A New Framework for Behavior Modeling of Organs and Soft Tissue using the Boundary-Element Methods

Dan Koppel[†*], Shiv Chandrasekaran[‡], and Yuan-Fang Wang[†]
[†] Department of Computer Science
[‡] Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA, USA

Abstract

In this paper, a method for modeling the deformation behavior of organs and soft tissue is presented. The purpose is to predict the global deformation effect that arbitrary, time-varying external perturbations have on an organ. The perturbation might be caused by an instrument (e.g., through the surgeon's grasping and pinching actions), or it might be from organ-organ, organ-body-wall collisions in a bodily cavity. A methodology, employing (1) a surface representation based on the Boundary-Element Method—or BEM, of the deformation equations and (2) recently developed linear-algebra techniques (known as the "Hierarchical Semi-Separable" matrix representation—or HSS), is proposed. We demonstrate that the proposed framework achieves an almost linear time complexity of $O(n^{1.14})$, a significant speed up comparing to the traditional $O(n^3)$ schemes employing brute-force linear-algebra solution methods based on Finite-Element Method (FEM) formulations. Furthermore, unlike some previous approach, no restriction is placed on the external perturbation pattern and how it can change over time.

1. INTRODUCTION

A computer organ model that captures faithfully the *appearance*, *structure*, and *behavior* traits of an organ can be of a significant clinical value, such as in surgical planning, surgery simulation, and surgeon training.

We seek an approach that *is* physically-based, like the finite-element methods (FEM), but achieves a speed comparable to that of the efficient kinematic methods [8, 5, 4, 10], popular in the computer graphics and animation industry. The research makes the following contributions:

- Proposing a methodology employing a surface representation based on the boundary-element methods (BEM) [7] that is physically correct but results in a problem size significantly smaller than FEM-based formulations ($O(n^2)$ for BEM vs. $O(n^3)$ for FEM, where n is the resolution along an axis),
- Incorporating novel linear-algebra techniques (known as the "Hierarchical Semi-Separable" matrix representation—or HSS) that achieves almost linear time complexity of $O(n^{1.14})$ in the solution process, a significant speed up comparing to the traditional $O(n^3)$ schemes for FEM, and
- Relaxing the unrealistic "point-like" disturbance model used in some previous approach [6], and placing no restriction on the external perturbation pattern and how it can change over time.

2. ORGAN DEFORMATION MODELING

As mentioned, the BEM employs a surface representation for improved speed over the FEM. Certain assumptions enable such a faster method, and the most important is that the material is homogeneous and isotropic, which is exploited by the math to yield surface rather than volume integrals. Furthermore we have assumed that the movement of instruments is slow relative to propagation of transverse/longitudinal waves within the medium of the organ tissue. These considerations together allow: (1) highly simplified equations relating displacements to forces and (2) statics assumption (the configuration of the system is a function of the forces applied at that moment and not of the previous history).

BEM-based deformation starts with the well know Navier's equation [7]:

$$\mu\left(\frac{\partial^2 u_i}{\partial x_k \partial x_k} + \frac{1}{1-2\nu}\frac{\partial^2 u_k}{\partial x_i \partial x_k}\right) + f_i = 0 \qquad (1)$$

where $1 \le i,k \le 3$. $\mathbf{u} = (u_1, u_2, u_3)$ represents a displacement field.[1] When the object is not subjected to any

[*]The author's current address: STI Medical Systems, 733 Bishop Street, Suite 3100, Honolulu, HI 96813.

[1]We use bold-face characters to denote vectors and matrices and plain-face characters to denote scalars. Furthermore, we follow the standard tensor summation convention, where repeated indices are implicitly summed over their range.

outside forces (f), $\mathbf{u} = \mathbf{0}$ everywhere in the object. However, when the object is deformed by an outside influence, $\mathbf{u}$ captures the amount of movement or deformation from the resting state. λ and μ are known as the Lamé constants, which completely characterize isotropic materials.

Somigliana's equation is the basis of the BEM for elasticity. The useful thing about this equation is that it involves the deformation and traction vectors located on the surface of the body only. In other words, it is not necessary to know what is happening in the interior of the body. In effect, it is a way of "integrating by parts" Navier's equation. Due to the space limit, we will skip the derivation here and provide the final expression (detail can be found in [7]):

$$\int t_i u_{ik} d\Gamma = \int t_{ik} u_i d\Gamma + c u_k(\mathbf{d}) \tag{2}$$

where t = external force/area applied at the surface, Γ the surface, and Ω the interior volume. c is 0 for $\mathbf{d} \notin \Omega$, $\frac{1}{2}$ for $\mathbf{d} \in \Gamma$, and 1 for $\mathbf{d} \in \Omega$ [7]. Somigliana's equation is an equation involving surface variables only. Since these variables are the ones that define the shape of the object as seen from outside, this situation is ideal. Furthermore, this equation is *linear* in the unknowns. What remains is to take this surface integral equation, discretize it, and solve it.

2.1. Discretization

Similar to the FEM, we start by triangulating the surface. Our deformation and traction "control-points" are placed in the center of the triangles and we use a zeroth-order approximation to compute the deformations and tractions away from the control-points. Thus, we take Somigliana's equation and break down the surface integral into a sum of integrals, each evaluated within a specific triangle. Since the values of the variables are assumed to be constant within the triangle, these can be factored out, leaving an integral which can be computed offline. After all these integrals are computed (this can be done in closed form), we are left with an equation of the following form:

$$\mathbf{Hu} = \mathbf{Gt} \tag{3}$$

where $\mathbf{u}$ is $3n \times 1$ vector representing all the deformation displacements (at the center of each triangle) and $\mathbf{t}$ is also a $3n \times 1$ vector representing the tractions at the centers of the triangles. Both $\mathbf{H}$ and $\mathbf{G}$ are square matrices. Thus, this is a system of 3n equations with 6n unknowns. We have chosen to formulate the boundary condition as a "mixed" one

$$\mathbf{K}(\mathbf{u} - \mathbf{a}) + \mathbf{t} = 0 \tag{4}$$

where $\mathbf{K}$ is a 3×3 matrix representing the local spring constant of the external interaction (for simplicity, this can be assumed to be equal to $k\mathbf{I}$, where $\mathbf{I}$ is the 3×3 identity matrix) and $\mathbf{t}$ is the traction applied by the external object to the object of interest follows Hooke's law, where the position of repose for the external object is the vector $\mathbf{a}$. Now, to solve for $\mathbf{u}$, we can combine Eqs. 3 and 4 to eliminate $\mathbf{t}$ and obtain

$$(\mathbf{G}^{-1}\mathbf{H} + \mathbf{K})\mathbf{u} = \mathbf{Ka} \tag{5}$$

This is now in the form of Ax = b, where $\mathbf{A} = \mathbf{G}^{-1}\mathbf{H}$ is computed offline and updated online only by adding a diagonal matrix $\mathbf{K}$. The HSS algorithm described below can handle these updates efficiently.

3. Numerical Schemes

Efficient numerical schemes for solving Eq. 5 is important because by itself the BEM is not sufficient to yield fast results for real-world applications with dynamically-changing boundary conditions. Others using this method have made specific assumptions about the external perturbation being "point-like" (e.g., the external disturbance is a poking action by a sharp instrument making a point contact with the organ surface) in order to achieve a higher speed [6]. While this is of value, our premise is that this excessively limits the generality of the problem to be solved. The answer we have found is to allow the perturbation to retain its generality, but achieve speedup by (1) employing iterative linear solvers, aided by (2) the use of pre-conditioners based on a recently developed linear-algebra approximation method called "Hierarchically Semi-separable Representation" (HSS) [3, 2].

3.1. Iterative Solvers

Equations of the form **Ax = b** can be solved either through direct or iterative techniques. The latter refers to algorithms that proceed in repetitive stages such that the estimated solution consistently improves at each stage. This approach has been studied extensively in the literature [9] and is the method of interest to us here because

- To the best of our knowledge, no one seems to claim a numerical scheme for inverting large, non-symmetric, and dense matrices (as those in BEM, Eq. 5) in real time.
- Because the solution is being progressively refined in an iterative scheme, processing can be terminated prematurely and still yield a valuable approximate answer. Time constraint can thus be managed by providing modeling results with varying degrees of accuracy and detail.

The main working principle for iterative solvers is the concept of Krylov sequences [9]. For the case of $\mathbf{Ax} = \mathbf{b}$, a discrete set of vectors, consisting of $\{\mathbf{b}, \mathbf{Ab}, \mathbf{AAb}, \cdots, \mathbf{A}^{(m-1)}\mathbf{b}, \cdots\}$, can be generated. Assuming $\mathbf{A}$ to be non-singular, the entire set of these vectors will span the solution space. In essence, the idea of an iterative solver is to search in progressively larger subspaces for the best solution $\mathbf{x}$ (in the least squares sense). Each step utilizes the last solution as a starting point to refine $\mathbf{x}$. The next subspace is obtained from the previous one by adding the next vector in the sequence of the Krylov vectors. There are several iterative

methods for solving non-symmetric linear systems (such as CGNR, BiCG, CGS, BiCGSTAB [9]). We concentrate on a commonly used algorithm known as "Generalized Minimal Residual Method" (GMRES).

3.2. Pre-Conditioning

Terminating numerical iteration before reaching the full n count ensures efficiency at the expense of accuracy. A logic question to ask is if it is possible at all to obtain both high efficiency and high accuracy? The answer lies in pre-conditioning. Pre-conditioning refers to implicitly multiplying the equation **Ax = b** by the inverse of a matrix **P** (which closely resembles **A**, but for which a linear system can be solved more efficiently than for **A**) with the purpose of creating a well-conditioned system requiring fewer iterations to converge. A useful guide to pre-conditioning can be found in [9]. Specifically, the introduction of the pre-conditioner **P** causes the following changes in the linear system: $\mathbf{b} \Rightarrow \mathbf{P}^{-1}\mathbf{b}$ and $\mathbf{Ax} \Rightarrow \mathbf{P}^{-1}\mathbf{Ax}$. (It must be pointed out that $\mathbf{P}^{-1}$ is not computed explicitly but implicitly by solving an equation of the form $\mathbf{P}\tilde{\mathbf{x}} = \mathbf{x}$.)

4. HSS method

While many "generic" pre-conditioners (e.g., ILU0 [9]) exist, they do not take advantage of the particular matrix structure induced by the physical interaction pattern in deformation. Hence, we have chosen to use a pre-conditioner that captures such physical interaction patterns in GMRES. This is called the "Hierarchically Semi-separable Representation". In this section we provide an intuitive explanation of the HSS. The exact mathematical formulation and the detail of how the HSS representation can be used for speeding up linear-system-solve and matrix-vector-multiplication (the most important steps used in iterative solvers) can be found in [3, 2].

The HSS representation exploits the fact that matrices originating from physical systems tend to have some structure and more specifically tend to have off-diagonal block elements which, to good approximation, are not full-rank. The description "to good approximation" needs some clarification. Given some matrix **B**, if an SVD is performed on **B** and the trailing singular values are below a small threshold, we can force these values to exactly zero with little change in the resulting matrix. This process is analogous to lossy data compression. In the same manner that a jpeg image can be for most purposes virtually indistinguishable from the original and yet achieve a significant file size reduction, a matrix can also be "data compressed".

To simplify the following discussion, we will assume that the matrix size N is a power of 2 (although this is not a requirement for HSS to function correctly). And we will assume that we have a block partitioning of the matrix, both for the rows and columns, and furthermore that the number

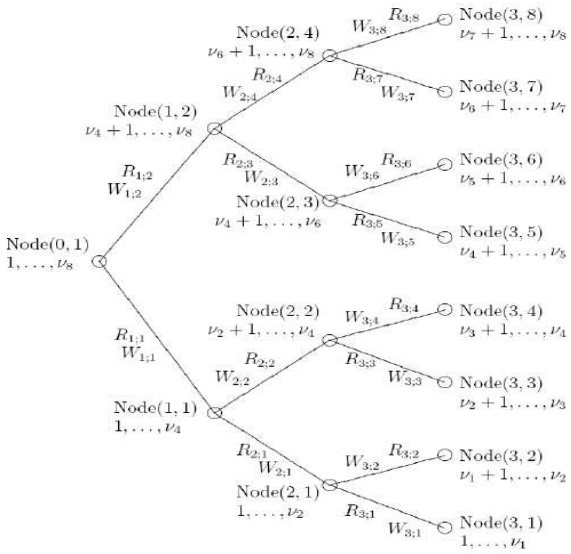

Figure 1. Merge tree with three resolution levels, representing an 8x8 block matrix. (The index preceding the semi-colon refers to the resolution level.)

of partitions is a power of 2 and is the same for the rows as for the columns. We additionally assume that the matrix is physically-based and that the ordering of the physical locations is done through nested dissection. [2]

The basis of the HSS matrix representation is the realization that any matrix can be described by a binary tree graph (known as a "merge tree" and shown in Fig. 1) where the leaves hold two sets of basis vectors spanning the matrix's block rows and columns respectively (referred to as **U** and **V**, but not shown on the figure) and where the edges hold matrices that connect the basis vectors for a finer block resolution with those for a coarser resolution (**R** and **W** in the figure). Consider a block row of a matrix for which the block element lying on the matrix's diagonal has been removed. This is called a "Hankel" block and is a collection of many column vectors (since it's a *block* row) and this collection can be described by an orthonormal set of basis column-vectors. We illustrate this with an example (Fig. 2(a)) where we have an 8×8 block matrix, and each block element is also 8×8. Hence, a Hankel block row is $[1 \times 8] \times [(8-1) \times 8] = 8 \times 56$, as shown in Fig. 2(a).

If the matrix had no particular meaning (e.g., generated by a random-number generator), then most likely the number of basis vectors required to express this block row would

[2] Nested dissection maps a set of discrete 2D points to a 1D ordering so that we can use it to express the indices contained in vectors and matrices. In essence, for every entry in a matrix, we express the row and column numbers in base 2 and then interlace the digits of the row and column numbers from the most significant bit to the least significant bit. Every entry now has a single number associated with it. This number can now be used to order the matrix entries.

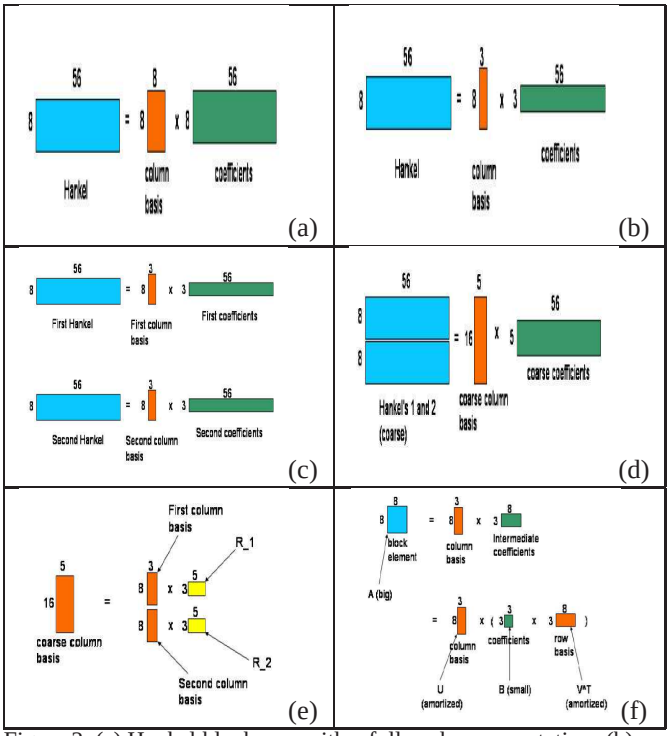

Figure 2. (a) Hankel block row with a full-rank representation, (b) Hankel block row with a reduced-rank representation, (c) Two adjacent fine resolution block rows with low-rank representation, (d) Combined coarse resolution block row, (e) The two sets of fine resolution basis vectors are "spliced" together (after being "massaged") to yield one set of coarse resolution basis vectors, and (f) A block element's reduced "descriptor set" obtained using column and row bases.

be equal to the length of each column-vector (assuming that the block row has an actual width that is greater than its actual height). However, for a block row that is part of a physically-generated matrix, the column vectors will be correlated to each other so that they do not maximally span the allowable space, but rather reside on some subspace. In essence, this is similar to the concept from Principle Component Analysis (PCA), which looks at the space spanned by a set of vectors and ignores the directions which are either not traversed or are only slightly traversed. Fig. 2(b) shows this reduced-rank representation for the same Hankel block row in Fig. 2(a). And this (reduced) set of basis vectors is then stored in the binary tree graph (see Fig. 1) at the leaf which corresponds to this block row.

As stated above (and displayed in Fig. 1 as **R**'s), the edges store matrices that "connect" the basis vectors at one resolution with those at a different resolution [3]. We now explain the meaning of this statement. At the finest resolution we have a set of block rows. However, we can combine pairs of adjacent rows so that the total matrix will have half as many block rows, but each block row will be twice as high. This represents a coarser resolution. In terms of the tree graph, this corresponds to the level immediately above the leaves and of course there will be half as many nodes at this level than at the bottom level. The question now is the following. If one were to repeat the operation described above for obtaining a (reduced) set of basis vectors, but did it using these "coarser" (i.e., larger) blocks, could we somehow "connect" this result with the previously obtained "finer resolution" results? The answer is yes and is illustrated graphically in Figs. 2(c) and (d). Fig. 2(d) shows two block rows in Fig. 2(c) can be combined into one coarse resolution block row, also having a low-rank representation.

In this example, the rank of the fine resolution block rows is 3 and the rank of the coarse resolution block row is 5. There is no fixed and simple relationship between the ranks corresponding to different resolution levels. However, it is intuitive that as the block rows acquire longer column vectors, the rank needed to adequately describe them should either stay the same or increase. Continuing, we can relate the two sets of basis vectors from the fine resolution level with the set of basis vectors from the coarse resolution level. This is shown in Fig. 2(e). The matrices **R**, known as *translation operators*, provide the "connection" between the basis vectors for fine-resolution block rows and coarse-resolution block rows. Notice that this operation resembles a "splicing" (but performed only after "massaging" the smaller "splices" with the connector **R**'s). This process can be repeated recursively to create increasingly "coarser" representations for both block rows and block columns.

A given off-diagonal block element is part of some block row as well as some block column. Using the (reduced) basis column-vectors ($\mathbf{U}_i$) associated with that particular block row and the (reduced) basis row-vectors ($\mathbf{V}_j$) associated with that particular block column, we can express the block element as a set of (reduced) coefficients as shown in Fig. 2(f) (using a 8x8 block element as an example).

$$\mathbf{A}_{ij} = \mathbf{U}_i \mathbf{B}_{ij} \mathbf{V}_j^T \tag{6}$$

where $\mathbf{B}_{ij}$ is a small square matrix comprising a compact description of $\mathbf{A}_{ij}$. Since the same **U** is shared (i.e., amortized) by all block elements belonging to the same block row and the same **V** is shared by all block elements belonging to the same block column *and* the **B**'s are smaller than the **A**'s, "data compression" has taken place.

Now we arrive at our key point. At the finest resolution, these sets of reduced coefficients are not computed for *all* off-diagonal block elements but only for those that represent pairs of points nearby in physical space. For block elements which represent pairs of physical points not in close proximity, it is possible to closely approximate their interaction as part of a coarser level interaction between two *neighborhoods* of points, to which each of the two points belongs.

[3] The **W**'s in the figure serve the same function as the **R**'s except that they operate in the "dual" mode, by which we mean that all references to "row" and "column" are interchanged. I.e., "row" $\Leftrightarrow$ "column".

To make an analogy, when we compute the force between the earth and the moon, we do not concern ourselves with the specific structure of the either body (e.g., shapes of mountain ranges or craters), but rather treat each body as an idealized point mass. The similarity in spirit with the Fast Multipole Method (FMM) [1] is also evident: the influence of nearby objects should be described individually, while the influence of far-away objects should be treated collectively as arising from groups of objects. Doing so will achieve greater space and time efficiency with little loss of accuracy for physically-generated matrices.

Finally, the effectiveness of the HSS and the associated operations in solving dense, non-symmetrical linear systems **Ax=b** will be reported in the next section on experimental results. The theoretical detail of these operations can be found elsewhere [3, 2].

5. Experimental Results

We have conducted many experiments to understand the accuracy and efficiency of the proposed modeling framework. Due to page limit, we present here only timing and accuracy results obtained by running the algorithm presented in the previous sections. The simulations were done on a Macintosh Power-PC (2.1 GHz G5 with 1.5 GB RAM running OSX 10.4.8) and the programming language used was OCaml (a functional programming language: for more info see http://caml.inria.fr). The HSS-library is a mixture of OCaml and low-level assembly.

Online time analysis (Fig. 3) Here we discuss one of the most important results of the work. Normally, solving a system of type Ax = b has a time-complexity of $O(n^3)$ for a problem size of n. We present data from deformation modeling showing that the time-complexity of the algorithm we have elaborated above grows as $O(n^{1.14})$, which is a dramatic improvement in asymptotic behavior. *These results illustrate the effectiveness of using the HSS matrix representation and associated algorithms for deformation modeling.*

We illustrate this point by showing in Fig. 3 plots of the online time as a function of many important system parameters. Fig. 3(a) presents the speed-accuracy trade-off "envelope". By using cases with known ground-truth, we set the parameters so that online timing is optimized, subject to the condition that the computed solution deviate from ground-truth by at most the specified amount. This yields a relationship between time and error, which is shown in Fig. 3(a) for the case of problem size $n = 1056$.

In Fig. 3(b) we display the online timings as a function of problem size, for a fixed relative accuracy = .1. We also show the best fitting power law involving the above exponent ($t = 1.43 \cdot 10^{-5} n^{1.14}$ compared to $t = 2.07 \cdot 10^{-10} n^3$ when using LAPACK's optimized linear-system solver "dgesv": the "break-even" point is at n = 400).

In Fig. 3(c)-(f), we display the effects on online timing due to changing the characteristics of the body and of the perturbing influence. The body's properties are its (1) shape and (2) constitutive properties, while the perturbation's main property is (3) its "width" (i.e., how much contact it makes with the body). We show that the modeling algorithm is not very sensitive to these variations.

For (1), we obtain timing results for a set of ellipsoids of varying eccentricities, ranging from oblate to prolate. Fig. 3(c) shows the results. In addition, we obtain timings for a prolate ellipsoid (major/minor = 4) which is subjected to a constant curvature along the major-axis, yielding a crescent-like shape. Fig. 3(d) shows online timing results for different values of curvature.

For (2), we measure the effect that modulating the Poisson ratio has on timing. Fig. 3(e) shows this relationship. It can be seen that the Poisson ratio has little effect on timing except near .5, where the material becomes incompressible. At this point, the timing values diminish abruptly as this value of the Poisson ratio is the theoretical upper limit.

The object's geometry impacts the online statistics over a larger range of values. One possible explanation for this effect is that, as an object acquires more eccentricity in its shape, different dimensions have different characteristic lengths. Since nested dissection treats each dimension equivalently, it may be ignoring an object's favoring of a certain direction (e.g., a cigar). In this situation, we suspect that the Hankel blocks may not exhibit as low a rank as they would otherwise.

Finally, for (3), we consider the possible effect that the width of the perturbation might have on timing results. A small width corresponds to a "point-like" poke, whereas a large width corresponds to a large fraction of the body's surface being impinged on from outside. Fig. 3(f) shows these results. Increasing the perturbation "width" does not seem to incur any additional cost to the online timing. This suggests an advantage over methods employing the "point-like poke" assumption [6].

Graphical results Figs. 4 and 5, and Supplementary Submission Here we present some graphical results based on BEM modeling. Naturally, simulation speed information cannot be conveyed here. But the realism of simulating deformations through the BEM method can be made somewhat apparent. In Fig. 4 we display a virtual organ (elongated and curved) under the influence of an external perturbation. The depth and width of the perturbation is constant in time but its point of application moves azimuthally. In Fig. 5 we show the effects of a plane that descends and impacts the organ from above. The plane is of infinite extent and its normal deviates slightly from the z-direction. At each frame the object becomes more deformed.

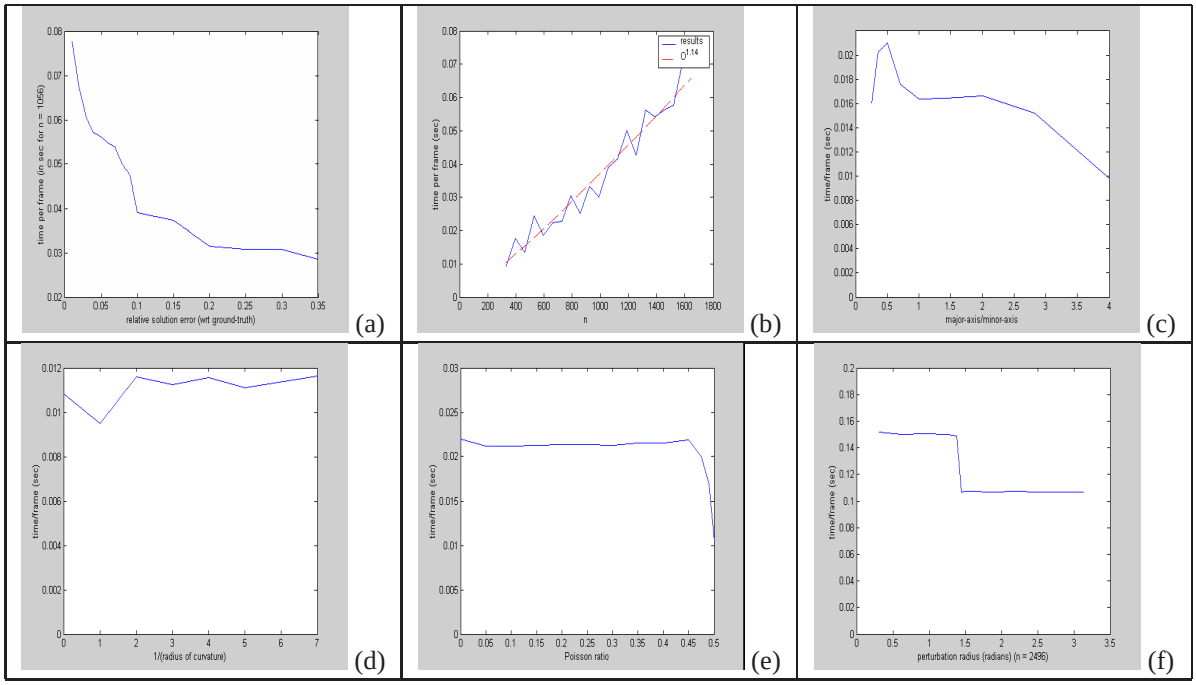

Figure 3. Online runtime a function of (a) accuracy (n = 1056), (b) problem size. The best-fitting power-law curve has an exponent of 1.14 (normal matrix inversion has exponent of 3), (c) ellipsoid eccentricity (problem size = 432 and specified error = .10), (d) ellipsoid with major and minor axes = 4 and 1, respectively: a crescent-like shape (problem size = 432 and specified error = .10), (e) Poisson ratio (problem size = 432 and specified error = .10), and (f) perturbation width (problem size = 2496 and specified error = .10).

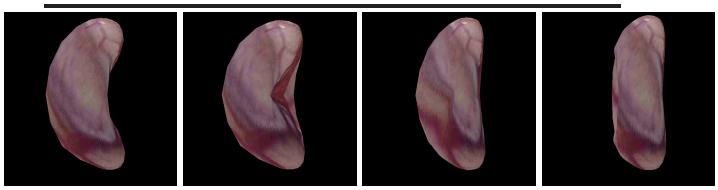

Figure 4. Sequence showing results of impinging on virtual organ with perturbation of constant depth and moving in time along azimuthal direction

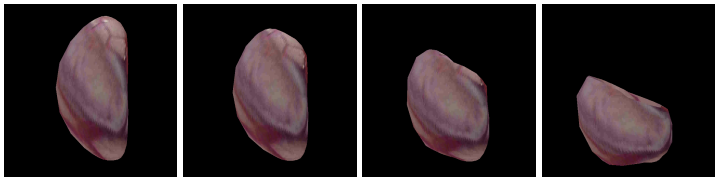

Figure 5. Sequence showing results of impinging on virtual organ with a downward moving plane

References

[1] J. Carrier, L. Greengard, and V. Rokhlin. A Fast Adaptive Multipole Algorithm for Particle Simulations. *SIAM J. Sci. Stat. Comput.*, 9:669–686, 1988. 5

[2] S. Chandrasekaran and M. Gu. Fast and Stable Algorithms for Banded Plus Semi-Separable Matrices. *SIAM J. Matrix Anal. Appl.*, 25(2):373–384, 2003. 2, 3, 5

[3] S. Chandrasekaran, M. Gu, and T. Pals. A Fast ULV Decomposition Solver for Hierarchically Semiseparable Representations. *Siam Journal of Matrix Analysis Applications*, 28(3):603–622, 2006. 2, 3, 5

[4] W. Funck, H. Theisel, and H.-P. Seidel. Vector Field Based Shape Deformations. In *Proceedings of ACM SIGGRAPH 2006*, volume 25, July 2006. 1

[5] J. Huang, X. Shi, X. Liu, and K. Zhou. Subspace Gradient Domain Mesh Deformation. In *Proceedings of ACM SIGGRAPH 2006*, volume 25, pages 1126–1134, July 2006. 1

[6] D. L. James and D. K. Pai. Artdefo - accurate real time deformable objects. In A. Rockwood, editor, *Siggraph 1999, Computer Graphics Proceedings*, pages 65–72, Los Angeles, 1999. Addison Wesley Longman. 1, 2, 5

[7] J. H. Kane. *Boundary Element Analysis in Engineering Continuum Mechanics*. Prentice Hall, Englewood Cliffs, NJ, 1994. 1, 2

[8] S. Kircher and M. Garland. Editing Arbitrarily Deforming Surface Animations. In *Proceedings of ACM SIGGRAPH 2006*, volume 25, pages 1098–1107, July 2006. 1

[9] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2003. 2, 3

[10] L. Shi, Y. Yu, N. Bell, and W.-W. Feng. A fast multigrid algorithm for mesh deformation. In *Proceedings of ACM SIGGRAPH 2006*, volume 25, pages 1108–1117, Jul. 2006. 1