# Semi-Realistic Balloon Simulation

Paul Tarantino*
University of California, Santa Cruz

## Abstract

This paper describes a technique to accurately simulate a balloon expanding and bursting. Physical simulation is achieved through use of a spring-damper model with point masses to represent the surface of the balloon. Visual characteristics of the balloon, such as variable opacity and variable shininess are also addressed by this approach using material attributes in the SGI lighting model. Spring constants are user-definable as a function of height and internal pressure of the balloon is also controlled by the user. Bursting is achieved by using spring thresholds which allow for fractures in the surface and lead to a chain reaction of spring failures. Balloon models can be created in a 3D sculpting program by an isosurface algorithm and can be used by the simulation.

By using physically based modeling techniques, the problem can be simplified and easily implemented using a few simple data structures. To maintain simplicity and minimize elapsed time between frames, collision detection will not be implemented in this model. By using SGI graphics hardware to render the balloon, interactive rates can be achieved for simple balloon models, however, more complex balloons will result in longer delays between frames.

**Keywords:** computer animation, balloons, physically based modeling, simulation, natural phenomena.

## 1 Introduction

Physically-based models of natural phenomena have been a major topic in computer animation for the past ten years. Accurately representing flexible objects such as cloth, muscles, facial expressions, and liquids has been the topic of many published articles. Soft objects are difficult to simulate in computer graphics because of the complex and varying surface structure of the object.

*Computer Science Dept, Room 57AS, Santa Cruz CA 95064, USA. E-mail pault@cse.ucsc.edu

As far as I know, no one has implemented a simulation of a balloon which can be pressurized, and exploded. The balloon simulation should reflect the fact that the material is being stretched as it is pressurized. As the material is stretched, it becomes more transparent and it also becomes shinier. The simulation should also reflect the elasticity of a balloon, which may not be constant over the entire surface and it should model the event when the surface develops a tear and the balloon bursts. The bursting event should model the deformation of the surface of the balloon over time.

This animation may have several applications in studying related phenomena that may occur too quickly in real life for the human eye to see. Realistic computer animation of pressurized objects that burst may be used in medicine for simulating hemorrhages and aneurysms. Engineering applications that could benefit may include stress testing tires and underwater equipment, and testing air-tight seals on aircraft. Animation of bursting objects would also bring the entertainment industry one step closer to modeling realistic situations using computer graphics.

## 2 Background

Successful implementations of soft objects include point masses connected by springs and dampers [TF88], implicit surfaces generated from a volume of particles [DG95, WH94], and layered structures of meshes controlled by free-form deformations [CHP89]. Features including elasticity, inelasticity, viscoelasticity, and plasticity are needed to represent these different materials [TF88]. As research continues, new computer graphics effects are being developed which model new and interesting natural phenomena.

Terzopoulos [TF88] successfully implemented soft objects by using point masses connected by springs and dampers. He first creates a mesh of nodes that are controlled by using a variety of partial differential equations that model viscoelasticity or elastoplasticity (figure 1). The result is a large system of simultaneous ordinary differential equations. The second step is to

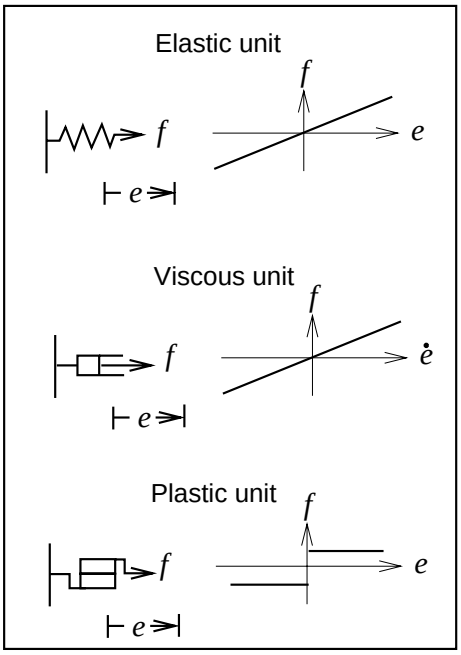


Figure 1: Uniaxial deformation units and their response to applied forces. (a) Elastic spring. (b) Viscous dashpot. (c) Plastic slip unit.

integrate the semidiscrete system through time, thus simulating the dynamics of deformable models. Models of deformable curves, surfaces, and solids are defined earlier by Terzopoulos [TPBF87] and are based on simplifications of elasticity theory.

Desbrun [DG95] also used a system of point masses, but used them inside the volume instead of on the surface. He used a technique of sending "seeds" from each particle in all directions to determine the subvolume region that each particle had. The seeds interacted with fields from other particles to determine the subvolumes as well as the surface of the soft object. Collisions between objects were done by using a bounding box around the surface seeds and testing against other objects. Conservation of volume was also attained by adjusting the subvolumes until the sum matched the initial volume of the object.

Significant work has been done on the particular topic of modeling cloth. Since cloth is used for realistic animation of humans and interior environments, both very popular animation topics, an accurate model of representing the draping properties was developed. Weil's paper on the synthesis of cloth objects [Wei86] gave a new, more realistic simulation of cloth by approximating the appearance of a piece of cloth which is suspended at certain constraint points. Other techniques of simulating cloth at that time were limited to things like texture mapping. Thalmann and others [CYMTT92, VCMT95] have done work on synthetic clothing for computer animated actors which makes use of complex collision detection algorithms. Their algorithms can simulate different kinds of cloth by controlling factors like bending strain and shearing strain. Breen, House, and Wozny [BHW94] demonstrated a physically-based technique for predicting the drape of a wide variety of woven fabrics using particle systems. By using empirical data from the *Kawabata Evaluation System* fabric measuring equipment, were able to tune the model by deriving energy functions based on the sample's nonlinear mechanical properties and then using the model to reproduce the fabric's characteristic large-scale draping behavior.

Witkin [Wit95] gives a concise formula for the binary force associated with Hook's spring law

$$\mathbf{f}_a = [k_s(|\mathbf{l}| - r) + k_d \frac{\dot{\mathbf{l}} \cdot \mathbf{l}}{|\mathbf{l}|}] \frac{\mathbf{l}}{|\mathbf{l}|}, \mathbf{f}_b = -\mathbf{f}_a$$

where $\mathbf{f}_a$ and $\mathbf{f}_b$ are the forces on $a$ and $b$, respectively, $\mathbf{l} = \mathbf{a} - \mathbf{b}$, $r$ is the rest length, $k_s$ is a spring constant, and $k_d$ is a damping constant. $\dot{\mathbf{l}}$, the time derivative of $\mathbf{l}$, is just $\mathbf{v}_a - \mathbf{v}_b$, the difference between the two particles' velocities. In this equation, the spring force magnitude is proportional to the difference between the actual length and the rest length, while the damping force magnitude is proportional to $a$ and $b$'s speed of approach.

## 3 Implementation

My approach to implementing this project is based on modeling the balloon using techniques developed by Terzopoulos [TPBF87, TF88], which are useful for modeling elastic substances. The surface is subject to an internal force representing the air pressure inside the balloon. The materials used to represent the surface are updated based on the local state of the balloon surface to reflect the shininess and opacity of stretched rubber. The idea is to make the balloon as natural looking as possible without introducing an unnecessary amount of complexity.

### 3.1 Surface Structure

The surface of the balloon is modeled by a mesh of particles. Each particle has a mass associated with it and is connected to its neighbors with springs. Each spring has a spring constant and a damping constant that provide the attraction/repulsion forces needed to represent an elastic surface. Position and velocity of each point mass are calculated by Euler integration methods. The damper can be adjusted and used to eliminate any prolonged vibrations in the surface.

The spring constants may not be the same for all springs. They can be set as a function of the height of the balloon. This gives the balloon the realistic behavior of expanding in the middle more than at the

top or bottom due to the variance in the thickness of the surface at those regions.

### 3.2 Forces

The balloon requires a force that represents the air pressure inside. This force is an internal force pushing out at each point on the surface mesh and is in the direction of the normal of each point. The normal of each point are calculated for every time step by averaging the normals of the polygons that the point is associated with. The forces on each point will be equalized, before the balloon explodes, by the springs that connect each point. The springs will have to stretch in order to compensate for an increase in air pressure.

As the balloon bursts, the air pressure will drop until it reaches an equilibrium with the environment. Simulating this pressure change correctly adds unnecessary and expensive fluid dynamic complications, so this algorithm does not address that factor.

### 3.3 Materials

As the surface of the balloon stretches, it should look shinier and it should get more translucent. The shininess effect is achieved by updating the material used for the GL lighting model. The shininess is a function of the distance that the springs are from their resting distance. This is calculated for each polygon that is rendered.

The opacity of the surface is also a function of the distance that the springs are from their resting distance. The alpha value of the material is updated with the shininess before each polygon is drawn. Calculating visibility is too complicated for this short term project, and since there should only be two polygons that contribute to any given pixel, drawing the polygons out of order should not be too big of a problem. As the balloon bursts, the surface opacity should approach 1.0 as the springs return to their approximate resting distance and this will prevent opacity related visual problems from occurring.

### 3.4 Source of Balloon

The actual model of the balloon, including the point locations and edges, is being produced by a previous graphics project that implemented a 3D sculpting program.

The 3D sculpting program has two main components which enable the user to define and generate a smooth surfaced balloon. First, the user will design a three-dimensional sculpture by chipping away at an initial cube of material. The material is represented by an octree that gets updated after each chip is processed. The sculpture can be saved to file so that it can be used at later times for viewing or further work. This can be considered as the modeling stage and is based on a previous work.

The second part consists of generating a convoluted surface around the sculpture by getting several parameters from the user:

- Field density (number of fields per cube)
- Field density (number of fields per unit area)
- Cube Size for surface algorithm.

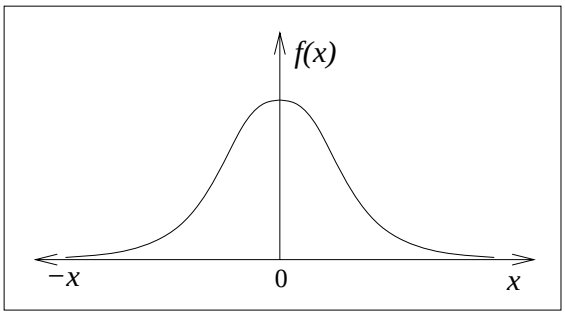


Figure 2: Typical Gaussian function.

The convolution code will use the simplified gaussian function, similar to that in figure 2 for the potential at any point in space:

$$f(x,y,z) = \sum_{k=1}^{n} b_k e^{(\frac{-(x^2+y^2+z^2)}{2})}$$

Where $k$ is all points inside the octree cubes, and $b_k$ is the coefficient. This function will be used by the marching cubes based algorithm to test if the point in space is inside or outside of the surface based on a threshold value.

Figure 3: Basic cube from sculpting program.

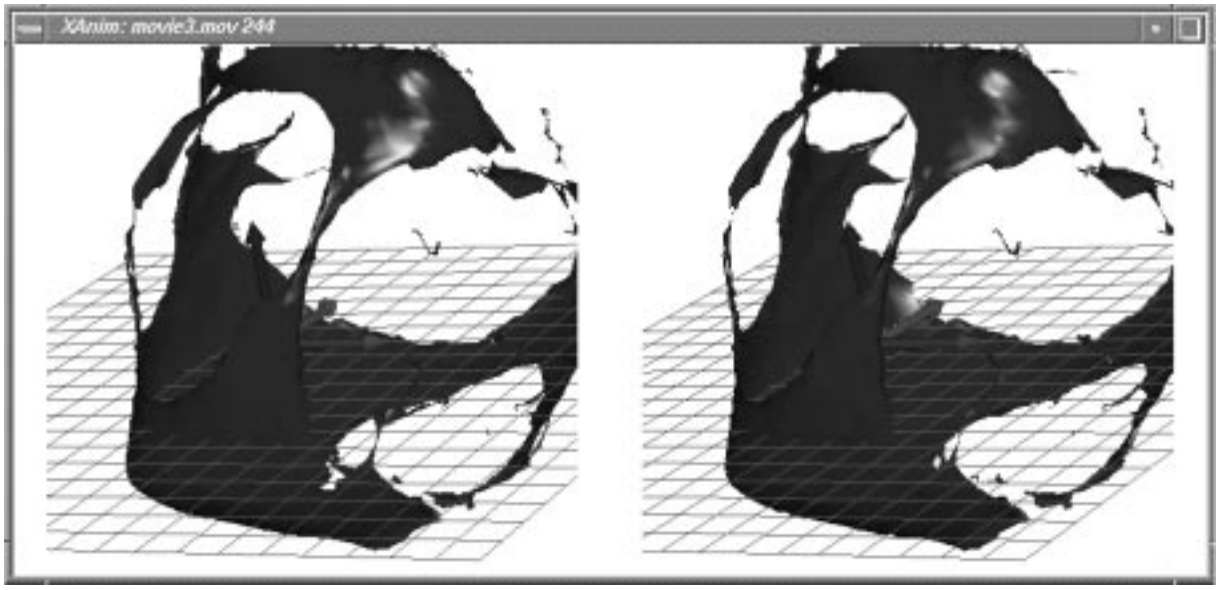


Figure 8: Triangular sculptured balloon exploding (in stereo).

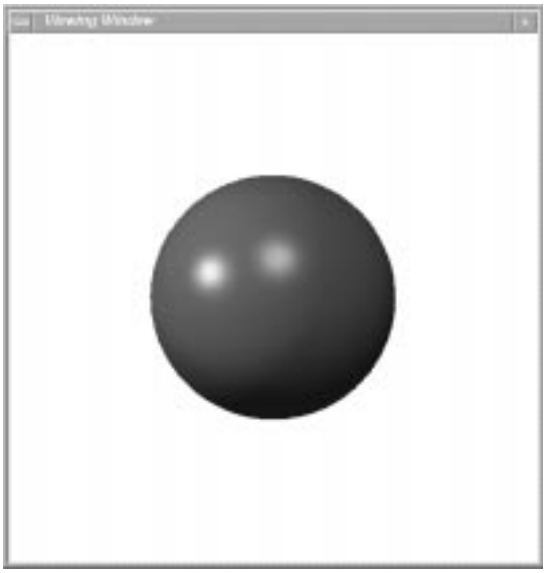

Figure 4: Surface created with one point.

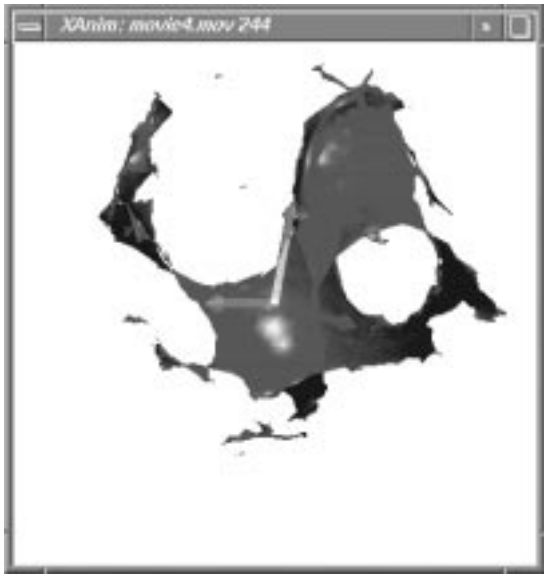


Figure 5: Spherical balloon exploding.

The marching cubes algorithm starts at point (0,0,0) and looks at several random points until it finds one that is on or near the surface, a cube of user defined size (representing resolution) is used at that point and the faces of the cube are tested to determine where the surface intersects, depending on which faces the surface intersects, that piece of the surface is defined and stored on a list of triangles that will be used to draw the final surface. The algorithm continues by processing an adjacent cube whose location is determined from the results obtained from the previous cube. When all cubes are adjacent to all others used by the algorithm, then the surface has bee completely determined and the algorithm is done.

The list of triangles, with surface normals can then be drawn or saved to disk to produce the smoothed surface of the balloon. The polygons that it produces will not be identical in shape or size. The polygons, which are all triangles, will be used to create the points and the spring links between the points. This randomness of connections and spring lengths should contribute to a more natural looking and behaving balloon. If the results of the balloon simulation look good for a simple sphere, more complex balloon shapes can be created

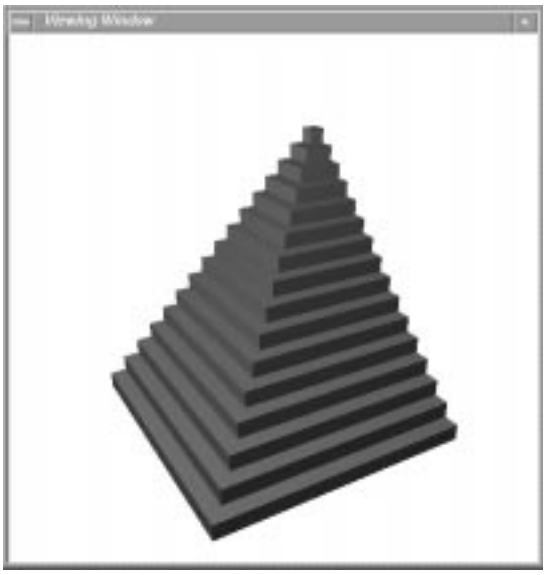

Figure 6: Triangular sculpture.

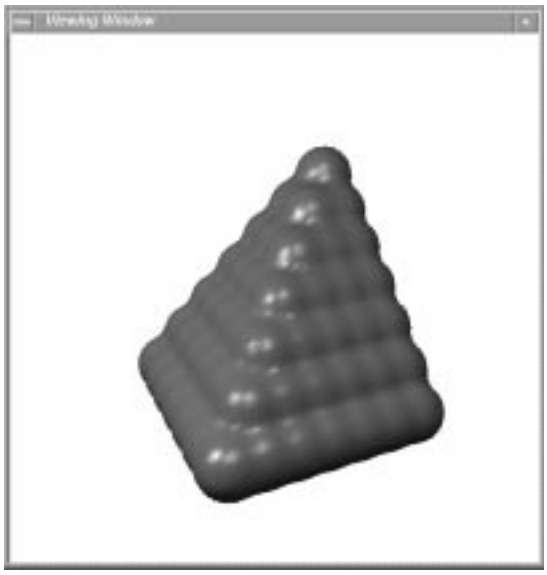

Figure 7: Triangular sculptured balloon.

with the sculpting program and converted into balloons for the simulation. This could create a new type of artwork *balloon art*, where you design the balloons with the sculpting program, and then you blow them up with the balloon simulator!

### 3.5 Data structures and algorithm

| | |
|---|---|
| float | Normal[3] |
| float | Position[3] |
| float | Velocity[3] |
| float | Acceleration[3] |

Table 1: Data structure for each point in the balloon surface mesh.

| | |
|---|---|
| int | Point1 |
| int | Point2 |
| float | RestLength |
| float | Length |
| float | InitSpringConst |
| float | CurrentSpringConst |
| int | Active |
| float | ForceMagnitude |

Table 2: Data structure for each edge in the balloon surface mesh.

Once the triangles of the surface are created, they are loaded into the simulation application and are uniquely sorted into data structures for points, edges, and triangles (see tables 1, 2, and 3). The point and edge data structures are implemented as arrays. Each

| | |
|---|---|
| int | Points[3] |
| int | Edges[3] |
| float | Opacity |
| float | Shininess |
| int | Active |
| struct Triangle * | next |

Table 3: Data structure for each triangle in the balloon surface mesh.

edge uses array indices to keep track of the two points that are associated with it. Similarly, the triangles use array indices to keep track of the three points as well as the three edges that are associated with them. The triangles themselves are loaded into a linked list which is traversed each time step for rendering.

Once the surface has been read into the data structures, the simulation is ready to begin. The user must set the time interval, the internal air pressure, and a weighted spring constant factor before starting simulation. The user can change the time interval or the internal air pressure between any time steps, however, it is not recommended that the user change the weighted spring constant factor more than once, as this usually causes spring constants to vary too widely and can cause unpredictable results.

The simulation algorithm calculates the forces, velocities, and positions of each point for a given time step interval using the formula from section 2 along with standard Euler integration. The algorithm is sequential and is repeated for each time step. It is made up of the following five parts:

- Process all of the points and calculate their position and velocity. Also initialize acceleration and normal

vectors to (0,0,0).

- Process all of the active edges and calculate the spring forces for the two points. Check if the length of the edge is greater than the threshold value and set active FALSE if so. If still active, add the forces to the two points.

- Process all of the active triangles. If more than one edge is inactive, set triangle inactive, else, calculate normal and add to the three points. Also calculate opacity and shininess based on the average length of the three edges divided by the threshold length.

- Process all points again and normalize their normal vectors and multiply scalar air pressure by the normal and add the resulting vector to the acceleration vector.

- Draw all of the active triangles.

### 3.6 Explosion

During the simulation, the balloon will expand as the points are pushed outward by the internal air pressure. Each spring will stretch until its length exceeds the threshold length. If the spring length exceeds the threshold, the edge is set inactive and no longer contributes spring forces on the points. Once several edges have become inactive the remaining springs pull the points even farther apart and a rip begins to form. The springs along the rip begin to exceed the threshold in a chain reaction until the rip severs the surface into separate pieces. Several rips can form simultaneously, resulting in the balloon exploding into many pieces. Large pieces of the surface survive the ripping and begin to shrink due to the spring forces of the remaining edges. These pieces fly away due to the physical simulation of the acceleration and velocity of the points.

### 3.7 Visualization

The simulation is written in GL and C and runs on an SGI graphics workstation. A lighting model is used to implement the specular and translucent attributes of the surface. Of course, stereoscopic viewing is a necessary viewing feature for a more realistic looking simulation. Since real-time animations are not provided directly by this application, stereo methods, such as Crystal-Eyes glasses, can not be used. Therefore, the stereo method used is a simple combination of both left and right eye views in the graphics window. The graphics window is saved for each frame and the user can view a movie of the animation using simple mirror-based glasses.

## 4 Results

Surprisingly realistic results were achieved after only one full week of development of this implementation. Simulations of a spherical surface and a triangular surface, both consisting of approximately 3000 polygons, were run. Pictures of the sculptures that were used for the surfaces are in figures 3 and 6. The surfaces, which were used as input to the simulation, were created from the sculptures and are shown in figures 4 and 7. The simulations, which were run on an Indigo2 with a 200 MHz CPU and 64 MB of memory, achieved frame rates of approximately one per second. Sample pictures from the simulation are shown in figures 5 and 8. Figure 8 shows the left and right eye views of a stereoscopic image of the triangular balloon exploding. Mirror glasses that restrict each eye to its corresponding view were used to view the stereo simulation. Approximately 400 frames from each simulation were saved and used to create animation movies.

Experimentation with the spring constant and damping constant showed that a spring constant of 20.0 and an damping constant of 1.0 produced simulations that looked realistic. Spring constants higher than 20 resulted in unstable surfaces due to abnormal stretching of the point mesh which produced premature explosions and did not allow for the balloon surface to expand first. Damping constants greater than 1.0 resulted in an over compensation of the point velocities which produced unstable surfaces as well.

Selection of a threshold spring length was done through experimentation. In order to produce a realistic simulation, the balloon must be allowed to expand before it explodes, however, if it expands too much, it will fill the viewing window and exhibit infinite volume properties that are not characteristic of real balloons. It was determined through trial and observation that selecting a threshold that is approximately twice the average resting length of the edges produces a realistic simulation.

The internal pressure was not as critical of a factor, however the pressure must be sufficiently large to cause the balloon to explode or else it will achieve a stable equilibrium and remain inert. In addition, if the internal pressure is set to a very high level, the balloon will simply disintegrate into many small particles. An internal pressure of approximately 10.0 worked well with the simulations that were discussed in this paper.

The time interval was the most critical factor in achieving a realistic simulation. If the interval was too large, then the Euler integration would create an unstable surface and polygons would fly apart. Also, if the time interval were too small, then it would take many frames before anything interesting would happen. Through much experimentation, a time interval of 0.025 seconds was selected. This time interval usually produced a complete explosion animation in approximately 400 frames.

### 4.1 Problems

In some cases, such as the sphere balloon, the surface is sufficiently symmetrical that the rips in the surface occur at many locations simultaneously. This is undesirable, since the balloon resembles a disintegration rather than an explosion. In these cases, an artificial rip is introduced after the balloon expands to the point where other rips start to occur. The artificial rip allows the surface to break up into larger pieces, resulting in a more realistic explosion.

Given the fact that this simulation was developed on machines that did not all have alpha bit-plane capabilities, the implementation of variable opacity was not completed. Also, in order to insure accurate transparency rendering, the polygons must be drawn in visibility order. Due to time constraints, visibility ordering was not implemented. Experimentation of opacity and shininess calculations were done, however, and were found to greatly increase the time per frame when GL material settings for shininess and opacity were bound for each polygon drawn. A possible solution to this is to average the opacity and shininess for all of the polygons and set these values once per frame.

## 5 Conclusion

The simulation of exploding balloons is achieved through physical modeling of a surface which consists of point masses connected by springs. Realistic animations of different shaped balloons were created using the technique described in this paper. Several issues such as self-collision and computational fluid dynamics were not addressed due to the complexity and cost of implementing them. Realistic animations were produced despite this fact and frame rates of approximately one per second for a 3000 polygon balloon were achieved. This explosion simulation is not just limited to balloons, but can be used to model many different explosive situations by adjusting spring and damper constants and by designing surfaces that model objects other than balloons.

## References


[BHW94] David Breen, Donald House, and Michael Wozny. Predicting the drape of woven cloth using interacting particles. *Computer Graphics (ACM SIGGRAPH Proceedings)*, pages 365–372, 1994.

[CHP89] John Chadwick, David Haumann, and Richard Parent. Layered construction for deformable animated characters. *Computer Graphics (ACM SIGGRAPH Proceedings)*, 23(3):243–252, August 1989.

[CYMTT92] Michel Carignan, Ying Yang, Nadia Magnenat-Thalmann, and Daniel Thalmann. Dressing animated synthetic actors with complex deformable clothes. *Computer Graphics (ACM SIGGRAPH Proceedings)*, 26(2):99–104, July 1992.

[DG95] Mathieu Desbrun and Marie-Paul Gascuel. Animating soft substances with implicit surfaces. *Computer Graphics (ACM SIGGRAPH Proceedings)*, pages 287–290, August 1995.

[TF88] Demetri Terzopoulos and Kurt Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. *SIGGRAPH '88 Conference Proceedings*, 22(4):269–278, August, 1988.

[TPBF87] Demetri Terzopoulos, John Platt, Alan H. Barr, and Kurt Fleischer. Elastically deformable models. *SIGGRAPH '87 Conference Proceedings*, 21(4):205–214, July 1987.

[VCMT95] Pascal Volino, MArtin Courchesne, and Nadia Magnenat-Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. *Computer Graphics (ACM SIGGRAPH Proceedings)*, pages 137–144, August 1995.

[Wei86] Jerry Weil. The synthesis of cloth objects. *SIGGRAPH '86 Conference Proceedings*, 20(4):49–54, August, 1986.

[WH94] Andrew Witkin and Paul Heckbert. Using particles to sample and control implicit surfaces. *Computer Graphics (ACM SIGGRAPH Proceedings)*, pages 269–277, July 1994.

[Wit95] Andrew Witkin. An introduction to physically-based modeling. *ACM SIGGRAPH '95 Course Notes*, 34:B1–C12, August 1995.