Mitchell Luban

Timothy Luciani

P7 Write-up

Amortized Cloth Simulation

In recent years the simulation of cloth has become almost trivial. Most consumer computers nowadays contain the necessary hardware to simulate and render realistic cloth in close to, if not exact, real time. Though this feat may have seemed staggering to researchers in the last decade, the fact that it has been done makes us want to push the envelope even more. Even though simulating cloth on a typical consumer machine can be done, limitations still occur: Most machines must cap the number of particles that it can generate if it wants the results to remain real-time; As with the previous point, numerous cloth are sure to generate the same, if not worse, conditions as uncapped-particle simulation; and finally, cloth simulations on one machine are limited to the users present with the machine. Using the methods that will be outlined in this paper, we hope to solve each of the aforementioned limitations. Our solution, amortizing the cloth over a network of machines, will allow for the simulation of large particle-system cloth, the display of multiple particle cloths per scene, as well as the simultaneous viewing of the created scene from any user who can log into the server.

Since the goal is to create large particle-system cloths that multiple users can view, we decided to tackle the mass-spring method of simulation from a different angle. Since every user will want to view the same scene, the simulation will only need to be done once on a single back-end machine as opposed to once per client. By localizing the simulation on a server (back-end), the client machine’s only concern will be gathering the data from the server and rendering it to the screen.

One of the biggest obstacles we initially felt we had to overcome was the transferring of vertex data from the server to the user. Since each vertex has three floats, x, y, and z, a cloth of 40 by 40 particles requires 4800 floats per time step. This estimates to roughly 14 kilobytes being transferred per time step. Not too expensive for high bandwidth areas in theory, and this proved to be the case in practice as well. The actual results of the network measurements will be outlined in detail later in the paper. On the initial handshake, the client receives an array of vertex data, which represents the scene. This transfer is one time only, since the landscape of the scene does not change with time. Then, every time frame, the client requests a new “step” from the server. This packet contains the cloth’s current geometry, which was computed on the GPU of the back-end server.

Furthermore, something that caught us off guard was that the CPU ended up simulating and rendering the cloth almost twice as fast as the GPU, which completely went against our initial intuition. The chart below demonstrates how the two processing units performed with their initial stress tests. We then thought to put the two to a few more stress tests by adding multiple cloths to the scene. As expected, the GPU vastly out performed the CPU in this regard. We believe this catch up in performance is due to overcoming the overhead that Cuda exhibits. Every call into a kernel creates overhead, as small as it may be. Because this overhead is finally overcome when multiple cloths are added and simulated in a scene, the GPU becomes the obvious choice in amortizing the system.

We decided to launch multiple clients on one machine and compare its results to running multiple cloth in one program. The chart to the left show our results. As one can see, running multiple clients on a single machine allowed rendering on each client to be done in real-time. There was a small hiccup when four clients were run simultaneously, but we believe that may have been do to a glitch in the machine and not the actual computation. Each client was given a 40 by 40 particle cloth to display in its window. In comparison to this, we decided to run mulitiple cloth in one client. The results were not surprising. Running two 40 by 40 cloth in one client resulted in 25 frames per second, four resulted in 17 frames per second, and five resulted in 13 frames per second. Although there were seven clients running at approximately 35 FPS, our computer was only simulating one cloth. When we are able to have multiple servers running on various computers, we will see the true advantage of amortized graphics.

Although we have implemented the front end to back end communication of the cloth, there are still several goals in which we would like to implement. I mentioned earlier that during the initial handshake, we transmit vertex object data. As of right now, that part of the handshake does not work completely. We hope to have the clients traverse the same scene at the same time, with multiple flags being simulated and displayed simultaneously, but the first step in this will be transmitting the object data. Through our measurments, we have found that the simlulation of multiple flags on the GPU is not feasible. Not only do the flags dip below real-time display, the amount of data for this to be accomplished on one GPU will not fit. As a result, our next major change to the project will be to split the numerous simulation up to multiple machines or GPU’s. By extending the back-end of the simulation, we predict that we will also be able to increase the number of clients that can log into it. Finally, and probably least important, we plan on texturing the entire scene as well as adding lighting along with normals. This step is more for ascetics, but a step that we would like to see accomplished nonetheless. Our only worry about this is the latency in which it may take to send the texture from the server to the client. Even though this is a one time transfer, backup may occur when multiple users are tyring to enter the system, each requiring the texture data to accurately view the scene.

In conclusion, we have found our amortization program to be very successful to this point. The algorithm chosen demonstrates the raw power of splitting computational duties to specific machines behaved precisely as expected, proving that amortization is the way to go for simulations.