Large Scale Particle Simulation on the GPU

Isaih Porter, Rishma Mendhekar, and James Lee

Connecticut College

Abstract

This study examines the increased performance of large-scale particle simulation on the Graphics Processing Unit (GPU) against conventional implementation on the CPU. Usage of General Purpose GPU (GPGPU) programming, which utilizes massively parallel computational algorithms, has grown substantially over the last decade and particle simulation is one of such examples. We developed a particle simulation program using a Compute Shader on the GPU to calculate particle motion with a 3 dimensional Perlin noise algorithm. The current implementation shows around 60 frames per second (FPS) in 4K resolution for about 8 million particles of a point primitive type as well as a quad sprite model. The performance gain over the equivalent version on the CPU is about a 200x speedup in frame rate. Both the CPU and GPU versions of the program were created using C# and HLSL with Unity and DirectX. The GPU used for testing was a Nvidia Geforce GTX 1080 and the CPU an Intel Core i7-6700k @ 4GHz. We deployed this program for the art installation of The Posture Portrait Project at Connecticut College to achieve an image-dissolving visual effect where each particle is generated from image pixels. Moving forward, we will be implementing boids flocking algorithms using a Compute Shader. This particle motion will require significantly more computation than Perlin noise as each particle will need to be aware of each other particle's location.

Introduction

We are researching methods of creating a large-scale particle simulation using a GPU. One of the problems with dealing with such a large amount of data is that the simulation takes substantial computational time due to the CPU's inability to process and interpret large numbers of pixels in real time. The goal of the research is taking most of the stress away from the CPU and giving the GPU most of the work. There is already a trend of software moving towards general-purpose computing on graphics processing units (GPGPU); such as Adobe Photoshop and Premier, and even MATLAB[5,7]. We analyze and compare the efficiency of the GPU to that of a CPU in order to optimize a real-time particle field using large, dense data. We utilize DirectX in conjunction with Unity, resulting in a Unity plug-in that creates particles based off of millions of pixels from an image or video. This data has been gathered by recording students walking in front of an Xbox Kinect; each of these short videos contain millions of pixels that are redrawn at 60 frames per second. The particles will be simulated depending on the pixels from images.

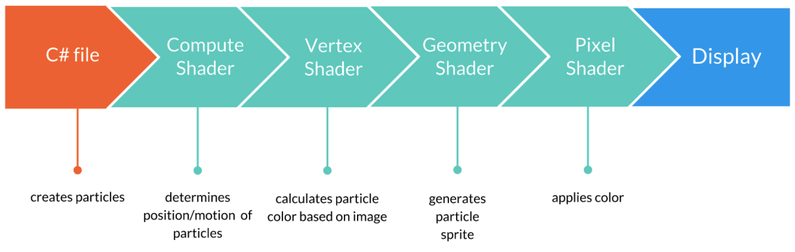

Figure 1. Graphics pipeline

Methods

First, we conducted a literature review on large-scale particle systems[1,2,3,6]. First, we created a particle simulation program which uses the GPU to calculate the motion of the particles. The program is made up of four components: a C# file, a compute shader which calculates the motion of each particle, a vertex shader which calculates particle color, and a pixel shader which applies color to each particle. When a user uploads an image, the program generates particles based on the pixels of the image. For example, if the user selects a 4K image (2048 by 4096 pixels) then there will be around 8 million particles. Each particle moves independently and randomly based on a Simplex noise algorithm implemented in the compute shader. We experimented with a variety of motion patterns before implementing the Simplex Noise and plan on implementing numerous motion patterns from which the user can choose.

We then created a second version of the program which uses the CPU to calculate the motion of the particles. More specifically, the motion of each particle is calculated in the C# file before the particle data is passed to the shaders. The CPU version of the program was created so that a comparison between the traditional method (CPU calculated motion) and a GPU-accelerated version of the program could be made. The comparison was achieved by tracking the frame rate of the simulation achieved by each of the two implementations. Finally, although the original particle structure was a point, we converted to a billboard structure so that the particles will face Unity's camera regardless of its position. This more advanced structure not only looks more aesthetically pleasing, it also will require more computation on the GPU.

For the tests, an Alienware machine was used: Intel I7-6700k CPU with 32GB RAM and Nvidia Geforce GTX 1080 GPU. By allowing the program to run using varying image sizes from 100 thousand particles all the way up to 8 million particles. By allowing each image to be simulated using the two different implementations, we collected and recorded the FPS (frames per second) over 30 second intervals.

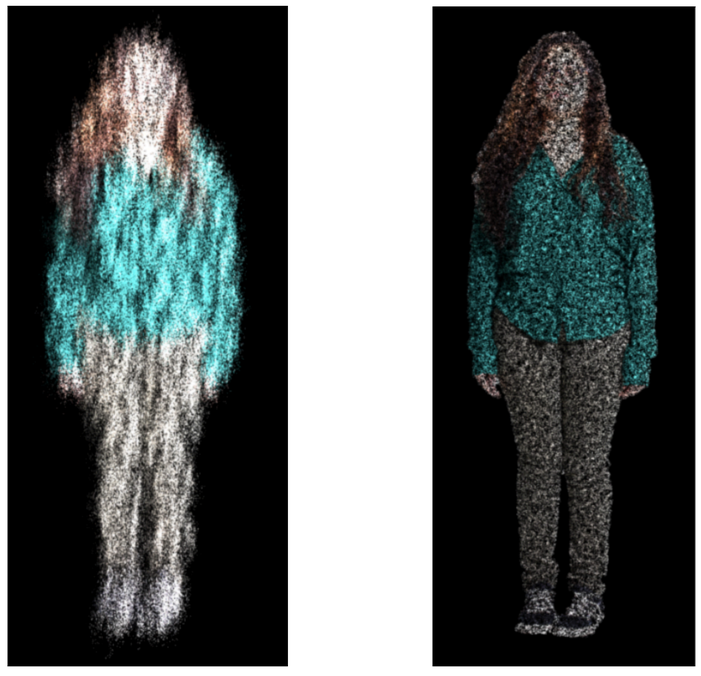

Figure 4. Caption in Calibri, 36 points, bold.

Results

The graph below illustrates the difference in performance between CPU and GPU implementations of the particle simulation.

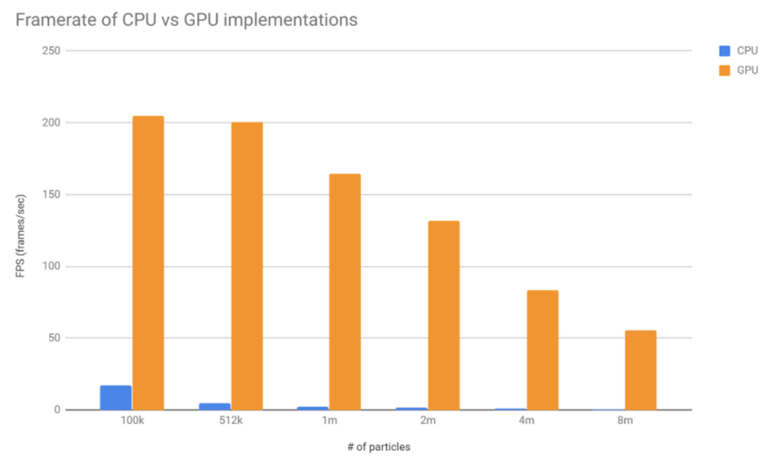

Figure 3. Framerate of CPU vs GPU Implementation

While the CPU can handle smaller number of particles at a somewhat acceptable frame rate (around 30 FPS), this implementation quickly became unbearable to watch when the number of particles was increased. The version of our program that uses a compute shader to change the particles' properties can maintain a smooth 60 FPS even with 8 million particles.

Conclusion

From the results found by this study, it is clear that Direct Compute technology drastically improves the performance of large-scale particle simulations. With most modern displays running at 60 frames, it is clear that the CPU needs more help with general computation by the GPU in order to create smooth simulations of millions of particles. GPGPU programming can be used to accelerate all kinds of software that includes particles simulations. Our future work regarding GPGPU simulation is to create an optimized version of Boid's flocking[8]. Like our Simplex noise experiments, our goal is to create a flocking program that simulates a massive number of boids at a high frame rate using the GPGPU to calculate boid movement.

Sources Cited

1. Direct to video. 2009. A thoroughly modern particle system. http://directtovideo.wordpress.com/2009/10/06/a-thoroughly-modern-particlesystem/
2. Jesper Hansson Falkenby. 2014. Physically-based fluid-particle system using DirectCompute for use in real-time games. Belkinge Institute of Technology. http://www.diva-portal.org/smash/get/diva2:832945/FULLTEXT01.pdf
3. Lutz Latta. 2004. Building a Million Particle System. https://www.gamasutra.com/view/feature/130535/building_a_millionparticle_system.php
5. "MATLAB GPU Computing Support for NVIDIA CUDA-Enabled GPUs." MATLAB & Simulink , www.mathworks.com/discovery/matlab-gpu.html.
6. Nathan Carr. Kyle Hegeman. Gavin S.P. Miller. 2006. Particle-Based Fluid Simulation on the GPU. https://link.springer.com/content/pdf/10.1007%2F11758549_35.pdf
7. "Photoshop Graphics Processor (GPU) Card FAQ." Adobe , helpx.adobe.com/photoshop/kb/photoshop-cc-gpu-card-faq.html.
8. Reynolds, Craig. "Boids Background and Update." *Reynolds Engineering and Design*, 29 June 1995, www.red3d.com/cwr/boids/.