Clustering Method for Fast Deformation with Constraints

Jin Huang*
State Key Lab of CAD&CG
Zhejiang University

Xinguo Liu
Microsoft Research Asia

Hujun Bao
State Key Lab of CAD&CG
Zhejiang Universit

Baining Guo
Microsoft Research Asia

Heung-Yeung Shum
Microsoft Research Asia

Abstract

We present a fast deformation method for flexible objects. The deformation of the object is physically modeled using a linear elasticity model with a displacement based finite elements method, yielding a linear system at each time step of simulation. We solve this linear system using a precomputed force-displacement matrix, which describes the object response in terms of displacement accelerations to the forces acting on each vertex. We exploit the spatial coherence to effectively compress the force-displacement matrix to make this method practical and efficient by applying the clustered principal component analysis method. And we developed a method to efficiently handle the additional constraints for interactive user manipulation. At last large deformations are addressed based upon the compressed force-displacement matrix by combining a domain decomposition method and tracking the rotational motions. The experimental results demonstrate fast performances on complex large scale objects under interactive user manipulations.

CR Categories: I.3.5 [COMPUTER GRAPHIC]: Computational Geometry and Object Modeling —Physically based modeling; I.3.7 [COMPUTER GRAPHIC]: Three-Dimensional Graphics and Realism—Animation

Keywords: Deformation, Finite Element Method, Clustered Principal Component Analysis, Domain Decomposition Method

1 Introduction

Deforming and animating soft objects have a wide range of applications in geometric modeling, computer animation, video games and surgery simulation, since many real world objects are soft and deformable. Physically based modeling has become an important approach to graphics modeling and animation. One of the simplest physically based models is the mass-spring system [Chadwick et al. 1989; Tu and Terzopoulos 1994], which has been successfully used to simulate clothes [Baraff and Witkin 1998; Choi and Ko 2002; Bridson et al. 2003]. Recent research work showed a trend away from the simple mass-spring systems toward the more sophisticated finite element method (FEM) [Cook et al. 1989], since it is physically more accurate. Using FEM, an object's deformation behavior can be easily specified by a few material properties (which have physical meaning) instead of by adjusting a large number of spring constants as in a mass-spring system.

*e-mail: hj@cad.zju.edu.cn

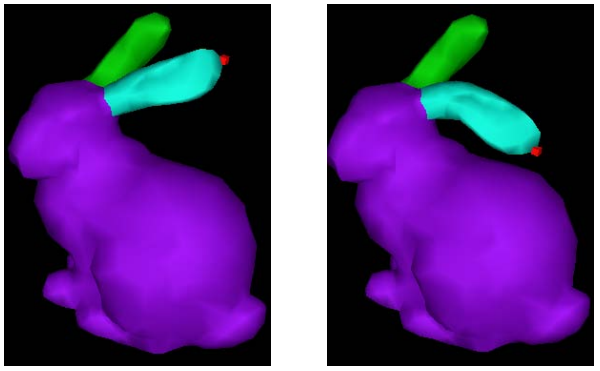


Figure 1: Simulation result using our clustering method with DDM. Model before deformation (left). Model after deformation by dragging the point in red (right), and the model is colored according to DDM region.

In FEM based deformation approaches, there are two models to measure the strain with respect to deformation in terms of displacement. One is the linear elasticity model that approximates the elastic forces as the product of a constant stiffness matrix and the displacement vector, yielding a numerically fast and stable simulation system. However, it can only model small deformations accurately. The other is the non-linear elasticity model that models large deformations accurately with the cost of reevaluating the stiffness matrix at every time step, yielding a slow simulation and introducing numerical instabilities. By partitioning complex non-rigid behavior into global rotational motion and local deformations, a fast and stable simulation for large deformations can be obtained using the linear elasticity model [Müller et al. 2002; Müller and Gross 2004; Capell et al. 2002a; James and Pai 2002b].

Finite element methods divide the object undergoing deformations into a set of elements, and approximate the continuous dynamic equations over the elements. It is easy to get thousands of elements for objects of modest scale in graphics applications (e.g. the Stanford Bunny model). Therefore the straightforward FEMs are not fast enough for interactive simulation tasks commonly required in graphics applications.

In this paper, we present an efficient acceleration method for simulating dynamic deformations of flexible objects. We take the linear elasticity model and the finite element method for dynamic deformation modeling, and solve the deformation using the implicit Euler method. The efficiency is achieved by precomputing a force-displacement matrix in a preprocessing step, such that the next frame deformation could be obtained by a matrix and vector multiplication.

The force-displacement matrix is usually dense and huge, so the computational cost is still too large to achieve fast simulation for a large scale object. To deal this problem, we compress the matrix using a clustering method, i.e. the clustered principal compo-

nent analysis (CPCA) method presented in [Sloan et al. 2003]. This compression step not only dramatically reduces the storage of the force-displacement matrix, but also accelerates the multiplication, thus accelerating the simulation in turn. Another advantage of our approach with the compressed force-displacement matrix is that it allows for dynamically introducing constraints for user manipulation.

In addition, we handle large rotational deformations by tracking a global rotation matrix and modeling the deformation in the object's reference coordinate frame. For complex objects with many soft and long components, we combine our method with the Domain Decomposition Method (DDM) [Quarteroni and Valli 1999] to simulate large deformations, as shown in Figure 1. And DDM improves the result compared with the linear blend described in [Hauser et al. 2003].

1.1 Related Work

In the last two decades, many ingenious physically based techniques for modeling deformable objects have been proposed, which can be found in geometrical modeling, surgical simulation and computer animations. We will only review some work related to the finite element method and the linear elasticity model for solid shapes. See [Gibson and Mirtich 1997] for a general survey.

Finite element solvers are computationally expensive. There are many approaches for quickly solving the discretized equations using adaptive methods to avoid wasting time on minor details. Some recently proposed representative work are [Capell et al. 2002b; Grinspun et al. 2002; Wu et al. 2001; Debunne et al. 2001]. Capell et al. proposed a multiresolution framework for dynamic simulation using volumetric subdivision [Capell et al. 2002b]. By adaptively refining the basis functions, Grinspun et al. proposed another simple framework for adaptive simulation [Grinspun et al. 2002], called CHARMS. Wu et al. [Wu et al. 2001] and Debunne et al. [Debunne et al. 2001] developed other kinds of adaptive methods using progressive meshes and LOD tetrahedron meshes. Debunne et al. also took adaptive time steps during simulation [Debunne et al. 2001] to further avoid unnecessary computation.

Modal analysis with finite element methods, which decompose non-rigid dynamics into a sum of independent vibration modes, has been a well established mathematical tool in mechanical engineering [Cook et al. 1989]. By discarding the small-amplitude, high-frequency modes, an efficient and stable simulation with visually acceptable accuracy can be obtained [Pentland and Williams 1989; Stam 1997; James and Pai 2002a; Hauser et al. 2003].

Our clustering method is basically equivalent to the modal analysis based method [Hauser et al. 2003] when there is only one cluster. We argue that we can achieve more accuracy with the same number of modes by dividing a dynamic deformation into several clusters.

Recently, James et al. [James and Pai 1999; James and Pai 2002b; James and Fatahalian 2003] proposed several data driven approaches to interactive dynamic simulation. These approaches basically precompute the Green's function, and model the deformation as a boundary value problem in terms of the precomputed Green functions. And it has been shown that the discrete Green functions of real world objects can be computed from some measured quantities [Pai et al. 2001].

Our approach also derives a deformation model using a precomputed matrix, called a force-displacement matrix. Our method models the interior of the solid objects as well as the boundary, which differentiates our work from those of James et al [James and Pai 1999; James and Pai 2002b; James and Fatahalian 2003]. And we compress the huge force-displacement matrix using the CPCA method, while James et al. [James and Fatahalian 2003] deal with it using wavelet decomposition based on a multiresolution representation of the object's boundary surface. The advantage of the CPCA based compression method is that it does not require any special (multiresolution) structure on the mesh of the object undergoing deformation.

The rest of this paper is organized as follows. After introducing the formulations for the displacement based FEM with the linear elasticity model in Section 2, we propose our force-displacement matrix based simulation method in Section 3. Then we address large deformations using DDM in Section 4. At last, we show some results and conclude this paper in the last two sections.

2 Formulation

Let $\Omega \subset R^3$ be a solid shape to be deformed, and $\mathbf{p}(\mathbf{x},t) : \Omega \times R \rightarrow R^3$ be the time dependent motion function of the shape. The motion function $\mathbf{p}(\mathbf{x},t)$ can be represented as the sum of the rest state and a displacement $\mathbf{q}(\mathbf{x},t)$:

$$\mathbf{p}(\mathbf{x},t) = \mathbf{x} + \mathbf{q}(\mathbf{x},t).$$

In the finite element method, the shape domain Ω is divided into elements of finite size. And the continuous displacement field in each element is interpolated using the displacement $\mathbf{q}_i(t)$ on the nodal points and a set of piecewise smooth basis functions $\{\phi^i(\mathbf{x}) : i = 1,2,\ldots,n\}$ on Ω.

$$\mathbf{q}(\mathbf{x},t) = \mathbf{q}_i(t))\phi^i(\mathbf{x}).$$

Let $\mathbf{q}(t) = [\mathbf{q}_1(t),\ldots,\mathbf{q}_n(t)]^T$ be the time dependent displacement vector. Then the Euler-Lagrange equation of dynamic deformations becomes:

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{D}\dot{\mathbf{q}} + \mathbf{K}\mathbf{q} = \mathbf{f}_{ext} + \mathbf{f}_{\psi}.$$

where $\mathbf{M},\mathbf{D},\mathbf{K}$ are the mass, damping, and stiffness matrices of the simulation system, $\mathbf{f}_{ext}$ is the external force added to the deformable object, and $\mathbf{f}_{\psi}$ is the constraint force added on the constrained node set ψ. In our system, the constrained nodes set ψ is dynamically determined during the simulation according to user manipulation. Since only relatively small local deformations are considered, the linear elasticity model is adopted in our approach, i.e. the stiffness matrix $\mathbf{K}$ is also a constant matrix. Therefore, $\mathbf{M}$, $\mathbf{D}$ and $\mathbf{K}$ are all constant matrices.

Let h be the time step for simulation, and $\Delta\dot{\mathbf{q}} = \dot{\mathbf{q}}(t+h) - \dot{\mathbf{q}}(t)$. We use the implicit Euler method to solve the above dynamic equation, yielding the following linear system:

$$(\mathbf{M} + h\mathbf{D} + h^2\mathbf{K})\Delta\dot{\mathbf{q}} = h(\mathbf{f}_{ext} - \mathbf{D}\dot{\mathbf{q}} - \mathbf{K}(\mathbf{q} + h\dot{\mathbf{q}}) + \mathbf{f}_{\psi}). \quad (1)$$

After solving the above linear system, the displacement vector at next the time stamp can be easily obtained by:

$$\mathbf{q}(t+h) = \mathbf{q}(t) + h(\dot{\mathbf{q}} + \Delta\dot{\mathbf{q}}).$$

For simplicity, we rewrite (1) as following:

$$\mathbf{A}\Delta\dot{\mathbf{q}} = \mathbf{f}. \quad (2)$$

where $\mathbf{A} = \mathbf{M} + h\mathbf{D} + h^2\mathbf{K}$, and $\mathbf{f} = h(\mathbf{f}_{ext} - \mathbf{D}\dot{q} - \mathbf{K}(\mathbf{q} + h\dot{\mathbf{q}}) + \mathbf{f}_{\psi})$. Note that $\mathbf{A}$ is also a constant matrix when we fix the time step h during simulation.

3 Force-Displacement Matrix

There are many methods for solving the linear system in (2). Since the simulation matrices **M**, **D**, **K** are usually very sparse and large, some iterative algorithms, such as conjugate gradient algorithm, have usually been used in previous work[Hauth and Etzmuss 2001]. However, those iterative algorithms would become extremely slow when a deformed object consists of up to thousands of finite elements, yielding a slow simulation. In this section, we will present a novel approach to efficiently solve it.

Our approach is based on the observation that if $\mathbf{A}^{-1}$ is available (for convenience, we denote $\mathbf{G} = \mathbf{A}^{-1}$ from now), the solution of (2) can also be obtained simply by matrix and vector multiplication:

$$\Delta\dot{\mathbf{q}} = \mathbf{A}^{-1}\mathbf{f} = \mathbf{G}\mathbf{f} \tag{3}$$

Note that each 3×3 block element $(g_{ij})_{3\times 3}$ of **G** gives the i-th vertex's response in terms of displacement acceleration to the force acting on the j-th vertex for a time period of h. Therefore, we call **G** the force-displacement matrix of the deformation system. In paper [Bro-Nielsen and Cotin 1996; J. Lang and Seidel 2003], same concept is used. [Bro-Nielsen and Cotin 1996] condenses the linear system by assuming that no forces are applied to internal nodes, then inverts it as the force-displacement matrix. While [J. Lang and Seidel 2003] uses discrete Green's functions matrix of BVP as the force-displacement matrix.

However, it is not practical to accomplish the simulation by straightforwardly applying (3), since the inverse matrix **G** is usually dense, which may require a huge amount of memory and flops doing the matrix-vector multiplication in (3) at each time step, yielding a even slower simulation for large scale models. Therefore it is necessary to first compress the force-displacement matrix. We note that there has been considerable efforts in directly approximating an inverse matrix using a sparse matrix [Benzi and Tuma 1999; Benzi et al. 2000; Bridson and Tang 2001]. However they are developed to obtain a better preconditioner for fast convergence.

Recall that the force-displacement matrix gives the object's response in terms of displacement acceleration to forces acting on a vertex. Nearby vertices may have a similar response to the same forces, i.e, spatial coherence among the row vectors of the force-displacement matrix exists. Based on this observation, we propose to compress the force-displacement matrix using clustered principal component analysis [Sloan et al. 2003]. Using the original CPCA method, the row vectors of **G** are grouped into several clusters. In each cluster, the conventional principal component analysis are applied to approximate the cluster. Many iteration steps are usually required until convergence is reached. The clustering results for the Bunny and Torusknot model are shown in Figure 2. The Bunny has 878 vertices and 3 clusters, and the Torusknot has 517 vertices and 6 clusters. We take 8 eigen vectors in each cluster for both models.

Therefore, the approximation to the force-displacement matrix takes the following form:

$$\mathbf{G} \approx \begin{pmatrix} \mathbf{U}_1\sigma_1\mathbf{V}_1^T \\ \cdots \\ \mathbf{U}_k\sigma_k\mathbf{V}_k^T \end{pmatrix} \equiv \tilde{\mathbf{G}}. \tag{4}$$

where k is the cluster number, V_i/U_i and σ_i are the i-th cluster's left/right eigen vectors and eigen values.

One of the main advantages of the approximation in (4) is that it provides a way to do fast matrix-vector multiplication in (3) for simulation:

$$\Delta\dot{\mathbf{q}} = \mathbf{G}\mathbf{f} \approx \begin{pmatrix} \mathbf{U}_1\sigma_1\mathbf{V}_1^T \\ \cdots \\ \mathbf{U}_k\sigma_k\mathbf{V}_k^T \end{pmatrix} \mathbf{f} = \begin{pmatrix} \mathbf{U}_1\sigma_1(\mathbf{V}_1^T\mathbf{f}) \\ \cdots \\ \mathbf{U}_k\sigma_k(\mathbf{V}_k\mathbf{f}) \end{pmatrix} \tag{5}$$

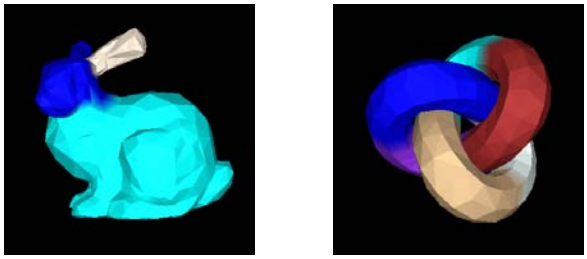


Figure 2: Force-displacement matrix clusters (for one coordinate component) of the Bunny (left) and Torusknot (right) models. In each model, the vertices of the same cluster are drawn in the same color.

Just like the classical Modal Analysis, at beginning of each simulation frame, we must project node position x in cluster c into the sub-space of linear combination of V_c.

In this scheme, if clusters have similar size, and choosing same number of modes m for each cluster, the cost of matrix-vector multiplication is about $O((1+c)mN)$, where c is the number of clusters, N is the DOF of the system. Compared with classical Modal Analysis, which is similar to one cluster in our method, the accuracy is slightly better (see Table 1) when choosing proper number of clusters. And our method extends Modal Analysis to give user the ability of choosing different number of modes in different cluster for speed-accuracy trade-off.

bunny	20N	60N	100N	140N
c=1	708.42 0.23636	170.67 0.14927	125.39 0.13114	105.70 0.12048
c=3	1616.9 0.24757	154.77 0.13998	113.38 0.12228	95.364 0.11144
c=6	2894.8 0.33766	210.59 0.14612	136.73 0.12703	106.91 0.11362

torus	20N	60N	100N	140N
c=1	1315.9 0.27955	87.757 0.08216	48.688 0.06588	39.225 0.05927
c=3	1252.7 0.22121	84.452 0.07579	43.710 0.06081	35.658 0.05461
c=6	1604.7 0.24724	103.48 0.07325	48.139 0.06104	35.881 0.05387

Table 1: Accuracy of matrix approximation.
The first row of the table is the number of needed flops for $\tilde{\mathbf{G}}\mathbf{f}$ matrix-vector multiplication. In each cell, top value is $\|G-\tilde{G}\|_F$, bottom value is the sum of all eigen-values of $G-\tilde{G}$.

For an object with 10K vertices, **G** is a 30K×30K matrix, and takes 3.6GB storage. It is obviously not practical to run the CPCA method on such a huge matrix. So, it is necessary to reduce the dimension of the row vectors before applying the CPCA method. Recalling the physical meaning of the force-displacement matrix, a vertex also should have similar responses to the forces acting on nearby vertices. Therefore, we first do another vertex clustering using the vertices' connectivity and coordinates to simplify the mesh of the object down to a desired vertex number. Based on this vertex clustering, we then sum up the corresponding columns in the force-

displacement matrix **G**, yielding some dimension reduced row vectors. We then apply the CPCA method to group the dimension reduced vectors into some clusters. Note that the resulting clusters are also applicable to the original vectors. So, we group the original vectors into clusters accordingly, and perform a SVD step in each cluster.

Another issue that needs to be addressed is how to compute the force-displacement matrix, i.e., the inverse matrix of **A**. It is still an open mathematical problem to efficiently invert a huge matrix. We use the conjugate gradient method to compute **G**'s i-th column g_i by solving $\mathbf{A}g_i = e_i$, where e_i is the i-th canonical basis. This may take a long time for large scale model with thousands of vertices, but it can be done as a preprocessing step.

3.1 Constrained Problem

It is desirable and necessary for a simulation system to allow user manipulation, reaction to external forces, and motion restriction in a particular way. In general, such tasks can be accomplished by adding constraints into the simulation system. We first address the basic point-to-nail constraint, which constrains a set of vertices to some specific positions.

Adding point-to-nail constraints to the system is equivalent to specify some of $\Delta\dot{q}$ in (2) and (3). So, for a constrained vertex v, the acceleration $\Delta\dot{q}_v$ is known while the constrained force $f_{\psi v}$ acting on v is unknown, and the total force f_v is unknown in turn. In the following we will introduce a novel method to solve unknown position and unknown forces by utilizing both matrices **A** and **G**.

Without loss of generality, we assume that the constrained set is the first m vertices. Let subscript 1 denote the constrained set of vertices ψ, and subscript 2 denote the other free set. Then (2) and (3) can be rewritten as

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} \Delta\dot{q}_1 \\ \Delta\dot{q}_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}$$
$$\begin{pmatrix} \Delta\dot{q}_1 \\ \Delta\dot{q}_2 \end{pmatrix} = \begin{pmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} \tag{6}$$

Note that $\Delta\dot{q}_1$ and f_2 are known quantities, and $\Delta\dot{q}_2$ and f_1 are unknowns in the above formulas. By substituting $\Delta\dot{q}_1$, we have

$$\begin{aligned} f_1 &= A_{11}\Delta\dot{q}_1 + A_{12}\Delta\dot{q}_2 \\ &= (A_{11}\Delta\dot{q}_1 + A_{12}G_{22}f_2) + A_{12}G_{21}f_1. \end{aligned}$$

So, the constrained f_1 can be obtained by

$$f_1 = (I - A_{12}G_{21})^{-1}(A_{11}\Delta\dot{q}_1 + A_{12}G_{22}f_2). \tag{7}$$

And $\Delta\dot{q}_2$ can be obtained by

$$\Delta\dot{q}_2 = G_{21}f_1 + G_{22}f_2. \tag{8}$$

4 Handling Large Deformation Using DDM

Due to the use of the linear elasticity model, the above approach is only applicable to small deformations near the rest state. Otherwise, there will be noticeable deformation exaggeration, especially when the object undergoes a global rotation. As long as the local deformation is small, it has been shown that such error due to the global rotation can be handled by tracking the global rotation and formulating the deformation in the object's local/reference coordinate frame. Demetri Terzopoulos and Andrew Witkin proposed such a method in [Terzopoulos and Witkin 1988] to fulfill the linear elastic equation. If object is not rotating rapidly, ignoring the motion of rigid reference will not lead to large error. Let $R_{3\times3}$ be the estimated global rotation matrix (see [Müller et al. 2002]) and

$$\mathbf{R} = \begin{pmatrix} R_{3\times3} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & R_{3\times3} \end{pmatrix}_{3n\times3n}.$$

Then, after transforming the force and displacement quantities into the local coordinate frame and applying formula (3), the global large deformation can be easily simulated using:

$$\mathbf{R}\mathbf{A}\mathbf{R}^T\Delta\dot{\mathbf{q}} = \mathbf{f}, \quad \Delta\dot{\mathbf{q}} = \mathbf{R}\mathbf{G}\mathbf{R}^T\mathbf{f}. \tag{9}$$

The above formulation is not suitable for a object with long and soft components, since there does not exist such a global rotation matrix. For such objects, we propose to partition the whole object into simple sub-objects and solve the deformations using Domain Decomposition Methods (DDM). DDM has been extensively studied in applied mathematics and mechanical engineering for partial differential equations (PDEs) in the last two decades [Quarteroni and Valli 1999], which is indeed a basic concept of numerical methods for PDEs in general. The principle of DDM is to split the original domain of computation in smaller simpler subdomains, compute local simplified solutions, and use efficient algebraic solvers to properly interface these solutions.

Suppose that the object Ω is partitioned into p non-overlapping sub-objects $\Omega = \Omega_1 \cup \ldots \cup \Omega_p$. The shared vertices between neighboring sub-objects are duplicated. Under the DDM framework, each sub-object Ω_i is modeled independently, yielding a system matrix $\mathbf{A}_i$, a force-displacement matrix $\mathbf{G}_i$. According to (9), we have

$$\mathbf{R}_i\mathbf{A}_i\mathbf{R}_i^T\Delta\dot{\mathbf{q}}_i = \mathbf{f}_i, \quad \Delta\dot{\mathbf{q}}_i = \mathbf{R}_i\mathbf{G}_i\mathbf{R}_i^T\mathbf{f}_i. \tag{10}$$

where $\mathbf{R}_i$ is the global rotation matrix of sub-object Ω_i being tracked during simulation, and $\mathbf{q}_i$ and $\mathbf{f}_i$ are respectively the displacement vector and force vector of Ω_i. Denote $\mathscr{A}_i = \mathbf{R}_i\mathbf{A}_i\mathbf{R}_i^T$, $\mathscr{G}_i = \mathbf{R}_i\mathbf{G}_i\mathbf{R}_i^T$, $\breve{\mathbf{q}} = \begin{pmatrix}\mathbf{q}_1^T & \cdots & \mathbf{q}_p^T\end{pmatrix}^T$, $\breve{\mathbf{f}} = \begin{pmatrix}\mathbf{f}_1^T & \cdots & \mathbf{f}_p^T\end{pmatrix}^T$, and

$$\mathscr{A} = \begin{pmatrix} \mathscr{A}_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \mathscr{A}_p \end{pmatrix}, \mathscr{G} = \begin{pmatrix} \mathscr{G}_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \mathscr{G}_p \end{pmatrix}.$$

Then, we have

$$\mathscr{G}_i = \mathscr{A}_i^{-1}, \quad \mathscr{G} = \mathscr{A}^{-1}, \quad \mathscr{A}\Delta\breve{\dot{\mathbf{q}}} = \breve{\mathbf{f}}, \quad \text{and } \Delta\breve{\dot{\mathbf{q}}} = \mathscr{G}\breve{\mathbf{f}}.$$

Recall that there are some duplicate vertices between the neighboring sub-objects, which should have the same displacement quantities to maintain the geometry continuity. This can be accomplished by introducing a vector of Lagrange multipliers λ to enforce a group of point-to-point constraints:

$$\mathbf{B}\Delta\breve{\dot{\mathbf{q}}} = \begin{pmatrix}\mathbf{B}_1 & \cdots & \mathbf{B}_p\end{pmatrix}\begin{pmatrix}\mathbf{q}_1^T & \cdots & \mathbf{q}_p^T\end{pmatrix}^T = \mathbf{c}.$$

where $\mathbf{c}$ is a zero vector, and the matrix $\mathbf{B} = \begin{pmatrix}\mathbf{B}_1 & \cdots & \mathbf{B}_p\end{pmatrix}$ is constructed such that each row, say the k-th row, of $\mathbf{B}$ corresponds to a pair of duplicated vertices, say the i-th and the j-th vertex, and $\mathbf{B}_{ki} = -\mathbf{B}_{kj} = I_{3\times3}$. Other elements of $\mathbf{B}$ are 0. Therefore, B is a highly sparse matrix, and we have:

$$\begin{pmatrix} \mathscr{A} & \mathbf{B}^T \\ \mathbf{B} & 0 \end{pmatrix}\begin{pmatrix} \Delta\breve{\dot{\mathbf{q}}} \\ \lambda \end{pmatrix} = \begin{pmatrix} \breve{\mathbf{f}} \\ \mathbf{c} \end{pmatrix}. \tag{11}$$

Note that other constraints for user manipulation can be easily formulated by appending more rows to $\mathbf{B}$, and some values to $\mathbf{c}$. The solution of (11) can be obtained by:

$$\begin{aligned} (\sum_i \mathbf{B}_i \mathscr{G}_i \mathbf{B}_i^T)\lambda &= \sum_i \mathbf{B}_i \mathscr{G}_i \mathbf{f}_i - \mathbf{c} \\ \Delta\dot{\mathbf{q}}_i &= G_i(\mathbf{f}_i - \mathbf{B}_i^T \lambda) \end{aligned} \tag{12}$$

Since all $\mathbf{B}_i$ are very sparse, $\sum_i \mathbf{B}_i \mathscr{G}_i \mathbf{B}_i^T$ can be easily computed. And we do SVD for this matrix to compute the solution of λ for numerical stability.

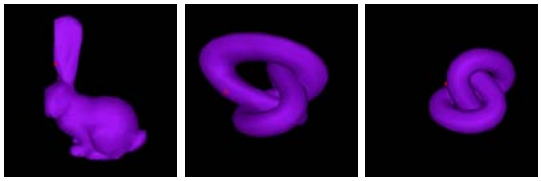
5 Experimental Results

We have implemented our methods in C++. In this section we present some experimental results

Figure 3 (a) and (b) show two simulation results using our clustering method without DDM. The simulation can runs at about 100 fps on both the Bunny and the Torusknot models.

In Figure 4, we compare the simulation results of three methods: straightforward FEM (a), modal analysis [Hauser et al. 2003] (b), and our clustering method with DDM (c). This figure clearly shows that the results of the first two methods suffer from serious simulations error, especially in the part of bunny ears, while our method with DDM can generate a good simulation result, which are visually error free. The straight forward FEM method can only run at about 5 fps. The modal analysis method uses 30 modals, and can run at about 45 fps. In our clustering method with DDM, the bunny is partitioned into 3 parts as shown in Figure 1, each part has 2 clusters, and each cluster keep 6 eigen vectors. Our method can run at bout 18 fps. Note that performance of the clustering method with DDM is much slower than that without DDM. This is caused by the cost to compute the SVD for solving the Lagrange Multiplier vector in (12). Optimizing this code should greatly increase our method's performance.

More simulation results by dragging different parts of the bunny model in our system are shown in the accompanying video.



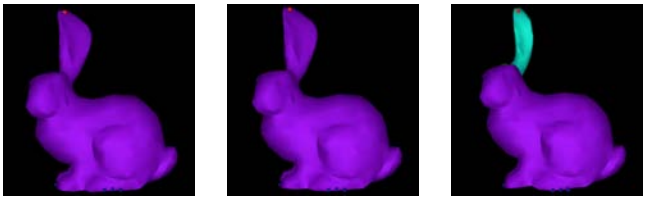
(a) (b) (c)

Figure 3: Two simulation results using our clustering method without DDM. (a) Bunny, 6 clusters, 8 eigen vectors per cluster. (b) Torusknot, 6 clusters, 8 eigen vectors per cluster. (c) is the original Torusknot's rest shape.

6 Conclusion and Discussion

We have proposed a physically based deformation method using the precomputed force-displacement matrix. The advantages of our method are

- It is very fast. The fast performance is achieved by precomputing a factorized force-displacement matrix, and using a fast matrix-vector multiplication algorithm in (5).



(a) (b) (c)

Figure 4: Simulation results comparison between (a) straightforward FEM, (b) modal analysis and (c) our clustering method with DDM (the model is colored according to DDM region).

- It is stable, and allows for relatively large time step, since essentially the implicit Euler method is used to solve the underlying differential equations.
- Large deformations are handled very well by tracking global rotations and using domain decomposition methods. DDM improves the result compared with the linear blend described in [Hauser et al. 2003]. And compared with [Müller et al. 2002; Müller and Gross 2004], our algorithm can utilize more precomputing information.
- It allows for dynamically introducing new constraints for user manipulations.

Our method also has some limitations. First, inherent from the linear elasticity model, the large deformation inside a component/subobject cannot be handled. But it does not cause any problem for the models in our experiments. Secondly, the simulation time step is fixed, and cannot be changed dynamically. We choose the time step as 0.05 second, such that the simulation looks natural at 20 fps. In addition, an extensive precomputation is needed to compute the force-displacement matrix and factorize it using CPCA. At last, the constraint number is limited, because we directly solve the constrained quantities. If there are too many constraints, the performance may be slowed. Our experiments showed that we can do fast deformation with fewer than 300 constraints.

There are many topics for future work. An interesting one is to develop a multiresolution representation for the force-displacement matrix, as what James et al. did with the precomputed Green functions [James and Pai 2003]. Another area for future work is to handle collisions during simulation. And it is also worthwhile to deal with more types of constraints. Currently, we can only handle point-to-nail, point-to-point, and force constraints. And beside of SVD decomposition on G, LU decomposition on A is worth to trying.

7 Acknowledgments

This project is supported in partial by 973 Program of China under Grant No.2002CB312102, and NSFC under Grant No.60021201 and No.60033010.

References

BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, 43–54.

BENZI, M., AND TUMA, M. 1999. A comparative study of sparse approximate inverse preconditioners. *Applied Numerical Mathematics: Transactions of IMACS 30*, 2–3, 305–340.

BENZI, M., CULLUM, J. K., AND TUMA, M. 2000. Robust approximate inverse preconditioning for the conjugate gradient method. *SIAM Journal on Scientific Computing 22*, 4, 1318–1332.

BRIDSON, R., AND TANG, W.-P. 2001. Multiresolution approximate inverse preconditioners. *SIAM Journal on Scientific Computing 23*, 2, 463–479.

BRIDSON, R., MARINO, S., AND FEDKIW, R. 2003. Simulation of clothing with folds and wrinkles. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, 28–36.

BRO-NIELSEN, M., AND COTIN, S. 1996. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *Computer Graphics Forum 15*, 3, 57–66.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIC, Z. 2002. Interactive skeleton-driven dynamic deformations. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 586–593.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIC, Z. 2002. A multiresolution framework for dynamic deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, 41–47.

CHADWICK, J. E., HAUMANN, D. R., AND PARENT, R. E. 1989. Layered construction for deformable animated characters. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, ACM Press, 243–252.

CHOI, K.-J., AND KO, H.-S. 2002. Stable but responsive cloth. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 604–611.

COOK, R. D., MALKUS, D. S., AND PLESHA, M. E. 1989. *Concepts and Applications of Finite Element Analysis*, 3rd edition ed. John Wiley & Sons.

DEBUNNE, G., DESBRUN, M., CANI, M.-P., AND BARR, A. H. 2001. Dynamic real-time deformations using space & time adaptive sampling. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, 31–36.

GIBSON, S. F. F., AND MIRTICH, B. 1997. A survey of deformable modeling in computer graphics. Tech. Rep. TR-97-19, Mitsubish Electric Research Lab., Cambridge, November.

GRINSPUN, E., KRYSL, P., AND SCHRÖDER, P. 2002. Charms: a simple framework for adaptive simulation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 281–290.

HAUSER, K., SHEN, C., AND O'BRIEN, J. F. 2003. Interactive deformations using modal analysis with constraints. In *Proceedings of Graphics Interface 2003*, 247—256.

HAUTH, M., AND ETZMUSS, O. 2001. A high performance solver for the animation of deformable objects using advanced numerical methods. In *Proceedings of Eurograhics*, 137–151.

J. LANG, D. K. P., AND SEIDEL, H.-P. 2003. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *proceedings of Graphics Interface* (June).

JAMES, D. L., AND FATAHALIAN, K. 2003. Precomputing interactive dynamic deformable scenes. *ACM Trans. Graph. 22*, 3, 879–887.

JAMES, D. L., AND PAI, D. K. 1999. Artdefo: accurate real time deformable objects. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 65–72.

JAMES, D. L., AND PAI, D. K. 2002. Dyrt: dynamic response textures for real time deformation simulation with graphics hardware. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 582–585.

JAMES, D. L., AND PAI, D. K. 2002. Real time simulation of multizone elastokinematic models. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 927–932.

JAMES, D. L., AND PAI, D. K. 2003. Multiresolution green's function methods for interactive simulation of large-scale elastostatic objects. *ACM Trans. Graph. 22*, 1, 47–82.

MÜLLER, M., AND GROSS, M. 2004. Interactive virtual materials. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, Canadian Human-Computer Communications Society, 239–246.

MÜLLER, M., DORSEY, J., MCMILLAN, L., JAGNOW, R., AND CUTLER, B. 2002. Stable real-time deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, 49–54.

PAI, D. K., VAN DEN DOEL, K., JAMES, D. L., LANG, J., LLOYD, J. E., RICHMOND, J. L., AND YAU, S. H. 2001. Scanning physical interaction behavior of 3d objects. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, 87–96.

PENTLAND, A., AND WILLIAMS, J. 1989. Good vibrations: model dynamics for graphics and animation. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, ACM Press, 215–222.

QUARTERONI, A., AND VALLI, A. 1999. *Domain Decomposition Methods for Partial Differential Equations*. Oxford Science Publications, Clarendon Press, Oxford.

SLOAN, P.-P., HALL, J., HART, J., AND SNYDER, J. 2003. Clustered principal components for precomputed radiance transfer. *ACM Trans. Graph. 22*, 3, 382–391.

STAM, J. 1997. Stochastic dynamics: Simulating the effects of turbulence on flexible structures. *Comput Graphics Forum 16*, 3 (Sept.), 159–164.

TERZOPOULOS, D., AND WITKIN, A. 1988. Physically based models with rigid and deformable components. *IEEE Comput. Graph. Appl. 8*, 6, 41–51.

TU, X., AND TERZOPOULOS, D. 1994. Artificial fishes: physics, locomotion, perception, behavior. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM Press, 43–50.

WU, X., DOWNES, M. S., GOKTEKIN, T., AND TENDICK, F. 2001. Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes. *Comput Graphics Forum 20*, 3 (Sept.), 349–358.