Fluid Simulation with Smoothed Particle Hydrodynamics(SPH) method accelerated with Compute Shaders

Final project for the practical course in Computergrafik 2016

Fabian Klemp
FH Aachen

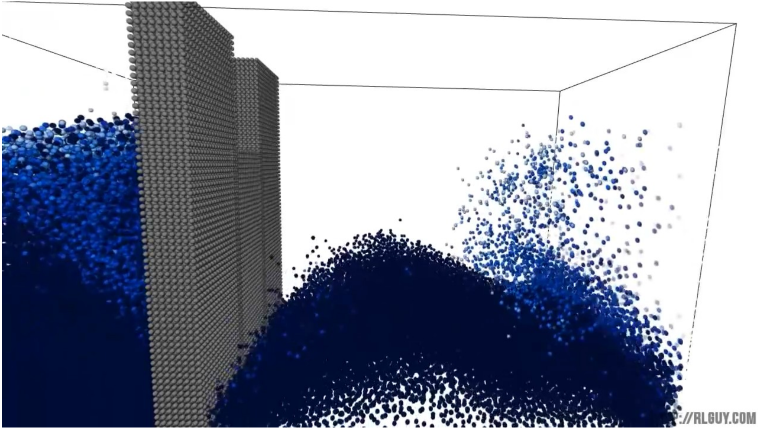
30 June, 2016

Table of Contents

Fluids

- Liquids, e.g. water
- Gasses, e.g. air
- Plasmas

Motivation



source: youtu.be/iHACAIfYeiQ

Navier-Stokes-Equations

Equations which describe the motion of viscous fluids.

We use the Navier-Stokes-Equations for incompressible fluids with constant density:

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

where $\boldsymbol{v}$ is the velocity, $\boldsymbol{g}$ is the gravity, $\boldsymbol{p}$ is the pressure and ρ and μ are the material properties density and dynamic viscosity.

Smoothed Particle Hydrodynamics

Physical property Φ_i at position r_i is computed in a sphere with radius h:

$$\Phi_i = \sum_j m_j W(h, \boldsymbol{r_i} - \boldsymbol{r_j})$$

W is the weighting function which sums to 1 over radius h and drops to 0 outside of h.

Smoothed Particle Hydrodynamics

$$\rho_i \approx \sum_j m_j \frac{315}{64\pi h^9}\left(h^2 - \|\boldsymbol{r_i} - \boldsymbol{r_j}\|^2\right)^3$$

$$p_i = \rho_i - \rho_0$$

$$\frac{\nabla \boldsymbol{p_i}}{\rho_i} \approx \sum_j m_j \left(\frac{\boldsymbol{p_i}}{\rho_i^2} + \frac{\boldsymbol{p_j}}{\rho_j^2}\right) \frac{-45}{\pi h^6}\left(h - \|\boldsymbol{r_i} - \boldsymbol{r_j}\|\right) \frac{\boldsymbol{r_i} - \boldsymbol{r_j}}{\|\boldsymbol{r_i} - \boldsymbol{r_j}\|}$$

$$\frac{\mu}{\rho_i}\nabla^2 \boldsymbol{v_i} \approx \frac{\mu}{\rho_i} \sum_j m_j \left(\frac{\boldsymbol{v_j} - \boldsymbol{v_i}}{\rho_j}\right) \frac{45}{\pi h^6}\left(h - \|\boldsymbol{r_i} - \boldsymbol{r_j}\|\right)$$

$$\frac{dv_i}{dt} = \boldsymbol{g} - \frac{\nabla \boldsymbol{p_i}}{\rho_i} + \frac{\mu}{\rho_i}\nabla^2 \boldsymbol{v_i}$$

Compute Shader

- Introduced with OpenGL 4.3
- Written in GLSL
- Can directly interface with other OpenGL buffers contrary to OpenCL, CUDA, etc.
- Run asynchronous per default
- Meant for simpler computational tasks

Compute Shader Example

```
#version 430

layout(local_size_x = 32, local_size_y = 1,
        local_size_z = 1) in;

layout(std430, binding = 0) buffer Elements {
    vec4 eles[];
} elements;

uniform int work_items;
uniform float time;

void main() {
    uint id = gl_GlobalInvocationID.x;
    if ( id <= work_items) {
        elements.eles[id] =
            vec4(id*0.5, sin(time+((3.14/2)*id)), 0.0, 0.0)↩
                ;
    }
}
```

Compute Shader Example

Use:

```
glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 0, buffer_id);
glDispatchCompute(1, 1, 1);
```

Concrete Implementation

- Sort particles in voxel of length h
- Limit interaction with neighbour particles
  - Only interact with n particles
  - Only interact with particles within the interaction radius h
  - Do this while preventing bias

Concrete Implementation

Compute Shaders:

- voxelize

  calculate voxel id for each particle

Concrete Implementation

Compute Shaders:

- voxelize

  calculate voxel id for each particle
- sort

  sort by voxel id

Concrete Implementation

Compute Shaders:

- voxelize
  calculate voxel id for each particle
- sort
  sort by voxel id
- sortPostPass
  rewrite position and velocity data

Concrete Implementation

Compute Shaders:

- voxelize

  calculate voxel id for each particle
- sort

  sort by voxel id
- sortPostPass

  rewrite position and velocity data
- voxelIndex

  calculate index from voxels to particles

Concrete Implementation

Compute Shaders:

- voxelize

  calculate voxel id for each particle
- sort

  sort by voxel id
- sortPostPass

  rewrite position and velocity data
- voxelIndex

  calculate index from voxels to particles
- findNeighbours

  find the next n neighbours in neighbouring voxels

Concrete Implementation

Compute Shaders:

- voxelize

  calculate voxel id for each particle
- sort

  sort by voxel id
- sortPostPass

  rewrite position and velocity data
- voxelIndex

  calculate index from voxels to particles
- findNeighbours

  find the next n neighbours in neighbouring voxels
- computeDensityPressure

  compute density and pressure values for each particles

Concrete Implementation

Compute Shaders:

- voxelize
  calculate voxel id for each particle
- sort
  sort by voxel id
- sortPostPass
  rewrite position and velocity data
- voxelIndex
  calculate index from voxels to particles
- findNeighbours
  find the next n neighbours in neighbouring voxels
- computeDensityPressure
  compute density and pressure values for each particles
- integrate
  compute pressure gradient, viscosity term, acceleration, integrate velocity and position

Concrete Implementation

A few words about findNeighbours:

- Search in each of the neighbouring voxels until *n* neighbours within the interaction radius are found
- Compute random offset into voxel for each shader invocation then proceed sequentially
- Alternate searching direction by evenness of particle id

Demo

Video/Demo

Main problem

Main problem

I am a computer scientist not a physicist.

Main problem

I am a computer scientist not a physicist.

The algorithms work(AFAIK) but the numerics are shit.

Lessons learned

- Be sure to know your domain. Otherwise get an expert
- Before trying to accelerate something on the GPU implement it on the CPU
  - Debugging is much easier/possible
  - Easier to get working(more built-ins, more libs, etc.)
  - Verification of performance and correctness
- Be sure to do you research properly. Turns out nobody really uses this method anymore

Appendix

Source code:

github.com/Faerbit/sphfluidsim

Source and implementation ideas:

"Smoothed Particle Hydrodynamics" OpenCL Programming
Webinar Series by AMD (November 29, 2010)
Screencast Slides

Thank you for your attention

Any Questions?
Feedback?