MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Linked Volumetric Objects for Physics-based Modeling

Sarah F. F. Gibson

TR97-20 December 1997

Abstract

In volume graphics, objects are represented as 3-dimensional arrays of sampled data elements. Unlike surface-based graphics, where the surfaces of graphical objects are represented by contiguous 2D polygons or curved spline patches, volumetric models can represent both object surfaces and object interiors. A volumetric object representation is necessary for visualizing complex internal structure and for physically accurate modeling of interactions between solid objects with arbitrary shape and material composition. In this paper, algorithms for manipulating volumetric objects are presented. These algorithms use a linked-element object representation for efficient implementation. Implementation details are presented along with the results of tests for algorithm timing and efficacy.

IEEE Transactions on Visualization and Computer Graphics

Revision History:—

1. First version: Nov. 4, 1997

Linked Volumetric Objects for Physics-Based Modeling

Sarah F. F. Gibson
MERL - A Mitsubishi Electric Research Laboratory
201 Broadway, Cambridge, MA, 02139

Abstract

This paper discusses a linked volumetric representation for graphical objects that enables physics-based modeling of object interactions such as: collision detection, collision response, 3D object deformation, and interactive object modification by carving, cutting, tearing, and joining. Object manipulation algorithms are presented along with implementation details and the results of timing tests.

1 Introduction

In volume graphics, objects are represented as 3-dimensional arrays of sampled data elements. Unlike surface-based graphics, where the surfaces of graphical objects are represented by contiguous 2D polygons or curved 2D spline patches, volumetric models can represent both object surfaces and object interiors. A volumetric object representation is necessary for visualizing complex internal structure and for physically accurate modeling of interactions between solid objects with arbitrary shape and material composition. In this paper, algorithms for manipulating volumetric objects are presented. These algorithms include collision detection, collision response calculation, object deformation, and object modification by carving, cutting, tearing, and joining. These algorithms use a linked-element object representation for efficient implementation. Implementation details are presented along with the results of tests for algorithm timing and efficacy.

2 Motivation

Volumetric object representations are necessary whenever the internal object structure is important for the visual rendering of a graphical object or for the simulation of interactions between objects. For example, volumetric image data of human anatomy can be used to effectively visualize internal anatomical structure as illustrated in Figure 1.

Because the physics of an arbitrary object cannot be determined from a hollow object model, the representation of internal structure is important for physically realistic modeling as well as visualization. The cut plane of the image in Figure 1 illustrates the detailed interior structure of human tissue. In order to accurately model the mechanical behavior of such tissue, we require a graphical representation that can incorporate this complex structure. Both the deformation of objects with complex geometry or heterogeneous material properties and the cutting or tearing of solid tissues require some representation of object interiors.

Modeling the cutting or tearing of objects with complex structure is a challenging problem for surface-based object models, since object cutting requires the generation of new object surfaces along the cutting path. The problem of clipping a surface-based or geometric object model by an arbitrary 2D plane has been addressed in constructive solid geometry (CSG) (e.g. [32]) and polygon rendering. However, there has been little progress towards enabling cutting through surface-based graphical objects along an arbitrary curved path. Both determining the intersection of the cutting path with the object and constructing the new surface along the object's cut surface are challenging problems. Related work in CSG provides mathematical techniques for building new surfaces of intersecting solids [25], [26]. However, applying these methods would require constructing a surface or solid representing the knife path, limiting interactivity. In addition, when the cut is made through a surface-based object model that does not contain information about interior structure, the color, texture, and other features of the cut surface must be fabricated in

order to make the cut look realistic[1]. In contrast to the complexity of cutting through surface-based representations, it will be shown here that it is relatively straight forward to cut through a linked volumetric object. In addition, interior elements in the volumetric object can be used both to influence the cut path (for example by providing variable resistance to cutting) and to determine the appearance of the new cut surface.

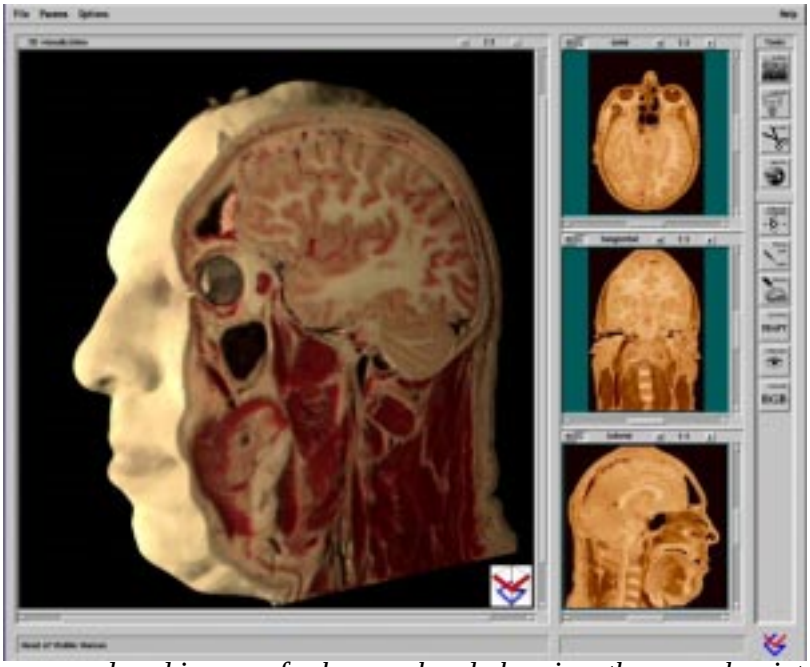


Figure 1. Volume rendered image of a human head showing the complex interior structure of human anatomy. A graphical representation of this interior structure is important both for visualizing the anatomy and for physically realistic modeling of complex tissue interactions. Image courtesy of University of Mannheim, VIRIM group. Data from the Visual Human Project, National Library of Medicine.

An increasing number of applications in computer graphics and animation such as surgical simulation, computer games and system maintenance training, use physics-based graphics for object manipulation. In physics-based graphics, physical laws govern object motion and interactions between graphical objects. When non-rigid objects interact, collisions must be detected, and the response to collisions must be modeled by energy and momentum transfer between objects and by object deformation. While finite element methods (FEM) use volumetric object representations and provide sophisticated mathematical techniques for modeling the physical behavior of solid deformable objects, FEM is computationally demanding and requires careful pre-processing and object meshing for accurate modeling. Because many computer graphics applications require interactivity in addition to physical realism, it has proven difficult to incorporate FEM into most real-time applications. The need to balance realism with the demands of real-time object manipulation and user feedback may require the compromise of modeling interactions that are physically plausible rather than strictly physically realistic. This paper presents a volumetric object representation that has been used to generate physically-plausible simulations of object interactions including object collisions [14], object deformation [15], [16], haptic rendering of graphical objects, and the cutting, tearing and joining of deformable objects.

[1] One way to texture the new surface, when then the model is based on 3D image data, is to map the volumetric data onto the cut surface, although thus can be costly when the cut path is not planar. This method uses a hybrid object model consisting of both the surface model and the volumetric image data.

3 Background

3.1 Volume Graphics

Volumetric data are generated by a number of sources including: medical or industrial scanning such as Magnetic Resonance Imaging, Computed Tomography (CT) or Industrial CT; scientific computing in computational fluid dynamics, finite element analysis, or geometric modeling; and scientific measurements from geological and meteorological surveys. The volumetric data can be sampled on regular or irregular grids or it can be scattered data. The data can consist of both scalar and vector quantities. Since the late 1970's, Volume Rendering techniques have been used to visualize such volumetric data. A variety of Volume Rendering algorithms have been developed for rendering regular grids, irregular grids, and scattered data, for displaying scalar and vector data, and for producing object surface shading, tissue classification, and other visual effects (e.g. see [22]). Because of the demanding computational and memory requirements of Volume Rendering, a great deal of effort has gone into designing fast algorithms and special purpose hardware to enable real-time volume rendering (e.g. see [27], [36]).

Kaufman, Cohen, and Yagel introduce the field of Volume Graphics in [23]. Volume Graphics deals with the synthesis [21], modeling [46], manipulation [14], and rendering (e.g. [22]) of volumetric objects. Kaufman and his collaborators have shown that many of the visual effects produced by conventional graphics, including object shading, anti-aliasing, inter-object reflectance, and radiosity calculations, can also be performed on volumetric objects [38] and [39], and [47]. Recently, attention in Volume Graphics has been given to object manipulation, including haptic interaction with volumetric objects [1] and [34], and physically realistic modeling of object interactions [14], [15], [18].

3.2 Physics-based Graphics

Physically realistic simulation of object interactions has recently become a focus for computer graphics and animation research. Much progress has been made towards fast detection of collisions between complex polygonal models (e.g.[30], [8]) and several researchers have developed methods for simulating the transfer of energy and momentum between rigid polygonal graphical objects upon collision (e.g. [2], [33]). There has also been some work in physics-based modeling of interactions between volumetric objects. For example: Gibson [14] and He and Kaufman [18] have investigated collision detection algorithms for volumetric objects; Greene [17] used a discrete representation of occupied space for guiding the stochastic growth of graphical plants; robotics researchers have used a discrete space representation for obstacle avoidance in path planning (e.g. [29]); and some researchers have investigated haptic rendering of volumetric objects with force-feedback [1], [34].

3.3 Deformable-tissue Modeling

In many applications, realistic object modeling requires the deformation of soft tissues or objects. As discussed above, volumetric representations have advantages over surface-based models because the interior structure impacts the physics of object interactions. However, because volumetric objects consist of a large number of elements, most object deformation techniques are computationally intensive, require careful set-up of the problem and significant pre-processing.

The two most common techniques that have been used for modeling the deformation of volumetric objects are FEM and mass-spring systems. In mass-spring systems, discrete elements with finite mass are linked by springs. When an element is perturbed, the resultant spring forces are used to compute the system dynamics. In a dynamic simulation, the mass-spring system can be represented by a system of second order differential equations with dimensionality proportional to the number of mass elements. Given an initial object configuration and a disturbing force, the dynamic deformation of the object is simulated by advancing the object configuration forward in time using a variety of numerical integration techniques. Mass spring systems have been used to

animate graphical objects (e.g. [44], [7]), for facial animation (e.g. [43], [48], [28]) and in surgical simulation (e.g. [24]).

In FEM, the object is treated as a continuum rather than a discrete array of mass points. Objects are divided into continuous elements that are connected at discrete node points. During simulation of object deformation, displacements of points between the nodes are calculated using a reconstruction function that is a linear combination of displacements of the node points. The weights of the linear equations are determined using a discrete set of interpolating or "shape" functions that are applied to the rest position of the desired point. Both the partitioning of the object into elements and the choice of the shape functions affect the accuracy of the FEM solution. Hence, they are chosen according to the required accuracy and computation limits. As in mass-spring systems, a dynamic FEM simulation requires the solution of a large system of second order differential equations with dimensionality proportional to the total number of element nodes. FEM methods are typically used in off-line scientific computing such as the analysis of mechanical structures. However, FEM have also been used in graphical applications for fabric modeling (e.g. [9]), animation (e.g. [6], [11]) and surgical simulation (e.g. [20], [10], [5]).

In order to achieve interactivity, simulation of object deformation can be accelerated by a significant reduction in the number of elements in the system [20] or data pre-processing. Bro-Nielsen [5] solves for displacements of only surface elements and assumes a small number of externally applied forces. Pentland and Williams [37] pre-calculate the deformation modes of a given object and calculate deformations for an arbitrary force as a superposition of these deformation modes. Bro-Nielsen and Cotin [4], [10] pre-calculate responses to infinitesimal forces and deformations for each node in the element and then approximate the global deformation as a linear superposition of these pre-calculated responses.

Both finite element and mass-spring methods have a number of limitations because they are computationally demanding and because they are inadequate for modeling the large, non-linear deformations that are found, for example, in human tissues (see [16] for further discussion). In this paper we discuss a fast, 2-step algorithm that was presented in [15] for approximating the deformation of volumetric objects. We also present experimental results that suggest that this algorithm will be useful for modeling human tissue.

3.4 Sculpting Graphical Objects

A number of researchers have investigated the sculpting of volumetric objects. Galyean and Hughes [13] represent object material as a 3D array of discrete element values between zero and one, where 1 represents the presence of solid material, 0 represents the absence of material, and values between 0 and 1 represent either partial voluming or reduced material density. Additive sculpting tools increase element values up to a value of 1 and subtractive tools reduce element values. Element values of the object and tools are filtered with a lowpass filter to reduce aliasing at edges and surfaces. The marching cubes algorithm [31] is used to generate a polygonal surface model for object manipulation and visualization. Wang and Kaufman [46] use a similar sculpting technique but use Volume Rendering for visualization. Avila and Sobierajski [1] added a haptic input device to provide force feedback during sculpting and allow tools to modify 3 different values for each element: material color, density, and an index for material classification. Simulations of NC milling have used similar techniques to model the removal of material by a milling machine (e.g. see [45] and [19]).

While these volumetric sculpting techniques have potential applications in volume editing and geometric design, the resultant array of intensity values lacks the physical sense of being an object. This means that pieces cut away from the rest of the volume can not be manipulated as individual objects and that these approaches are not easily extended for sculpting deformable materials.

Bro-Nielsen advocates the use of FEM models because in theory they could be used to simulate physically realistic cutting and tearing behavior. Terzopoulos and Fleischer use a FEM object representation to model fracturing or tearing in an off-line simulation of a relatively small 2D deformable mesh in [42]. When stresses at a given node exceed some limit, they zero material property weighting functions at the mesh node to produce a discontinuity that represents the fracture. However, accurate modeling of arbitrary cutting through a FEM model would require reworking the equations governing the system, repeating pre-processing steps, and remeshing of the model in order to provide a higher resolution mesh at high stress points (such as at the knife tip) with each intervention.

4 Linked Volumetric Object Representations

Elements in a volumetric object encode information about visual and/or physical properties of the object. As illustrated in Figure 2, each element can be represented by a single sampled value, such as image intensity, or a more complex data structure that encodes information such as color, transparency, edge strength, material properties, and connections to neighboring elements. By encoding many attributes for each element, we are able to model complex materials, structures and behaviors. However, since volumetric objects can consist of thousands to millions of elements, (an MRI image of size 256 x 256 x 256 contains 16 million data points) this representation can pose significant challenges for rendering, data storage and retrieval, and tissue modeling.

```
struct SimpleElementStruct {
  char intensity;              /* image intensity from data source */
} SimpleElement;


struct LinkedElementStruct {
  char r, g, b, a;             /* color and transparency for rendering */
  char type;                   /* index into material properties */
  float x, y, z;               /* element position */
  struct LinkedElement *top, *bottom, *right, *left, *front, *back;
                               /* pointers to neighboring elements */
} LinkedElement;
```

Figure 2. Sample data structures. SimpleElement is a data structure for a volumetric object with a single value per element. LinkedElement is a more complex data structure that stores the element's color and transparency, an index to material properties, the element position, and linking pointers to neighboring elements.

If a volumetric object is represented by a static array of regularly spaced elements, element positions are implicitly stored in the data organization and are determined by the array indices. In irregular arrays, such as deformable object models, the position of each element must be explicitly stored in the element data structure. In general, when object elements lie in a grid with fixed topology, connections to neighboring elements are implicit from the data organization and neighbors can be located at a predictable index offset. However, when the object is not sampled on a grid or when the object topology changes because elements are added, removed, or the object is cut, then an explicit representation of connections between neighbors is required. Figure 2b presents a data structure that stores pointers to neighbors when a link exists and a NULL pointer when a neighbor is absent.

As well as providing a means for modeling changes in object topology, an explicit representation of neighboring links can foster fast propagation of information or forces due to object interactions. This feature is used in the deformation algorithm discussed in section 5.4. In addition, unlike an explicit representation of neighbor connections, implicit representations require the storage of

empty elements when an object does not fill the entire volumetric array. For this reason, an explicit representation may make more efficient use of memory.

5 Physics-Based Modeling

5.1 Collision Detection

Detecting object collisions is a fundamental requirement for a physics-based graphics simulation. Efficient algorithms are essential because collision detection is generally performed more frequently than all other operations. Because of the large number of elements in a volumetric object, techniques developed for surface-based graphics can not be directly applied. However, collisions can be simply and efficiently detected for relatively large objects using an algorithm presented in [14] and illustrated in Figure 3.

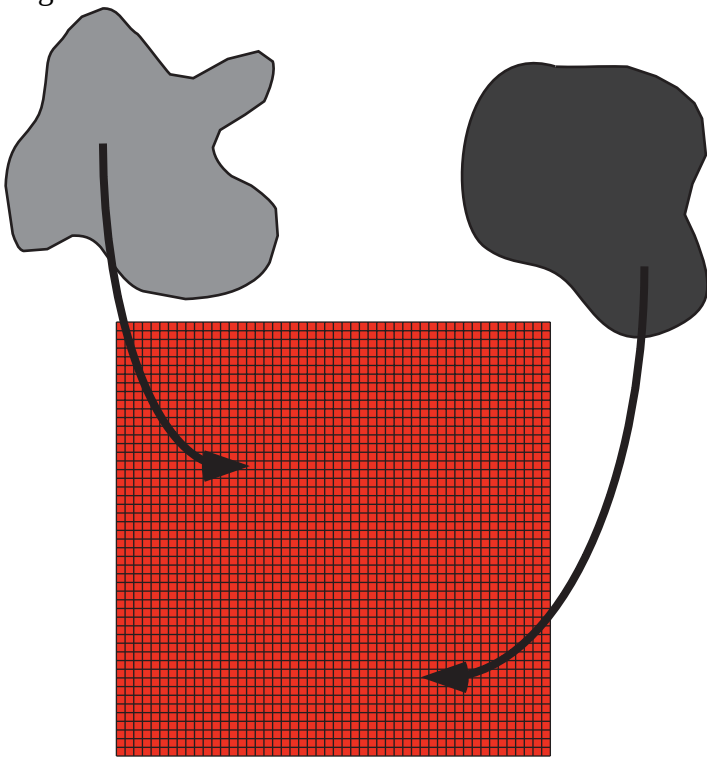


Figure 3. In a straight-forward collision detection algorithm for volumetric objects, all object elements are mapped into a regular grid of occupancy map cells that span the interaction space. A collision is detected if an object element is mapped into a cell that is already occupied by another object.

In this algorithm, each object is stored in its own volumetric array. An additional volumetric array, called the occupancy map, represents the entire object interaction space at the desired resolution for collision detection. Each cell of the occupancy map contains either a pointer to an object element or a null pointer. For each object in the system, object elements are mapped into the occupancy map by writing the element's address into the appropriate occupancy map cell. When an object moves, the occupancy map is updated by removing and then re-mapping element pointers of the moving object. If an element is mapped into an occupancy map cell that is already occupied by a pointer to another object, a collision between the two objects is detected. Because element pointers are stored in the occupancy map, the colliding elements can be easily identified for calculating the collision response.

This method can be used for both regularly spaced and irregularly spaced volumes as long as the element spacing is comparable to the occupancy map cell size. Self intersections can be detected by

noting when an occupancy map cell is already occupied by an element from the same object. In case adjacent object elements map into the same occupancy map cell, neighbor pointers in the element data structure are used to screen potential self intersections for this possibility.

There are a number of straight-forward improvements that can enhance the speed of this collision detection algorithm. First, since collisions occur at object surfaces, it is only necessary to map surface elements or a shell of elements near the surface into the occupancy map. Second, instead of maintaining a large occupancy map to represent the entire object interaction space, this collision detection algorithm can be applied in smaller, temporary occupancy maps that span only the volumes of potential overlap between objects in the system. These volumes of potential overlap can be determined quickly using boxes bounding the volumetric objects and exploiting techniques developed for surface-based graphics for fast collision detection between bounding boxes. Finally, hierarchical data structures can be used to quickly determine which sub-volumes of potentially colliding objects should be written into the occupancy map. He and Kaufman [18] used hierarchical data representations without an occupancy map representation of the interaction space to detect the overlap of volumetric objects. They found that an octree-based representation of static, regularly spaced data yields efficient collision detection for reasonably sized objects.

5.2 Collision Response

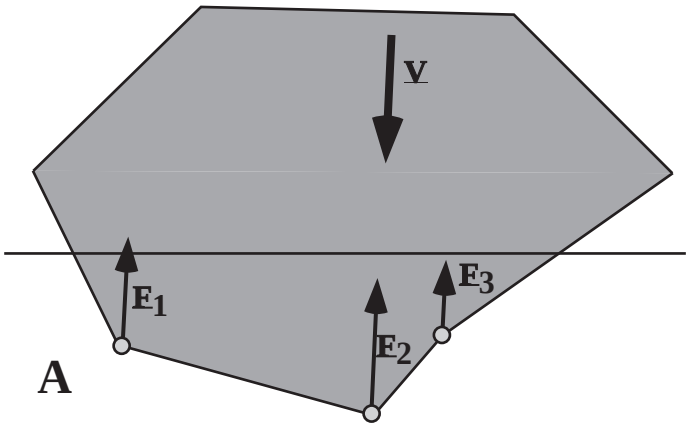
Once a collision between graphical objects is detected, the next step is to determine the response to the collision. Believable simulations require physics-based collision responses. In a system where objects are moved interactively but do not have energy or momentum, the desired effect is usually the prevention of object interpenetration. In systems of rigid objects, collisions result in an exchange of energy and momentum according to physical laws. In systems of non-rigid objects, collisions result in energy exchange, energy dissipation, and energy storage in the form of object deformations. Under some circumstances, graphical objects may be expected to break, tear or fracture upon impact.

When the goal is to prevent object interpenetration, a standard back-tracking algorithm is used to find the point of collision. If object penetration is detected for a proposed new object position, the object is not moved to that position. Instead, the step size of the moving object is reduced. If a penetration is detected using this new step size, the step size is again reduced. Otherwise, the step size is increased slightly and the new position is investigated. This process is repeated until the maximum allowable step increment is determined. With this back-tracking approach, step sizes must be small enough to prevent objects from appearing to leap-frog over one another.

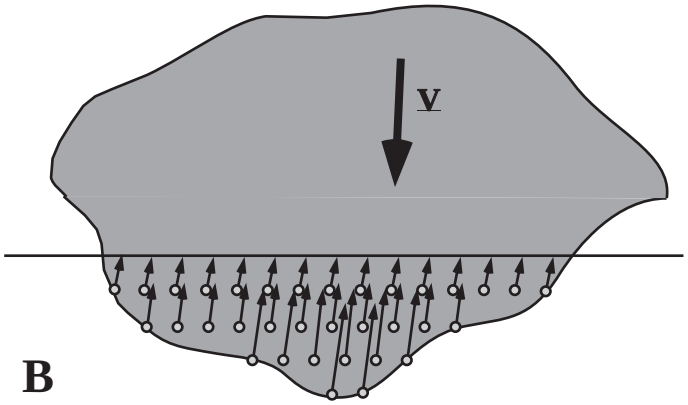
There are two basic methods that have been used in computer graphics for calculating collision responses for rigid objects. These are penalty-based methods (e.g. [40]), where restoring forces are introduced to separate penetrating objects, and analytic methods (e.g. [2] and [33]), where contact forces are determined analytically from constraints that prevent object inter-penetration and guarantee physically realistic dynamics. While analytic methods are more accurate and numerically stable than penalty-based methods, penalty methods have the following advantages for volumetric objects. First, penalty-based methods have been successfully applied to deformable object models while current analytic approaches are limited to rigid bodies[2]. Second, a tractable solution of an analytic method relies on a relatively small number of contact points. In surface-based graphics, contact points can be generalized to occur at polygon vertices or edges [2], and hence the number of contact points is relatively small for object models represented by a reasonable number of surface polygons. In a volumetric model of the same object, the number of contact points could be several orders of magnitude larger, making an analytic solution less tractable.

[2] One exception being the work of Baraff and Witkin [Baraff and Witkin, 1992] who used analytic methods for simulating collisions between flexible bodies with a limited deformation model.

Penalty-based methods apply a restoring force to object vertices that are found to penetrate another object. The restoring force acts to separate the interpenetrating objects during a dynamic simulation. The strength of the restoring force is generally a function of the depth of penetration and the object stiffness. A large stiffness puts a higher penalty on object penetration, but also requires small integration time steps for numerical stability. Figure 4 illustrates in 2D the calculation of restoring forces of interpenetrating polygonal and volumetric objects.



$$\mathbf{F}_r = - k \, (\text{dist}(v_1) + \text{dist}(v_2) + \text{dist}(v_3)) \, \underline{\mathbf{V}}$$
$$= - k \, (\mathbf{F_1} + \mathbf{F_2} + \mathbf{F_3})$$



$$\mathbf{F}_r = - k \, (\Sigma \, \text{dist}(v_i)) \, \underline{\mathbf{V}}$$
$$= - k \, \Sigma \, \mathbf{F_i}$$

Figure 4. In penalty-based methods, collision forces are calculated as a function of the distance that an object has penetrated another object. For polygonal object models, the response force can be calculated from the depths of penetrating vertices. For most polygonal object models, the number of penetrating vertices is relatively small (tens or hundreds of contact points). However, for volumetric objects, the force is summed over all penetrating volumetric elements and the number of such elements may be quite large (possibly thousands or more penetrating elements).

In the volumetric approach, forces for each penetrating element are summed together to determine the restoring force. In order in increase the speed of the collision response calculation, the distance of the element from the nearest surface is encoded into each element structure. The approach for calculating collision responses is summarized as follows:

At each time step:

1) detect all penetrating object elements
2) use the object material stiffness and the distance-to-surface value of each element to calculate restoring force contributions for each element
3) sum the restoring forces and torques and adjust the state of the dynamic system

5.3 Object Deformation with Linked Volumes

For applications such as surgical simulation, physically plausible deformation must be achieved at interactive rates for objects with complex geometry and internal structure. While FEM and mass-spring methods have been used for reasonably complex objects[3], they require significant preprocessing. Here we discuss an algorithm from [15] for deforming volumetric objects that exploits the fact that complex system behavior can result when a large number of elements each follow simple rules. The deformation algorithm consists of two processes that are applied relatively independently. Both processes maintain relationships between local elements. Forces applied to the object affect local elements and are propagated to other parts of the volume by local interactions.

The first process, 3D ChainMail, responds to applied forces by quickly approximating the deformed object shape. The approximate shape is guaranteed to satisfy local inequality constraints between neighboring elements but it may not produce a configuration of the volumetric elements in which element spacings are optimal. The second process, an elastic relaxation, adjusts element positions to reduce the local system energy, which is a function of distances between neighboring elements. Optimal distances between elements result in a system of low energy. Distances that are too large or too small result in a higher energy system. If the system has a single optimal configuration and if the forces used to move elements towards this configuration are a linear function of differences from the optimal element spacing, then the system behaves like a linear elastic system. The elastic relaxation process tends to relax the shape of the object, smoothing out the approximate shape produced by 3D ChainMail. The result is that when an object is directly manipulated, it assumes an approximate shape very quickly and then relaxes to a more natural shape as the simulation proceeds.

Although inertia and damping are not explicitly modeled in the deformation algorithm, objects modeled with these two procedures behave like a system with inertial properties and a damped behavior. The inertial behavior results from the fact that in 3D ChainMail, elements do not move unless they violate constraints with their local neighbors. If constraints are violated, an element moves a minimum distance to satisfy the constraints. The system exhibits damped behavior because the elastic relaxation procedure is a closed, negative feedback system that adjusts an element position by a partial step towards the optimal position. When the step towards the optimal position is small, the system is more damped. As the step size -- the gain of the closed feedback system -- increases, the system becomes critically damped and eventually under-damped.

5.3.1 3D ChainMail

In the 3D ChainMail algorithm, each element is linked to its top, bottom, left, right, front and back neighbors. After an element is displaced by an applied force, distances between the moved element and its neighbors may violate a maximum or minimum distance constraint. If this occurs, the affected neighbors are moved until their constraints are again satisfied. Neighbors of these points may also have to be moved to satisfy additional constraints. When an element is displaced, the algorithm propagates the deformation through the object in a single time step. This provides fast response to user input, even though the shape assumed by the object may not minimize the system energy. 3D ChainMail is particularly fast for tissues with homogeneous (though possibly

[3] Stephan Cotin reports an FEM system that provides interactive manipulation of an 8000 node liver model. However, the model requires many hours of preprocessing to achieve interactivity. Personal communication from Stephan Cotin, INRIA, Sopia Antipolis, France, April, 1997.

anisotropic) materials because disturbances can be propagated through the volume by considering each volume element at most once and by comparing each element to at most one neighbor.

Figure 5 illustrates the deformation of a 2D mesh of points using 2D ChainMail. In the following, the discussion will focus on 2D objects. However, the extension to 3D is straightforward and both 2D and 3D versions of this algorithm have been implemented (see Section 6.2).

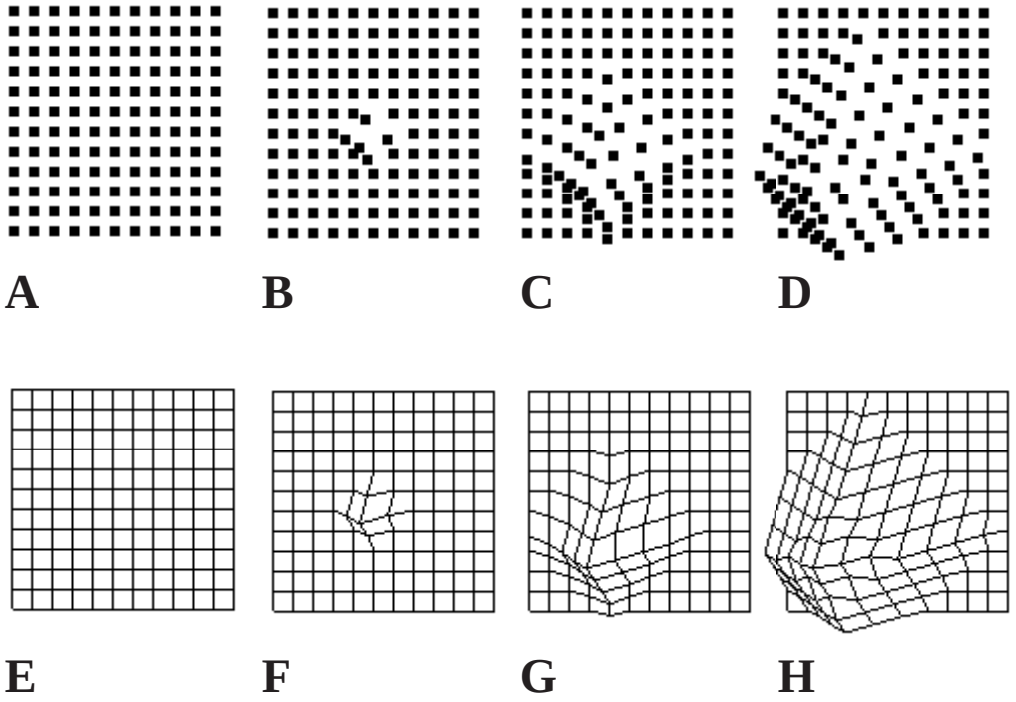


Figure 5 illustrates the deformation of a 2D object made up of linked elements. A single element is manipulated and other elements move to satisfy minimum and maximum distance constraints. In the top row, the object is represented by a point at each element. In the bottom row the links between elements are drawn.

Two types of lists are maintained in 2D ChainMail: 1) a list of pointers to moved elements and their previous positions; and 2) lists of elements than are candidates for movement. The four movement candidate lists are classified according to whether list elements are the top, left, bottom, or right neighbors of their sponsoring element. Each candidate element is processed in turn, starting from the element where the force is applied and then proceeding in order through the right, left, top, and bottom candidate lists. When processing an element, the distance between the element and its sponsoring element is checked. If constraints on this distance are violated, the element is moved a minimum distance until the constraints are satisfied. When an element is moved -- either under direct control of the user or indirectly in response to a neighbor's movement -- the element becomes a sponsor to its neighbors that have not been moved and these neighbors are appended to their respective movement candidate lists.

The constraints on distances between elements are defined as follows: each element must lie within a horizontal range of $\partial x_{\min}$ and $\partial x_{\max}$ from its left and right neighbors and within a vertical range of $\partial y_{\min}$ and $\partial y_{\max}$ from its top and bottom neighbors as illustrated in Figure 6a. These constraints limit the stretching and compression of the material. In addition, each element must lie within $\pm \partial y\_horiz_{\max}$, from its horizontal (left and right) neighbors and within $\pm \partial x\_vert_{\max}$, from its vertical (top and bottom) neighbors as illustrated in Figure 6b. These constraints limit the material shear.

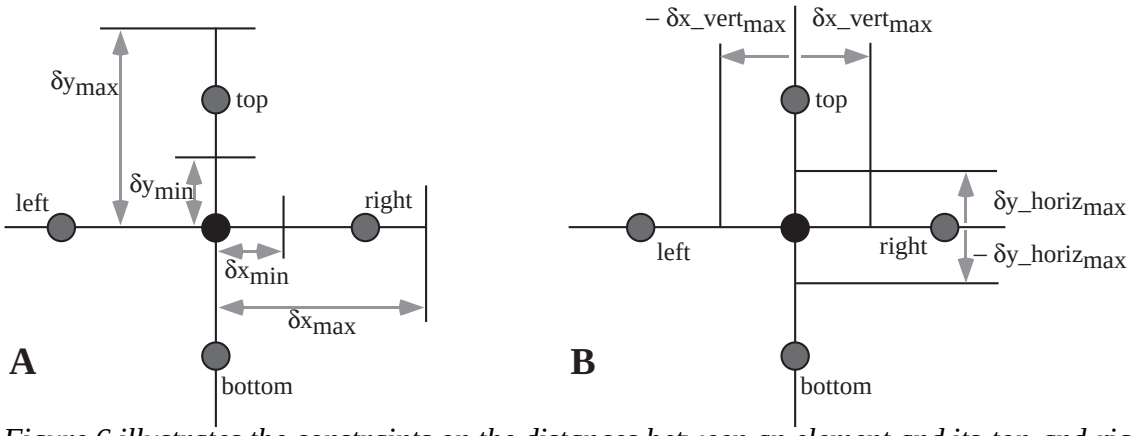


Figure 6 illustrates the constraints on the distances between an element and its top and right neighbors. In 6a, the constraints that limit stretching and contraction are illustrated. The y-value of the top element must lie within the distance range of $\partial y_{\min}$ *to* $\partial y_{\max}$ *from the y-value of the central element. Similarly, the x-value of the right element must lie within a range of* $\partial x_{\min}$ *to* $\partial x_{\max}$ *from the x-value of the central element. In 6b, the constraints that limit shear are illustrated.*

The 2D ChainMail algorithm is summarized as follows:
1) When a force is applied to an element: i) the element is moved; ii) a pointer to the element and its old position are stored in the list of moved elements; and iii) its four neighbors are added to the appropriate movement candidate lists.

2) Each candidate list is processed in order until all of the candidate lists are exhausted or a collision is detected, in which case moved elements are returned to their previous positions using the moved candidate list, and a smaller step towards the desired position is attempted. The candidate lists are processed in the following order: right, left, top, bottom.

3) The right candidate list is processed in the following manner. Beginning with the first element in the list, the maximum and minimum distance constraints between the neighbor and its sponsoring (left) neighbor are checked. If the distance constraints are violated, the element is moved by a minimum distance until the constraints are satisfied. For example, if an element is too close to its sponsoring left neighbor, it is moved to the right until the two elements are separated by $\partial x_{\min}$. Considering both stretch and shear constraints, the new element position can be calculated as follows:

$$\text{if } \left(x - x_{left} \right) < \partial x_{\min}, \quad x = x_{left} + \partial x_{\min}$$
$$\text{else if } \left(x - x_{left} \right) > \partial x_{\max}, \quad x = x_{left} + \partial x_{\max}.$$

$$\text{if } \left(y - y_{left} \right) < -\partial y\_horiz_{\max}, \quad y = y_{left} - \partial y\_horiz_{\max}$$
$$\text{else if } \left(y - y_{left} \right) > \partial y\_horiz_{\max}, \quad y = y_{left} + \partial y\_horiz_{\max}.$$

If a right candidate is moved, its top, right and bottom neighbors are added to the appropriate candidate lists. Each right candidate is processed in turn until no right candidates remain.

4) The left list is processed in a similar way except that left elements are sponsored by their right neighbors and movement of a left element causes its bottom, left, and top neighbors to be added to the candidate lists.

5) The top and bottom lists are also processed in a similar manner except that the top and bottom elements are sponsored by their bottom and top elements respectively and movement of a top (or bottom) element causes only a top (or bottom) element to be added to the appropriate candidate list.

This algorithm must be modified slightly for non-convex objects. In non-convex objects, if the right or left neighbor of a moved top (or bottom) element does not have a bottom (top) element, it should be added to the appropriate candidate list. This may require that candidate lists be visited more than once to exhaust all elements from the candidate lists.

The algorithm is especially fast for three reasons: 1) no element is considered more than once for each deformation, 2) the deformation is propagated outwards from the selected point until constraints are no longer violated so that the propagation is terminated as soon as possible and 3) each element is compared to only one neighbor (its sponsoring neighbor) for determining its movement. The first two properties result from sequence in which elements are added to candidate lists. The last property is valid for objects with homogeneous material properties and is proven in Appendix A.

5.3.2 Elastic Relaxation

The approximate object shape produced by 3D ChainMail does not necessarily have an optimal energy configuration. Hence, an elastic relaxation is applied to locally adjust relative object positions and reduce the system energy. The system energy depends on the distances between object elements. If these distances fall within an optimal range, the system energy is low. When the distances are outside of this range, the system energy is higher. During elastic relaxation, forces are applied to each element sequentially, in an iterative, closed feedback system that adjusts element positions to reduce the system energy.

The nature of the forces that govern the element position adjustments determines the tissue response to applied deformations. For example, if the relaxation forces are a linear function of element displacements from their optimal positions, then the stress vs. strain response of the simulated material is linear. Experimental results presented in Section 6.2 illustrate that non-linear relaxation force functions can produce non-linear tissue behavior. In addition, the force function also determines the elasticity of the tissue. If the force function relaxes the system towards a single optimal configuration, then the object behaves elastically. However, if the system has a range of optimal configurations, then the object may assume a configuration that is different from the original shape and the object behaves plastically.

Elastic relaxation is applied between applications of 3D ChainMail and whenever processing time is available. The time constant and damping of the simulated tissue response can be modified by adjusting the magnitude of relaxation forces and how frequently this process is applied.

5.4 Cutting, Tearing, and Connecting Linked Volumes

Applications such as surgical simulation require the modeling of cutting, tearing, and connecting objects. Current volumetric sculpting methods assume that the object is a static 3D array of density values. These density values are increased or decreased to represent addition or removal of material but there is no way to physically simulate the actions of cutting, tearing, or joining. However, using a linked volumetric model, cutting and tearing are performed by breaking connections between neighbors while objects are connected by forming new links between object surface elements.

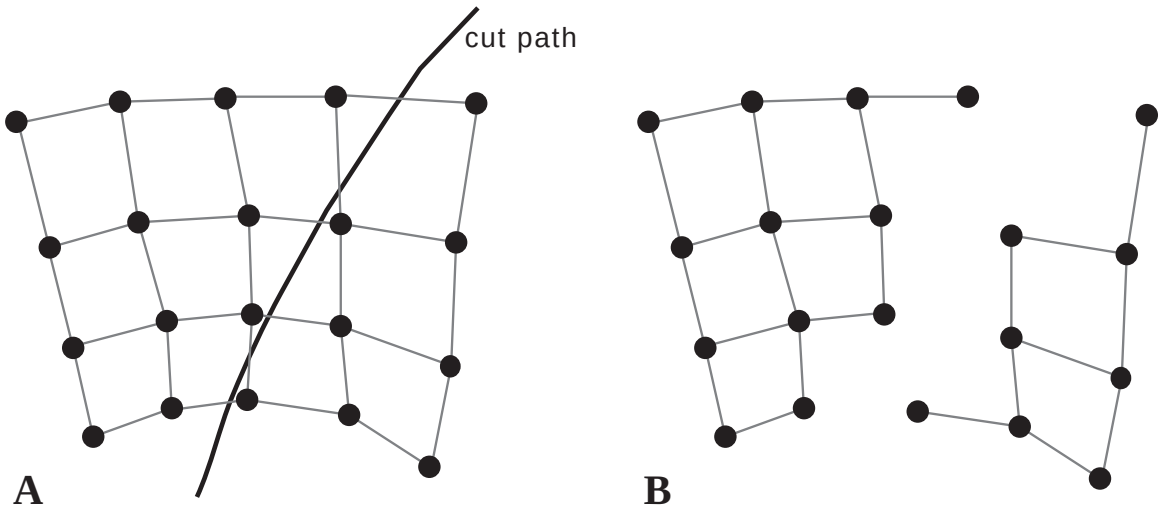


Figure 7. When cutting volumetric objects, links between elements that lie along the cutting path are broken by removing the connections between appropriate neighbors.

Figure 7 illustrates cutting, where connections are broken along the path of a knife instrument as it is passed through the virtual object. Intersections between the knife path and the object are detected by moving the knife volume through the occupancy map and checking for collisions. The occupancy map is modified slightly so that cells corresponding to object elements contain element pointers while cells between linked elements that do not fall into adjacent occupancy map cells contain pointers to both of the linked elements. If the knife path encounters a cell occupied by an element, the element is removed and appropriate neighbor connections are removed. If the knife passes through a cell indicating a link between two elements, the connection between those two elements is broken. This algorithm is illustrated in Figure 8.

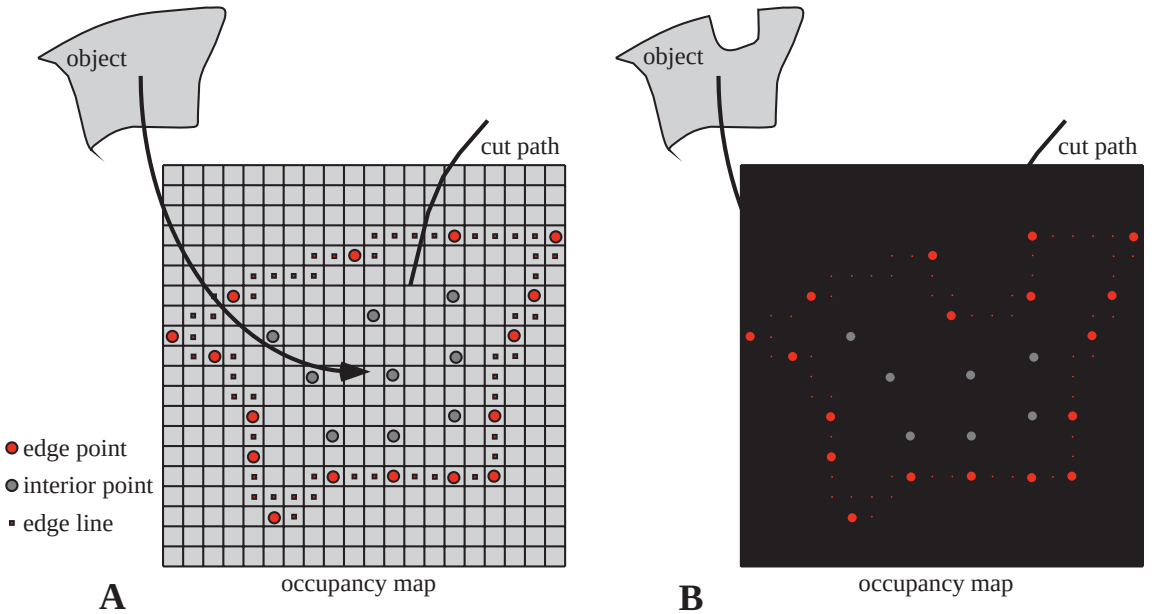


Figure 8. To model tissue cutting, both object elements and the links between elements are mapped into the occupancy map. When the cut path intersects a cell containing an element, the element and its links to neighbors are removed. When the cut path intersects a link between two elements, then the connection between the two elements is removed. In practice, only links between elements on the object surface need to be mapped into the occupancy map as shown here.

Tearing occurs when the distance between two elements is stretched beyond an allowable limit, for example, when two parts of an object are pulled in opposite directions. When a limit violation between two elements cannot be resolved by moving neighboring elements, the connection between the elements is broken.

For joining objects together, occupancy map cells in the vicinity of the joining instrument are searched for object elements that have missing neighbor connections. Elements with complementary missing neighbors are paired and joined. This process is illustrated in Figure 9.

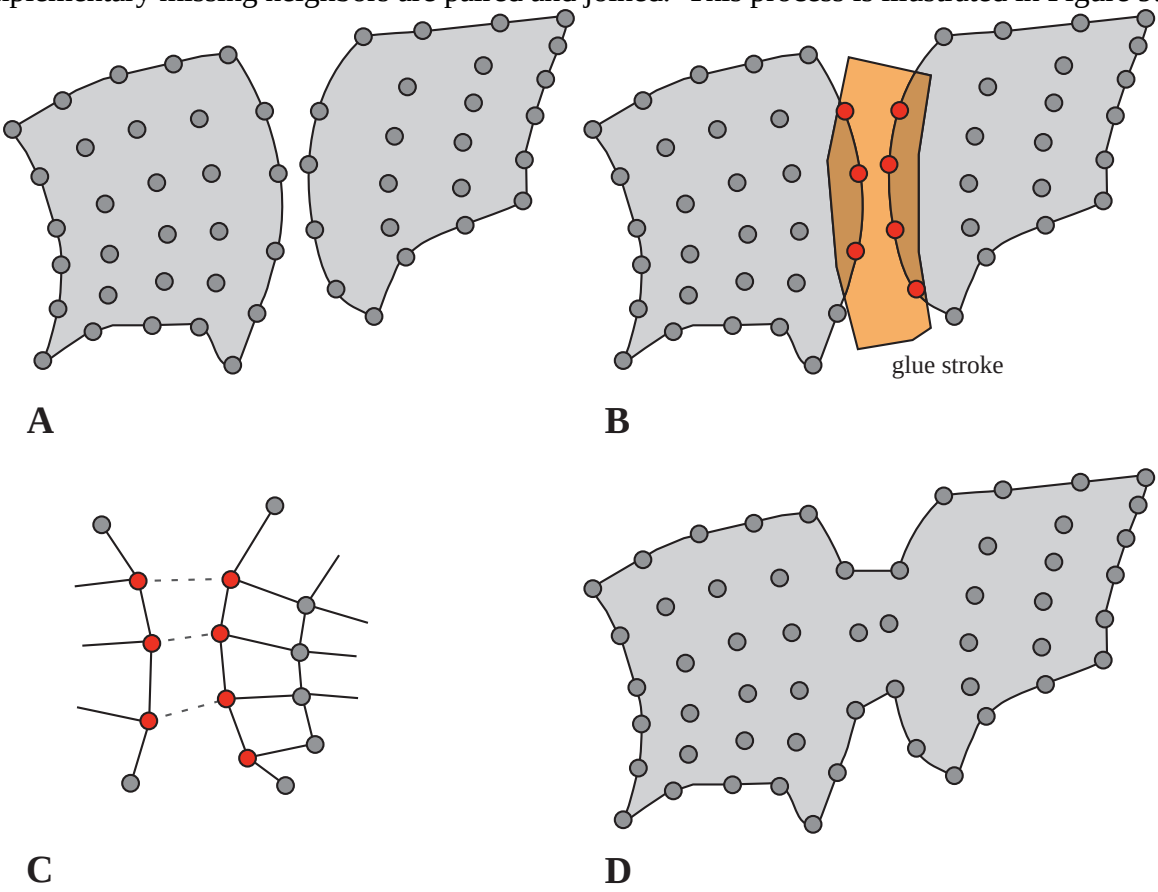


Figure 9. To join two elements together, occupancy map cells in an area or volume surrounding the joining instrument are searched for elements with missing neighbors. Those elements that have corresponding missing elements are paired. For example, an element missing a right neighbor is paired with an element missing a left element. Finally, paired elements are joined by setting the appropriate neighbor links.

We have implemented the 2D system shown in Figure 10 where objects can be moved (resulting in object translation), cut, grasped and moved (resulting in object deformation), tacked into place, glued together, and erased interactively using the computer mouse. Pointing and clicking on the buttons on the right side of the user interface switches between these modes. The system was implemented in C, Tcl/Tk, and OpenGl and runs on an SGI platform.

Figure 10. A 2D system where objects consist of an array of linked elements. By selecting a tool on the right side of the user interface and manipulating the tool with the computer mouse, objects can be moved, cut arbitrarily, deformed, tacked into place, glued together, and erased interactively.

6 Experimental Results

6.1 Object Collisions

6.1.1 Implementation details

Volumetric collision detection has been implemented in a number of 2D and 3D systems ranging from computer-based jigsaw puzzles and 2D drawing tools to 3D object manipulation. Most of the systems have been implemented using the algorithm described in Section 5.1, where a single occupancy map representing the entire interaction volume is maintained and updated as objects move about. All systems have been implemented in C on either SGI or HP platforms. Many of the systems prevent object interpenetration and self intersection of deforming objects by backtracking the object when a collision is detected as described in Section 5.1.

A number of the methods have been implemented for improving the speed of the collision detection algorithm. For example, significant speedups are obtained when only surface elements are mapped into the occupancy map. In addition, by suspending the mapping of object elements into the occupancy map as soon as a collision is detected in order to begin backtracking, the speed of interactive systems that prevent object interpenetration has been significantly increased.

6.1.2 Timing experiments

A number of timing tests were made in order to demonstrate the performance of the collision detection algorithm. The tests involved two objects: a stationary sphere with a radius of 30 voxels located at the center of the interaction space; and a cube, initially located at the side of the interaction space. During the test, the cube was moved in steps of size 5 voxels along a path through the center of the interaction space. Two sets of experiments were performed. In the first, the cube was permitted to penetrate the sphere and collisions were observed by recording all of the overlapping elements of the two objects for each step. In the second, objects were prevented from penetrating each other by combining collision detection and backtracking. Two sizes of cubes were used in the tests, a cube with dimensions 31x31x31 and a cube with dimensions 15x15x15. The collision detection was performed for two versions of the collision detection algorithm, the

first mapping all of the cube elements into the occupancy map and the second mapping only surface elements into the occupancy map.

The results from these tests are presented in Table 1. Times are reported in milliseconds. The system was integrated in C on an SGI Indigo2 without particular efforts to optimize the code. The reported times are for all operations in each step except rendering. Since collision detection timing depends on the degree of overlap between objects, timing has been reported, in the collision detection test: for the cube in free space; and fully inside the sphere; and, in the system for preventing object interpenetration: for the cube in free space; in initial contact with the sphere; and when touching the sphere. When objects are prevented from interpenetrating, the object elements are mapped into the occupancy map only until a collision is detected, and then backtracking is used to reduce the step size. Hence, in tests of this process, object element ordering will affect the timing. For this reason, in the object penetration tests, the cube was placed at the center of each of the 6 faces of the interaction space and moved from left to right, right to left, top to bottom, etc. through the sphere at the center of the interaction space. The times reported in Table 1 are the timing results from left to right and the average times (in brackets) for the 6 directions of motion.

object (no. of elements)	**collision detection**		**no penetration**		
	free space	collision	free space	impact (avg)	touching (avg)
large cube (29791 els)	134	156	133	153 (238)	34.8 (96.9)
small cube (3375 els)	13.7	15.5	13.8	14.3 (42.5)	0.32 (9.94)
large cube surface (5402 els)	19.7	23.7	19.6	24.7 (25.4)	8.85 (10.6)
small cube surface (1178 els)	3.76	4.42	3.79	4.11 (4.74)	0.32 (1.05)

Table 1. Timing for volumetric collision detection. Times, measured in milliseconds, are for moving or attempting to move the cube with a step size of 5 voxels. Collision detection times are the times required to detect and record all of the overlapping points (or surface points) in the objects as the cube is passed through the center of a large sphere. No penetration times are the times for each step when the objects are prevented from penetrating each other. See the text for more detail.

These timing tests show that collisions can be detected and object interpenetration can be prevented at interactive rates between reasonably large objects using a simple collision detection algorithm (even the worst case of collisions using all element points for a 31x31x31 volume can be performed at more than 6 frames per second without optimization). The use of surface element mapping, data hierarchies, bounding boxes, and other techniques developed for surface-based graphics can greatly improve the speed of the collision detection.

6.2 Object Deformation

6.2.1 Implementation details

The combination of 3D ChainMail with an elastic relaxation process has been implemented in a number of 2D and 3D applications. One example application is a 3D system for manipulating volumetric, deformable objects with continuous control of object mechanical properties so that materials ranging from rigid to deformable and elastic to plastic can be modeled. A second application is a 2D drawing tool that allows objects to be input from a library or drawn by hand and then interactively deformed and manipulated. In this application, the deformation technique is combined with collision detection so that objects are prevented from interpenetrating. All of the applications have been implemented in C on an SGI platform. Objects are interactively manipulated with a computer mouse or 3D input device. User interfaces were implemented with Tcl/Tk and deforming objects are visualized rendering all elements for 2D objects and surface points of 3D objects using OpenGL. We have attained interactive deformation of objects with as many as 50x50x50, or 125,000 elements, on an SGI Indy.

6.2.2 Timing experiments

Because all interactions are local, and because elements are moved at most once with each application of both 3D ChainMail and the elastic relaxation step, the time taken to deform an object will vary at worst linearly with the number of elements in the system. Figure 11 show the result of timing tests for 3D ChainMail applied to a 2D system where the number of elements, the object deformability, and the amount of displacement of a control element were varied. Five square objects were used of the following sizes: 10x10, 50x50, 100x100, 150x150, 200x200. In separate experiments, the deformability of the object was set to rigid, moderately deformable (medium), and very deformable (loose) by increasing the range of allowable distances between neighbors. In each experiment, a control element located at the center of the object was moved at a constant rate, 100 times around a square with dimensions: 10x10, 50x50, and 100x100. The figure reports the results of movement around the squares of dimensions 50x50 (small deformation), and 100x100 (large deformation). Rigid objects show the worst response because all elements in the object are considered and moved for each displacement of the control element[4]. However, even in this worst-case scenario, the time required to "deform" the object increases only linearly with the number of elements in the object. For deformable objects, the response time is better than linear because the deformation often does not have to be communicated to all of the object elements. In a very deformable object, a small deformation is communicated to a small number of local neighbors and the deformation is very fast.

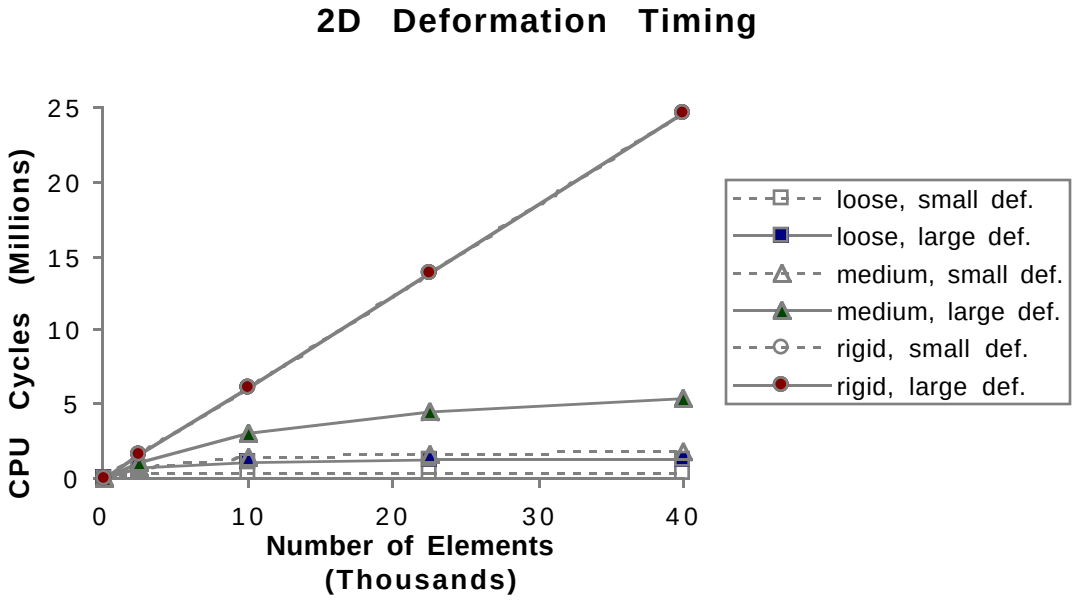


Figure 11. Results of timing experiments for a 2D system. Objects with 100, 2500, 10000, 22500, and 40000 elements were deformed by moving a control element at the center of the object at a constant rate 100 times around a square of size 50x50 (small deformation) and 100x100 (large deformation). The results for objects with deformability set to rigid, moderately deformable (medium), and very deformable (loose) are presented.

[4] While it is important for a deformation algorithm to be well behaved for objects that lie in the continuum between rigid and deformable, in practice it is probably not advisable to use a deformable modeling technique to manipulate rigid objects. However, these experimental results for rigid objects can be used to compare this approach with other volumetric deformation techniques, such as mass-spring methods, which behave badly for stiff objects.

6.2.3 Tissue response measurements

The volumetric deformation approach presented in this paper was originally developed for simulating human tissue. Human tissue has a number of complex behaviors that have been reported in the biomechanics literature (e.g. see [12]). One of these behaviors is that the stress in a system loaded at a constant rate of deformation varies non-linearly with the amount of deformation. A second behavior is hysteresis during loading and unloading: the internal forces (or stress) of the material increases at a different rate when a load is applied than it decreases when the load is removed. A third behavior, a process known as relaxation, is that when living tissue is deformed with a constant rate to a given length and then held at that length, the internal stress decreases with time towards a lower stress level. In order to test the material properties of the combination of ChainMail and elastic relaxation, a 1D deformable system has been implemented. One end of the 1D chain was fixed and the other end could be displaced at a constant rate along the direction of the chain. As described below, the tests show that the combination of ChainMail and elastic relaxation is capable of modeling all three behaviors and hence that it is a good candidate for modeling living tissues.

This paper has described an approach for modeling deformable objects that is very general. 3D ChainMail ensures that an object with even complex geometric structure will at least assume a shape that satisfies minimal constraints and the elastic relaxation process determines the characteristic behavior of the tissue. A variety of force functions can be used by the elastic relaxation to determine the internal system energy and calculate local adjustments of element positions. If the relaxation force is a linear function of the difference between the distances between elements and the optimal element spacing, then the internal force, or stress, of the material varies linearly with deformation. If the function used to determine stress and adjust element positions is quadratic or more complex, then the stress vs. deformation curve is non-linear. Figure 12 shows the result of several tests on the 1D chain described above. Two of the curves resulted from applying a linear force function in the elastic relaxation step and three curves resulted from a quadratic force function. In practice, a wide variety of both analytic and measured force functions could be applied by the elastic relaxation process by using lookup tables to determine forces for a given element separation. By setting maximum and minimum separations between elements, 3D ChainMail limits the extent of influence of the force functions.

Because the inequality constraints of the 3D ChainMail algorithm have different effects when the object is stretched or compressed, like living tissue, an object modeled with 3D ChainMail will behave differently when it is loaded and unloaded. In order to test this hypothesis, 3D ChainMail and a quadratic elastic relaxation function were applied to the 1D chain. The chain was stretched at a constant rate to a given length and then compressed at the same rate while recording the total internal force. As illustrated in Figure 13, the resultant behavior showed hysteresis in the stress vs. deformation curves that is similar to the hysteresis that has been measured in human and animal tissue [12].

Because of the damped behavior of the deformation system due to using a small gain in the elastic relaxation feedback loop, the system exhibits a behavior similar to tissue relaxation, in which, when living tissue is stretched at a constant rate to a given length and then held fixed at that length, the internal stress in the tissue decreases with time towards a lower value. Figure 14 shows the result of a test that demonstrates relaxation in the deformation algorithm. A 1D chain was stretched to half of its maximum length and then its length was fixed while the internal force was measured as a function of time. A quadratic force function was used to calculate internal forces and to adjust element positions in the elastic relaxation.

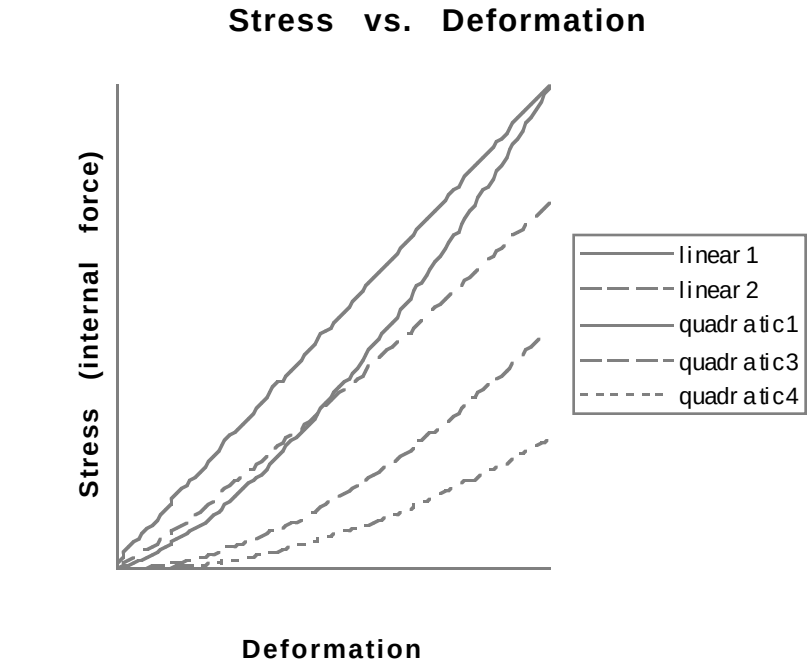


Figure 12. Measured internal force as a function of deformation for loading at a constant rate. The two linear curves resulted from using a linear function to calculate internal forces and adjust element positions during elastic relaxation. The three non-linear curves resulted from using a quadratic function to calculate stress and adjust element positions.

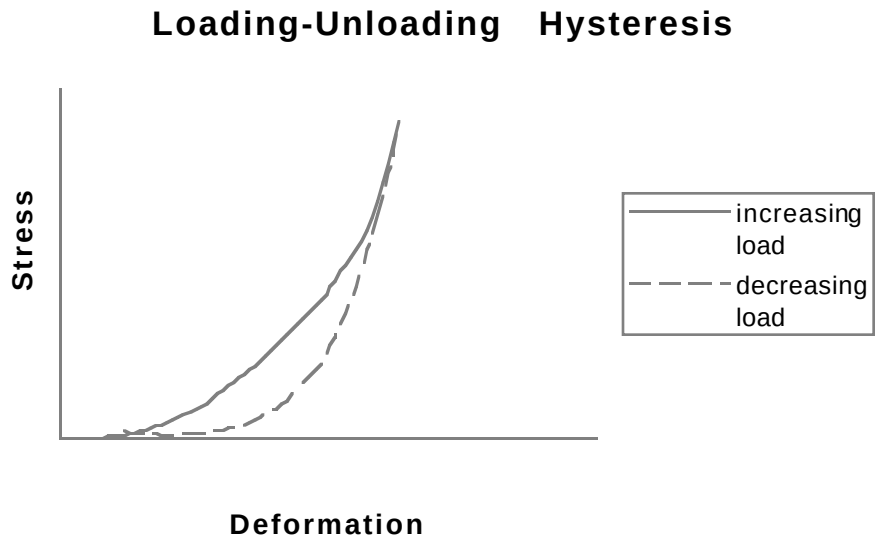


Figure 13. Stress vs. constant loading and unloading in a 1D chain. The object length is stretched to given point at a constant rate of deformation and then reduced at the same length. As in living tissues, the loading and unloading curves exhibit hysteresis.

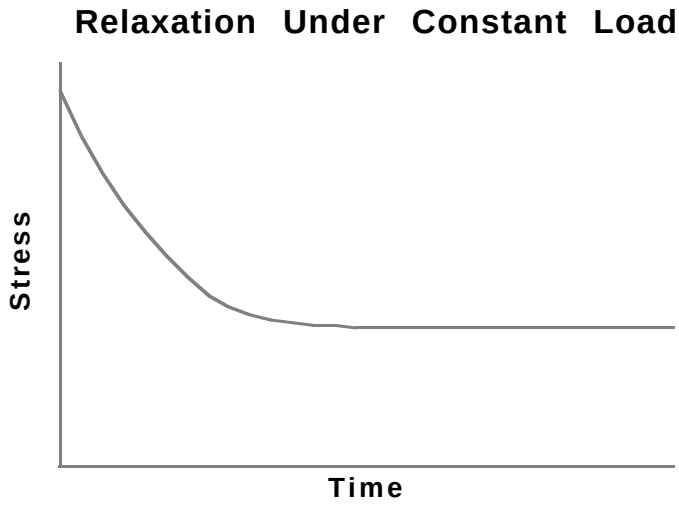


Figure 14. The result of an experiment in which a 1D chain was stretched to half its maximum length, the length was fixed, and the stress was measured as a function of time. Like living tissue, the material exhibits relaxation: the stress decreases as a function of time to a lower value.

6.2.4 Parameter assignment

The experiments that have been presented here were designed to test tissue behaviors, rather than to validate the approach for specific materials. For this reason, parameters were not chosen to correspond to the mechanical properties of specific materials or tissues and the measured internal forces are based on heuristic functions (i.e. linear and quadratic curves) rather than known or measured relationships. In general, there are 3 ways to chose material parameters for specific tissues: 1) choose parameters to match the output of this system to predictions of a more rigorous mathematical model such as non-linear, large deformation FEM; 2) choose parameters to match measured tissue responses; and 3) allow surgeons or other specialists to interactively tune the parameters until they are satisfied with the tissue behavior. Once the tissue parameters are set, similar techniques could be used to validate the tissue model as long as the parameter setting technique and the validation technique are independent.

7 Discussion and Future Work

Volumetric methods provide a powerful tool for modeling, visualizing, and interacting with graphical objects. This paper has presented a linked volumetric structure and a set of algorithms to model physics-based interactions between objects such as: collision detection, object deformation, and the arbitrary cutting and joining of objects. Many of these phenomena are particularly difficult to model with surface-based graphics. These algorithms have been implemented in C and have been shown to be effective and interactive for reasonably sized volumetric objects.

The results of a number of experiments for testing timing and the material properties resulting from these algorithms have been presented. For example, it has been shown that the 3D ChainMail and elastic relaxation algorithms result in deformation where: the system dynamics exhibit inertial and damping behavior; tissue characteristics can range from rigid to deformable, and elastic to plastic; tissues with non linear stress-deformation curves can be modeled; and, similar to living tissues, materials can exhibit relaxation and hysteresis between loading and unloading.

Although the work that has been presented in this paper has demonstrated the potential and the feasibility of volumetric methods, there are many areas that require further investigation. Some examples of these areas are listed here. First, improvements can be made to algorithms that have been presented here by implementing changes in the collision detection algorithm, developing a more general extension to 3D ChainMail that is less axis dependent and can handle heterogeneous materials, and developing more efficient data structures for visualization, physics-based modeling, and haptic rendering. Second, additional physics-based algorithms for volumetric objects, such as algorithms that model collision responses or that handle liquids as well as solids [41], need to be developed. Third, an important step for practical applications will be the experimental verification of simulated material properties by comparing predicted results to experimental data and the output of rigorous mathematical computation as discussed in 6.2.4. Fourth, because rendering is often a bottleneck for real-time interaction with volumetric objects, efforts must be made to produce real-time, high quality visual and haptic rendering of deforming volumes and to provide realistic, high quality rendering of object surfaces. Finally, hybrid graphics systems that combine both surface-based and volumetric models in rendering and physical modeling are needed so that objects can be represented in an optimal format.

Acknowledgments

Thanks are extended to E. Gibson, H. Lauer, and B. Mirtich for helpful discussions and editorial comments on early drafts of this manuscript.

Appendix A: Proof of 2D ChainMail Comparison Theorem

Theorem:

In 2D ChainMail, each element can be compared to a single neighbor when the object has constant deformation limits throughout its volume.

Proof:

Because the starting position for each element already satisfies local constraints for neighbors that have not moved, an element must only be compared against those neighbors that have already moved. If the candidate is moved to satisfy constraints with respect to the moved neighbors, then the unmoved neighbors will be added to movement consideration lists and constraints between the element and these neighbors will be satisfied later.

For elements in the left (or right) candidate lists, only the sponsoring right (left) neighbor has been moved prior to movement consideration. Hence, for left and right candidates only one neighbor must be considered.

For top (or bottom) neighbors, it is possible that two neighbors, the sponsoring bottom (or top) neighbor and its left (or right) neighbor, were moved prior to consideration. However, it is shown here that when an element from the top candidate list satisfies deformation constraints relative to its sponsoring bottom neighbor, then it automatically satisfies the constraints of its left neighbor as long as the left neighbor was previously placed to satisfy its own bottom neighbor.

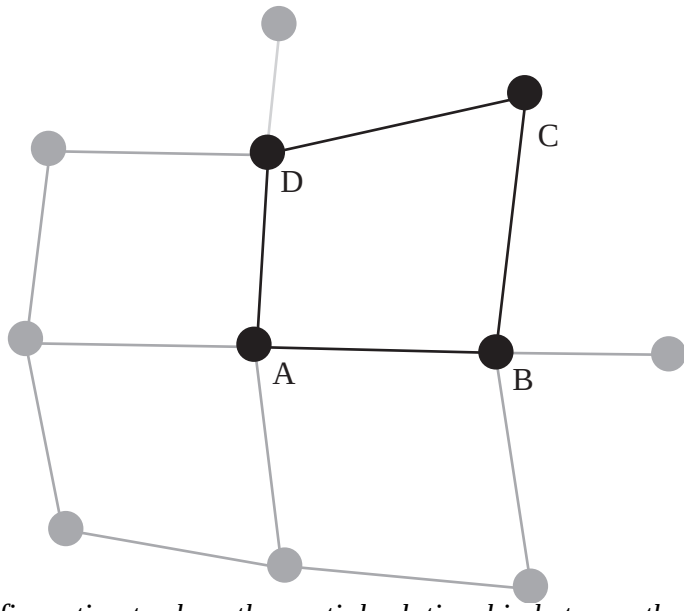


Figure A1. Grid configuration to show the spatial relationship between the point A, and its right neighbor, B, its top neighbor, D, and its right/top diagonal neighbor, C.

For the point, A, if the top and right neighbors, D and B respectively, satisfy the deformation constraints with respect to A (see Figure A1), then:

$$\begin{aligned}(x_B, y_B) &\in \left([x_A + \partial x_{\min}, x_A + \partial x_{\max}],\ [y_A - \partial y\_horiz_{\max}, y_A + \partial y\_horiz_{\max}]\right) \\ &\equiv \left([x_{\min_{B/A}}, x_{\max_{B/A}}],\ [y_{\min_{B/A}}, y_{\max_{B/A}}]\right), \text{ and}\end{aligned}$$

$$\begin{aligned}(x_D, y_D) &\in ([x_A - \partial x\_vert_{\max}, x_A + \partial x\_vert_{\max}],\ [y_A - \partial y_{\min}, y_A + \partial y_{\max}]) \\ &\equiv ([x_{\min_{D/A}}, x_{\max_{D/A}}],\ [y_{\min_{D/A}}, y_{\max_{D/A}}]).\end{aligned}$$

In the above expression, $x_{\min_{P/A}}$ means the minimum x value for the point P given the position of A, and $x \in [x_0, x_1]$ implies that the value of x lies between x_0 and x_1.

In addition, C, the top neighbor of B, satisfies the deformation constraints with respect to B:

$$\begin{aligned}(x_C, y_C) &\in ([x_{\min_{B/A}} - \partial x\_vert_{\max},\ x_{\min_{B/A}} + \partial x\_vert_{\max}],\ [y_{\min_{B/A}} - \partial y_{\min},\ y_{\min_{B/A}} + \partial y_{\max}]) \\ &= \left([x_A + \partial x_{\min} - \partial x\_vert_{\max},\ \ x_A + \partial x_{\max} + \partial x\_vert_{\max}],\right. \\ &\quad \left.[y_A - \partial y\_horiz_{\max} - \partial y_{\min},\ \ y_A + \partial y\_horiz_{\max} + \partial y_{\max}]\right) \\ &= \left([x_{\min_{D/A}} - \partial x_{\min},\ \ x_{\max_{D/A}} + \partial x_{\max}], [y_{\min_{D/A}} - \partial y\_horiz_{\max},\ \ y_{\max_{D/A}} + \partial y\_horiz_{\max}]\right).\end{aligned}$$

Hence, in addition to satisfying the distance constraints with respect to its bottom neighbor B, (x_C, y_C) also satisfies the deformation constraints with respect to the left neighbor, D. Therefor, when an element of a top candidate list is moved to satisfy constraints with respect to its bottom neighbor, it automatically satisfies constraints with respect to its previously considered left neighbor. Hence, for elements of the top candidate list, only one neighbor must be considered to satisfy both sets of constraints. A similar argument can be made for elements of the bottom candidate list.

8 Bibliography

[1] R. Avila and L. Sobierajski, “A haptic interaction method for Volume Visualization”, proc. IEEE Visualization’96, ed. R. Yagel and G. Nielson, pp. 197-204.1996.

[2] D. Baraff "Analytical methods for dynamic simulation of non-penetrating rigid bodies", proc. SIGGRAPH’89, pp. 19-28, 1989.

[3] D. Baraff and A. Witkin, “Dynamic simulation of non-penetrating flexible bodies”, proc. SIGGRAPH’92, pp. 303-308, 1992.

[4] M. Bro-Nielsen and S. Cotin, “Real-time volumetric deformable models for surgery simulation using finite elements and condensation” proc. Eurographics, ‘96, Vol. 15, pp. 57-66, 1996.

[5] M. Bro-Nielsen, “Fast finite elements for surgery simulation”, proc. Medicine Meets Virtual Reality - V, 1997.

[6] D. Chen, "Pump it up: computer animation of a biomechanically based model of muscle using the finite element method", PhD thesis, Media Arts and Sciences, MIT, 1991.

[7] J. Christensen, J. Marks, and T. Ngo, “Automatic motion synthesis for 3D mass-spring models”, The Visual Computer, Vol. 13, pp. 20-28, 1997.

[8] J. Cohen, M. Lin, D. Manocha, and K. Ponamgi, “I-COLLIDE: an interactive and exact collision detection system for large-scaled environments:, proc. Symposium on Interactive 3D Graphics, ACM Siggraph, pp. 189-196, 1995.

[9] J. Collier, B. Collier, G. O’Toole, and S. Sargand, “Drape prediction by means of finite element analysis”, J. of the Textile Institute, Vol. 82, pp. 96-107, 1991.

[10] S. Cotin, H. Delingette, J.M. Clement, L. Soler, N. Ayache, and J. Marescaux, "Geometrical and physical representations for a simulator of hepatic surgery", proc. Medicine Meets Virtual Reality IV, 1996.

[11] I. Essa, S. Scarloff, and A. Pentland, “A unified approach for physical and geometric modeling for graphics and animation”, proc. Eurographics, Vol. 11, pp. C129-138, 1992.

[12] Y. Fung, Biomechanics: mechanical properties of living tissues, Springer-Verlag, New York, 1993.

[13] T. Galyean and J. Hughes, “Sculpting: an interactive volumetric modeling technique”, proc. SIGGRAPH’91, pp. 267-274, 1991.

[14] S. Gibson, "Beyond volume rendering: visualization, haptic exploration, and physical modeling of element-based objects", in Visualization in Scientific Computing (proc. Eurographics workshop on ViSC), eds. R. Scateni, J. van Wijk, and P. Zanarini, Springer-Verlag, pp. 10-24, 1995.

[15] S. Gibson, “3D ChainMail: a fast algorithm for deforming volumetric objects”, proc. Symposium on Interactive 3D Graphics, ACM Siggraph, pp. 149-154, 1997.

[16] S. Gibson, C.Fyock, E. Grimson, T. Kanade, R. Kikinis, H. Lauer, N. McKenzie, A. Mor, S. Nakajima, T. Ohkami, R. Osborne, J. Samosky, A. Sawada, “Volumetric Object Modeling for Surgical Simulation”, accepted for publication, Med. Image Analysis, Dec., 1997.

[17] N. Greene, “Voxel space automata: modeling with stochastic growth processes in voxel space”, proc. SIGGRAPH’89, pp. 175-184, 1989.

[18] T. He and A. Kaufman, “Collision Detection for Volumetric Models”, Proc. IEEE Visualization’97, 1997.

[19] Y. Huang and J. Oliver, “NC milling error assessment and tool path correction”, proc. SIGGRAPH’94, pp. 287-294, 1994.

[20] I. Hunter, T. Doukoglou, S. Lafontaine, and P. Charette, "A teleoperated microsurgical robot and associated virtual environment for eye surgery", Presence, Vol. 2, pp. 265-280, 1993.

[21] A. Kaufman, "Efficient algorithms for 3D scan-conversion of parametric curves, surfaces, and volumes", proc. SIGGRAPH’87, pp. 171-179, 1987.

[22] A. Kaufman, Volume Visualization, IEEE Computer Society Press, Los Alamitos CA, 1991.

[23] A. Kaufman, Daniel Cohen, Ronald Yagel, "Volume Graphics", IEEE Computer, Vol 23, 7, pp. 51-64, 1993.

[24] R. Koch, M. Gross, F. Carls, D. von Buren, G. Fankhauser, Y. Parish, “Simulating facial surgery using finite element models”, SIGGRAPH’96, pp. 421-428, 1996.

[25] G. Kriezis, N. Patrikalakis, F. Wolter, “Topological and differential equation methods for surface intersections”, Computer-Aided Design, Vol 24(1), pp. 41-55, 1990.
[26] S. Krishnan and D. Manocha, “An efficient surface intersection algorithm based on the lower dimensional formulation”, ACM Trans. on Computer Graphics, Vol. 16(1), pp. 74-106, 1997.
[27] P. Lacroute and M. Levoy, “Fast volume rendering using a shear-warp factorization of the viewing transform”, proc. SIGGRAPH’94, pp. 451-457, 1994.
[28] Y. Lee, D. Terzopoulos, and K. Waters, “Realistic modeling for facial animation”, proc. SIGGRAPH’95, pp. 55-62, 1995.
[29] J. Lengyel, M. Reichert, B. Donald, and D. Greenberg, “Real-time robot motion planning using rasterization computer graphics hardware”, proc. SIGGRAPH’90, pp. 327-335, Dallas, TX, August, 1990.
[30] M. Lin, “Efficient collision detection for animation and robotics”, Ph.D. thesis, Dept. Electrical Engineering and Computer Science, U. California, Berkeley, 1993.
[31] W. Lorensen and H. Cline, “Marching Cubes: a high resolution 3D surface construction algorithm”, proc. SIGGRAPH’89, pp. 163-169, 1989.
[32] P-G. Maillot, “Three-dimensional homogeneous clipping of triangle strips”, in Graphics Gems, ed. J. Arvo, AP Professional, New York, 1991.
[33] B. Mirtich, J. Canny, "Impulse-based simulation of rigid bodies", proc. 1995 Workshop on Interactive 3D Graphics, pp. 181-188, April, 1995.
[34] A. Mor, S. Gibson, J. Samosky, “Interacting with 3-dimensional medical data: haptic feedback for surgical simulation”, proc. Phantom Usergroup Workshop, Sept. 1996.
[35] B. Naylor, “SCULPT: an interactive solid modeling tool”, proc. Graphics Interface’90, pp. 138-148, 1990.
[36] R. Osborne, H. Pfister, H. Lauer, N. McKenzie, S. Gibson, W. Hiatt, T. Ohkami, “EM-Cube: an architecture for low-cost real-time volume rendering”, proc. SIGGRAPH/Eurographics Workshop on Graphics and Hardware, pp. 131-138, 1997.
[37] A. Pentland, J. Williams, "Good vibrations: modal dynamics for graphics and animation", Computer Graphics, Vol 23, 3, pp. 215-222, July, 1989.
[38] L. Sobierajski and A. Kaufman, "Volumetric ray tracing", proc. Volume Visualization Symposium, Washington, DC, pp. 11-18, 1994.
[39] L. Sobierajski and A. Kaufman, "Volumetric radiosity", Technical Report 94.01.05, Computer Science, SUNY Stony Brook, 1994.
[40] D. Terzopoulos, J. Platt, A. Barr, K. Fleischer "Elastically deformable models", Proc. SIGGRAPH’87, pp. 205-214, July, 1987.
[41] D. Terzopoulos, J. Platt, K. Fleischer "Heating and melting deformable models (from goop to glop)”, proc. Graphics Interface’89, pp. 219-226, 1989.
[42] D. Terzopoulos and K. Fleischer "Modeling inelastic deformation: viscoelasticity, plasticity, fracture", Computer Graphics, Vol 22, 4, pp. 269-278, Aug., 1988.
[43] D. Terzopoulos, K. Waters, "Physically-based facial modeling, analysis, and animation", J. Visualization and Computer Animation, Vol. 1, pp. 73-80, 1990.
[44] X. Tu and D. Terzopoulos, “Artificial fishes: physics, locomotion, perception, behavior”, proc. SIGGRAPH’94, pp. 43-50, 1994.
[45] T. van Hook, “Real-time shaded milling display”, proc. SIGGRAPH’86, pp. 15-20, 1996.
[46] S. Wang and A. Kaufman, "Volume sculpting", ACM Symposium on Interactive 3D Graphics, Monterey, CA, pp. 151-156, April 1995.
[47] S. Wang and A. Kaufman, "Volume sampled elementization of geometric primitives", Proceedings Visualization '93, San Jose, CA, pp. 78-84, October 1993.
[48] K. Waters, "A Muscle model for animating three-dimensional facial expression”, proc. SIGGRAPH’87, pp. 17-24, 1987.