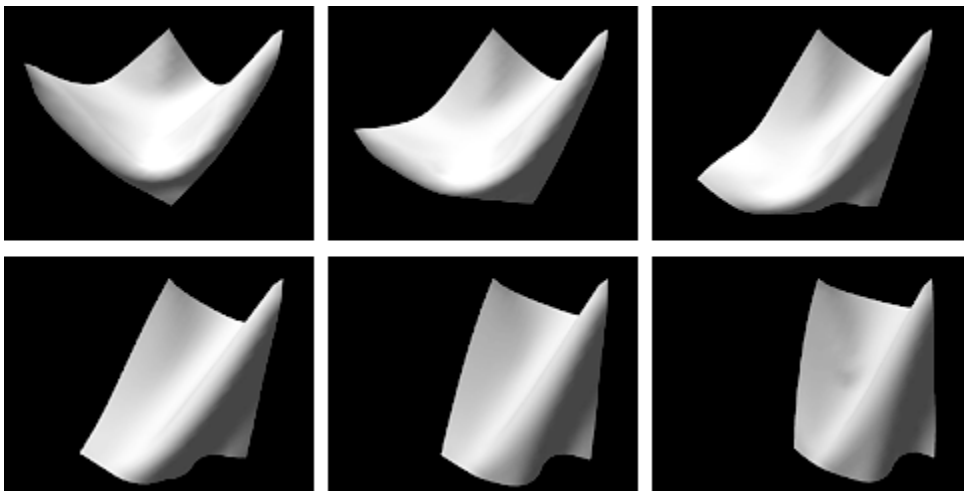
6.837: Computer Graphics Fall 2010

Programming Assignment 3: Physical Simulation

Due November 3rd at 8:00pm.

Physical simulation is used in movies and video games to animate a variety of phenomena: explosions, car crashes, water, cloth, and so on. Such animations are very difficult to keyframe, but relatively easy to simulate given the physical laws that govern their motion. In this assignment, you will be using springs to build a visually appealing simulation of cloth, as shown below.



The remainder of this document is organized as follows:

1 Getting Started

To get started, take a look at the sample solution for a demonstration of what you'll be implementing.

Run the `a3soln` file with no parameters and watch as the "cloth" falls. You may reset the simulation by pressing `r`, or you can flap the cloth around by pressing `s`. If you wish to toggle the wireframe view, press `w`. Your task will be to build a similar simulation.

The other files in this directory provide a simple skeleton OpenGL application. In addition, it implements a basic particle system simulation. Compile the code with `make` and run it with `a3`.

For this assignment, you will have to put some thought into the design of your solution. Thus, you should spend some time thinking about what sorts of functions or classes to write before you begin.

2 Summary of Requirements

This section summarizes the core requirements of this assignment. Implementation advice will appear later in this document. Again, you do not need to use the provided starter code (in fact, the provided starter code won't be very helpful except for OpenGL setup and camera control).

You are to implement a particle system that supports three types of forces: gravity, viscous drag, and springs. Each of these forces will be necessary to create your cloth simulation.

You should also implement two numerical methods for solving ordinary differential equations: Euler and Runge-Kutta. These are described later in this document. The command line of your application should allow the user to choose the solver and stepsize (like the sample solution).

Using springs, you are to assemble a piece of cloth. It should be at least an eight-by-eight grid of particles. The types of springs you should add are described later in this document.

Your application should display a wireframe animation of the cloth. You will receive extra credit for implementing smooth-shading similar to the one shown in the sample solution.

Your application should also allow the user to move the cloth in some way. It can be as simple as a keystroke that toggles a predefined motion, as implemented in the sample solution.

You should provide an executable called `a3` that takes two parameters. The first should be a character, either `e` or `r`, that selects the solver (Euler or Runge-Kutta). The second should be the stepsize used by the solver.

3 Physical Simulation Review

The core component of particle system simulations are forces. Suppose we are given a particle's position $\mathbf{x}_i$, velocity $\mathbf{x}_i'$, and mass m_i. We can then express forces such as gravity:

$$\mathbf{F}(\mathbf{x}_i, \mathbf{x}_i', m_i) = m_i \mathbf{g}$$

Or perhaps *viscous drag* (given a drag constant k):

$$\mathbf{F}(\mathbf{x}_i, \mathbf{x}_i', m_i) = -k\mathbf{x}_i'$$

We can also express forces that involve other particles as well. For instance, if we connected particles i and j with an with an undamped spring of rest length r and spring constant k, it would yield a force of:

$$\mathbf{F}(\mathbf{x}_i, \mathbf{x}_i', m_i) = -k(||\mathbf{d}|| - r)\frac{\mathbf{d}}{||\mathbf{d}||}, \text{ where } \mathbf{d} = \mathbf{x}_i - \mathbf{x}_j.$$

Summing over all forces yields the net force, and dividing the net force by the mass gives the acceleration $\mathbf{x}_i''$. Thus, the motion of all the particles can be described in terms of a second-order ordinary differential equation:

$$\mathbf{x}'' = \mathbf{f}(\mathbf{x}, \mathbf{x}')$$

In this expression, $\mathbf{x}$ describes the positions of all the particles ($\mathbf{x}$ has $3n$ elements, where n is the number of particles). The function $\mathbf{f}$ sums over all forces and divides by the masses of the particles.

The typical way to solve this equation numerically is by transforming it into a first-order ordinary differential equation. We do this by introducing a variable $\mathbf{v} = \mathbf{x}'$. This yields the following:

$$\begin{bmatrix} \mathbf{x}' \\ \mathbf{v}' \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \mathbf{f}(\mathbf{x}, \mathbf{v}) \end{bmatrix}$$

For notational convenience, we rewrite this equation as:

$$\mathbf{y}' = \mathbf{g}(\mathbf{y})$$

where $\mathbf{y} = [\mathbf{x}\ \mathbf{v}]^T$ and $\mathbf{g}(\mathbf{y}) = [\mathbf{v}\ \mathbf{f}(\mathbf{x}, \mathbf{v})]^T$. Doing so allows us to use a variety of existing techniques for solving such equations. The simplest method is the explicit *Euler method*:

$$\mathbf{y}(t + h) = \mathbf{y}(t) + h\ \mathbf{g}(\mathbf{y}(t))$$

This method, while easy to implement, is very unstable for all but the simplest particle systems. As a result, one must use very small step sizes (h) to achieve reasonable results.

There are numerous other methods that provide greater accuracy and stability. One such method is the fourth-order *Runge-Kutta* method, which (at a very high level) works by taking multiple samples of the derivative at various points and computing the final result as a linear combination. It can be stated as follows:

$$\begin{aligned}
\mathbf{k}_1 &= h\ \mathbf{g}(\mathbf{y}(t)) \\
\mathbf{k}_2 &= h\ \mathbf{g}(\mathbf{y}(t) + \mathbf{k}_1/2) \\
\mathbf{k}_3 &= h\ \mathbf{g}(\mathbf{y}(t) + \mathbf{k}_2/2) \\
\mathbf{k}_4 &= h\ \mathbf{g}(\mathbf{y}(t) + \mathbf{k}_3)
\end{aligned}$$

$$\mathbf{y}(t+h) = \mathbf{y}(t) + \mathbf{k}_1/6 + \mathbf{k}_2/3 + \mathbf{k}_3/3 + \mathbf{k}_4/6$$

The reasoning behind the strange numbers is beyond the scope of this discussion, but suffice it to say that they are derived from Taylor series expansions. For now, for this assignment, it is sufficient to know that both methods can be used to solve the differential equation by stepping forward in time in increments of the step size h.

4 Particle System Cloth

The previous section describes how to simulate a collection of particles that are affected by gravity, drag, and springs. In this section, we describe how these forces can be combined to yield a reasonable (but not necessarily accurate) model of cloth.

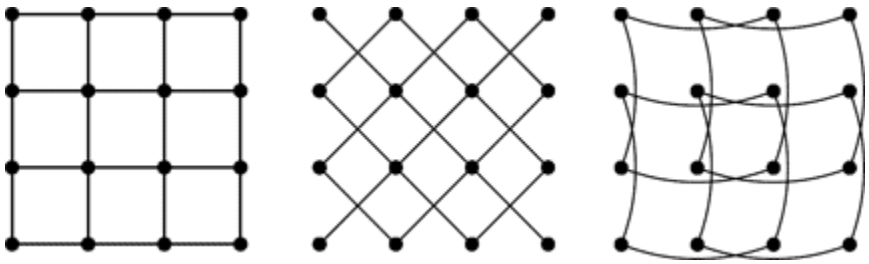


Figure 1: Left to right: structural springs, shear springs, and flex springs

We begin with a uniform grid of particles and connect them to their vertical and horizonal neighbors with springs. These springs keep the particle mesh together and are known as *structural springs*. Then, we add additional *shear springs* to prevent the cloth from collapsing diagonally. Finally, we add *flexion springs* to prevent the cloth from folding over onto itself. Notice that the flex springs are only drawn as curves to make it clear that they skip a particle—they are still "straight" springs with the same force equation given earlier.

While these springs keep the cloth in a reasonable shape, they do not alone give reasonable motion. Without some sort of dampening force, the springs will theoretically bounce around forever without losing energy.

We know this is not what happens in real life: cloth, when moved, will settle down quickly to a motionless state.

There are many ways to emulate this behavior, but for this assignment you should just use viscous drag to slow down the particles. In addition to making the motion more realistic, the drag force also counters the inherent instability of explicit integrators.

To display your cloth, the simplest approach is to draw the structural springs. For extra credit, you can draw it as a smooth surface like the sample solution, but this is not required. If you do choose to draw the cloth as a smooth surface, you'll need to figure out the normal for the cloth at each point.

5 Implementation Notes

This is a challenging assignment that requires you to be a good code designer and tester. To ensure partial credit, we suggest the following steps.

First, think about how you are going to implement the Runge-Kutta method. It requires that you evaluate the forces multiple times per step at different points. So, you should write a function that evaluates all forces given *any* values of particle positions and velocities.

After you implement the Runge-Kutta method, test this on something similar to the particle shower in the starter code. You should achieve roughly the same results.

Then, implement the gravity force, viscous drag force, and spring force. Try a single particle connected to a fixed point by a spring (basically, a pendulum). You should make sure that the motion of this particle appears to be correct. Note that, especially with the Euler method, you will need to provide a reasonable amount of drag, or the system will explode. The Runge-Kutta method should be much more stable, but you will still want a little viscous drag to keep the motion in check. If you are able to demonstrate this simple example, you will receive **60%** of the available points.

The next step is to extend your test to multiple particles. Try connecting maybe four particles with springs to form a chain, and fix one of the endpoints (you can fix a particle by zeroing the net force applied to it). Make sure that you can simulate the motion of this chain. As a general rule, more particles and springs will lead to more instability, so you will have to choose your parameters (spring constants, drag coefficients, step sizes) carefully to avoid explosions.

If you have reached this point successfully, you will have earned **75%** of the available points on the assignment. Take a snapshot of your code, just in case the full cloth implementation does not work out. We recommending using Git for version control (available on Athena). Although there is a bit of a learning curve to using a version control system, having a safety net is more than worth it.

Finally, you're ready to implement cloth. If you have designed your code reasonably well, it shouldn't be too tough to add the necessary springs. Make sure that you use reasonable rest lengths, and start small. You may want to draw your springs to make sure you're adding all the right ones in. Write your code very carefully here; it is easy to make mistakes and connect springs to particles that don't exist.

Don't be too discouraged if your first test looks terrible, or blows up because of instability. At this point, your Euler solver will be useless for all but the smallest stepsizes, and you should be using the Runge-Kutta solver almost exclusively.

If you manage to have a moving wireframe cloth, then you're at **90%**. All that's left is to add the necessary user interface elements, such rendering the cloth, moving it around, and so on. And that's **100%**.

You may also find these notes from the SIGGRAPH 2001 course on physically based modeling by Andrew Witkin and David Baraff helpful.

6 Extra Credit

The list of extra credits below is a short list of possibilities. In general, visual simulation techniques draw from numerous engineering disciplines and benefit from a wide variety of techniques in numerical analysis. Please feel free to experiment with ideas that are not listed below.

6.1 Easy

- Add a random wind force to your cloth simulation that emulates a gentle breeze. You should be able to toggle it on and off using a key.
- Rather than display the cloth as a wireframe mesh, implement smooth shading. The most challenging part of this is defining surface normals at each vertex, which you should approximate using the positions of adjacent particles.
- Implement a different object using the same techniques. For example, by extending the particle mesh to a three-dimensional grid, you might create wobbly gelatin. If you choose to implement this, please provide a different executable.
- Provide a mouse-based interface for users to interact with the cloth. You may, for instance, allow the user to click on certain parts of the cloth and drag parts around.
- Implement frictionless collisions of cloth with a simple primitive such as a sphere. This is simpler than it may sound at first: just check whether a particle is "inside" the sphere; if so, just project the point back to the surface.

6.2 Medium

- Implement an adaptive solver scheme (look up adaptive Runge-Kutta-Felhberg techniques or check out the MATLAB `ode45` function).
- Implement an implicit integration scheme. Such techniques allow much greater stability for stiff systems of differential equations, such as the ones that arise from cloth simulation. An implicit Euler integration technique, for instance, is just as *inaccurate* as the explicit one that you will implement. However, the inaccuracy tends to bias the solution towards stable solutions, thus allowing far greater step sizes. The sample solution demonstrates such a technique, which can be activated with `i` on the command line.
- Extend your particle system to support constraints, as described in this document. This extra credit is actually an assignment in 6.839.

6.3 Hard

- Implement a more robust model of cloth, as described in this paper.
- Simulate rigid-body dynamics, deformable models, or fluids. In theory, particle systems can be used to achieve similar effects. However, greater accuracy and efficiency can be achieved through more complex physical and mathematical models.

7 Submission Instructions

You are to write a `README.txt` that answers the following questions:

- How do you compile and run your code? Provide instructions for Athena Linux. If your executable requires certain parameters to work well, make sure that these are specified.
- Did you collaborate with anyone in the class? If so, let us know who you talked to and what sort of help you gave or received.
- Were there any references (books, papers, websites, etc.) that you found particularly helpful for completing your assignment? Please provide a list.
- Are there any known problems with your code? If so, please provide a list and, if possible, describe what you think the cause is and how you might fix them if you had more time or motivation. This is very important, as we're much more likely to assign partial credit if you help us understand what's going on.
- Did you do any of the extra credit? If so, let us know how to use the additional features. If there was a substantial amount of work involved, describe what how you did it.
- Got any comments about this assignment that you'd like to share?

As with the previous assignment, you should create a single archive (`.tar.gz` or `.zip`) containing:

- All source code necessary to compile your assignment.
- A compiled executable named `a3`.
- The `README.txt` file.

Submit it online using the Stellar website by **November 3rd @ 8:00pm**.

This assignment does not require the submission of an artifact.