Modeling Realistic Inflation and Deflation Effect on Closed 3D Geometric Mesh for Computer Animation

Russel Ahmed Apu[1], Marina Gavrilova[2]

[1]Department of Computer Science, University of Calgary, **apu@cpsc.ucalgary.ca**
[2]Department of Computer Science, University of Calgary, **marina@cpsc.ucalgary.ca**

Abstract:

The paper presents a new method to model inflation/deflation deformation effect of closed 3D polygonal mesh in semi-real time. The method involves converting an arbitrary closed isotropic mesh to anisotropic mesh with internal nodes using a new algorithm called Projective Cascading Propagation (PCP). We demonstrate how this method could be used to model solid objects effectively and eliminate the problems associated with ordinary Mass Spring system. We also prove that the illustrated method will generate mesh that could still be solved in linear time using implicit solvers. The method also considers collisions and responses during deformation.

Keywords: Projective Cascading Propagation (PCP), Implicit Differential Equation (IDE), mass spring system, constrained particle system, volume preservation, sparse matrix.

1. Introduction

Inflation and deflation of objects are used in computer animation to model flexible animated characters, imploding effects, explosion, morphing etc. Although finite element methods are more accurate, they are computationally extensive [2][3][4]. However, in computer animation accuracy is usually not an issue. An alternative approach is to use Physically Based Modeling (PBM) [9][10]. The method uses a constrained particle system and models the deformation by means of computing interacting forces in the system and advancing deformation using discreet time steps [12]. A typical mass spring system has a number of particles (i.e. vertices) and a number of associations (i.e. linear spring force) and solves deformation of that system using an ODE solver [9]. However such methods have limitations and realistic results are hard to obtain. Models solved using ODE solvers lack stability and fast responses to forces in the system [12]. Recent advances in PBM enable us to use IDE integrators that solve some of the underlying problems [1] (i.e. gaining energy, instability etc.).

Volumetric deformation, collision and responses are harder to model realistically using PBM. Forces in the system cannot propagate unless springs are deformed (from ideal length) locally. This results in exaggerated local deformation and instability (a wiggling effect). Since characters in computer animation are often considered to be rigid, ordinary IDE is still insufficient for the purpose. Therefore, we present an efficient method that could be used to produce realistic volumetric deformation effects (fig. 1). We introduce the new method called Projective Cascading Propagation (PCP) which allows us to generate a mesh with internal nodes. The method allows more flexibility than the frontal method [3]. We prove that the method preserves the sparseness of the linear force gradient matrix used in the IDE solutions [1][7][13]. This ensures that the system could be solved efficiently (in linear time) using IDE solvers.

2. Related Work

Mass spring system is a very popular method for animating different deformable objects. P. Volino introduced a simple and effective scheme for implementing such a system for computer animation [11]. However, the paper that made mass spring system more practical and realistic is the one from David Baraff in SIGGRAPH 1998 [1]. Baraff demonstrated

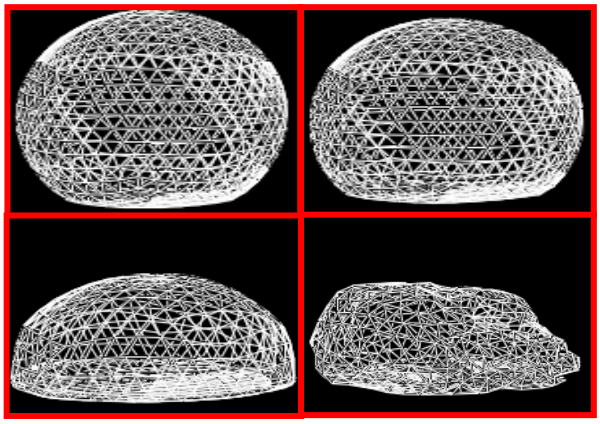

Fig. 1: Modeling deflation of a sphere using mass spring based constrained particle system.

the use of implicit solver that eliminated the notorious problem of gaining kinetic energy inherent to ODE solvers [12]. He also introduced methods to achieve axial constraints for effective collision response. On the other hand in ACM SSMA 1999, G. Hirota outlined an easy and effective method for fast computation of closed polygonal mesh, which we adopt for our system [6]. In 11th Eurographics Workshop 2000, D. Bourguignon demonstrated the power of anisotropy over conventional isotropic structure, as well as outlined methods to realistically model deformation by conforming to volume preservation [3]. It also demonstrated how mass spring system could perform far better using the sampling internal nodes and connectivity.

3. The Simulation Model

The Object model used in this paper consists of a geometric object with n vertices (or nodes) and d faces and a set of $(3n \times 3n)$ matrices to solve the mass spring system [1][5]. In addition to list of vertices we also maintain list of edges and list of tetrahedrons (generated during the execution of the PCP algorithm presented in later section). Finally, we maintain a set of attributes for each vertex (i.e. mass and surface node flag), edge (i.e. rest length and spring coefficient) and tetrahedron (i.e. rest volume). A description of the particle system applied to the model is given below.

3.1. The Mass Spring System

For a simple ideal spring the force f_s can be described in terms of displacement, that is, $f_s = k_s x$ where k_s is the spring constant and x is the displacement length of the spring relative to the rest length [9]. The rest length is the natural length of the spring when no external forces are applied. This allows us to view the whole system as a second order differential equation. The phase/state in a given instance (at time t) can be defined as the vector $\langle x, \dot{x} \rangle = \langle x_1, x_2, x_3, v_1, v_2, v_3 \rangle$ in a six dimensional space for a single particle in 3D space. At any point during the simulation, the force can be computed using the following equation:

$$a = \text{ Particle a, } b = \text{ Particle b, } r = \text{ Rest Length}$$

$$\vec{f}_a = -\left[k_s(|\vec{l}| - r) + k_d \frac{\vec{\mu} \bullet \vec{l}}{|\vec{l}|} \right] \frac{\vec{l}}{|\vec{l}|} \qquad (1)$$

$$\vec{f}_b = -\vec{f}_a$$

Where f_a and f_b are the forces on a and b, respectively, $\vec{l} = \hat{a} - \hat{b}$; where $\hat{\tau}$ is the coordinate of τ, k_s and k_d are the spring constant and dampening constant for spring ab respectively. $\vec{\mu} = \vec{v}_a - \vec{v}_b$ is called the time derivative where $\vec{v}_i$ is the velocity vector of particle i.

3.2. The Constrained Particle System

In general for any practical application, we would like to gain more control or specify constrains that will produce a desired animation (i.e. hold some of the nodes stationery, do seams, constrain the velocity on collision etc.). In order to apply constraints over the motion of each particle (i.e. during collision response) we can take away degree of freedom from a vertex along arbitrary axis. We construct a 3×3 matrix W_i for each particle as follows:

$$W_i = \begin{cases} \frac{1}{m_i} * \mathbf{I}; & \text{if } \mathrm{dof}(i) = 3 \\ \frac{1}{m_i} * (\mathbf{I} - p_i p_i^{\mathrm{T}}); & \text{if } \mathrm{dof}(i) = 2 \\ \frac{1}{m_i} * (\mathbf{I} - p_i p_i^{\mathrm{T}} - q_i q_i^{\mathrm{T}}); & \text{if } \mathrm{dof}(i) = 1 \\ \varnothing^{3\times 3}; & \text{if } \mathrm{dof}(i) = 0 \end{cases} \qquad (2)$$

Here, $\mathbf{I}$ is a 3×3 identity matrix, m_i is the mass of particle i. The function $\mathrm{dof}(i)$ returns the degree of freedom of particle i. When $\mathrm{dof}(i)$ is 1 the displacement of the vertex is restricted along vector p_i. When $\mathrm{dof}(i)$ is 1 the displacement of the vertex is restricted along the plane defined by orthogonal vectors p_i and q_i. The formula above allows us to define/constrain the particles.

To compute the acceleration of a particle, the force must be divided by the mass. Therefore, the inverse of the mass m_i of node i is multiplied by the identity matrix. Then, we construct the diagonal matrix $\mathbf{M}^{-1}$ by replacing only diagonal elements of a $\varnothing^{3n \times 3n}$ matrix. To constrain the particles, we use all of the W_i in the implicit equation replacing $\mathbf{M}^{-1}$ in equation (5).

3.3. Volume computation and preservation

We performed two different volume preservation techniques for the method: implicit and soft explicit. Implicit volume preservation involves penalty forces along the faces of a deformed tetrahedron [3]. This enables resistance to local deformation. However this method is computationally extensive. We also perform a soft volume preservation technique that involves applying radial penalty forces to the surface vertices as the global volume deviates from ideal state (fig. 2b). Thus we compute force applied to each of the triangle in the mesh and add the force to the force accumulator

of each of the vertices that define the triangle [6]. The force on a triangle is calculated as:

$$\begin{aligned}
&\text{from gaseous law,}\quad \mathrm{P}_1V_1 = \mathrm{P}_2V_2,\quad \mathrm{P}_2 = \frac{\mathrm{P}_1V_1}{V_2}\\
&\therefore \left|\vec{f}_j\right| = \mathrm{P}_2A_j\\
&\text{let, } \{u_1,u_2,u_3\} = \text{ Vertices of face } j\\
&\text{and, } \hat{i} = \text{ the coordinate of vertex } i\\
&\therefore \vec{f}_j = \left[\frac{\mathrm{P}_1V_1}{V_2}\{(\hat{u}_2-\hat{u}_1)\bullet(\hat{u}_3-\hat{u}_1)\}\right] * \eta(\mathrm{u}_1,\mathrm{u}_2,\mathrm{u}_3)\\
&\quad \text{where, } \eta(\mathrm{u}_1,\mathrm{u}_2,\mathrm{u}_3) = \left[\frac{(\hat{u}_2-\hat{u}_1)\times(\hat{u}_3-\hat{u}_1)}{|\hat{u}_2-\hat{u}_1| * |\hat{u}_3-\hat{u}_1|}\right]
\end{aligned} \tag{3}$$

Here V_1 is the volume of the object at initial state, and P_1 is defined as the pressure coefficient, our way to control the intensity of the volumetric forces. The volume of a tetrahedron with a base triangle $T_i = \langle \alpha, \beta, \gamma \rangle$ and apex vertex at the origin can be computed as (given that the origin is not inside the object):

$$\begin{aligned}
\mathrm{Volume}(T_i) &= \frac{1}{6}\begin{vmatrix} x_\alpha & y_\alpha & z_\alpha \\ x_\beta & y_\beta & z_\beta \\ x_\gamma & y_\gamma & z_\gamma \end{vmatrix}\\
\mathrm{Volume}(\mathrm{Mesh}) &= \sum\nolimits_{T_j \in \mathrm{Mesh}} \mathrm{Volume}(T_j)
\end{aligned} \tag{4}$$

3.4. Implicit DE solver

An ODE solver takes a time step based on instantaneous force. When the system has stiff spring, forces fall-off rapidly as the spring retains its original shape. As a result, a regular ODE solver accumulates energy to the system and to prevent aberrant oscillation the time step has to be really small. To resolve this problem we use the implicit solvers (IDE) for constrained particle system. An IDE solver does not gain kinetic energy and allows us to take larger time steps [1]. The catch is that each time step is computationally more extensive than ODE solvers (i.e. Runge Kutta 4).

Let us assume that the state vectors are functions of time. Therefore given current state $\mathrm{x}(t_0)$ and velocity $\mathrm{v}(t_0)$ for the system of particles, we have to determine $\mathrm{x}(t_0+\Delta t)$ and $\mathrm{v}(t_0+\Delta t)$. Therefore our DE can be represented as:

$$\begin{aligned}
&\frac{d}{dt}\begin{pmatrix} x \\ v \end{pmatrix} = \begin{pmatrix} v \\ \mathbf{M}^{-1}f(x,v) \end{pmatrix}\\
&\text{with, } \Delta x = \mathrm{x}(t_0+\Delta t) - \mathrm{x}(t_0), \Delta v = \mathrm{v}(t_0+\Delta t) - \mathrm{v}(t_0)
\end{aligned}$$

$$\begin{aligned}
&\therefore \begin{pmatrix} \Delta x \\ \Delta v \end{pmatrix} = \Delta t \begin{pmatrix} v_0 \\ \mathbf{M}^{-1}f_0 \end{pmatrix}\\
&\text{where, } v_0 = \mathrm{v}(t_0),\ f_0 = \mathrm{f}(t_0)
\end{aligned} \tag{5}$$

Here $\mathbf{M}^{-1}$ is a diagonal matrix with $1/m_i$ on row (i,i). This is by far the equation for the forward Euler (ODE) method. For the backward (implicit) Euler stepping, we take a step based on the gradient of the destination state:

$$\begin{pmatrix} \Delta x \\ \Delta v \end{pmatrix} = \Delta t \begin{pmatrix} v_0 + \Delta v \\ \mathrm{M}^{-1}f(x_0+\Delta x, v_0+\Delta v) \end{pmatrix} \tag{6}$$

A Taylor series expansion (approximated to the first order) can be used to expand the function and obtain the following:

$$\begin{aligned}
\mathrm{D}\,\Delta v &= \Delta t \mathrm{M}^{-1}\left(f_0 + \Delta t \frac{\partial f}{\partial x} v_0\right)\\
\text{where, } \mathrm{D} &= \left(\mathrm{I} - \Delta t \mathrm{M}^{-1}\frac{\partial f}{\partial v} - \Delta t^2 \mathrm{M}^{-1}\frac{\partial f}{\partial x}\right)
\end{aligned} \tag{7}$$

The equation above can be viewed as a system of linear equation in the form $\mathrm{D}\chi = E$, where $\chi = \Delta v$. This system is then solved using a numerical method such as the bi-conjugate gradient descent method [5][7][14]. One important fact here is that the structure of the matrix D depends on the force gradient matrix $\partial f/\partial x$. Finally, to implement collision responses, the matrix W is used instead of M^{-1} in (7).

4. METHOD DESCRIPTION

We took a polyhedral triangular mesh as an input and generated a complex volumetric mesh with internal nodes and connectivity. This mesh is passed to the IDE solver where deformation is coordinated with forces of gravity and pressure. The system is then allowed to collapse by cascading reduction of target length of springs and global volume. Stiffness of surface spring (relative to the inner system) determines how elastic the skin of the object is. Stiffer spring tend to produce more profound wrinkle effects. Collisions are tested during every time step.

4.1. Problems with regular isotropic mesh

A typical geometric mesh has slow reaction and oscillation problems. Let us consider a 2D circular system of mass spring bouncing on a surface (fig. 2). Since, springs has to deform in order to exert forces to neighboring nodes, the top springs cannot react until the deformation effects propagates gradually through

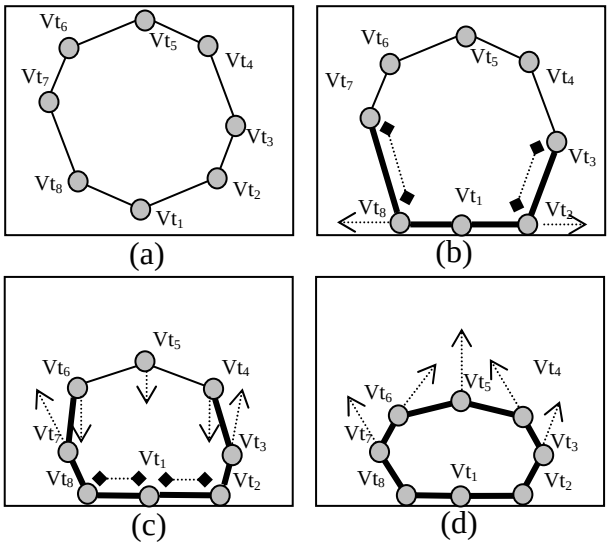

Fig 2: Propagation of spring forces during deformation. Arrow indicates repulsion and diamonds indicate attraction forces. Springs in bold are deformed.

the silhouette (fig. 2). In the meantime, the springs start to oscillate. This effect is profound for the case of stiff springs.

One way to allow the forces to propagate and diffuse is to connect all nodes internally (fig 3). However the system would be far too impractical to solve using IDE integrators because the number of springs in the system is $O(n^2)$, where n is the number of particles. A better approach is to generate internal nodes and ensure some form of connectivity.

4.2. Generating the mesh

Generating the mesh is a two-step process. First we generate a number of sample points that satisfy certain criteria and then use the Projective Cascading Propagation (PCP) algorithm to ensure proper connectivity of the internal nodes.

There are several known methods to generate sample points. The simplest way is to generate random points and eliminate nodes that fail certain test. To generate these sample points, two user-defined constants Φ and Ψ are used to constrain the density of the nodes (see the test below). A different approach is to generate randomized points and use PBM to minimize error using penalty forces. This is more effective for densely packed system. However the method is more complex. It has been observed that for our case, dense packing of inner nodes does not necessarily improve visual quality. As long as internal nodes are uniformly and randomly distributed, the

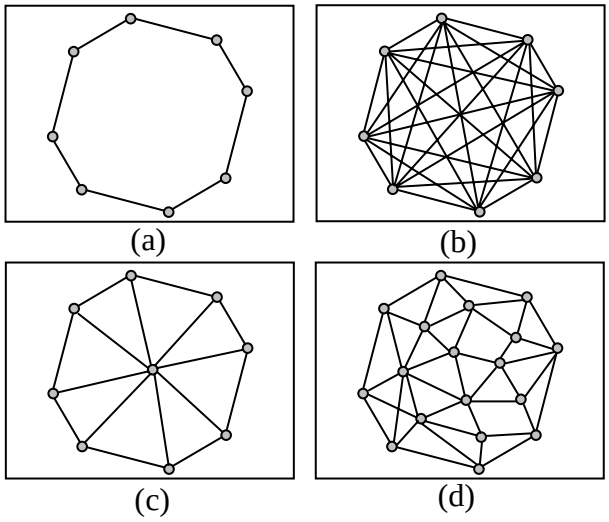

Fig. 3: Different connectivity for mesh deformation. (a) regular isotropic (b) complete anisotropic connectivity (c) single internal node (d) multiple nodes

simulation appears to be realistic without any noticeable visual artifact. Every internal node must satisfy the following tests (fig. 4):

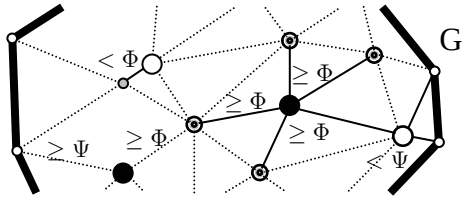

Fig. 4: Test scenario: black nodes succeed test and white nodes fail the test.

Let, $\Phi \in \mathbb{R} =$ Internal Clearence Coefficient

and $\Psi \in \mathbb{R} =$ Surface Clearence Coefficient

Both Φ and Ψ are predefined constant in the system

$\lambda(i,j) =$ Distance between vertex i and j

$\lambda^{\dagger}(i) =$ Distance between vertex i and surface of mesh

$$\Gamma(i) = \begin{cases} true & ; \text{ vertex } i \text{ is inside the mesh} \\ false & ; \text{ Otherwise} \end{cases}$$

$G =$ Set of vertices in the mesh

$$\forall i \forall j [((i \in G) \wedge (j \in G) \wedge (i \neq j)) \rightarrow (\lambda(i,j) \geq \Phi)]$$
$$\forall i [(i \in G) \rightarrow (\lambda^{\dagger}(i) \geq \Psi)]$$
$$\forall i [(i \in G) \rightarrow \Gamma(i)]$$

Since the quantity Φ and Ψ depends on the shape and size of the object (fig. 4), we rescale the object so that its bounding box is normalized. We say a box is normalized if the minimum component of its dimension has unitary length.

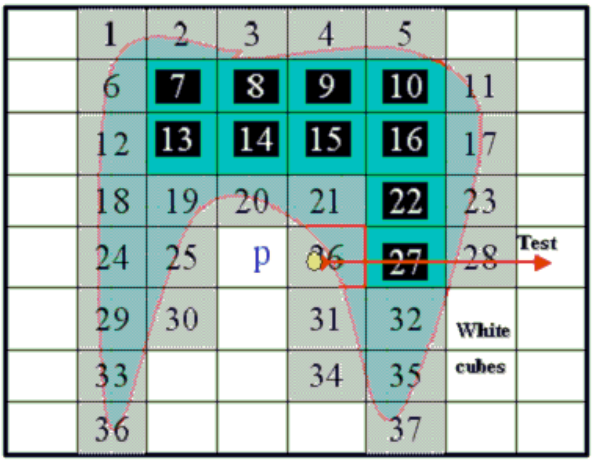

Fig. 5: Dividing the object into quadrants and indexing each occupied cube. A fast constraint test is performed for the point. (white cube=NULL)

For fast point generation, we divide the object's bounding box into a number of quadrants (a 3D grid of cubes). Each quadrant is assigned a unique index (fig. 5). To generate a new point, we obtain a random tuple $B \in \mathbb{Z} \times \mathbb{R}^3$ so that B is mapped in a random quadrant with a random 3D offset. The new vertex is tested against all object in that quadrant followed by all the neighboring quadrants (26 at most). If any of the tests fails, the vertex is discarded. The size of the quadrants is chosen based on Φ and Ψ so that only a constant number of vertices could be obtained inside one quadrant. We found that this method is significantly faster than testing a new vertex against all existing point.

Considering the nature of the volumetric deformation we intend to model, a new algorithm was designed. PCP is a greedy algorithm designed to produce an effective uniform topology that associates every internal node with its neighbors. It generates a number of edges to link and bind all internal nodes with the mesh. The PCP algorithm starts at the surface and cascades the linking process towards the nearest unlinked internal vertices (fig. 6). Because at every stage each node links with the nearest neighbor, the active nodes propagate deeper into the structure (fig. 7). The algorithm ensures that regardless of the shape of the object (i.e. see simple and complex shapes in fig. 11), the linking process is consistently progressing towards unlinked internal nodes in increasing distance from the surface. We define projective multiplicity as the number of rays thrown from each vertex. From our experiment, we found out that higher projective multiplicity generates more convincing animation, however takes more time. We also found that multiplicity has to be at least 3 for any practical simulation.

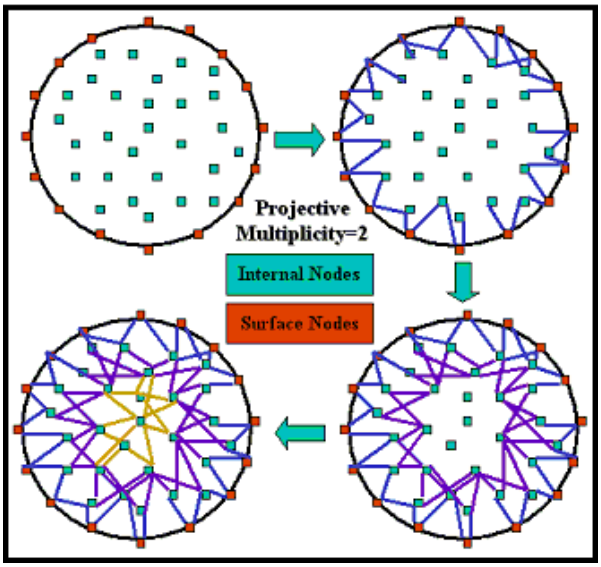

Fig. 6: A simplified (2D) algorithm walkthrough for PCP based mesh generation

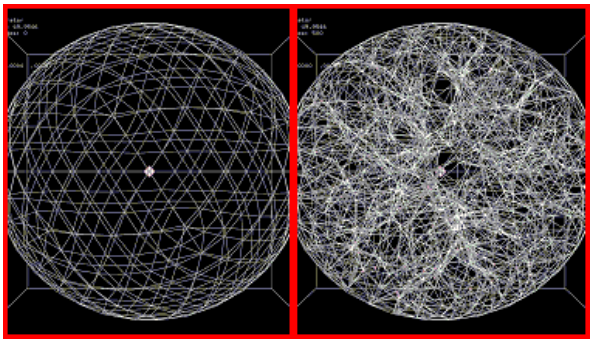

Fig. 7: Sphere before conversion (left) and after PCP based mesh generation (right)

The whole process starts by generating a list of all vertices and coloring them black. Surface nodes are then colored white. At every stage we project a constant number of rays from each white node to the neighboring black nodes. Selections of these vertices are based on a priority function that takes into account the distance, degree, surface distance etc. into consideration. A reasonable combination that seems to work well could be obtained using trial and error. The next task is to remove the white vertices from the list and color the newly connected black vertices as white vertices and repeat the process as long as there are black nodes left in the list. The algorithm for PCP is given below in pseudocode (fig. 8):

```
1     Procedure CPM_Link_Nodes(M)
2       Make a Queue Q of V(M)
3         While there is black nodes left do
4           W←Dequeue_white(Q)
5           Update_Priority(Q)
6           For j←1 to ORDER do
7             B←Extract_Non_White_MAX(Q)
8             Link (B,W)
9             Color[B] ←Gray
10            If Black_Count(Q)=0 then return
11              For every node r in V(M) do
12                If (Color[r]=Gray)
                    Color[r] ←white
```

Fig. 8: PCP algorithm for linking internal nodes

4.3. Proof of Sparseness of Mesh Using PCP and Implicit DE Solvers

Lemma 1: Given an isotropic mesh Π, with n nodes and a PCP based conversion $\tilde{\Pi}$ with k internal nodes and a constant projective multiplicity λ, the number of non zero entries in the linear system in (7) is $O(|\mathrm{V}(\tilde{\Pi})|) = O(n+k)$.

Proof: Since our system has n mesh vertices and k internal vertices the size of the linear system in (7) is $3(n+k)$. The term $\partial f/\partial v$ in the IDE equation deals with the dependency of forces over velocity of particles, which is for most cases kinematics dampening and friction. Therefore, they are only in the block diagonal form contributing $O(n+k)$ nonzero terms. Secondly, we observe that the matrix $\partial f/\partial x$ have non zero entries for blocks (array[3x3]) that resemble a spring between node i and j. Other entries has Jacobi gradient equal to zero. Now, we know that for a regular non intercepting isotropic mesh, $|\mathrm{E}(\Pi)| = O(|\mathrm{V}(\Pi)|) = O(n)$. On the other hand since every vertex projects exactly λ edges during PCP, the number of additional edges generated are $O(n+k)$. Since we only have block non-zero entries for every edge between vertices there can only be $O(n+k)$ nonzero entries in the Matrix A of the linear system Ax=b. The LHS matrix A is really sparse (size of the matrix is $O(|\mathrm{V}(\Pi)|^2)$ and number of nonzero entries is $O(n)+O(n+k)=O(n+k)$.

The quantity k is user defined. It determines the quality and accuracy of the simulation and pertains to the realism. However, the simulation will run slower if higher values of k are used. Finally, the upper limit for k is constrained by the choice of Φ, Ψ and the shape of the object. If the program fails to generate k internal nodes, then it generates the highest number of possible nodes k' under given constrains. If the user is not satisfied with this quantity the simulation could be executed with a lower value of Φ and Ψ.

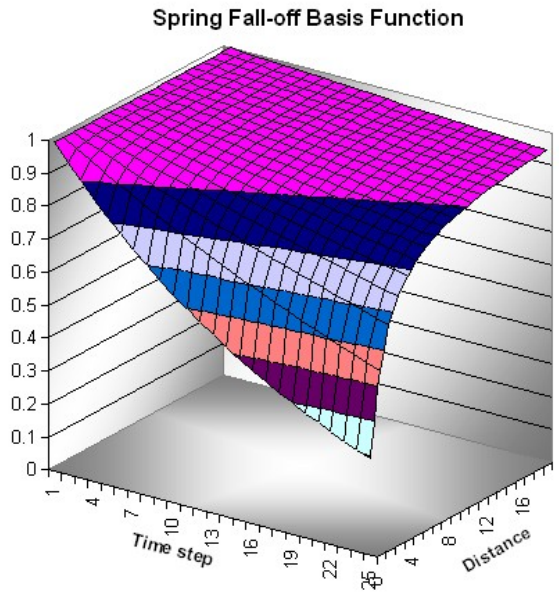

Fig. 9: Fall-off basis function to collapse spring lengths in (8). This function allows local control around the pivot during the simulation

4.4. Modeling Deflation and Inflation Effects

Deflation effect is modeled using a successive collapse of target volume and spring length. A number of pivots is set up inside the object before the simulation. Every time step, the spring collapses based on a fall-off function (fig. 9). The formula for this function is given below:

$$r_{t+1}^{ij} = \left(1 - \frac{R}{\mathrm{Piv\_Dist}(i,j)+1}\right) \cdot r_t^{ij};\ R \in [0,1) \subset \mathbb{R} \quad (8)$$

Where, r_t^{ij} = Target length of spring ij before time step

r_{t+1}^{ij} = Target length of spring ij after time step

$\mathrm{Piv\_Dist}(\omega)$ = Distance between Pivot and segment ω

R = Depletion Coefficient

This function (8) allows us to deform the mesh volume locally. It is important to set the target global volume of frame $t+1$ equals to the calculated volume of frame t. There are three different methods we applied to model the volumetric collapsing. In the first method, we interactively moved the pivot around to control the collapsing. In the second method we used predefined path for the pivot. In the third method, we used a number of predefined presets of pivots each triggered after a certain number of steps. Finally, any

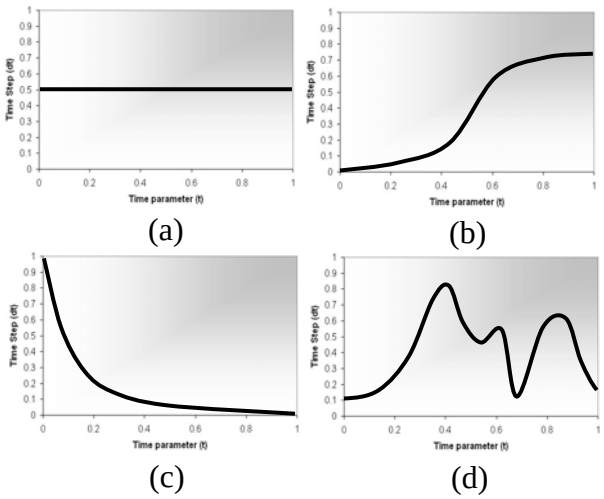

Fig. 10: Different time warp function. The entire simulation time is parameterized as $t \in [0,1]$ and after every frame simulation is advanced by $\mathrm{dt}(t)$. The warp functions above are (a) Constant (b) Elastic (c) Explode (d) Custom

of these alternatives will bring the same result, however they could be used to model a variety of deformation (i.e. balloon, punctured tire, collapsing structure etc.).

One important point to consider during the deflation process is the surface area reduction. In the case of an elastic object (i.e. a rubber balloon) the surface area is reduced as the pressure goes down. Therefore we don't see as many wrinkles during deflation. In order to allow such deformation, we use a fall-off basis function similar to (8) to reduce the global area. Surface area reduction could be achieved by a variety of methods. The method we used is based on bend force. First we compute the average bend force of connected edges for every node. Then we modify spring length , based on the following formula (9):

$$l_{t+1}^{ij} = \left[1 - \frac{1}{2} R' * \left(\frac{\bar{f}_{bend}(i) + \bar{f}_{bend}(j)}{\mathrm{MAX}(\{\bar{f}_{bend}(\sigma) \mid \sigma \in \mathrm{V}(G)\})} \right)\right] l_t^{ij} \quad (9)$$

Where, $R' \in [0,1) =$ Surface Relaxation Coefficient

$\bar{f}_{bend}(i) =$ Average bend force of the neighb. of vtx. i

In order to model the inflation effect, we first modeled deflation using the object in regular time-step, and then reversed the animation. In addition to that, we applied non-uniform time step using a suitable time warp function (fig. 10) to render the animation such that, we produce a variety of imploding effects (fig. 11). For example, the explode warp (fig. 10c) is good for modeling objects such as airbags, explosion etc. On the other hand, the elastic warp (fig. 10b) is good for modeling gradual effects (i.e. inflating a balloon).

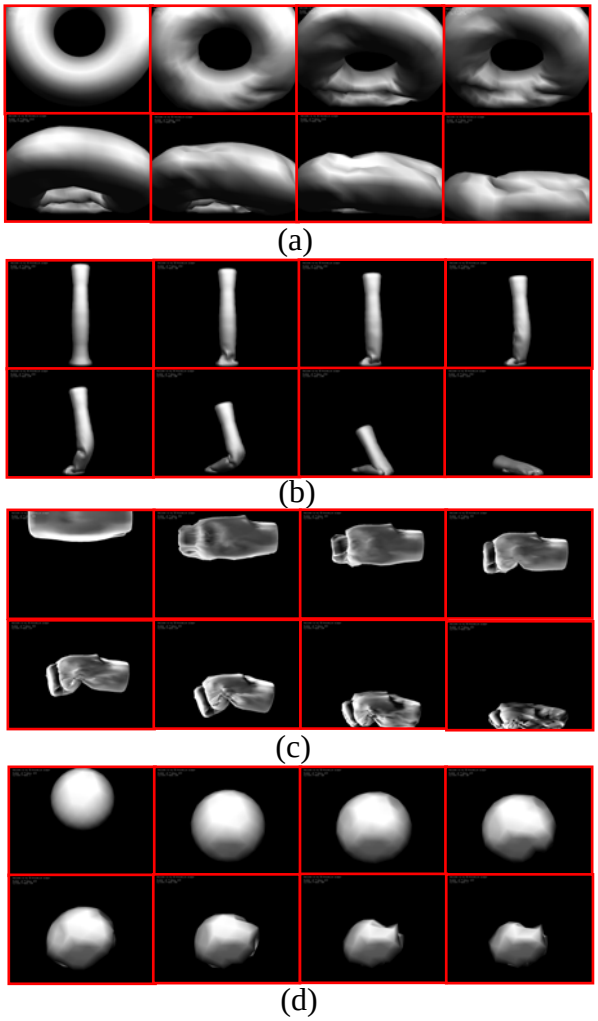

Fig 11: Sample Animation strip using the method. (a) Torus (b) Handle (c) Head (d) Sphere

5. EVALUATING THE RESULT

We have found that it is a good practice to preview the simulation interactively in real-time using ODE Euler solver and then render the simulation using IDE solvers. Our comparison of both methods has revealed that due to the connectivity of the PCP based mesh, the system is much more tolerant to oscillation and instability using ODE solvers. Often, there is little difference between ODE and IDE solution in terms of visual quality. However, ODE based solutions can be a lot faster than implicit methods. However, when the stiffness of spring is very rigid (i.e. a space craft hull collapsing) ODE cannot produce realistic results.

Runtime analysis has shown that the convergence of the linear system is more time consuming when there are large and nonlinear force gradients present in the system. Including volumetric and bend forces will

require use of Biconjugate Gradient Descent (BCGD) method. The problem is that the latter method takes twice as much as time to converge. In practice we found it to be a good solution to drop terms from the equation (8) to make the linear system symmetric and positive definite. If time steps are small enough so that volumetric force gradients are small, they could be discarded from (9) altogether. This does not affect the quality significantly, but speeds up the simulation.

The results (in fig. 11) demonstrate some realistic effects such as bending and wrinkles that could be obtained using this method. Figure 11a demonstrates an imploding torus shaped object with wrinkles forming on the skin as the structure leans and bends towards the front. Figure 11b demonstrates a realistic folding effect on a long tube-like object due to a collision. Figure 11c models a realistic collision and volumetric collapsing effect. These results are highly desirable in character animations. However, ODE based methods are not stable enough to produce such results. The method discussed in this paper can deliver these realistic deformation effects without significant computational cost compared to other precise methods (i.e. finite elements). The simulation does not exhibit any oscillation or instability problems. Effects such as wrinkles and folds are inherent to the method and could be controlled by differentiating the stiffness constants of the surface springs. Runtime of the simulation was reasonably efficient. The Euler solver works in real-time; therefore it could be used during interactive design of models. Although IDE solvers were computationally extensive, a reasonable semi-realtime performance was achieved (Table 1). By, considering CGD method [7] instead of BCGD [14], and dropping volumetric gradient the IDE system could be solved even faster without significant loss of quality.

Table-1: Runtime analysis of the four models shown in figure 11

Object Name	Total Vtx.#	Total Spring#	Surface Face#	Euler FPS	IDE FPS
Handle	1432	6837	1861	16.62	0.66
Head	4397	24808	7791	5.14	0.23
Sphere	662	3535	320	33.25	1.42
Torus	1424	9542	1848	12.51	0.50

6. CONCLUSION

In this paper we presented an interesting method to model deformation such as deflation and inflation of solid objects for use in computer animation. The method could be used to model various effects, e.g. crash, explosion, inflation, deflation, character effects. Our method was efficient and demonstrated realistic visual effects. The problems of instability and oscillation inherent to mass spring system were eliminated. Future improvement may include a better PBM solver, faster mesh generation techniques and methods to collapse or inflate the structure. Finally, we would like to acknowledge Prof. B. Wywill for his valuable contribution to the project and NSERC and GEOIDE agencies for their funding.

7. References

[1] D. Baraff and A. Witkin. Large Steps in Cloth Simulation, Computer Graphics (proc. SIGGRAPH 98), 1998.

[2] D. Baraff. Dynamic simulation of Non-penetrating Rigid Bodies, PhD Thesis, Cornell, May 1992.

[3] D. Bourguignon and M. P. Cani, Controlling Anisotropy in Mass-Spring Systems, Computer Animation and Simulation 2000 (proc. 11th EuroGraphics Workshop), 2000.

[4] D. Terzopoulos, J. C. Platt, and A. H. Barr. Elastically deformable models. Computer Graphics (Proc. SIGGRAPH), 21:205-214, 1987.

[5] G. Golub and C. Van Loan. Matrix Computations. John Hopkins University Press, 1983.

[6] G. Hirota, R. Maheshwari, M. C. Lin, Fast Volume-Preserving Free Form Deformation Using Multi-Level Optimization, ACM Symposium on Solid Modeling and Applications, 1999.

[7] J. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-TR-94-125, Carnegie Mellon University, 1994.

[8] James H. Clark. Hierarchical geometric models for visible surface algorithms. *CACM*, 19(10):547-554, October 1976.

[9] M. Kass. An Introduction To Physically Based Modeling, chapter introduction to continuum dynamics for Computer Graphics (Proc. SIGGRAPH), pages 49-58, 1990.

[10] M. Kass. Can Introduction to physically based modeling, SIGGRAPH course Notes, ACM SIGGRAPH, 1995.

[11] P. Volino, M. Courchesne, and N. Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. Computer Graphics (SIGGRAPH), pages 137-144, 1995.

[12] R. Lobb. Physically Based Modeling. Course lecture notes, University of Calgary. http://pages.cpsc.ucalgary.ca/ ~jungle/richard/PBMCourse/ PBMNotes.pdf

[13] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. Numerical Recipies. Cambridge University Press, 1986.

[14] www.netlib.org, Bi-conjugate Gradient Method.