GPGPU IMPLEMENTATION OF SCHRÖDINGER'S SMOKE FOR UNITY3D

Oleg Iakushkin [a], Anastasia Iashnikova, Olga Sedova

Saint Petersburg State University, 7/9 Universitetskaya nab., St. Petersburg, 199034, Russia

E-mail: [a] o.yakushkin@spbu.ru

The paper describes an algorithm for Eulerian simulation of incompressible fluids — Schrödinger's Smoke. The algorithm is based on representing models as a system of particles. Each particle represents a small portion of a fluid or amorphous material. A particle has a certain 'lifespan', during which it may undergo various changes. The Schrödinger's Smoke solver algorithm was implemented in Unity3D environment. We used ArrayFire library to transfer the bulk of computational load relating to simulation of physical processes to GPGPU — it allowed real-time interaction with the model. The solution we developed allows to model interactions between vortex rings — i.e., their collisions and overlapping — with a high degree of physical accuracy.

Keywords: particle system, GPGPU, Unity3D, ArrayFire

1. Introduction

We examined the work on Schrödinger's Smoke [1] that demonstrates a new approach to modelling Eulerian incompressible fluids. We transferred its algorithm to the multi-platform Unity3D development environment using ArrayFire, a library for transferring computing to GPGPU.

2. Approach

Initially, we wanted to use the Compute Shader technology available in Unity3D as the main computational tool in order to transfer most of the algorithm calculations to the GPGPU. However, we had to abandon this idea because Compute Shader turns out to be unreasonably complicated when it comes to working with the FFT and IFFT methods for three-dimensional arrays of points in a complex space. To solve the problem, we chose the ArrayFire cross-platform library—it limited our ability to compile a Unity3D project for WebGL and mobile platforms, but still allowed to target Mac OS, Linux and Windows that support OpenCL, CPU and CUDA backends.

3. Implementation Details

We used two main frameworks to implement our idea:

• Unity3D — a cross-platform engine that provides Shuriken programmable particle control system;

• ArrayFire — a library for accelerated computation of mathematical operations that supports parallel computing on various GPGPU coprocessors.

3.1. Particle System

We used the Unity3D Particle System component to visualize the results of the Schrödinger's Smoke algorithm.

The particle system is a method used in computer graphics and interactive simulations to represent objects that do not have clear geometric boundaries. This makes it possible to create practically any type of natural effects—e.g., explosions, jets of steam, nebulae, rain, snow, water splashes, fire smoke, etc.

The particle system controls groups of particles—that is, small flat images or mesh objects. Any particle has a certain 'lifetime' (usually a few seconds) during which the particle may undergo various changes.

3.2. ArrayFire Library

The ArrayFire library provides a simplified interface that allows to run different mathematical computations in parallel on various GPU devices. It is an open source universal computing tool that facilitates the development of software for parallel and massively parallel architectures, including processors, graphics processors, and other hardware accelerators.

The library has a number of features that are important for our task:

- its basic implementation in C, C ++, and CUDA has a c # shutter;
- it integrates and extends many mathematical methods available in CUDA and OpenCL, including FFT, BLAS and LAPACK implementations, as well as signal and image processing;
- it speeds up code development by allowing to replace a large number of code lines in CUDA or OpenCL with just a few lines of the ArrayFire code;
- it allows author to choose the platform where the calculations are performed by providing CUDA, OpenCL and CPU implementations for most of its mathematical methods;

- it supports manipulation of vectors, matrices and three-dimensional arrays.

4. Our GPGPU Implementation Speed-Up

We implemented the Schrödinger's Smoke algorithm and tested it on CPU and GPU.

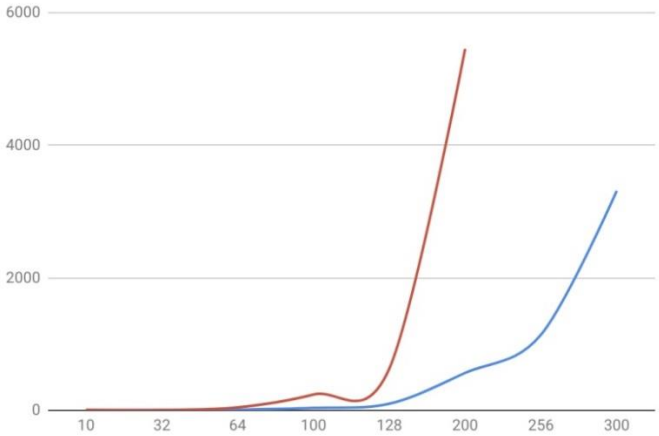

Figure 1. A benchmark (averaged on a set of three runs) of our implementation for a given amount of particles (x-axes) to a runtime in milliseconds (y-axes). The blue line (bottom) shows the time spent by the GPU implementation, while the red line (top) shows the CPU time. The results were computed on an i5 7th generation Intel CPU, with GPU integrated on the same i5 chip

The test results shown in Figure 1 demonstrate that the Schrödinger's Smoke algorithm can be accelerated even for the use on embedded GPUs without any drastic modifications of its logic.

5. Acknowledgement

This research was supported by SPbU (Saint Petersburg State University) grant no. AAAA-A18-118071790047-9 (id: 28612502). The authors would like to acknowledge the reviewers for the valuable recommendations that helped improve this paper.

6. Conclusion

We developed a solution that allows us to simulate particle systems with high physical accuracy using real-time GPGPU technologies. This solution makes it possible to simulate the interaction of two vortex rings—their collisions and intersections. In future work, we will test our implementation on a supercomputing system and further refine its performance. We will also integrate our current results into an archaeological modelling project [2, 3].

References

[1] Albert Chern, Felix Knöppel, Ulrich Pinkall, Peter Schröder, Steffen Weißmann. Schrödinger's smoke. // Journal ACM Transactions on Graphics (TOG) Vol. 35 Issue 4, pp. 77-90.

[2] Iakushkin, O., Fatkina, A., Plaksin, V., Sedova, O., Degtyarev, A.,Uteshev, A. / Reconstruction of stone walls in form of polygonal meshes from archaeological studies. Lecture Notes in Computer Science, Springer, 2018, Vol. 10963, pp. 136-148.

[3] Oleg Iakushkin, Dmitrii Selivanov, Liliia Tazieva, Anna Fatkina, Valery Grishkin, Alexei Uteshev 3D Reconstruction of Landscape Models and Archaeological Objects Based on Photo and Video Materials // Lecture Notes in Computer Science. Springer, 2018, Vol. 10963. pp. 160-169.