Check for updates

Simulation

Simulation: Transactions of the Society for Modeling and Simulation International
2017, Vol. 93(10) 797–807
© The Author(s) 2017
DOI: 10.1177/0037549717699073
journals.sagepub.com/home/sim
SAGE SCS

A fast mass spring model solver for high-resolution elastic objects

Mianlun Zheng, Zhiyong Yuan, Weixu Zhu and Guian Zhang

Abstract

Real-time simulation of elastic objects is of great importance for computer graphics and virtual reality applications. The fast mass spring model solver can achieve visually realistic simulation in an efficient way. Unfortunately, this method suffers from resolution limitations and lack of mechanical realism for a surface geometry model, which greatly restricts its application. To tackle these problems, in this paper we propose a fast mass spring model solver for high-resolution elastic objects. First, we project the complex surface geometry model into a set of uniform grid cells as cages through *cages mean value coordinate method to reflect its internal structure and mechanics properties. Then, we replace the original Cholesky decomposition method in the fast mass spring model solver with a conjugate gradient method, which can make the fast mass spring model solver more efficient for detailed surface geometry models. Finally, we propose a graphics processing unit accelerated parallel algorithm for the conjugate gradient method. Experimental results show that our method can realize efficient deformation simulation of 3D elastic objects with visual reality and physical fidelity, which has a great potential for applications in computer animation.

Keywords

Elastic object, fast mass spring model solver, high-resolution geometry model, *cages mean value coordinates, conjugate gradient method, graphics processing unit acceleration

1 Introduction

With the enormous improvement in computer's calculation ability, computer graphics and virtual reality technology are widely applied in computer animation, games and virtual surgery etc. where 3D elastic object simulation plays a central role.[1–3] Based on this, efficient and visually realistic deformation effects of high-resolution geometry models of 3D elastic objects are needed in many scenes. By far, the most widely used methods to simulate elastic objects are through geometric models, finite element models and mass spring models (MSMs).

The geometric models are typically coordinates-based models, such as *cages mean value coordinates (*Cages MVC) which can produce mesh deformation quite efficiently, but lack mechanics properties of elastic objects.[4] The finite element method is a kind of numerical solution method which belongs to mechanics and physical problems.[5] In practice, a finite element method requires plenty of complex numerical calculations and it would be difficult to achieve real-time effects with high-resolution models. MSMs discretize the continuous elastic objects to form points as masses and connect them with springs.[6] In this way, the deformation results of elastic objects are obtained by calculating the mechanical changes of springs. This method has characteristics of high velocity and perfect efficiency.

Recently, Liu et al. proposed the fast MSM to solve a standard mass spring system using fast implicit solution, which transfers the mass spring problem into an optimization problem.[7] It is a promising way to generate efficient deformation simulation of low-resolution surface geometry models. However, when confronting simulation of high-resolution 3D elastic objects, the fast MSM fails to reflect the internal structure and mechanics properties of the surface geometry model. Furthermore, the fast MSM is not suitable to achieve real-time computation and high resolution surface geometry models because its efficiency will largely decrease as the model's resolution increases. These limitations greatly affect the application of the fast MSM.

School of Computer, Wuhan University, PR China

Corresponding author:
Zhiyong Yuan, School of Computer, Wuhan University, Wuhan, Hubei 430072, P.R. China.
Email: zhiyongyuan@whu.edu.cn

To tackle the limitations of the fast MSM, in this paper we propose a fast MSM solver for high-resolution elastic objects. First, we project the complex surface geometry model into a set of uniform grids as cages which can reflect the surface geometry model's internal structure and mechanics properties and acquire physical fidelity. Also, we use the *Cages MVC method to construct the mapping relation between the geometry model and simulation cages. Then, the original Cholesky decomposition method in the fast MSM solver is replaced by the conjugate gradient method, which makes the fast MSM solver more efficient, so that it is applicable for the detailed surface geometry model. Finally, we realize the parallel algorithm for the conjugate gradient method by graphics processing unit (GPU) acceleration. This paper achieves efficient deformation simulation of 3D elastic objects with visual reality and physical fidelity.

2 Related work

In recent years, there are quantitative findings concentrating on the deformation simulation of mesh models.[8–10] Among geometric models, Floater put forward MVCs and 3D MVCs, which map the internal vertex coordinates of polyhedron to every vertex through barycenter coordinate mapping.[11,12] Then Tao Ju et al. applied the MVC method to the deformation of a closed triangular mesh, but it has an obvious flaw in the mapping relation realization of non-convex polyhedron.[13] To deal with this problem, Lipman et al. proposed harmonic coordinates, which however has the disadvantage of lacking a shape-preserving property.[14] Then green coordinates are presented to solve the problems above, in which it not only uses positions of polyhedron vertices but also includes vectors of every simple surface of polyhedron into the calculation of internal vertices' locations to achieve better deformation results.[15] These three methods mentioned above all belong to geometric deformation models, meaning that they use a polyhedron (or a cage) to cover the target model and construct a mapping relation between them. By controlling the locations of vertices in the cage and mapping the changed locations back to the original model, the deformed model can finally be rendered. However, these three methods adopt only one cage to control the model during the simulation computation and hence it constrains the effect of deformation. Further, Garcia et al. proposed the *Cages MVC that controls the target model through several cages, which enables the control to become more flexible.[4] And in different cages, the model can have a greater fine and smooth deformation effect by using a smooth function.[4] Besides, Gao et al. proposed a data-driven approach for realistic shape morphing. The method does an interpolation calculation through a pair of source and target models and a database of similar models to reach the deformation effect.[9] Since the geometry based deformation models have advantages of simplicity and velocity, they are widely applied in game and animation fields. However, this method cannot present the physical effect of an object during the process of deformation, which means that it lacks physical reality.

There exists a lot of physics based animation methods in which the finite element method and the MSM are the most widely used ones. Miller proposed the total Lagrangian explicit dynamic (TLED) finite element method which can provide a perfect simulation effect for a non-linear model.[16] This method is different from the Lagrange method in that it allows the precomputation of all the derivatives. Therefore, the method is capable of simulating a non-linear model efficiently, but the TLED method has a critical limitation for time steps in order to ensure stability. For example, for a mesh with 10 elements per side, the time step has to be less than 0.0013 s which leads to the impossibility of real time simulation. In addition to the TLED method, Gilles et al. utilized Lagrangian mechanics to reflect the internal physical characteristics on each sample point of the model,[17] and Geoffrey et al. presented an algorithm for finite element simulation of elastoplastic solids.[18] All these finite element methods above do have problems with the lack of real-time display. For instance, in the work by Gilles et al.,[17] the frame rate for even low resolution turtle model is 7 FPS. Matthieu et al. presented a method for simulating highly detailed geometric models with heterogeneous material properties using very coarse grids.[19] This technique results in significant improvements in an efficient physics based simulation of highly detailed objects. Bargteil and Cohen developed a quadratic tetrahedral element to demonstrate elastic bodies with non-linear rest shapes.[20]

Since the MSM has the feature of simplicity and easy application, it is widely employed in deformation simulation.[21] Liu et al. came up with the fast MSM in which a kind of fast solution for implicit equations for a mass spring system is presented.[7] This method regards a dynamics equation in each time step as a minimization problem. When solving the minimization problem, the linear system equation is independent of the current state while at the final linear solution process. Therefore, we can complete preprocessing computation to achieve the aim of fast solution. However, the physical structure of the MSM is comparatively stable, thus the application of this method is constrained to one-dimensional or two-dimensional structures like hair and cloth. If we apply the 3D geometry mesh model into the MSM, it cannot reflect the internal structure and mechanics properties and fails to acquire physical fidelity, which is necessary to be solved. Moreover, efficiency of the fast MSM is considerable but its advantage becomes weaker when confronting a 3D model. In the experiments, the frame time for a curtain is 50 ms but it is 203 ms for a dog model with the same resolution. Besides fast MSM, Martin et al. realize the simulation effect of various complicated elastic materials on an artistic level through combining example-based simulation and computational mechanics.[22] San-Vicente et al. put forth a cubical MSM design based on a tensile deformation test and a non-linear material model,[23] and its experiment results demonstrate that the MSM can be substituted with non-linear finite elements and can be applied to the simulation of elastic materials such as soft tissue.

3 Basics

3.1 *Cages MVC

The method of *Cages MVC was first proposed by Garcial et al., where it could control the displacement of a target model through a number of divided cages.[4] Moreover, *Cages MVC employs a smooth function to make the control of a target model more flexible and it also enables the cages to obtain more fine and smooth displacement results. According to Floater et al.,[12] the coordinate $\boldsymbol{x}(p)$ of a point p inside a cage which consists of n vertices is calculated by an affine combination function as follows:

$$\boldsymbol{x}(p) = \sum_{i=1}^{n} \lambda_i(v_i, p) * \boldsymbol{x}(v_i) \tag{1}$$

where v_i denotes the ith vertex on the cage and λ_i means the MVC function corresponding to the ith vertex. 3D MVC utilizes tetrahedron as a processing unit, as shown in Figure 1.

λ_i in Equation (1) is defined as follows:

$$\lambda_i = \frac{w_i}{\sum\limits_{j=1}^{n} w_j} \tag{2}$$

$$w_i = \frac{1}{r_i} \sum_{v_i \in T} \mu_{i,T} \tag{3}$$

$$\mu_{i,T} = \frac{\beta_{jk} + \beta_{ij} \boldsymbol{n}_{ij} \boldsymbol{n}_{jk} + \beta_{ki} \boldsymbol{n}_{ki} \boldsymbol{n}_{jk}}{2 \boldsymbol{e}_i \boldsymbol{n}_{jk}} \tag{4}$$

where β_{ij} is the included angle between segment $[p,v_i]$ and $[p,v_j]$ and $\boldsymbol{n}_{ij}$ is the unit normal vector that points inside to the tetrahedron and is perpendicular to triangle $[p,v_i,v_j]$ as follows:

$$\boldsymbol{n}_{ij} = \frac{\boldsymbol{e}_i \times \boldsymbol{e}_j}{\| \boldsymbol{e}_i \times \boldsymbol{e}_j \|} \tag{5}$$

Similarly, we can obtain values for β_{jk}, β_{ki}, $\boldsymbol{n}_{jk}$, and $\boldsymbol{n}_{ki}$. When locations of each vertex v_i on the cage change during the simulation, it will lead to a new $\boldsymbol{x}'(v_i)$. After that we should recalculate the position of target vertex p which is covered by the cage as follows:

$$\boldsymbol{x}'(p) = \sum_{i=1}^{n} \lambda_i(v_i, p) * \boldsymbol{x}'(v_i) \tag{6}$$

However, there exists a problem among the neighboring cages that is position discontinuity. We adopt a smoother function $\boldsymbol{s}(p)$ to replace $\boldsymbol{x}(p)$ obtained before as follows[4]:

$$\boldsymbol{s}(p) = b(p)\boldsymbol{x}(p) + (1 - b(p))J(p) \tag{7}$$

In Equation (7), $b(p)$ stands for the boundary weight function, and $J(p)$ is responsible for guaranteeing smooth transitions among neighboring cages and will behave similarly to standard cage-based transformations as follows:

$$J(p) = \sum_{v_i \in B_c} W(v_i, p) L_{v_i}(p) \tag{8}$$

Here the weight function $W(v_i,p)$ takes charge of computing how much the transformation $L_{v_i}(p)$ from each boundary cage is blended and its value is normalized. B_c represents vertices on the boundary of neighboring cages and $L_{v_i}(p)$ denotes a smooth and local transformation.

Since vertices on the boundary of neighboring cages are covered by a series of cages, we set $j_c(v_i)$ to express all these cages and all the cages adjacent to this cage are Adj_c. In a newly emerged cage $j_c(v_i)$, we obtain MVC function $\lambda(u,p)$ for the boundary vertex v_i, and u is each vertex of $j_c(v_i)$. Through the newly formed coordinate w, we can now solve out $W(v_i,p)$ and $L_{v_i}(p)$ in Equation (8) and therefore $J(p)$ is obtained.

What we can learn from the definition of $b(p)$ is that when p belongs to the boundary of a cage, $b(p)$ equals to 0. But it equals to 1 when p lies on a plane irrelevant to the intersection boundary. Through such feature we can obtain:

$$b(p) = f_h\left(\prod_{c_i \in Adj(c)} \left(1 - \sum_{v_i \in Bc} \lambda(v_i, p)\right)\right) \tag{9}$$

where f_h is a smoothing function with parameter $h \in (0,1]$ which satisfies $f_h(0) = f_h(1) = 0$, $f_h(x) = 1(x \geqslant h)$ and $f_h \geqslant 0$. h is used to contract or extend f in the range of $(0,1]$, and its size can be changed to meet different needs. Obtaining values of $b(p)$ and $J(p)$, we can finally

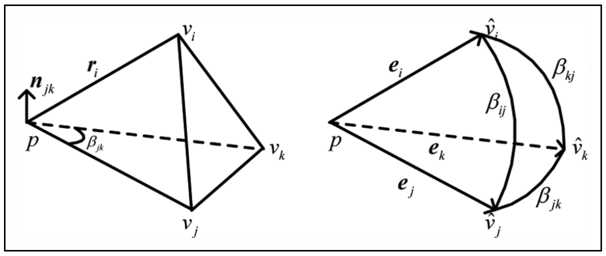

Figure 1. 3D MVC.

calculate the newly formed $\boldsymbol{s}(p)$ which makes it possible to map the deformed cages back to surface mesh model more smoothly.

3.2 Fast MSM

The fast MSM is a method to solve a standard mass spring system that obeys Hooke's Law with a fast implicit solution.[7] During the solving process, the method transfers the mass spring problem into an optimization problem, which is used to obtain accurate deformable results. According to Newton's second law we can obtain:

$$\boldsymbol{q}_{n+1} - 2\boldsymbol{q}_n + \boldsymbol{q}_{n-1} = t^2 M^{-1} \boldsymbol{f}(\boldsymbol{q}_{n+1}) \tag{10}$$

where $\boldsymbol{q}_n \in R^{3m}$ represents the location matrix of each mass at time n. The internal forces of each mass at time n can be described as $\boldsymbol{f}(\boldsymbol{q}_{n+1})$ and $\boldsymbol{f} \in R^{3m}$. t stands for the time step and $\boldsymbol{M} \in R^{3m\times 3m}$ is a diagonal matrix whose element is the weight of each corresponding mass. Since the equation is non-linear, we have to convert it into a linear one for the solution. In addition, in order to simplify the expression, the unknown variable $\boldsymbol{q}_{n+1}$ is expressed as $\boldsymbol{x}$ and the known variables $2\boldsymbol{q}_n - \boldsymbol{q}_{n-1}$ as $\boldsymbol{y}$. Consequently, the equation is simplified as follows:

$$\boldsymbol{M}(\boldsymbol{x} - \boldsymbol{y}) = t^2 \boldsymbol{f}(\boldsymbol{x}) \tag{11}$$

The solution for Equation (11) is the extreme point of the following equation, as follows:

$$g(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{y})^{\mathrm{T}} \boldsymbol{M}(\boldsymbol{x} - \boldsymbol{y}) + t^2 E(\boldsymbol{x}) \tag{12}$$

where $E(\boldsymbol{x}) : R^{3m} \to R.$ is the potential function and $\boldsymbol{f} = -\nabla E$. Therefore, the problem is transferred to solve the minimal value of $g(\boldsymbol{x})$. On the basis of Hooke's Law, the elastic potential energy can be defined as follows:

$$\frac{1}{2}k(\,\|\boldsymbol{p}_1 - \boldsymbol{p}_2\| - r)^2 \tag{13}$$

here, the spring's initial length is r, positions of its two ends are $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, and its stiffness coefficient is k. Through the transformation of triangle inequality and including external forces into the energy equation, the energy can be ultimately illustrated as follows:

$$\begin{aligned} E(\boldsymbol{x}) = \min_{d \in U} \tfrac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{L}\boldsymbol{x} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{J}\boldsymbol{d} + \\ \tfrac{1}{2}\sum_{i=1}^{s} k_i \boldsymbol{d}_i^2 + \boldsymbol{x}^{\mathrm{T}}\boldsymbol{f}_{\mathrm{ext}} \end{aligned} \tag{14}$$

in which $\boldsymbol{U}$ is the set of spring directions at rest state, s is the total number of springs, and $\boldsymbol{f}_{ext}$ stands for external force. Furthermore, $\boldsymbol{d}$, $\boldsymbol{L} \in R^{3m\times 3m}$, and $\boldsymbol{J} \in R^{3m\times 3m}$ are defined as follows:

$$\begin{cases} \boldsymbol{d} = r((\boldsymbol{p}_1 - \boldsymbol{p}_2)/\|\boldsymbol{p}_1 - \boldsymbol{p}_2\|) \\ \boldsymbol{L} = \left(\sum_{i=1}^{s} k_i \boldsymbol{A}_i \boldsymbol{A}_i^{\mathrm{T}}\right) \otimes \boldsymbol{I}_3 \\ \boldsymbol{J} = \left(\sum_{i=1}^{s} k_i \boldsymbol{A}_i \boldsymbol{S}_i^{\mathrm{T}}\right) \otimes \boldsymbol{I}_3 \end{cases} \tag{15}$$

where $\boldsymbol{A}_i \in R^m$ is the incidence vector of the ith spring, and $\boldsymbol{S}_i \in R^s$ is the ith spring indicator. The matrix $\boldsymbol{I}_3 \in R^{3\times 3}$ is the identity matrix and $\otimes$ denotes the Kronecker product. By inserting the energy equation into Equation (12), the optimization problem is transformed as follows:

$$\begin{aligned} \min_{x \in R^m,\, d \in U} \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}(\boldsymbol{M} + t^2\boldsymbol{L})\boldsymbol{x} - t^2\boldsymbol{x}^{\mathrm{T}}\boldsymbol{J}\boldsymbol{d} + \\ \boldsymbol{x}^{\mathrm{T}}(t^2\boldsymbol{f}_{\mathrm{ext}} - \boldsymbol{M}\boldsymbol{y}) \end{aligned} \tag{16}$$

Solving the equation above by derivation, we have:

$$(\boldsymbol{M} + t^2\boldsymbol{L})\boldsymbol{x} = t^2\boldsymbol{J}d + \boldsymbol{M}\boldsymbol{y} - t^2\boldsymbol{f}_{\mathrm{ext}} \tag{17}$$

In Equation (17), the left part $\boldsymbol{M} + t^2\boldsymbol{L}$ keeps invariant, meaning that we can calculate this part during the preprocessing procedure without loop computation in the simulation process. For the right part of the equation, $\boldsymbol{M}$, t and $\boldsymbol{L}$ are all constants. Hence during the calculation of each time step, we only need to update $\boldsymbol{y}$, $\boldsymbol{d}$, and $\boldsymbol{f}$.

During the process of solving $\boldsymbol{x}$, since the coefficient matrix $\boldsymbol{M} + t^2\boldsymbol{L}$ is symmetric positive definite and also invariant, the Cholesky decomposition method is able to be applied to this sparse matrix to solve the equation.

4 Our method

In this section, we present a fast MSM solver for high-resolution 3D elastic objects. By projecting the complex surface geometry model into a set of uniform grid cells as cages using the *Cages MVC method, our method enables the surface geometry model to have internal structure and mechanics properties, thus acquiring physical fidelity. What is more, for the purpose of real-time simulation on high-resolution models, the original Cholesky decomposition method in the fast MSM is replaced by the conjugate gradient method,[7] and further it is accelerated by a GPU.

4.1 Surface geometry model projection

The fast MSM can solve the surface geometry model like cloth efficiently,[7] but one layer of surface mesh is not enough to illustrate the complicated internal structure of 3D elastic objects. To obtain deformation simulation results within physical fidelity, we project the complex surface geometry model into a set of uniform grid cells as cages through the *Cages MVC method to generate its internal structure and mechanics properties. The created

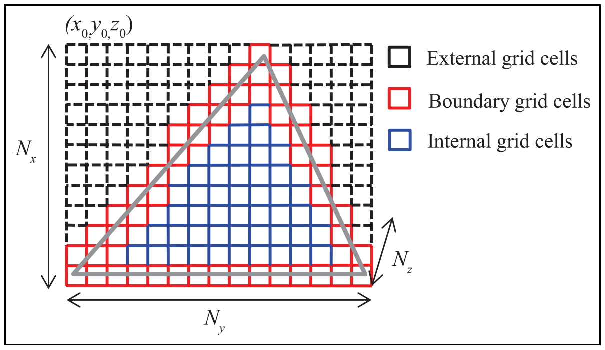

Figure 2. External, boundary, and internal grid cells.

uniform grid cells are used as cages which take responsibility for the simulation calculation, whereas the elastic object's surface mesh model is in charge of the simulation render and display. In such way, the elastic object can hold its geometry characteristics and its physical features of internal structure and mechanics properties as well.

Next, we will explain how to make the boundary and internal area of the surface geometry mesh model be covered by grid cells. First we obtain the surface mesh model's circumscribed cuboid within the range of $N_x * N_y * N_z$ which is calculated by the maximum and minimum coordinates of the surface mesh model. Next we utilize the cuboid to set a number of cubic grid cells with side length of L_0 to cover both the exterior side and interior side of the whole mesh model. Each grid cell consists of eight vertices and 12 edges and its index of the circumscribed cuboid is expressed as (p, q, r) at three axes (x, y, z) respectively. Suppose the start coordinate of the circumscribed cuboid is (x_0, y_0, z_0), therefore we can obtain the start coordinate of a certain grid inside the cuboid by $(x_0 + p * L_0, y_0 + q * L_0, z_0 + r * L_0)$.

As shown in Figure 2, we divide the cuboid into three kinds of grid cells, the black for external, the red for boundary, and the blue for internal ones, according to their relations with the surface mesh model. Among them, the external grid cell has no direct linkage to the elastic object structure, meaning we will ignore this part of the grid cells in simulation computation. Nevertheless, the boundary grid cell is a kind of grid area that only covers the entire boundary of the mesh model and the internal grid cell is the one that fills the inside of the mesh model. Both the boundary grid cells and the internal grid cells are called valid grid cells and we make use of these valid grid cells to solve the calculation for the mass spring problem.

The most vital step in circumscribed cuboid division is to make sure that all boundary grid cells can cover both the vertices and triangle facets of the surface model. For example, if there is a certain vertex v_1 in a triangle facet with coordinate of (x_1, y_1, z_1), what we can learn from the grid cell generation step above is that the index of the

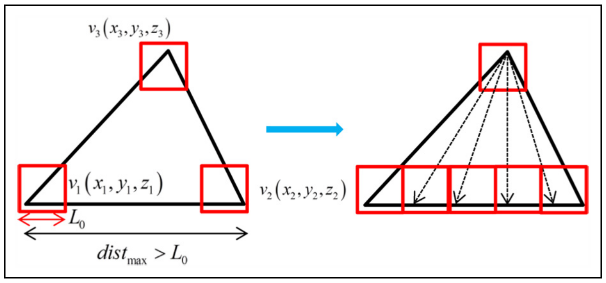

Figure 3. The ample points generation in a triangle facet.

vertex's covering cell is $(\lfloor (x_1 - x_0)/L_0, (y_1 - y_0)/L_0, (z_1 - z_0)/L_0 \rfloor)$. Similarly, we can get the other two cells' indices of vertices on the triangle facet and mark these three grid cells as boundary cells for the triangle facet. However, if the side length of the triangle facet exceeds L_0, the boundary cells containing three vertices maybe unable to cover the whole facet, as we can see in Figure 3. To deal with such problem, we have to produce some sample points on the triangle facet to generate more grid cells which can cover the entire facet.

Suppose that coordinates of the three vertices on a certain triangle facet are $v_1(x_1, y_1, z_1)$, $v_2(x_2, y_2, z_2)$, and $v_3(x_3, y_3, z_3)$, and its three side lengths are $dist_{1,2}$, $dist_{2,3}$, and $dist_{1,3}$ in which the maximum one is set as $dist_{\max}$ (here we assume the maximum side is $v_{1,2}$). When $dist_{\max}$ is longer than L_0, we set $n = \lceil dist_{\max}/L_0 \rceil$ as the partition sizes. The coordinates of the sample points we generate on the facet are $v_3 + n\left(\frac{i}{n} * v_{1,3} + \frac{j}{n} * v_{2,3}\right)$ where $i, j \in (0, n)$, and the rest can be done in the same way. In this way, we produce new sample points on the triangle facet to obtain needed grid cells to cover the whole facet.

After we have gained all valid grid cells, we then have the physical model for the mass spring problem. Here we regard each grid cell vertex as mass, and each grid vertex has at most 26 neighboring springs. In our method, all these valid grid cells are actually cages and we apply the *Cages MVC method to calculate the *Cages MVC of each vertex on the surface mesh model, which makes it possible to map cages back to the surface model after simulation solution. As described in Figure 4, p is a vertex of surface mesh model and $v_i (i = 1, 2, \ldots, 8)$ is the vertex of the cage. Since the tetrahedron is a processing unit in the MVC calculation, we regard a grid cell as a dodecahedron, and divide its six square facets into 12 triangle facets while calculating the MVC.

4.2 Conjugate gradient method for fast MSM

In order to retain detailed information of the surface mesh model, it is often required to set the cage's length as a rather small value during simulation processing. However, such action may result in an excessive number of masses

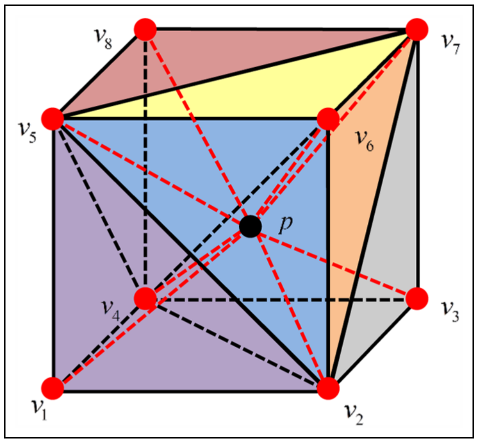

Figure 4. Vertex of the surface mesh model and its grid cell (cage).

and springs generated in the physical model. Moreover, the size of the coefficient matrix will easily reach more than 10^4, causing it to be impossible to solve the mass spring problem in real time.

Another point to note is that although in matrix $\boldsymbol{M} + t^2\boldsymbol{L}$, $\boldsymbol{M}$ is a diagonal matrix, but in fact $\boldsymbol{L}$ is not a band matrix. Therefore, if we use the Cholesky decomposition method to decompose the coefficient matrix, there can be no assurance that the decomposed matrix is sparse. It brings no apparent advantage to solving the equation on that basis. Furthermore, the size of the decomposed matrix is likely to exceed the memory allocated and lead to overflow.

Focusing on the problem above, we employ the conjugate gradient method to take the place of Cholesky decomposition to solve the equation. Because $\boldsymbol{M} + t^2\boldsymbol{L}$ is a symmetrical positive definite matrix, it is perfectly suitable to use the conjugate gradient method to decompose it. The conjugate gradient method will not destroy the sparsity of the matrices generated in and after the decomposition process. We use the conjugate gradient method to increase the scale that fast MSM can solve and get the advantage of a greater solution velocity to realize real-time simulation for large scale scenes.

4.3 GPU-accelerated conjugate gradient method

In the solution process of conjugate gradient method, basic operations are only between the matrix and vector, vector and vector, and vector and real number. Each element in these matrices and vectors is independent and unrelated with each other. Therefore, it allows us to apply GPU parallel acceleration to increase the solution rate,[24,25] and the algorithm is demonstrated in Algorithm 1.

While calculating on the GPU, the mainly involved operands are the matrix and vector.[24] For vectors, we can use continuous arrays to store them directly. However, for matrices which are all sparse, it will cause a large waste of GPU memory to store them in two-dimensional arrays and increase the operation complexity. Hence, we use another storage form to read and write matrices.

Algorithm 1 GPU acceleration based on the conjugate gradient algorithm.

```
1: For equation $(\boldsymbol{M}+t^2\boldsymbol{L})\boldsymbol{x}=t^2\boldsymbol{J}d+\boldsymbol{M}\boldsymbol{y}-t^2\boldsymbol{f}_{\text{ext}}$, set
   $\boldsymbol{A}=(\boldsymbol{M}+t^2\boldsymbol{L})$, $\boldsymbol{b}=t^2\boldsymbol{J}d+\boldsymbol{M}\boldsymbol{y}-t^2\boldsymbol{f}_{\text{ext}}$.
2: Provide the initial parameters $\boldsymbol{x}=\boldsymbol{b}$, $\boldsymbol{r}=\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}$, and
   initialize constants $\rho_1=\|\boldsymbol{r}\|^2$, $k=1$, permissible error $\varepsilon$ and
   maximum iteration times $k_{\max}$.
3: while $(\sqrt{\rho_1}>\varepsilon\|\boldsymbol{b}\|^2)$ and $(k<k_{\max})$ do
4:   if $k==1$ then
5:     $\boldsymbol{p}=\boldsymbol{r}$
6:   else
7:     $\beta=\rho_1/\rho_2$
8:     $\boldsymbol{p}=\boldsymbol{r}+\beta\boldsymbol{p}$
9:   end if
10:    $\boldsymbol{w}=\boldsymbol{A}\boldsymbol{p}$
11:    $\alpha=\rho_1/\boldsymbol{p}^{\mathrm{T}}\boldsymbol{w}$
12:    $\boldsymbol{x}=\boldsymbol{x}+\alpha\boldsymbol{p}$
13:    $\boldsymbol{r}=\boldsymbol{r}-\alpha\boldsymbol{w}$
14:    $\rho_2=\rho_1$
15:    $\rho_1=\|\boldsymbol{r}\|_2^2$
16:    $k=k+1$
17:  end while
18: return $\boldsymbol{x}$
```

Figure 5 shows how we construct the data structure to store sparse matrices, where three arrays, *Row*, *Col* and *Value*, are used. *Row* stores the index of a certain element in *Value* which is also the first element in a row of the matrix, while the last element of *Row* contains the number of nonzero elements in the sparse matrix. Each sparse matrix's nonzero element's column numbers are stored in Col and their values in *Value*. Here, *Col* and *Value* both use strategy of row major order to save data, meaning that we cannot write the current row's elements into arrays, unless all nonzero elements of the last row have been already saved in the array.

The operation to matrix is multiplication between the vector and matrix during computation. After we have loaded the vector and matrix data into the GPU memory respectively, set the number of operating threads as the number of rows in sparse matrix. A certain element value in *Row* can obtain the element's index range in *Col* and *Value*. By obtaining the data in *Value* and the column number in *Col*, we can finish the multiplication operation between the row elements and the vector. Once the row vectors of the matrix have been successfully traversed, the whole multiplication is done and we can get a result for the vector.

5 Experimental results

The environment for the experiments is Xeon E3-1230 V2 3.30 373 GHz quad-core processor, 4 GB internal storage,

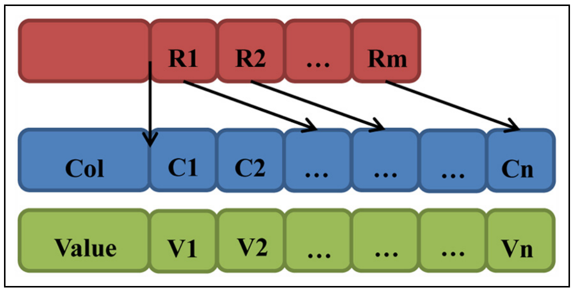

Figure 5. Storage form of sparse matrix in the GPU.

Nvidia GeForce 374 GTX 650 Ti graphics card, VS 2012, OpenGL, CUDA 6.5.

In the Figure 6, we present a comparison of the frame rate between our method and the fast MSM. Our method adopts the conjugate gradient method to solve Equation 17 and realize GPU acceleration whereas a Cholesky decomposition is utilized in the fast MSM. The sample model used is a Bunny model with 20,667 vertices and 41,330 triangle facets. We adjust the unit length of the grid cell to get different numbers of valid grid cells, and the number of valid vertices changes according to valid grid cells'. In the figure, the blue stands for the GPU accelerated conjugate gradient method, the red for the conjugate gradient method without GPU acceleration and the green for the fast MSM. The figure illustrates that the conjugate gradient method ensures a higher speed of solution compared to the Cholesky decomposition. Moreover, when accelerated by the GPU, the method performs significantly better. Even if we increase the number of valid grid cells, the advantage is still apparent. Therefore, for a higher-resolution model, our method achieves a better performance than the fast MSM.

It is worth noting that in the paper we do not compare our method with other most commonly used physics methods like the finite element method (FEM). The reason is that the FEM adopts tetrahedron elements as the processing unit but our method and the MSM use springs, therefore it is unable to make the two methods' simulation environment all the same. Consequently, here we show the comparison results of our method and the fast MSM, which indicates that our improvements based on fast MSM can achieve better and apparent efficiency.

To prove the effectiveness of our method, we have adopted six different models to perform the experiments. And the experiments are fulfilled under different force conditions. Firstly, we project the elastic object's surface geometry model into uniform grid cells by the method mentioned in section 4 and then set up mapping relation between mesh model and grid cells using *Cages MVC. After that, exert corresponding forces to the model and solve the dynamic solutions via method in Section 4 and the final results are obtained. The acting forces include dragging, pushing, pressing, and free falling.

As shown in Figure 7, we demonstrate the deformation effect under the impact of a dragging force. During the experiment, we applied three types of dragging force onto the model. Firstly, the head of the Armadillo model is fixed which means that the grid cells of this part cannot move while under dragging. Then we drag the left leg of the model by applying a force of a constant value. Under the effect of a dragging force, the grid cells deform and therefore the surface model also deforms due to the mapping relation. The result shows that under a constant dragging force, the left leg moves to the left to reach an equilibrium state, and its deformation also leads to the right leg's deformation. Besides, we make the Armadillo's leg still and exert force on its left palm and forehead respectively. From Figure 7 we can learn that the dragging force based deformation simulation of our method is natural.

The scene shown in Figure 8 describes the deformable simulation of the Dragon model, involving a raising and

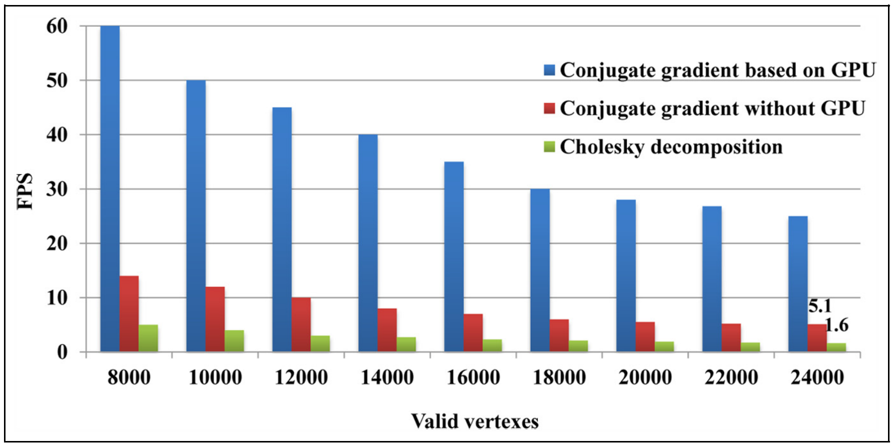

Figure 6. Comparison of frame rate between our method and the fast MSM.

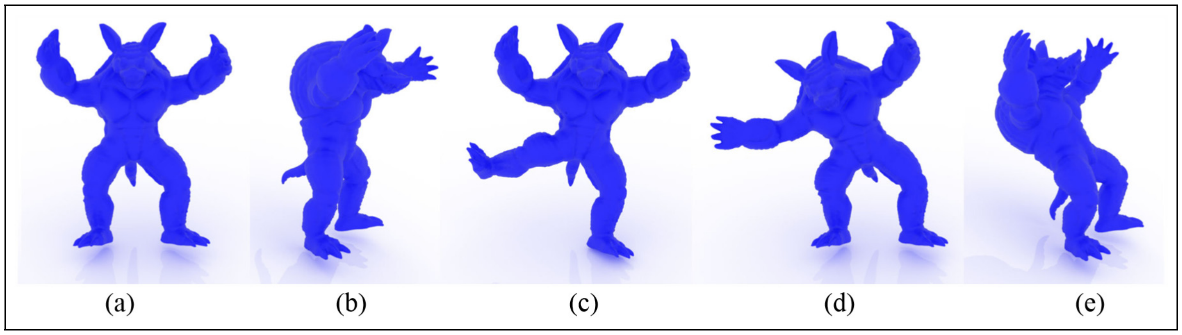

Figure 7. Deformation of Armadillo model under three types of dragging force. From left to right are: (a) initial front view, (b) initial left view, (c) head fixed and leg dragged, (d) leg fixed and palm dragged, and (e) leg fixed and forehead dragged.

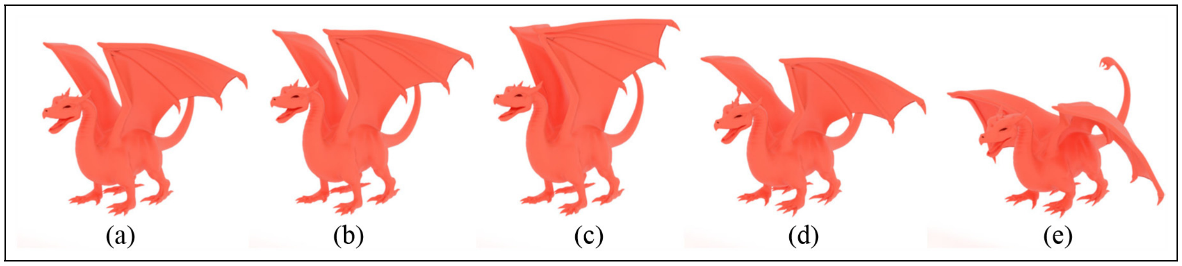

Figure 8. Deformation of Dragon model under raising and pushing force. From left to right are: (a) initial view, (b) leg fixed and raise the head and wings, (c) raising deformation reaches a stationary state, (d) leg fixed and push the head and wings, and (e) pushing deformation reaches a stationary state.

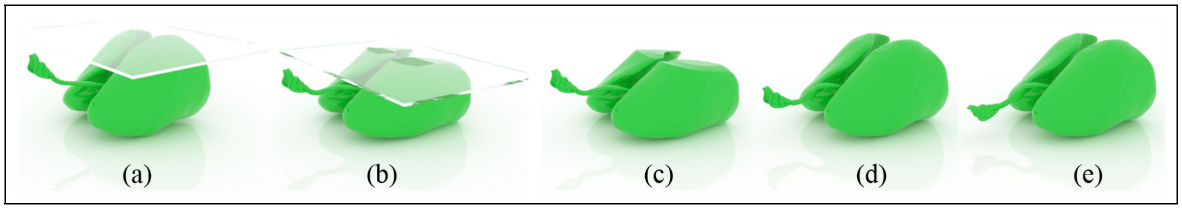

Figure 9. Deformation of Lung model under pressing force by a glass plane. From left to right are: (a) initial view, (b) place the model on the ground press the lung by a glass plane, (c) remove the glass plane, (d) deformable part restores to the original state, and (e) the trachea part sags.

pushing force. Fix the leg of the Dragon model at a certain point and then apply an upward force to the model to raise its head and wings. When the model deformation reaches an equilibrium state, we then reverse the force to make the deformable part downward as shown in the figure. When the model's wings and head move downward, its knees also bend as a result.

The two experiments above test the deformation effect under a constant force. However, it is quite important to ensure that the deformable model could return to its original state after releasing the force. Figure 9 demonstrates that our method can achieve such goal. We place a glass plane onto the Lung model and therefore its top part and trachea begin to move down correspondingly. After a while, the plane is removed so that the main body of the Lung model returns to its original state and obtains a velocity upward because of accumulating certain elastic potential energy. Whereas the trachea part keeps its sagging position.

In addition to the pressing force, a pulling force is employed to inspect the model's restoration capability. We fix vessels of the Heart model and pull its chamber to a certain position. As seen in Figure 10, the chamber deforms back to its original position after we release the force. The above two experiments testify the practicability of our method further.

Free falling simulation is a significant part of elastic object simulation. We use the Bunny and Brain model to perform the experiments and solve the problem of collision detection. In a fast simulation of the MSM, there is no

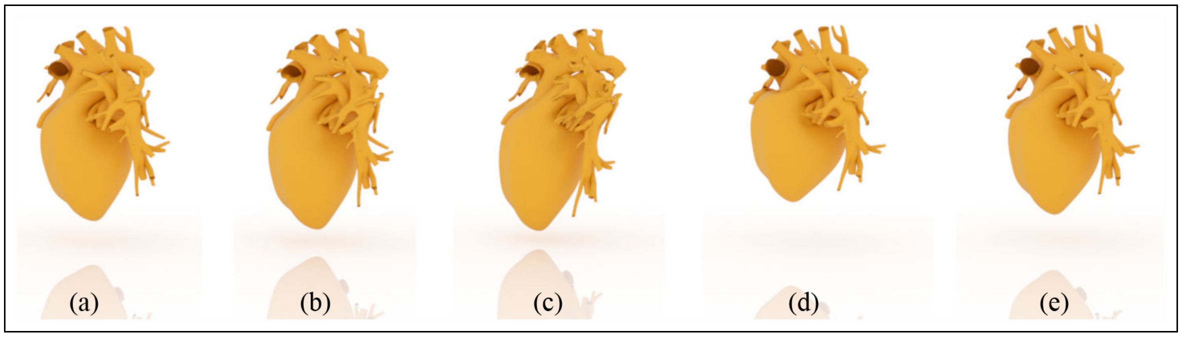

Figure 10. Deformation of Heart model under a pulling force. From left to right are: (a) initial view, (b) vessels fixed and pull the heart chamber, (c) release the force, (d) and (e) deformable part restores to the original state.

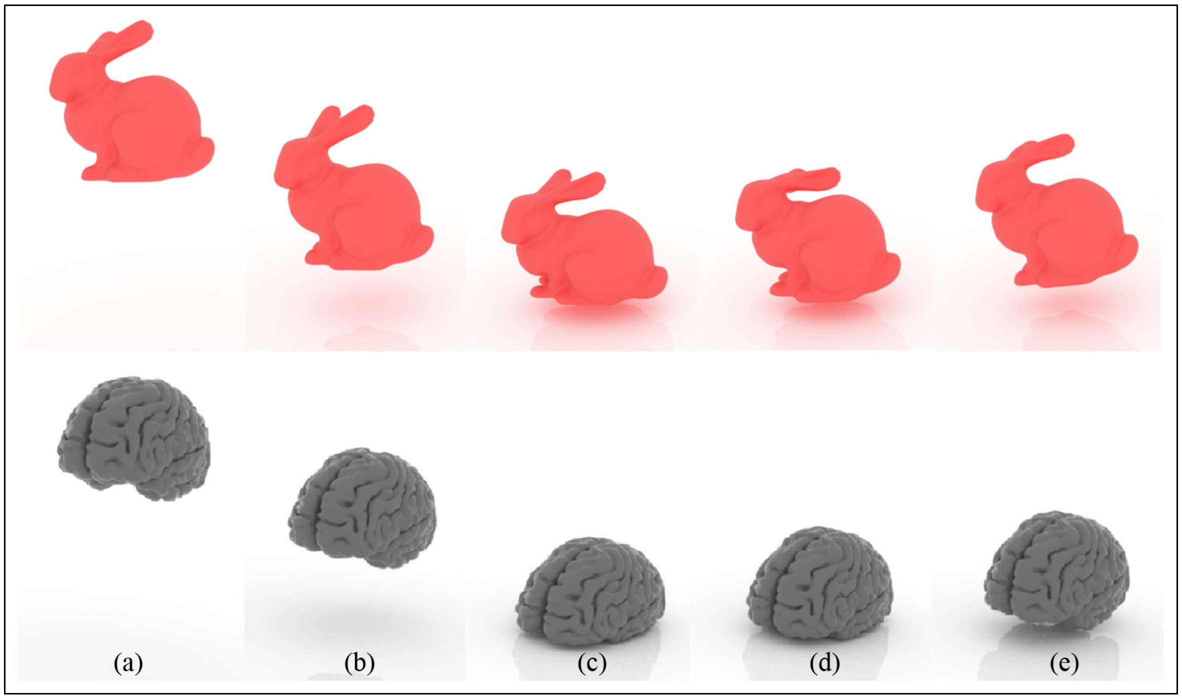

Figure 11. Deformation of Bunny and Brain models under free falling. From left to right are: (a) initial view, (b) models falls down, (c) models hit the ground, (d) models reverse speeds, and (e) models leap upward.

single variable denoting the velocity. So we use the location difference between this moment and the last moment to calculate the displacement and velocity.

After the objects collide with the ground their speeds are invariable but the direction should then be the opposite. Suppose that the location of the current moment is q_n, and the current velocity is $\boldsymbol{v}_n = \boldsymbol{q}_n - \boldsymbol{q}_{n-1}$. If the collision happens, we set $\boldsymbol{v}_{n+1} = -\boldsymbol{v}_n$ and $\boldsymbol{q}_{n+1} = \boldsymbol{q}_{n-1}$ to reverse the velocity. In Figure 11, we can find that even for the model with a complicated surface structure, the deformation effect is still natural and real through the method in this paper. For the Bunny, because its ears and the front part of the body are suspended in midair, these two parts do not stay completely static after the collision happens, but they move down a little. Ultimately, the two models pop up from the ground because of the reversed velocity.

Table 1 lists model parameters and execution times for the above experiments. It can be inferred from the table that the number of valid grid cells and springs will affect the running time. The Lung model and Heart model have a similar vertex number but pretty different valid grid cell numbers, thus leading to different frame rates. Yet in practice, we can control the number of valid grid cells and springs by adjusting the unit length of the grid cell to satisfy various needs. The experimental results illustrate that without GPU parallel acceleration, the model we present

Table 1. Execution time of deformation simulation on the CPU and GPU.

Models	Vertices	Triangle facets	Valid grid cells	Springs	CPU (FPS)	GPU (FPS)
Armadillo	12,602	25,200	8338	164,343	8.1	30.0
Dragon	23,620	47,140	11,536	232,240	5.5	23.9
Lung	43,040	86,085	6499	127,843	10.1	29.5
Heart	43,594	108,357	3126	62,465	19.9	53.4
Bunny	20,667	41,330	9946	193,589	7.3	30.1
Brain	55,024	110,047	13,661	264,481	5.1	20.0

cannot satisfy the need of real-time simulation perfectly for rather large scale scenes. But if we accelerate the solving on a GPU, the calculation efficiency is increased significantly so that we are able to assure the real-time performance for high-resolution elastic objects. In the supplementary video we provided, if the size of the model is not extremely large, the simulation frame rates of the model will exceed 24 because of our improvements and GPU acceleration. Such frame rates assure viewers' real time experience.

6 Discussions

In the experiments above, we have achieved efficient simulation of a high-resolution elastic object with visual reality and physical fidelity. While during experiments, we have found two limitations for our method.

One limitation is that the relationship between the force and displacement is linear, due to Hooke's Law of springs. However, in Figure 9, if we press the Lung model with a glass plane to a extremely low position, its deformation effects will begin to distort and the model is not as stable as before. Therefore, it is better to improve the linear equation for force and displacement when a model is under a rather large force.

The other limitation is that during the deformation process, our method lacks self-collision detection, which is essential in complex deformation simulation scenes. In Figure 11 for example, if we make the Bunny model fall down from a rather high position, its ears will penetrate its body because of the inertia force. Therefore, it is crucial to add self-collision detection into our method to avoid such situation.

7 Conclusions and future work

In this paper, we propose a fast MSM solver for high-resolution 3D elastic objects. To retain the surface geometry model's internal structure and mechanics properties and acquire physical fidelity, we project the complex surface geometry model into a set of uniform grids as cages which are used in *Cages MVC to construct the mapping relation between the geometry model and cages model, which can reflect its internal structure and mechanics properties. To efficiently solve the high-resolution surface geometry model, we replace the Cholesky decomposition method in the fast MSM solver with the conjugate gradient method, which can make the fast MSM solver more efficient for detailed surface geometry models. Finally, we propose a GPU-accelerated parallel algorithm for the conjugate gradient method. The experimental results demonstrate that our method can realize efficient deformation simulation of 3D elastic objects with visual reality and physical fidelity, which has a great potential for applications in computer animation.

In future work, we will focus on improving the linear equation for model displacement and applied force, such as adding a non-linear displacement calculation equation when the model is under an extremely large force. Moreover, currently the proposed method is a proof-of-concept only at the experimental stage, hence, it is not yet of practical use due to certain limitations. For example, we should introduce collision detection into our method to achieve robust deformation effects with high physical fidelity and support more complex interactions far beyond the simple deformation simulation. And we should also incorporate certain mature techniques to accommodate self-collision caused by elastic deformation. Therefore, our ongoing research efforts are concentrated on seeking an efficient optimization method to guarantee the physical accuracy in an absolute sense.

Funding

This work was supported by the the National Natural Science Foundation of China (grant number 61373107) and the Science and Technology Program of Wuhan, China (grant number 2016010101010022).

Supplemental material

A supplementary video for this article is available on the journal's website.

References

1. Liao X, Yuan Z, Hu P, et al. GPU-assisted energy asynchronous diffusion parallel computing model for soft tissue deformation simulation. *Simulation* 2014; 90: 1199–1208.
2. Johnsen S, Taylor Z, Han L, et al. Detection and modelling of contacts in explicit finite-element simulation of soft tissue biomechanics. *Int J Comput Assist Radiol Surg* 2015; 10: 1873–1891.
3. Duan Y, Huang W, Chang H, et al. Volume preserved mass–spring model with novel constraints for soft tissue deformation. *IEEE J Biomed Health Inform* 2016; 20: 268–280.
4. García FG, Paradinas T, Coll N, et al. * cages: A multilevel, multi-cage-based system for mesh deformation. *ACM Trans Graph* 2013; 32: 1–24.
5. Bonet J and Wood RD. *Nonlinear continuum mechanics for finite element analysis*. Cambridge: Cambridge University Press, 1998, 38(5): 106.
6. Provot X. Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In: *Graphics interface*. Mississauga, Canada: Canadian Information Processing Society, 2001, 23(19): 147–154.
7. Liu T, Bargteil AW, O'Brien JF, et al. Fast simulation of mass-spring systems. *ACM Trans Graph* 2013; 32: 1–214.
8. Gao L, Lai Y, Huang Q, et al. A data-driven approach to realistic shape morphing. *Comput Graph Forum*, 2013; 32(2): 449–457.
9. Chen D, Levin DI, Sueda S, et al. Data-driven finite elements for geometry and material design. *ACM Trans Graph* 2015; 34: 1–74.
10. Frouard J, Quillen AC, Efroimsky M, et al. Numerical simulation of tidal evolution of a viscoelastic body modelled with a mass-spring network. *Mon Not R Astron Soc* 2016; 458: 2890–2901.
11. Floater MS. Mean value coordinates. *Comput Aided Geom Des* 2003; 20: 19–27.
12. Floater MS, Kós G and Reimers M. Mean value coordinates in 3D. *Comput Aided Geom Des* 2005; 22: 623–631.
13. Ju T, Schaefer S and Warren J. Mean value coordinates for closed triangular meshes. In: *ACM transactions on graphics (TOG)*. New York: ACM, pp.561–566.
14. Joshi P, Meyer M, DeRose T, et al. Harmonic coordinates for character articulation. In: *ACM transactions on graphics (TOG)*. New York: ACM, pp.71.
15. Lipman Y, Levin D and Cohen-Or D. Green coordinates. In *ACM transactions on graphics (TOG)*. New York: ACM, pp.78.
16. Miller K, Joldes G, Lance D, et al. Total lagrangian explicit dynamics finite element algorithm for computing soft tissue deformation. *Commun Numer Methods Eng* 2007; 23: 121–134.
17. Gilles B, Bousquet G, Faure F, et al. Frame-based elastic models. *ACM Trans Graph* 2011; 30: 1–15.
18. Irving G, Teran J and Fedkiw R. Invertible finite elements for robust simulation of large deformation. In: *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on computer animation*, pp.131–140. Geneve, Switzerland: Eurographics Association.
19. Nesme M, Kry PG, Jeřábková L, et al. Preserving topology and elasticity for embedded deformable models. In: *ACM transactions on graphics (TOG)*. New York: ACM, pp.52.
20. Bargteil AW and Cohen E. Animation of deformable bodies with quadratic bézier finite elements. *ACM Trans Graph* 2014; 33: 1–27.
21. Baraff D and Witkin A. Large steps in cloth simulation. In: *Proceedings of the 25th annual conference on computer graphics and interactive techniques*, pp.43–54. New York: ACM.
22. Martin S, Thomaszewski B, Grinspun E, et al. Example-based elastic materials. In: *ACM transactions on graphics (TOG)*. New York: ACM, pp.72.
23. San-Vicente G, Aguinaga I and Celigueta JT. Cubical mass-spring model design based on a tensile deformation test and nonlinear material model. *IEEE Trans Vis Comput Graph* 2012; 18(2): 228–241.
24. Zhou Y, He F and Qiu Y. Optimization of parallel iterated local search algorithms on graphics processing unit. *J Supercomput* 2016; 72(6): 2394–2416.
25. Zhou Y, He F and Qiu Y. Dynamic strategy based parallel ant colony optimization on GPUs for TSPs. *Sci China Inform Sci* 2017; DOI: 10.1007/s11432-015-0594-2.

Author biographies

Mianlun Zheng is a postgraduate student at the School of Computer at Wuhan University. Her research interests include computer simulation, virtual reality and computer graphics.

Zhiyong Yuan is a professor at the School of Computer of Wuhan University. He gained a PhD degree in control science and engineering from Huazhong University of Science and Technology. He received his BS degree in computer applications and an MS degree in signal and information processing from Wuhan University. His research interests include computer simulation, virtual reality and intelligent medical systems.

Weixu Zhu is a master student at the School of Computer at Wuhan University. His research interests include virtual reality, human-computer interaction and artificial intelligence.

Guian Zhang is a PhD student at the School of Computer at Wuhan University. His research interests include computer vision and pattern recognition.