Dynamic Particle Coupling for GPU-based Fluid Simulation

Andreas Kolb, Nicolas Cuntz
Computer Graphics Group, University of Siegen, Germany
Email: {andreas.kolb,nicolas.cuntz}@uni-siegen.de

Abstract

The main research approaches in Computational Fluid Dynamics (CFD) are grid-based (Euler) or based on particle motion (Lagrange). For interactive fluid simulation, techniques have been developed to use the Graphics Processing Unit (GPU) to speed up the computation using an Eulerian approach.

This paper describes an approach for setting up a Lagrangian particle based fluid simulation on the GPU. This builds upon earlier results on simulation of uncoupled particles. The major contribution of this work is a new approach for modeling dynamic particle coupling solely based on individual particle contributions. This technique does not need global sorting or an explicite solution of the *n*-nearest neighbor problem.

1 Introduction

The simulation of fluid is important, e.g. in modeling physical systems (climate, ocean), but also in interactive computer graphics applications. Reeves [Ree83] used particle systems in the context of the motion picture Star Trek II. Further applications are the simulation and animation of soft objects [DG96] and the control of implicit surfaces [WH94].

In Computational Fluid Dynamics (CFD) one main research approach is grid-based (Euler). In contrast to the Eulerian approaches, the particle-based approach (Lagrange) makes mass conservation equations and convection terms dispensable. This reduces the complexity of the simulation and the particles can directly be used to render the fluids surface.

Smoothed particle hydrodynamics (SPH) was introduced by Ginhold and Monaghan [GM77]. SPH models the dynamics of fluids based on particle motions applying forces to ensure the Navier-Stokes equations. Müller et.al. [MCG03] present an optimized software implementation allowing interactive simulations for a few thousand particles.

For interactive grid-based fluid simulation, techniques have been developed to use the Graphics Processing Unit (GPU) to speed up the computation [Har03]. The simulation of uncoupled particle motions on GPUs has been introduced recently [KSW04, KLRS04].

The main contribution of this paper is a dynamic particle coupling technique for GPU-based simulation. As proof of concept, this technique has been added to the uncoupled particles in [KLRS04] to realize SPH derived from Müller et.al. [MCG03] on the GPU.

The remainder of the paper discusses related research (Section 2) and describes the new approach to map SPH on the GPU (Section 3). Section 4 gives details on the implementation and Section 5 shows some results.

2 Related Work

This section discusses the basic principles of Smoothed Particle Hydrodynamics (SPH), some important aspects of GPU programming and gives an overview on the uncoupled particle simulation on the GPU.

2.1 Smoothed Particle Hydrodynamics

The main concept of SPH is the usage of *radial symmetric smoothing kernels* to distribute a quantity A_j of a particle j at position $\mathbf{P}_j$ to its neighborhood

$$\bar{A}(\mathbf{P}) = \sum_j m_j \frac{A_j}{\bar{\rho}_j} W(\left|\mathbf{P} - \mathbf{P}_j\right|, h), \tag{1}$$

where m_j is the particle's mass, W is the smoothing kernel with max. radius h and $\bar{\rho}_j = \bar{\rho}(\mathbf{P}_j)$ is the density given by the smoothed particle masses

$$\bar{\rho}(\mathbf{P}) = \sum_j m_j W(\left|\mathbf{P} - \mathbf{P}_j\right|, h). \tag{2}$$

Smoothing kernels $W \in C^2[0,h]$ must be normalized, i.e. $\int W(\mathbf{P})\, d\mathbf{P} = 1$ in order to guarantee mass preservation.
The resulting pressure and viscosity forces for particles i are deduced from the Navier-Stokes equations (more details in Müller et.al. [MCG03]):

$$\begin{aligned} \vec{\mathbf{f}}_p(\mathbf{P}_i) &= -\sum_j m_j \frac{p_i + p_j}{2\bar{\rho}_j} \nabla W(\left|\mathbf{P}_i - \mathbf{P}_j\right|, h), \\ \vec{\mathbf{f}}_v(\mathbf{P}_i) &= \mu \sum_j m_j \frac{\vec{\mathbf{v}}_j - \vec{\mathbf{v}}_i}{\bar{\rho}_j} \nabla^2 W(\left|\mathbf{P}_i - \mathbf{P}_j\right|, h). \end{aligned} \tag{3}$$

Here $p_j = k(\bar{\rho}_j - \rho_0)$ is the pressure with gas constant k and rest density ρ_0 and μ is the fluid viscosity constant.
To model the surface tension, Müller et.al. [MCG03] use the so-called color field $\bar{c}$. The gradient of the color field yields the inward normal field of the fluid, whereas the divergence of the normal field is a measure for the curvature κ of the fluid surface

$$\bar{c}(\mathbf{P}) = \sum_j \frac{m_j}{\bar{\rho}_j} W(\left|\mathbf{P} - \mathbf{P}_j\right|, h), \quad \vec{\mathbf{n}}(\mathbf{P}_i) = \nabla \bar{c}(\mathbf{P}_i), \quad \kappa = -\frac{\nabla^2 \bar{c}}{|\vec{\mathbf{n}}|}. \tag{4}$$

Müller et.al. [MCG03] deduced the surface traction force using the tension coefficient σ

$$\vec{\mathbf{f}}_s(\mathbf{P}) = \begin{cases} -\sigma \nabla^2 \bar{c}(\mathbf{P}) \frac{\vec{\mathbf{n}}(\mathbf{P})}{|\vec{\mathbf{n}}(\mathbf{P})|} & |\vec{\mathbf{n}}(\mathbf{P})| > \varepsilon \\ 0 & \text{otherwise} \end{cases}, \tag{5}$$

with a given threshold $\varepsilon > 0$ to avoid numerical problems in case $|\vec{\mathbf{n}}(\mathbf{P})| \approx 0$.

2.2 GPU Programming Aspects

From an algorithmic point of view, the programmable Graphics Processing Unit (GPU) can be seen as a *fast, parallel 2D array processor*. Its native array format is 2D (*texture*), where each array element has up to 4 components (color components). For processing, the input and output data array are distinct. The programmability of the GPU addresses two different stages in the data processing on the GPU:

Vertex Program: Selects the data elements to be processed in the 2D output array.

Fragment Program: Performs the processing of the selected output data elements.

For more details scientific computation on the GPU see Strzodka et.al. [SDK05] and the *General Purpose Computation Using Graphics Hardware (GPGPU)* website `www.gpgpu.org`.

2.3 Particle Simulation on the GPU

The key idea of performing uncoupled particle simulation on the GPU is to use a fragment program to update 2D arrays storing the state parameters of all particles. The coordinates of the 2D data elements serve as particle id over several 2D arrays carrying the state parameters.

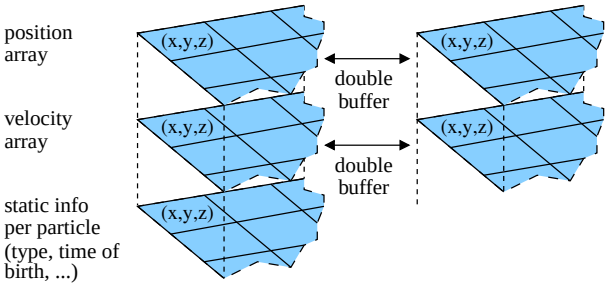

Figure 1: Data storage concept for particle systems.

Because input and output arrays are distinct on GPUs, two arrays for the data are needed to perform two or more consecutive time-steps. A flip-flop algorithm exchanges the role of input and output arrays (Figure 1). In most cases, first order, e.g. Euler integration is used to compute the particle motion. Applying higher precision integration, possibly more than two data arrays have to be used in a ring-buffer like manner.

3 Dynamic Particle Coupling

This section discusses the technique to model the dynamic particle coupling for the flow simulation. Section 3.1 explains the general idea of handling the SPH equations. Section 3.2 discusses aspects of mapping this scheme onto the GPU.

3.1 Separable Summands

The general goal in computing the forces in Eq. (3) and (5) is to avoid the explicite determination of the particles in the neighborhood of a 3D position $\mathbf{P}$ (Eq. (1)).
This is achieved by accumulating the individual contribution of each particle in a 3D array which discretizes 3D-space. From an abstract point of view, a smoothed quantity $\bar{A}$ must be represented as $\bar{A}(\mathbf{P}) = \sum_j g(j,\mathbf{P})$ with an appropriate function g yielding *separable summands*. Here, j gives access to all attributes of particle j. After having handled all particles, the resulting quantity can be retrieved by sampling the 3D array.
The pressure force (3) and the color field (4) are separable, since all necessary quantities for particle j are known or can be sampled in the density field. The viscosity force $\vec{\mathbf{f}}_v$ is only defined at a particle position, since the velocities of particle i and j are involved.
In order to compute the 3D force field based on Eq. (3) and (5), the following steps are performed. First, the density field $\bar{\rho}$ is computed and stored. After computing the color field $\bar{c}$, its gradient $\nabla\bar{c}$ and Laplacian $\nabla^2\bar{c}$ are stored as a second 3D array with four components. Then, the separable part of the viscosity force $\vec{\mathbf{f}}_v^{\text{sep}}$ and the pressure force $\vec{\mathbf{f}}_p$ are computed and stored in another 3D array (*force field*). Finally, during the velocity integration for particle i, the surface force field and the missing part of the viscosity force $\vec{\mathbf{f}}_v^{\text{part}}$ is added to the force sampled in the force field.

$$\vec{\mathbf{f}}_v^{\text{sep}} = \mu \sum_j m_j \frac{\vec{\mathbf{v}}_j}{\bar{\rho}_j} \nabla^2 W(\left|\mathbf{P}_i - \mathbf{P}_j\right|, h), \qquad \vec{\mathbf{f}}_v^{\text{part}} = -\mu \vec{\mathbf{v}}_i \nabla^2 \bar{c}(\mathbf{P}_i)$$

3.2 Computing 3D Quantities on the GPU

The goal is to compute the force fields described in Section 3.1 using the GPU. Since the native output format of the GPU is a 2D array, the 3D discretization has to be formulated as a stack of 2D arrays, called *slices*. The contribution of a single particle will be present in several slices.
In order to compute the 3D quantity, for each slice all particles i that contribute to that slice, are required. Therefore, a "point" with radius h (called *point sprite*) is drawn at the particle position $\mathbf{P}_i$ onto the slice. For each pixel on the point sprite the corresponding 3D position $\mathbf{P}$ and the relevant particle position $\mathbf{P}_i$ are known, so that the contribution of particle i at position $\mathbf{P}$ can be computed. Particle that do not contribute to the current slice are automatically clipped at properly defined clipping planes.

4 Implementation Details

The simulation renderer has been implemented in C++ using OpenGL and Cg for GPU programming, using an NVIDIA GeForce 6800 GT with 256 MB graphics memory.
In order to perform the computation as pointed out above, 2D output arrays of type float with accumulation functionality are needed. Additionally, during rendering of the point-sprites, varying array coordinates are needed in order to properly compute the 3D locations of each rasterized sprite pixel. Currently, only 16 bit `ATI_float`-textures in combination

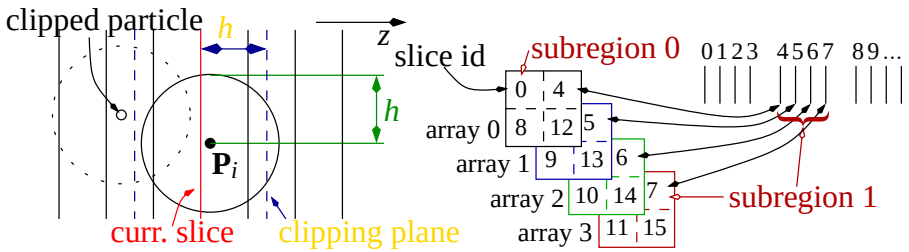

Figure 2: Set up of the clipping planes for a given slice (left) and referencing of 2D array slices in four larger 2D arrays using subregions (right).

with the `point_sprite_NV`-extension [NVI05] fulfill this requirement. This texture format, however, can not use the so-called render-to-texture mechanism, which enforces the use of an inefficient copy functionality to re-use the output arrays as input array later on.
Sampling the 3D quantity stored in a 3D array, i.e. in a 2D array stack, is done with linear interpolation within the slices and between the two adjacent slices w.r.t. the 3D sampling location, yielding a trilinear interpolation scheme.
In order to speed up the computation of the 3D quantities, so-called multiple-render-targets (MRT) are used to compute four slices in parallel. Additionally, subregions are used to represent several slices in a larger 2D array. This minimizes the number of arrays needed to represent a 3D quantity. Since the MRT mechanism allows only computation at the same output position for all four arrays, the slices reference to subregions in the four arrays in an alternating manner. Thus, four adjacent slices can be computed in parallel (Figure 2).

5 Results & Conclusion

We have presented a method to realize a flow simulation based on the SPH approach completely on the GPU. The current system is a prove of concept for the results of

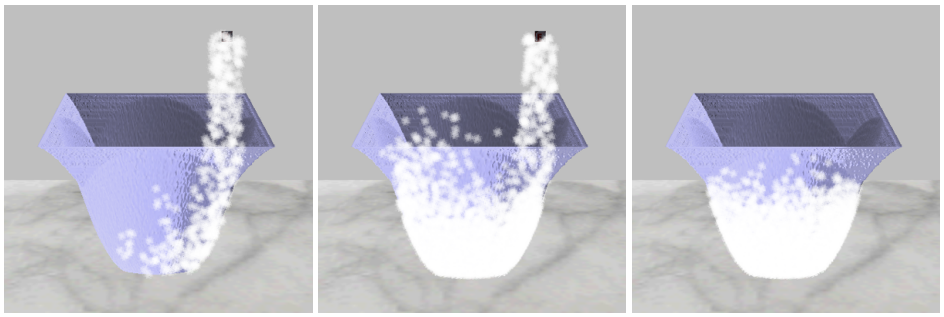

Figure 3: The three images show different states when filling a cup with water. This example runs at 12 FPS.

the presented approach. While there is much room for optimizations, it is already possible to simulate a liquid at a reasonable frame rate. The frame rate strongly depends on several parameters: The number of particles, the resolution of the force field textures, the smoothing kernel radius h and the distribution of the particles in space. The example in Figure 3 runs at 12 FPS, where nearly 2400 particles are simulated in a 32^3 force field using a kernel radius of 20% of the force field dimension (Figure 4). Increasing the resolution to 64^3, yields ≈ 2.3 FPS, whereas decreasing the resolution to 16^3 gives 14.5 FPS. Further optimizations will probably result in a much better performance. Apart from performance issues, the visual output could be improved considerably by integrating a rendering of free fluid surfaces.

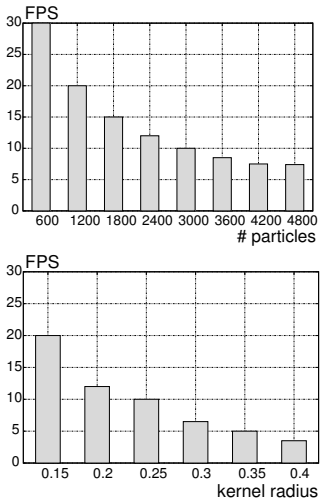

Figure 4: The cup simulation with kernel radius 0.2 and force field resolution 32^3, with varying number of particles (top) and varying kernel radius for 2400 particles (bottom).

References

[DG96] M. Desbrun and M.-P. Gascuel. Smoothed particles : A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation*, pages 61–76, 1996.

[GM77] R.A. Gingold and J.J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Notices of the Royal Astronomical Society*, 181:375–389, 1977.

[Har03] M. Harris. *Real-Time Cloud Simulation and Rendering*. PhD thesis, CS Dep., University of N. Carolina at Chapel Hill, 2003.

[KLRS04] A. Kolb, L. Latta, and C. Rezk-Salama. Hardware-based simulation and collision detection for large particle systems. In *Proc. Graphics Hardware*, pages 123–131, 2004.

[KSW04] P. Kipfer, M. Segal, and R. Westermann. Uberflow: A GPU-based particle engine. In *Proc. Graphics Hardware*, pages 115–122. ACM/Eurographics, 2004.

[MCG03] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Sym. on Comp. Animation*, 2003.

[NVI05] NVIDIA Corporation. OpenGL extension. http://developer.nvidia.com/object/nvidia_opengl_specs.html, 2005.

[Ree83] W. Reeves. Particle systems - technique for modeling a class of fuzzy objects. In *ACM Proceedings SIGGRAPH*, volume 2, pages 91–108, 1983.

[SDK05] R. Strzodka, M. Doggett, and A. Kolb. Scientific computation for simulations on programmable graphics hardware. *Simulation Practice & Theory*, 2005. to appear.

[WH94] A. Witkin and P. Heckbert. Using particles to sample and control implicit surfaces. In *ACM Proceedings SIGGRAPH*, pages 269–277, 1994.