Deformation and Fracturing Using Adaptive Shape Matching with Stiffness Adjustment

Makoto Ohta, Yoshihiro Kanamori, Tomoyuki Nishita

The University of Tokyo

7-3-1 Hongo, Bunkyo-ku

Tokyo, 113-0033 Japan

Tel. (+81)03 5841 4096 Fax. (+81)03 5803 7288

email: {mktoota,pierrot,nis}@nis-lab.is.s.u-tokyo.ac.jp

Abstract

This paper presents a fast method that computes deformations with fracturing of an object using a hierarchical lattice. Our method allows numerically stable computation based on so-called shape matching. During the simulation, the deformed shape of the object and the condition of fracturing are used to determine the appropriate detail level in the hierarchy of the lattices. Our method modifies the computation of the stiffness of the object in different levels of the hierarchy so that the stiffness is maintained uniform by introducing a stiffness parameter that does not depend on the hierarchy. By merging the subdivided lattices, our method minimizes the increase of computational cost.

Keywords: interactive deformation, soft body, shape matching, fracturing

Introduction

Physically-based simulations of deformations and fracturing can provide realistic animations of everyday materials such as cloth, skin of animals or human characters, and thus have been actively studied since the pioneer work by Terzopoulos [1]. For interactive applications like games, however, a large number of existing techniques are not feasible due to the lack of requirements, i.e., computational efficiency, robustness or controllability of deformations.

To fulfill these requirements, Müller *et al.* [2] recently established a simple technique called *shape matching*. Their approach approximates an object with a set of particles. The particles are moved independently according to external forces, and then pulled to the optimal positions that are found by calculating a rigid transformation that matches the original configuration of particles to the current configuration. While this computation is derived geometrically and not based on physical laws, it yields visually-plausible deformations efficiently in a numerically-stable way. Shape matching is considered promising for the use in entertainment as several researches extend this direction. Rivers and James [3] proposed a method that utilizes a uniform lattice to accelerate the computation and to handle smoother deformations than those of Müller *et al.* Steinemann *et al.* [4] introduced an adaptive lattice to further accelerate shape matching and also used the lattice for dynamic level-of-detail control.

Although the adaptive technique by Steinemann *et al.* offered a large contribution to improve the efficiency of shape matching, it has a problem that the results of deformations by their method largely differs from those using a uniform lattice. This is because, in the adaptive lattice, optimal positions calculated for consecutive levels are largely different, which may prevent users from predictable parameter tuning. As for fracturing, their method maintains connections between cells in the lattice in order to detect topological changes and demonstrates explicit cutting of deformable objects. However, it does not provide a mechanism to handle fracturing as the results of large deformations when using an adaptive lattice.

In this paper, we present an adaptive shape matching technique that significantly alleviates the deformation artifacts of Steinemann *et al.*'s method. Our method adjusts the behaviors of deformations between different levels in an adaptive lattice with negligible cost. As for fracturing, we offer an algorithm to adaptively subdivide the lattice to create complex crack surfaces caused by large deformations. Excessively subdivided regions around crack surfaces are merged in order to reduce the computational costs. This extension is quite beneficial for maintaining deformations interactive even for a user session where complex fracturing frequently occurs.

Related Work

Physically-based simulations of deformable objects have significantly advanced in the two decades [1] in computer graphics. Representatives of physically-based methods are a *mass-*

spring system and a *finite element method* (FEM). Mass-spring systems [1] are fast to compute, easy to understand and implement and thus widely used, however, suffer from numerical stability. More sophisticated approaches include FEMs. While FEMs and their variants are well matured in material engineering, they are also introduced in the field of computer graphics [5, 6, 7, 8, 9, 10, 11, 12]. There are also mesh-free formulations [2, 13, 14]. These techniques simulate realistic deformations based on physically accurate computation, but are performed offline due to the high computational costs. Müller *et al.* [15, 16] significantly accelerated deformations based on a FEM using simplified formulation and a low-resolution mesh. It is worth mentioning that they also handled fracturing that occurs as a result of extreme deformations in real time, however, the rough crack surfaces due to the low-resolution mesh yields poor visual quality. Please refer to [17] for more details on the recent advances in physically-based techniques.

Müller *et al.* [2] proposed a deformation technique called *shape matching*, targeting at interactive applications such as games and surgical simulations. The shape matching technique approximates an object with a set of particles, and thus this technique can be categorized into mesh-less methods. While this computation is not based on physical laws, it is fast and numerically stable, and yields visually plausible deformations. In the next section, we introduce the algorithm of shape matching as well as its variants [3, 4] because they form the basis of our method.

Deformation through Shape Matching

Our method is based on shape matching because it is fast and numerically stable under all conditions, and thus suitable for interactive applications. The following sections explain the concept of shape matching and discuss the possible problems.

Point-based Deformation Through Shape Matching

Here we explain the fundamental algorithm of shape matching proposed by Müller *et al.* [2]. Their method approximates an object with a set of particles. In each time step, the particles are moved independently according to external forces, and then pulled to the positions of a certain shape called the *goal shape* (see Figure 1 (a)). The goal shape is found by assuming the object were a rigid body and by aligning the particles with their original positions. This computation is similar to the iterative closest point (ICP) algorithm [18] in the computer vision community. Specifically, a rotation matrix $\mathbf{R}$ and the translation vectors are computed to minimize the sum of the squared distances of each node i ($i = 0, 1, 2, \cdots$) between the initial position $\mathbf{x}_i^0$ and the current position $\mathbf{x}_i$. The translation vectors are given as centers of mass, i.e., $\mathbf{x}_{cm} = \sum_i m_i \mathbf{x}_i / \sum_i m_i$ and $\mathbf{x}_{cm}^0 = \sum_i m_i \mathbf{x}_i^0 / \sum_i m_i$. Concerning with the rotation matrix $\mathbf{R}$, first, a matrix $\mathbf{A}$ that contains both scaling and rotation is calculated:

$$\mathbf{A} = (\sum_i m_i \mathbf{p}_i \mathbf{q}_i^T)(\sum_i m_i \mathbf{q}_i \mathbf{q}_i^T)^{-1}, \tag{1}$$

where m_i is mass of each node i, $\mathbf{p}_i = \mathbf{x}_i - \mathbf{x}_{cm}$ and $\mathbf{q}_i = \mathbf{x}_i^0 - \mathbf{x}_{cm}^0$. The rotation matrix $\mathbf{R}$ can be found via polar decomposition $\mathbf{A} = \mathbf{RS}$, where $\mathbf{S}$ is the scaling part of the matrix

$\mathbf{A}$. The goal position $\mathbf{g}_i$ of each node i is then computed as

$$\mathbf{g}_i = \mathbf{R}(\mathbf{x}_i^0 - \mathbf{x}_{cm}^0) + \mathbf{x}_{cm}, \tag{2}$$

and the velocity $\mathbf{v}_i$ and the position $\mathbf{x}_i$ of each node i can be decided as

$$\mathbf{v}_i(t+h) = \mathbf{v}_i(t) + \alpha \frac{\mathbf{g}_i(t) - \mathbf{x}_i(t)}{h} + h\frac{\mathbf{f}_{ext}(t)}{m_i}, \tag{3}$$

$$\mathbf{x}_i(t+h) = \mathbf{x}_i(t) + h\mathbf{v}_i(t+h), \tag{4}$$

where t is the present time, h is the interval of a time step, $\mathbf{f}_{ext}$ is an external force and α $(0 < \alpha \leq 1)$ is a parameter that simulates stiffness.

Müller *et al.*'s method uses β $(0 \leq \beta < 1)$ to extend Equation (2) to

$$\mathbf{g}_i = (\beta\mathbf{A} + (1-\beta)\mathbf{R})(\mathbf{x}_i^0 - \mathbf{x}_{cm}^0) + \mathbf{x}_{cm}. \tag{5}$$

β controls the stiffness of the object; the object becomes soft if β is large while the object becomes hard if β is small. However, their method is limited to simple deformations and cannot handle complicated deformations. To alleviate this, Müller *et al.* also proposed the use of *clustering*; dividing an object to some clusters, performing shape matching on each cluster and then blending the deformations between clusters.

Lattice-based Deformation Through Shape Matching

Rivers and James proposed *lattice shape matching* (LSM) [3] to make the deformations much smoother than the clustering results of Müller *et al.* Additionally, LSM can simulate fracturing.

The overview of LSM is as follows. First, LSM approximates an object with a uniform lattice, and places *nodes* at the lattice points. The nodes are used as particles for shape matching. LSM then makes a cubical cluster region for each node to smooth the behavior of deformations. For each cluster region, LSM performs shape matching with regard to the nodes in the cluster region.

The size of the cluster region is determined by $w \in \{1, 2, \dots\}$; the length of an edge of the cluster region is $2w+1$ (see Figure 1 (b)). w is used to control the stiffness of the object.

A problem of LSM is about its poor scalability. Because LSM uses a uniform lattice, it suffers from high computational costs for representing complicated shapes or fine crack surfaces.

Lattice-based Deformation Applying Adaptive Subdivision

Steinemann *et al.* [4] performed shape matching using an octree-based hierarchical data structure instead of a uniform lattice. Their method can handle complicated objects efficiently by adaptively and dynamically changing the subdivision levels of the lattice according to the viewpoint or the state of deformations. However, the subdivision levels are not dynamically changed in the case of fracturing, and thus their method is not so efficient for fracturing. Even worse, it holds a problem that, when changing the subdivision levels, the behavior of deformations changes because the goal positions are different between the levels and the consistency between them is not considered.

Proposed Method

Overview

Our method handles elastic objects in 3D, and deals with collisions, deformation and fracturing. Input objects are assumed to be represented by a polygonal mesh.

Our method simulates deformation and fracturing using an adaptively subdivided lattice, preserves the stiffness of the object when the subdivision level of a lattice is changed and enables visually more plausible animations of deformation and fracturing than previous methods. Our method consists of a preprocessing step and run-time computation of deformation and fracturing, as described below.

Preprocessing

In preprocessing, our method creates an octree-based data structure. First, our method represents an object with a lattice at each subdivision level, and places nodes at the center of mass of each lattice cell. Second, it creates an octree by establishing creating parent-child relationships between nodes that are in equivalent positions at different subdivision levels. Finally, at each subdivision level, if the mass m_i of a node i exceeds a threshold, we select the parent node and remove its children for the purposes of the deformation computation; otherwise, the children nodes remain (see Figure 2 (a)).

A cluster for a node at the finest subdivision level is defined as a cubical region whose size is $2w + 1$. The cluster for a parent node is defined as the union of all cluster regions of its children. By repeating this process recursively, we can define a cluster region for each node.

Deformation

Shape Matching Using an Adaptively Subdivided Lattice

Our method computes deformation through shape matching for each cluster region. However, a lattice cell does not always lie completely within a cluster region in the case of an adaptively subdivided lattice, and Equation (1) needs to be modified as follows. If a lattice cell whose center is node i is partially covered by a cluster region R, we recursively find node i's descendants that are completely within the region and compute the sum of their mass $m_{i,R}$. The matrix $\mathbf{A}$ in Equation (1) is then substituted by

$$\mathbf{A} = (\sum_{i \in R} m_{i,R} \mathbf{p}_i \mathbf{q}_i^T)(\sum_{i \in R} m_{i,R} \mathbf{q}_i \mathbf{q}_i^T)^{-1}, \tag{6}$$

which uses $m_{i,R}$ instead of m_i in Equation (1) (see Figure 2 (b)).

Stiffness Consistency when The Subdivision Level of Nodes are Changed

Steinemann *et al.*'s method [4] bears a problem that the behaviors of deformation differ between different subdivision levels. This happens because the goal shapes of parent nodes are different from those of their children and therefore inconsistent.

Our method improves the consistency of the deformations before and after changing the subdivision level of a node by using the parameter β (see Equation (5)), which controls the stiffness of the object and is independent of the size of a cluster region. Each node's goal position $\mathbf{g}_i$ is decided by Equation (5).

The Derivation of Appropriate β

To simplify the explanation, we show the derivation of the value of β in 1D. In this case, the shape of the object is a segment and its data structure is a binary tree; each parent node has two children.

We decide that the mass of a node at the smallest subdivision level is 1, the value of β used for shape matching in the cluster region of this node is 0, the size of this cluster region is w and the mass of a node i is m_i. If the size of a cluster region is w, the length of a cluster region corresponding to a node with the smallest subdivision level is $2w + 1$ and the length of a cluster region of its parent node is $2w + 2$, because this region is the union of regions of its children nodes (see Figure 3 (a)). Now, we assume that an object whose initial length is $2w + 2$, is deformed by an external force and its length is x (see Figure 3 (b)). Regarding the two children nodes, the length of the initial shape of this object in a cluster region of one child node is $2w + 1$, and, after deformation, its length becomes $\frac{(2w+1)x}{2w+2}$, and the position of its center of mass is $\frac{(2w+1)x}{2(2w+2)}$ from the left end of this segment. If we match the cluster region of the child node, whose length is $2w + 1$, to this segment by fitting these centers of mass, the position of the left end of this cluster region is $\frac{(2w+1)x}{4w+4} - \frac{2w+1}{2} = \frac{(2w+1)(x-2w-2)}{4w+4}$

from the left end of this segment (see Figure 3 (c)). Considering shape matching for another child node, the total length of the goal shape is $x - 2 \times \frac{(2w+1)(x-2w-2)}{4w+4} = \frac{x}{2w+2} + 2w + 1$ (see Figure 3 (d)). In the case of the parent node, if we consider shape matching without scaling using Equation (2), the length of its goal shape is $2w + 2$ (see Figure 3 (e)). On the contrary, if we use shape matching with scaling, the length of the goal shape of the parent node is x (see Figure 3 (f)) and this case corresponds to shape matching using the following equation: $\mathbf{g}_i = \mathbf{A}(\mathbf{x}_i^0 - \mathbf{x}_{cm}^0) + \mathbf{x}_{cm}$. To fit the result we integrate these two results at the rate $(1 - \beta) : \beta$; applying shape matching using Equation (5) to the result using the two children nodes, we get the equation $(1 - \beta)(2w + 2) + \beta x = \frac{x}{2w+2} + 2w + 1$, from which we can obtain the value of β;

$$\beta = \frac{1}{2w + 2}. \tag{7}$$

β is determined only by w and is independent of the deformed shape x. The above is the case of a node at the smallest subdivision level and its parent node. In the same way, we calculate the value of β for nodes with coarser subdivision levels recursively, the value of β of each node in 1D is defined as

$$\beta_{1D} = 1 - \frac{2w + 1}{2w + m_i}. \tag{8}$$

From this 1D case and experiments, the value of β is defined as, in 2D,

$$\beta_{2D} = 1 - (\frac{2w + 1}{2w + \sqrt{m_i}})^2, \tag{9}$$

and in 3D,

$$\beta_{3D} = 1 - (\frac{2w+1}{2w+\sqrt[3]{m_i}})^3. \tag{10}$$

Changing the Subdivision Level of Nodes

We explain the condition for changing the subdivision level of nodes. During shape matching in each cluster region, the scaling matrix $\mathbf{S}$ is calculated via polar decomposition $\mathbf{A} = \mathbf{RS}$. The diagonal components in the matrix $\mathbf{S}$ show the rate of magnification of the cluster region along each axis. If these values exceed a threshold, the strain in this cluster region by deformation is considered large and the node is replaced by its children nodes.

Fracturing

Fracturing is decided by the distance between two nodes neighboring each other at the smallest subdivision level. The neighbor nodes are the 26 neighbors along three axes and diagonal directions. When an object is fractured, crack surfaces are generated as follows. Suppose that fracturing occurs between lattice cells l_i and l_j that are neighbors each other, and crack surfaces are generated between their adjacent faces f_i and f_j. The borders of these new crack surfaces are not matched to polygonal meshes, therefore, our method uses Marching Squares. See Figure 4 for an example of fracturing of a sphere. If the lattice cells are simply separated off, the boundary of the crack surfaces does not follow the object's surface (i.e., polygonal mesh), and the crack surfaces exhibit brick-like regular displacements of the lattice. To reform the boundary, our method uses Marching Squares; the boundary is dertermined whether each vertex of f_i and f_j is inside or outside of the object and displaced according to the distance to the object's surface. To make the crack surfaces more plausible, our method gives an offset to each vertex of f_i and f_j using Perlin Noise.

Merging Nodes

In our method, nodes that have a common parent are dynamically merged during fracturing if the rates of magnification along each axis are below a threshold in order to reduce the computational cost that increases with the number of nodes. Nodes to be merged are grouped using connectivity information and then substituted with their parent node. Note that the number of children for a parent node might be reduced by fracturing. If so, the position and the mass of such a parent node are updated to the center of mass and the sum of mass of its children nodes, respectively (see Figure 5 (a)). Even after merging, the crack surfaces can be represented in detail because the vertex coordinates of crack surfaces are interpolated in the cell of the parent node.

Results

The experiments in this paper were conducted on a PC with a Pentium 4 3.0GHz and 1.0GB RAM, and the program was written with C++ and OpenGL. We show the results of the 3D simulations of deformations and fracturing with collisions between other objects and the deformable object itself. For collision detection, our method allocates a sphere that centers each node and has a radius proportional to the mass of the node, and detects sphere-sphere collisions.

Figures 7 (a) - (c) show the results of the simulation with the Bunny model. Lattice cells are dynamically subdivided if the distortions are large, separeted off if fracturing occurs, and merged if the distortions become small. The number of nodes representing the Bunny is 2248 in LSM while 207 in our method. In addition, the computational time is 59 msec per timestep in LSM while 3 msec per timestep in our method. That is, our method achieved about 20 times speed up compared with LSM.

Figures 7 (d) - (i) show the results of the simulation uses the Elephant models. The performance of this simulation are 45 fps with 5 objects, 14 fps with 10 objects and 7 fps with 15 objects.

Comparing Deformations

In this section, we discuss the effect of stiffness control based on the parameter β. We compare Steinemann *et al.*'s method and our method with regard to the accuracy of deformations using a uniform lattice, i.e., LSM. Each simulation starts from the same initial condition. The accuracy of deformations are calculated as the differences of coordinates of nodes, and their averages and standard deviations are plotted. The coordinates are normalized so that the longest diagonal length of the bounding box becomes 1.

Figure 6 shows the results of freely-falling Glob models simulated by the three methods. After 500 time steps (Figure 6(b)), the average and standard deviation are: 0.203 and 0.113 by Steinemann *et al.*'s method, 0.077 and 0.036 by our method, respectively. Compared with Steinemann *et al.*'s method, our method yields approximately $1/3$ averages and standard deviations. Figure 6(c) shows the graph of the averages. As demonstrated in these results, we conclude that our method is better in terms of numerical accuracy and the visual quality than Steinemann *et al.*'s method.

Effect of Merging Nodes on Computational Cost

We show the effect to computational cost of merging nodes in the 3D simulation of fracturing with and without merging where the Bunny model is fixed by its head and fractured by the force of gravity (see Figures 7 (a) - (c)). Figure 5 (b) shows the graph of the computational cost, which is reduced by merging nodes.

Conclusion and Future Work

This paper has presented a fast simulation technique of deformation and fracturing based on shape matching by representing objects with a hierarchical lattice, and has represented crack surfaces in detail by applying adaptive subdivision. Our method has the following three contributions: Our method

- does not change the stiffness of a deformable object, unlike Steinemann *et al.*'s method, when adaptive subdivision is applied,

- applies dynamic subdivision to fracturing objects considering the condition of deformation and fracturing, and
- reduces computational cost by merging nodes if these distortions are little.

As for a limitation of our method, external forces are slowly propagated through a deformable object because, within a single time step, the external forces only affects the cluster regions that contain particles pushed or pulled by the forces. Moreover, the deformation methods based on shape matching do not account for the laws of physics; for example, the volume of a deforming object is not preserved.

For future work, we would like to solve these problems without loss of the interactive speed.

References

[1] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *Proc. of SIGGRAPH 87*, pages 205–214, 1987.

[2] M. Müller, B. Heidelberger, M. Teschner, and M. Gross. Meshless deformations based on shape matching. In *Proc. of SIGGRAPH 05*, pages 471–478, 2005.

[3] A. R. Rivers and D. L. James. FastLSM: Fast lattice shape matching for robust real-time deformation. In *Proc. of SIGGRAPH 07*, number 82, 2007.

[4] D. Steinemann, M. A. Otaduy, and M. Gross. Fast adaptive shape matching deformations. In *Proc. of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 87–94, 2008.

[5] A. W. Bargteil, C. Wojtan, J. K. Hodgins, and G. Turk. A finite element method for animating large viscoplastic flow. In *Proc. of SIGGRAPH 07*, number 16, 2007.

[6] G. Debunne, M. Desbrun, M. P. Cani, and A. H. Barr. Dynamic real-time deformations using space and time adaptive sampling. In *Proc. of the 28th annual conference on Computer graphics and interactive techniques*, pages 31–36, 2001.

[7] D. Metaxas and D. Terzopoulos. Dynamic deformation of solid primitives with constraints. In *Proc. of SIGGRAPH 92*, pages 309–312, 1992.

[8] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler. Stable real-time deformations. In *Proc. of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 49–54, 2002.

[9] N. Molino, Z. Bao, and R. Fedkiw. A virtual node algorithm for changing mesh topology during simulation. In *Proc. of SIGGRAPH 04*, pages 385–392, 2004.

[10] J. F. O ’Brien and J. K. Hodgins. Graphical modeling and animation of brittle fracture. In *Proc. of SIGGRAPH 99*, pages 287–296, 1999.

[11] J. F. O ’Brien, A. W. Bargteil, and J. K. Hodgins. Graphical modeling and animation of ductile fracture. In *Proc. of SIGGRAPH 02*, pages 291–294, 2002.

[12] N. Sukumar, N. Mos, B. Moran, and T. Belytschko. Extended finite element method for three-dimensional crack modeling. *International Journal for Numerical Methods in Engineering 48*, 11:1549–1570, 2000.

[13] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. Point based animation of elastic, plastic and melting objects. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 141–151, 2004.

[14] M. Pauly, R. Keiser, B. Adams, P. Dutre, M. Gross, and L. J. Guibas. Meshless animation of fracturing solids. In *Proc. of SIGGRAPH 05*, pages 957–964, 2005.

[15] M. Müller and M. Gross. Interactive virtual materials. *Graphics Interface 2004,* pages 239–246, 2004.

[16] M. Müller, M. Teschner, and M. Gross. Physically-based simulation of objects represented by surface meshes. In *Proc. of Computer Graphics International 2004*, pages 26–33, 2004.

[17] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson. Physically based deformable models in computer graphics. *Computer Graphics Forum 25*, 4:809–836, 2005.

[18] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, A 4 4:629–642, 1987.

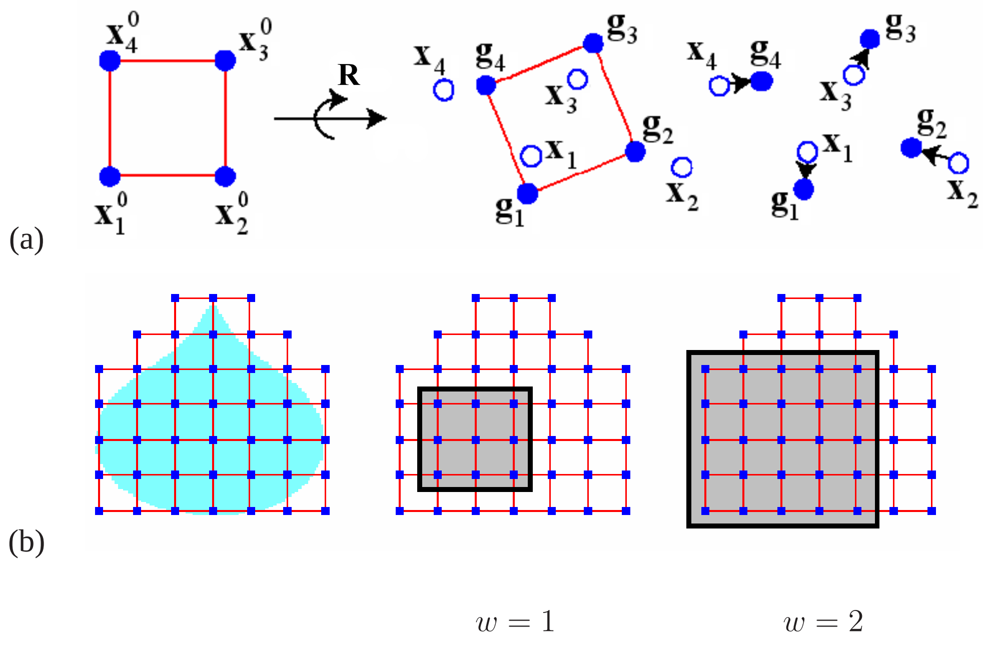
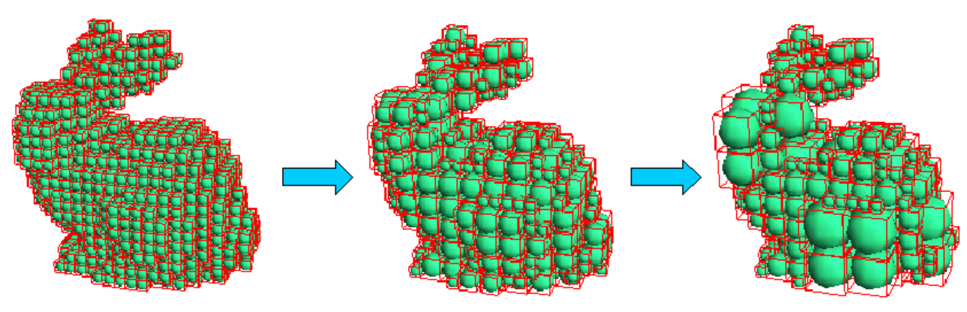


Figure 1: **2D illustrations for shape matching and cluster regions:** (a) A circle represents a node and lines connecting circles represent the initial shape of the object. The **left** image is the initial shape, and the **middle** image is the goal positions through shape matching. The arrows in the **right** image is the restoration force added to each node towards the goal shape. (b) An object is enclosed by a lattice and nodes are placed on those lattice points (**left**). Note that, for 3D deformations by the methods of Steinemann *et al.* and ours, nodes locate at centers of lattice cells, not at lattice points. A part surrounded by black thick lines shows a cluster region. It becomes a 3×3 square region if $w = 1$ (**middle**) and a 5×5 square region if $w = 2$ (**right**).

(a)



(b)

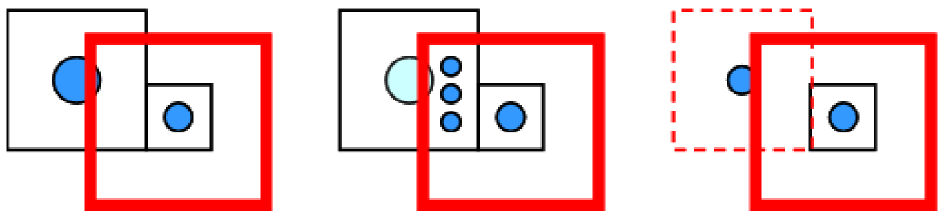


Figure 2: **Adaptive subdivision of a deformable model:** (a) First, we start that all nodes are the finest subdivision level (**left**). If the mass of each node exceeds a threshold, we change children nodes to this node. The (**middle**) image shows the result to change to nodes at one time, and the (**right**) image shows the result to change to nodesat two times. (b) The red square is a cluster region in question, the blue circles represent nodes and the black squares are a lattice. If a part of a lattice is in a cluster region (**left**), the mass of a part of the node in this cluster region are computed by checking its descendant (**middle**), and we change the mass of the node in this cluster region to $m_{i,R}$ in Equation (6) (**right**).

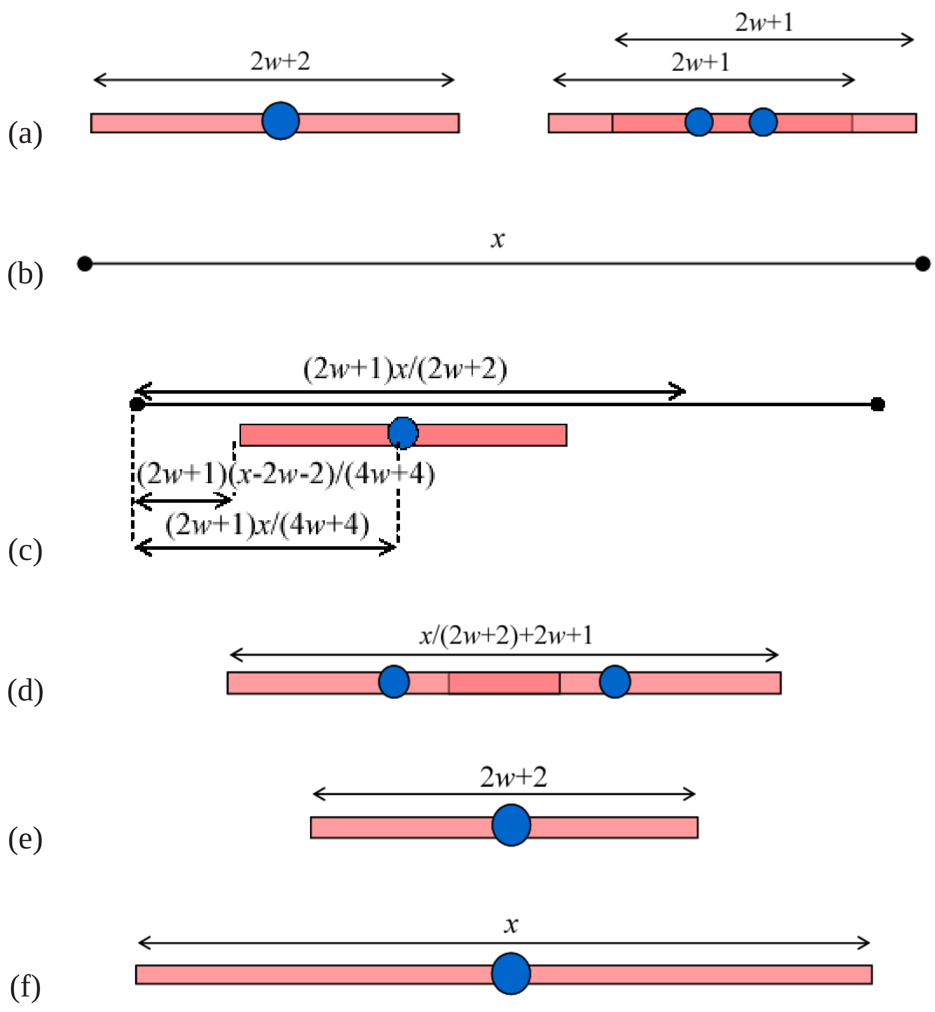


Figure 3: **Modification of adaptive shape matching for consistency in a 1D example:** Blue circles show nodes and red bars show the goal shapes. Suppose that (a) a parent node (**left**) and its two children nodes (**right**) are deformed so that (b) the length of the shape becomes x. Without scaling, the length of the goal shape becomes (d) $x/(2w+2)+2w+1$ when applying shape matching to (c) each child node, while it becomes $2w+2$ when applying to (e) the parent node. We linearly blend the results (f) with and (e) without scaling using an appropriate parameter β so that the length of the goal shape matches with (d) that with the two children nodes.

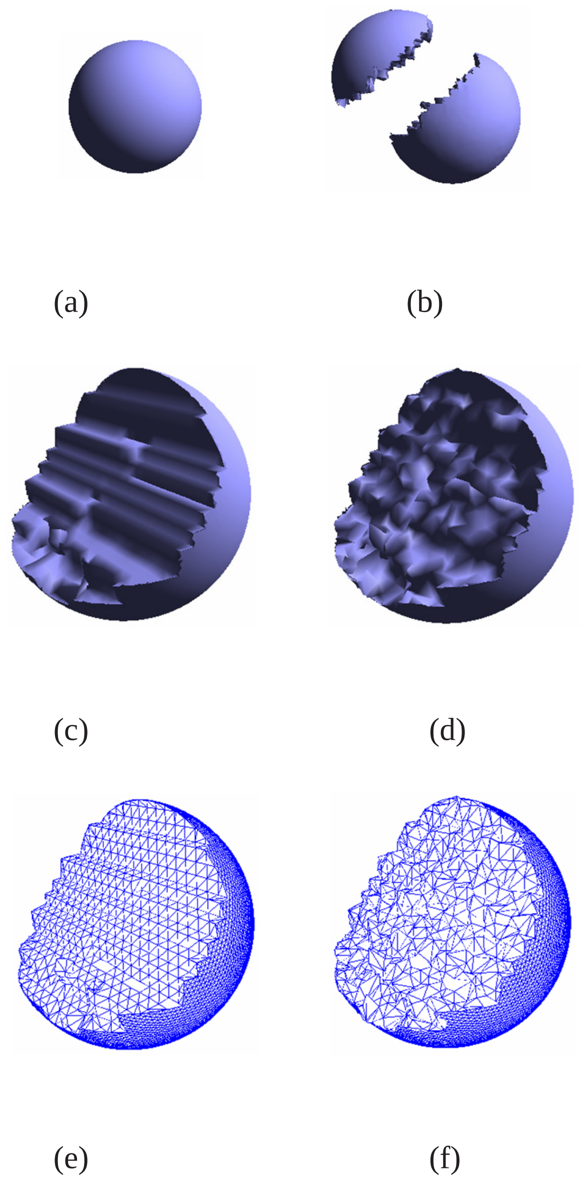
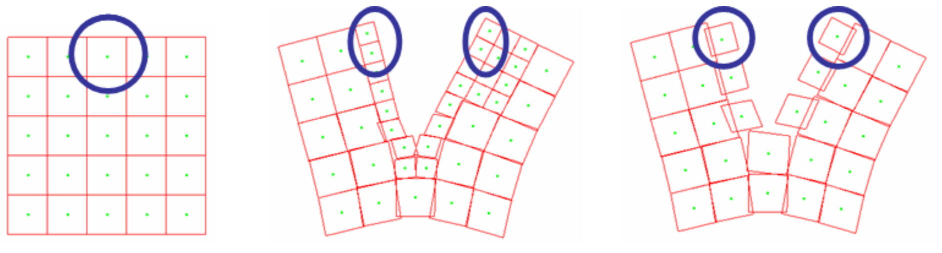


Figure 4: **The process of creating crack surfaces:** When (a) a sphere is (b) fractured, crack surfaces are generated (c-f). (c, e) If the crack surfaces are represented with raw lattice cells, the results look unnatural because of brick-like displacements. To improve the visual plausibility, (d, f) we displace the surface using Perlin Noise.

(a)



(b)

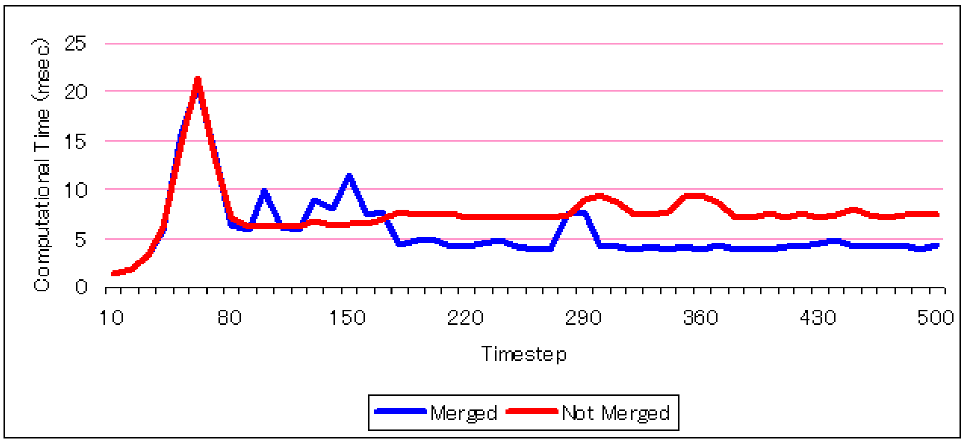
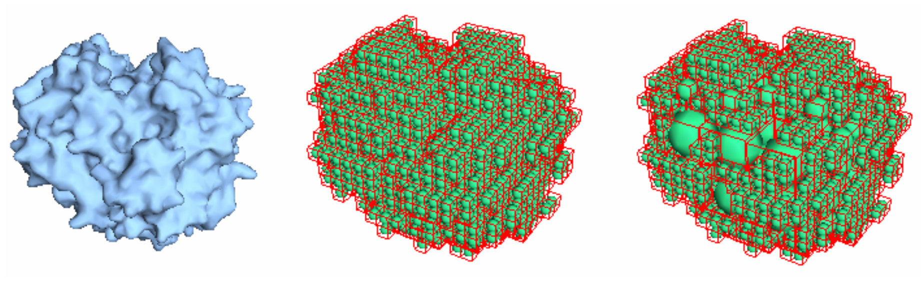
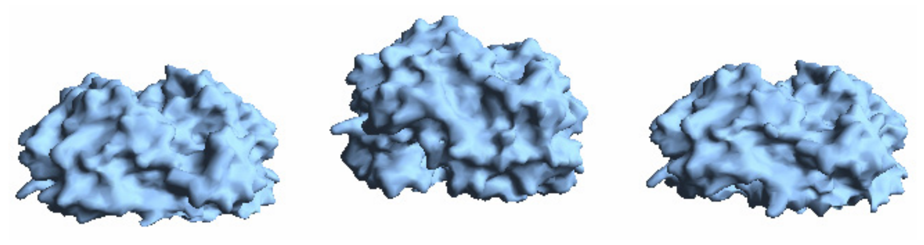


Figure 5: **The illustrations of merging nodes and the graph of computational times:** (a) The red lines represent a lattice and the green points represent nodes. If we look at nodes surrounding blue circles, the number of nodes is one before fracturing (**left**) and the number of nodes is four and these are divided into two groups (**middle**). The nodes in each group are then merged and substituted by a parent node (**right**). (b) In this graph, the vertical axis shows computational time per time step and the horizontal axis shows the time step. These computational costs are nearly equal before fracturing (80 time step). After fracturing, simulation time without merging is about constantly 8 msec. On the contrary, after fracturing, simulation time with merging temporarily increases to compute merging nodes, but, after this, simulation time is about 4 msec.

(a)



(b)



(c)

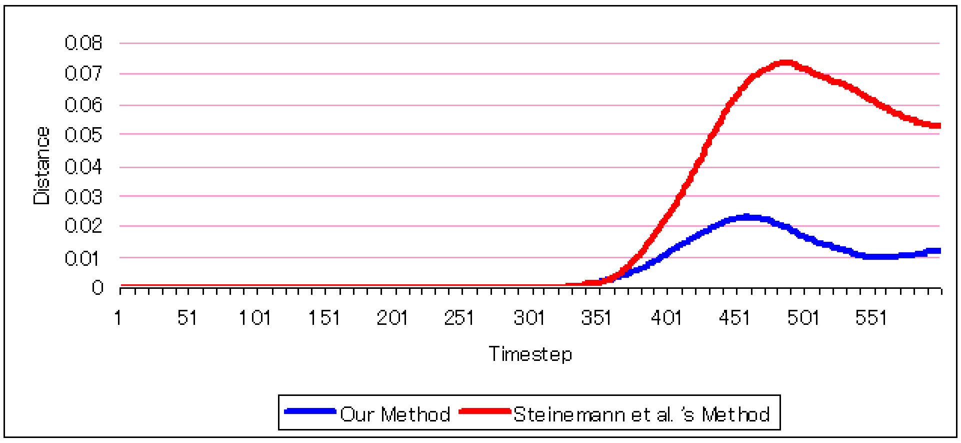
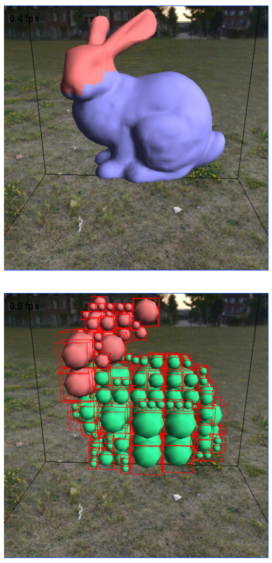
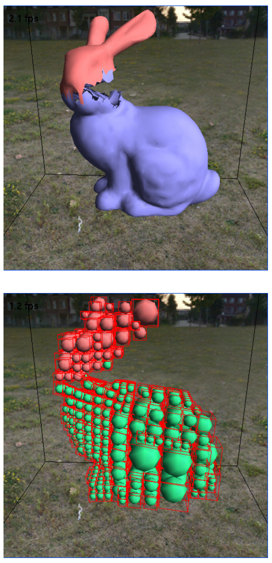


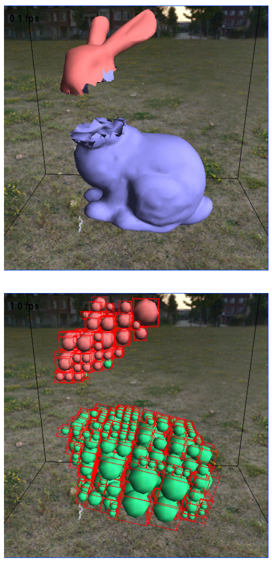
Figure 6: **Comparison of deformation by each method:** (a) From left, the object represented by polygonal meshes, a uniform lattice and an adaptively-subdivided lattice. The green spheres represent nodes centering lattice cells. (b) From left, the results of LSM, Steinemann *et al.*'s method and the proposed method. (c) At 320 time step, an object collides with the ground and deforms, and after that, the difference becomes large. Our method yields visually better deformation as the difference with respect to the result of LSM is smaller than that of Steinemann *et al.*



(a) No. of nodes: 207

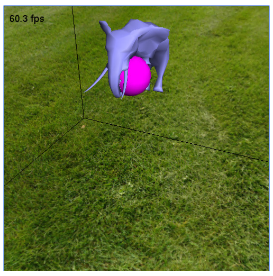


(b) No. of nodes: 623

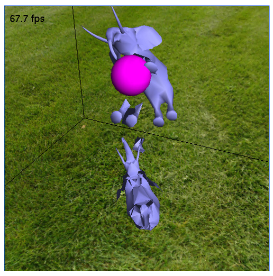


(c) No. of nodes: 343

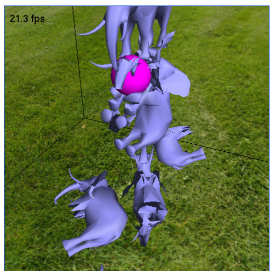
(d)



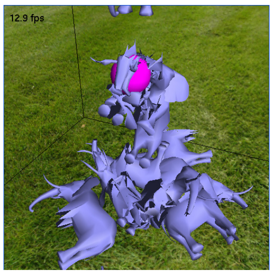
(e)



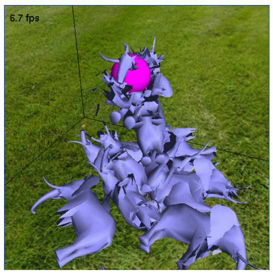
(f)



(g)



(h)



(i)

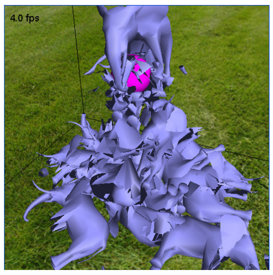


Figure 7: **Fracturing of 3D models:** (a) - (c) The head (red) of the Bunny model is fixed, and the model is fractured by the gravity. The results in the top row are rendered with polygonal meshes while the images in the second row show show nodes (green spheres) and lattices (red lines). (d) - (i) Fifteen Elephant models are fallen, collide with each other and fractured. If an object touched the pink sphere in the middle, the touching part is fixed.