IMPROVED MESHLESS DEFORMATION TECHNIQUES FOR REAL-TIME INTERACTIVE COLLABORATIVE ENVIRONMENTS

Alex Henriques and Burkhard Wünsche
Graphics Group, Department of Computer Science, University of Auckland, New Zealand
burkhard@cs.auckland.ac.nz

Keywords: Deformable models, real-time simulation, interaction techniques, shape matching, virtual environments.

Abstract: Meshless deformation based on shape matching is a new technique for simulating deformable objects without requiring mesh connectivity information. The approach focuses on speed, ease of use and stability at the expense of physical accuracy. In this paper we introduce improvements to the technique that increase physical realism and make it more suitable for use in interactive real-time environments such as games and virtual surgery applications. We also present intuitive real-time interaction techniques for picking, pushing and cutting objects simulated using meshless deformation based on shape matching. For deformable collision detection and response, we present a new method for surface meshes based on previous volumetric methods.

1 INTRODUCTION

Advances in graphics hardware and rendering techniques have made real-time interactive virtual environments increasingly realistic. In the past few years such applications and in particular computer games have started to incorporate rigid-body physics, which are easily controlled and readily simulated using fast libraries like ODE (Smith, 2006). As processing power increases further and physics cards are introduced, the natural progression is to include real-time deformable object simulation into virtual environments.

In 2005 meshless deformation based on shape matching was introduced as a new technique for simulating deformable objects (Müller et al., 2005). The technique is fast, easy to use, unconditionally stable, and has low memory requirements. These factors make the technique particularly interesting for virtual surgery applications and highly interactive real-time environments like computer games.

In this paper we present improvements to this technique. Soft caps on surface area expansion are introduced to create a natural limit on deformations. A modification is introduced to prevent inverted object states. We also present efficient interaction techniques, i.e. picking, pushing and cutting, for use with objects simulated using meshless deformation based on shape matching. Finally, we adapt for use with surface meshes a tetrahedral based collision detection and response method. All methods and interaction techniques can be executed in real time and easily integrated into traditional 3D rendering or game engines.

Section 2 introduces the meshless deformation technique in more detail, while section 3 details our improvements to the technique. Section 4 describes the interaction techniques available in the application we have developed, and section 5 describes the collision detection and response methods we implemented. Finally, section 6 summarizes our results, and section 7 concludes.

2 MESHLESS DEFORMATION

"Meshless Deformations Based on Shape Matching", or *meshless deformation* for short, was recently developed as a technique for dynamically simulating deformable objects (Müller et al., 2005). The approach is geometrically motivated, requires no preprocessing, and is both simple to compute and unconditionally stable.

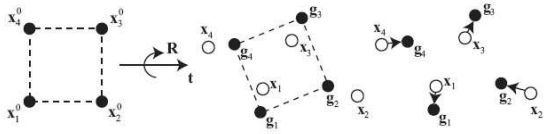

Figure 1: First, the original shape $\mathbf{x}_i^0$ is matched to the deformed shape $\mathbf{x}_i$. Then, the deformed points $\mathbf{x}_i$ are pulled towards the matched shape $\mathbf{g}_i$ (adapted from (Müller et al., 2005)).

2.1 The Technique

In meshless deformation, each object is represented by a set of points, or *point cloud*. No connectivity information is required. Each point in the point cloud moves and responds to forces independently of other points, while meshless deformation ensures the object retains its overall shape. Let the initial configuration (i.e. positions) of points be $\mathbf{x}_i^0$, and the deformed configuration of points at some later time be $\mathbf{x}_i$. To preserve the object's shape, Meshless Deformation moves and rotates the initial shape $\mathbf{x}_i^0$ as closely as possible onto the actual shape $\mathbf{x}_i$ (see figure 1). The translated and rotated initial shape now defines the set of *goal positions* $\mathbf{g}_i$. Every timestep, each point is moved a fraction α of the way towards its goal position. This gives the point cloud a tendency to preserve its initial shape.

The optimal transformation from $\mathbf{x}_i^0$ to $\mathbf{g}_i$ minimizes the sum of the squared distances between $\mathbf{g}_i$ and $\mathbf{x}_i$. The problem is the same as 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 (Horn, 1987). This corresponds to minimizing the following sum.

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

where $\mathbf{R}$ is a pure rotation matrix. In meshless deformation, the weights are the point masses, $\mathbf{t}_0$ is the centre of mass of the initial shape, and $\mathbf{t}$ is the centre of mass of the current shape. This equation can be extended to allow linear and quadratic matching by replacing $\mathbf{R}$ with a linear deformation matrix $\mathbf{A}$ or quadratic deformation matrix $\widetilde{\mathbf{A}}$. Linear deformations allow stretch and shear, while quadratic deformations additionally allow bends and twists. A tendency towards the undeformed state is introduced by combining $\mathbf{A}$ or $\widetilde{\mathbf{A}}$ with $\mathbf{R}$, resulting in a final deformation matrix $\mathbf{F}$.

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

where β is a user defined constant between 0 and 1. When β is low, the tendency is largely towards a rigid undeformed state; when β is high, the tendency is more towards the quadratic match, resulting in a softer more deformable object.

2.2 Clusters

Because meshless deformation matches a quadratically deformed version of the initial object, deformation is limited to combinations of stretch, shear, bend and twist over the entire object. This means local deformations – those deforming only one part of an object – are impossible. Higher order deformations, e.g. the cubic deformation of a string given two bends, are also impossible.

As a partial solution to these limitations, Müller et al. divide the set of particles into overlapping clusters with separate deformation matrices. This can greatly increase the range of deformation. However, applications are largely limited to objects with mostly independent subparts that deform only quadratically. More complex entities like cloth need to be divided into finely grained clusters for plausible simulation. But, this is inefficient and inaccurate, and better performed by mass-spring systems.

3 IMPROVEMENTS

3.1 Surface Area Preservation

Meshless deformation matches a goal configuration to the deformed point cloud as closely as possible. However, the goal configuration matched frequently has greater or lesser volume than the original object, which is generally undesirable. To preserve volume, meshless deformation scales the deformation matrix such that the goal configuration's volume is identical to the original object's volume. The problem with such blind scaling is that when for example a force squashes the object along one dimension, the volume of the goal configuration pre-scaling can be very small. To preserve volume the scaling factor must be very large to compensate, and the other two dimensions are scaled up drastically in response as illustrated in figure 2.

3.1.1 Suggested Solutions

If an airtight balloon filled with water were thrown gently at a wall, the volume of water inside would remain constant. But the balloon would not behave as in figure 2, because of resistance to *surface area*

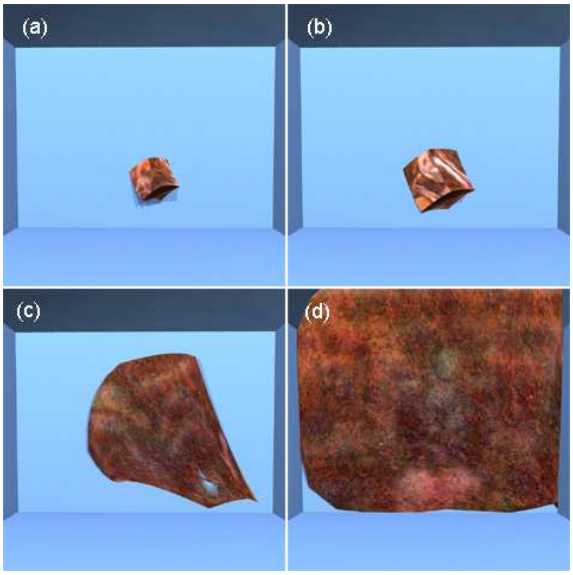

Figure 2: Volume preservation without surface area preservation results in unrealistic surface expansion as a cube is pushed into a wall.

stretch. Clearly then, a method to constrain surface area is needed. Some algorithms use mesh-based explicit surface area preserving forces (Teschner et al., 2004a). For meshless deformation, possible solutions include the following:

1. Limit forces applied to objects. If the vertices are not subject to large forces, they will not move so far out of their original configuration that blind volume-preservation scaling will produce such extreme surface area changes.
2. Limit the maximum velocities of vertices. As with 1, if the vertex velocities are constrained to within a maximum, extreme configurations will be more difficult to produce.
3. Limit α and β. If α is large, the vertices will return quickly to their goal positions, lessening the likelihood of extreme configurations being produced. If β is small, the tendency of the cube to return to an undeformed state will override the quadratic transformation if it matches an extreme configuration.
4. Have vertices propagate a constraint force through to adjacent vertices.
5. Limit the transformation matrix somehow so that it doesn't match extreme configurations.

1, 2, and 3 used in various combinations are quite successful in combating this problem. 4 is an interesting option, but would require connectivity information to be implemented efficiently. These methods also require tightly regulated parameters, so by definition cannot be unconditionally stable. 5 on the other hand is simple to implement, efficient, and can achieve unconditional stability.

The simplest way to constrain surface area using 5 is to cap the Frobenius norm of the linear deformation matrix $\mathbf{A}$.

$$\|\mathbf{A}\|_F^2 = \sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2$$

Drastic increases in surface area are caused by large stretch or shear values, which are the contributors to $\|\mathbf{A}\|_F^2$. Therefore, by limiting $\|\mathbf{A}\|_F^2$, we limit stretch and shear. If we want to cap the amount of quadratic deformation for visual reasons, the following methods can also be trivially extended from $\mathbf{A}$ to $\widetilde{\mathbf{A}}$. In the subsequent computations, we use the term $\|\mathbf{A}\|$ as shorthand for the Frobenius norm.

3.1.2 Methods of Clamping

There are several ways to clamp $\|\mathbf{A}\|$; here are three.

1. Let rows of $\mathbf{A}$ be termed $\mathbf{r}_i$. If any $\|\mathbf{r}_i\|^2$ exceeds a user selected c_{max}, scale $\mathbf{r}_i$ by x such that $\|x\mathbf{r}_i\|^2 = c_{max}$.
2. Cap the magnitude of $\mathbf{A}$ at c_{max}. To do this, if $\|\mathbf{A}\|^2 > c_{max}$ update $\mathbf{A}$ as

   $$\mathbf{A} \leftarrow \gamma\mathbf{A} + (1-\gamma)\mathbf{R}$$

   where $\mathbf{R}$ is the rotation matrix from equation 1 and γ is derived from the solution to the quadratic equation

   $$\|\gamma\mathbf{A} + (1-\gamma)\mathbf{R}\|^2 = c_{max}.$$

   Note that because of the choice of c_{max} the function is monotonically increasing when $0 \leq \gamma \leq 1$ and hence the quadratic equation has exactly one solution. The final matrix $\mathbf{F}$ in equation 1 is then calculated as:

   $$\begin{aligned}\mathbf{F} &= (\gamma\mathbf{A} + (1-\gamma)\mathbf{R})\beta + \mathbf{R}(1-\beta)\\ &= \gamma\beta\mathbf{A} + \beta\mathbf{R} - \gamma\beta\mathbf{R} + \mathbf{R} - \beta\mathbf{R}\\ &= \gamma\beta\mathbf{A} + (1-\gamma\beta)\mathbf{R}.\end{aligned}$$

   hence γ is a simple beta modifier, i.e., it makes the deformation more rigid.
3. As a cheaper imitation of 2, simply set

   $$\gamma = \frac{c_{max}}{\|\mathbf{A}\|^2}.$$

The first method works well, but restricts deformation along each axis regardless of deformation in the other axes. The second and third methods on the other hand restrict the sum of deformations along all axes, so maximum deformation along one axis prevents further deformation along the other axes. The appropriate method would seem to depend on the physical properties of the object. Visually we could not distinguish between methods 2 and 3.

3.1.3 Further Extensions

These three methods solve the blow-up problem well, but introduce a slight problem with visual plausibility. A soft object falling to the ground will flatten to the point where the deformation magnitude ϕ is capped, then deformation will jerk to a stop. To solve this we suggest a "soft" cap rather than a hard one. This would take the form of a monotonically increasing function f such that for an intermediate threshold c and a maximum threshold m,

$$f(\phi) = \begin{cases} \phi & \phi \leq c \\ < m & \phi > c \end{cases}$$

In other words, an object with deformation magnitude ϕ exceeding the soft cap c will have ϕ reduced towards c. To prevent unrealistically large deformations, m indicates a hard cap below which ϕ will always be reduced. Here is an example function:

$$f(\phi) = \begin{cases} \phi & \phi \leq c \\ m - \left(\frac{c}{\phi}\right)(m-c) & \phi > c \end{cases}$$

3.2 Inversion

Recall that the central equation to be minimized in meshless deformation is

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

Müller et al. present the most referenced solution to this problem (referred to as that of absolute orientation) derived in (Horn, 1987). In his paper however Horn mentions that the **R** obtained may be a reflection, rather than a rotation, in cases where reflection provides a better fit.

In traditional photogrammetric applications of the absolute orientation problem, the data may seldom be corrupted enough to produce a reflective **R**. When applied to physical objects undergoing large deformations however, the vertices can frequently be deformed enough that the optimal **R** is a reflection. The inverted object produced is an unacceptable result for homogeneous objects, because it would require massive self-penetration.

3.2.1 Determinant Cube Root Solution

Müller et al. do not specifically mention what to do when a reflective **R** is produced. The only related comment is made when discussing volume preservation of the linear transformation matrix **A**:

> To make sure that volume is conserved, we divide **A** by $\sqrt[3]{det(\mathbf{A})}$ ensuring that $det(\mathbf{A}) = 1$.

When $det(\mathbf{A})$ is negative, $\sqrt[3]{det(\mathbf{A})}$ is also negative. The subsequent division results in an **A** that produces a non-inverted, volume preserving goal position configuration. This configuration is obtained however by a simple reflection of each optimal position through the origin. **A** no longer describes a minimization of goal position with respect to vertex position. Thus the goal positions will tend to be far away from their respective vertex positions, and the integration step will produce large velocities. The result is a blowup.

When taken literally, the method deals with an inverted goal match by producing a blowup. If $\sqrt[3]{det(\mathbf{A})}$ is constrained to its absolute value, the method results in a stable, inverted object configuration. Neither result is acceptable.

3.2.2 Modified R Extraction Solution

Rather than make a modification to the transformation matrix after **R** has been calculated, a modified algorithm is proposed by Umeyama (Umeyama, 1991) that strictly produces an optimal rotation matrix **R**. Implementing this modification involves only a simple addition to the singular value decomposition solution method of Arun et al. (Arun et al., 1987).

This method solves the inversion problem, but only partially. While **R** will always be a rotation, **A** may still contain a reflection (assuming the absolute value of $\sqrt[3]{det(\mathbf{A})}$ is used). The final transformation matrix $\mathbf{F} = \beta\mathbf{A} + (1-\beta)\mathbf{R}$ then will always have a tendency towards a non-inverted configuration. But with β close to 1, the tendency will be slow, and may produce physically implausible results. Ideally **A** would calculated in a manner that never produced reflections–this remains for future work.

4 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.

The picking mode allows the user to grab and manipulate any object vertex with a spring force. The spring force acts towards the cursor position (represented by a red sphere), and can also be moved back and forth along the camera's look direction using the mousewheel. Spring forces can be locked in place, allowing the user to change modes or create new spring forces. In this manner objects can readily be "fixed" in deformed positions (see figure 3). This mode is useful for precisely manipulating an object's position, deformation and orientation.

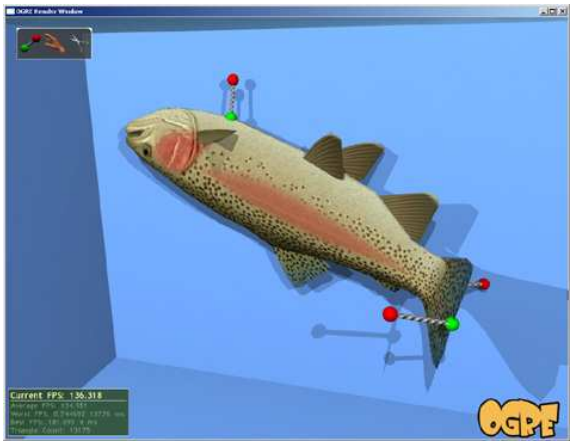

Figure 3: A deformed model of a trout fixed using two locked pick points.

The pushing mode allows the user to manipulate objects with a pushing force. The cursor position is represented in 3D space as in the picking mode, and collision response forces are applied to any objects near the cursor. This mode is useful for moving several objects at once, as when clearing a path or area.

4.1 Cutting

The cutting mode allows the user to sever objects into separate pieces. The user controls a cutting implement which can be arbitrarily orientated in 3D space. When the user holds down the left mouse button, the two "blades" of the cutting implement converge, and any intersecting objects are severed along the plane of the cutting implement.

The cutting operation takes as input an object and a cutting plane, and splits the object along that cutting plane. This simplified version of the general cutting problem does not allow partial cuts, and always reveals a planar internal surface. We found it necessary to disallow partial cuts for two reasons: Firstly, partial cuts would require complex representations of internal structures in order to correctly simulate subsequent deformations. While objects could be defined with an interior structure this would limit applications and efficiency. Alternatively an external surface can be dynamically created (imagine cutting jelly), but this can make cuts difficult to control precisely. The second and more important problem with partial cuts is that the area surrounding a partial cut needs to undergo complex deformations in order to be represented realistically. An edge can sag, fray, and be deformed independently of the edge on the opposite side of the cut. Meshless deformation–which allows only quadratic deformation over the whole cluster–cannot represent such complex and subtle deformations.

4.1.1 Cutting Implementation

Our cutting implementation cuts an object along a plane as follows. First, all triangles completely in the plane's positive halfspace are discarded, while all triangles completely in the plane's negative halfspace are kept. Triangles straddling the cutting plane are divided into smaller subtriangles, creating a neat edge aligned with the cutting plane. Finally, the newly created vertices touching the cutting plane are fed into a Delaunay triangulation algorithm, which seals up the exposed cross-section (see figure 4). Possible future improvements include adding vertices inside the exposed cross-section to allow for more regular triangulation.

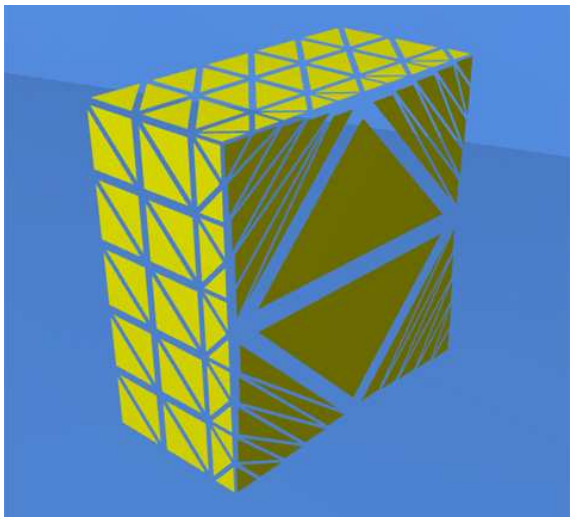

Figure 4: After a cut, the exposed internal hole is sealed up with a Delaunay triangulation.

5 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 (Teschner et al., 2004b).

The collision detection and response techniques used by Müller et al. (Müller et al., 2005) involve spatial hashing (Teschner et al., 2003) and penetration depth estimation (Heidelberger et al., 2004) on tetrahedral meshes. We give a brief overview of the method here.

1. Using a spatial hashing approach, each point is classified as colliding if it intersects a tetrahedron.
2. Colliding points are classified as *border points* if they are connected by an edge to a non-colliding point.
3. For each border point a penetration depth and direction is calculated based on connected edges' *intersection points* and corresponding surface normals.
4. Penetration depths and directions are propagated inwards to the remaining colliding points in a breadth-first manner.

One major disadvantage of this method is that it requires a tetrahedral mesh. Many applications, for example games, use only surface meshes. With this in mind we adapted the method for use with surface meshes.

In steps 1 and 2, we need to classify colliding and border points without tetrahedra. Using spatial hashing, we test each edge for intersection with nearby surface mesh triangles. On intersection, we classify the edge point in the triangle's positive halfspace as non-colliding, and the edge point in the triangle's negative halfspace as colliding. We also record the length along the edge of the intersection point. If the same edge intersects multiple triangles, the two edge points' classifications are with respect to their closest triangle along the edge. The colliding points so classified are the border points. Remaining points are classified as colliding if they can be reached from a border point without passing through a non-colliding point.

Step 3 remains the same. Step 4 requires significant modification however – figure 5b shows the penetration depths and directions calculated without modification. The problem here is that without a tetrahedral mesh, border points only exist on the surface of the mesh around the intersecting triangles, and not inside the mesh around deeply penetrated areas.

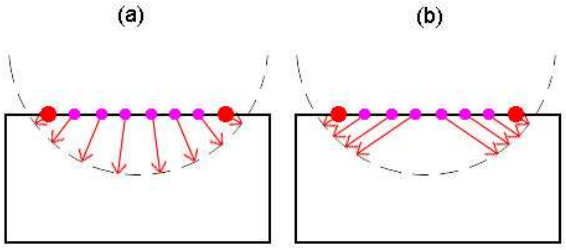

Figure 5: (a) Ideal response forces. (b) Response forces created using a consistent penetration depth estimation technique.

This results in unrealistic propagated penetration directions. Rather than use propagation, for each non-border colliding point we simply calculate the penetration direction as a weighted sum of each border point's penetration direction, where the weights are inversely proportional to the number of edges in the shortest edge path between the colliding point and border point. To calculate penetration depth, we find the length of a ray cast from the colliding point to the surface along the penetration direction. The results are as in figure 5a.

Compared to the original tetrahedral method, our surface mesh collision technique is slower and subject to classification errors and erratic behaviour. While the method works for simple applications further research is necessary to make it more stable and hence suitable for computer games and similar applications where tetrahedral meshes are not available.

6 RESULTS

We have developed a framework for testing interactive simulation environments and implemented within it meshless deformation based on shape matching together with our improvements. The user may pick, push or cut deformable objects in real time.

We found that simple objects with limited modes of deformation are simulated best, while objects composed of simple subcomponents are simulated well with clusters. Objects with a very high number of deformation modes, such as cloth, cannot be simulated efficiently (Rubin, 2006).

Due to the improvements implemented, extreme forces and deformations no longer produce stable inversions or erratic behaviour due to temporary inversions. Further, large forces no longer result in surface area blowups, allowing the use of arbitrary stiffness (α and β) values, forces and speeds. Objects that are particularly soft or moving at great speeds no longer

jerk to a sudden stop when their deformations exceed a certain amount, instead gradually reaching a maximum deformation between soft and hard caps.

Our experiments show that a variety of different objects can be simulated plausibly. The simplest example is a beach ball. We also found that a trout was simulated quite well, being quite rubbery, with skin not subject to local deformations. More complicated objects such as a rubber torus were also simulated well. This is probably because while complicated, the object is not one users have a lot of experience with, so deformation that isn't physically accurate can readily be seen as plausible. Contrast this with a realistic human face model we experimented with. The deformations of a human face are something we are intimately familiar with, and any deviation from physical accuracy can be easily noticed. We also found that to achieve an acceptable range of deformations corresponding to the muscle groups of the face, clusters needed to be divided very precisely – we had to implement a special export tool to allow precise cluster specification in a 3D modeling program. Even then, we found clusters very difficult to manipulate into giving plausible facial animations, and boundaries between clusters were often noticeable. The final type of object we tested was skin. Skin requires local deformation at arbitrary points. To achieve this the skin needed to be split up into many small clusters. The results were plausible, however with many clusters efficiency is low. Cloth simulation led to similar results.

Usability. Informal user testing indicates that our environment and all our interaction techniques were intuitive and easy to use. The ability to push, pull, fix and cut deformable colliding objects significantly increased user enjoyment.

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 timestep on 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). Suitable speed for simple virtual surgery applications could be achieved by optimising our algorithms and/or implementing them on the GPU. However where deformable behaviour is less important, for example in most games, we predict it will be at least several years before deformable object simulation is the best marginal use of processing power.

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

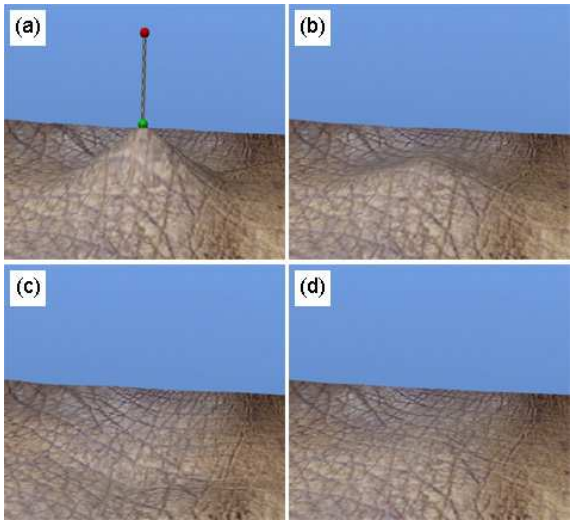

Figure 6: Behaviour of a 5×5 cluster skin patch in response to a user pick.

7 CONCLUSION

We have implemented an improved algorithm for meshless deformation based on shape matching. Our improvements include soft capped surface area preservation, and the prevention of inverted states. We have also implemented several interaction techniques allowing users to interact with objects realistically and intuitively. Collision detection and response have been implemented based on spatial hashing and accurate penetration depth estimation techniques. We have also adapted the collision method for use with triangular surface meshes, for applications such as games where tetrahedral meshes are not available. Informal user testing indicates that users find our envi-

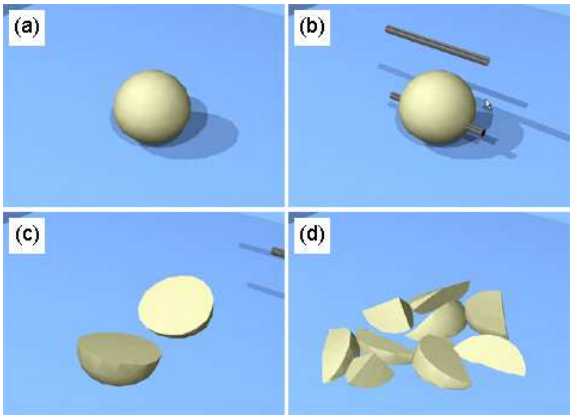

Figure 7: Cutting an object: (a) during cut, (b) immediately after cut, (c) two resulting halves have rolled apart, (d) after further cuts.

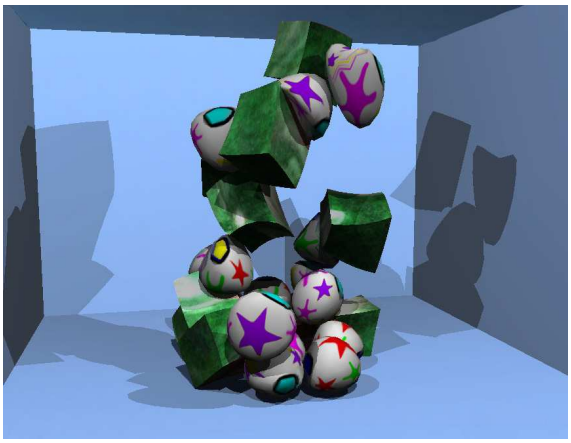

Figure 8: Large scale simulation of deformable objects.

ronment significantly more enjoyable and immersive than a comparable rigid body physics environment.

Disadvantages include that simulating local deformations requires division of the object into fine grained clusters, which can be inefficient. Precise cluster divisions can also be difficult to specify. For large scale objects and scenes, efficiency improvements are necessary. Finally, the cut operation does not support partial cuts or incisions, which would be useful for virtual surgery applications or games.

In summary, we believe that the techniques implemented have promising potential as applied to a virtual surgery simulator, games, or any other environment where speed and immersive interactions are required but physical accuracy is not.

8 FUTURE WORK

One major problem limiting meshless deformation's use in some applications is the lack of robust local deformation. One avenue of investigation might be to integrate a mass-spring system, which is usually disabled, but where user picks activate mass-spring behaviour in the pick's local region. Mass-spring areas around a partial cut or incision could similarly be activated. For larger cuts, but not complete severances, a method of dynamically partitioning new clusters may be possible that would allow "flapping" behaviour, similar to a tennis ball nearly cut in half with both halves "talking" like a mouth.

REFERENCES

Arun, K., Huang, T., and Blostein, S. (1987). Least-squares fitting of two 3-D point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):698–700.

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

Horn, B. (1987). Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642.

Müller, M., Heidelberger, B., Teschner, M., and Gross, M. (2005). Meshless deformations based on shape matching. *ACM Trans. Graph.*, 24(3):471–478.

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

Smith, R. (2006). Open Dynamics Engine home page. `http://www.ode.org`.

Teschner, M., Heidelberger, B., Mueller, M., Pomeranets, D., and Gross, M. (2003). Optimized spatial hashing for collision detection of deformable objects.

Teschner, M., Heidelberger, B., Muller, M., and Gross, M. (2004a). A versatile and robust model for geometrically complex deformable solids. *Computer Graphics International, 2004. Proceedings*, pages 312–319.

Teschner, M., Kimmerle, S., Zachmann, G., Heidelberger, B., Raghupathi, L., Fuhrmann, A., Cani, M.-P., Faure, F., Magnetat-Thalmann, N., and Strasser, W. (2004b). Collision detection for deformable objects. In *Eurographics State-of-the-Art Report (EG-STAR)*, pages 119–139. Eurographics Association, Eurographics Association.

Umeyama, S. (1991). Least-squares estimation of transformation parameters between two point patterns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 13(4):376–380.