Improved Meshless Deformation Techniques for Plausible Interactive Soft Object Simulations

Alex Henriques and Burkhard Wünsche

University of Auckland, Dept. of Computer Science, Graphics Group, Private Bag 92019, Auckland, New Zealand
{ahen045,burkhard4u}@gmail.com,
WWW home page: http://www.cs.auckland.ac.nz/~bwue001

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.

Key words: deformable models, real-time simulation, interaction techniques, shape matching, virtual environments

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 efficient libraries like ODE [12]. As processing power increases further and physics cards are introduced, the natural progression is to include real-time deformable object simulation into virtual environments.

Existing solutions can be divided into pre-animations, kinematic methods, geometric methods, and physically-based methods. Pre-animated simulations are achieved by modelling a limited range of interactions using a human animator, motion capture, or more complex physically-based techniques. The simulations are stored in movie or 3D animation formats and are triggered when the user performs certain predefined operations such as cutting in a specified region. This type of simulation does not allow arbitrary interactions, but is fast and can be achieved using game engines, flash animations and other widely available tools.

Kinematic methods do not represent material properties and forces and include direct mesh manipulation and implicit surfaces. Free-form deformation associates object coordinates with locations in a surrounding mesh of control points [11]. If the control mesh is deformed the object deforms with it. The technique is quite simple, but offers limited forms of manipulations, and makes it difficult to implement cutting operations.

We use the term geometric methods for techniques which use physical properties to kinematically deform regions of an object's geometry. Delp et al. [4] represent different types of tissue using polygonal surface meshes. When interacting with the tissue, the nearest contact point to the surgical instrument is calculated. Affected vertices in a predefined area around this contact point are deformed using a polynomial interpolation.

Physically-based methods include mass-spring systems and finite element methods. Mass-spring systems represent soft objects as a set of points where neighbouring points are connected by springs which simulate the elasticity of the material. The method is easy to implement and cutting can be modeled by removing springs [8]. Finite element simulations model the volumetric nature of soft objects and describe its deformation behaviour using a set of differential equations incorporating material parameters [2]. The resulting simulations are physically realistic but are computationally expensive.

While the above presented methods have been continuously improved in recent years, they are still either computationally expensive or only offer a limited number of interactions and are difficult to implement and integrate into commonly available graphics engines. "Meshless deformation based on shape matching"', or *meshless deformation* for short, is a new technique for simulating deformable objects [9]. 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. 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 summarises our results, and section 7 concludes.

2 Meshless Deformation

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 $\mathbf{g}_i$ minimises 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,

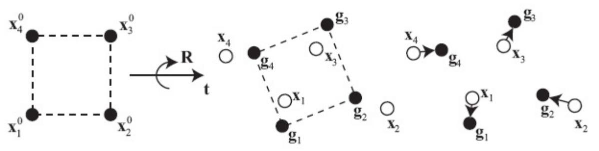


Fig. 1. First, the original shape $\mathbf{x}_i^0$ is matched to the deformed shape $\mathbf{x}_i$. Then, the deformed points $\mathbf{x}_i$ are pulled towards the matched shape $\mathbf{g}_i$ (adapted from [9]).

find the optimal transformation between them [7]. 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 $\mathbf{F}$.

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

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.

2.1 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.

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 frequently has a different 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 other two dimensions are scaled up drastically in response as illustrated in the top row of figure 2.

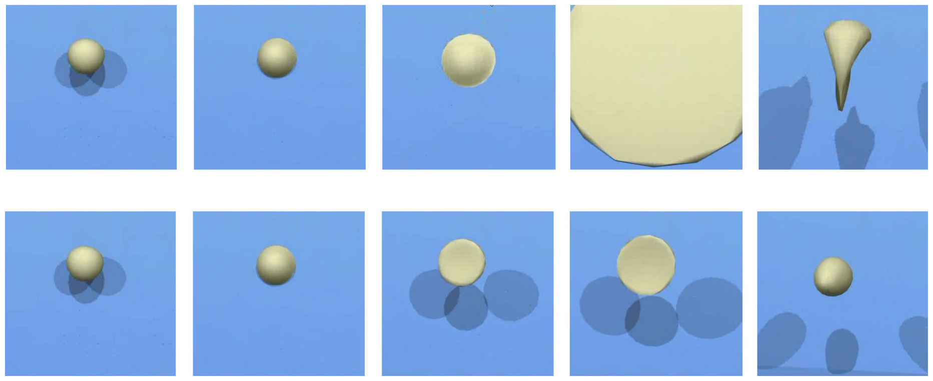


Fig. 2. A ball smashed with high force against a wall. The original meshless deformation algorithm (top) is unconditionally stable but leads to unrealistic intermediate configurations. Our modification (bottom) limits the scale and shear factors and results in more realistic looking results.

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 the top row of figure 2, because of resistance to *surface area* stretch. Clearly then, a method to constrain surface area is needed. Some algorithms use mesh-based explicit surface area preserving forces [14]. 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, this will make the occurrence of extreme configurations much less likely.
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 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 **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 **A** to $\widetilde{\mathbf{A}}$. In the subsequent computations, we use the term $\|\mathbf{A}\|$ as shorthand for the Frobenius norm.

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

1. Let rows of **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 **A** at c_{max}. To do this, if $\|\mathbf{A}\|^2 > c_{max}$ update **A** as

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

   where **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 **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.

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 in 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, the more the deformation magnitude ϕ exceeds the soft cap c, the more it will be reduced such that it never exceeds the hard cap m. 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 minimised 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 by Horn [7]. In this paper, however, Horn mentions that the $\mathbf{R}$ 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 $\mathbf{R}$. When applied to physical objects undergoing large deformations, however, our experiments showed that the vertices frequently are deformed enough that the optimal $\mathbf{R}$ is a reflection. The inverted object produced is an unacceptable result for homogeneous objects, because it would require massive self-penetration.

Determinant Cube Root Solution Müller et al. do not specifically mention what to do when a reflective $\mathbf{R}$ is produced. The only related comment is made when discussing volume preservation of the linear transformation matrix $\mathbf{A}$:

> To make sure that volume is conserved, we divide $\mathbf{A}$ 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 $\mathbf{A}$ 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. $\mathbf{A}$ 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.

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 [16] 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. [1].

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

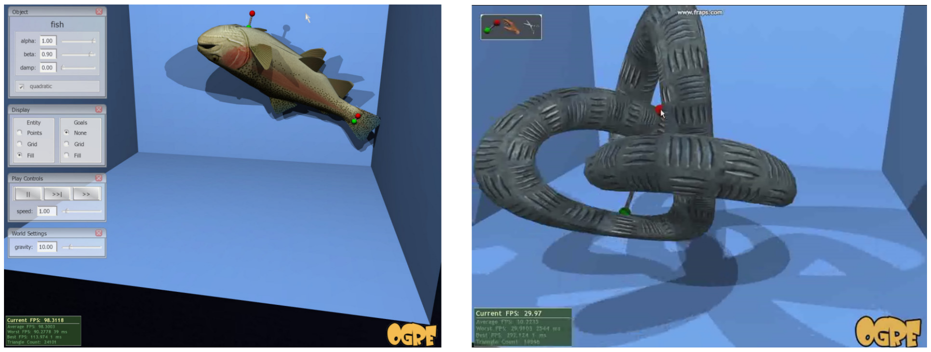
In order to make a virtual world more realistic it is necessary to enable the user to interact with objects in a believable manner. Simulating both the look and feel of materials increases realism and the immersive experience. Furthermore advanced interactions are required for many applications such 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 mouse wheel. Spring forces can be locked in place, allowing the user to change modes or create new spring forces. In this manner objects can be moved around, bend, and ”fixed” in deformed positions (see figure 3). This mode is useful for precisely manipulating an object's position, deformation and orientation.

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 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.



```
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
```

```
Current FPS: 29.97
Average FPS: 30.2235
Worst FPS: 29.9103 2544 ms
Best FPS: 292.124 1 ms
Triangle Count: 19946
```

Fig. 3. A deformed model of a trout fixed using two locked pick points (left) and a torus model violently moved around using the picking mode (right).

To change the orientation of the scissors, the user can move the scissors towards and away from the view point using the mouse wheel. The scissors can be rotated about the y-axis by holding down shift and pressing the left mouse button.

Our implementation splits an object in two along a plane by using two clipping operations and removing parts outside the clipping plane. If the clipped triangle is a quadrilateral it is divided into two triangles. Note that the cutting plane is derived from the orientation of the cutting tool used in the application. Hence we do not have to deal with partial cuts and the internal surface revealed by the cuts is always planar.

The next step is to seal the exposed cross-sections of the divided object. New surfaces are created by applying a Delaunay triangulation to the newly created vertices touching the cutting plane (see figure 4). 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 the triangulation is performed, the object is tetrahedralised and divided into clusters again.

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 [15].

The collision detection and response techniques used by Müller et al. [9] involve spatial hashing [13] and penetration depth estimation [5] on tetrahedral meshes:

1. Using spatial hashing classify all points intersecting a tetrahedron as colliding.
2. Colliding points connected by an edge to a non-colliding point are *border points*.
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.

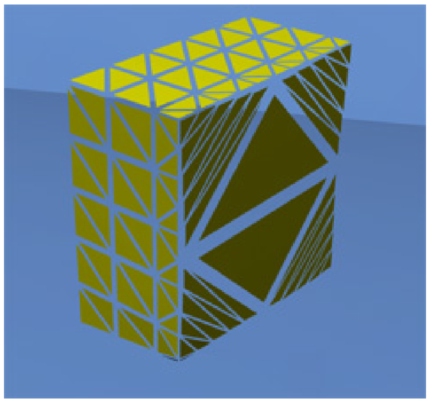


Fig. 4. After a cut, the exposed internal hole is sealed up with a delaunay triangulation.

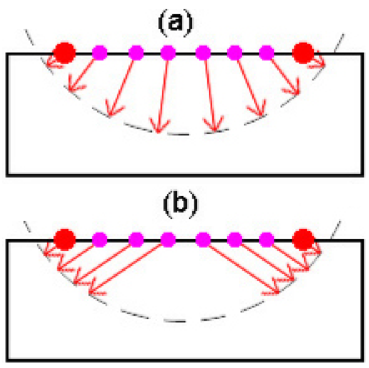


Fig. 5. Response forces: ideal (a) and using a penetration depth estimate.

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 tetrahedrons. 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. 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.

6.1 Simulation Capabilities

In order to determine the suitability of meshless deformation for interactive simulations we tested it on a variety of objects using different deformations:

Simulation of jelly-like cubes and spheres: The four basic modes of deformation possible are stretch, shear, bend and twist. We tested the real-time rigid, linear, and quadratic deformation of a deformable cube and sphere subjected to user interaction. Visual plausibility was good; the objects behaved as one would expect.

Simulation of a soft L-shaped bar: This test investigates the deformation behaviour for large displacements applied to highly concave objects. We would expect to be able to bend both sections of the bar together or apart, but using only one cluster this is not possible as illustrated in figure 6 (a)-(b). In particular even with quadratic behaviour the bar twists and bounces strangely (c). A significant improvement is achieved by using one cluster for each branch of the L-bar. Results are plausible (see figure 7), however where clusters meet at the corner of the bar, deformation is uneven when the bar is pulled straight (c).

Simulation of complex objects: We tested a variety of complex objects and found that objects with limited bending modes, such as a trout, are simulated well (left image of figure 3). It seems that the quadratic deformation modes of the trout model (which is a surface model) correspond well with the type and magnitude of deformations of a real trout which are restricted by its rigid skeletal structure. Surprisingly also some very complicated objects such as an intertwined rubber torus look realistic (right image of figure 3). The main reason for this seems to be that users are not familiar with its behaviour. An informal user survey with students revealed that there was no consensus how the object should behave when deformed. In contrast we are intimately familiar with the deformations of a human face 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.

Simulation of soft tissue: For the final test we tested the suitability of meshless deformation for virtual surgery simulations [6]. Good results were obtained when applying large deformations to blobby objects such as kidney shaped models and convoluted tube like structures. A trained user could easily notice that the deformations were physically not realistic, however, when using clusters they were physically plausible and sufficient for simulating process related surgical tasks.

Modelling local deformations, e.g., for a section of skin, is difficult. Figure 8 (a) shows that for a skin patch consisting of one cluster ($\alpha = 1.0$, $\beta = 0.6$), picking simply

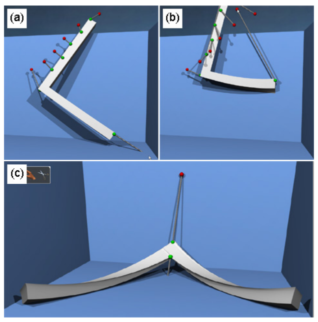


Fig. 6. L-bar without clusters being pulled apart.

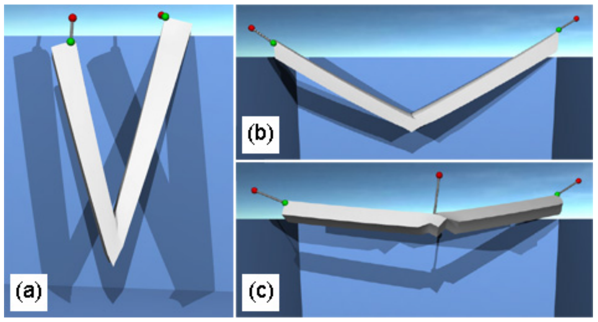


Fig. 7. The same operation for an L-bar modelled with two clusters.

attempts to move and deform the entire skin patch. When the skin patch is divided into 2×2 clusters deformation is more plausible but still limited, with the dividing lines between clusters quite visible in figure 8 (b)-(d). Using 5×5 clusters significantly increases visual plausibility to a satisfactory level (figure 9). Using such a large number of clusters is, however, quite inefficient and less accurate than using a mass-spring system with a similar resolution.

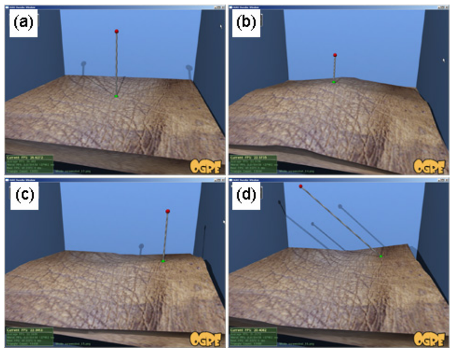


Fig. 8. (a) pick on a single cluster skin patch, (b)-(d) picks on a 2×2 cluster skin patch.

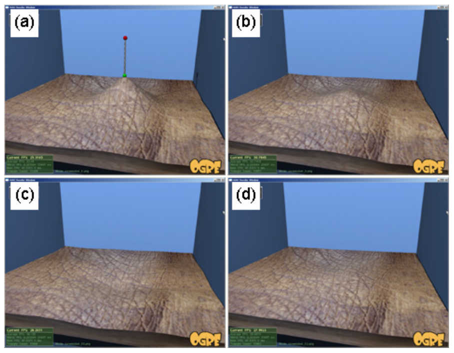


Fig. 9. Behaviour of a 5×5 cluster skin patch in response to a user pick.

In summary 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, can not be simulated efficiently [10]. However, the method seems to work well for complicated objects which have simple deformation modes (e.g., a trout) or with where the user is unfamiliar with its behaviour (e.g., an intertwined rubber torus).

Local deformations can not be modelled efficiently and are best approximated by clusters. An alternative solution are hybrid models combining meshless deformation and, for example, a mesh spring model. Such multi-resolution representation where non-linear responses are only considered in the immediate vicinity of the applied force in order to obtain real-time non-linear deformations exist already, however, they require model representation which are not suitable for game engines and other common graphics engines for virtual environments [3].

6.2 Suitability for Real-Time Collaborative Interactive Environments

We identified a range of criteria which a soft object simulation technique for real-time interactive collaborative environments, such as computer games or Virtual Worlds, must fulfill.

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. In particular the cutting tool proofed surprisingly popular in our informal user testing (figure 10).

Stability. Due to the improvements implemented, extreme forces and deformations no longer produce stable inversions or erratic behaviour due to temporary inversions. Furthermore, 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.

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 11). Higher speeds, e.g., for simple virtual surgery applications, 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 applications which require plausible localised deformation of an arbitrary region, e.g., motor skill training of surgeons, our environment is less suitable. Also, even when the simulation is visually plausible, it is usually not physically accurate.

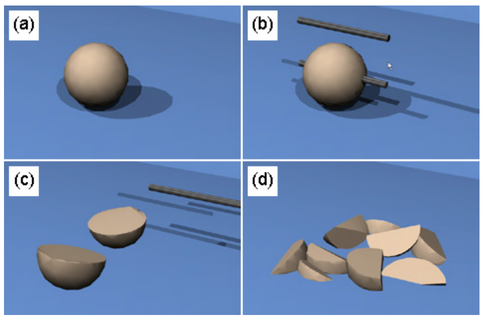


Fig. 10. Cutting an object: (a) during cut, (b) immediately after cut, (c) two resulting halves have rolled apart, (d) after further cuts.

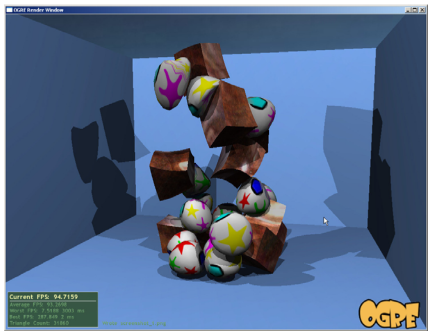


Fig. 11. Large scale simulation of deformable objects.

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 techniques enabling users to interact with deformable 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 environment 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

1. Arun, K., Huang, T., and Blostein, S. Least-squares fitting of two 3-D point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence 9*, 5 (1987), 698–700.
2. Berkley, J., Turkiyyah, G., Berg, D., Ganter, M., and Weghorst, S. Real-time finite element modeling for surgery simulation: An application to virtual suturing. *IEEE Transactions on Visualization and Computer Graphics 10*, 3 (2004), 314–325.
3. De, S., Lim, Y.-J., Manivannan, M., and Srinivasan, M. A. Physically realistic virtual surgery using the point-associated finite field (paff) approach. *Presence: Teleoperators and Virtual Environments 15*, 3 (2006), 294–308.
4. Delp, S. L., Loan, P., Basdogan, C., and Rosen, J. M. Surgical simulation: an emerging technology for emergency medical training. *Presence: Teleoperators and Virtual Environments 6*, 2 (1997), 147–159.
5. Heidelberger, B., Teschner, M., Keiser, R., Muller, M., and Gross, M. Consistent penetration depth estimation for deformable collision response. *Proceedings of Vision, Modeling, Visualization VMV04, Stanford, USA* (2004), 339–346.
6. Henriques, A., Wünsche, B. C., and Marks, S. An investigation of meshless deformation for fast soft tissue simulation in virtual surgery applications. *International Journal of Computer Assissted Radiology and Surgery 2 supplement 1 (Proceedings of the 21st International Congress and Exhibition on Computer Assisted Radiology and Surgery (CARS 2007)* (June 2007), 169–171.
7. Horn, B. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A 4*, 4 (1987), 629–642.
8. Kühnapfel, U., Cakmak, H., and Maass, H. Endoscopic surgery training using virtual reality and deformable tissue simulation. *Computers & Graphics 24* (2000), 671–682.
9. Müller, M., Heidelberger, B., Teschner, M., and Gross, M. Meshless deformations based on shape matching. *ACM Trans. Graph. 24*, 3 (2005), 471–478.
10. Rubin, J., and Wünsche, B. C. 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_Jonathan Rubin.pdf`.
11. Sederberg, T. W., and Parry, S. R. Free-form deformation of solid geometric models. *ACM Transactions on Graphics 20*, 4 (1986), 151–160.
12. Smith, R. Open Dynamics Engine home page, 2006. `http://www.ode.org`.
13. Teschner, M., Heidelberger, B., Mueller, M., Pomeranets, D., and Gross, M. Optimized spatial hashing for collision detection of deformable objects, 2003.
14. Teschner, M., Heidelberger, B., Muller, M., and Gross, M. A versatile and robust model for geometrically complex deformable solids. *Computer Graphics International, 2004. Proceedings* (2004), 312–319.
15. Teschner, M., Kimmerle, S., Zachmann, G., Heidelberger, B., Raghupathi, L., Fuhrmann, A., Cani, M.-P., Faure, F., Magnetat-Thalmann, N., and Strasser, W. Collision detection for deformable objects. In *Eurographics State-of-the-Art Report (EG-STAR)* (2004), Eurographics Association, Eurographics Association, pp. 119–139.
16. Umeyama, S. Least-squares estimation of transformation parameters between two point patterns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 13*, 4 (1991), 376–380.