Real-Time Interaction Techniques for Meshless Deformation Based on Shape Matching

Alex Henriques and Burkhard Wünsche

Graphics Group, Department of Computer Science, University of Auckland, New Zealand
Email: burkhard@cs.auckland.ac.nz

Abstract

Meshless deformation based on shape matching is a new technique for simulating deformable objects which handles point-based objects and does not need connectivity information. The technique has been first presented in 2005 and is of interest to all fields which require fast, stable simulations which do not need to be physically correct. In particular the technique seems very suitable for use in virtual surgery applications and highly interactive real-time environments such as computer games. However, in contrast to traditional physically simulations, virtual environments require more complex and intuitive real-time interaction paradigms in order to increase the look and feel of the simulation and the immersive experience. We introduce techniques for picking, pushing and cutting objects simulated using meshless deformation based on shape matching. All interactions can be performed in real time, are unconditionally stable, easy to integrate into 3D rendering and game engines, and are easy-to-use and intuitive.

Keywords: deformable modeling, real-time simulation, interaction techniques, shape matching, virtual environments

1 Introduction

Advances in graphics hardware and rendering techniques have made it possible to develop realistic real-time interactive virtual environments. Typical examples are computer games, applications in architecture and urban design and to some extend visualisation applications in science, engineering and medicine. Despite these advances most of these applications still use models based on rigid-body physics due to their simplicity, easy control, and the existence of readily available fast simulation libraries such as ODE [1].

In 2005 meshless deformation based on shape matching was introduced as a new technique for simulating deformable objects. The technique does not require connectivity information for objects, is fast, unconditionally stable, and has low memory requirements. Consequently the technique might be very suitable for use in virtual surgery applications and highly interactive real-time environments such as computer games.

In this paper we introduce efficient interaction techniques, i.e. picking, pushing and cutting, for use with objects simulated using meshless deformation based on shape matching. The techniques can also be applied to other simulation methods but are particularly suitable for meshless deformation because when correctly implemented the technique is unconditionally stable. Furthermore since meshless deformation does not require connectivity information we do not have to worry about the geometry (e.g. triangle aspect ratio) of the mesh representing a deformable object, and we can use models represented by point clouds. Consequently all interaction techniques can be executed in real time and can be easily integrated into a traditional 3D rendering or game engine.

Section 2 introduces the meshless deformation technique in more detail, section 3 describes the interaction techniques available in the application we have developed. Finally, section 4 summarises our results, and section 5 concludes.

2 Meshless Deformation

"Meshless Deformations Based on Shape Matching" is a recently developed technique for dynamically simulating deformable objects [2]. The underling model is geometrically, as opposed to physically, motivated. It is unconditionally stable, does not require any pre-processing, and is simple to compute.

2.1 The Technique

Meshless deformation treats each object as a *point cloud*, or set of points, with no connectivity information required. To understand the basic idea,

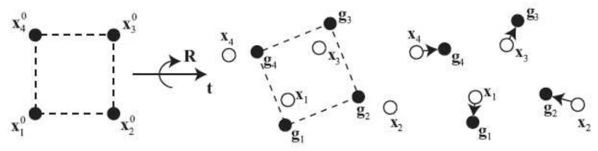


Figure 1: First, the original shape $\mathbf{x}_i^0$ is matched to the deformed shape $\mathbf{x}_i$. Then, the deformed points $\mathbf{x}_i$ are pulled towards the matched shape $\mathbf{g}_i$ (adapted from [2]).

let the initial configuration of points be $\mathbf{x}_i^0$, and the deformed configuration of points at some later time be $\mathbf{x}_i$. As a set of unconnected particles, each $\mathbf{x}_i$ responds to gravity and collisions, but no force acts to retain the overall object's shape. Meshless deformation's solution is to take the initial configuration $\mathbf{x}_i^0$, then move and rotate it as closely as possible onto the actual configuration $\mathbf{x}_i$ (see Figure 1). The rotated version of the initial configuration is now the set of *goal positions* $\mathbf{g}_i$ which minimise the least squares distance to actual positions. Each particle is pulled towards its goal position after each time step, retaining the object's initial shape.

The fundamental equation that finds the optimal transformation from $\mathbf{x}_i^0$ to $\mathbf{g}_i$ is that of "absolute orientation": given coordinates of a set of points as measured in two different Cartesian coordinate systems, find the optimal transformation between them [3]. To find this optimal transformation, the following sum is minimised.

$$\sum_i w_i \left(\mathbf{R}(\mathbf{x}_i^0 - \mathbf{t}_0) + \mathbf{t} - \mathbf{x}_i\right)^2$$

where $\mathbf{R}$ is a pure rotation matrix. In meshless deformation, $\mathbf{t}_0$ is the centre of mass of the initial configuration, and $\mathbf{t}$ is the centre of mass of the actual configuration. Müller et al. extend this equation by adding linear and quadratic matching; $\mathbf{R}$ is replaced by a linear deformation matrix $\mathbf{A}$, or a quadratic deformation matrix $\widetilde{\mathbf{A}}$. Thus, the goal positions can be not only a rotated version of the initial configuration, but a stretched, sheared, bent and twisted version. To produce a tendency towards the original undeformed state in linear and quadratic matching, $\mathbf{R}$ is combined with $\mathbf{A}$ or $\widetilde{\mathbf{A}}$ to produce a final deformation matrix $\mathbf{F}$.

$$\mathbf{F} = \beta\widetilde{\mathbf{A}} + (1 - \beta)\,\mathbf{R}$$

where β is a user defined constant between 0 and 1. Low β indicates a tendency mostly towards the rigid matched state, while high β indicates a tendency towards the quadratic match. The last important constant is α, which defines the proportional distance the points move towards their goal positions $\mathbf{g}_i$ every time step. When $\alpha = 1$, each point moves precisely to its goal position.

In summary, meshless deformation effectively transforms the original object by a matrix representing stretch, shear, bend and twist to find the closest match to the deformed object, then pulls the deformed object towards the goal positions represented by the transformed object.

2.2 Clusters

The primary disadvantage of meshless deformation is that goal positions are calculated by transforming the object with at best a quadratic deformation matrix, hence only 27 deformation modes are possible. Physical expressiveness may seem high, but significant limitations become apparent for objects more complicated than cubes and beach balls. These limitations are with respect to higher order deformation and local deformation. Consider two common objects as examples. A slithering snake might have two bends in it, which requires at least cubic deformations during animation, so it cannot be deformed with global quadratic equations. As a second example consider a sweatshirt with a hood. When using global deformations raising or lowering the hood is impossible to perform without bending the entire object. Note that quadratic equations do not have a compact support, i.e. are non-zero virtually everywhere.

To extend meshless deformation for local and higher order deformation, Müller et al. divide the set of particles into overlapping clusters, each with its own deformation modes and matrix. An entity consisting of multiple interacting clusters has a much greater range of deformation than an entity consisting of only one cluster. The shortcomings of the original implementation and our improvements for obtaining more physically realistic simulations are discussed in [4].

2.3 Evaluation

The advantages of meshless deformation are clear: it is fast and very easy to set up and tweak. The primary disadvantage of meshless deformation is that modes of deformation are quite limited. Clustering increases freedom, but is generally only well suited to objects with a small number of subparts, each of which deform at most quadratically. The only way to model more complex objects like cloth is to divide them into many fine grained clusters. But this is extremely inefficient and not very accurate – methods like mass-spring systems would be more suitable.

3 Interaction Techniques

In order to make a virtual world more realistic it is necessary to enable the user to interact with objects in a believable manner. Simulating both the look and feel of materials increases realism and the immersive experience. Furthermore advanced interactions are required for many applications such as virtual surgery simulations. In this section we introduce techniques for picking, constraining, pushing and cutting objects simulated using meshless deformation based on shape matching.

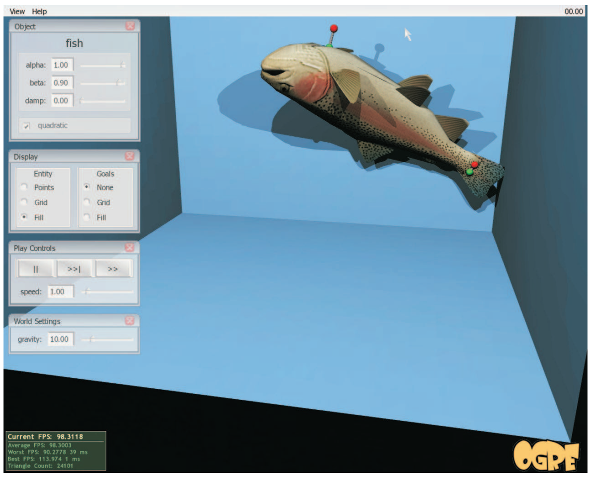
3.1 Picking

The main function of the picking mode is to grab objects and manipulate them with a spring force. The user can press the left mouse button to grab an object vertex, then drag the mouse around to control the direction of the spring force acting on that vertex. The spring force acts towards the position of the cursor represented by a red sphere. When the user moves the mouse, the red sphere moves along a plane facing the user. The mouse wheel can move the red sphere away from (mouse wheel up) or towards (mouse wheel down) the user. This moves the red sphere's plane of movement away from or towards the user, while keeping the plane's normal unchanged.

While dragging a spring force around, the user can release the left mouse button to stop the force and release the spring. Alternatively, the user can click the right mouse button to lock the force (i.e. the red sphere) in place. The user can then move around, change modes, or create a new spring force, while the original spring force remains in position. This makes it easy to "fix" an object in a deformed position. An example is shown in figure 2. To remove a locked in spring force, the user can click and drag on the red sphere to regain control of it then release the left mouse button, or press a key to remove all spring forces from every object.

3.2 Pushing

The main function of the pushing mode is to move objects by pushing them. A solid sphere follows the user's cursor in the same manner as the red sphere of the active spring force does in the picking mode above. Any objects colliding with the sphere undergo collision response forces. This is designed to mimic the user pushing objects around with his hand.



fish

alpha: 1.00
beta: 0.90
damp: 0.00
[✓ quadratic]

Entity: Points, Grid, [✓ Fill]
Goals: [✓ None], Grid, Fill

speed: 1.00

gravity: 10.00

```
Current FPS: 98.3118
Average FPS: 98.3003
Worst FPS: 90.2778 39 ms
Best FPS: 113.974 1 ms
Triangle Count: 24101
```

Figure 2: A deformed model of a trout fixed using two locked pick points.

3.3 Cutting

The main function of the cutting mode is to cut objects into separate pieces. The cursor turns into two cylinders designed to mimic a cutting instrument, e.g. a pair of scissors. To cut an object, the user moves the "scissors" to the appropriate position relative to the object, then holds down the left mouse button to begin the cutting process. The two "blades" of the scissors move closer together, and when they meet, every object the scissors intersect is severed along the plane of the scissors, creating two new separate objects.

To change the orientation of the scissors, the user can move the scissors towards him (mouse wheel down), away from him (mouse wheel up), or he can rotate the scissors about the y axis by holding down shift and dragging the left mouse button up or down.

3.3.1 Cutting Implementation

The cutting tool splits an object along a plane defined by the orientation of the scissors-shaped cursor. This simplifies the general cutting problem somewhat, as (a) we do not have to deal with partial cuts, and (b) the internal surface revealed by the cuts is always planar.

First, we define a *sever* operation which, taking an object o and a cutting plane c, removes all of o in c's positive halfspace and neatly seals up the exposed cross-section. The *cut* operation then consists of two *sever* operations: $sever(o, c)$ and $sever(o_{clone}, -c)$, where o_{clone} is a clone of o and $-c$ is c with normal reversed.

The first step of *sever* is to separate o's triangles into categories. Triangles with $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$

in the positive halfspace of the cutting plane are discarded. Triangles with $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ all in the negative halfspace of the cutting plane are kept. The remaining triangles straddle the cutting plane, and are cut along c to obtain a clean edge. These triangles have either exactly one or exactly two vertices in c's negative halfspace. The former kind are shortened to produce the clean edge; the latter kind are cut to form two subtriangles (see Figure 3).

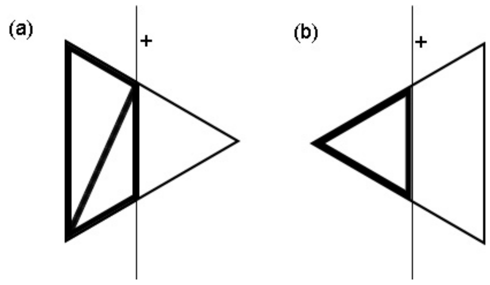


Figure 3: Triangles are made flush with the cutting plane's surface by creating two smaller triangles (a) or by shortening triangle edges (b).

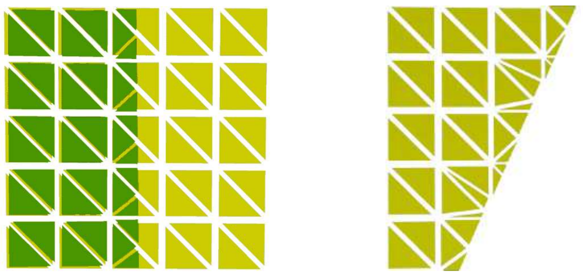


Figure 4: Triangulation for a cut down the centre of an object shown as overlay (left) and around a diagonal cutting plane.

When this process is carried out over every triangle, a neat edge aligned with the cutting plane is produced. Figure 4 shows the results for an axis-aligned cutting plane (left) and for a diagonal cutting plane (right).

The next step is to seal the exposed cross-section. A surface is created by triangulating the newly created vertices touching the cutting plane with a Delaunay triangulation algorithm (see Figure 5). The triangles tend to be irregularly shaped because only vertices around the edge of the surface are fed into the algorithm. With no vertices in the centre, each triangle needs to span edge to edge. An improvement to our method would add new vertices inside the edges before running the Delaunay triangulation algorithm, resulting in more consistently sized and shaped triangles.

After triangulation is performed, the object is tetrahedralised and divided into clusters again.

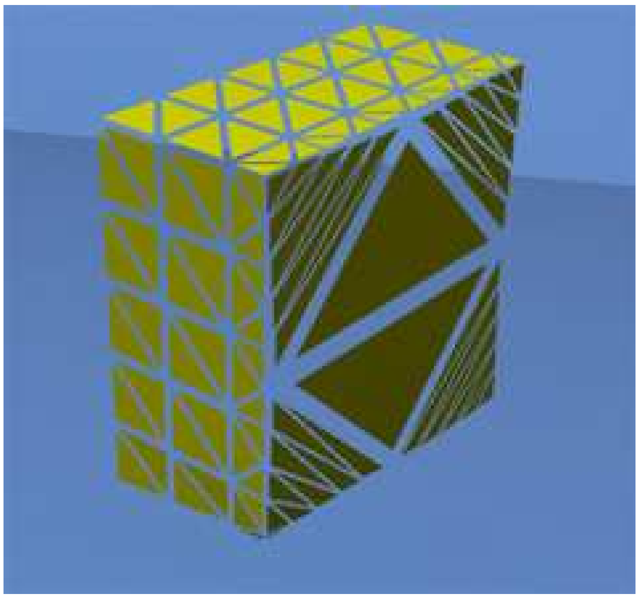


Figure 5: After a cut, the exposed internal hole is sealed up with a Delaunay triangulation.

Another possibility would be to keep what was left of the old tetrahedrons and clusters, but possible cluster degeneracy would need to be dealt with.

3.3.2 Problems

A requirement of the Delaunay triangulation algorithm is that the input is free of duplicate vertices (i.e. vertices with near identical positions). To achieve this, we simply create a separate list of duplicated cutting surface vertices, ensuring every new vertex added to the list has a unique position. The Delaunay triangulation is performed on this separate list. The edges of the surface produced, consisting of duplicate vertices, are thus sharp. This is desirable in must cutting applications. A rough cut could be easily achieved by displacing vertices along the cutting plane with a noise-based fractal function.

Since we use a 2D Delaunay triangulation algorithm we converted the 3D coordinates of the vertices of the cutting plane into 2D coordinates within this plane. The triangle vertex indices resulting from the algorithm can then be used to index the original 3D vertices. Unfortunately the particular implementation we used required that the input 2D points be sorted in order of increasing x. To preserve the mapping from 2D to 3D we created a data structure consisting of a 2D coordinate and an index into the 3D vertices' array, where the 3D vertex indexed is mapped to the 2D coordinate. The array of these data structures is then sorted into order of increasing x. The triangle resulting from the Delaunay triangulation index into this array, from which we can extract the correct index into the 3D vertex array.

3.4 Collision

Several types of methods are available for detecting and responding to collisions between deformable objects. These include bounded volume hierarchies, stochastic methods, distance fields, spatial subdivision, and image-space techniques [5].

Our application uses spatial hashing [6] and penetration depth estimation [7] techniques. We found that collision detection was a performance bottleneck however. No "best way" to perform collision detection for deformable objects has been decided on yet, and future research will improve this area.

4 Results

We have implemented a meshless deformation algorithm based on shape matching and developed a test bed for simulation applications and interaction techniques [4] based on the Ogre 3D graphics engine [8]. The user can pick, push or cut deformable objects in real-time. Simple objects with limited modes of deformation are simulated best. Objects composed of simple subcomponents are simulated well with clusters. Objects with a very high number of deformation modes, such as cloth, can not be simulated efficiently [9].

Usability. We found that all interaction techniques were intuitive and easy to use and that they significantly increased user satisfaction (enjoyment) when interacting with the virtual environment. This is a strong indication that the implemented techniques are a useful addition to highly-interactive immersive environments although more formal tests are necessary to confirm this observation. The pick application works best for objects which deform globally, such as the trout shown in figure 2, whereas simulating locally deformable objects requires us to use multiple clusters as demonstrated in figure 6. The cutting tool proved particularly popular with users and significantly increased the look and feel of interacting with 3D objects (see figure 7).

Ease of implementation. We found meshless deformation relatively easy to implement and integrate into the 3D rendering engine Ogre. There are only two main differences between current 3D engines and what is required for deformable object simulation. Firstly, rigid objects have static sharable meshes, while deformable objects require updates to individual vertex positions every time step on

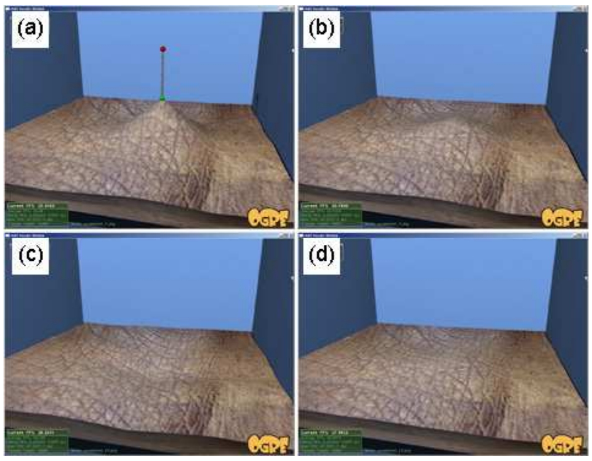


Figure 6: Behaviour of a 5×5 cluster skin patch in response to a user pick.

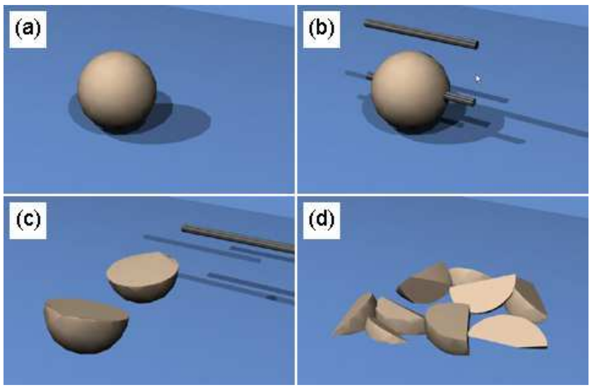


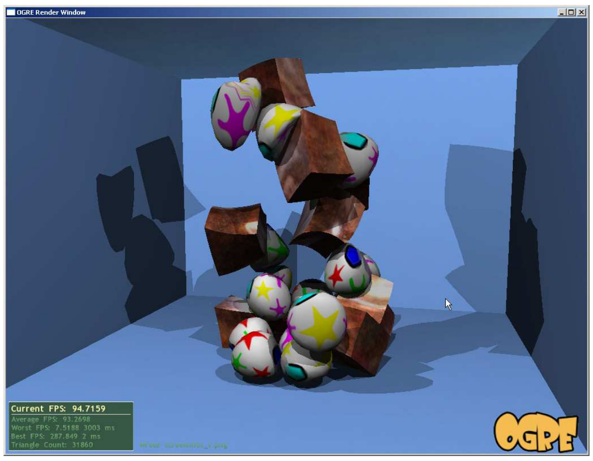
Figure 7: Cutting an object: (a) before cut, (b) during cut, (c) two resulting halves have rolled apart, (d) after further cuts.

their own mesh instance. Secondly, collision detection and response is a much slower, more difficult task for deformable objects.

Performance. Our environment is comparatively fast: We can simulate dozens of simple 32 tetrahedron objects with collisions in real-time and unconditional stability (see figure 8). Significantly better results could be achieved by optimising our algorithms and/or implementing them on the GPU.

Tweakability. The "gooeyness" and stiffness of each object can be easily modified using the α and β parameters. Further collision-response parameters can also be tweaked. The strength of surface area preservation can be specified with a force response curve. Volume preservation is automatic, but can be adapted to use a force response curve as well.

Disadvantages. The primary disadvantage of our environment is the lack of robust local deformation. For complex virtual surgery applications which of-



```
Current FPS: 94.7159
Average FPS: 93.2698
Worst FPS: 7.5188 3003 ms
Best FPS: 287.849 2 ms
Triangle Count: 31860
```

Figure 8: Large scale simulation of deformable objects.

ten require plausible localised deformation of an arbitrary region, our environment is less suitable. Also, even when simulation is visually plausible, it is usually not physically accurate.

5 Conclusion

We have implemented an improved algorithm for meshless deformation based on shape matching and we have presented several novel techniques to interact with these objects in a realistic and intuitive way. All interactions are performed in real time, are unconditionally stable and easy to integrate into 3D rendering and game engines. Informal user studies suggested that all interaction techniques significantly increase user satisfaction (enjoyment) when interacting with the virtual environment. Cutting could also easily be adapted to serve as a fracturing implementation.

Disadvantages are that performing local deformations requires models with sufficiently small clusters which is often not efficient. Also more improvements are necessary in order to apply our techniques to large scale objects and scenes. The cut operation so far can only perform full cuts and does not support local incisions which would be useful for a virtual surgery application or games where the player might want to slash an opponent.

In summary we believe that the implemented techniques are a useful addition to many highly-interactive immersive environments where speed and a more immersive feel are required but physical accuracy is not important.

6 Future Work

When cutting an object many new triangles are created along the cutting plane. Currently we give each new triangle unique vertices which can result in an uneven look when using different vertex normals. In future we intend to utilise a hash table for vertices and normals similar to the one introduced by Wyvill et al. [10].

References

[1] "Open Dynamics Engine home page." `http://www.ode.org`.

[2] M. Müller, B. Heidelberger, M. Teschner, and M. Gross, "Meshless deformations based on shape matching," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 471–478, 2005.

[3] B. Horn, "Closed-form solution of absolute orientation using unit quaternions," *Journal of the Optical Society of America A*, vol. 4, no. 4, pp. 629–642, 1987.

[4] A. Henriques, "Meshless deformation for real-time soft tissue simulation," BSc Honours dissertation, University of Auckland, 2006. (to be published in Oct 2006).

[5] M. Teschner, S. Kimmerle, G. Zachmann, B. Heidelberger, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnetat-Thalmann, and W. Strasser, "Collision detection for deformable objects," in *Eurographics State-of-the-Art Report (EG-STAR)*, pp. 119–139, Eurographics Association, Eurographics Association, 2004. `http://www-evasion.imag.fr/Publications/2004/TKZHRFCFMS04`.

[6] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross, "Optimized spatial hashing for collision detection of deformable objects," 2003.

[7] B. Heidelberger, M. Teschner, R. Keiser, M. Muller, and M. Gross, "Consistent penetration depth estimation for deformable collision response," *Proceedings of Vision, Modeling, Visualization VMV04, Stanford, USA*, pp. 339–346, 2004.

[8] "OGRE 3D home page." `http://www.ogre3d.org`.

[9] J. Rubin, "A framework for interactive and physically realistic cloth simulation," 780 project report, University of Auckland, Feb. 2006. `http://www.cs.auckland.ac.nz/~burkhard/Reports/2005_SS_JonathanRubin.pdf`.

[10] G. Wyvill, C. McPheeters, and B. Wyvill, "Animating soft objects," *The Visual Computer*, vol. 2, pp. 235 – 242, Aug. 1986.