1 Real-Time Spring Mesh Deformations

Ulises Amaya **Victor Zordan**

University of California Riverside, Riverside Graphics Lab

Universidad Autonoma del Estado de Morelos, Facultad de Ciencias

Abstract. Animating deformations on solid objects is a very interesting and most useful research area in computer graphics. Research has been done in this area by approaching the problem both via linear strain measures and non-linear strain tensors, all having advantages and disadvantages. In the first approach the problem arises with large deformations, as it yields a unnatural growth in volume when computing large deformations. Problems are related to the computation of rotations. The advantage is that the computations are not costly as they are linear. In the second approach, the problem arises because of the complexity of the computations, and thus, makes it not possible to compute it in real-time. The advantage is that it is much more accurate specially in large deformations when compared with the linear approach. The objective of this work is to be able to simulate solid object deformations in real-time as accurate as possible but keeping the calculations as simple as possible. We are basing this project on the work presented by M.Müller et. al. on Stable Real-Time Deformations. The goal of this project is to set up a 3-D animated spring mesh.

Keywords: Physically Based Animation, Stiffness, Damping, Elasticity.

1.0.1 1 Introduction

This work is a physically based dynamic simulation of a deformable solid object. In the computer graphics world people want to model the real world as accurate as possible but having in mind that the computations need to be efficient. This is not as easy as sometimes it seems. Usually there is a trade-off between the accuracy and the efficiency of the computations. There has been a lot of research in how to reduce the gap within this trade-off. In dynamic simulation we can have realistic motion and high level of control as well as specify constraints. We are very interested in learning how to set up and control these characteristics. The main objective behind this work is to become familiar, understand the difficulties, and see the different solutions that have been given to the problems of physically based dynamic simulations of deformable objects. To achieve this, we will construct our own version of a deformable spring mesh having in mind that a visual accuracy and efficiency in the computations is very important.

1.0.2 2 Constructing The Base Case

We begin by remembering the classic physics that we were once taught when we where kids.

$$F = K(X - Xo)$$

Where F is the force, K is the stiffness constant and X - Xo is the distance from the rest length of the spring.

We can use this force law in addition to the following basic equations:

$$F = m * a$$

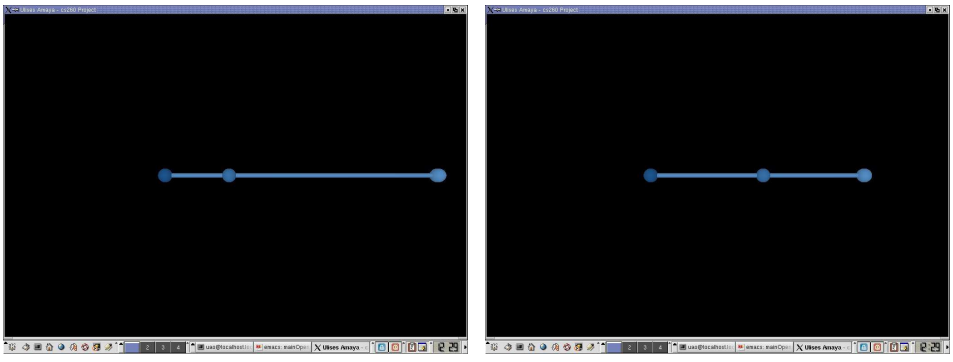
$$a = \frac{F}{m}$$

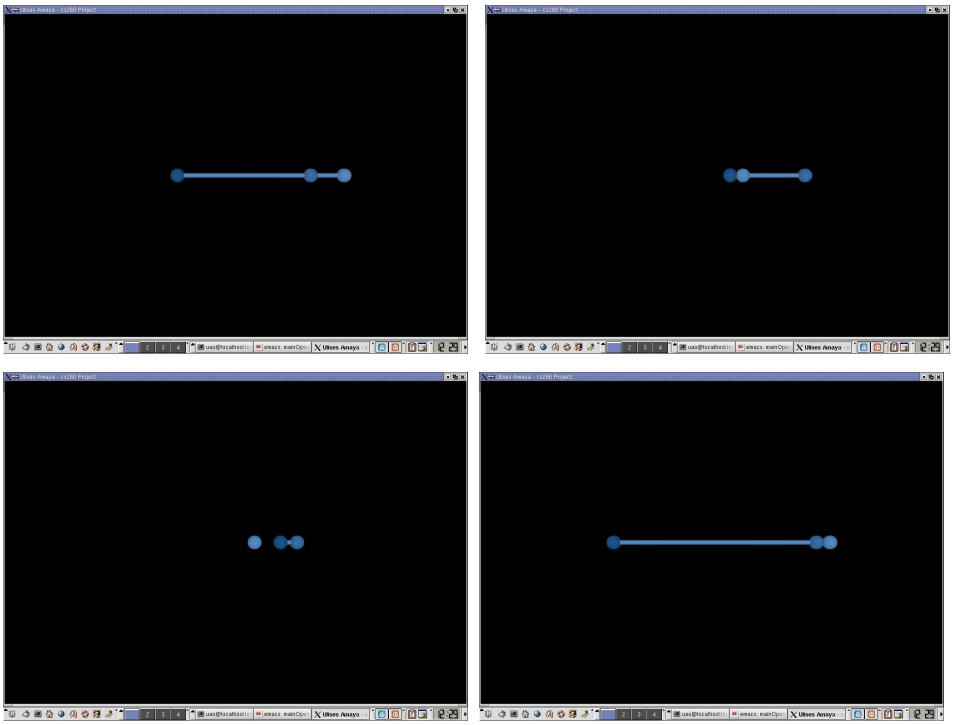
$$V_{t+1} = (a * dt) + V_t$$

$$Pos_{t+1} = (V_{t+1} * dt) + Pos_t$$

Where a is the acceleration, dt is the time interval, V is the velocity at time t and Pos is the position of the particle also at time t. So, the particles are determined by position and velocity.

This is the first approach, used for the base case which is the 1-D. This approach is only ideal since we are not taking the damping term into account. We simulate the object using 3 particles and a spring between each of them on one axis. With the force described as above we would end with an oscillating structure. Unfortunately this was not the case, the simulation showed an unexpected increase of the force in the structure as opposed to the oscillating structure we wanted. After analyzing the equations and the program we realized that there where many little problems, this is where the research starts teaching lessons and the interesting things arise.





1-D simulation with no damping. The initial state can be seen and how it turns into an unstable simulation where particles pass through one another and eventually the system blows up.

A careful analysis of the system setup and its simulation tells us the following. First the simulation has very particular initial conditions. The way it was set up is that we have a particle at location 0,0 the origin, then a particle at 0,4 and then the last one at 0,8 all in the X-axis. This sets up the system in its rest position with a rest length of 0.4cm.There is no force being applied and thus, there is no movement. To start the simulation the third particle is displaced 0.5cm and then we start counting from there, t = 0. This means that the particles 1 and 2 are at their rest position when the 3rd is not. In a natural system there would already be a force being applied to the particle 2 from 3 and thus it would not be at its rest position. Its important to note that the units used are the most common, meaning the metric system. The particles are of 1kg of weight, and the distances are in centimeters. The stiffness constant used is of 1000 units, the time step is of 0.001 seconds. The other problem is that its very difficult to simulate a spring mesh particle system accurately since problems of numeric precision arise. Every oscillation of the system involves many time steps and if there is a small numeric problem then this one would be carried along and in a few seconds it would be noticeable. There are several ways to approach this problems, one is by decreasing the time step and thus decreasing the numeric problems. Another solution is to include a damping term. This

last approach will help stabilize the system and make it more real.

It is also important to note that in this system there are no boundaries and no physical limitations, the particles can be taken as far as one wants, the springs will stretch and will not break. Of course since this is not a real system, when simulating we could end up with an unreal simulation. This first part of the code was done using C programming language.

After having the previous simulation, we decided to add the damping term to make it more real and stable. The damping term is proportional to the relative velocity and acts in the direction of the spring force. In one dimension the force term is as follows:

$$F = -K[|x2 - x1| - Xr_{12}] - B(V2 - V1)$$

After inputting the damping term in the equation we where able to see in the simulation a clear change. There was no more increase in the force of the system and the simulation behaved as a real spring mesh. The constants used where K = 1000 and B = 70 units respectively. The rest of the variables where kept with the same values as before. We could see how important it was to keep the system as close to reality as possible when simulating and reality has damping and thus we needed damping.

1.0.3 3 Constructing the 2-D case

This case was very difficult. I personally thought that it was not going to be so difficult since the base case was already working and as the name suggests, it is the case where the others are based upon. But unfortunately this did not turn out like that. We begin by analyzing the equations in two dimensions:

$$F_x = -K * (Pos2_x - Pos1_x) + B * V_x$$

$$F_y = -K * (Pos2_y - Pos1_y) + B * V_y$$

$$a_x = \frac{F_x}{m}$$

This is for the X and Y axis respectively. As we can see the equations are now more complex since they involve components of each axis for each particle and this means using cosines and sines with the angles in between. The above equations give us the force that a particle feels from its neighbor, but if the particle has more than one neighbor then the forces must be added for all neighbors. In the 2-D case we constructed a basic triangular structure keeping it as simple as possible, so each particle has two neighbors and there are a total of three particles and three springs.

There are basically two ways of dealing with the 2-D equations. One is by considering cartesian components of each physical property for each axis, this is what we have in the previous equations regarding force . In this case we work with individual components X and Y for F, V, Accel. and Pos. The other way is by considering the use of vectors. In the latter case we would need to use vector operations such as the dot product and the cross product, normalizations and sums or substractions in terms of vectors. I decided to use the first approach since it seemed closer to what I did for the 1-D case, big mistake, it turned out to be much more difficult. Also in this stage of the project, I decided to move from C to C++ programming language to make it more maintainable and easier to program and to follow. All this mixed changes and approaches resulted in a chaos that just delayed the whole simulation of the 2-D case. The first results where totally unstable, we had several problems of keeping the numeric precision, problems porting the code to the C++ language and problems of separating the two components of each physical property.

Lets first analyze the case without damping, then we will add it to make the simulation more real. The x component of the force for particle 3 due to particle 1 is:

$$f^i_{13x} = -K_{13}\left[\sqrt{(x^i_3 - x^i_1\ )^2 + (y^i_3 - y^i_1)^2} - r_{13}\right]\cos\theta^i_{13}$$

The x component of the force for particle 3 due to particle 2 is:

$$f^i_{23x} = -K_{23}\left[\sqrt{(x^i_3 - x^i_2\ )^2 + (y^i_3 - y^i_2)^2} - r_{23}\right]\cos\theta^i_{23}$$

If we interchange 2 for 3 and 3 for 2 we obtain the same result, but with the sign changed, which is exactly what we should have. For the Y coordinate we use the sineθ.

The damping force is proportional to the relative velocity on the line that joins the particles 1 and 3. The X component of the vector which has the direction from particle 1 to 3 is obtained by multiplying this component times the cosine of the angle between the x-axis and the line 1 to 3. After simplifying we obtain:

$$\frac{\left[(x_3^i-x_1^i)(v_{3x}^i-v_{1x}^i)+(y_3^i-y_1^i)(v_{3y}^i-v_{1y}^i)\right](x_3^i-x_1^i)}{(x_3^i-x_1^i)^2+(y_3^i-y_1^i)^2}$$

And so the X component of the Force in particle 3 due to particle 1 will be:

$$F_{3x}^i = f_{3x}^i - B\frac{\left[(x_3^i-x_1^i)(v_{3x}^i-v_{1x}^i)+(y_3^i-y_1^i)(v_{3y}^i-v_{1y}^i)\right](x_3^i-x_1^i)}{(x_3^i-x_1^i)^2+(y_3^i-y_1^i)^2}$$

And so consequently the velocity with the damping term looks like this:

$$V_{3x}^{i+1} = v_{3x}^i + \frac{dt}{m}\left(f_{13x}^i - B\frac{\left[(x_3^i-x_1^i)(v_{3x}^i-v_{1x}^i)+(y_3^i-y_1^i)(v_{3y}^i-v_{1y}^i)\right](x_3^i-x_1^i)}{(x_3^i-x_1^i)^2+(y_3^i-y_1^i)^2} + f_{23x}^i - B\frac{\left[(x_3^i-x_2^i)(v_{3x}^i-v_{2x}^i)+(y_3^i-y_2^i)(v_{3y}^i-v_{2y}^i)\right](x_3^i-x_2^i)}{(x_3^i-x_2^i)^2+(y_3^i-y_2^i)^2}\right)$$

Where *i* is the time-step and *i+1* is the new time-step. We are only left to get the position of the particle so that we can draw it on the screen which is given by:

$$x_3^{i+1} = x_3^i + dt\frac{v_{3x}^i+v_{3x}^{i+1}}{2}$$

Now, the true reason why we are taking the average of the velocities is because when we did the first calculations in the two dimensional simulations it was not stable, after carefully reviewing the equations and analyzing the simulation we realized that one possible error could come from the fact that if we only take one velocity through the whole time step then we are introducing an error since in reality this velocity will vary and we are taking only one velocity at one given moment of time through a length of time. By taking the average between the initial and final velocities in the time step, we reduce this error significantly. We realized that this error was also present in the one dimensional simulation but it became bigger and more noticeable when all particles where having two neighbors and the arrangement was more complex.

At this moment of the research we realized how complex it was getting and that moving into the third dimension will be very challenging. Dr. Zordan's wisdom and experience pointed towards using vectors instead of our current approach of Cartesian coordinates. So we decided to make the 2-D case work with vectors and then move to the third dimension. It was time to take a look to our old elementary-first grade physics notes.

3.1 2-D Case with Vectors Lets remember how the vectors are used and how they work by explaining the equations we used before in terms of vectors. The position vector of a particle p at time i is defined by:

$$r_p^i = x_p^i \hat{\imath} + y_p^i j$$

Where the vectors i and j are the unit vectors (of magnitude 1). Given this, we can see that the force on particle 3 due to particle 1 is:

$$f_3^i = -K_{13} \left[|r_3^i - r_1^i| - r_{13} \right] u_{13}^i$$

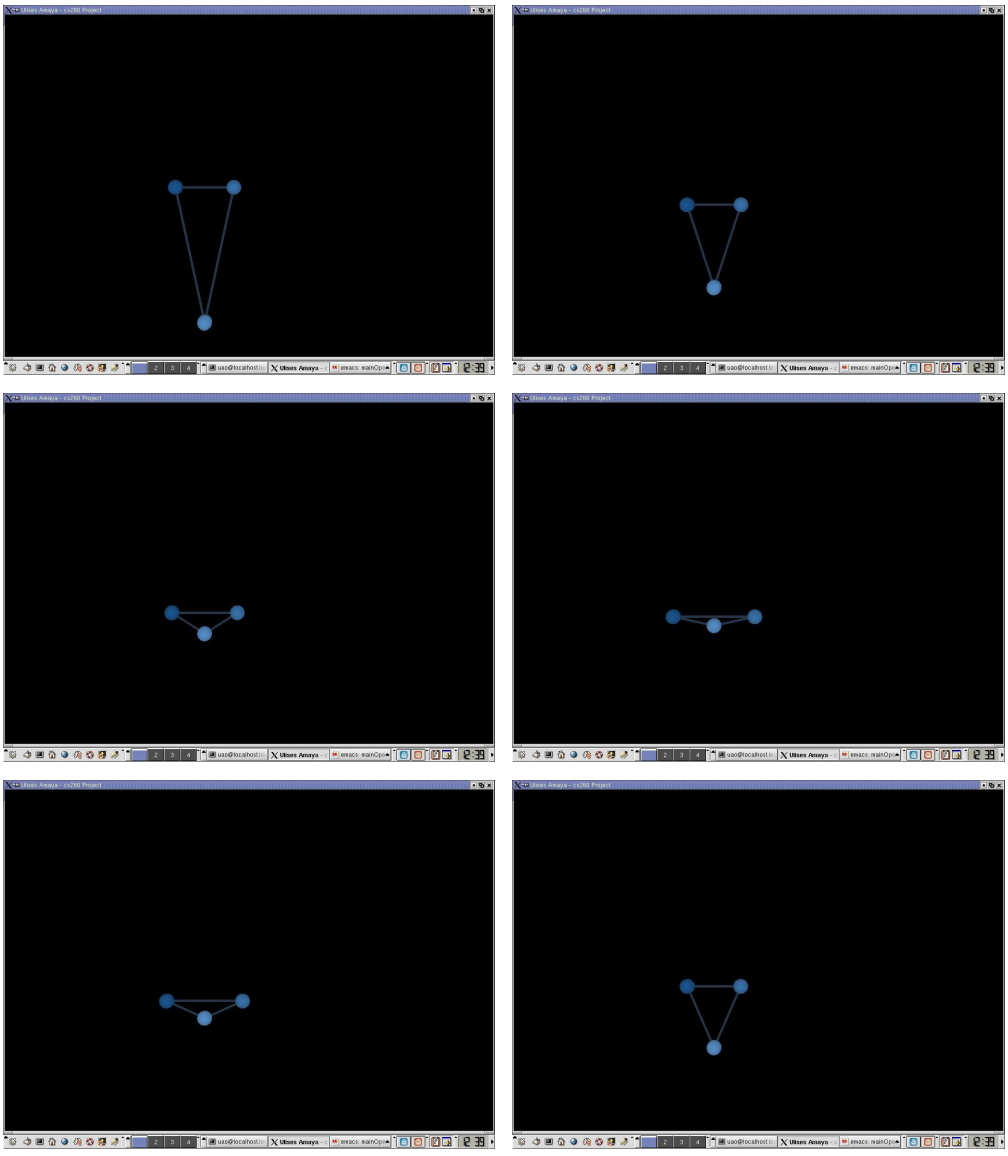
$u_{13}^i$is a unit vector in the directionsf rom particle1 to particle 3. As we mentioned before, the damping is proportional to the component of the relative velocity along the line that joins the two particles. This relative velocity can be written as:

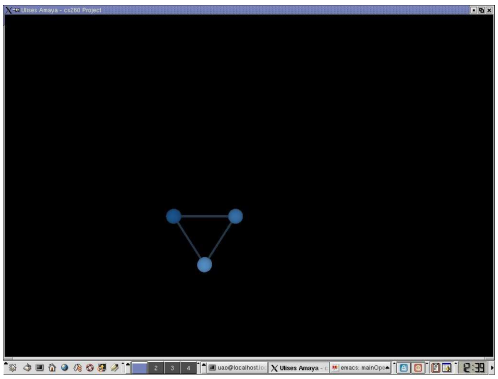
$$\mathbf{V}r_{13}^i = \mathbf{v}_3^i - \mathbf{v}_1^i$$

And thus, the component of the velocity along the line that joins the two particles will be the scalar product:

$$u_{13}^i \cdot Vr_{13}^i$$

Coding this was much easier but longer since we needed to define in the program how to compute scalar products, length of a vector, normalizations, substraction, addition and multiplication of vectors and how to get a unitarian vector. Once having this working, coding the above equations was straight forward, then performing the simulation was a nice and very enjoyable success.





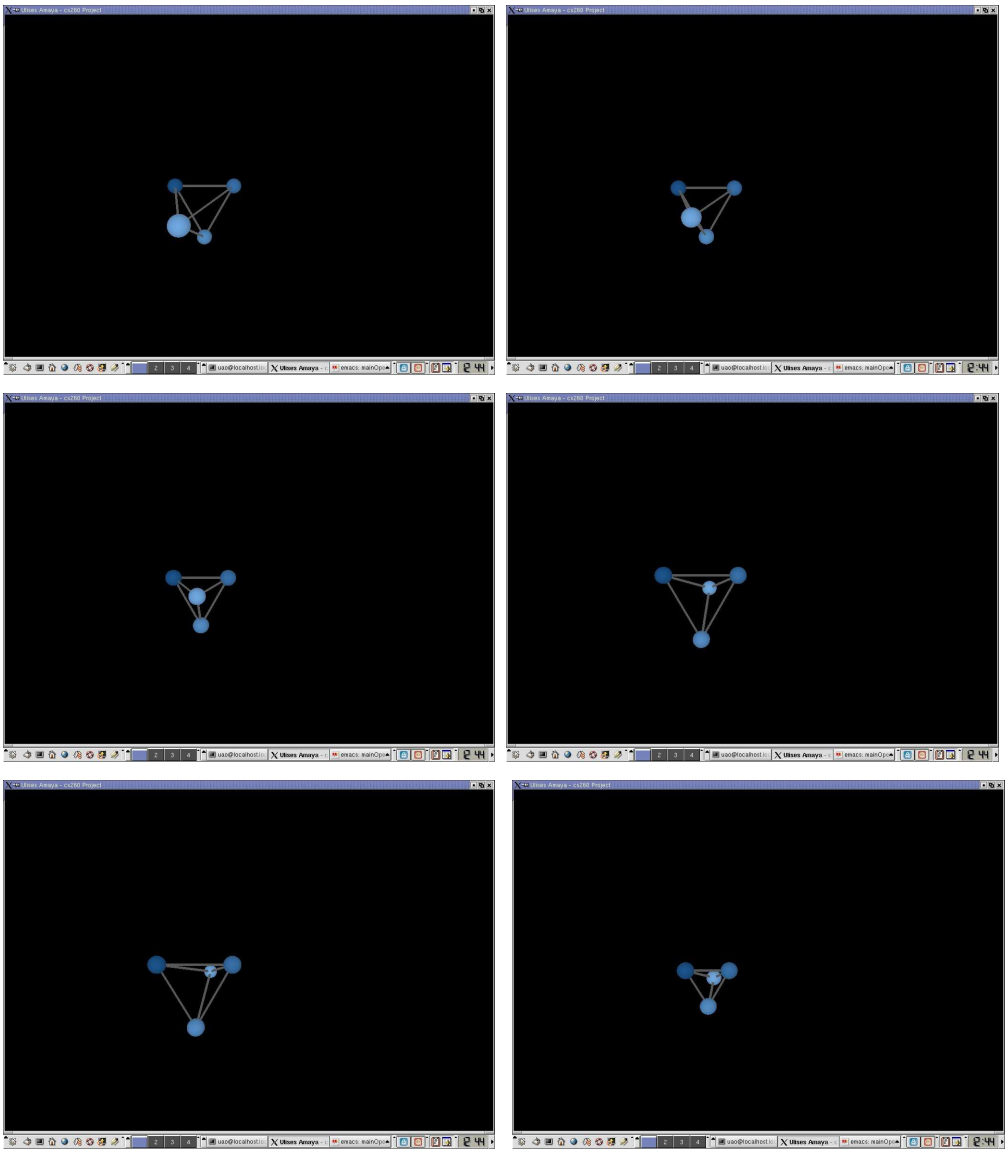
2-D case with damping of 20, we can see a real behavior in the triangular spring mesh as it ends in its rest position.

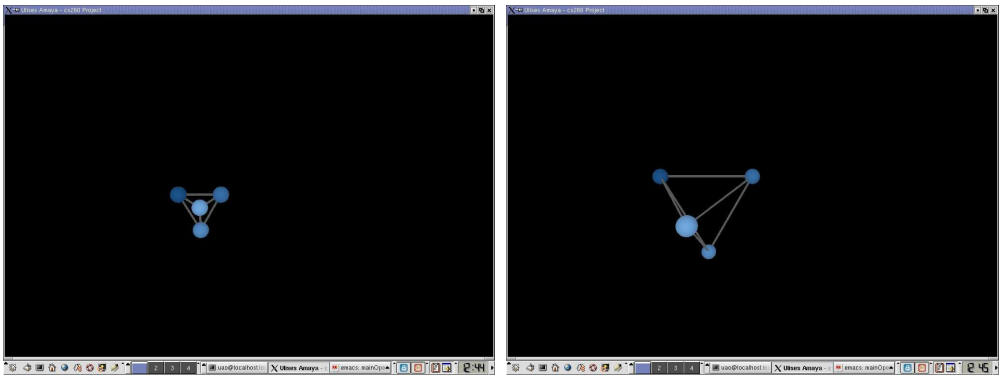
1.0.4 4 Constructing the 3-D case

This case was much easier having the 2-D case working with vectors, as we only needed to add the third coordinate to our vectors in the calculations (the z axis in other words) and thus we have the position vector as follows:

$$r_p^i = x_p^i \hat{\imath} + y_p^i j + z_p^i k$$

Of course the initializations and number of particles would change, for this case we now have a basic construction of a pyramid with a triangular base, so the number of particles increases as well as the number of springs and neighbors to each particle and thus the calculations. It would have been much more complicated using the previous method to do the computations as opposed to the vector approach, and this is because now we have 3 axis, thus 3 Cartesian coordinates and two different angles to work with. It was difficult already in two dimensions.





3-D case with 20 of damping. The pyramid construction shows real behavior and a stable simulation

5 The Algorithm The implemented simulation algorithm is the following:

Force Vectors [i] = 0.0 for i = 1...n

Base Vectors are (1,0,0) (0,1,0) (0,0,1)
loop on all particles

Calculate Force due to every neighboring particle
Sum the Forces
Calculate Velocity
Calculate Position
Draw Particle in space

end loop

1.0.5 6 Applications

This area of research is very useful for animating any solid object deformations, it can be used to model cloth, skin or other physical objects. This program can be easily changed to be able to simulate a solid object with different properties in it by changing the stiffness and damping along the mesh as well as the rest length and the mass of each particle. We can also simulate different forces being applied to the spring mesh. This research focuses on the use of classic physics and keeps the computations as simple as possible in order to achieve results in real time.

1.0.6 7 Conclusions

This research was very useful to make myself realize of the complexities and challenges that arise in real world research. The area is fascinating and the possibilities are very big. I also realized how important research is for life in general as its accomplishments can be used in all areas of science even if they are not meant for that. Research involves not only knowledge of one particular area but rather a conjunction of many, and experience counts as a great ally if you have it. For this particular research it is important to note that there are big efforts to make these simulations in real time. This involves that the computations should not be mathematically too complex. There is always a trade off between accuracy of the simulation and the speed. Accurate results have been achieved using partial differential equations that describe the static and dynamic behavior or the object but this is not real time as the computers take several minutes to get the results. As stated previously this research was done keeping computations simple and thus achieving results in real time but unfortunately not predicting exact results.

1.0.7 8 References

Müller, M., Dorsey, J., Leonard, M., Jagnow, R. Cutler, B. Stable Real Time Deformations 2000. MIT.

Terzopolus, D., And Witkin , A. 1998. Physically based models with rigid and deformable components. IEEE *Computer Graphics & Applications.* Oxford Univ. Presss, NY.

Terzopolus, D. Platt, J., Barr, A. And Fleisher, K. 1987 Elastically deformable models. In *Computer Graphics Proceedings*, Annual Conference Series, ACM SIGGRAPH 87, 205-214.

1.0.8 SPECIAL THANKS

To my father:

Dr. Alejandro Amaya

for his continuous support

and contributions to this work.

To My family:
for all their
support and love
throughout my
entire life.

To Monica:
For her love
courage and
all things we share.

To Dr. V. Zordan:
For his patience that
was taken to the limits
and his guidance through
the course of this work.

To My friends:
In Mexico, Switzerland,
and the U.S.
For their inspiration
and courage given
at all times.