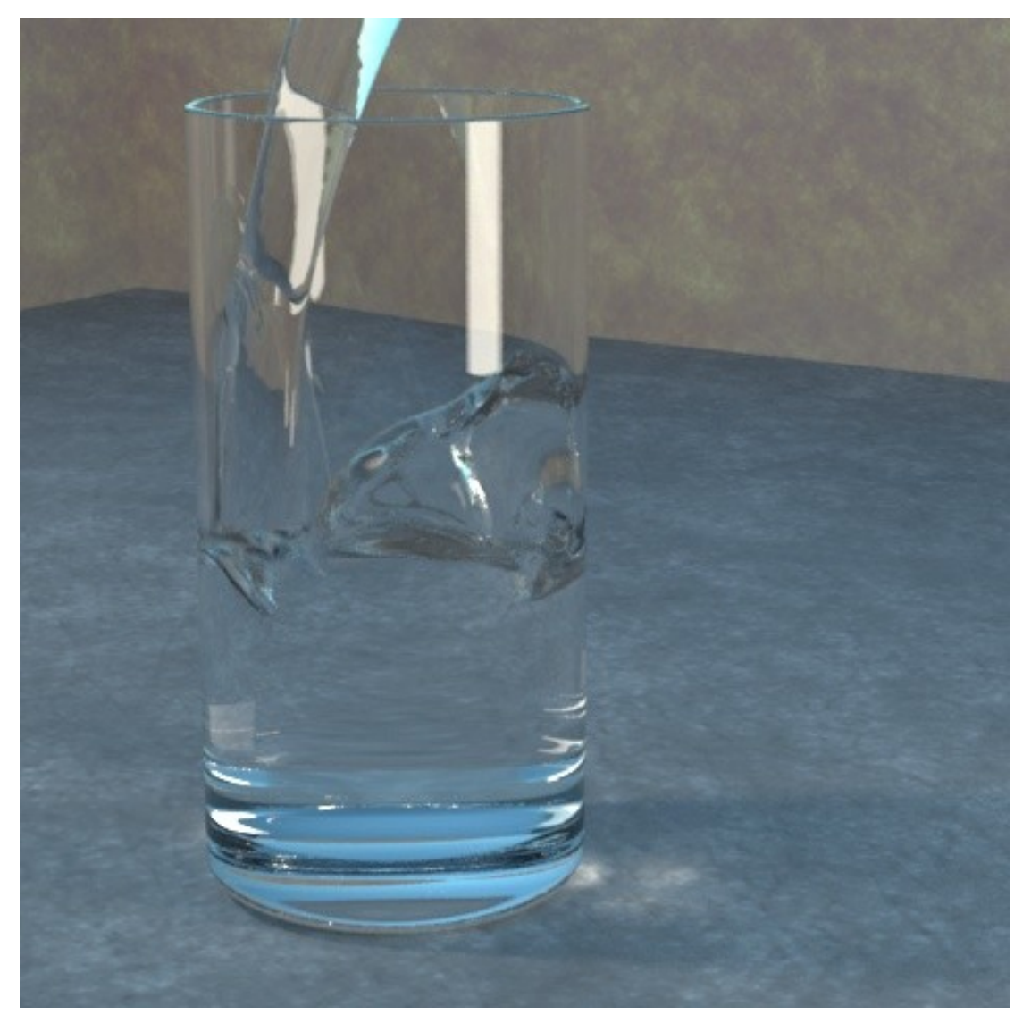
GPU Speedy Particles and Grids
(A GPU Implementation of the 3D Particle Level Set Method)

Wen Zheng & Douglas Enright, Ph.D.

Dept. of Computer Science, Stanford University and NVIDIA Research & Enright Labs

zhw@stanford.edu & dpephd-research@yahoo.com



Abstract

The use of grid-based interface techniques such as the level set method of Osher and Sethian offers the allure of a simple, yet highly flexible interface representation through the use of a simple scalar (level set) function. However this ease of use and flexibility comes at a cost with the need to store and compute using a volumetric scalar function as opposed to a more storage and operationally count efficient 2D mesh representation. Numerical diffusion and consequently inaccurate interface representation plagues grid-based methods on low-resolution grids typically used due to these computational volumetric inefficiencies. Volumetric level set operations are highly parallel and with the advent of large memory GPUs combined with previous algorithmic improvements such as augmenting the level set with particles (the Particle Level Set Method), high throughput ("speedy") and flexible grid-based interface techniques for use in real-time systems is quickly becoming a reality.

Particle Level Set Method

The Particle Level Set (PLS) method of Enright, et al. [1] is a Lagrangian enhancement of the grid-based Level Set (LS) method of Osher and Sethian (see [3] for additional information) for the accurate capturing of complex topological interfaces on low-resolution grids. The PLS method maintains the robust topological properties of the LS method, i.e. merging and pinching off, making its use as an interface representation for overturning and separating free surface flows (drops and waves) and other physical phenomena ideal. Some examples of the robustness of this technique for the use of free surface animations can be found in [2] and seen in the water pouring image contained in the upper right-hand side of the poster.

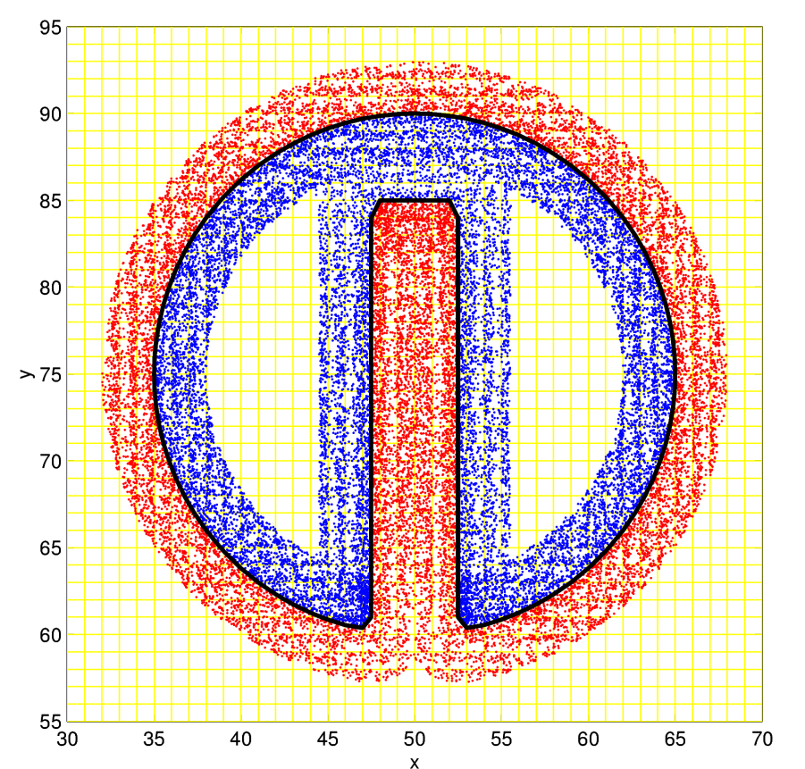
The method represents the interface as a signed distance function ϕ which is evolved in a flow field according to:

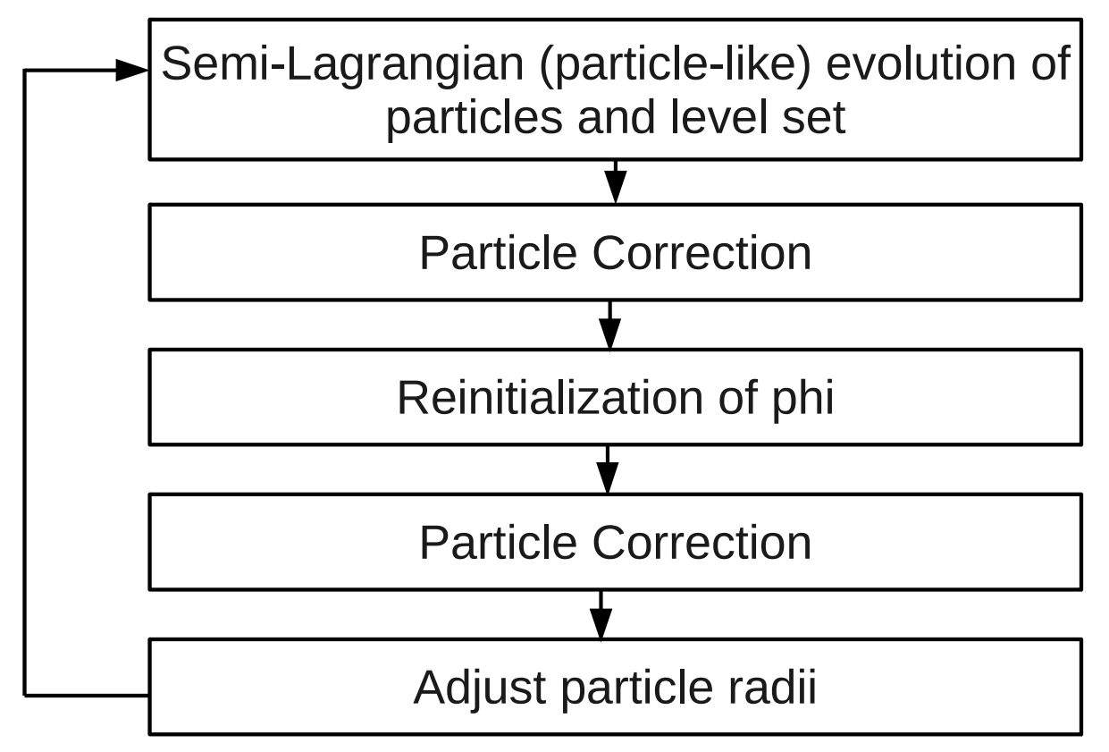
$$\phi_t + \vec{v} \cdot \nabla \phi = 0.$$

The interface itself is the 0-isocontour. Grid-based interface representations are subject to excessive amounts of numerical diffusion when evolved on low-resolution grids, adding particles nearby the interface which store distance information to the interface plus what side of the interface they reside on can be used to correct numerical evolution generated diffusion errors. The particles are evolved using the same velocity field as the level set by $d\vec{x}/dt = \vec{v}(\vec{x}, t)$

A particle correction and signed distance reconstruction (reinitialization) phase then takes place. Finally, the particles then resample the corrected level set values. This procedure along with a 2D visual depiction of the particles and level set comprising the PLS representation of the interface are shown below. The PLS method is evolved using a semi-Lagrangian method for speed since it does not result in a degraded interface representation and allows for a larger timestep than the standard CFL constrained time step.

A high-order RK3+WENO(5) reinitialization method is used. Even with relatively speedy particle-like evolution algorithms, the need to reinitialize the level set with a slower, but highly parallelizable, grid-based high-order RK3+WENO(5) method imposes a severe overhead to the overall runtime of a 3D PLS method. Fast Marching (FM) reinitialization has been shown to be a robust method, but FM methods do not parallelize well due to the heap-based nature of the method.





GPU Computing

The majority of the operations comprising the PLS can be performed in parallel. The evolution of each of the points in the grid and the particles occurs independently of each other. This architecturally attractive feature of the PLS method makes it especially suited for parallel throughput architectures such as the massively parallel GPU architecture from NVIDIA [6]. Instead of using 1, 2, 4, 8, or 12 CPU cores supplied with relatively slow DDR3 memory (~12 GB/s), an NVIDIA GPU such as a Tesla C2050 makes available 448 cores connected to GDDR5 memory which provides a very fast 144 GB/s of peak memory bandwidth. The ever increasing device memory size on the Tesla HPC product line makes high-fidelity PLS interface capturing now possible.

3D GPU PLS

Our GPU implementation of the 3D PLS method extensively utilized the parallel design patterns provided by the Thrust library [7], especially for parts not straightforward or efficient to implement using simple CUDA kernels. Use of the highly abstracted Thrust C++ library along with CUDA avoids the need to learn more dedicated graphics APIs and constructs such as OpenGL Vertex Buffer Objects (VBOs) and High-Level Shader Languages (HL-SLs) which have been used previously to implement PLS and similar methods on a GPU [4,5]. Our implementation is based upon the previous work of Zheng [8].

Considering all parts of the algorithm, the level set correction by the particles is one of the most difficult parts to be implemented on a GPU. Every particle need to compute a correction value for each of the eight grid points around it. Naively implemented, this involves approximately *N* atomic operations per each cell where *N* is the number of particles per cell. In addition, only those particles which have escaped from the level set surface need to compute correction values, so a compaction procedure is needed for efficiency. This is done by using the `remove_if` function in the Thrust library. Then escaped particles are replicated into eight copies, so that every cell around a particle has a copy. The cell index is then given to the corresponding copy of particle as its key value. To cluster particles by cell, the `sort_by_key` function in the Thrust library is called. This facilitates the following step of collecting correction values per cell by calling the `reduce_by_key` function. As a result, we attain a maximum correction value per each key value, i.e. per each cell.

Hardware & Software

For the GPU computational results and runtimes presented, we utilized a Tesla C2050 card. A Tesla C2050 GPU contains 488 streaming processing cores able to access 3.0 GB of total device memory with 144 GB/s of peak memory bandwidth. The single-threaded (1T) timings were obtained with a Core i7-2630QM (2.0 GHz). g++ v. 3.4 was used with -O3 -fomit-frame-pointer -fpermissive. Per Frame runtimes reported are calculated from the average time per step for the entire run.

Conclusions

<u>GPUs do indeed make the highly parallelizable particle and grid operations comprising our 3D PLS method "speedy".</u> The 200^3 3D Deformation Test was sped up by a factor of 45x on a Tesla C2050 as compared to a Core i7-2630QM single-threaded runtime. Even if perfect parallel scalability could be achieved on the quad-core Core i7-2630QM, a 10x GPU:CPU speedup would still be achieved. The use of a high productivity parallel programming API such as Thrust and CUDA enabled the generation of a 3D PLS implementation without excessive programmer effort. Use of a Fast Sweeping or similar method for reinitialization would likely result in additional performance gains. A comparable level set calculation, i.e. recovering the starting sphere, would take at least a 400^3 grid with an approximately 5-10x longer runtime, making the use of a PLS method critical for realistic real-time performance.

References

1. Enright, D. et al., "A Hybrid Particle Level Set Method for Improved Interface Capturing", *J. Comp. Phys.* 183, (2002).
2. Enright, D., Marschner, S. and Fedkiw, R., "Animation and Rendering of Complex Water Surfaces", *ACM TOG* 21(3), (2002).
3. Osher., S. and Fedkiw, R., "The Level Set Method and Dynamic Implicit Surfaces", Springer-Verlag, New York (2002).
4. Cuntz, N. et al., "Particle Level Set Advection for the Interactive Visualization of Unsteady 3D Flows", *Computer Graphics Forum* (2008).
5. Xing, M. et al., "Real-Time Marker Level Set on GPU", *Int. Conf. on Cyberworlds (CW'08) (*2008*).*
6. Nickolls, J. and Dally, W., "The GPU Computing Era", *IEEE Micro* 30(2), (2010).
7. Hoberock, J. and Bell, N., "thrust – Code at the speed of light", http://code.google.com/p/thrust/, [Accessed February 14, 2011].
8. Zheng, W., "Parallelizing the Particle Level Set Method", NVIDIA GTC 2010, San Jose, CA (2010).

Acknowledgments

D.E. would like to acknowledge previous support of a NSF Mathematical Sciences Postdoctoral Research Fellowship (DMS-0202459). W.Z. wishes to thank NVIDIA Corporation for a 2011 NVIDIA Graduate Fellowship. Tesla C2050 timings were made possible by an AY2011 Startup Award from the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Additional computer resources were made available by Associate Professor Ron Fedkiw.

Three-Dimensional Deformation Field Test Case [1]

200^3 semi-Lagrangian Level Set Method with High-Order Reinitialization

Frame #: 0
Frame #: 10
Frame #: 80
Frame #: 120
Frame #: 160
Frame #: 200
Frame #: 240

200^3 semi-Lagrangian Particle Level Set Method with High-Order Reinitialization

Frame #: 0
Frame #: 10
Frame #: 80
Frame #: 120
Frame #: 160
Frame #: 200
Frame #: 240

Tesla C2050: 0.00 s	Tesla C2050: 33.74 s	Tesla C2050: 269.93 s	Tesla C2050: 404.89 s	Tesla C2050: 539.85 s	Tesla C2050: 674.82 s	Tesla C2050: 809.78 s
i7-2630QM (1T): 0.00 s	i7-2630QM (1T): 1509 s	i7-2630QM (1T): 12070 s	i7-2680QM (1T): 18105 s	i7-2630QM (1T): 24140 s	i7-2630QM (1T): 30175 s	i7-2630QM (1T): 36210 s

The "Length" of the above Tesla C2050 GPU run.

1.75 in., **i7-2630QM(1T) Frame #10 = 78 in, almost 2 entire Tesla C2050 runs!**

42" == 809.78 s