DEFORMABLE OBJECTS USING FAST LATTICE SHAPE MATCHING

MARWAN KALLAL

1. Introduction

I attempt to use the results of Rivers and James [Rivers and James(2007)] on Fast Lattice Shape Matching (FLSM) to make our already fuzzy bunny deformable as well. I have omitted collision and other physics in the interest of time and am focusing on deformations upon movements of single particle masses. I will also be using the unoptimized version SLSM (Slow LSM), to focus on the dynamics of these deformations.

2. Explanation of FLSM

2.1. **Constructing the Lattice.** The first step to FLSM is to build a lattice, or grid, that encloses the mesh. We start by creating a bounding box around the mesh. From here we round the dimensions of the bounding box up to fit an integral number of grid squares. Each grid square has an associated particle which will be used for movement and other calculations.

2.2. **Creating Particles.** From here we need to check which grid squares are actually in contact with the inside of the mesh. We can do this using a standard triangle mesh voxelization algorithm [Rosen(2009)]. Each particle will have a mass associated with it to properly simulate physical responses. To differentiate between inner and outer particles, we can set the particle mass of outside particles to 0.

2.2.1. *Assigning Vertices.* Now that we have our grid and particles, we can assign vertices to their associated particles. This allows us to move the vertices of the mesh as the particles move, essentially sharing the transform of the particle with that vertex. To do this, we can iterate through the vertices of the mesh, and calculate which grid square they lie in. We can then assign that vertex to the particle in that grid square.

2.3. **Creating Regions.** Shape matching regions make up the building blocks of the deformation system. We start by specifying a region half-width w. This region size will also change the dynamic properties of the mesh, with larger regions making the mesh stiffer. This is because the way that region center of mass is calculated, and therefore the final rotations of the regions will be smaller with the displaced particles having less weight relative to the total. Using masses of 0 for particles outside the mesh is important for making sure that their potential movement doesn't affect the deformation of the object. The regions will also overlap to make sure that the object doesn't flop around and fall apart. We can make a region centered around every particle as long as no part of the region exceeds the range of the bounding box (and thus the lattice we constructed).

2.4. **Dynamics.**

2.4.1. *Particle Movement.* Particles are affected by outside forces such as gravity and collisions. This creates movement among the particles, which will then move regions, affecting the final resting position of the particles for that timestep.

Date: December 17, 2017.

2.4.2. *Region Movement.* When particles move, this changes the center of mass of the regions. In order to move the region properly and assign goal positions, we need to know both the original center of mass of the region $\mathbf{c}_r^0$ and the newly deformed one $\mathbf{c}_r$. We can calculate them both for each region using:

$$\frac{1}{RegionTotalMass} \sum_{i \in RegionParticles} \tilde{m}_i \mathbf{x}_i$$

Now we must find a way to rotate and translate the regions in a way that preserves the shape of the particles within the region as much as possible.[Rivers and James(2007)] proposes an approximation of the least squares rotation given the position of the deformed particles relative to the original region. This method uses an intermediate matrix value $\mathbf{A_r}$ (shown below) and then a polar decomposition to extract the rotational component.

$$\mathbf{A_r} = \sum_{i \in Region} \tilde{m}_i (\mathbf{x}_i - \mathbf{c}_r)(\mathbf{x}_i^0 - \mathbf{c}_r^0)^\top$$

From here we can take the polar decomposition of $\mathbf{A_r}$ to obtain the rotational component, $\mathbf{R_r}$. To get the final transformation we need for the region, we use the two centers of mass we found earlier, $\mathbf{c}_r^0$ and $\mathbf{c}_r$, and our newly obtained $\mathbf{A_r}$, as shown below:

$$\mathbf{T}_r = [\mathbf{R}_r(\mathbf{c_r} - \mathbf{R}_r \mathbf{c}_r^0)]$$

2.4.3. *Setting Particle Positions.* Because we know the regions which each particle belongs to, we can place a goal position $\mathbf{g}$ using the average of the transformations for the regions containing it. We can then use the relative distance between original and goal positions of the particles, as well as the outside forces mentioned above, to find the velocities and then final positions of the particles as shown below.

$$\mathbf{v}(t+h) = \mathbf{v}(t) + \frac{\mathbf{g}(t) - \mathbf{x}(t)}{h} + h\frac{\mathbf{Forces}(t)}{m}$$

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\mathbf{v}(t+h)$$

Now that we have the final positions of the particles, we can move the associated vertices with the same transformations as the particles to make the mesh itself deform realistically.

3. What I Learned

I learned quite a bit about graphics and deformable objects from this projects. Having worked with the bunny and the dynamic fur before this, I had an idea of how the dynamics might work, with spring like coefficients and the like. I had originally thought of doing a 3 dimensional grid of springs and point masses, like some of the cloth dynamics demos, but wanted to explore a deeper problem. I read through some other papers as well, such as [James and Pai(1999)] which described much more complicated, but realistic, ways of deforming objects with volume preservation and many tunable parameters. The problem was that I couldn't fully understand the math they were using, so I moved on.

I finally found the [Rivers and James(2007)] paper, which seemed to be much more in my reach mathematically. I first set out to understand the basic building blocks of the problem. It seems similar at first glance to the spring problem, with particles trying to stay within their regions, but it gives you more tunability for floppiness and area of effect when deformed than a straight up spring lattice will. One of the more interesting parts of the paper was the fast summation they used to precompute and reuse as many values as possible, and is why the paper is called FLSM. I will cover the way this works in Section 3. This was also good to learn, showing a way that we can significantly reduce processor time when doing summations by finding ways to reuse previously calculated values.

4. Optimizations

There are many potential optimizations we can make, many of them included in the [Rivers and James(2007)] paper.

4.1. Fast Summation.

The main breakthrough of the paper was the introduction of a fast summation algorithm that stores as much information as possible to be reused in future computations.

5. My Implementation

I began by using the bunny mesh we had been using before. I generated a bounding box and rounded it up to the nearest grid square so that we could have integral particles. You can see this process in `lattice.h::get_lattice_bounds()`. I then assigned each grid square a particle and to that particle a set of vertices to share their transforms with. This is done in `lattice.h::create_lattice()` which then calls `Particle::setPosition()`. Next I started to assign regions. For simplicity I ignored the mesh when constructing particles and regions, and just used the expanded bounding box as our object. I created regions with half-width $\mathbf{w} = 1$, as shown in `Region()`. All of the necessary values have been set in the particles. All that's left now is to find the centers of mass of the new regions, use the least squares rotation of the regions and apply those transforms to the particles and associated vertices.

5.1. State of Affairs.

The code is not currently fully functional. I have created what I stated in Section 5, and have some comments detailing the approach for the rest in the code, marked with `//TODO:`. The most important pieces of pseudocode can be found in `lattice::deform_timestep()`, detailing the procedures to take and values to set in order to deform the bunny.

References

[James and Pai(1999)] Doug James and Dinesh Pai. Artdefo: accurate real time deformable objects. In *Proceedings of the 26th annual conference on computer graphics and interactive techniques*, SIGGRAPH '99, pages 65–72. ACM Press/Addison-Wesley Publishing Co, July 1999. ISBN 0201485605.

[Rivers and James(2007)] Alec R. Rivers and Doug L. James. FastLSM: Fast Lattice Shape Matching for Robust Real-Time Deformation. *ACM Trans. Graph.*, 26(3), July 2007. ISSN 0730-0301. doi: 10.1145/1276377.1276480. URL `http://doi.acm.org/10.1145/1276377.1276480`.

[Rosen(2009)] David Rosen. Triangle Mesh Voxelization. `http://blog.wolfire.com/2009/11/Triangle-mesh-voxelization`, 2009.