Fast Volume-Preserving Free Form Deformation Using Multi-Level Optimization

Gentaro Hirota Renee Maheshwari * Ming C. Lin

Department of Computer Science
University of North Carolina at Chapel Hill
{hirota, renee, lin}@cs.unc.edu
www.cs.unc.edu/~geom/ffd

Abstract

We present an efficient algorithm for preserving the total volume of a solids undergoing free-form deformation using discrete level-of-detail representations. Given the boundary representation of a solid and user-specified deformation, the algorithm computes the new node positions of the deformation lattice, while minimizing the elastic energy subject to the volume-preserving criterion. During each iteration, a non-linear optimizer computes the volume deviation and its derivatives based on a triangular approximation, which requires a finely tessellated mesh to achieve the desired accuracy. To reduce the computational cost, we exploit the multi-level representations of the boundary surfaces to greatly accelerate the performance of the non-linear optimizer. This technique also provides interactive response by progressively refining the solution. Furthermore, it is generally applicable to lattice-based free-form deformation and its variants. Our implementation has been applied to several complex solids. We have been able to achieve an order of magnitude performance improvement over the traditional methods.

Keywords: Free-Form Deformation, Physically Based Modeling,

1 Introduction

Engineering design and shape styling often require the capability to manipulate flexible objects, including bending, twisting, compressing or stretching parts of the model or its entire geometry. Several modeling techniques and design methodologies have been proposed to provide intuitive manipulation and interactive deformation of flexible objects [Bech94]. One of the most versatile and powerful tools for representing and modeling flexible objects is free-form deformation (FFD) introduced by Sederberg and Parry [SP86]. FFD unifies both the free-form surfaces and solid modeling into a common framework for deforming solid geometry, as well as surfaces, in a free-form manner. A more general extension to FFD (EFFD) was later presented by Coquillart [Coqu90, CP91].

However, none of these methods associates any physical constraints with geometric deformation of solids. Recently, the integration of geometric design and physically-based modeling techniques has emerged as an attractive alternative; it is a natural and systematic approach to constraint-based design, shape blending and a variety of solid modeling problems [Auma92, TQ94, QT95a, QT95b, RSB95, RSB96, AB97, GMP98]. The principles of physics govern the dynamic behaviors of objects in the physical world. Direct manipulation and interactive sculpting of geometric models should also be compliant with the laws of physics in order to give designers an intuitive grasp of the object. One of the important governing laws of Newtonian physics is the conservation of mass. When the density of a given material is constant, this implies the preservation of volume.

A volume-preservation constraint allows designers to keep the required relative proportionality of object sizes (in terms of their volumes) in the design of a complex assembly consisting of multiple parts. Another example is the design of a container whose capacity is given a priori. The volume inside should be preserved when the designer modifies the shape of the container. Volume-preserving free-form deformation is not only a useful tool for modeling, but also is a powerful visual aid in engineering animation and virtual prototyping as well. This technique can also be used to automatically create the standard squash and stretch effects in computer animation and help bring life to the animated characters.

Main Contribution: We present a new approach for volume-preserving free-form deformation using multi-level-of-detail representations. Our method has following characteristics.

- **Total Volume Preservation of *Embedded* Solids:** Given a boundary representation of solid geometry and user-specified constraints, the algorithm preserves the total volume of *embedded* solid geometry. The *hard* constraint of volume preservation is satisfied by an augmented Lagrangian method.
- **Capability of Handling *Large* deformations:** Our algorithm computes the new node positions of the deformation lattice

*Vital Images, Inc., reneem@vitalimages.com

by minimizing the elastic energy of deformation. Quadratic energy functions have often been used because they can linearize the minimization problem. Such linear behavior, however, defies the intrinsically non-linear nature of all large deformations. We use a spring network whose energy function simulates non-linear elasticity.

- **Efficiency and Interactivity:** Our algorithm computes the volume deviation and its derivatives based on an approximation method that requires a finely tessellated mesh to achieve the desired accuracy. To reduce the computational cost, we exploit the multi-level-of-detail representations of the boundary surfaces to provide progressive refinement while the user interactively manipulates the object and examines the deformed shape.
- **Versatility:** Our method is applicable to any models, including polyhedra and solids defined by NURBS or Bezier surfaces, as long as multi-LOD meshes for the model can be computed.

To the best of our knowledge, none of the previous methods for volume-preserving FFD have achieved these goals *simultaneously*.

Although our implementation is based on the trivariate Bernstein basis FFD, the proposed technique is generally applicable to other lattice-based free-form deformations and their variants [SP86, GP89, Coqu90, MJ96]. Our algorithm simulates the physical behaviors of the solids subject to the volume-preserving constraint. We have been able to achieve an order of magnitude speedup over the conventional optimization methods.

Organization: The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 explains the core of our method based on triangular approximation.. Section 4 describes a novel method which exploits multi-resolution representations to accelerate the optimization convergence and adapt this method to deform solids with curved boundaries. Section 5 presents our implementations and gives the performance results on several complex models. Section 6 concludes the paper with future research directions.

2 Related Work

2.1 Free-Form Deformation

An $R^3 \rightarrow R^3$ mapping was defined for deforming solids in [Barr84] and brief mention of deformation was made much earlier in [Sabin70, Bezier74]. Free-form deformation (FFD) was first formally proposed in [SP86] both as a representation for free-form solids and as a method for sculpturing solid models. One of many advantages of FFD is versatility, i.e. its general applicability to all representations of embedded geometry. Through a 3D parallelpiped lattice, the users can manipulate the geometry of the embedded object.

Griessmair and Purgathofer [GP89] utilized a trivariate B-spline representation for FFD. A general extension to FFD was proposed in [Coqu90, CP91], by allowing the combination of multiple general lattice structures to form arbitrary shaped spaces. A generalized deCasteljau approach to 3D free-form deformation based on [Barr84] was developed by Chang and Rockwood [CR94]. It allows the user to modify the axes defined as Bezier curves during the deformation, but restricts ways in which the surrounding spaces can be altered. MacCracken and Joy designed a FFD technique based on the Catmull-Clark subdivision methodology that successively refines a 3-dimensional lattice into a sequence of lattices converging uniformly to a region of 3D space [MJ96].

[BB91, HHK92] presented methods for direct manipulation of the deformed object, leading to better control of the deformation and a more intuitive user-interface. Given a user's selection of input points on the objects, these techniques automatically compute the necessary movement of the control points using a least-square formulation.

2.2 Physically-Based Deformation

In computational mechanics, finite element methods (FEM) have been widely used to simulate deformation [OP92, LeTal94, GL84, Donz95]. Application of FEM in computer animation can also be found in [GTT89]. FEM are usually geometry dependent. Elements are generated directly from solid models by meshing, whereas FFD is independent of the embedded geometry. The continuities across finite elements are usually just C^0 since higher order continuity is not essential for simulating elastic solids [OP92]. In geometric modeling or computer graphics applications C^1 or higher continuity is often desirable for aesthetic or manufacturing reasons. FFD can guarantee such continuity.

FEM is, however, compatible with FFD. In FEM, each element is a partition in the internal space of the solid model. In FFD, the deformation function consists of piecewise polynomial functions defined in similar partitions. We can use each 'partition' of FFD as an element. Such a combination of FEM and FFD is used in [RSB95, RSB96] and [FVT97] to simulate static and dynamic behaviors respectively. Each FFD lattice can be seen as an element with trivariate Bernstein polynomials for shape functions. An embedded object is approximated by an elastic unit cube or cubes aligned with the parameter space. Therefore, the physical behavior is independent of the embedded geometry in the simulation. Various numerical methods [OP92] in computational mechanics are used to integrate elastic energy inside heterogeneous materials. Such methods are applicable to integrate the energy function for the solid geometry embedded in FFD, but the computational cost is significantly higher.

2.3 Volume Preserving Deformation

There are three different definitions of volume preservation:

(a) Local volume is analytically preserved.
(b) Local volume is numerically preserved.
(c) Global volume is preserved.

(a) is the strongest condition. If a deformation function satisfies (a), the volume of any differential element (local volume) is constant. [SP86] implies that there is a special class of FFDs that belongs to this category. This condition is so strict that admissible deformation seems to encumber free user manipulation. (b) also implies local volume preservation, but in a much relaxed sense based on "weak formulation." This is a well-known technique which simulates incompressible materials by using FEM [GL84].

In this paper, we focus on (c). We are not concerned with the local volume change, but only the total volume of a solid (global volume). [RSB95,RSB96] proposed a method that preserves the volume of a unit cube in a deformation lattice. [AB97] was the first to show preservation of the total volume of an embedded solid, but deformable objects were limited to polyhedra. This technique uses a generalized direct manipulation FFD based on a

least-square energy function proposed by [BB91, Bech94] and is not ideal for very large deformations (see Section 3).

No algorithm published so far is capable of applying *large* deformations to *embedded* solid geometry of arbitrary topology with *curved boundaries* at *interactive* rates.

3 Mathematical Formulations

In this section, we present the mathematical formulation for volume-preserving free-form deformation using a triangular approximation method.

3.1 Deformation Function

We define a deformation function φ that transforms the original space into a deformed one via the transformation:

$$\varphi : \boldsymbol{x} \rightarrow \boldsymbol{x}^{\varphi} \ (= \varphi(\boldsymbol{x})) \tag{1}$$

We have chosen the original FFD [SP86] for the ease of discussion and demonstration. This method uses a set of node points (also called control points) $\boldsymbol{X}_I$ that deform the entire space that contains the object. The deformation is independent of the representation of the embedded geometry. The nodes present intuitive handles for interactive manipulation of the deformation. Using the control points, the deformation can now be described as:

$$\varphi(\boldsymbol{x}) = \sum_{I=1}^{n} \phi_I(\boldsymbol{x}) \boldsymbol{X}_I \tag{2}$$

where the function ϕ_I defines the scalar field that specifies the influence of the nodes in the space, and n is the number of nodes. We also define $\boldsymbol{X}$ as a $3 \times n$ matrix

$$\boldsymbol{X} = \left[\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots, \boldsymbol{X}_n\right]. \tag{3}$$

In our implementation, we use the trivariate tensor product Bernstein polynomials for ϕ_I:

$$\varphi(\boldsymbol{x}) = \sum_{i=0}^{n_u} \sum_{j=0}^{n_v} \sum_{k=0}^{n_w} B_i^{n_u}(u) B_j^{n_v}(v) B_k^{n_w}(w) \overline{\boldsymbol{X}}_{ijk} \tag{4}$$

where the $\overline{\boldsymbol{X}}_{ijk} (= \boldsymbol{X}_{i(n_v+1)(n_w+1)+j(n_w+1)+k+1})$ construct a 3D lattice of node points, $B_l^n(t)$ are the Bernstein polynomials, and (u, v, w) is the parameterization of the original position $\boldsymbol{x}^T$=(x, y, z):

$$\left[u \quad v \quad w\right] = \left[x \quad y \quad z \quad 1\right] \boldsymbol{A} \tag{5}$$

$\boldsymbol{A}$ is a 4 x 3 matrix, which represents an affine transformation. It is defined such that the FFD lattice encloses the volume being deformed.

3.2 Problem Definition

We assume that a triangle mesh can be obtained from the surface boundary of the embedded solid, either provided by the user or generated using a standard boundary tessellation algorithm. We also assume the deformation can be approximated by "per-vertex" mapping, where the points on each triangle are linearly interpolated *after* the mapping of triangle vertices. We will discuss the mathematical accuracy of this approximation in Section 4. The triangle mesh consists of m vertices, which are denoted by a $3 \times m$ matrix $\boldsymbol{P} = \left[\boldsymbol{P}_1, \boldsymbol{P}_2, \cdots, \boldsymbol{P}_m\right]$. $\boldsymbol{P}$ is mapped by the function φ that is defined by node points $\boldsymbol{X}$. Therefore the volume V of the deformed solid is a function V of $\boldsymbol{X}$. The volume deviation $\Delta \mathrm{V}(\boldsymbol{X})$ from the original shape is also a function of $\boldsymbol{X}$; i.e. $\Delta \mathrm{V}(\boldsymbol{X}) = \mathrm{V}(\boldsymbol{X}) - \mathrm{V}(\boldsymbol{X}_{org})$ where $\boldsymbol{X}_{org}$ denotes the original configuration of nodes before user manipulation.

We also define a deformation energy function $\mathrm{E}(\boldsymbol{X})$ of the solid. $\mathrm{E}(\boldsymbol{X})$ simulates the potential energy of elastic solids, and is a measure of the amount of deformation. The user can interactively deform the object by moving one or more node points. This operation may change the volume V as well as the energy $\mathrm{E}(\boldsymbol{X})$. Our goal is to find a new configuration of $\boldsymbol{X}$, which preserves the total volume of embedded geometry. We can formulate the problem as a constrained minimization, in which we search for the minimum energy configuration of the node points subject to the constraint of volume preservation:

$$\min \ \mathrm{E}(\boldsymbol{X}) \ \text{subject to} \ \Delta \mathrm{V}(\boldsymbol{X}) = 0 \tag{6}$$

The user normally would specify more constraints, namely, by pinning down the positions of several nodes.

3.3 Volume Computation

The total volume of the solid is computed by summing the volume contributions of each triangle of the polygonal mesh. Each contribution is the volume swept out by the triangle through its projection onto the x-y plane. This is shown in Figure 1.

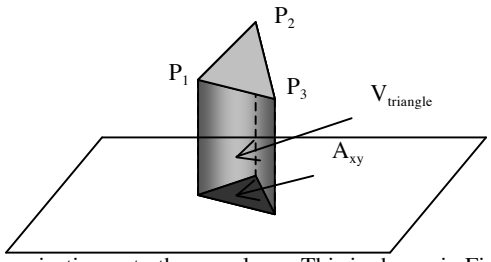


Figure 1: The volume contribution of a triangle is the volume swept out by the triangle through its projection onto the x-y plane.

This volume is calculated by multiplying the area of the projected triangle A_{xy} with the average height of the triangle as follows:

$$V_{triangle} = A_{xy} \left(P_{1,z} + P_{2,z} + P_{3,z} \Big/ 3 \right)_{\text{x-y plane}} \tag{7}$$

The area of the projection is found by

$$A_{xy} = \left(\tfrac{1}{2}(P_2 - P_1) \times (P_3 - P_1)\right)_z \tag{8}$$

which is positive in the case where the triangle faces upward, otherwise negative. The total volume V is then

$$V = \sum_i V_{i^{th} triangle} \tag{9}$$

where $V_{ith\ triangle}$ is the volume contribution of the i^{th} triangle. Note that we do not consider self intersection.

The volume deviation (ΔV) between the current and original states of the object can be measured simply by taking the difference between the two total volumes:

$$\Delta V = V - V_{org} \tag{10}$$

The derivatives of the volume deviation (w.r.t. the node vector $\boldsymbol{X}$) are also computed to facilitate the minimization process explained later. Using the chain rule:

$$\underset{(1\times 3n)}{\frac{\partial V}{\partial \boldsymbol{X}}} = \underset{(1\times 3m)}{\frac{\partial V}{\partial \boldsymbol{P}}}\ \underset{(3m\times 3n)}{\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{X}}} \tag{11}$$

The components of $\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{X}}$ can be computed as corresponding values of : ϕ_i in equation (2).

$\frac{\partial V}{\partial \boldsymbol{P}}$ can be computed by looking at the volume deviation contribution from each triangle (Figure 2).

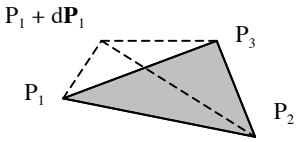


Figure 2: The volume deviation induced by the displacement $d\boldsymbol{P}_1$ of the vertex $\boldsymbol{P}_1$.

If a point $\boldsymbol{P}_1$ on a triangle moves a distance $d\boldsymbol{P}_1$, the volume deviation can be expressed as

$$dV = d\boldsymbol{P}_1 \bullet \left((\boldsymbol{P}_2 - \boldsymbol{P}_1)\times(\boldsymbol{P}_3 - \boldsymbol{P}_1)/6\right). \tag{13a}$$

Therefore

$$\frac{\partial V}{\partial \boldsymbol{P}_1} = (\boldsymbol{P}_2 - \boldsymbol{P}_1)\times(\boldsymbol{P}_3 - \boldsymbol{P}_1)/6. \tag{13b}$$

This is the contribution of a triangle to $\partial V \big/ \partial \boldsymbol{P}$. Again, the summation over all triangles gives the total value of the derivative.

3.4 Deformation Energy Function

The elastic deformation energy is a functional of the deformation function φ. Since φ is defined by the node points $\boldsymbol{X}$, the deformation energy is a function of $\boldsymbol{X}$. The choice of the deformation energy may appear to be insignificant since in the scope of physically-based modeling we are merely trying to incorporate *physically sound* behavior to the object so that users can manipulate it intuitively. But a poor choice of the deformation energy makes the behavior of deformation unpredictable. For large deformations, in particular, special care is required. The following discussion is based on literature of solid mechanics such as [Ciar88], [Ogde84], and [LeTal94]. [TPBF87] describes the similar theory using differential geometry terminology.

The elastic deformation energy measures the amount of deformation. The deformation is essentially local stretches in various directions. If the mapping φ: $x \rightarrow x^*$ is simply a rigid transformation, meaning that it preserves the distances between all particles (no stretches), the energy must be zero. The local deformation is governed by the *deformation gradient* $\boldsymbol{F}=\nabla\varphi$, a 3×3 matrix. The *right Cauchy-Green tensor* $\boldsymbol{C}=\boldsymbol{F}^{\mathrm{T}}\boldsymbol{F}$ measures the length of an elementary vector after deformation, and is insensitive to rigid body transformations.

Let **E** be the energy density function of an elastic solid under the deformation φ. The total energy is obtained by integrating **E** over the entire volume of the solid. The ***axiom of frame indifference*** states that **E** may not depend on the frame in which the deformation φ is observed [LeTal94]. A right Cauchy-Green tensor $\boldsymbol{C}$ contains all information in $\boldsymbol{F}$ except for rotation. Hence by the axiom of frame indifference, **E** can only be expressed as a function of the $\boldsymbol{C}$ for elastic materials unless zero E is allowed for a non-rigid transformation φ (spurious zero-energy mode).

In fact, the simplest law uses a quadratic function of the right Cauchy-Green tensor $\boldsymbol{C}$ [LeTal94]. Since $\boldsymbol{F}$ and $\boldsymbol{C}$ are a linear and quadratic functions of $\boldsymbol{X}$ respectively, **E** is at least a quartic function of $\boldsymbol{X}$. Although quadratic energy functions of $\boldsymbol{X}$ have been used in many direct manipulation FFD methods[FVT97, HHK92, RSB95, RSB96] and analysis of small deformation [OP92], they are unsuitable for large deformations because quadratic functions are either allowing spurious zero-energy mode or they are *not* frame indifferent.

We have chosen the energy function of a spring network that connects 14 neighboring nodes in the FFD lattice. In the color plates of our examples. Nodes are shown as cubes; springs are drawn as line segments between those nodes. The energy function can be written as:

$$\mathrm{E}_{\mathrm{spring}}(\boldsymbol{X}) = \sum_j \frac{k}{2}\left(\sqrt{(\boldsymbol{X}_{e_j} - \boldsymbol{X}_{s_j})\cdot(\boldsymbol{X}_{e_j} - \boldsymbol{X}_{s_j})} - L_j\right)^2 \tag{14}$$

where j is the index of a spring, s_j and e_j are the indices of nodes which are connected by the spring. L_j is the natural length of the spring.

Despite its simplicity, this spring energy function is *frame-indifferent* because only the distances between nodes affect the energy. A rigid body transformation does not have any effect on the function. Spurious zero-energy modes are not allowed for non-rigid transformations, either. Recall that the essence of deformation is the change of distances between particles. The spring network captures this essence. Interestingly, the polynomial degree of $\mathbf{E}_{\mathbf{spring}}$ is well over quartic (infinity). Therefore, the spring-network is capable of handling any large deformation. The spring network presents an intuitive metaphor for users, and its physical behavior is quite predictable. The drawback of the spring-network, however, is that it overestimates the energy because the actual deformation of the object is smaller than the deformation of the FFD lattice. We also have to emphasize that this energy function has no connection to the embedded geometry. The deformation may not be exactly what the user would expect from the shape of the solid model.

The minimization algorithm described later requires the first derivative of the energy function, which can be easily obtained by the partial differentiation of $\mathbf{E}_{\mathrm{spring}}(\boldsymbol{X})$.

3.5 Numerical Method for Constrained Minimization

Let us now restate our constrained minimization problem:

$$\begin{aligned} &\min\ \mathrm{E}_{spring}(\mathbf{X}) \\ &\text{subject to } \Delta\mathrm{V}(\mathbf{X}) = 0 \end{aligned} \tag{15}$$

The problem can be converted to a saddle point finding problem:

$$\max_{\lambda}\min_{\boldsymbol{X}}\ L \tag{16}$$

where L is called *Lagrangian*, which is in the form of

$$L = \mathrm{E}_{spring}(\boldsymbol{X}) - \lambda\ \Delta \mathrm{V}(\boldsymbol{X}). \tag{17}$$

λ is an unknow parameter called *Lagrange Multiplier*. The solution satisfies two conditions:

$$\begin{aligned} \frac{\partial L}{\partial \lambda} &= \Delta \mathrm{V}(\boldsymbol{X}) = 0 \\ \frac{\partial L}{\partial \boldsymbol{X}} &= \frac{\partial}{\partial \boldsymbol{X}} \mathrm{E}_{\text{spring}}(\boldsymbol{X}) - \lambda \frac{\partial}{\partial \boldsymbol{X}} \mathrm{V}(\boldsymbol{X}) = 0 \end{aligned} \tag{18}$$

Here note that

$$\frac{\partial}{\partial \boldsymbol{X}} \Delta \mathrm{V}(\boldsymbol{X}) = \frac{\partial}{\partial \boldsymbol{X}} \mathrm{V}(\boldsymbol{X})$$

The first condition coincides with the original constraint. The second condition corresponds to the resulting influence by both the spring forces and the volume preserving forces. These volume preserving forces can be seen as the hydrostatic pressure concentrating on node points. The Lagrange Multiplier scales the hydrostatic pressure properly against the spring forces to reach an equilibrium point between the two.

For computational efficiency, we use a slightly more complex *Augmented Lagrangian* method [Flet87] which is widely used for solving mechanical engineering problems [GL84, Donz95]. Figure 3 illustrates the adaptation of the algorithm to our specific problem. Here, we explain our algorithm in a rather informal way by using associated physical concepts.

First the Lagrangian L is augmented with a penalty term $\frac{\sigma}{2}(\Delta \mathrm{V}(\boldsymbol{X})^2)$, which penalizes against volume deviation. This term simulates the stored energy of a fictitious compressible material similar to air. The coefficient σ scales the penalty term appropriately in the optimization process. The augmented Lagrangian L_a and its derivative are

$$\begin{aligned} L_a &= \mathrm{E}_{spring}(\boldsymbol{X}) + \frac{\sigma}{2}(\Delta \mathrm{V}(\boldsymbol{X}))^2 - \lambda\ \Delta \mathrm{V}(\boldsymbol{X}) \\ \frac{\partial L_a}{\partial \lambda} &= \Delta \mathrm{V}(\boldsymbol{X}) \\ \frac{\partial L_a}{\partial \boldsymbol{X}} &= \frac{\partial}{\partial \boldsymbol{X}} \mathrm{E}_{\text{spring}}(\boldsymbol{X}) + \sigma \Delta \mathrm{V}(\boldsymbol{X}) \frac{\partial \mathrm{V}(\boldsymbol{X})}{\partial \boldsymbol{X}} \\ &\quad - \lambda \frac{\partial}{\partial \boldsymbol{X}} \mathrm{V}(\boldsymbol{X}) \end{aligned} \tag{19}$$

The gradient of the penalty term $\sigma \Delta \mathrm{V}(\boldsymbol{X})\, \partial \mathrm{V} / \partial \boldsymbol{X}$ can be viewed as an approximated "air pressure" acting on node points.

λ is initialized to be zero, therefore there is no hydrostatic pressure which preserves the volume at the beginning. Instead, the "air pressure" works against compression or decompression of the volume. This makes the problem an unconstrained minimization problem because the penalty term does not impose a hard constraint. This unconstrained nonlinear minimization can be solved by typical gradient descent algorithms [PTVF92]. The minimization process converges to an equilibrium node configuration $\boldsymbol{X}$, which balances between the spring forces and the penalty forces due to air pressure. As a result, the volume deviation ΔV is reduced. If ΔV is not smaller than the tolerance ($\boldsymbol{\varepsilon}$, specified by the user), the unconstrained minimization is applied again. But in the second round, we have a good guess for the hydrostatic pressure, which is the air pressure. Therefore we can update λ in such a way that the constraint forces (the hydrostatic pressure) match the penalty forces (the air pressure) i.e.

$$\sigma \Delta \mathrm{V}(\boldsymbol{X}^{(1)})\, \partial \mathrm{V} / \partial \boldsymbol{X} = -\lambda^{(1)}\, \partial \mathrm{V} / \partial \boldsymbol{X}. \tag{20}$$

We can eliminate the common factor $\partial \mathrm{V} / \partial \boldsymbol{X}$ and obtain

$$\sigma \Delta \mathrm{V}(\boldsymbol{X}^{(1)}) = -\lambda^{(1)}. \tag{21}$$

Now, both the air and the hydrostatic pressures are in effect, which further reduces ΔV. In the second round, λ is updated in such a way that both of the air pressure and the hydrostatic pressure match the new hydrostatic pressure (see "Lagrange Multiplier Update" in Figure 3). This process is repeated until ΔV reaches a value below the user specified tolerance.

The convergence of this method can be accelerated by increasing the penalty scalar σ. However, an excessively large value of σ leads to numerical instability. We use an internal feed-back loop which enforces the linear convergence of the algorithm. The loop monitors the convergence rate of ΔV, and doubles σ until ΔV is reduced to half of the value in the previous iteration [Powe69]. The advantage of this method is that we do not have to know the proper value of σ a priori. The algorithm automatically finds the minimum value of σ that guarantees linear convergence. In practice, however, this enforcement loop sometimes increases the value of σ to be undesirably large, which jeapordizes the accuracy of the unconstrained minimization. In our implementation, we set an upper bound for σ to avoid the adverse effects.

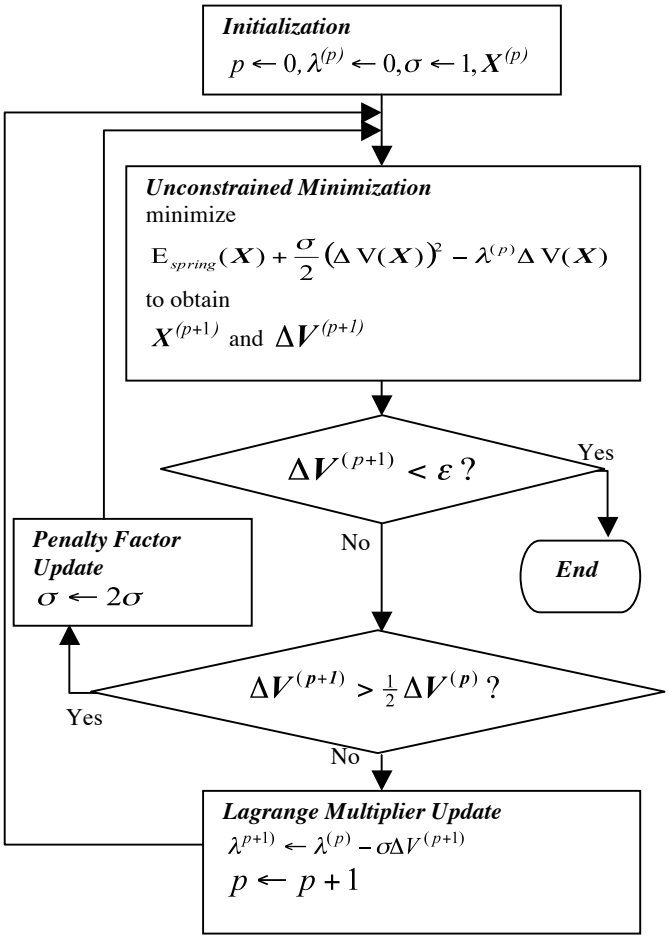


Figure 3: Constrained Minimization by *Augmented Lagrangian Method*. The Penalty Factor Update enforces linear convergence.

4 Multi-Level Refinements

Most solid objects we encounter in solid modeling and engineering animation have curved boundary surfaces. The method described above does not handle such objects directly. In this section, we present a novel approach that extends the computational framework in Section 3 to general objects using multi-level-of-detail representations.

4.1 Computation of Volume as Numerical Integration

Let us first examine the complexity of computing the volume enclosed by a deformed curved boundary. For the purpose of illustration, we use a parametric surface in this discussion. However, the basic argument is applicable to other types of surfaces as well.

The general form of a parametric surface is given as

$$\{\boldsymbol{x} \mid \boldsymbol{x}^T = [\mathrm{f}(s,t), \mathrm{g}(s,t), \mathrm{h}(s,t)]^T, (s,t) \in S\}, \quad (22)$$

where **f**, **g**, and **h** are scalar functions, and S is the domain of the surface in parameter space s-t. The volume enclosed by this surface (before deformation) is (according to [GMP98]):

$$V_{org} = \int_S z\, n_z\, ds dt, \quad (23)$$

where n is the normal vector on the surface (faces outside of the solid), and n_z is its z component. After the deformation φ: $\boldsymbol{x} \rightarrow \boldsymbol{x}^{\varphi}$, the surface becomes

$$\boldsymbol{x}^{\varphi} = \begin{bmatrix} x^{\varphi} \\ y^{\varphi} \\ z^{\varphi} \end{bmatrix} = \begin{bmatrix} \varphi_x([\mathrm{f}(s,t) & \mathrm{g}(s,t) & \mathrm{h}(s,t)]^T) \\ \varphi_y([\mathrm{f}(s,t) & \mathrm{g}(s,t) & \mathrm{h}(s,t)]^T) \\ \varphi_z([\mathrm{f}(s,t) & \mathrm{g}(s,t) & \mathrm{h}(s,t)]^T) \end{bmatrix} \quad (24)$$

The corresponding volume is

$$V = \int_S z^{\varphi} n_z^{\varphi} ds dt = \int_S z^{\varphi} \left(\frac{\partial x^{\varphi}}{\partial s} \frac{\partial y^{\varphi}}{\partial t} - \frac{\partial x^{\varphi}}{\partial t} \frac{\partial y^{\varphi}}{\partial s} \right) ds dt \quad (25)$$

As we saw in section 3.1, φ is defined as a linear combination of column vectors of $\boldsymbol{X}$. Therefore the volume function $\mathrm{V}(\boldsymbol{X})$ is a cubic function of $\boldsymbol{X}$. Next, we will examine the practicality of pre-computing this function for an arbitrary value of $\boldsymbol{X}$. Suppose we are using tricubic Berstein polynomial FFD, each of x^{φ} y^{φ} and z^{φ} has 4^3=64 terms. Therefore, after expansion, $\mathrm{V}(\boldsymbol{X})$ has 64^3(≅260,000) independent terms, whose coefficients must be integrated over S independently. The degrees, in terms of s and t, of the function to be integrated are far higher than the degrees of **f**, **g**, and **h** (3^3 =27 times higher), which makes the pre-computation very unattractive. But the fundamental problem of this approach is that the domain of integration S often has very irregular boundaries (e.g. trimming curves). An on-the-fly numerical integration, on the other hand, seems to be a practical solution.

4.2 Error Estimation of Triangular Approximation Method

Now we have to determine what kind of numerical integration method to use. If we look at the Equation (25) carefully, we find $n_z^{\varphi} ds\, dt$ is the area of a differential element $dsdt$ projected onto the $x^{\varphi} - y^{\varphi}$ plane and z^{φ} is the height of the element. Comparing this with Figure 1, Equation (7) and (8), we see the triangular approximation as a discretized version of integration (25). Since it is a quadrature method based on piecewise linear approximation, the total error is $O(h^2)$ where h is a sampling interval. The number of samples m (which is the number of the vertices of triangles) is $O(1/h^2)$ because the sampling happens in the 2D domain S. It implies that the integration error is $O(1/m)$, hence $O(1/N_{Tri})$ where N_{Tri} is the number of triangles. *If we want to reduce the error to half, we have to double the number of triangles.*

The accuracy of the constrained minimization depends on the accurate evaluation of the constraint function

$$\Delta \mathrm{V}(\boldsymbol{X}) = \mathrm{V}(\boldsymbol{X}) - \mathrm{V}(\boldsymbol{X}_{org}). \quad (26)$$

The approximated version is written as

$$\Delta \widetilde{\mathrm{V}}(\boldsymbol{X}; M^{(N_{Tri})}) = \widetilde{\mathrm{V}}(\boldsymbol{X}; M^{(N_{Tri})}) - \widetilde{\mathrm{V}}(\boldsymbol{X}_{org}; M^{(N_{Tri})}). \quad (27)$$

where $M^{(N_{Tri})}$ denotes the approximation of the boundary surface with N_{Tri} triangles. The integration error is

$$
\begin{aligned}
\mathrm{R}(\boldsymbol{X}; M^{(N_{Tri})}) &= \Delta\widetilde{\mathrm{V}}(\boldsymbol{X}; M^{(N_{Tri})}) - \Delta\mathrm{V}(\boldsymbol{X}) \\
&= (\widetilde{\mathrm{V}}(\boldsymbol{X}; M^{(N_{Tri})}) - \mathrm{V}(\boldsymbol{X})) \\
&\quad - (\widetilde{\mathrm{V}}(\boldsymbol{X}_{org}; M^{(N_{Tri})}) - \mathrm{V}(\boldsymbol{X}_{org})) \qquad (28) \\
&= O(1/N_{Tri}) \\
&\approx C/N_{Tri}
\end{aligned}
$$

The constant C can be roughly estimated by two computations of the volume with different numbers of triangles:

$$
\begin{aligned}
&\mathrm{V}(\boldsymbol{X}; M^{(N_{Tri})}) - \mathrm{V}(\boldsymbol{X}; M^{(2N_{Tri})}) \\
&= \mathrm{R}(\boldsymbol{X}; M^{(N_{Tri})}) - \mathrm{R}(\boldsymbol{X}; M^{(2N_{Tri})}) \qquad (29) \\
&\approx C/N_{Tri} - C/2N_{Tri} = C/2N_{Tri}
\end{aligned}
$$

We can carefully select N_{Tri} such that $|C/N_{Tri}|$ is well below a given tolerance. We can compute $\Delta\mathrm{V}(\boldsymbol{X})$ with reasonable confidence, unless the deformed surface is highly discontinuous. The obvious problem of the triangular approximation is that it requires relatively high tessellation to achieve high accuracy. However, a large value of N_{Tri} slows down the evaluation of $\Delta\mathrm{V}(\boldsymbol{X})$ significantly.

4.3 Multi-Level Optimization

We propose a Multi-Level Optimization algorithm to alleviate the problem of the expensive $\Delta\mathrm{V}(\boldsymbol{X})$ evaluation. The algorithm is designed based on the following observations:

a) The number of steps (N_{step}) required for the unconstrained minimization (see Figure 3) is greatly affected by the initial guesses of $\boldsymbol{X}$, σ, and λ.

b) There is no significant relationship between the number of steps N_{step} and the number of triangles N_{Tri}. In other word, the minimization requires about the same number of steps regardless of the resolution of triangle mesh.

c) Each step of the minimization requires the re-computation of the deformed vertex positions, the volume deviation, and its derivatives, while the computation of the spring energy and its derivatives is relatively inexpensive. Therefore, the computational cost of each step is proportional to the number of vertices (m), and also N_{tri}.

a) and c) imply that we should avoid applying the minimization with bad initial guesses when the resolution of triangle mesh is high (i.e. N_{Tri} is large). Before we use a fine tessellation (large N_{Tri}) to achieve high accuracy, we need a good initial guess. Suppose we have a coarser mesh, i.e. an approximation of the same model with fewer triangles. The behavior of the $\Delta\mathrm{V}(\boldsymbol{X})$ for the coarser mesh is similar to that of that for a fine mesh. By applying the minimization to the coarser mesh, we can obtain an approximate solution of X, which can be used as a "good" initial guess for the minimization to the fine mesh. (b) and (c) imply that the minimization for the coarser mesh requires less time. The minimization for the fine mesh converges faster because of the improved initial guess. Thus we can expect that this two-level optimization is faster than one-level optimization. If it is the case, we can also expect that the minimization for the coarser mesh can be accelerated by a good initial guess obtained by applying minimization to an even coarser mesh. Thus, given multiple levels of mesh representation, we can successively refine the optimization.

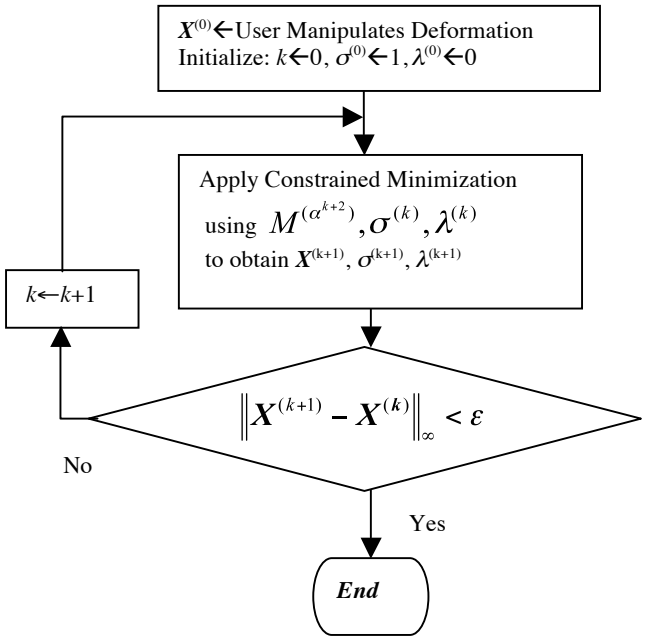


Figure 4: Multi-Level Optimization Algorithm

This "multi-level boot-strapping" process is illustrated in Table 2. The user moves the selected nodes (light ones) to new "fixed" positions. By this manipulation, a torus (see Plate 1) is compressed. As a result, the volume is reduced. Now, the volume-preserving energy minimization is invoked for the lowest resolution mesh. The first minimization run converges quickly despite its large N_{step} value of 90, due to a coarse tessellation with a very small N_{Tri} value of 16. The resulting parameters $\boldsymbol{X}^{(1)}$, $\sigma^{(1)}$, and $\lambda^{(1)}$ are fed to the next level of minimization with N_{tri}=64, which converges after much fewer steps (N_{step}=15). Note that, at this point, $\boldsymbol{X}$ is already very close to the final position (The node moves at most 3.7% of the object size after this level.) shown in the bottom row of the table**.** This progressive nature of the multi-level optimization method enables fast visual feedback to the user, which facilitates interactive manipulation of the solid model. The algorithm monitors the changes of $\boldsymbol{X}$ between successive levels. If the maximum difference among $\boldsymbol{X}$'s components is less than a tolerance ε (0.1% of the object size in our implementation), the algorithm assumes the optimization reached convergence. In this particular example, the final volume deviation ΔV is 0.05% of the original volume V_{org}. The flowchart of the algorithm is given in **Figure 4**.

In Figure 4, α is the rate of refinement between levels. α must be significantly larger than one. Otherwise, two successive levels will have about the same numbers of triangles, and the X computed at both levels may have very close values, which would results in a premature convergence. We empirically picked a value of 4 for α, which means that, at each refinement, the resolution of triangles doubles in each direction of the surface parameter space.

To verify the superior performance of multi-level optimization, we also applied the minimization process directly to the finest mesh (with N_{Tri}=4096 in this example) without multi-level-of-detail representations. The total computation cost for this run was more than 10 times higher than that of the multi-level optimization method. Note that if we did not use the multi-level optimization method, we would have to use a finer mesh based on a conservative error estimation, making the total computation cost even higher. More examples are presented in section 5..

Multi-level meshes can be generated by tessellating curved or polygonal boundary surfaces. If a finely tessellated mesh is available, one can also generate multi-level meshes by using various simplification algorithms [CVM+96,CMO97,GH97] preferably with local and global errors so that surface deviation of simplified model from the original one is minimized. In any case, the generated mesh may not be self-intersecting, a condition which is not handled properly by our algorithm. Note also that it is no longer a good idea to approximate a large flat area by a polygon because a deformation turns the polygon into a curved surface. A relatively uniform distribution of triangles is desirable.

5 Implementation and Performance

5.1 Implementation

We have successfully implemented our algorithm in C++. The image generation and the user interface are developed using the OpenGL and GLUT library. All timing data was obtained on an SGI Onyx2 with 195MHz R10000 MIPS processors.

5.2 Results

We have tested our system on various models undergoing deformation. The color images are available at our web site (http://www.cs.unc.edu/~geom/ffd). We analyze its performance using four examples here.

1) **Torus compression** (Plate 1 and Table 2): This example is described in section 4. A torus (left image) is deformed by a 3×3×3 lattice. The upper and lower 3×3 nodes in light color are vertically moved toward the center, compressing the torus (center image) and then fixed in their places. The algorithm finds the new positions of the central nodes to regain the original volume (right image). The multi-level meshes are generated by uniformly sampling the parameter space.
2) **Torus stretch** (Plate 2): The same torus is now stretched by using a 5×2×2 lattice (center image). The upper and lower 2×2×2 nodes in light color are fixed to enforce strong stretching at both ends. After the optimization process, the central 2×2 nodes have moved all the way across to the diagonal sides to produce lateral contraction.
3) **Rounded Club Partial Compression** (Plate 3): A club with rounded ends (left image) is deformed by a 5×2×2 lattice. The left side of the club is compressed (center image) and the right side swells to compensate the volume lost by the compression. The club is modeled as a sphere deformed by the FFD. The multi-level meshes are generated by a polygonal simplification algorithm [GH97] applied on a densely sampled sphere. The number of triangles at each level is approximately the same as for the torus in previous example.
4) **Cow Bending** (Plate 4): A cow model is bent by a 5×2×2 lattice. The rather unnatural aspect of the user-deformed cow (center image) is "improved" by our algorithm (right image). The cow model has 5804 triangles and multi-level meshes are generated in the same way as for the rounded club.

Figures 5,6,7, and 8 illustrate the convergence of the multi-level optimization. The x-axis show cumulative CPU time, and the y-axis shows the maximum error of node positions estimated as the distance deviation between the nodes before and after the optimization. The distance is normalized by the size of the object. In all our examples the node points converge to the approximately correct positions in a fraction of a second. These results demonstrate that our method can be used for interactive applications. In our implementation, the intermediate states of nodes and objects are rendered during the unconstrained minimization iterations, thus providing even faster feedback to the user.

Table 1 shows the performance improvement of the multi-level optimization over the usual optimization method. We have consistently achieved about an order of magnitude speedup using the multi-level optimization. The torus stretch example has an interesting property that is worth mentioning. If we start the algorithm from a fine mesh (N_{Tri}≥4096), the shape converges to a different configuration than it does if we start with a coarse mesh (Plate 6). This is not surprising because there are often multiple local minimum states in near symmetric constraints. Our method seeks only a local minimum, therefore the result may not be consistent for different starting levels. Plate 7 is an example of very large deformation. The rounded club is bent almost 180°. Our algorithm handles these situations without numerical problems. In Plate 5, The bottom half of a water pitcher model (6176 triangles) is compressed by the user. As a result of the optimization, the upper part is expanded.

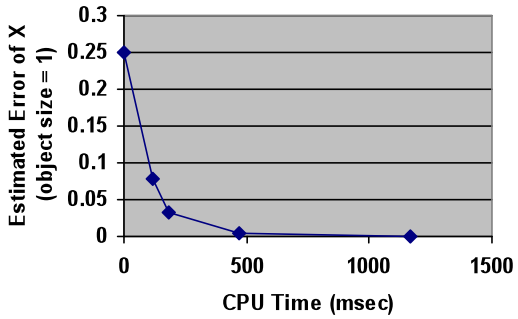


Figure 5: Convergence Curve for Torus compression.

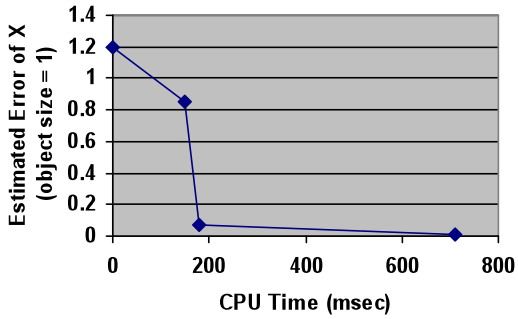


Figure 6: Convergence Curve for Stretched Torus.

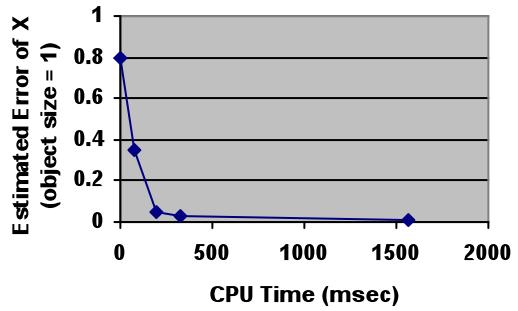


Figure 7: Convergence Curve for Rounded Club Partial Compression.

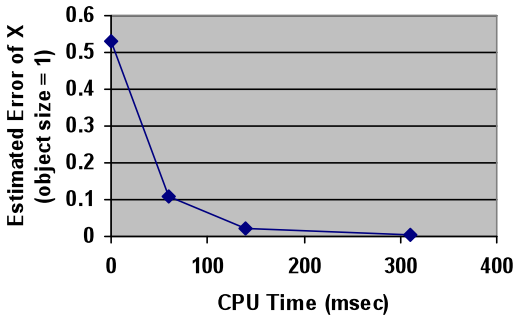


Figure 8: Convergence Curve for Bending Cow.

Example	T_1	T_2	T_1/T_2	N_{Tri}	$\Delta V/V_{org}$
Torus Compression	21000	1760	12	4096	0.05%
Torus Stretch	677400	11080	61	65536	0.08%
Club Partial Compressioin	40990	3530	12	16384	0.07%
Cow Bending	15810	2240	7	5804	0.07%

Table 1: Performance Improvement by Multi-Level Optimization. T_1 and T_2 are the total CPU time (msec) without and with Multi Level Optimization respectively. $\Delta V/V_{org}$ is the final relative volume deviation after Multi-Level Optimization. N_{Tri} is the mesh size at the terminal level.

6 Conclusion and Future Work

We have presented an efficient algorithm for volume-preserving free-form deformation using multi-level-of detail representations and a triangular approximation. This method is capable of large deformation, efficient, versatile. It gives designers and engineers real-time visual feedback and an intuitive physical feel of free-form solids, during geometric design and shape modification. We are currently improving our system by adding linear constraints to enable direct manipulation and continuity preservation. We also plan to extend this technique to the preservation of inertia tensor and center of mass. Higher order integration methods such as Gauss quadrature should also be investigated.

7 Acknowledgements

We are grateful to Army Research Office, National Science Foundation and Intel for their support. We would also like to thank Kenneth E. Hoff III for providing us an interactive FFD package, and Andrei State for technical advice.

8 References

[AB97] F. Aubert and D. Bechmann, Volume-preserving space deformation, *Computer & Graphics*, 21(5), pp. 625-639, 1997.
[Auma92] G. Aumann, Two algorithms for volume-preserving approximation of surfaces of revolution. *Computer-Aided Design*, 24(12), pp. 651-657, 1992.
[Barr84] A. Barr. Global and local deformations of solid primitives. *ACM Computer Graphics*, vol. 18, pp. 21--30, 1984.
[Bech94] D. Bechmann, Space Deformation Models Survey. *Computer & Graphics*, 18(4), pp. 571-586, 1994
[BB91] P. Borrel and D. Bechmann, Deformation of n-dimensional objects. *Intl. Journal of Computational Geometry and Applications*, 1(4), 1991. Also in *ACM Symposium on Solid Modeling*, pp. 351-370, 1991.

[Bezier74] P. Bezier, Mathematical and practical possibilities of UNISURF. in *Computer Aided Geometric Design*, Barnhill and Riesenfeld, eds., Academic Press, pp. 127-152, 1974.
[BF93] R. L. Burden, J. D. Faires, Numerical Analysis 5th ed.,. PWS Publishing Company, 1993.
[Ciar88] P. G Ciarlet., Mathematical Elasticity. North-Holland, 1988.
[CR94] Y. Chang and A. Rockwood, A generalized deCasteljau approach to 3D free-form deformation. *Computer Graphics (Proc of SIGGRAPH'94)*, pp. 257-260, 1994.
[CVM96+] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks and W. Wright, "Simplification Envelopes". *Proc. of ACM SIGGRAPH'96*, pp. 119-128, 1996.
[CMO97] J. Cohen, D. Manocha and M. Olano, "Simplifying polygonal models using successive mappings". *Proc. of IEEE Visualization'97*, pp.395-402, 1997.
[Coqu90] S. Coquillart, Extended free form deformation: a sculpturing tool for 3D geometric design. *Computer Graphics(Proc. of SIGGRAPH'90),* 24(4), pp. 187-193, 1990
[CP91] S. Coquillart and P. Jancene, Animated free-form deformation: An interactive animation technique*. Computer Graphics (Proc. of SIGGRAPH'91)*, vol. 25, pp. 23-26, 1991
[Deca96] P. Decaudin, Geometric Deformation by Merging a 3D Object with a Simple Shape. *Graphics Interface'96*, p 55--60. Toronto, Canada, pp. 23-26 , 1996
[Donz95] P. S. Donzelli A Mixed-Penalty Contact Finite Element Formulation for Biphasic Soft Tissues, PhDThesis, Dept. of Mechanical Engineering, Aeronautical Engineering and Mechanics, Rensselaer Polytechnic Institute, Troy, NY, 1995.
[Flet87] R. Fletcher, Practical Methods of Optimization 2nd ed.,. PWS Publishing Company, 1993.
[FVT97] P. Faloutsos, M. van de Panne and D. Terzopolous, Dynamic Free-Form Deformations for Animation Synthesis. *IEEE Trans. on Vis. and Computer Graphics*, 3(3), pp. 201-214, 1997.
[GH97] M. Garland and P. S. Heckbert, Surface Simplification Using Quadric Error Metrics. *SIGGRAPH'97*, 1997, pp. 209--216
[GMP98] C. Gonzales-Ochoa, S. McCammon and J. Peters, Computing moments of objects enclosed by piecewise polynomial surfaces. *ACM Trans. on Graphics*, 17(3), pp. 143-157, 1998.
[GL84] R. Glowinski and P. Le Tallec, Numerical solution of problems in incompressible finite elasticity by augmented Lagrangian methods II. Three-dimesional problems. *SIAM Journal on Applied Mathematics*, 44(4), pp. 710-733, 1984.
[GP89] J. Griessmair and W. Purgathofer, Deformation of solids with trivariate B-splines. Eurographics'89, pp.134-148.
[GTT89] J. Gourret, N.M. Thalmann, D. Thalmann. Simulation of Object and Human Skin Deformations. *SIGGRAPH'95*, 1989, pp. 21--30.
[HHK92] W. S. Hsu, J. F. Hughes and H. Kaufman, Direct Manipulation of Free-Form Deformations. *Computer Graphics*, 26(2), pp. 177-184, 1992.
[KR91] A. Kaul and J. Rossignac. Solid-interpolating deformations: construction and animation of PIPs. In *Proc. Eurographics*, 1991, pp. 493--505.
[LCSW95] S. Lee, K. Chwa, S. Shin, and G. Wolberg. Image Metamorphosis Using Snakes and Free-Form Deformations. *SIGGRAPH 95 Conference Proceedings*, pp. 439--448, 1995.
[LeTal94] P. Le Tallec, Numerical methods for solids. in *Handbook of Numerical Analysis,* Ciarlet and Lions, eds., North-Holland, 1994.
[MJ96] R. MacCracken and K. Joy, Free-form deformation with Lattices of arbitrary topology. *Computer Graphics (Proc. of SIGGRAPH'96)*, pp. 181-188, 1996.
[Ogde84] R. W. Ogden, Non-Linear Elastic Deformation,. Dover Publications, Inc., 1984.
[OP92] N. Ottosen and H. Petersson, Introduction to Finite Element Method. Prentice Hall, 1992.
[Parent95] R. Parent. Implicit Function Based Deformations of Polyhedral Objects. In *Implicit Surfaces '95*, 1995.
[[Powe69] M. J. D. Powell, A method for nonlinear constraints in minimization problems, in *Optimization,* R. Fletcher, eds., Academic Press, London, 1969.
[PTVF92] W. H. Press, S. A. Teukolsky, W. T. Vettering, B. P. Flannery, Numerical Recipes in C, 2nd ed., Cambridge, 1994.
[QT95a] H. Qin and D. Terzopoulos, Dynamic NURBS swung surfaces for physics-based shape design. *Computer Aided Design*, 27(2), 1995.
[QT95b] H. Qin and D. Terzopolous, Dynamic manipulation of Triangular B-Splines, *Proc. of ACM Symposium on Solid Modeling'95*, pp. 351-360, 1995.
[RSB95] A. Rappoport, A. Sheffer and M. Bercovier, Volume-preserving free-form solids. *ACM Symposium on Solid Modeling'95*, pp. 361-370, 1995.
[RSB96] A. Rappoport, A. Sheffer and M. Bercovier, Volume-preserving free-form solids. *IEEE Trans. on Vis. and Computer Graphics*, 2(1), pp. 19-27, 1996.
[Sabin70] M. A. Sabin, Interrogation techniques for parametric surfaces, *Proc. of Computer Graphics'70*, 1970.
[SP86] T. Sederberg and S.. Parry. Free-Form Deformation of Solid Geometric Models. *ACM Computer Graphics (SIGGRAPH '86 Proceedings)*, vol. 20, pp. 151--160, 1986.
[TPBF87] D. Terzopoulos, J. Platt, A. Barr and K. Fleischer, Elastically Deformable Models, *ACM Computer Graphics (SIGGRAPH'87 Proceedings)*, v.21(4), pp. 205-214, 1987.
[TQ94] D. Terzopolous and H. Qin, Dynamic NURBS with geometric constraints for interactive sculpturing, ACM Trans. on Graphics, 13(2), pp. 103-136, 1994.

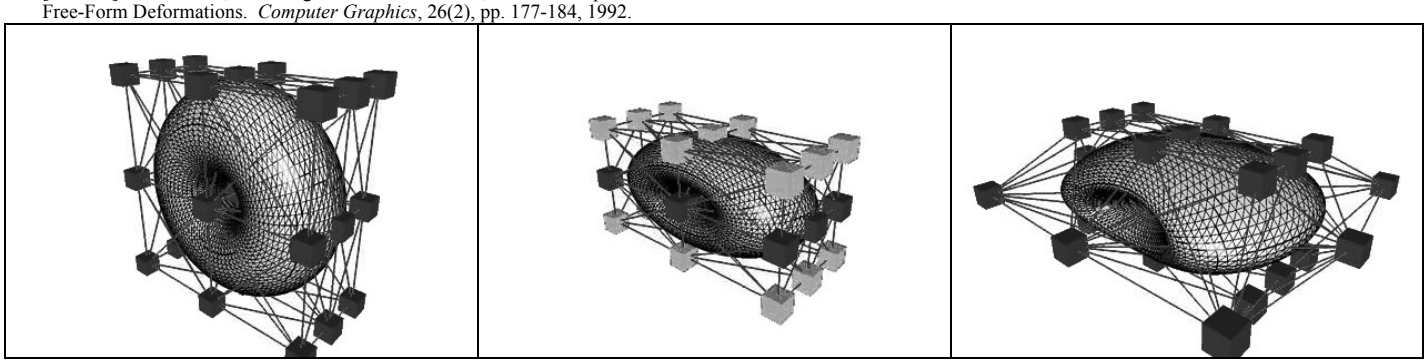


Plate 1: Torus Compression Example

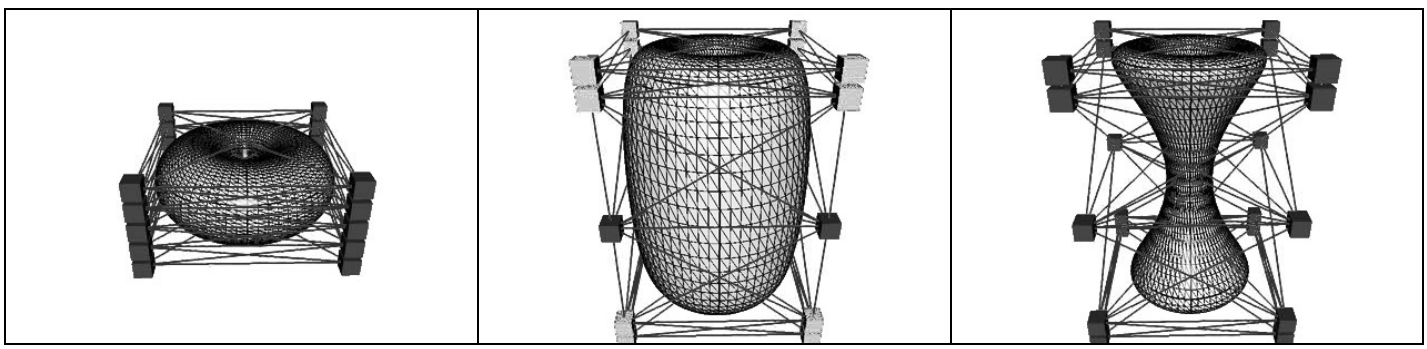


Plate 2: Stretched Torus Example. The triangle mesh at the convergence is 16 times denser than the mesh shown here.

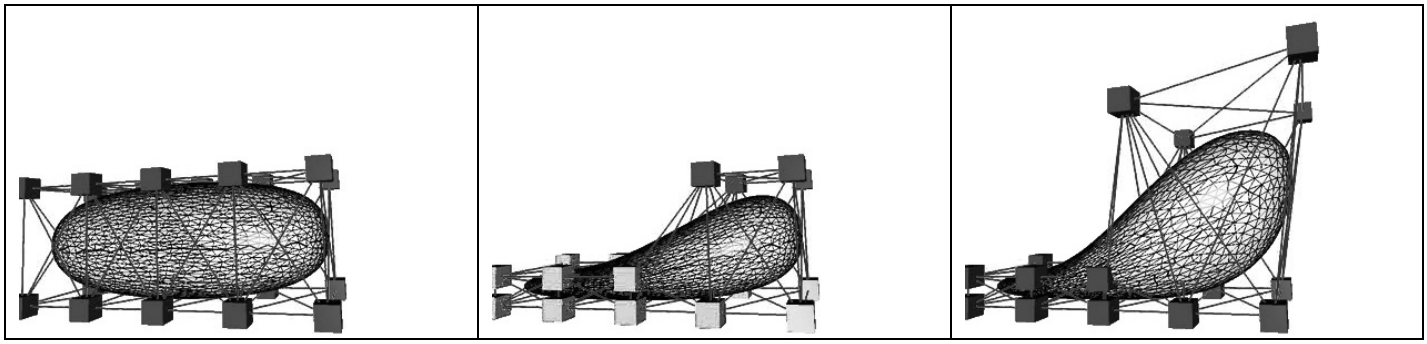


Plate 3: Rounded Club with Partial Compression

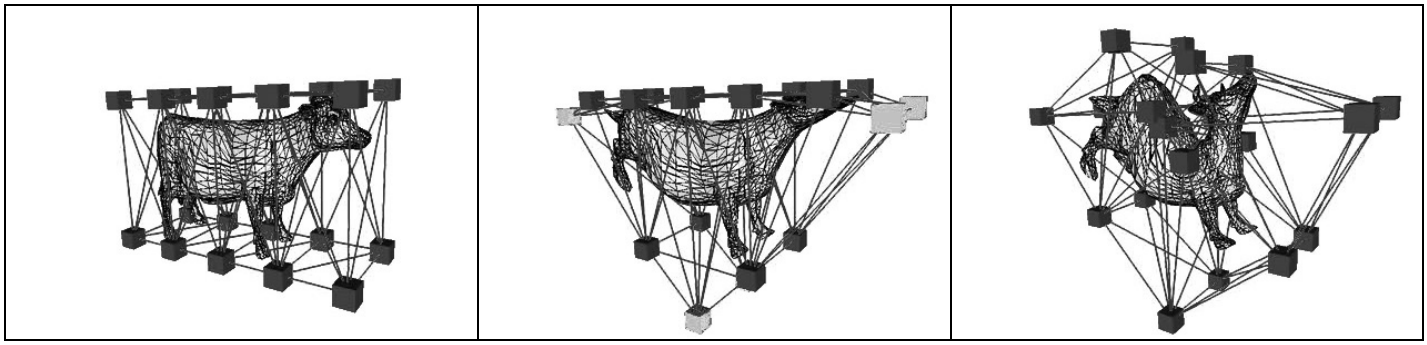


Plate 4: Bending Cow

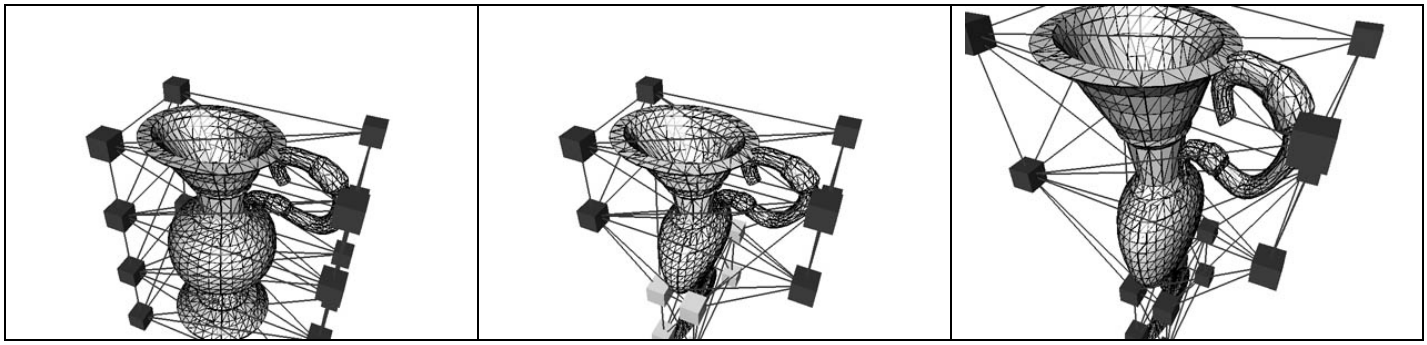


Plate 5: Water Pitcher Bottom Compression

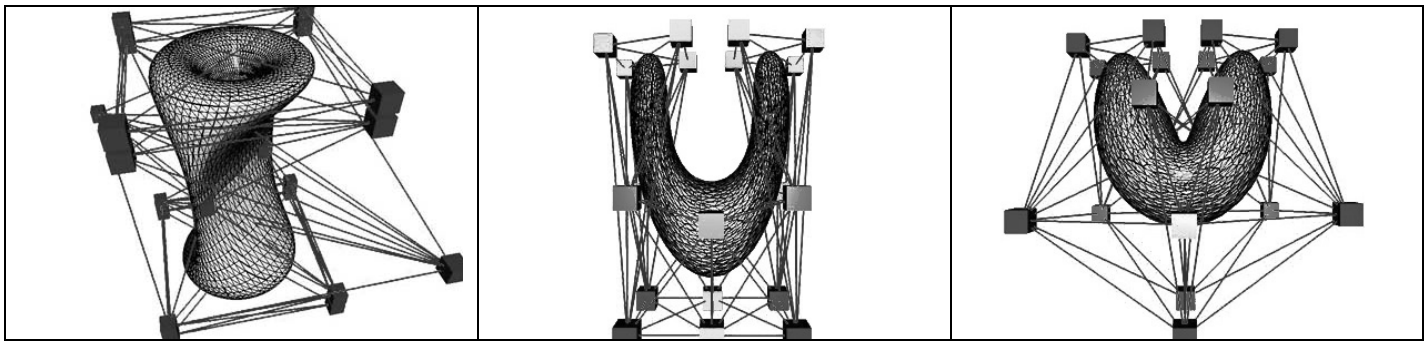
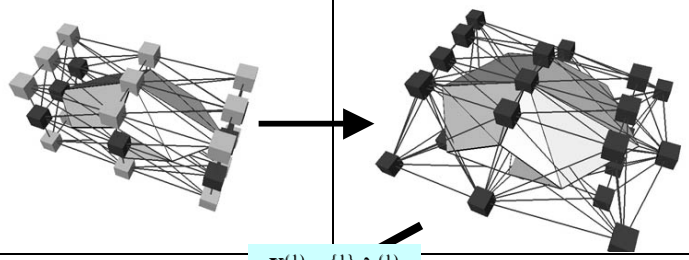
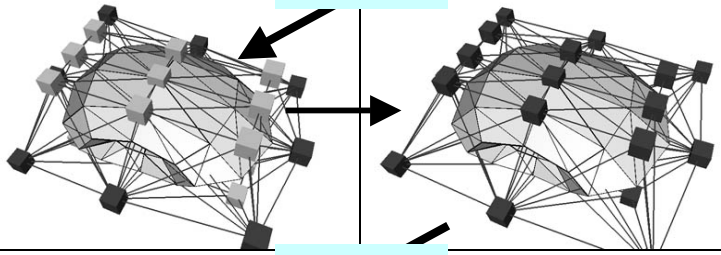
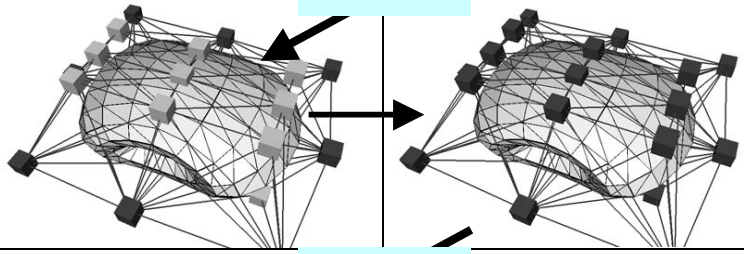
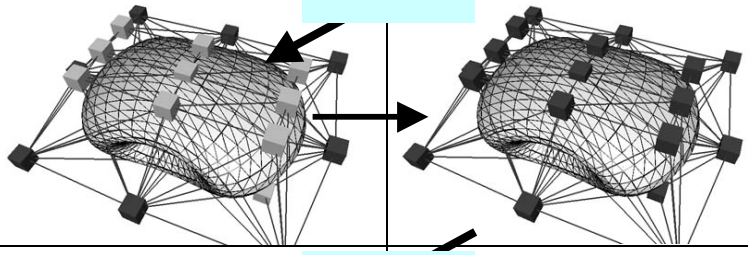
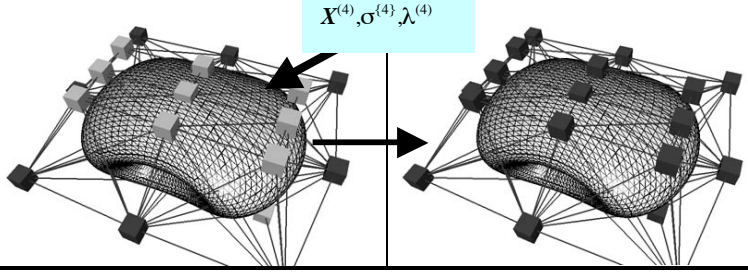


Plate 6: A Different Equilibrium State

Plate 7: Large Deformation. The club in plate 3 is bent by a user (left) resulting in the right one.

Table 2: Performance chart of multi-level optimization for a compressed torus example (Plate 1). The level k corresponds to the minimization at the k^{th} level of an approximation with N_{Tri} triangles. Light and dark cubes denote fixed and free nodes respectively. Node points $X^{(k)}$, penalty factor $\sigma^{(k)}$, and Lagrange multiplier $\lambda^{(k)}$ are inherited from one level to the next. ΔX is the maximum change in the coordinates of node points X relative to the size (the maximum extent) of the object. N_{step} is the total number of internal minimization steps at each level. T is CPU time (in msec) spent for the minimization at the k^{th} level. T_c is cumulative CPU time.

k	N_{Tri}	Before minimization.	After minimization	ΔX	N_{step}	T	T_c
0	**16**			**25%**	**90**	**120**	**120**
		$X^{(1)},\sigma^{\{1\}},\lambda^{(1)}$					
1	**64**			**7.9%**	**15**	**60**	**180**
		$X^{(2)},\sigma^{\{2\}},\lambda^{(2)}$					
2	**256**			**3.3%**	**19**	**290**	**470**
		$X^{(3)},\sigma^{(3)},\lambda^{(3)}$					
3	**1024**			**0.4%**	**13**	**700**	**1170**
		$X^{(4)},\sigma^{\{4\}},\lambda^{(4)}$					
4	**4096**			**0.01%**	**3**	**590**	**1760**