A hybrid condensed finite element model with GPU acceleration for interactive 3D soft tissue cutting

By Wen Wu* and Pheng Ann Heng

To meet the requirement of computer-aided medical operations, apart from the real-time deformation, it is also necessary in the design to simulate the tissue cutting and suturing in a surgery simulation. In this paper, we present a model on topology change and deformation of soft tissue, referred to as the hybrid condensed finite element model, based on the volumetric finite element method. The most important advantage of our model is its ability to achieve an interactive frame rate for the topology change in surgical simulation on standard PC platform. This is achieved through two innovations. One is to apply the condensation technique, by fully calculating the volumetric deformation in the operation part while only calculating the surface nodes in the non-operation part. Secondly, the major calculation work in the Conjugate Gradient solver for cutting and deformation is migrated from the CPU to the contemporary GPU to promote the calculation. Test examples have been given to show the feasibility and efficiency of the model. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: surgical simulation; soft tissue cutting and deformation; the GPU computation; the hybrid condensed finite element model

Introduction

In recent decades, the minimally invasive microsurgery has been developed as a renovation technique. It has the advantages of reducing the operation time and the traumatizing degree of patients. It requires the surgeons to be highly experienced in hand-eye coordination skill during operations. Therefore, the surgery simulation is expected to play an essential role in the future surgical training.

A good surgical simulation system should provide surgeons with visual, tactile and behavioral illusion of reality.1 To meet this requirement, it is crucial to solve the problem in the development of a surgical simulation system to achieve simulation realism with interactive speed. In the previous works, various models have been presented. Among those, the mass-spring model is most widely applied in modeling deformable objects due to its simplicity. Picinbono et al.2 used the mass-spring model to simulate the stiff character of the liver capsule, while Delingette et al.3 expressed the fat tissue elasticity as a network of springs on a 3-simplex mesh. Although these mass-spring models are easy to construct, they lack realism since a discrete set of masses are used in the models to simulate the continuum mechanics. Compared to finite element models (FEM), the range of possible dynamic behaviors of spring models is more constrained in mass-spring model.4 Bro-Nielsen et al.5 discussed real-time simulation of deformable objects using a 3D solid volumetric fast finite element model. In their solution, three optimized methods are used during the computation: firstly, make use of the condensation technique to reduce the computation time by confining the full computation only to the surface nodes of the mesh; secondly, the system matrix is explicitly inverted for the transformation calculation in order to achieve a very low calculation expense; and finally, selective matrix vector multiplication is applied according to the sparse character of the force vector. Their system enables a solid deformation for relatively large

*Correspondence to: Wen Wu, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong.
E-mail: wwu1@cse.cuhk.edu.hk

Copyright © 2004 John Wiley & Sons, Ltd.
Fundamental Theory

In finite element method, the continuum or object is subdivided into elements joined at discrete node points. The solution is subjected to constraints at the node points and the element boundaries so that continuity between elements is achieved.

We chose the four nodes linear tetrahedral element to model the soft tissue in 3D domains. The displacement variation \( \mathbf{u} = (u, v, w) \) within an element can be given by the nodal displacements and the shape function as follows

\[
\mathbf{u} = \begin{bmatrix} \mathbf{I} \mathbf{N}_i, \mathbf{I} \mathbf{N}_j, \mathbf{I} \mathbf{N}_m, \mathbf{I} \mathbf{N}_p \end{bmatrix} \begin{bmatrix} \mathbf{u}_i, \mathbf{u}_j, \mathbf{u}_m, \mathbf{u}_p \end{bmatrix}^T
\]

where \( \mathbf{I} \) is a \( 3 \times 3 \) identity matrix and the shape function \( N_e \) is defined as:

\[
N_e = \frac{a_e + b_e x + c_e y + d_e z}{6V}, \quad e = i, j, m, p
\]

In which \( V \) represents the volume of the element. The other constants are defined by cyclic interchange of the subscripts in the order \( i, j, m, p \).

Linear Elasticity

The state of deformation at a point in the solid is described by the strain tensor, which is the second order tensor. The physical behavior of soft tissue may be considered as linear elastic if its displacement and deformation remain small. The second order term can be neglected. We use a linear strain approximation to represent the tissue deformation,
written as concentrated loads. The work done by these forces can be evaluated body forces, distributed surface forces and concentrated loads acting at the points \( f \) and \( u \) are surface forces applied to the object surface \( dS \), and \( p_i \) are concentrated loads acting at the points \( (x_i, y_i, z_i) \). From (1) and (3), the strain \( \varepsilon \) at arbitrary point in the element can be expressed in terms of the nodal displacements and the shape function

\[
\varepsilon = \mathbf{B} \mathbf{U}
\]

Condensation

Condensation is a process by which some of the degrees of freedom are eliminated from the overall equilibrium equations prior to the continuation of other phases in the solving. In finite element method, it is efficient to employ condensation at a certain stage to eliminate variables that do not directly participate in the rest of the solution.

To establish the equations used in condensation, we rewrite the sparse linear system as the block matrixes form

\[
\begin{bmatrix}
K_{ii} & K_{ih} \\
K_{bi} & K_{bb}
\end{bmatrix}
\begin{bmatrix}
u_i \\
b_i
\end{bmatrix} =
\begin{bmatrix}
f_i \\
b_f
\end{bmatrix}
\]

where the \( i \) represents the condensed degree of freedom (DOF), and the \( b \) is the retained DOF. The upper part of (7) can be solved for \( u_i \) as

\[
u_i = K_{ii}^{-1}f_i - K_{ii}^{-1}K_{ib}b_i
\]

To substitute this equation into the lower part of (7) for \( u_b \), we can obtain the new equations for \( u_b \)

\[
K_{bb}' = K_{bb} - K_{ib}K_{ii}^{-1}K_{ib}
\]

and the reduced force vector \( f_b' \) is given by

\[
f_{b}' = f_{b} - K_{ib}K_{ii}^{-1}f_i
\]

The new stiffness matrix \( K_{bb}' \) in the reduced equation (9) is obviously denser than that in the original one. The size of the equations is just the same as what derived from the surface FEM. It means that, the condensed one is mathematically equivalent to the volumetric FEM, bearing the volume character in the solution but only in the equal computation expense of the surface FEM solution.
Structure of Hybrid Condensed Finite Element Model

In surgeries, most operations are concentrated on the local pathologic area of organs. In these regions, tissues undergo cutting or tearing, thus more precise models are required. We assume the non-linear deformation or the topology changes only occur in the operation part throughout the whole surgery. With this assumption, the key idea of the hybrid FEM algorithm is to separate the areas containing the non-linear elastic properties or topology changes from the others. It includes:

- Operation part where the surgery occurs. It corresponds to the pathological structures and only represents a small part of the whole model in a simulator. In this part, tearing and cutting are simulated.
- Non-operation part where only the deformation occurs. But it greatly contributes to the realism of the simulation.

Different models treat areas with different properties respectively to optimize the trade-off between the computation time and the realism of the simulation.

The illustration of the hybrid model is shown in Figure 1. Since these two parts connect with the common nodes, additional boundary conditions are added to both models. We rewrite the linear system equations of the operation area (12) and non-operation area (13) as a block matrix form respectively:

\[
\begin{bmatrix}
K_{11} & K_{12} \\
K_{12}^T & K_{22}
\end{bmatrix}
\begin{bmatrix}
A_1 \\
A_2
\end{bmatrix} =
\begin{bmatrix}
P_1 \\
P_2
\end{bmatrix}
\quad(12)
\]

\[
\begin{bmatrix}
K_{11}^1 & K_{11}^2 \\
K_{12}^1 & K_{22}^2
\end{bmatrix}
\begin{bmatrix}
A_1 \\
A_2
\end{bmatrix} =
\begin{bmatrix}
P_1 \\
P_2
\end{bmatrix}
\quad(13)
\]

where the subscript 1, 2 and 1 represent the operation area, non-operation area and the common nodes shared with these two areas respectively, the superscript 1, 2 denote the common nodes represented in the operation area and the non-operation area respectively. \(P_1\) and \(-P_1\) is the force and counterforce respectively applied to the common nodes when we analyse these two parts. From (13) we obtain:

\[
(K_{II}^2 - K_{12} \cdot K_{22}^{-1} \cdot K_{2I}) \cdot A_I = P_I - K_{12} \cdot K_{22}^{-1} \cdot P_2
\quad(14)
\]

Representing (14) in another form:

\[
P_I = K' \cdot A_I + P'
\quad(15)
\]

where

\[
K' = K_{II}^2 - K_{12} \cdot K_{22}^{-1} \cdot K_{2I}, \quad P' = K_{12} \cdot K_{22}^{-1} \cdot P_2
\quad(16)
\]

Substituting (15) into (12), we can get a new matrix system (17), from which the displacements of the operation area \(A_1\) and \(A_I\) can be obtained.

\[
\begin{bmatrix}
K_{11} & K_{1I} \\
K_{1I} & K_{II} + K'
\end{bmatrix}
\begin{bmatrix}
A_1 \\
A_I
\end{bmatrix} =
\begin{bmatrix}
P_1 \\
-P'
\end{bmatrix}
\quad(17)
\]

The next step is to obtain the deformation of the non-operation area. As the inside nodes of the non-operation area are unrelated with any action of the surgeon even invisible, we can just treat them as redundant nodes for the simulation. Computing these nodes only brings meaningless burden of calculation. We adopt condensation to remove them from the computation process. This keeps the original physical character of the volumetric system but the size of the condensed matrix equation is equal to the FE surface model. Therefore, we can rewrite the non-operation area equation in the condensed form as

\[
\begin{bmatrix}
K_{II}^2 - K_{12} \cdot K_{22}^{-1} \\
K_{1I} - K_{II} \cdot K_{22}^{-1}
\end{bmatrix}
\begin{bmatrix}
A_I \\
A_i
\end{bmatrix} =
\begin{bmatrix}
P_I \\
P_s
\end{bmatrix}
\quad(18)
\]

The subscript \(i\) represents the internal nodes to be condensed out and \(s\) represents the surface nodes to be retained. From (18), we have:

\[
(K_{ss} - K_{si} \cdot K_{ii}^{-1} \cdot K_{is}) \cdot A_i = P_i - K_{si} \cdot K_{ii}^{-1} \cdot P_i \\
+ (K_{si} \cdot K_{ii}^{-1} \cdot K_{il} - K_{sl}) \cdot A_I
\quad(19)
\]

Because the internal nodes haven’t been acted by the external force, the term of \(K_{si} \cdot K_{ii}^{-1} \cdot P_i\) is equal to zero. Then we can derive the new matrix equation \(K' A_i = P'\) which only relates to the variables of the surface nodes.

\[
K' = K_{ss} - K_{si} \cdot K_{ii}^{-1} \cdot K_{is}, \quad P' = P_s + (K_{si} \cdot K_{ii}^{-1} \cdot K_{il} - K_{sl}) \cdot A_I
\quad(20)
\]
It should be noticed that, the form of $P^*$ is different from the one derived from the condensation used in general FEM. It has one term that relates to the common nodal displacements $A_I$ in the hybrid FEM.

All terms in $K$, $P$, $K'$ and $P'$ are known constants and keep unchanged in the whole surgery simulation process because no topology change will occur in the non-operation part. Thus they are calculated in the preprocessing stage. This stiffness matrix in (17) will be updated according to the topology changed during the simulation. However, it is clear that the order of this system is far less than the order of the global FEM system.

In summary, the interactive update and display of the hybrid model is achieved through the processing on the operation area. The model of this area is updated based on the imposed displacement or forces on the boundary. During this stage, the displacements of the nodes and forces applied on the common nodes are computed. Then we can obtain the displacements of nodes on the surface of the non-operation model according to displacements of the common nodes.

**Calculation onto the GPU**

As mentioned above, the interaction occurs in the operation area, where the collision of the scalpel and the tissue model is detected and the cutting operation is processed. The topology of the model is changing on-the-fly during the simulation. The corresponding equation (17) is constructed and solved in every different processing step. Along with the size of the model increasing, the time spent by the collision detection rises as a linear function of the model scale. And the time of constructing the new global stiffness matrix is squared. Solving the linear system is about cubed complexity, which takes almost seventy to eighty percent of the time executed by the operation area. Therefore, solving the linear algebraic equations is the bottleneck of the whole simulation. It directly impacts the feedback speed of the system. In our system, we use the Conjugate Gradient iteration to solve the linear equations. The basic principle is to keep the condition number of the matrix as small as possible.

In this section, we introduce our method of mapping the calculation kernel onto the GPU. The Conjugate Gradient solver includes three primary operations: multiplication of the sparse matrix and the vector, addition of two vectors and the sum-reduction operation. The core component is the multiplication of the sparse matrix and the vector. We migrate this part of calculation from the CPU into the fragment processor of the GPU, to take advantage of the fragment processor in its efficiently manipulation of the local texture memory on the mathematical calculation. The basic principle is to load matrices and vectors as textures into the GPU, and then rasterize a proper quad of pixels for invoking the fragment program, which process the actual calculation in the fragment processor. The result can be obtained as the color value, transferred directly to the next pass for execution or readback to the CPU.

**Texture Representation of Matrices**

For the sparse matrix-vector multiplication: $y_i = \sum_{j=0}^{n} a_{ij} \cdot x_j$, the matrix representation in the GPU should consider both the system data structure and the GPU single-instruction-multiple-data (SIMD) character.

Since the matrix is sparse and symmetric, we use an one-dimensional array $M$ to store the upper-right non-zero elements of upper-right of the matrix in our system. Two accessorials arrays are needed to restore the original matrix (Figure 2), $N_1$ for the indices of the diagonal elements in $M$ and $N_2$ for the column index of every non-zero element.

Various mapping methods have been proposed in previous works \(^1^1,^1^2,^1^7\). In our solution, an improvement is made to the method from \(^1^2\). The diagonal and off-diagonal elements of the matrix are stored separately by different layout textures. Two dependent textures are used to help fetch off-diagonal elements and proper vector elements efficiently in the fragment program computation. The entire process of multiplication for the off-diagonal elements is as illustrated in Figure 3.

**Figure 2. The data structure of the sparse matrix.**

\[
M = [\star, \star, \ldots, \star]
\]
\[
N_1 = [0, 3, 5, 7, 8]
\]
\[
N_2 = [0, 2, 3, 1, \ldots, 4]
\]
Ty is the texture keeping the multiplication result, and Tx has the same layout, which stores the vector x. Texec element keeps all the non-zero (off-diagonal) elements of the matrix row by row and the index of every row-starting point in Texec element can be found by Texec index. The texture coordinates of the corresponding elements in vector x are stored in Texec st. In 12, rows were divided into multiple groups in preprocessing, and each group related with a certain size of the texture has the equal number of non-zero elements. The specified fragment program processes each group. In the interactive simulation, we should try to avoid such trivial work and construct textures directly. We add one more content to Texec index keeping the total number of non-zero elements in each row, which is inspired from Buck’s presentation. Accordingly, in the fragment program, we use two instructions to check and handle the case where the pass number of processing is greater than the number of non-zero elements in every row. To take full advantage of the 4-channels type of texture element, we rearrange the content of these textures with two different designs.

**Method A**

Textures which have the same layout can be integrated into one texture. As described above, using RGB channels of textures, Texec element ⊕ Texec st and Texec diagonal ⊕ Texec index can be combined into one respectively (Figure 4). This compact representation not only saves the texture memory, but also reduces the number of instructions on texture fetches. The pass number executed by the fragment program is decided by the maximum number of non-zero elements of the sparse matrix rows.

**Method B**

In this method, we use all RGBA channels of Texec element to keep the non-zero (off-diagonal) elements. Because one texture coordinates can decide four elements at one time, if the number of non-zero elements isn’t a fourfold number, it will be padded with zeros to let all channels filled. Texec st should provide all four coordinates of the corresponding elements of x in Texec x at one time. As the original one texture can’t fulfill the requirement, we use two textures Texec st-a and Texec st-b instead of Texec st to keep this information (Figure 5).

This strategy has more compact style of non-zero elements representation. The fragment program is more complex and a little bit longer. It has 13 more instructions compared with last one, 5 of which are texture fetches. The fragment program executes four multiplications at one time. The current pass result needs be accumulated with that from the previous pass kept in a one channel texture. Therefore four more instructions are used to accumulate these four components before executing the accumulation between two consecutive passes.

**Implementation and Experiments**

We have implemented all these algorithms by VC++. In the pre-processing stage, we build an initial FEM with both geometric shapes and physical parameters of tissues. The geometric model of the patient organ was generated based on CT/MRI images. The physical characters include the boundary conditions of the model such as the displacement constraints and the external forces, the initial strains and some material parameters.
After marking the surgery area interactively on the mesh to determine the operation part, the non-operation part and the interface between the two parts, the sub-matrices \( K^0, P^0, K/C3 \) and \( P/C3 \), which remain constant throughout the simulation can be calculated in advance. The data we employed comes from the Visible Human Project. 19 We created a volume tetrahedral mesh of a short portion of the upper leg (Figure 6) for the experiment. The thighbone is not shown in the figures for clarity, thus leaving a hole in the model. The nodes fixed on the thighbone have no displacements. Three nodes on the surface of the operation area are imposed with certain forces to simulate the stresses caused by other organs or forces applied by surgical tools. These forces can also cause the initial deformation of the soft tissue. When the scalpel collides with the tissue and the press is big enough, the cutting operation occurs (see Figure 7). Figure 8 shows the deformation of the tissue during the cutting process. For the model of which the non-operation area has 633 nodes (416 surface nodes) and 2134 tetrahedrons, the average execution time for obtaining the displacement of the non-operation with condensed model is almost four times faster than that of non-condensed model (Table 1).

Table 2 compares the time of the sparse matrix and vector multiplication, which run in Conjugate Gradient solver of our system, on the GPU and the CPU respectively with different degrees of freedom of the operation area. It’s the time of one multiplication on NVIDIA GeForceFX 5950 Ultra with 5.2.1.6 driver, 1.5 GHz Pentium 4 CPU. The size of the linear equation derived from the operation area model is from 456 to 2190. The maximum number of non-zero elements per row decides the pass number of the fragment program, and at each pass the result are added into that in the previous pass. Because method B makes full use of the 4-channels character to compactly fill non-zero elements and coordinates indices into the texture, the pass number required can be remarkably reduced so that time spent on the data transfer between two consecutive passes can be saved. That makes method B more efficient than method A. Along with the increase of the data scale processed, the performance of the method B on the GPU can reach 2.15 times faster than the CPU.
implementation. Even for the method A on the GPU, an improvement rate of 1.14 times can be achieved. It proves that GPU implementation shows predominance in processing large scale data due to its high degree of parallelism, of which the CPU is lacking.

### Conclusion and Discussion

In this paper, we proposed a hybrid condensed finite element model with the GPU hardware acceleration for surgical simulation of tissue cutting and deformation. The model processes the tissue as two different types. For the first type of the tissue which is not involved with any cutting operation, the deformation is calculated only to the surface nodes, by a condensation technique, so that no computation is required to a large number of interior nodes. For the second type of the tissue involved with the cutting operations, more comprehensive process is applied. In our solution, as the new topology constructs on-the-fly during the cutting procedure, the stiffness matrix is refreshed to build the new linear equations for the nodal displacements. Conjugate Gradient iteration is adopted to solve the problem. To improve the efficiency, we migrate the most computational intensive part of the calculation onto the GPU for further acceleration. Two methods of mapping data onto the GPU have been proposed in accordance with the application data structure, and as a result, the simulation can reach an interactive frame rate, even with topology change to the model in a certain degree of complexity.

As a future work, the force applied by surgical instruments shouldn’t be restricted in the operation area. Some other physical characters can be added to the model to enhance the realism. On the other hand, current methods of mapping the calculation on the GPU are sensitive to the bandwidth of the sparse matrix. As long as one row has a high bandwidth, a high number of passes would be required so that the cost could be sharply increased. In fact, we find that most of rows have non-zero elements less than twenty. A direct method to solve this problem is to optimize the mesh prior to the simulation to make the incident edges for each node fairly uniform in number. Another possible solution is to investigate a better layout of texture for storing the data.

**Table 1. Time of two models for 633 system nodes in the non-operation area (Pentium4, 1.5 GHz)**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>456</td>
<td>188</td>
<td>47</td>
<td>1.913</td>
<td>1.118</td>
<td>0.622</td>
</tr>
<tr>
<td>960</td>
<td>188</td>
<td>47</td>
<td>1.808</td>
<td>1.023</td>
<td>1.116</td>
</tr>
<tr>
<td>1137</td>
<td>188</td>
<td>47</td>
<td>1.811</td>
<td>1.027</td>
<td>1.500</td>
</tr>
<tr>
<td>1602</td>
<td>188</td>
<td>47</td>
<td>1.837</td>
<td>1.056</td>
<td>1.731</td>
</tr>
<tr>
<td>1854</td>
<td>190</td>
<td>48</td>
<td>1.846</td>
<td>1.057</td>
<td>2.288</td>
</tr>
<tr>
<td>2100</td>
<td>231</td>
<td>58</td>
<td>2.106</td>
<td>1.140</td>
<td>2.323</td>
</tr>
<tr>
<td>2190</td>
<td>237</td>
<td>60</td>
<td>2.196</td>
<td>1.161</td>
<td>2.503</td>
</tr>
</tbody>
</table>

**Table 2. Time of the sparse matrix-vector multiplication, which runs in the Conjugate Gradient solver of our system on the GPU and CPU respectively, with different degrees of freedom of the operation area.**

It is the time of one multiplication on NVIDIA GeForceFX 5950 Ultra with 5.2.1.6 driver, 1.5 GHz Pentium 4 CPU. (Source: Time : ms)
ACKNOWLEDGEMENTS

This work was supported by the Research Grants Council of the Hong Kong Special Administrative Region under Central Allocation Grant (project no. CUHK1/00C) and Earmarked Research Grant (project no. 4356/02E).

References