Adaptive Refinement for Mass/Spring Simulations

Dave Hutchinson, Martin Preston, Terry Hewitt
Computer Graphics Unit, Manchester Computing,
University of Manchester, Manchester, M13 9PL, United Kingdom.
Email : {*Dave.Hutchinson* | *preston* | *hewitt*}*@mcc.ac.uk*
WWW : *http://info.mcc.ac.uk/CGU/*

Abstract. Mass/Spring networks are commonly used to produce simulations of deformable bodies for computer animation. However, such an approach can produce inaccurate results if too coarse a discretisation is employed, and so many animators use excessively large (and slow) networks. In order to remove the 'guesswork' from such an approach this paper presents a mechanism for adaptively refining portions of such systems to a required accuracy, thereby producing more pleasing results at a reduced computational cost. Following a discussion of the use of such an approach in simulating a deformable sheet, we present several characteristic examples to demonstrate its suitability.

Keywords: *Computer Animation, Simulation, Deformable Bodies, Spring-Mass Approximations, Adaptive Refinement.*

Animations: *http://info.mcc.ac.uk/CGU/research/animation/defrm.html*

1 Introduction

Keyframing techniques have been used for many years in the animation of deformable bodies [13, 3]. However, the realism of such an approach is heavily reliant on the skills of the animator, and so several researchers have adapted dynamic simulation to this area in the hope of reducing the burden on animators. Two fundamental solutions have been proposed, both of which discretise the body being simulated into finite elements, but which differ in the simulation model being employed.

The first group requires the derivation of differential elasticity equations, which are then integrated through time. Terzopoulos, Platt, Barr & Fleischer's [15] technique has since been extended to allow areas of rigidity in the bodies to be comfortably modelled [16], and for inelastic deformation to be simulated [14]. The second group disregards physical theory and employs a simulation method based on discretising the body into a number of masses, or particles [8],

[0]Published in 7^{th} Eurographics Workshop on Animation & Simulation, ©Springer, This copy for personal use only.

whose connectivity is maintained through constraint forces (usually generated from a damped spring abstraction). Systems of this nature have been used to simulate flesh around rigid bodies [6], the behaviour of cloth-like sheets [1, 12, 7], decomposable solids [4], fluids [9], and even worms [10].

The second group is most commonly used for two principal reasons: it is easier for the animator to integrate this approach with simulations of rigid bodies and it is far easier to implement. However, the strategy of approximation used in mass/spring networks only leads to plausible results if the *correct* granularity is chosen, and for practical cases this granularity is not obvious. If too coarse an approximation is employed then an incorrect animation will be generated, but unfortunately the animator will often have no way of knowing how 'right' this solution is. Conversely, if too fine a network is employed then a more correct answer *may* emerge, at the expense of increased computation. Consequently animators are often forced to either tinker with the model, or endure inaccurate results (thereby negating the advantages of dynamic simulation). Thingvold & Cohen [17] attempted to address this by employing a B-spline representation of the sheet being simulated, which could be refined in response to detected inaccuracy. However, the tensor nature of the surface meant that it was impossible to concentrate effort in particular regions (entire bands of the surface must be refined at once), so the technique has a tendency to produce excessively complicated, and slower, simulations.

This paper addresses that problem by presenting a mechanism for adaptively refining the network of spring/masses to concentrate effort *only* where it is needed. Using this system the animator defines an initially coarse approximation, and additionally a measure which indicates the accuracy required for this case. The system then proceeds to simulate the body in the normal way, but where potential inaccuracies are detected the body is refined only in the affected region, and then simulation may proceed. Whilst such an approach could be employed in the animation of either sheets or volumes, this paper concentrates on an implementation of adaptively refined sheets, though the extension to the simulation of volumes is discussed briefly in Section 7.

This approach has the advantages of:

- No longer requiring the animator to guess the discretisation required.
- Concentrating effort only where it leads to a more accurate solution, thereby producing the result more quickly than previous refinement techniques [17].
- Retaining the advantages of spring/mass networks (ease of both implementation and integration with the simulation of other bodies).

The rest of the paper is structured as follows. Section 2 describes existing deformable body techniques. Section 3 outlines our approach, with particular reference to the manner in which the animator can select the required accuracy (the refinement conditions). Following this Section 4 presents the implementation of this technique in detail. Such an adaptively refined body may be used in the simulation of a larger virtual world, and so in Section 5 we identify the considerations required to include collision detection and response. Section 6 presents results and timings for a number of characteristic cases, before Section 7 identifies the advantages, disadvantages and future work suggested by this approach.

2 Related Work

As described earlier two families of solution have been proposed for deformable body simulation, both of which rely on discretising the body into a finite number of components. The first group, proposed by Terzopoulos *et. al.* [15], deforms these finite elements using elasticity theory which works by recognising that the force applied to an element is (in varying proportions) absorbed or passed onto its neighbouring elements in a manner governed by its elasticity. Using a series of equations (based on Lagrange's ordinary differential equation) we can integrate through time to discover the forces caused, and hence the motion. Multigrid methods can be used to solve these sets of equations. This approach was extended to allow rigidity to be simulated (by incorporating a rigid "reference component") [16], and also to model inelastic deformations (so the body won't necessarily return to its original shape if stretched) [14]. These elaborations improve the original model, but Gascuel and Puech [5] noted that a remaining weakness is the difficulty of determining the spatial discretisation for non-trivial bodies. Palazzi and Forsey [11] described a multilevel approach to surface deformation which enables an animator to control the flexibility of a deformable object. External forces may have local or global effects on the surface depending on which levels of the multi-resolution representation they are applied to.

The second family of solutions employ a form of dynamic simulation which disregards the physics of object deformation, preferring instead to concentrate on a method which produces plausible, rather than thoroughly accurate results. By splitting the body into a network of masses connected by constraint forces, which behave like damped springs, a very simple integration process can be used to simulate the forces moving through the body, indirectly leading to visible deformations. Such a system suffers from numerous problems: sensitivity to integration step size, instability of solids simulated in this way (as springs operate only in one dimension it is possible for elements to invert themselves) and high computational cost for large networks. However, this approach is widely used largely because it is trivial to implement.

Thingvold & Cohen [17] improved on this model by representing this network as a B-spline surface, where particles corresponded to control points. Where the simulation detected problems (such as excessive stress) a region was refined [2]. However, the tensor nature of the surface meant that bands of the surface must be refined together. In many cases, such as when modelling wrinkling at the ends of sheets, the entire surface must be refined considerably. Whilst this implementation approach often leads to slow simulations, the application of dynamic refinement was shown to lead to more accurate results without the guesswork required in conventional particle systems.

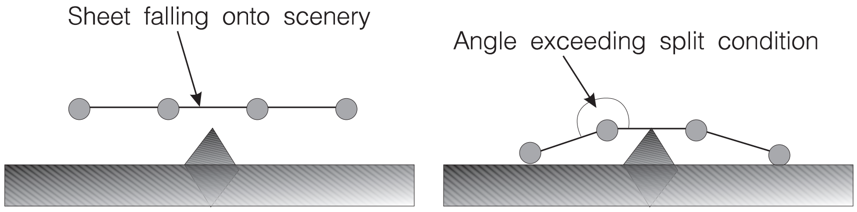


Figure 1: When the springs entering a mass point become non-planar, a discontinuity is detected

3 Outline of Technique

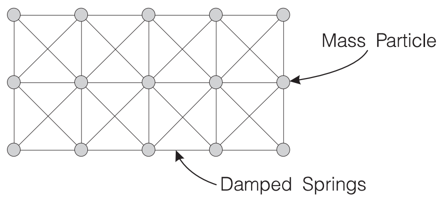
In this section we present an overview of the mechanisms used to improve the accuracy of spring/mass networks for simulating sheets before section 4 presents a practical implementation. In 3.1 we discuss the principal reasons for the inaccuracy of coarse networks, and identify the metric that an animator can use to guide refinement in our technique. In 3.2 we consider the response to any inaccuracies, i.e., how we refine portions of the network by adding new masses & springs without changing the properties of the sheet.

3.1 Detection of Inaccuracy

Networks of masses, connected by damped springs, attempt to approximate the behaviour of deformable bodies using a primitive model for the transmission of energy. The popularity of this model is primarily due to the pleasing nature of the results for very simple simulations. However, the difficulty of using measured physical properties means that significant inaccuracies appear in the result. Unfortunately, because these simulations are used in cases where the animator knows few of the true physical properties of the sheet, we cannot easily detect where these inaccuracies are. The 'success' of one of these simulations is instead related to the visual properties of the surface when rendered, and so we must concentrate our efforts on making the result *look* acceptable.

When coarse approximations of sheets are simulated the most visible errors occur when the surface creases. As the network may only bend on predefined lines (where the springs lie) these straight lines are frequently visible. Whilst spline surfaces can be passed through the point masses to mask the worst effects of this [17], they cannot cater for all situations. Therefore we will use, as our measure of inaccuracy, the presence of unsightly (and inaccurate) discontinuities.

Creases occur at mass points when opposite springs connecting neighbouring point masses become non-planar, as shown in Figure 1. Here the neighbouring masses have been moved out of the plane, but as the springs must operate in straight lines, a discontinuity appears. The angle at which this crease becomes excessive is related both to the rendering technique, and the situation, so it would be unwise to develop an automatic tolerance algorithm. Instead our technique relies on the animator specifying an angle which governs the accuracy he or




```
for(u=0; u < nU; u++)
  for(v=0; v < nV; v++)
    simulate_point(u,v,dt)

void simulate_point(u,v,dt) {
  find forces caused \
    by connecting springs and
    external influences;
 calculate new acceleration, \
    velocity and position;
}
```

Figure 2: a. Sheets are split into a number of discrete masses connected by damped springs in the vertical, horizontal and diagonal directions, and the simulation algorithm for this sheet is shown on the right.

she requires in the finished animation. Our 'inaccuracy' test can therefore be termed:

Split condition : If the angle between two springs joining a mass from opposite directions exceeds a specified tolerance S (in either direction), then we would produce an unacceptable simulation, and so must concentrate effort on refining this area of the model.

Our response to a split condition is to back the simulation up to a point at which it was acceptable (normally the previous time step), introduce more masses around the crease point, and then re-run that portion of the simulation to see whether we are now simulating to an acceptable degree of accuracy. Before we go on to discuss the mechanics of this (in section 3.2) we must also identify how this 'split tolerance' changes with the coarseness of the simulation. If the sheet is being approximated coarsely, then the animator will probably wish to remove *any* creases. However, if the surface has been refined considerably, then any minor discontinuities will be less obvious. To cater for this we also introduce a delta-angle δS, which is the angle added to the tolerance with the addition of each area of refinement. So, an initial refinement is caused when neighbouring springs exceed S, and a further refinement will be made if the angle exceeds $S + \delta S$. Finally we also employ a maximum level control, which an animator can employ to restrict the degree to which a sheet will refine in extreme cases.

3.2 Refinement

Our response to the detection of inaccuracy is the addition of masses and springs around the area where a discontinuity has occurred. However, in order to ensure that such a refined area continues to respond in the correct manner, we must maintain three constraints:

1. Every particle in our network must behave in the same way. If this is not so then the response of a particular mass to other forces may become unpredictable.

2. A given portion of the sheet must *behave* as though it has constant mass, independent of the number of particles used to simulate this area. If this is not the case then we will produce different, rather than more accurate, results using a refined network.

3. Forces must move across the refined portion at the same speed as the unrefined areas, otherwise shearing will occur.

Before discussing the approach taken to meet these conditions it is helpful to identify how they are catered for in the conventional uniform sheet. Figure 2 shows a sheet along with the simulation algorithm which is normally used. Here, at each time step, we iterate across the sheet and for each point calculate the forces affecting it due to connecting springs, the change in acceleration, and finally the new positions after the time step. If a force is applied at one end of the sheet during a simulation, it takes n time steps to reach a mass point n springs away. The effect of collisions with the environment is determined by calculating the mass of the portion of the sheet which is in collision. As the conventional network has uniformly spaced, and identical, masses this can be achieved by summing the effect of the relevant particles.

In order to refine the mass/spring network we need to add extra masses and springs. However, if we add masses to the system we run the risk of altering the properties of the sheet, and so we must pay special attention to ensuring the first two properties have been maintained. Thingvold & Cohen [17] did this by adding particles of lesser mass (thereby meeting the second constraint), but had difficulty satisfying the first, i.e., as we deal with single particles on each time step, the reduced mass particles would react differently to those present in the initial sheet.

We adopt the following approach: when we add particles we always use the same consistent mass (thereby meeting the first constraint), and we modify the simulation process to meet the remaining two constraints, i.e., to preserve the behaviour of the newly refined sheet.

To support this we employ a non-uniform representation of the sheet, and a form of simulation which deals with refined regions using different time step sizes. This new representation can be *thought of* as a multi-level or 'hierarchical sheet', and in this section we will discuss the simulation process as though we implement the data structure in a hierarchical fashion, before introducing the more optimal (yet similar) structure used in our implementation in Section 4.

At the beginning of the simulation we begin with a conventional m by n grid of points, at level 0. When we detect inaccuracies we generate mass points and springs in the level above, where *the mass of each new point is exactly the same as those in the level below.* At this new level, as shown in Figure 3, masses are spaced twice as finely. Upon refinement we generate 8 new points, (shown as filled circles in Figure 3), which surround the discontinuity point. For

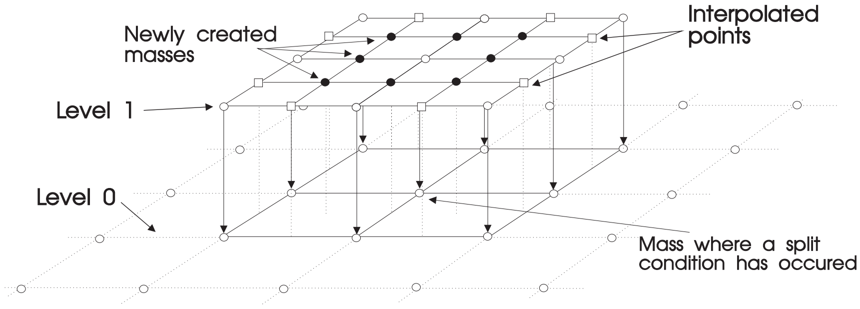


Figure 3: The sheet is modelled as a hierarchy of mass/spring networks. Here we show a single refinement around a point (note that the bend of the sheet is not shown)

these masses to be able to contribute to the behaviour of the sheet we must also connect them by springs.

Some of the points we connect these springs to exist in this new level, but some of them do not (as they will be related to mass points which haven't been refined). These different points can be grouped into 2 categories: those that correspond to mass points in the level below (shown as unfilled circles in Figure 3), and those which have no counterpart in the previous level (shown as unfilled squares). During the simulation of this level (which we discuss in greater detail below) we deal with these 'non-existent on level n' points specially. Where the point corresponds to a mass on level $n-1$, we use that point (i.e., we traverse the data structure). Where the point doesn't correspond we generate the point dynamically (by examining the points adjacent to it on level $n-1$, and finding the mid-point).

For this data structure to work we also need to make some changes to the simulation represented by the function `simulate_point()`. The principal differences are the addition of a level parameter to indicate which level the points are on, and the response to a refined area. If the function detects a refined area, i.e., the particle it is currently concerned with is present on the next level, then it initiates another loop across the points in the next higher level, which matches the loop executes at level 0. However, at this level the time step used is proportionally smaller.

This system achieves our conditions in the following ways:

1. The particles have uniform mass, so respond to forces in the normal way.

2. The stiffness of a spring doubles for each level of refinement to prevent regions of increased mass behaving differently. This increases the stability of the model, and allows the magnitude of forces passing through different refinement levels to be maintained. It also ensures that different refinement levels behave in a similar manner.

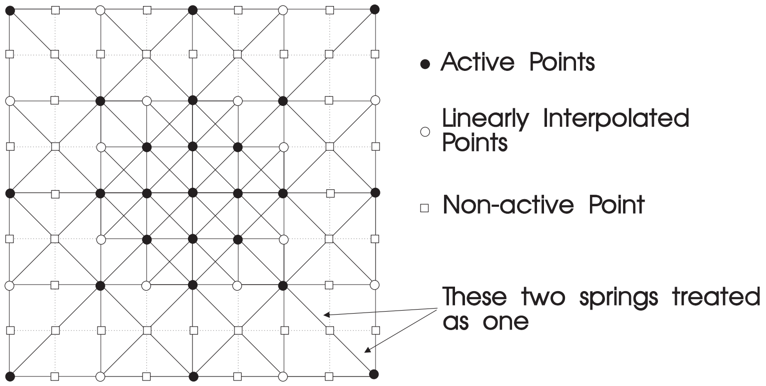


Figure 4: We flatten the sheet, storing the mass points on a uniform grid. The portion shown here corresponds to that shown in Figure 3.

3. The response of the body due to its mass is governed by the time in which it is allowed to respond, and the magnitude of that mass. By employing smaller step sizes for those points which correspond to smaller regions of the sheet, we achieve the same effect as a proportionally smaller mass would [17], as we employ Euler integration. We must, however, take care to account for transfer of momentum between objects, so that they respond in a predictable way, and this is discussed further in Section 5. The success of these modifications is demonstrated further in section 6.

4. The smaller step size for higher levels also allows us to ensure ripples pass across the surface at uniform speeds. The new version of `simulate_point()` achieves this by executing refined portions in the same time step as the unrefined portions.

4 Implementation of Adaptive Refinement

The previous section described in outline form how we refine the sheet network to cope with inaccuracies, using a hierarchical data structure which models the network at different levels of refinement, and a recursive process which simulates refined portions at the same speed as unrefined ones. In this section we now discuss the practical issues involved with implementing this approach, with Section 4.1 presenting the form of implementation used, and Section 4.2 discussing the manner in which we handle refinement in this new structure.

4.1 The Flattened Data Structure

Although traversing the hierarchical data structure in a recursive manner works well, it is by no means the most efficient method. Such a system requires repeatedly recursing up and down the hierarchy and redundancy is introduced when neighbouring regions are visited several times. Consequently we have developed a more optimal implementation method, where the simulation is performed within one loop and the data structure is effectively flattened into one level, as shown in Figure 4. We generate this 'flattened' representation by creating a new grid whose resolution matches the finest refinement possible of the sheet, for example if we are simulating a sheet which at level 0 is $5x5$, but which contains only one patch refined to 2 more levels, then the flattened sheet would consist of $17x17$ points. Whilst we could generate this flattened sheet statically (using the animators selected maximum refinement), we have instead chosen to begin with a sheet of a predefined refinement (2 levels), which is regenerated if further refinement is required (as this reduces the amount of memory consumed for the average case).

The simulation process works as follows. During each timestep T (which corresponds to t at level 0 in Section 3) of the simulation we iterate over the flattened sheet several times (in order to achieve the same effect as the recursive simulation), each time simulating the sheet for a 'sub-time-step' t. This smaller step size is calculated to correspond to the finest `dt` used in the hierarchical structure. So if the system has been refined only once then the small time step will be $T/2$, and for a system that has been refined to n levels then each step will be $T/(2^n)$.

For each t we must determine the behaviour of each mass point. As this is an optimisation of the hierarchical structure, we must be careful to ensure that we produce the same effects. We achieve this by storing, in the flattened sheet, a record of which hierarchical level each point is a member of. When we are on a sub-time-step which would correspond to moving a particular point in the hierarchical structure, then we perform conventional simulation (and also keep a record of the velocity calculated for this point), but where the point would not feel a change in acceleration the point mass is moved according to its previously calculated velocity (i.e., constant velocity, disregarding the effects of any connecting springs).

For example, if a system has been refined in some region once then $t = T/2$. Those points that are refined will feel forces from their neighbours on every time step t, whereas the unrefined regions only feel a force once over the whole time T. In general, for n-level refinements, if a point is at a time step where it doesn't feel a force then it is assumed to have constant velocity, the value of which will have been calculated during the last time step in which it felt a force.

This method affects only those points that are active at some level in our logical hierarchical structure. However, the flattened sheet is also likely to contain a large number of inactive points, who are present only in case of later refinement. These masses play no part in the simulation, but could incur a performance penalty as each of them must be examined at each t step. To avoid this penalty when the maximum refinement level is high, we optimise the loop by maintaining a binary tree of active points alongside the main structure, so that we do not need to step through the entire structure each time step. There is also a small overhead required for calculating which particles affect each other depending on the level they are at. This is because a particle at level three may have spring connections to particles at level 2,1 or 0. In our simulations this has proved to have a negligible effect on performance.

4.2 Refining the Flattened Structure

When a region is refined, we must activate some of the masses in the flattened structure, and velocities and positions must be assigned to them. These could be obtained by averaging the positions and velocities of surrounding particles. Unfortunately this does not always give good results because if a region is being refined, then an *inaccuracy* has occurred, and any averages are also likely to be misleading. Consequently, when we detect an inaccuracy around a single particle, we perform the following process

1. Back the simulation up a single t for the inaccurate particle, and all those immediately neighbouring it which would effect it on this t step. To find this set of points we examine those springs which connect to this mass, and follow them to those points which are active on this step.

2. Introduce the new masses by activating those points in the uniform sheet (as shown in Figure 4) which surround the inaccurate particle.

3. Calculate the position & velocity of these points by averaging the properties of the neighbouring mass point. As we have backed the simulation up for these particles these new averages will be accurate.

4. Re-run that t step for both the new points, and those which we have backed-up, in order to determine the new position of the sheet.

5 Collision Response for Refined Bodies

The detection of collisions between a sheet which is composed of a large number of polygons, and the environment, is aided by the use of adaptive refinement, as fewer polygons will be required. However, as we are using a complicated representation the calculation of the response caused by collisions is more complicated. Because our refined sheet contains a larger number of masses, any moving object colliding with it will experience a different effect depending on the amount of refinement and therefore number of masses it interacts with. This can be overcome in the following way.

We want an unrefined section of the sheet to react in the same way to collision with a moving object as a refined section would and also that a moving object will receive the same momentum change from a refined section as an unrefined one. If we know what percentage of the sheet is in collision with a moving object, then given that we also know the mass of the whole sheet (before any refinement), it is easy to calculate the mass m_a of the touching section of the sheet (that is the mass of the unrefined sheet which is in contact with the colliding object). The mass m_m of the points in the touching region may exceed m_a (as in our model refined masses are the same as unrefined), and so we must scale the momentum transfer between the two objects by the ratio of the mass in the

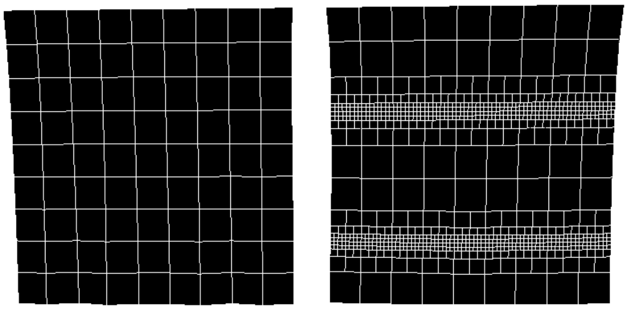


Figure 5: a. Unrefined sheet hanging. b. Same sheet with two bands of refinement.

touching region and the actual mass of the sheet. The effect of this is to increase any momentum transfer to particles in the sheet by m_m/m_a, and to decrease the total momentum transfer from masses in the sheet to the moving object by m_a/m_m.

This means that any object colliding with a refined portion of the sheet will interact in the same way as a collision with an unrefined section.

6 Results & Discussion

To demonstrate the advantages of adaptively refined mass/spring networks this section presents three representative case studies. The algorithm described in Section 4, was implemented on a HP 9000/735 workstation, using a distributed graphics framework (HEDGE). The force exerted by springs due to deformation was proportional to the extension length, and damping was introduced as being proportional to the *rate* of extension. The sample implementation employs the technique discussed in section 4. At each time step an Euler integration process is used to determine the response of each particle to the relevant forces. Collision detection was performed using a simple and accurate, but inefficient algorithm which enabled us to model the response of the sheet to the environment. Self-collision was not detected, but the authors feel that Volino and Thalmann's technique [18] would be suitable for this technique. A primitive friction model was employed, so none of these examples include the effect of the sheet sliding across the scenery.

The first example, shown in Figure 5, is included to demonstrate how our non-uniform sheet representation preserves the properties of the more conventional uniform network. Two sheets were initially placed flat in the scene, one of which consisted of 10x10 masses, and the second had two bands of high refinement. One side of each sheet was clamped to a particular height, and then both sheets were allowed to drop in response to gravity. During the simulation both sheets retained the same dimensions, thereby demonstrating the mass preserving property of our algorithm. Note that, in order to achieve these results, considerable attention was paid to numerical accuracy. The refined sheet requires more calculation, and this runs the risk of introducing inaccuracies into the result, and so we employ increased precision when storing each attribute of the sheet.

The second and third examples are included to demonstrate both the ability of the adaptive refinement technique to identify areas of interest, and to considerably reduce the computation required. Each sheet initially consisted of 10x10 mass points, and had the maximum refinement degree set to prevent the sheet refining beyond 73x73. For each we set S to 25^o and δS to 15^o. The second animation consisted of dropping a sheet onto a polygonal model of a coffee cup. The third animation involved dropping the sheet over 4 poles. After a number of time steps the simulation was stopped, and the results examined. The state of the animation after 100 time steps for the cup animation is shown in Figure 6, and after 170 steps of the pole animation the sheets position in shown in Figure 7. The full animations are available on the accompanying web page (`http://info.mcc.ac.uk/CGU/research/animation/defrm.html`). Each picture shows both the refinement map, which identifies the nature of the network, and a shaded representation of the surface, which shows what a user of this technique would actually see.

The simulation of the sheet falling over the cup clearly demonstrates the ability of our adaptive refinement technique to detect where inaccuracy is caused by edges in the scenery, and the response caused (increasing the number of mass points in these regions). Note that, as the model of the cups' handle is extremely primitive, and has a square cross section, the network hasn't had to refine a great deal to accurately follow it.

The sheet hanging over the poles shows the detection of the scenery, but also indicates where creases and folds have been identified by the algorithm. This is clearly visible at the corners of the sheet, where the material falls back, causing a fold. Our primitive friction model is also causing creasing between the poles, as the sheet quickly reaches maximum extension in the mass/springs directly between the poles, but the neighbouring particles have greater freedom. A sequence of pictures taken from the animation is also shown in the colour plate in the Appendix.

Table 1 shows the measured execution times for these two examples. Without adaptive refinement an animator would have to employ a very fine network, so to compare the improvements we also performed simulations for the maximum 73x73 grid. The bulk of computation for the cup example is consumed by the collision detection, however it is clear that the refined sheet reduces this considerably (as there are fewer polygons in the sheet). The proportion of the computation caused by dynamic simulation is also considerably lower, and the extra load caused by detecting, and backing up the simulation to accommodate refinement, makes little difference. The full 73x73 grid sheet required ~18 hours of computation to reach this state, while the adaptive sheet needed ~1 hour.

The pole example does not suffer from such a severe collision detection cost, but the improvement due to adaptive refinement is still considerable. With the full case requiring $1\frac{1}{4}$ hours of computation, and the refined sheet needing only ~12 minutes to execute 170 time steps. Both of these examples clearly show the advantage of using an adaptive refinement technique which concentrates

Example	Cup 73x73	Cup Adaptive	Poles 73x73	Poles Adaptive
Num. Steps	100	100	170	170
Collision Detection	64319	3929.2	3580	687.0
Dynamic Simulation	474	16.3	669.8	76.3
Backing up	0.0	0.78	0.0	0.29
Refinement	0.0	0.71	0.0	1.2
Total	64793	3947.0	4598	764.79

Table 1: Sheet Statistics (all times measured in seconds)

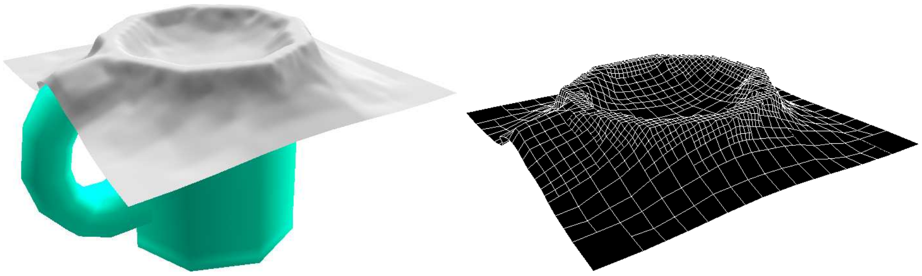


Figure 6: a. Rendered sheet over a mug   b. Refinement map.

effort only where required; if the same examples were simulated using a B-spline network (as per. [17]) the entire sheet would require refinement, and so would require computation closer to the maximum 73x73 sheet.

7 Conclusions

We have presented a method for adaptively refining spring/mass networks, which has the advantages of both optimising the simulation of such systems and adapting the amount of refinement required for a particular environment without prior knowledge of it. We have shown that this system can produce the same visually

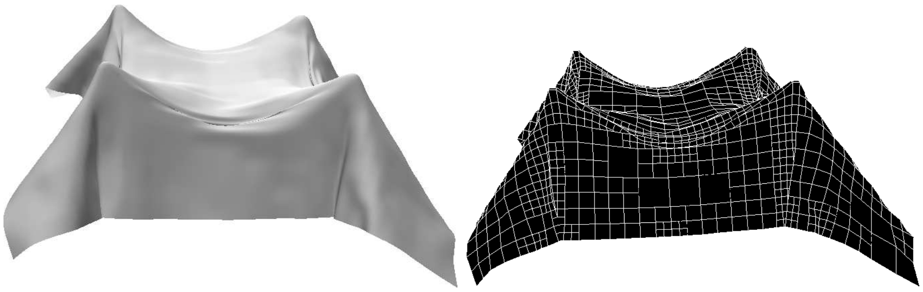


Figure 7: a. Rendered sheet over 4 poles (not shown) b. Refinement map.

pleasing result for less computational cost than a fine uniform discretisation. Although this method shares the weakness of ordinary particle systems (principally physical inaccuracy) a mechanism of adaptive refinement goes some way to ameliorating the worse effects of these inaccuracies.

We intend to concentrate our further work in two principal areas. Firstly we would like to develop more optimal schemes for implementing adaptive sheets. An unsplitting system (which detects planarity and responds by removing masses) is currently being developed. Special attention must be made to removing masses only where planarity is likely to be sufficient for more than a few steps (otherwise too much time will be spent refining again). We also plan to investigate the use of a non-rectangular grid which will then allow us to perform a discontinuity meshing (thereby reducing the complexity required to drape sheets over non-axis aligned edges).

We would also like to extend these techniques to the simulation of volumes. Our initial work in this area is concerned with developing a robust hypercube data structure for the volume, which allows us to accurately model indentations in the solid when it comes into contact with the environment.

Acknowledgements

The authors would like to thank all the staff and students at the Computer Graphics Unit for their help and encouragement. Dave Hutchinson would like to thank the Engineering and Physical Sciences Research Council for their financial support during this work.

References

1. D. E. Breen, D. H. House, and M. J. Wozny. Predicting the Drape of Woven Cloth Using Interacting Particles. In *Proceedings of SIGGRAPH '94*, pages 365–372, 1994. In Computer Graphics proceedings, Annual Conference Series.
2. E. Cohen, T. Lyche, and L. L. Schumaker. Algorithms for Degree Raising of Splines. *ACM Transactions on Graphics*, pages 171–181, July 1985.
3. S. Coquillart and P. Jancène. Animated Free-Form Deformation: An Interactive Animation Technique. *Computer Graphics*, 25(4):23–26, July 1991.
4. M. Desbrun and M-P. Gascuel. Highly Deformable Material for Animation and Collision Processing. In *5^{th} Eurographics workshop on Animation & Simulation*, 1994.
5. M-P. Gascuel and C. Puech. Dynamic Animation of Deformable Bodies. In S. Coquillart, W. Straßer, and P. Stucki, editors, *From Object Modelling to Advanced Visual Communication.* Springer–Verlag, 1994.
6. M-P. Gascuel, A. Verroust, and C. Puech. Animation with Collisions of Deformable Articulated Bodies. In *1^{st} Eurographics workshop on Animation & Simulation*, 1990.
7. D. Crochemore J. Louchet, X. Provot. Evolutionary Identification of Cloth Animation Models. In D. Terzopoulos and D. Thalmann, editors, *Computer Animation and Simulation '95 (Proceedings of 6^{th} Eurographics Workshop on Animation & Simulation)*, pages 30 – 43. SpringerWien, 1995.

8. A. Luciani, S. Jimenez, J. L. Florens, C. Cadoz, and O. Raoult. Computational Physics : A Modeler-Simulator for Animated Physical Objects. In *Proceedings of Eurographics '91*, 1991.
9. G. Miller and A. Pearce. Globular Dynamics : A Connected Particle System for Animating Viscous Fluids. *Computer and Graphics*, 13(3):305–309, 1989.
10. G. S. P. Miller. The Motion Dynamics of Snakes and Worms. *Computer Graphics*, 22(4):169–178, August 1988.
11. L. F. Palazzi and D. R. Forsey. A Multilevel Approach to Surface Response in Dynamically Deformable Models. In *Computer Animation '94*, 1994.
12. X. Provot. Deformation Constraints in a Mass-Spring Model to describe Rigid Cloth Behaviour. In *Proceedings of Graphics Interface '95*, 1995.
13. T. W. Sederberg and S. R. Parry. Free Form Deformation of Solid Geometric Models. *ACM SIGGRAPH Computer Graphics*, 20(4):151–160, Aug 1986.
14. D. Terzopoulos and K. Fleischer. Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture. *Computer Graphics*, 22(4):269–278, August 1988.
15. D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically Deformable Models. *Computer Graphics*, 21(4):205–214, July 1987.
16. D. Terzopoulos and A. Witkin. Physically Based Models with Rigid and Deformable Components. *IEEE Computer Graphics & Applications*, pages 41–51, November 1988.
17. J.A. Thingvold and E. Cohen. Physical Modeling with B-Spline Surfaces for Interactive Design and Animation. In *1990 Symposium on Interactive Computer Graphics*, pages 129–137. ACM SIGGRAPH Computer Graphics, 1990.
18. P. Volino and N. M. Thalmann. Efficient Self-collision Detection on Smoothly Discretized Surface Animations Using Geometrical Shape Regularity. *Computer Graphics Forum* (Proceedings of Eurographics '94), 13(3), 1994.

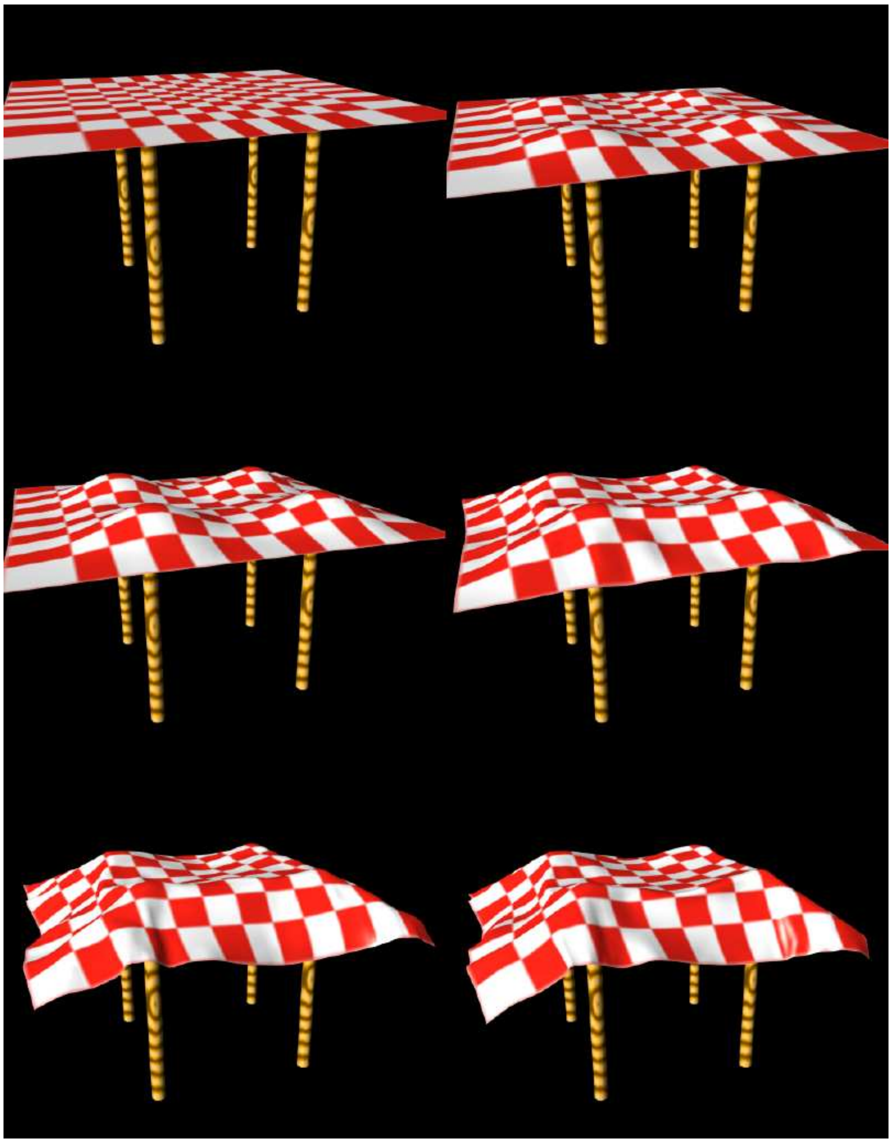


Figure 8: Six images taken from the pole animation.