Real-time Global Deformations*

Yan Zhuang†
Computer Science Department, University of California, Berkeley, CA 94720-1776
John Canny‡
Computer Science Department, University of California, Berkeley, CA 94720-1776

Real-time simulation and animation of 3D global deformations is the bottleneck of many applications, such as surgical simulator. In this paper, we present a system that simulates physically realistic large deformations of soft objects in real-time. We achieve the physical realism by modeling the global deformation using geometrically nonlinear finite element methods. We achieve real-time dynamic simulation of models of reasonable complexity by preprocessing the LU-factorization of a small number of large matrices. To reduce the time and space required for such a preprocess, and the time of the back-substitution, we apply modified nested dissection to reorder the finite element mesh. We also introduce an efficient method that handles deformable object collisions with almost no extra cost.

1 Introduction

Physically realistic modeling and manipulation of deformable objects has been the bottleneck of many applications, such as human tissue modeling, character animation, surgical simulation, etc. Among the potential applications, a virtual surgical training system is the most demanding for the real-time performance because of the requirement of real-time interaction with virtual human tissue.

In this paper, we address the bottleneck problem of real-time simulation of physically realistic large *global deformations* of 3D objects. In particular we apply the finite element method (FEM) to model such deformation. By *global deformation*, we mean deformations, such as large twisting or bending of an object, which involve the entire body, in contrast to poking and squeezing, which involve a relatively small region of the deformable object.

First, we point out that the application of linear elastic finite element methods to simulating large global deformation leads to unacceptable distortions (Figure 1 and 2). To avoid such distortions, we simulate global deformations using nonlinear finite element methods (section 3).

Secondly, we achieve real-time simulation by restricting time steps, which are not constants, to a small set of values. Finite element methods usually require solving a large sparse linear system at each time step. The restriction of time steps leads to a small number of possible matrices to invert. This enables us to pre-compute all inverse matrices, represented by its LU-factorization. With the precomputed LU-factorization, only back-substitution is needed at each time step of the simulation. In section 5, we discuss our modified nested dissection algorithm, which reduces both the time for LU-factorization and that of back-substitution.

Finally, we propose an efficient collision handling method for FEM in section 6. Simulating deformable object collisions using a penalty method [21] requires tiny time steps to generate visually satisfactory animations. A general impulse collision [1] is considered more efficient and accurate but still requires more computational power than collision-free dynamics. In section 6 we present an extremely simple and efficient collision time integration scheme, which makes the time integration of collision dynamics as cheap as that of collision-free dynamics.

2 Related Work

Our work of modeling and simulating a deformable object falls into the realm of physically based modeling.

*Supported by a Multi-Disciplinary Research Initiative grant for 3D Visualization, sponsored by BMDO with support from ONR.

†yzhuang@cs.berkeley.edu
‡jfc@cs.berkeley.edu

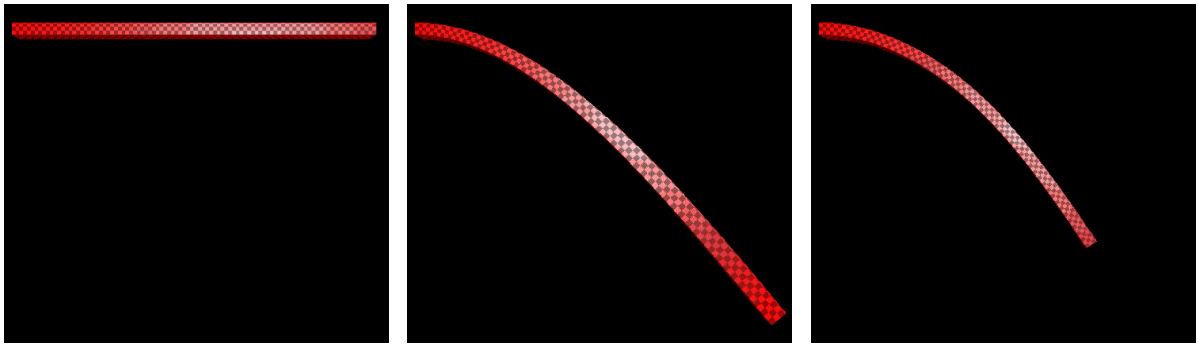


Figure 1: *The left image shows a beam at its initial configuration with a fixed left end and a free right end. The middle image shows the* distorted *deformation under gravity, using linear strain. The right image shows the* undistorted *deformation, under the same gravitational force, using quadratic strain (equation (5) and (6)).*

Witkin *et al*[25] summarizes the methods and principles of physically based modeling, which has emerged as an important new approach to computer animation and computer graphics modeling.

In general, there are two different approaches to modeling deformable objects: the mass spring model and the finite element model. Gibson and Mirtich [9] gives a comprehensive review of this subject.

The mass spring model has had good success in creating visually satisfactory animations. Waters [23] uses a spring model to create a realistic 3D facial expression. Provot *et al*[18] describes a 2D model for animating cloth, using double cross springs. Promayon *et al*[17] presents a mass-spring model of 3D deformable objects and develops some control techniques.

Despite the success in some animation applications, the mass spring models do not model the underlying physics accurately, which makes it unsuitable for simulations that require more accuracy. The structure of the mass spring is often application dependent and hard to interpret. The animation results often vary dramatically with different spring structures. The distribution of the mass to nodes is somewhat (if not completely) arbitrary. Despite its inaccuracy, it does not have visual distortion and it is computationally cheap to integrate over time because the system is, by its very nature, a set of independent algebraic equations, which requires no matrix inversions to solve.

As an alternative, finite element methods (*FEM*) model the continuum much more accurately and their underlying mathematics are well studied and developed. Another similar method is the finite difference method, which is less accurate but simpler and appropriate for some applications. Indeed a linear finite difference method over a uniform mesh is just a special case of FEM. Its accuracy and mathematical rigorousness make FEM a better choice for applications such as surgical simulations.

Terzopoulos *et al*[21, 20, 22] applies both finite difference and finite element methods in modeling elastically deformable objects. Celniker *et al*[15] applies FEM to generate primitives that build continuous deformable shapes designed to support a new free-form modeling paradigm. Pieper *et al*[16] applies FEM to computer-aided plastic surgery. Chen [3] animates human muscle using a 20 node hexahedral FEM mesh. Keeve *et al*[11] develops a static anatomy-based facial tissue model for surgical simulation using the FEM. Most recently, Cotin *et al*[5] presents real-time elastic deformation of soft tissues for surgery simulation, which only simulates static deformations.

James and Pai [10] model real-time static local deformations using the boundary element method (BEM). BEM has the advantage of solving a smaller system because it only deals with degrees of freedom on the surface of the model. However, the resulting system is dense. It is difficult to apply boundary element method to model non-homogeneous material.

Our work differs from the previous work by either one or all of the following: (1) we simulate large global deformations instead of small local deformations; (2) we simulate the dynamic behavior of soft objects rather than the static deformation.

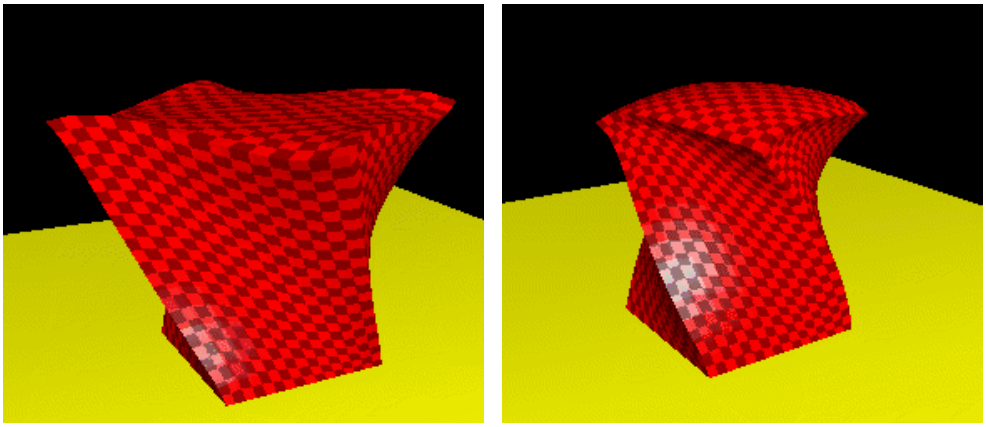


Figure 2: *The bottom of the object is fixed and its top is twisted. The top in the left image is distorted (grown bigger) because it is simulated using linear elasticity. The right image shows that the same distortion does not occur with nonlinear elasticity.*

3 Nonlinear Elasticity with FEM

By *global deformations*, we mean deformations that are large and involve the entire body, such as high amplitude bending and twisting (figure 1 and 2). These types of deformation often occur to soft objects, such as tissue in surgical simulations.

The theory of elasticity is a fundamental discipline in studying continuum material. It consists of a consistent set of differential equations that uniquely describe the state of stress, strain and displacement of each point within an elastic deformable body. It consists of equilibrium equations relating the stresses; kinematics equations relating the strains and displacements, constitutive equations relating the stresses and strains; and boundary conditions relating to the physical domain. The theory was first developed by Louis-Marie-Henri Navier, Dimon-Denis Poisson and George Green in the first half of the 19th century [24].

Synthesizing those equations allows us to establish a relationship between the deformation of the object and the exerted forces. However an analytic expression of such relationship is impossible, except for a small number of simple problems. *Finite element methods (FEM)* are one way to solve such a set of differential equations. From now on, we will discuss elasticity within the context of finite element methods.

When the geometry of the deformable object is complicated, it is impossible to obtain an analytic solution of an elastic deformation. FEM solves this problem by subdividing the object into small sub-domains with simple shapes (tetrahedra, hexahedra, etc.), called finite elements. The sub-division (mesh) does not only approximate the original geometry, but also leads to a discrete representation of the deformation.

In particular, we apply a *displacement based* finite element method to simulate such deformation. Namely displacements at vertices of the mesh, called nodes, will be calculated. The values at other points within the element are interpolated by continuous functions, usually low order polynomials, using the nodal values. The global equations (the relationship between all the nodal values) are obtained by assembling elementwise equations by imposing inter-element continuity of the solution and balancing of inter-element forces.[1] This essentially requires solving the following system of differential equations:

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{D}\dot{\mathbf{u}} + \mathbf{R}(\mathbf{u}) = \mathbf{F} \tag{1}$$

where $\mathbf{u}$ is the $3n$-dimensional nodal displacement vector; $\dot{\mathbf{u}}$ and $\ddot{\mathbf{u}}$, the respective velocity and acceleration vectors; $\mathbf{F}$, the external force vector; $\mathbf{M}$, the $3n \times 3n$ mass matrix; $\mathbf{D}$, the damping matrix; and $\mathbf{R}(\mathbf{u})$, the internal force vectors due to deformation. n is the number of nodes in the FEM model [29].

To our best knowledge the published research ([16, 3,

[1] Detailed discussions of FEM can be found in [19, 28].

11, 5]) assume small deformations in their virtual environment. The most simulated deformations are those caused by squeezing and poking at a relatively small surface region. The small deformation assumption leads to the often used linear elasticity model, which is based on the following linear strain approximations:

$$\epsilon_x = \frac{\partial u}{\partial x} \tag{2}$$

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \tag{3}$$

where x, y and z are the independent variables of the cartesian frame, and u, v and w are the corresponding displacement variables at the given point. Other terms of the strain at point (x, y, z), ϵ_y, ϵ_z, γ_{yz} and γ_{zx}, are defined similarly.

This linear strain makes the internal force vector linear with respect to nodal displacement vector. Namely it simplifies equation (1) to the following *linear* system:

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{D}\dot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{F} \tag{4}$$

This allows a preprocessing step that computes the constant stiffness matrix $\mathbf{K}$ and its LU factorization. This preprocessing step has been the key to real-time performance in previous works, such as [5], which animates deformations using a sequence of static equilibria.

The problem with this linear strain approximation is that it does not model finite rotation correctly. As a result, it introduces distortions when large global deformations occur (figure 1 and 2), because global deformations usually involve finite rotation of part of the object relative to the rest of it.

To further illustrate this distortion, let us subject an undeformed object to a rigid body rotation. Apparently, the rotation should not introduce any deformation to the object. Namely the strain at any point within the object should be zero. However equations (2) and (3) give a nonzero strain. This "artificial" strain leads to distortion, because the body has to deform in a certain way to balance the stress caused by such an "artificial strain".

To avoid the distortion, as shown in figures (1) and (2), we model the deformation using the exact strain, which is quadratic as following:

$$\epsilon_x = \frac{\partial u}{\partial x} + \frac{1}{2}\left[\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial w}{\partial x}\right)^2\right] \tag{5}$$

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} + \left[\frac{\partial u}{\partial x}\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\frac{\partial v}{\partial y} + \frac{\partial w}{\partial x}\frac{\partial w}{\partial y}\right] \tag{6}$$

The other 4 terms of the strain are defined similarly. It is easy to verify that the above nonlinear strain handles arbitrary large rigid body motions correctly. Namely no artificial strain will be introduced when we subject the object to a rigid body motion.

This quadratic strain makes (1) a nonlinear system, in which the internal force $\mathbf{R}(\mathbf{u})$ is no longer a linear term of nodal displacements. If we solve this nonlinear system using an implicit integration scheme such as [2], real time simulation is impossible for reasonably large meshes.

We observe that a soft material such as live tissue has small stiffness in all directions (not necessarily isotropic). This makes explicit time integration schemes appropriate because we can take large time steps. We apply the explicit Newmark scheme to equation (1), which leads to the following equations:

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \dot{\mathbf{u}}_n \triangle t_n + \frac{1}{2}\ddot{\mathbf{u}}_n \triangle t_n^2 \tag{7}$$

$$(\mathbf{M} + \frac{1}{2}\triangle t_n \mathbf{D})\ddot{\mathbf{u}}_{n+1} = \mathbf{F}_{n+1} - \mathbf{R}(\mathbf{u}_{n+1}) - \mathbf{D}(\dot{\mathbf{u}}_n + \frac{1}{2}\ddot{\mathbf{u}}_n \triangle t_n) \tag{8}$$

$$\dot{\mathbf{u}}_{n+1} = \dot{\mathbf{u}}_n + \frac{1}{2}(\ddot{\mathbf{u}}_n + \ddot{\mathbf{u}}_{n+1})\triangle t_n \tag{9}$$

Newmark scheme converts a nonlinear system to 3 linear systems (7), (8) and (9). The order of updating is (7), (8) and then (9).

4 Preprocessing the Inverse Matrices

The bottleneck of Newmark scheme is solving equation (8). It requires inverting a large sparse matrix $\mathbf{M} + \frac{1}{2}\triangle t_n \mathbf{D}$. This matrix is not a constant matrix because the time step $\triangle t_n$ may vary over time. Inverting a different large sparse matrix makes real time performance impossible.

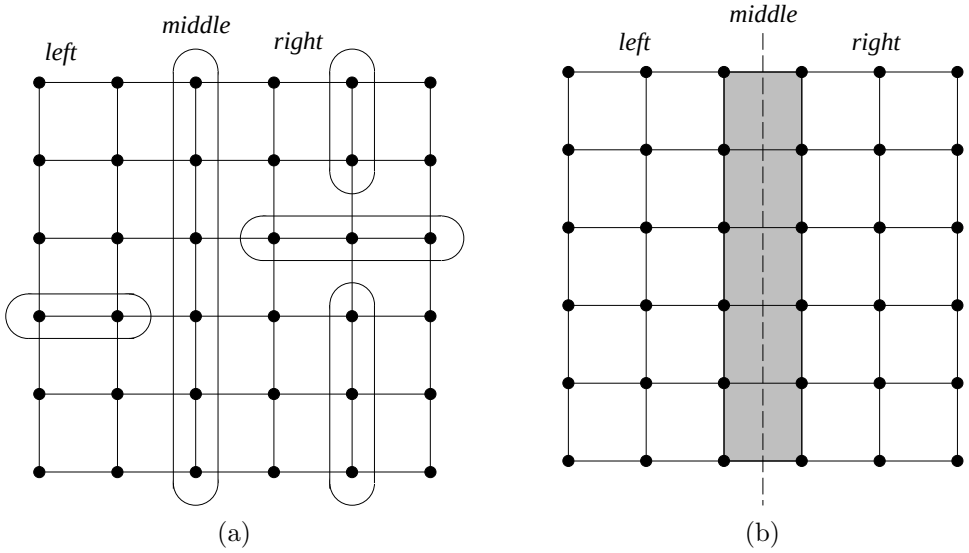


Figure 3: *(a) The dissector in the regular nested dissection algorithm is a layer of nodes. (b) The modified nested dissection algorithm uses a layers of elements (shaded), cut by a plane (or a line in 2D), as the dissector.*

To achieve real-time performance, Zhuang and Canny [27] approximated this matrix with its row-lumped diagonal matrix. This is equivalent to diagonalizing both the mass matrix $\mathbf{M}$ and the damping matrix $\mathbf{D}$. The diagonalization of $\mathbf{M}$ is acceptable because it still preserves the global inertia property of the object, although it does not preserve the local moment of inertia. The diagonalization of the damping matrix may lose important viscous elasticity property of the material. For simulations that require more physical realism, diagonalization of matrix $\mathbf{D}$ is not appropriate.

In this section, we propose a different treatment by preprocessing. The matrices $\mathbf{M}$ and $\mathbf{D}$ are contants. The only variable is $\triangle t_n$. The time step $\triangle t_n$ depends on the stability requirement of the system and the collision handling requirement. Instead of approximating these two matrices as [27] does, we restrict the time step $\triangle t_n$ to a small set of values. Let T be the largest time step allowed, we define the restricted set of allowed time steps as $\{T/2^i | i = 0, 1, \ldots, m\}$. We choose the value m such that $T/2^m < T_{min}$, where T_{min} is the minimum time step in the worst case.

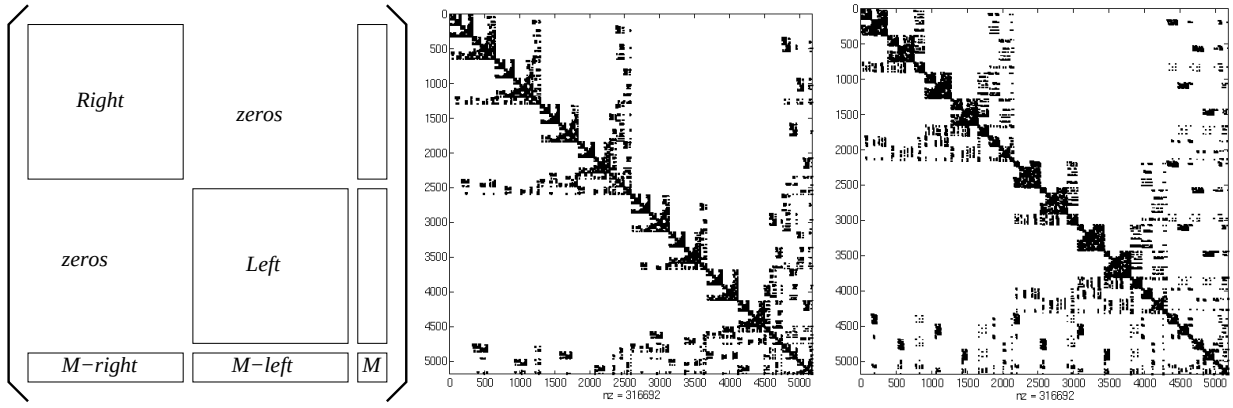
By restricting time steps to such a small set of values, we only have $m + 1$ possible matrices needed for the entire simulation. We can therefore pre-compute the $m + 1$ inverse matrices before the simulation begins.

Instead of precomputing the inverse of matrix $(\mathbf{M} + \frac{1}{2}\triangle t_n \mathbf{D})$, we precompute its LU-factorization. Given the LU-factorization, solving equation (8) only requires back-substitution. The time for back-substitution is determined by the number of nonzeros in the LU-factorization of $(\mathbf{M} + \frac{1}{2}\triangle t_n \mathbf{D})$. In section 5, we discuss how to reduce number of nonzeros in the LU-factorization.

5 Nested Dissection

A typical finite element simulation has to solve a large sparse linear system of large number of nonzero entries. For example, a $10 \times 10 \times 10$ linear hexahedral mesh for 3D linear elasticity gives a sparse matrix of 3993×3993 with 242435 non-zeros, which is about 1.5% of the size of a dense matrix with the same dimensions. To solve such a system efficiently, we have to avoid operating on zeros as much as we can. However how efficiently we can do so largely depends on the sparsity of the matrix: *the position of the nonzero entries.*

Given a finite element model of a physical problem, the values of non-zeros of $(\mathbf{M} + \frac{1}{2}\triangle t_n \mathbf{D})$ are determined by the underlying model, while the positions of those non-zeros are determined by the indices of the variables. For convenience, let us denote matrix $(\mathbf{M} + \frac{1}{2}\triangle t_n \mathbf{D})$ by $\mathcal{A}$. The entry (i, j) of $\mathcal{A}$ is nonzero



(a) (b) (c)

Figure 4: *(a) The block structure of sparse matrix $\mathcal{A}$ after the first dissection. (b) The sparse matrix structure generated by nested dissection. (c) The sparse matrix ordering using non-gird nested dissection.*

if and only if the variable x_i and x_j are related. Given such a sparse matrix $\mathcal{A} = \mathcal{LU}$, the LU-factorization takes $O(\sum_j d_j)$ space and $O(\sum_j d_j^2)$ time, where d_j is the number of non-zeros in each column vector of $\mathcal{L}$ [8]. If we assume that no exact numerical cancellations can occur, $\mathcal{L}$ will have non-zeros below the diagonal everywhere that $\mathcal{A}$ does. We define *fills* to be the below-diagonal entries in which $\mathcal{L}$ is nonzero and the corresponding entry of $\mathcal{A}$ is zero.[2]

Different ordering of the row and column vectors of the matrix $\mathcal{A}$ has no effect on the underlining physical problem that we are solving. However it dramatically change the number of fills. In order to reduce the space and running time for LU-factorization and the time of the corresponding back-substitution, we would like to minimize the number of fills. Unfortunately finding the order that gives the smallest fills is an NP-complete problem [26].

For sparse matrix that arises from *regular* finite element mesh, George[6] proposed a heuristic called *nested dissection* for ordering the variables of the system such that it gives a small number of fills.

Unfortunately a FEM mesh is often unstructured. In this section, we propose a modified nested dissection that works on any *unstructured* finite element mesh.

[2] $\mathcal{A}$ is symmetric.

5.1 Modified Nested Dissection

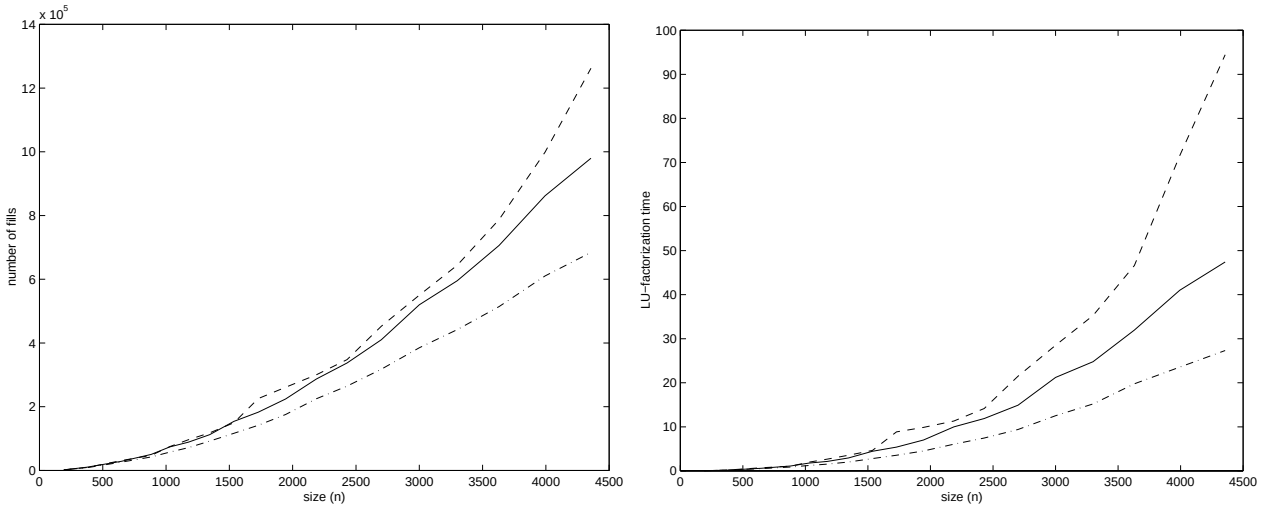
For the simplicity of the presentation, let us consider a 2-dimensional finite element mesh, where each node has one degree of freedom.[3] A mesh of n nodes leads to a sparse matrix $\mathcal{A}$ of size $n \times n$. An entry (i, j) is nonzero if and only if the node i and j are in the same element.

For the mesh generated on regular grid (3(a)) The regular nested dissection [6] algorithm recursively divide the unordered nodes into 3 groups: *left, middle* and *right*. The group *middle* is just a set of nodes that completely separate *left* and *right*. This algorithm orders the nodes such that its sparse matrix has fractal sparsity as shown in figure 4(b). After the first step of recursion, we immediately get 2 blocks of zeros as shown in figure 4(a), because *left* and *right* are *not* directly related.

Unfortunately regular nested dissection algorithm requires a finite element mesh defined on regular grid. For an unstructured mesh, it would be difficult to find *middle* to dissect the mesh. In computer graphics, most meshes are unstructured. This motivated us to extend regular nested dissection to unstructured finite element mesh.

Instead of separating the set of unordered nodes using a layer of nodes, we "cut" the mesh by a axis-aligned plane. It is easy to compute the set of elements

[3] This can be easily generalized to multiple degrees of freedom and 3-dimensional meshes.



(a) (b)

Figure 5: *The result for minimum degree ordering is plotted in dashed line, non-grid nested dissection in solid line, and the standard nested dissection in dash-dot line. (a) Number of fills. (b) Running time.*

cut by this plane. We let *middle* be the set of unordered nodes in the elements cut by this plane and continue recursively as the regular nested dissection.

This modified nested dissection can be applied to any unstructured finite element meshes, including tetrahedral meshes. Figure 3(b) shows one step of such a dissection. It also leads to the block structure as shown in figure 4(a), except that the size of the M-block is bigger. At the end, we still get an ordering that gives a sparse matrix with fractal sparsity (Figure 4(c)). Due to the larger size of matrix M, the two "wings" ($M - left$ and $M - right$) are wider at each level.

The pseudo code of the modified nested dissection is as following:

Modified-nested-dissection($\mathcal{E}$, *top*, *perm*)
  if $\mathcal{E}.length = 0$ **then**
    return.
  else if $\mathcal{E}.length = 1$ **then**
    for each node j in $\mathcal{E}[0]$ **do**
      if $perm[j]$!= -1 **do**
        $perm[j] = top$
        $top - -$
      endif
  else
    Find-dissector($\mathcal{E}$).
    $middle$ = all the elements cut by the dissector.
    $left$ = elements to the "left" of the dissector.
    $right$ = elements to the "right" of the dissector.
    Modified-nested-dissection($middle$, *top*, *perm*).
    Modified-nested-dissection($left$, *top*, *perm*).
    Modified-nested-dissection($right$, *top*, *perm*).
  endif

Before calling this function the first time, we initialize each entry of the permutation array *perm* to -1 (order unassigned), and we compute the centroid of each element. The function *Find-dissector* simply computes the 3 medians along x, y and z direction and compare the number of elements cut by the axis-aligned planes thru the medians and return as the dissector the plane with the minimum cut.

5.2 Running time

It takes $O(n)$ time to find the median given a list of n numbers [4]. The depth of the recursion is apparently $O(\log n)$. Thus the total running time is $O(n \log n)$.

5.3 Numerical Experiments

In order to measure the performance of the modified nested dissection algorithm, we compare its fills and LU-factorization time with that of regular nested dissection and that of *minimum-degree* algorithm [12, 7].

Test	Size(n)	nnz	Fills			
			random	minimum degree	modified nested-dissection	nested dissection
1	192	6348	3636	1836	2034	1980
2	375	15285	25722	9342	10269	8586
3	648	30060	89199	31041	28872	26145
4	882	42384	196641	47286	50481	42462
5	1029	52131	254718	74880	73845	57537
6	1176	60360	332766	96381	88785	71685
7	1344	69888	487701	117900	112365	91971
8	1536	82956	657612	149652	153936	116514
9	1728	94266	813186	225801	182817	141741
10	1944	107118	1036521	261072	224280	175743
11	2187	123993	1361250	300312	286929	224973
12	2430	138870	1687527	347985	337383	264609
13	2700	155532		452574	410346	317898
14	3000	176700		550215	519804	385200
15	3300	195630		644067	595296	442053
16	3630	216588		787410	706131	514395
17	3993	242535		1100817	861705	610515
18	4356	266004		1262097	979875	684378

Table 1: *Number of fills for different ordering.*

All the matrices are derived from a 3-dimension finite element mesh of different size. The comparison of number of fills is listed in table 1 and that of the LU-factorization time is listed in table 2.

Minimum-degree ordering is an alternative ordering proposed to handle general matrix. It is an greedy algorithm that does the ordering directly on the connectivity graph defined by the matrix. Our numerical experiments show that our modified nested dissection algorithm has an apparent advantage over the minimum-degree ordering, in both space and running time. Our modified nested dissection algorithm produces an order that has less fills, than the minimum-degree ordering, in 17 of 18 tests, while it has a better LU-factorization time in all 18 test. The result is plotted in figure 5. This suggests that the geometry of the mesh gives better heuristic than its connectivity graph.

Also it is worth to note that the modified nested dissection ordering requires significantly less time for LU-factorization while only having slightly less fills than the minimum-degree ordering. This shows that the modified nested dissection leads to better sparsity: non-zeros are more optimally positioned in the matrix.

6 Collision Handling

For collisions involving deformable objects, the collision time can be assumed finite (unlike the instantaneous collision of rigid bodies). This allows a larger time step for numerical integration.

The popular penalty methods [21, 20, 22] model the collision by adding an artificial spring of large stiffness at the point of collision. This stiff spring requires tiny integration time steps to stably simulate a collision. Various experiments show that the ratio between a collision free integration time step and that of a penalty collision is on the order of hundreds if not more.

This tempts us to develop new collision-handling methods that avoid adding extra artificial stiffness into the system. We will illustrate our collision-handling method, using a special case: collision between a rigid body and a single node of the FEM mesh of the deformable body. (figure 6). Later in this section, we will show that it is straightforward to extend this method to handle general collisions of deformable objects.

Consider the collision between a moving deformable body and a moving rigid body (figure 6). To simpli-

Test	Size(n)	nnz	LU factorization time (seconds)			
			random	minimum degree	modified nested-dissection	nested dissection
1	192	6348	0.07	0.05	0.04	0.04
2	375	15285	0.55	0.18	0.17	0.17
3	648	30060	3.38	0.70	0.58	0.53
4	882	42384	10.66	1.07	1.11	0.87
5	1029	52131	15.31	1.98	1.77	1.23
6	1176	60360	22.75	2.67	2.15	1.54
7	1344	69888	40.08	3.54	2.93	2.02
8	1536	82956	61.19	4.65	4.44	2.80
9	1728	94266	86.39	8.86	5.39	3.56
10	1944	107118	123.27	9.88	7.00	4.52
11	2187	123993	192.59	11.31	9.99	6.05
12	2430	138870	274.96	14.15	11.87	7.45
13	2700	155532		21.49	14.91	9.39
14	3000	176700		28.49	21.20	12.51
15	3300	195630		35.26	24.78	15.20
16	3630	216588		46.60	31.96	19.76
17	3993	242535		87.84	41.00	23.56
18	4356	266004		94.44	47.41	27.31

Table 2: *Time measured on HP9000/715.*

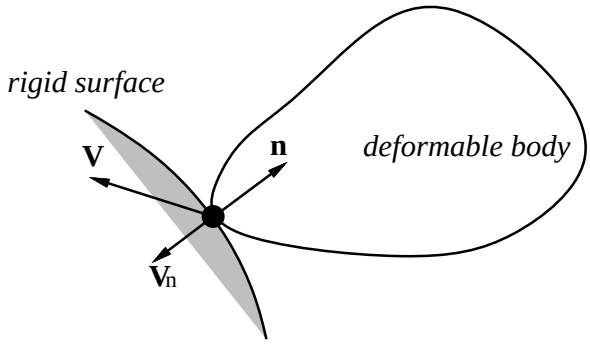


Figure 6: *A flexible body collides with a rigid body.*

fy the discussion, we use the moving frame attached to the moving rigid body instead of the fixed world frame. Namely all quantities are relative to the moving rigid body. Assume that at time t_n, the node p on the deformable object, with *relative* velocity $\hat{\mathbf{v}}(p)_n$, is colliding with the rigid surface of outward normal $\hat{\mathbf{n}}$. The non-penetration constraint requires that the normal component of the relative velocity of point p drops to zero at the moment of collision in the moving frame. Unlike a rigid body collision, the flexible body will maintain contact with the rigid body for a nonzero period of time. We enforce the non-penetration constraint at node p by setting the normal component of $\hat{\mathbf{v}}(p)_{n+1}$ to zero as following:

$$\hat{\mathbf{v}}(p)_{n+1} = \hat{\mathbf{v}}(p)_n + (\hat{\mathbf{v}}(p)_n \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} \tag{10}$$

By equation (9), we get

$$\hat{\mathbf{a}}(p)_{n+1} = \frac{2\hat{\mathbf{v}}(p)_{n+1}}{\triangle t_n} - \frac{2\hat{\mathbf{v}}(p)_n}{\triangle t_n} - \hat{\mathbf{a}}(p)_n \tag{11}$$

If we choose $\triangle t_{n+1} = \triangle t_n$, by equation (7), we have

$$\hat{\mathbf{u}}_{n+2} \cdot \hat{\mathbf{n}} = \hat{\mathbf{u}}_n \cdot \hat{\mathbf{n}} \tag{12}$$

This shows that the non-penetration constraint is enforced after two time steps, because there is no relative

motion of the deformable body normal to the surface of the rigid body.

This collision-handling integration scheme can be considered a special case of impulse [1]. For rigid body collisions, an impulse requires extremely small time steps for numerical integration because the rigid body collision is considered to occur instantaneously. However, for deformable body collisions, the collision time is finite. By delaying the non-penetration constraint by two time steps, we are able to integrate the impulse using large time steps.

This collision integration scheme can be generalized to collisions between deformable bodies and collisions that involve multiple point contacts. Multiple point collisions are modeled as a set of simultaneous single point collisions.

Unlike a general impulse [1, 13, 14], we do not have to distinguish the case that the colliding deformable objects quickly bounce away from each other and that one sticks to or slides on the surface of the other. The bouncing collision, the sticking and sliding contacts, are handled by exactly the same collision integration constraint. This collision constrain adds little extra cost to the dynamic simulation.

7 Conclusions and Future Work

We presented a simulation system that simulates global 3D deformations in real-time. Due to the distortion associated with linear strain, we simulate the global deformation using geometrically nonlinear finite element methods. The nonlinear FEM formulation is derived from the application of the nonlinear exact strain.

It is in general too expensive to solve such a nonlinear FEM system in real time. In order to achieve real-time performance, we pre-compute the LU-factorization of a small number of large sparse matrices. Such preprocessing is possible because we restrict the time steps to a small set of values. Our experiments show that usually we only need no more than 3 different values for time steps.

To reduce the time and space for LU-factorization and the time of back-substitution, we apply nested dissection to reorder the vertices in the finite element mesh. Such a reordering does not change the physical model that we are simulating. But it dramatically reduce the number of nonzeros in the LU-factorization.

We modified the regular nested dissection algorithm so that it works on unstructured finite element mesh. The modified nested dissection ordering takes 30% to 50% more time for the LU-factorization than the regular nested dissection, however it is more general than the regular nested dissection: *it is able to handle any unstructured finite element mesh.*

Our current implementation simply uses a cutting plane and separate the mesh using the elements intersected by the cutting plane. It seems that there is a better algorithm using sweeping-line (plane). By using a sweeping-line (plane) approach, we may be able to find a "thin" layer of nodes (rather than a layer of elements) to separate the mesh. This way, we would be able to generalize the regular nested dissection algorithm to unstructured mesh without giving away any performance. We are currently studying this approach.

We also introduced an efficient collision constraint. This constraint enables us to simulate the collisions with little extra computation, compared to a collision free simulation step. Our experiments show that this collision constraint handles collision much more efficient than penalty method.

Currently our system is able to produce real-time graphics for a mesh of several hundred vertices. We are interfacing our system to haptic devices, such as a Phantom, so that users can interact with deformable objects in real-time.

8 Acknowledgement

We thank Panayiotis Papadopoulos for sharing with us his FEM expertise and Jonathan Shewchuk for his insight in 3D meshing.

References

[1] David Baraff and Andrew Witkin. Dynamic simulation of non-penetrating flexible bodies. In *Computer Graphics: Proceedings of SIGGRAPH*, pages 303–308. ACM, 1992.

[2] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Computer Graphics: Proceedings of SIGGRAPH*, pages 303–308. ACM, 1998.

[3] David Chen. *Pump It Up: Computer Animation of a Biomechanically Basded Model of Muscle Using the Finite Element Method.* PhD thesis, MIT, 1992.

[4] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 10 edition, 1993.

[5] Stéphane Cotin, Hervé Delingette, and Nicholas Ayache. Real-time elastic deformations of soft tissues for surgery simulation. *IEEE Transcation on Visualization and Computer Graphics*, 5(1):62–73, January-March 1999.

[6] Alan George. Nested dissection of a regular finite element mesh. *SIAM Jounal of Numerical Analysis*, 10(2), 1973.

[7] Alan George and Joseph W. H. Liu. A fast implementation of the minimum degree algorithm using quotient graphs. *ACM Transaction on Mathematical Software*, 6(3), September 1980.

[8] Alan George and Joseph W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Inc., 1981.

[9] Sarah F. Gibson and Brian Mirtich. A servey of deformable models in computer graphics. Technical Report TR-97-19, Mitsubishi Electric Research Laboratories, Cambridge, MA, November 1997.

[10] Doug L. James and Dinesh K. Pai. Artdefo: Accurate real time deformable objects. *Computer Graphics: Proceedings of Siggraph*, pages 65–72, August 1999.

[11] E. Keeve, S. Girod, P. Pfeifle, and B. Girod. Anatomy-based facial tissue modeling using the finite element method. *IEEE Visualization*, 1996.

[12] Joseph W. H. Liu. Modification of minimum-degree algorithm by multiple elemination. *ACM Transaction on Mathematical Software*, 11(2), June 1985.

[13] Brian Mirtich and John Canny. Impulse-based dynamic simulation. In K. Goldberg, D. Halperin, J.C. Latombe, and R. Wilson, editors, *The Algorithm Foundations of Robotics*. A. K. Peters, Boston, MA, 1995. Proceedings from the workshop held in February, 1994.

[14] Brian Mirtich and John Canny. Impulse-based simulation of rigid bodies. In *Symposium on Interactive 3D Graphics*, New York, 1995. ACM Press.

[15] G. Celniker nad G. Gossard. Deformable curve and surface finite elements for free form shage design. *Computer Graphics*, 25(4), 1991.

[16] S. Peiper, J Rosen, and D. Zeltzer. Interactive graphics for plastic surgery: A task-level analysis and implementation. In *Symposium on Interactive 3D Graphics*, 1992.

[17] E. Promayon, P. Baconnier, and C. Puech. Physically-based deformations constrained in displacements and volume. In *EUROGRAPHICS*, 1996.

[18] X. Provot. Deformation constrains in a mass-spring model to describe rigid cloth behavior. *Computer Interface*, 1995.

[19] J. N. Reddy. *An Introduction to the Finite Element Method*. McGraw-Hill, Inc., 2nd edition, 1993.

[20] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. *Computer Graphics*, 22, August 1988.

[21] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. *Computer Graphics*, 21, July 1987.

[22] D. Terzopoulos and K. Waters. Physically-based facial modeling, analysis and animation. *Journal of Visualization and Computer Animation*, 1990.

[23] K. Waters. A muscle model for animating three-dimensional facial expression. *Computer Graphics*, 21(4), July 1987.

[24] H. M. Westergaard. *Theory of Elasticity and Plasticity*. Dover Publications, Inc., 1964.

[25] A. Witkin and *et al*. An introduction to physically based modeling. Course Notes, 1993.

[26] M Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal of Algrbraic Discrete Methods*, 2, 1981.

[27] Yan Zhuang and John Canny. Real-time simulation of physically realistic global deformations. *IEEE Visualization: Late Breaking Hot Topics*, October 1999.

[28] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method: Basic Formulation and Linear Problems*, volume 1. McGraw-Hill Book Company, 4th edition, 1989. linear finite element method, linear elasticity.

[29] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method: Solid and Fluid Mechanics Dynamics and Non-Linearity*, volume 2. McGraw-Hill Book Company, 4th edition, 1989.