A Material Point Method Imlementation in OpenCL

Chen, Hao
chenhao@andrew.cmu.edu
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

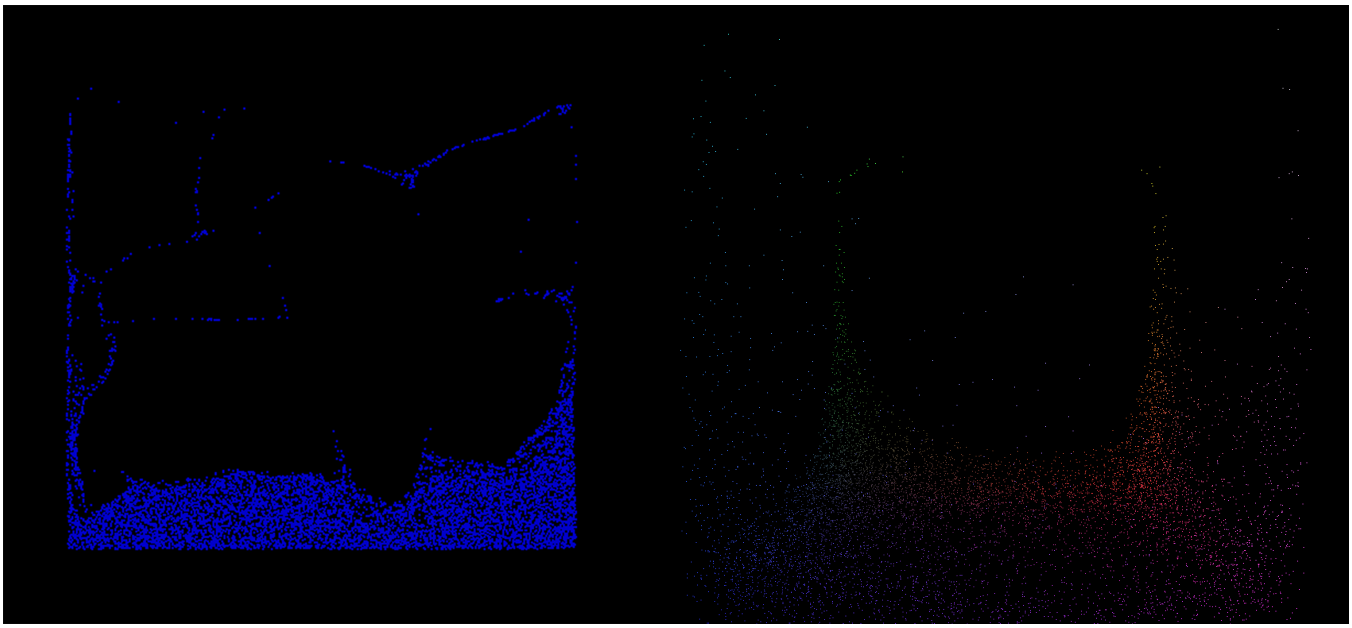

Figure 1: 2D and 3D fluid simulation demo

ABSTRACT

This project implements the Material Point Method in OpenCL, aim to be able to run on consuming-level platforms with considerable cross-platform capability. Even the OpenCL framework has sub-optimal functionality and lack of development supporting toolkit, we managed to develop a 2D fluid simulation as a proof-of-concept and a 3D final version.

CCS CONCEPTS

• **Computing methodologies** → **Modeling and simulation**.

KEYWORDS

material point method, opencl, simulation

Visual Computing System, May 04 – 06, 2022, Pittsburgh, PA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Chen, Hao. 2022. A Material Point Method Imlementation in OpenCL. In *Proceedings of (Visual Computing System)*. ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Because one of this project's goal is to be able to efficient enough to run on consuming-level personal computers, performance is our concern. The project can be divided into two parts: how to efficiently simulate fluid movement, which is computation, and how to present the simulation result, which is visualization.

For the computation part, this project uses OpenCL API to utilize GPU in order to parallelize the heavy computing tasks. As to the result visualization, we adopted an interoperability feature to efficiently render the simulation process onto screen with OpenGL.

Since OpenCL has a better cross-platform capability as its open nature, after implementing the simulation algorithm in OpenCL, it can be easily migrated to other heterogeneous platforms, not only limited by any graphics device vendor's specific product. OpenCL is supported by mainstream GPUs and CPUs, even by FPGAs. This can make more amazing fun opportunities happen, while maintain comparatively good performance.

2 RELATED WORK

Computer simulation methods have been popular in multiple fields ranging from engineering to entertainment industry. It can save the great cost to do experiments in real world and handling some complex problems which is not likely to be solved analytically. In computer graphics and entertainment industry, it can bring in beautiful and reasonable visual results.

Nowadays, the simulation methods can be roughly categorized into two groups, based on how they view the problem. Finite Element Method raised in 1943[2] is a classic Lagrangian method. It views the material as a set of primitives. It can be triangle, particle, or cube. By tracing the primitives over time, the material change over time can be simulated. However, due to its nature, when handling large deformation, some numeric precision problems may occur. People tried to mitigate the distortion by re-meshing, but finding a sound re-meshing strategy is never a simple problem itself.

Another way is to view the whole domain as a field, which is the Eulerian method. In the discrete settings, the space is usually divided into grids or cubes. Thanks to the naturally efficient neighbor look-up in grids, collision and pressure calculation is much easier with Eulerian method. However, it suffers from loss of energy when handing advection. The predefined domain restricts the problem solution space, too.

Under the multiple constraints encountered in both simulation ways, a class of hybrid method emerged. Material Point Method is a popular one of them. Hybrid methods typically combine particle and grid view of the problem. The simulation process is divided into two phases: advection and projection. By solving advection in the particle view and resolving projection in the grid view, the advantages of both methods can be taken. However, the transition step between particles and grids brings in some challenges to high performance computing.

Material Point Method is a numerical method for continuum material simulation, originated from 1995 by Sulsky, Chen, and Schreyer [12]. A bunch of variants was proposed in the following years. For example, the *BSMPM* by Steffen, Kirby, and Berzins in 2008[10], the generalized interpolation material point method by Bardenhagen and Kober in 2004[1], the convected particle domain interpolation technique by Sadeghirad, et al. in 2011[8], and so on.

In 2013, this method attracts great attention for its use for snow simulation in the Disney movie "Frozen".[11] The proposed technique produced convincing visual and physics snow result with some approximation tricks, though the computation workload is relatively large. It took weeks to complete the work for a single scene.

Later, various improved MPM methods showed up. This project employs the *Moving Least Square Material Point Method* variant proposed by Hu, et al. in 2018.[4] It is built upon the *APIC* method[6] , which captures more patterns during the particle-grid transition steps to mitigate the energy loss in the simulation process, especially the loss of the angle momentum. By keeping more degrees of freedom in this way, it is proved to be weak-form consistent. Hu's algorithm keeps the weak-form consistency while simplifying the implementation by reusing the intermediate results in *APIC* method.

However, the above algorithms are mostly implemented on CPU. GPU stories would be quite different. In some recent research[3], some other problems, e.g. atomic aggregation operations, high-performance sparse data structures, were investigated. However, all results are implemented in CUDA, which offers exclusive features on specific hardware. This is justifiable because of the popularity of certain hardware among the market. However, it also creates some barriers for the other platforms or users without certain level of hardware from specific vendor, stops a large group of possible users from enjoying the benefits. The migration cost is also increased with the hardware-specific implementation.

3 METHOD

This project adopts the *MLS-MPM* method metioned above. The implementation generally followed the Material Point Method process flow, with some modifications.

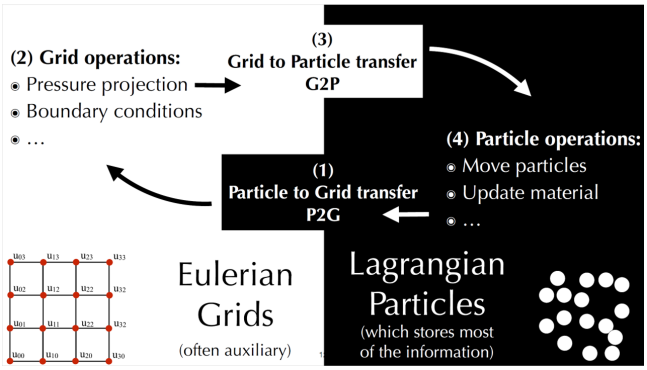

Figure 2: MPM simulation general flow from course GAMES-201 Advanced Physics Engine, by Yuanming Hu. (https://games-cn.org/games201/).

The four steps in the Material Point Method are merged into three by handling the particle operations at the end of the grid-to-particle step, as shown in the figure 3, because these two steps are both executed in the granularity of particle, and can be parallelized well. Reducing unnecessary kernel emission can avoid extra overhead, including CPU-GPU communication and redundant arguments passing.

The workflow of this program is:

Initialize OpenCL & OpenGL context First, the main program started on CPU brings up the OpenCL and OpenGL context based on the underlying system, along with some miscellaneous preparations, e.g., kernel compilation, shader program compilation & linking.

Create GPU buffer & fill in data Second, the CPU program creates necessary buffers on GPU device, and generates proper initial demo data and write to GPU device buffers. This is a one time process. Except for the initial random position data which is generated on CPU side and then filled into GPU buffer, other buffers are filled by firing GPU commands, which is more efficient. After this step, CPU will be agnostic about the data through all the following simulation process,

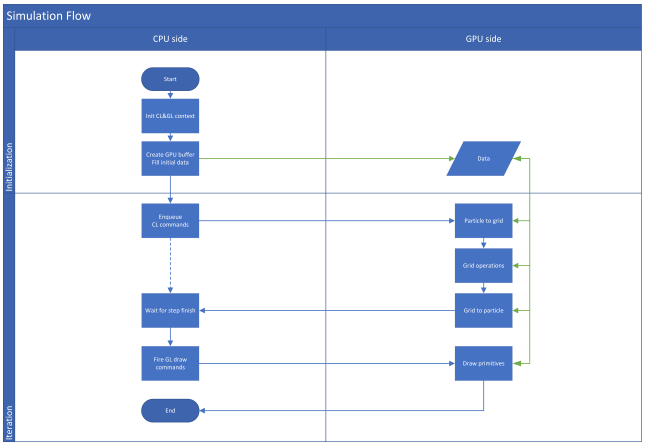

Figure 3: The flowchart of this project's simulation and rendering process.

Enqueue CL commands Third, the simulation steps start. The simulation work is divided into three kernels, which run with different group size settings. The particle-to-grid kernel with one dimensional size, which is equal to the number of global particles, distributes each particle's momentum to nearby grid nodes. The influence range is chosen to be 3 in each dimension, following the regular *APIC* practice. The grid-operation kernel will with a size equal to the number of grid nodes will add gravity influence globally, and then check boundary condition, zero the corresponding velocity components if any boundary grid node has the effect of letting particles escape the grid domain. The last grid-to-particle kernel gather momentum back to particles from nearby grid nodes, and update particles' data.

GPU execution GPU side will execute the commands in queue in order, without synchronization with CPU. This asynchronous way of execution of commands in batch avoid overhead caused by synchronization.

Wait for simulation finish CPU will wait for GPU device to finish the commands in the queue, then fire OpenGL rendering commands.

Fire GL draw commands OpenGL will draw the simulation result to screen, triggered by CPU command, but read data directly from GPU device memory buffer.

3.1 OpenCL & OpenGL Interoperability

Because one of this project's major concern is performance, we want to parallelize our workload as much as possible onto GPU (or other parallel computing device). As mentioned in the introduction, this project can be divided into two different parts: how to simulate and how to present. The simulate part has been decided to be built upon OpenCL, which is the major goal of this project; then we decided to put the drawing part onto OpenGL.

OpenCL & OpenGL interoperability allows us to share some buffer, which is on-device memory, between OpenCL and OpenGL, so that we can be saved from moving data around between host and device. All the computation and rendering work will be totally completed on GPU device with all the data it needs at hand. CPU just initializes the context and starting data, then is only responsible for some minimum scheduling work.

3.2 Multiple Buffering

We create double buffers for the shared particle position buffer between OpenCL and OpenGL. By using one buffer for read and another for write, and switching them after each iteration step, we can acquire some performance gain.

3.3 Adaptive Iteration Choice

The number of iterations is chosen in an adaptive way. It depends on the duration between this frame and last frame. This is to ensure we always simulate for enough time to fill in the gap between this frame and last frame, so that the speed of material movement can be more stable.

3.4 Customized Atomic Operations

OpenCL only provides limited atomic operations for integer scalar types, so to implement the momentum scattering in the particle-to-grid kernel, we have to implement the float vector atomic increment operations with type casting and atomic "compare and exchange" operation.

4 RESULTS

We implemented a 2D version as a proof-of-concept first, then developed 3D version based on it. The 2D version runs pretty fast on the low-end Nvidia MX450 laptop graphics card with 8192 particles and a 128 * 128 2D grid. The frame rate can reach 110 fps, which is much higher than the screen refresh rate, making the OpenCL implementation promising.

A later 3D version with 8192 particles and 32 * 32 * 32 3D grid can run on the same platform at 60 fps, showing a satisfactory result.

5 FUTURE WORKS

Even the current implementation shows expected results, but for practical use in the future, this project still has a long way to go. We can tell that there are some obvious possible improvements waiting to be implemented now. However, due to multiple constraints, like the time and limited manpower, I cannot finish them alone currently.

5.1 Alternatives for Atomic Operations

Because OpenCL lacks development supporting tools, e.g., debugger and profiling tools, we can hardly decide the performance killer. However, we managed to measure the kernel time consumption.

It shows that the particle-to-grid kernel took over 90% of the total GPU time. Without profiling data in finer granularity, we assume that the atomic operations in this kernel consumed the most of the time because of the heavy shared data contention.

Several previous researches took different approaches to mitigate this problem. The recent GPU CUDA implementation took advantage of Nvidia intrinsics [3], which is not an option in our case. Some others sort the particles according to their spatial position, and then gather by traversing grid nodes. Some other implementation update the grid node properties in an interleaved fashion [4].

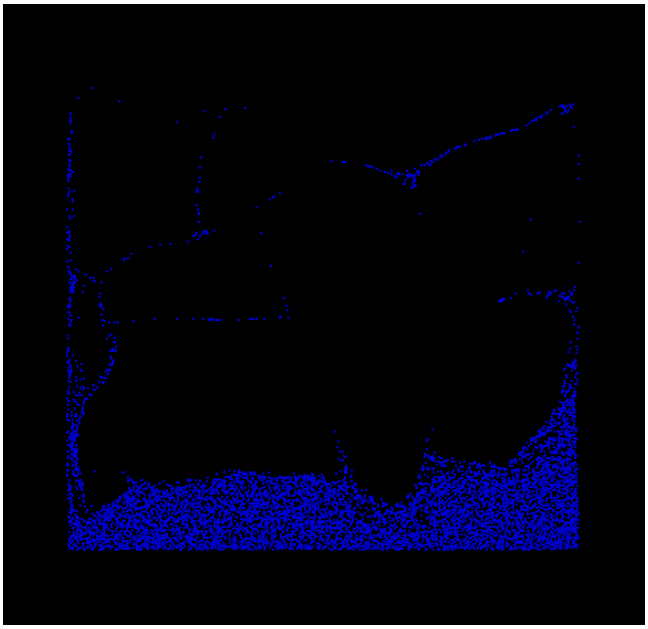

Figure 4: The screenshot of the 2D version of MPM implementation in OpenCL

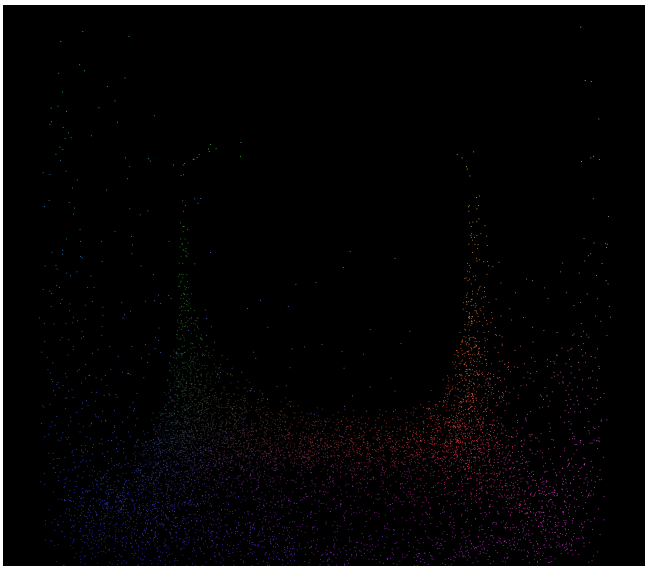

Figure 5: The screenshot of the 3D version of MPM implementation in OpenCL

The latter two methods can be investigated further for our use in the future.

5.2 Sparse Data Structure

Doing global operations in a large scale system certainly suffers from the "curse of dimension". A sparse data structure can improve the performance considerably. Some former researches proposed

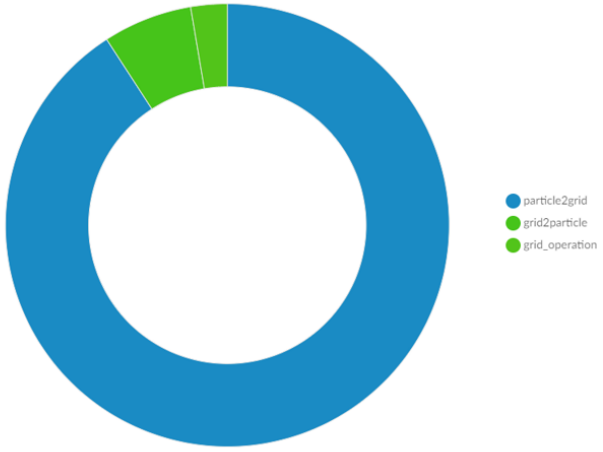

Figure 6: Kernel profiling result for the 3D version Material Point Method in OpenCL

promising data structures, e.g., *OpenVDB*[7], *SPGrid*[9]. The later work migrated *SPGrid* onto GPU [3], making it even more attractive. *SPGrid* exploits spatial locality by applying "Morton Pattern". Some work combines them together, making the sparse data structure choice more flexible. [5]

REFERENCES

[1] S. Bardenhagen and Edward Kober. 2004. The Generalized Interpolation Material Point Method. *CMES - Computer Modeling in Engineering and Sciences* 5 (June 2004).

[2] R. Courant. 1943. Variational methods for the solution of problems of equilibrium and vibrations. *Bull. Amer. Math. Soc.* 49, 1 (1943), 1–23. https://doi.org/10.1090/S0002-9904-1943-07818-4

[3] Ming Gao, Xinlei Wang, Kui Wu, Andre Pradhana, Eftychios Sifakis, Cem Yuksel, and Chenfanfu Jiang. 2018. GPU optimization of material point methods. *ACM Transactions on Graphics* 37, 6 (Dec. 2018), 1–12. https://doi.org/10.1145/3272127.3275044

[4] Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. 2018. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics* 37, 4 (Aug. 2018), 1–14. https://doi.org/10.1145/3197517.3201293

[5] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. 2019. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics* 38, 6 (2019), 201:1–201:16. https://doi.org/10.1145/3355089.3356506

[6] Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. 2015. The affine particle-in-cell method. *ACM Transactions on Graphics* 34, 4 (July 2015), 1–10. https://doi.org/10.1145/2766996

[7] Ken Museth. 2013. VDB: High-resolution sparse volumes with dynamic topology. *ACM Transactions on Graphics* 32, 3 (June 2013), 1–22. https://doi.org/10.1145/2487228.2487235

[8] A. Sadeghirad, R. M. Brannon, and J. Burghardt. 2011. A convected particle domain interpolation technique to extend applicability of the material point method for problems involving massive deformations. *Internat. J. Numer. Methods Engrg.* 86, 12 (2011), 1435–1456. https://doi.org/10.1002/nme.3110 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.3110.

[9] Rajsekhar Setaluri, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis. 2014. SPGrid: a sparse paged grid structure applied to adaptive smoke simulation. *ACM Transactions on Graphics* 33, 6 (Nov. 2014), 1–12. https://doi.org/10.1145/2661229.2661269

[10] Michael Steffen, Robert M. Kirby, and Martin Berzins. 2008. Analysis and reduction of quadrature errors in the material point method (MPM). *Internat. J. Numer. Methods Engrg.* 76, 6 (2008), 922–948. https://doi.org/10.1002/nme.2360 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.2360.

[11] Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. 2013. A material point method for snow simulation. *ACM Transactions on Graphics* 32, 4 (July 2013), 1–10. https://doi.org/10.1145/2461912.2461948

[12] Deborah Sulsky, Shi-Jian Zhou, and Howard L. Schreyer. 1995. Application of a particle-in-cell method to solid mechanics. *Computer Physics Communications* 87, 1 (May 1995), 236–252. https://doi.org/10.1016/0010-4655(94)00170-7