Computer Science Tripos – Part II – Project Proposal

Simulating Wavefronts in Real-Time using Wave Particles

T. M. Read Cutting, Downing College

Originator: Huw Bowles

13 October 2016

Project Supervisor: Dr R. Mantiuk

Director of Studies: Dr R. Harle

Project Overseers: Prof L. Paulson & Dr I. J. Wassel

Introduction

Simulating water in real-time (meaning an operating speed of at least 30 frames per second), continues to be a big challenge, consistently being used as a watermark for the current state of real-time graphics [1]. With the emergence of both Virtual and Augmented Reality, the applications of complex real-time computer graphics are only going to grow, from entertainment all the way through to full-on education, visual aids, simulations and training.

To accurately simulate fluids such as water, full three dimensional particle/grid-based simulation systems are required. This is something that until very recently, was impossible to do in real-time, although advances in GPU technology have allowed for small-scale simulations to become possible in real-time [2].

For large-scale systems, real-time applications use two dimensional height-map based techniques to simulate waves. While these systems can't capture the changes in topology caused by splashes or other phenomena where water overlaps itself, they can operate at a scale where three-dimensional techniques still aren't feasible.

Wave Particles are such a technique, that capture how wavefronts form and propagate by representing them with particles on a 2D plane. [3]. The purpose of this project is to implement this technique, porting some of the functionality from the CPU to the GPU for improved performance. Specifically porting my own implementation of the rigid-body mechanics to the GPU, which should provide some noticeable gains over the CPU-based solution used in the Wave Particles paper.

Starting point

I already have a theoretical understanding of the workings of the GPU, but am unfamiliar with programming against one in practise. I will have to learn HLSL, the shading language of Unity [4], which will then compile it to the native shading languages of other platforms for portability.

I have read various papers surrounding water and fluid simulation, but will need to thoroughly revise the mathematics involved.

I will start with an empty Unity project, in addition to learning the framework as I am unfamiliar with it. The default programming language for Unity projects is C# [5], which is similar to Java, a language I do have a lot of experience with. However, it is possible to invoke C/C++ code from Unity, something that I will investigate as it should speed up the execution of the CPU-based simulations.

The LaTeX in the example dissertation will be used as an aid in constructing my own one.

Finally, Huw Bowles, a professional graphics programmer, has given me advice on which papers to read and which technologies to use, to ensure I start in the right direction.

Resources required

For this project I shall mainly use my own quad-core computer with an Intel i5 CPU and nVidia GTX 1070 GPU.

I will use Git as source control, backing my code up on GitHub, an external hard drive, both my laptop and deskop, and Google Drive. Finally, the project will be implemented within the game engine Unity, as it will allow me to focus on the problem itself, without having to worry about setting up a lot of boilerplate.

If my computer and laptop both fail, I have money set aside in my budget to account for that. Finally, I have an arrangement with A. Toft (adt39) to use his well-specced computer if all else fails.

I will keep links to all the research I do on Google Drive, in addition to keeping a log of the research I do and the progress I make in a notebook.

Work to be done

The project breaks down into the following sub-projects:

1. Create a Unity project with the required structure, which will contain all the code that will be written for the project.
2. Implement a simple top-down 2D visualiser for the Wave Particles themselves, which can be overlaid onto a 3D scene. This visualiser can then also be used to view 'optimised' Wave Particles which could be implemented later. (See extensions)

3. Implement a CPU-based solution of Wave Particles, where the user can simply click on the 2D visualisation to simulate dipping or dragging an object through it. Setup at least five different environments with test-obstacles for the Wave Particles to bounce off in the simulation.

4. Render the wavefront formed by Wave Particles on a simplistic water surface, the focus of the project IS NOT to render the water in a realistic manner with reflections and refractions, just to simulate the behaviour of propagating waves.

5. Port the wave-particle and fluid-object interaction modules to the GPU. It is important that switching between the two implementations is simple to allow for them to be compared to each other.

Unit tests

In theory, real-time physics simulations in Unity are deterministic. However, they are very sensitive to the initial conditions of a system, so implementing unit-tests in terms of actor-behaviour can be tricky. An attempt will be made to set-up systems with consistent behaviour that can be used for unit-testing. Failing that, what can be unit-tested are the implemented equations and mathematics, which can be tested against expected results for certain inputs.

Success criteria

The project will be a success if I have completed the following:

1. A Wave Particle simulation in Unity that runs on both the CPU and GPU with the ability to easily compare the performance of each.

2. A fluid-object interaction system on both the CPU and GPU with the same criteria as above.

3. Have the complete GPU-based version of the simulation run at least 30 frames-per-second with 5000 Wave Particles present.

4. A 2D top-down overlay-visualiser of the wave-particles to enable easy visualisation of what is going on, for both explanatory and debugging purposes.

5. A basic water renderer for the wave fronts generated by the wave particles.

6. A system to manage the Wave Particles and object placement within the Unity editor.

7. Measurements comparing the performance of the CPU and GPU based implementations of specific features, in addition to, data of how the number of Wave Particles affect the performance of the simulation alongside the resulting visual outcome.

Possible extensions

If I manage to achieve all of the above, I will investigate the following possible ways of taking this project forward:

1. Optimise common wavefront types, by having them be represented by single Wave Particles that encode information about the shape of the wavefront. (eg. circular waves only need a single Wave Particle that encodes the origin of the disturbance (from which the resulting wave can be extracted), as opposed to having many particles arranged in a sphere. Additionally, similar information-encoding techniques can be used for straight-lines and maybe even certain kinds of wakes). This can also be easily visualised in the 2D visualising tool.

2. Handle the degenerate cases where splashes should occur, as these cannot be handled by a height-map based method such as Wave Particles. This is due the system being unable to handle changes in surface topology. This could by done using a simplified Smoothed Particle Hydrodynamics-like simulation, as the transient nature of splashes means certain forces could be ignored without a noticeable difference. The result of this can be compared with identical scenes in a full-featured offline water simulation package, such as the one included with the Blender animation software [6].

3. Investigate and implement more realistic water shaders for rendering the water. Measure how they affect the performance of the simulation in addition to how they compare to the real physical phenomena they represent.

4. Implement the ability to view the scene in virtual-reality. Importantly, this requires running at a framerate of at least 90 frames per second to help prevent motion sickness that can be exacerbated by lower framerates.

Timetable

Planned starting date is 2016-10-20.

1. **Michaelmas weeks 3–4, 2016-10-20 to 2016-11-02** Thoroughly read and understand the Wave Particles paper, reading further material as necessary. Make notes on further reading, keeping track of references! At this point, a first draft for the introduction to the dissertation can be constructed in parallel to the research around the area.

2. **Michaelmas weeks 5–6, 2016-11-03 to 2016-11-16** Learn the basics of the Unity framework by implementing a simple 2D visualiser of the Wave Particles. Register user-input on the 2D visualiser by simply displaying a circle where the user clicks.

3. **Michaelmas weeks 7–8, 2016-11-17 to 2016-11-30** Implement the Wave Particle simulation on the CPU, such that the Wave Particles are simply rendered on the 2D visualiser. Allow the user to click on the visualiser in order to "generate" waves, by simulating a spherical object falling in the water.

4. **Michaelmas vacation weeks 1–2, 2016-11-31 to 2016-12-14** Render the generated wavefronts caused by the Wave Particles within the Unity renderer. The focus here being to ensure that the surface shape of the waves is correct, not that accurate reflections/refractions are implemented.
5. **Michaelmas vacation weeks 3–4, 2016-12-15 to 2016-12-24** Implement fluid-object interaction on the CPU.
6. **Michaelmas vacation weeks 5–6, 2016-12-29 to 2017-01-11** Catch up on anything I may be behind on. If everything is going according to plan by this point, reschedule the timetable, make a decent headway on the dissertation concerning the parts of the project already achieved, and possibly investigate implementing any of the extensions in depth. Write a first draft of the progress report.
7. **Lent weeks 1–2, 2017-01-19 to 2017-02-01** Finish up the progress report, and port the relevant parts of the CPU implementation of the project to the GPU. By the end of this period the Wave Particle simulation itself should be implemented on the GPU.
8. **Lent weeks 3–4, 2017-02-02 to 2017-02-15** Finish port of code to GPU. The focus here being fluid-object interaction.
9. **Lent weeks 5–6, 2017-02-16 to 2017-03-01** Run tests comparing performance of implementation across GPU and CPU. Creating an appropriate databank of the results and documenting the most interesting findings.
10. **Lent weeks 7–8, 2017-03-02 to 2017-03-15** Catch up on anything I may be behind on. If everything is going according to plan, focus on getting a first draft of the dissertation done.
11. **Lent vacation, 2017-03-16 to 2017-27-04** Focus here is entirely on exams and the dissertation. However, if things are behind schedule, the entire Easter vacation should provide ample time to catch-up.
12. **Easter weeks 0–2, 2017-28-04 to 2017-10-05** Proofread the dissertation, getting feedback from people to polish things up in time for an early submission to focus on exams!

References

[1] Zak Kotzer. Rendering realistic water is still game development's moby dick. http://motherboard.vice.com/read/rendering-realistic-water-is-still-game-developments-moby-dick, 2015.

[2] Miles Macklin and Matthias Mller. Position based fluids. *ACM Transactions on Graphics (SIGGRAPH 2013)*, 32(4), 2013.

[3] Cem Yuksel, Donald H. House, and John Keyser. Wave particles. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3), 2007.

[4] Compute shaders in unity. https://docs.unity3d.com/Manual/ComputeShaders.html.

[5] Creating and using scripts in unity. https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html.

[6] Fluid simulation in blender. https://wiki.blender.org/index.php/Doc:2.4/Manual/Physics/Fluid.