + \nabla \cdot (pv) = 0$$

(conservation of mass)

$$(7) \quad \rho\left(\frac{\partial v}{\partial t} + v \cdot \nabla v\right) = -\nabla P + \rho g + \mu \nabla^2 v$$

(conservation of momentum)

- ρ - density
- v - velocity
- P - pressure
- μ - dynamic viscosity coefficient
- g - external force density field / gravitational acceleration

- Can simplify even more
  - Constant number of particles w/constant mass per particle
    - mass conservation
    - so can omit (6) completely
  - Can simplify (7) because "since the particles move with the fluid the substantial derivative of the velocity field is simply the time derivate of the velocity of the particles meaning that the convective term $v \cdot \nabla v$ is not needed for particle systems
    - $(7') \quad \rho\left(\frac{Dv}{Dt}\right) = -\nabla P + \rho g + \mu \nabla^2 v$
- We have three force density fields on the right hand side of (7)
  - $f = -\nabla P + \rho g + \mu \nabla^2 v$
  - Pressure, density, viscosity
  - f is the "change of momentum $\rho \frac{Dv}{Dt}$ on the left hand side"
- End up with an expression for the acceleration of particle i:
  - $a_i = \frac{dv_i}{dt} = \frac{f_i}{\rho_i}$
    v_i, f_i, ρ_i are velocity, force density field and the density field at the location of particle i

DISCRETIZATION

- The density of fluid

$$(3) \quad \rho_s = \sum_j m_j W(r - r_j, h)$$

- The pressure of fluid (modified version for symmetry)

$$(10) \quad f_i^{pressure} = -\sum_j m_j \frac{p_i + p_j}{\rho_j} \nabla W(r - r_j, h)$$

$$(12) \quad p = k(\rho - \rho_0)$$

where ρ_0, k are rest density and the gas constant

- The viscosity of fluid (modified version for symmetry)

$$(13) \quad f_i^{viscosity} = \mu \sum_j m_j \frac{v_j + v_i}{\rho_j} \nabla^2 W(r - r_j, h)$$

  - (possible interpretation; "look at the neighbors of particle i from i's own moving frame of reference. Then particle i is accelerated in the direction of the relative speed of its environment.")

- Also applies tech. for surface tension
- External forces are simply applied directly to the particles without any use of SPH

SMOOTHING KERNELS

- Stability, accuracy and speed is highly dependent on the choice of smoothing kernels
- Several considerations, specific smoothing kernels for difference force fields etc.
- Example kernel from Mueller:

$$(20) \quad W_{poly6}(\boldsymbol{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3, & 0 \le r \le h \\ 0, & otherwise \end{cases}$$

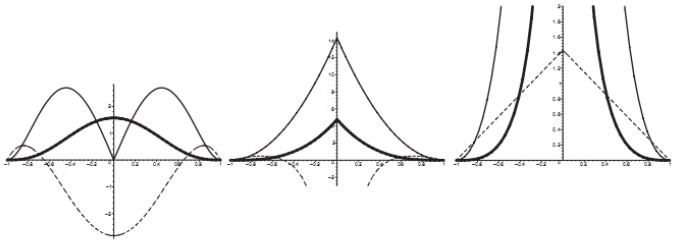


Figure 2: *The three smoothing kernels W_{poly6}, W_{spiky} and $W_{viscosity}$ (from left to right) we use in our simulations. The thick lines show the kernels, the thin lines their gradients in the direction towards the center and the dashed lines the Laplacian. Note that the diagrams are differently scaled. The curves show 3-d kernels along one axis through the center for smoothing length $h = 1$.*

BOUNDARY CONDITIONS (COLLISIONS, E.G. WALLS)

- Mueller do not use specific boundary conditions,
  - Simply affects the velocity/position of the particles directly.
  - Particle about to touch wall ==> deflect velocity
- Harada use more advanced techniques:
  - Pressure - Apply "pressure force" from the boundary:
    - We have constant pressure
    - d is the distance between particles
    - If we collide it will be with a distance less than d, $|r_{iw}| < d$
    - "Push" the particle back to the distance d in the normal direction of the "wall"

$$(11)[harada] \quad F_i^{press} = m_i \frac{(d - |\boldsymbol{r}_{iw}|)\boldsymbol{n}(r_i)}{dt^2}$$

  - Density
    - If particles are within a given range of a boundary, the boundary affects density

$$(13)[harada \quad \rho_i(\boldsymbol{r}_i) = \sum_{j \in fluid} m_j W(\boldsymbol{r}_{ij}) + Z_{wall}^{rho}(|\boldsymbol{r}_{iw}|)$$

where Z_{wall}^{rho} is the wall wight function", can be precomputed *as a lookup table (only depends on distance to particle)*

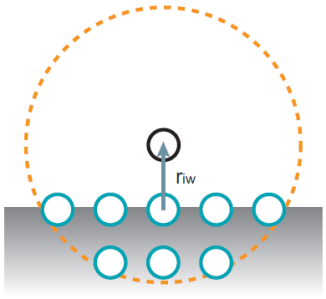


Fig. 1 Distribution of wall particles.

INTEGRATION

- Harada
  - "simple“ Euler
  - Saw no instability..

- Mueller
    - "Leap-Frog", a second order scheme.

SPH ON THE GPU

GPU CONSIDERATIONS

- Each particle can be given its own thread
- No need for synchronization or communication between threads
- A particle has many neighbors – high arithmetic intensity
- Particles have variable number of neighbors – however overall there is a coherence in the workload
-

PARALLELIZATION / THE NEIGHBORING PARTICLE PROBLEM

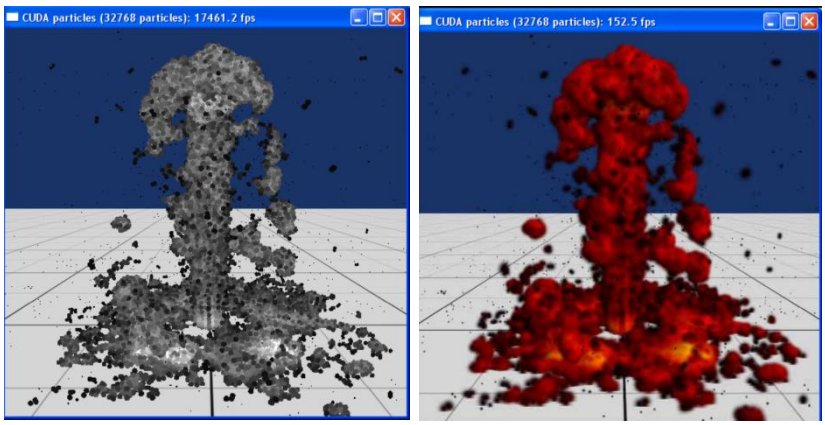
- We need to know how to efficiently and quickly find the neighboring particles in a given radius.
- A naïve exhaustive search requires n^2 comparisons.
- Solution:
    - Use a spatial subdivision (data) structure
- Considerations:
    - The data structure should enable sparse fluids in large environments
        - Can't simply do a 64x64x64 volume because that's too small
        - Can't just use a huge volume because that requires too much memory…
- Options
    - KD-tree / Octree – NOT good because requires recursion (bad for GPU)
    - Uniform grid (used by Mueller)
        - Simplest possible subdivision
        - Since the SPH smoothing kernels have finite support h (cutoff "range") a common way to reduce computational complexity is to use grid cells of size h.
        - Reduces from O(n^2) to O(nm), where n is cells and m is average number of particles per grid cell.
        - A particle is only compared against particles in neighboring cells
        - Large volumes require a lot of memory
    - Hashed uniform grid (used by PhysX SPH)
        - Concrete grid (64x64x64)
        - Abstract grid maps to the concrete grid (x,y,z) ➔ (x%64,y%64,z%64)
        - Fast neighbor finding (if we assume no collisions..)
        - Problem; false neighbors (a "neighbor" will actually be far away)
            - For SPH we ignore this because the SPH smoothing function means that "far" away particles will be ignored
    - Buckets (used by Harada)
        - Conceptually the same as a uniform grid
        - Implemented as a 3D texture where each voxel can have 4 particles (RGBA)
        - If more than 4 particles in a voxel ==> ignore... (unlikely to happen though)
        - Neighboring particles are found by looking at adjacent voxels
    - Sliced grid (used by Harada in later paper)
        - Modification that enables even more sparse fluids
        - Dynamically build a grid that only covers where the fluid exists

        - Ignore all the empty space in the volume
    -

THE ALGORITHM

- Build spatial data structure on particles
- For each particle
    1. Find neighbors
    2. Compute density
    3. Compute force and update velocity
    4. (For collisions with solids) Find neighboring triangles, check collision and update position
- On GPU – do everything in parallel

VISUALIZATION

- SPH gives us a particle field, but no surfaces
- How do we visualize this?
- Possiblities:
    - Extract a surface from the particle set
        - 3D isosurface extraction (marching cubes etc)
    - Ray-Marching
        - "Ray tracing" of the particle set
    - Point splatting
    - Image-space tricks (“billboarding”, blur etc)
        - Can use information we have in the particles to get a better effect
        - Example; use density in particles, can apply blur etc
        - 

PERFORMANCE/ RESULTS

Müeller - Particle-Based Fluid Simulation for Interactive Applications (2003):

- On the CPU
- 1.8 GHz P4 w/ Geforce4
- 2200 particles

- 20 fps for a and b, 5 fps for c (marching cubes)

(a) (b) (c)

Figure 3: *A swirl in a glass induced by a rotational force field. Image (a) shows the particles, (b) the surface using point splatting and (c) the iso-surface triangulated via marching cubes.*

Harada:

- On the GPU, but not CUDA, uses textures and shaders.
- Core2 2.93GHz
- 2GB RAM
- 8800GTX
- C++ / opengl with Cg for shaders
- 60 000 particles @ 17fps
- 28x times faster on GPU

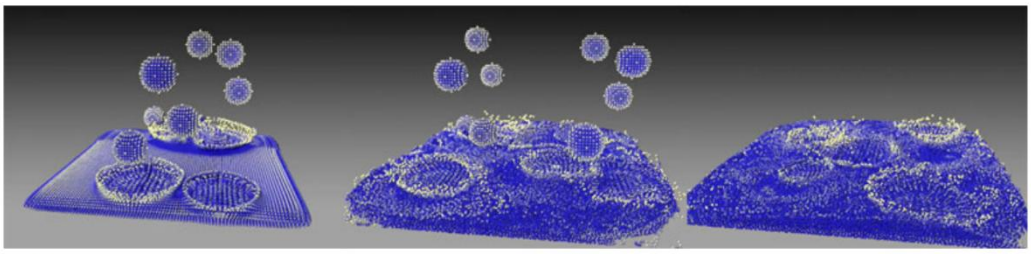


Fig. 4 An example of real-time simulation. Balls of fluid is fallen into a tank.

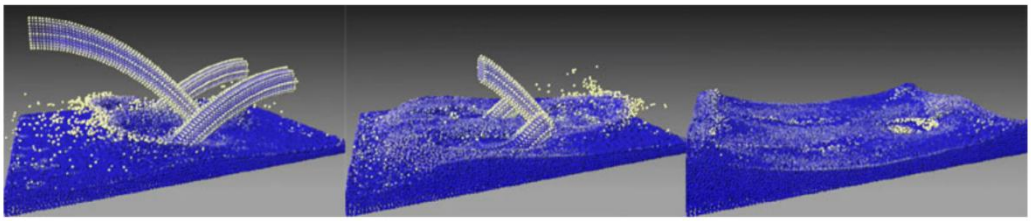


Fig. 5 An example of real-time simulation. A fluid is poured into a tank.

Table 1 Times (a) and (b) are the computation times for bucket generation and computation time for one time step including the rendering time (in milliseconds).

Number of particles	Time (a)	Time (b)
1,024	0.75	3.9
4,096	0.80	5.45
16,386	1.55	14.8
65,536	3.99	58.6
262,144	14.8	235.9
1,048,576	55.4	1096.8
4,194,304	192.9	3783.6

Table 2 Computation time on CPUs (in milliseconds) and speed increase of the proposed method (times faster).

Number of particles	Time	Speed increase
1,024	15.6	4.0
4,096	43.6	8.0
16,386	206.2	13.9
65,536	1018.6	17.3
262,144	6725.6	28.5

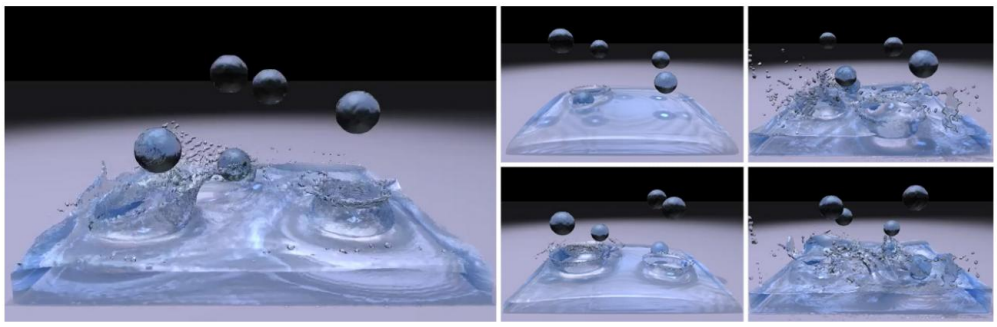


Fig. 7 Balls of fluid is fallen into a tank.

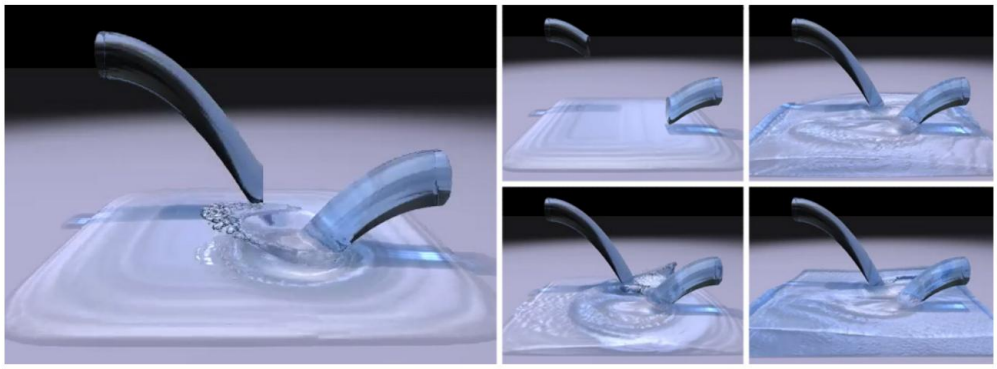


Fig. 8 A fluid is poured into a tank.