Asychronous Smoothed Particle Hydrodynamics

Andrew Pregent

1 Literature Review

1.1 Smoothed-Particle Hydrodynamics

Smoothed-particle Hydrodynamics (SPH) is a Lagrangian simulation method first proposed independently by L.B. Lucy and R.A. Gingold and J.J. Monaghan for simulations in astrophysics.[13, 5] J.J. Monaghan later extended the method to free surface flows.[14]

Smoothed-particle Hydrodynamics is a Lagrangian method which treats discrete masses of fluid as particles which can move freely throughout the simulation domain. These particles are subject to forces due to pressure and viscosity of the fluid. Forces are calculated for each particle based on its current position and velocity and then new position and velocity are computed by integrating these forces over a given time step.

The SPH update step is presented in Algorithm 1. The computation of acceleration has been split as suggested by Ihmsen et al. so that advection forces are considered when computing pressure forces.[10]

There are two main forces at play in SPH. The first is pressure force $\mathbf{a}_i^{pres}$ which is calculated from density using an Equation of State (EOS). There are many different choices for this, the one in Equation 1 is the same used by Reinhardt et al.[16]

$$p_i = \begin{cases} \kappa(\rho_i - \rho_0) & \text{if } \rho_i > \rho_0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Algorithm 1 Full SPH step using split forces calculation.

for $i \in \{1, 2, ..., n\}$ **do**
    Compute $\mathbf{a}_i^{visc} = \nu \Delta \mathbf{u}_i / \rho_i$
    Compute $\mathbf{a}_i^{ext}$
    $\mathbf{u}_i \leftarrow \mathbf{u}_i + (\mathbf{a}_i^{visc} + \mathbf{a}_i^{ext}) dt$
end for
for $i \in \{1, 2, ..., n\}$ **do**
    Compute ρ_i by Equation 2
    Compute p_i by Equation 1
end for
for $i \in \{1, 2, ..., n\}$ **do**
    Compute $\mathbf{a}_i^{pres} = -\nabla p_i / \rho_i$
end for
for $i \in \{1, 2, ..., n\}$ **do**
    $\mathbf{u}_i \leftarrow \mathbf{u}_i + \mathbf{a}_i^{pres} dt$
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \frac{1}{2}(\mathbf{a}_i^{visc} + \mathbf{a}_i^{ext} + \mathbf{a}_i^{pres}) dt^2 + \mathbf{u}_i dt$
end for

Equation 1 requires that we determine the density of the fluid at each particle's center point $\mathbf{x}_i$. Using the Parzen-Rosenblatt kernel estimation we can define the density field in terms of the particles as point samples as shown in Equation 2. The volume scaling factor V_i is approximated as m_i/ρ_i using the physical quantities of mass m_i and density ρ_i, which allows us to cancel density. We can then use this to estimate the density at each particle, ρ_i, which was previously unknown.[12]

$$\rho(\mathbf{x}) = \sum_i^n \rho_i W(\mathbf{x} - \mathbf{x}_i) V_i = \sum_i^n m_i W(\mathbf{x} - \mathbf{x}_i) \tag{2}$$

To calculate the pressure force we need a discretization of the gradient operator. The one presented here is due to Adams and Wicke.[1] One important property which they note the gradient of pressure should possess is that the resulting pressure force is symmetric between pairs of particles (respecting Newton's third law).[1] Because of this, the derivation of the gradient

operator is somewhat roundabout, requiring some algebraic manipulation to arrive at the form below.

$$\nabla p_i/\rho_i = \sum (pj/\rho_j^2 + p_i/\rho_i^2)(xj - xi)\nabla W(|x_j - x_i|)m_j \tag{3}$$

The second force we must calculate is the viscosity force $\mathbf{a}_i^{visc}$. Viscosity is modelled as the Laplacian of the velocity field. As with the gradient, we also insist in a discretization of the Laplacian which results in symmetric viscosity forces.[1]

$$\nabla \mathbf{u}_i = \sum (\mathbf{u}_j - \mathbf{u}_i)\nabla W(\mathbf{x}_j - \mathbf{x}_i)m_j/\rho_j \tag{4}$$

Until now we have avoided discussing the kernel W. The kernel should obey three properties: first is that the area under its curve must add to one; second, the kernel should vanish after h units (so that $W(h) = 0$), where h is called the support distance; and third, the kernel must be differentiable everywhere on its domain.[12] Many different kernels have been suggested, and we must be careful to use the kernel for the appropriate dimension. Using a kernel designed for $\mathbb{R}^2$ will not have an integral of 1 in $\mathbb{R}^3$ (or vice versa). The kernels presented in Equations 5 and 6 are due to M. Müller.[15]

$$W(\mathbf{r}) = \begin{cases} 315\frac{(1-|\mathbf{r}|/h)^3}{64\pi h^3} & \text{if } |\mathbf{r}| < h \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$\nabla W(\mathbf{r}) = \begin{cases} -45\mathbf{r}\frac{(1-|\mathbf{r}|/h)^2}{\pi * h^4} & \text{if } |\mathbf{r}| < h \\ \mathbf{0} & \text{otherwise} \end{cases} \tag{6}$$

1.2 Neighborhood Search

Every particle in the simulation enacts a force on and likewise has a force enacted on it by every other particle. This means there are $O(n^2)$ interactions

between particles in the system, which is obviously intractable when dealing with millions of particles. Thankfully, the magnitude of these forces vanishes as distance increases, and can be safely ignored after a certain threshold. If we assume an upper limit on the number of possible neighbors a given particle will interact with, we can even reduce the simulation from quadratic to linear.

Tree structures are ill suited to the streaming memory models of the GPU and were avoided when optimizing the neighborhood search. A simple alternative which is better suited to the GPU is the cell list. In a cell list, the simulation domain is split into a regular grid of cubes known as cells. Each cell contains an index to an arbitrary particle contained by it. Each particle has associated with it an index to the next particle in the cell. A unique index value is reserved for denoting that a cell is empty, and that the final particle in a cell list has been reached.[12]

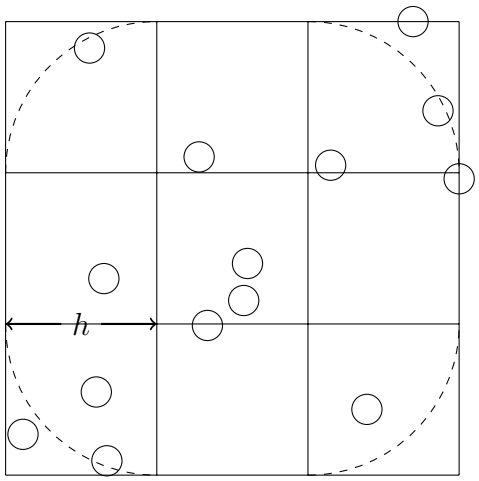

Figure 1: Neighborhood of a cell. Particles inside dashed region are possibly within h of a point in the center cell.

Now when we wish to list neighbors of a given particle we can look up the particle's grid index, and then iterate the particles that share that cell, and those neighboring cells within the given distance threshold. Commonly the cell size is chosen to be the same as the distance threshold, which is also usually the support size of our kernel W, so that we only need to search a $3 \times 3 \times 3$ cube of cells. This is the approach also taken in this implementation.

1.3 GPU Acceleration

The work of Ihmsen et al. provided much of the groundwork with an early parallel implementation.[10] For this search they used a Z-index sort, a space filling curve which provides a cache-friendly ordering for the particles. Amada et al. present a partial GPU implementation which relies on the CPU for the neighborhood search, providing the information to the GPU as a texture.[2] Harada et al. present an early fully GPU implementation[7]. Later Hérault make use of the programmable pipeline to create a CUDA implementation[9], which they later released as open source[8]. Finally, Rustico et al. extend this to multiple GPUs.[17]

1.4 Time-step

M. Desbrun and M. Gascuel applied the Courant-Friedrichs-Lewy criterion to SPH, providing an upper bound on the time step based on the kernel support size and the maximum particle velocity[4]. They suggest a using an adaptive time-step based on this condition.

Another approach is to avoid a global time-step altogether. P. Goswami and C. Batty propose segmenting the time-step by spatial chunks.[6]. Asychronous SPH allows every particle to have its own time frame[16][3][4]. This is more efficient when there are only a few fast particles, as is often the case.

The method of asynchronous SPH suggested by Reinhardt et al. integrate each particle forward in time individually.[16] When describing the method it will be useful to speak in terms of a particle's age, which is the total time which it has been integrated during the simulation. All particles in the simulation start at the same age, and ages of particles are tracked individually. A queue was used to sort the particles by their age, so that the youngest particle was always the next one to be stepped forward in time. This ensures that the neighborhood around the particle would all be older, and therefore only requiring that we backtrack particles - keeping the simulation stable.

To make the algorithm multithreaded, Reinhardt et al. suggest dividing the particles into multiple queues, a method due to Kale and Lew.[16, 11] However, now the particle being integrated might not be the youngest particle in its neighborhood, since other threads may be responsible for particles in the particles' neighborhood. In order to solve this, Reinhardt et al. suggest testing for this before integrating. If a younger particle is found, we

place the particle in a second list of pending particles. This list is routinely added back to the queue, in the hope that on dealing with a particle the second time another thread will have taken care of the younger particles.

References

[1] Bart Adams and Martin Wicke. "Meshless Approximation Methods and Applications in Physics Based Modeling and Animation." In: *Eurographics (Tutorials)*. 2009, pp. 213–239.

[2] Takashi Amada et al. "Particle-based fluid simulation on GPU". In: *ACM workshop on general-purpose computing on graphics processors*. Vol. 41. Citeseer. 2004, p. 42.

[3] Xiaojuan Ban et al. "Adaptively stepped SPH for fluid animation based on asynchronous time integration". In: *Neural Computing and Applications* 29.1 (2018), pp. 33–42.

[4] Mathieu Desbrun and Marie-Paule Gascuel. "Smoothed particles: A new paradigm for animating highly deformable bodies". In: *Computer Animation and Simulation'96*. Springer, 1996, pp. 61–76.

[5] Robert A Gingold and Joseph J Monaghan. "Smoothed particle hydrodynamics: theory and application to non-spherical stars". In: *Monthly notices of the royal astronomical society* 181.3 (1977), pp. 375–389.

[6] Prashant Goswami and Christopher Batty. "Regional time stepping for SPH". In: *Eurographics 2014*. Eurographics Association. 2014, pp. 45–48.

[7] Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi. "Smoothed particle hydrodynamics on GPUs". In: *Computer Graphics International*. Vol. 40. SBC Petropolis. 2007, pp. 63–70.

[8] A. Hérault et al. *GPU-SPH*. http://www.ce.jhu.edu/dalrymple/GPU/GPUSPH/Home.html.

[9] Alexis Hérault, Giuseppe Bilotta, and Robert A Dalrymple. "Sph on gpu with cuda". In: *Journal of Hydraulic Research* 48.sup1 (2010), pp. 74–79.

[10] Markus Ihmsen et al. "A parallel SPH implementation on multi-core CPUs". In: *Computer Graphics Forum*. Vol. 30. 1. Wiley Online Library. 2011, pp. 99–112.

[11] Kedar G Kale and Adrian J Lew. "Parallel asynchronous variational integrators". In: *International Journal for Numerical Methods in Engineering* 70.3 (2007), pp. 291–321.

[12] Doyub Kim. *Fluid engine development*. CRC Press, 2017.

[13] Leon B Lucy. "A numerical approach to the testing of the fission hypothesis". In: *The astronomical journal* 82 (1977), pp. 1013–1024.

[14] Joe J Monaghan. "Simulating free surface flows with SPH". In: *Journal of computational physics* 110.2 (1994), pp. 399–406.

[15] Matthias Müller, David Charypar, and Markus Gross. "Particle-based fluid simulation for interactive applications". In: *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 2003, pp. 154–159.

[16] Stefan Reinhardt et al. "Fully asynchronous SPH simulation". In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2017, pp. 1–10.

[17] Eugenio Rustico et al. "A journey from single-GPU to optimized multi-GPU SPH with CUDA". In: *7th SPHERIC Workshop*. 2012, p. 56.