The Visual Computer

Dynamic sculpting and animation of free-form subdivision solids

Kevin T. McDonnell,
Hong Qin

Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794-4400, USA
E-mail: {ktm,qin}@cs.sunysb.edu

Published online: 15 March 2002

This paper presents a sculptured solid modeling system founded upon free-form splines: (i) tri-variate B-spline solids for regular (i.e., topologically cuboid) shapes and (ii) dynamic MacCracken–Joy subdivision-based solids of arbitrary topology. Our primary contribution is that we integrate the geometry of sculptured free-form solids with the powerful physics-based modeling framework by augmenting pure geometric entities with material properties and physical behaviors. We have developed a sculpting system with an array of design tools that afford users the ability to deform and sculpt a variety of free-form solids via a three-dimensional input device.

Key words: Computer graphics – Interactive techniques – Dynamics – Solid sculpting – Animation

Correspondence to: K.T. McDonnell

1 Introduction

During the past decade, solid modeling has quickly gained popularity as a convenient and natural paradigm for representing, manipulating and interacting with three-dimensional objects in interactive graphics, animation, CAD/CAM, art and entertainment, scientific visualization, and virtual environments. This is primarily because a solid model offers engineers an unambiguous shape representation of a physical entity (Requicha and Rossignac 1992). To date, the vast majority of popular solid modeling approaches, as well as commonly-used solid modeling systems, are built upon the following geometric foundations: constructive solid geometry (CSG), boundary representation (B-reps), and cell decomposition. CSG techniques exploit semi-algebraic sets and Boolean operations to combine simple primitives such as blocks, spheres, cylinders, cones, and tori into more complex solids. B-reps typically define a solid in terms of its surface geometry along with extra topological information. Cell decomposition methods usually approximate solids by using a set of 2D cross-sectional slices or a group of cubical units such as identical voxels and by employing a hierarchically structured octree scheme. When the goals are to interactively sculpt solid objects, deform the solid geometry with ease in real-time, modify the solid topology, and conduct kinematic and dynamic analysis of physical solids, prior representations and the current state-of-the-art in solid modeling fall short in offering designers an array of flexible and powerful modeling and sculpting tools.

Free-form solid modeling, in contrast, provides modelers with a more flexible interface for designing a much wider range of objects than the aforementioned approaches. Typical examples of sculptured solids include tri-variate B-splines, Hermite solids and non-uniform rational B-splines (NURBS) solids. However, free-form solids such as tri-variate B-splines typically offer users more degrees of freedom (i.e., control points, weights, etc.) than what they can actually handle. In addition, free-form solids based on parametric geometry are constrained to modeling topologically regular shapes. It is very difficult to extend the geometric coverage of free-form solids to shapes of arbitrary topology without resorting to a large set of non-intuitive geometric constraints.

Subdivision-based solid modeling, such as the MacCracken–Joy subdivision scheme for volumetric construction (MacCracken and Joy 1996), allows users to create sculptured solids of arbitrary topol-

The Visual Computer (2002) 18:81–96
Digital Object Identifier (DOI) 10.1007/s003710100138

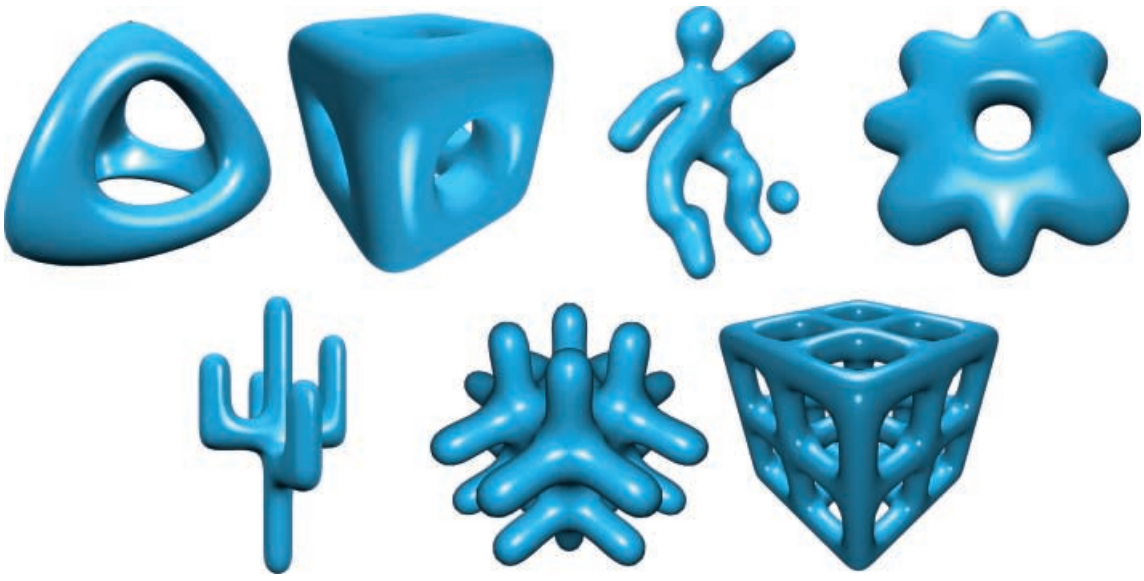


Fig. 1. Several dynamic subdivision-based solids created in our sculpting environment

ogy from user-specified initial control lattices. Despite their modeling superiority, subdivision solids also suffer from a large number of control points and an associated complex topological structure of their control lattices. These shortcomings severely restrain the modeling flexibility of subdivision solids as well as other popular free-form solids, since users essentially must interact with solid geometry through tedious and laborious operations on a large number of irregularly distributed control vertices.

In this paper, we systematically develop a physics-based modeling framework for free-form solids that can overcome many of the limitations associated with conventional solid modeling techniques. Within our novel dynamic modeling framework, free-form solids are equipped with mass and damping distributions, internal deformation energies, and other material properties. Consequently, users can sculpt solids in a physically plausible and accurate manner as if they are manipulating real-world *physical clay*. Figure 1 illustrates several objects that were sculpted in our system. Our free-form solids, whose control points are governed by differential equations of Lagrangian mechanics, respond dynamically to applied forces in an intuitive and natural fashion. Note that our prior efforts on physics-based modeling primarily centered on surface modeling techniques (Qin and Terzopoulos 1996; Qin et al. 1998). Surface modeling, in general, makes it difficult to realize the full potential of physics-based modeling due to its intrinsic deficiencies in modeling interior properties of physical solids (i.e., virtual clay). This paper pioneers the dynamic modeling methodology of spline-based solids through the physics-based derivation of tri-variate B-spline solids and subdivision-based solids of arbitrary topology. To achieve real-time sculpting performance, we discretize the continuum of both B-spline solids and subdivision solids into a set of finite-element-like cells and associate a novel mass–spring framework with free-form solids. Hence, general non-linear material properties can be attached to solid geometry with ease. We have developed a prototype solid modeling environment in which the real-time deformation and sculpting processes can be easily facilitated through a set of intuitive virtual tools and through the use of efficient numerical algorithms.

The remainder of this paper is organized as follows: We discuss the motivation and present our contributions in Sect. 2. In Sect. 3, we detail the geometry of free-form solids and review prior re-

search in relevant areas. In Sect. 4 we present our dynamic formulation and algorithms. In Sect. 5 we discuss the details of our implementation and sculpting system and then present our experimental results with time performance. Finally, we conclude the paper and outline future research directions in Sect. 6.

2 Motivation and contribution

At present, curve and surface modeling techniques are extensively used for representing a wide range of geometric shapes. However, such representations are far from adequate for modeling real-world objects when both interior properties and dynamic behaviors of the underling shape are of prime significance to modelers. In contrast, solid modeling has recently emerged as a very powerful paradigm that can greatly enhance existing surface modeling techniques because of its unique advantages over curve and surface modeling. Despite many advances in solid modeling during the past decade, conventional solid modeling techniques based on algebraic geometry can be rather rigid and inflexible. Free-form solids are a superior modeling candidate to CSG, B-reps and cell decomposition because (i) they have great potential to model a much wider range of real-world objects; (ii) they combine the benefits of free-form surface boundaries and interior geometry within a unified representation scheme; and (iii) they facilitate efficient algorithms for both interior interrogation and boundary evaluation.

Nevertheless, state-of-the-art free-form solids can be very difficult to use due to their bewildering number of degrees of freedom and their dependency on non-intuitive control point manipulation. Many more degrees of freedom (e.g., control points) are required than for surface models in order to represent the interior of solid geometry. Consequently, users are hampered by a large set of control points and its cumbersome manipulation. Additionally, complicated solids of arbitrary topology cannot be easily modeled with popular spline geometry such as tri-variate B-splines and NURBS due to the regular structure of their control vertices. Hence, considerable user intervention is necessary to perform the various routine tasks – such as rounding, blending, and trimming – that one sees in conventional CAD/CAM applications.

Strongly motivated by the recent advances of physics-based surface modeling techniques, we develop a novel dynamic modeling approach for solid modeling. Physics-based modeling attempts to overcome such shortcomings of geometric modeling through the integration of material attributes and physical behaviors with powerful geometric modeling techniques (Qin and Terzopoulos 1996; Qin et al. 1998). This approach alleviates the user's burden of managing large sets of degrees of freedom (e.g., a multitude of control points, knots, weights, etc.), which are typically required for the design of large, complicated objects. Note that the process of fine-tuning such low-level variables as control points is both tedious and cumbersome at best. It forces designers to make many small incremental changes in order to achieve the composite effect of larger deformation. When one interacts with parametric solids, it can be very difficult, sometimes impossible, to determine which control points lie in front of each other, relative to the current viewpoint on a 2D medium such as a computer screen. Furthermore, a purely geometric representation of a solid does not permit users to effectively validate physically relevant tests such as finite element analysis, kinematic simulation, and material property calculation. Physics-based modeling approaches allow users to focus more attention on the object and to more easily create a large array of shape variations permitted by the underlying solid mathematics. Based on our physics-based modeling methodology for surface design, we now forge ahead to tackle the challenging problem of integrating dynamic modeling algorithms with free-form solids in order to facilitate volumetric modeling, synthesis, and manipulation. Our primary contributions are as follows:

- We integrate material attributes with B-spline and subdivision solids and formulate a set of equations of motion for arbitrary free-form solids. The unified formulation permits users to easily and quickly deform a solid object in a physically plausible fashion. More importantly, the dynamic modeling method makes it much easier to define objects with anisotropic material distributions and to model even materially inhomogeneous objects.
- We develop a simple yet effective real-time numerical solver that employs a finite-difference approximation for the finite-element formulation of free-form solids. Our solver can achieve in-

teractive sculpting rates without sacrificing the modeling accuracy and deformation fidelity. In general, real-time performance is very difficult to accomplish when other mature finite-element solvers are utilized, even for the dynamic sculpting of rather simplified solid objects. Real-time interaction is especially important for animators, who frequently need immediate feedback in their applications.

- We discretize B-spline and subdivision solids into a set of cells bounded by a three-dimensional lattice. (In this paper, *lattice* stands for a three-dimensional network of points, edges, faces and cells, in contrast to a two-dimensional *mesh*.) A set of mass points and springs equipped with material and elastic properties is associated with geometric cells. A mass–spring discretization provides users with the intuitive mechanism for physically manipulating solid geometry. Furthermore, other downstream CAD/CAM applications such as finite-element meshing and analysis can be facilitated.
- The mathematics of parametric geometry for free-form solids always constrains the behavior of the mass–spring lattice, enforces the spline structure, and synchronizes two different representations throughout the sculpting task. Our *hybrid* model also consists of a set of *virtual springs*, whose purpose is to help preserve internal angles on each face of the lattice and thereby help maintain the object's general shape. In the absence of virtual springs, conventional springs are deployed to connect only immediate neighbors of any mass points. This can easily lead to an unstable configuration, which hampers the effective and meaningful sculpting of free-form solids.
- We implement a sculpting system for B-spline solids and subdivision-based solids that should appeal to both engineering designers and to non-technical users because it frees users from the need to understand the underlying complicated mathematics of free-form solids. A number of sculpting tools available in our system expedite the modeling and design tasks and offer users the realistic illusion of interacting with an elastic piece of virtual clay. The use of a three-dimensional input device enables users to freely navigate and manipulate any regions within free-form solids. Various physical properties of objects can be modified at run-time through a graphical user interface.

3 Background

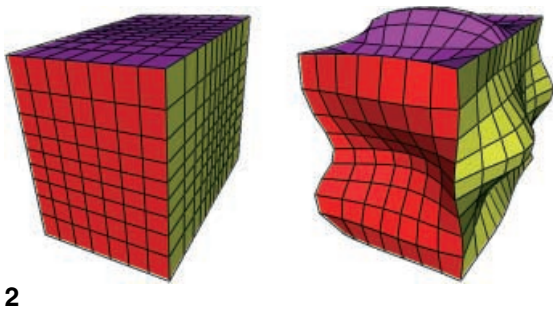
Free-form modeling (i.e., spline-based modeling) allows designers to create a large variety of very intricate curves, surfaces and solids by specifying a (possibly large) set of constraints. Typical constraints include locations of control points, values of weights and/or knots, and other geometric properties such as tangent and normal vectors, local curvature magnitude, etc. Among various modeling techniques and algorithms, B-splines, NURBS and subdivision schemes are of particular significance to users because of their modeling power for many man-made mechanical parts and natural objects. [For a comprehensive coverage of spline-based modeling, see Farin (1997); Mortenson (1997)]. This paper focuses on B-spline solids and subdivision solids primarily because (i) the B-spline solid is a special case of a subdivision solid and (ii) subdivision solids can represent complicated shapes of arbitrary topology with a small number of topologically irregular control points.

3.1 B-spline solids

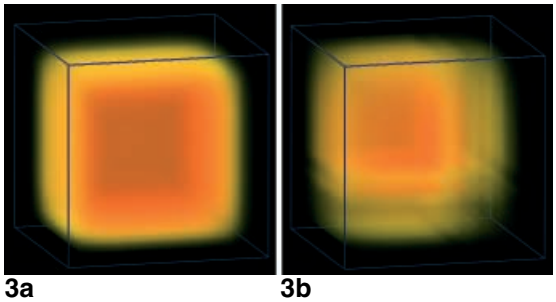
A continuous B-spline solid, $\mathbf{s}(u, v, w)$, is defined as the linear combination of a set of univariate basis functions, $B_{i,q}$, $B_{j,r}$ and $B_{k,s}$, with $(n+1) \times (m+1) \times (l+1)$ control points, $\mathbf{p}$:

$$\mathbf{s}(u, v, w) = \sum_{i=0}^{n} \sum_{j=0}^{m} \sum_{k=0}^{l} \mathbf{p}_{i,j,k} B_{i,q}(u)\, B_{j,r}(v)\, B_{k,s}(w) \ , \qquad (1)$$

where $B_{i,q}$, $B_{j,r}$ and $B_{k,s}$ are piecewise polynomials of order q, r and s, respectively. u, v and w are parametric variables. The parametric domain of B-splines can be determined by three sets of non-decreasing knot sequences. Univariate basis functions are then formulated recursively through knot vectors. In the interest of brevity, we refer readers to Qin and Terzopoulos (1996) for the explicit expression of B-spline basis functions and many of their attractive properties. Without loss of generality, we assume that u, v and w all belong to $[0, 1]$. The control point vector, $\mathbf{p}$, is the concatenation of all three-dimensional control points $\mathbf{p}(t) = [x, y, z]^{\mathrm{T}} : \mathbf{p} = [\mathbf{p}_{0,0,0}^{\mathrm{T}}, \mathbf{p}_{0,0,1}^{\mathrm{T}}, \ldots, \mathbf{p}_{n,m,l}^{\mathrm{T}}]^{\mathrm{T}}$, where $^{\mathrm{T}}$ denotes matrix transposition. Figure 2 illustrates a B-spline solid with $4 \times 4 \times 4$ control points and with knot vector $[0, 0, 0, 0, 1, 1, 1, 1]$.



2



3a 3b

Fig. 2. A dynamic B-spline solid (see Sect. 4.1) deformed in our sculpting system. Only the boundary is rendered

Fig. 3a,b. The mass distributions for the same $6 \times 6 \times 6$ B-spline solid but with different knot vectors. **a** The three knot vectors are [3, 3, 3, 3, 4, 5, 6, 6, 6, 6]. **b** The knot vectors are [3, 3, 3, 3, 3.5, 4, 6, 6, 6, 6]

Typically we use uniformly spaced knot vectors in our sculpting sessions, although non-uniform B-splines can also be represented in our systems. In Fig. 3 we show the approximate mass distributions of two B-spline solids with $6 \times 6 \times 6$ control lattices. In each solid, the corresponding control points have identical masses, but the different knot vectors blend and distribute the mass differently. The three knot vectors in the first solid are all [3, 3, 3, 3, 4, 5, 6, 6, 6, 6], while in the second solid all three are [3, 3, 3, 3, 3.5, 4, 6, 6, 6, 6].

3.2 Subdivision techniques

Subdivision schemes are widely used in curve and surface modeling. Chaikin (1974) introduced the concept of subdivision to the modeling community for generating a smooth curve from an arbitrary control polygon. In the limit, Chaikin's curve converges to a quadratic B-spline. Subsequently, a wide variety of subdivision schemes for modeling smooth surfaces of arbitrary topology have been derived following Chaikin's pioneering work on curve generation. The existing subdivision schemes can be broadly categorized into two distinct classes, namely, (i) approximating subdivision techniques and (ii) interpolating subdivision techniques.

Among the approximating surface schemes, the techniques of Doo and Sabin (Doo 1978) and Catmull and Clark (1978) generalize the idea of obtaining uniform bi-quadratic and bi-cubic B-spline patches, respectively, from a rectangular control mesh. Catmull and Clark developed an algorithm for recursively generating a smooth surface from a polyhedral mesh of arbitrary topology. The Catmull–Clark subdivision surface can be reduced to a set of standard B-spline patches except at a finite number of degenerate points. Loop (1987) presented a similar subdivision scheme based on the generalization of quartic triangular B-splines for triangular meshes. Most recently, non-uniform Doo–Sabin and Catmull–Clark surfaces that generalize non-uniform tensor-product B-spline surfaces to arbitrary topologies were introduced by Sederberg et al. (1998). The most well-known interpolation-based subdivision surface scheme is the "butterfly" algorithm proposed by Dyn et al. (1990). The butterfly method, like other subdivision schemes, makes use of a small neighborhood of vertices for subdivision. It requires simple data structures and is rather straightforward to implement. Nevertheless, it needs a topologically regular setting of the initial (control) mesh in order to obtain a smooth C^1 limit surface. Zorin et al. (1996) have improved this interpolatory subdivision scheme (which we call the *modified* butterfly scheme) that retains the simplicity of the butterfly scheme and results in much smoother surfaces even from irregular initial meshes.

3.3 MacCracken–Joy subdivision solids

Unlike subdivision curves and surfaces, little research has been published on modeling solids through subdivision techniques. MacCracken and Joy extended the subdivision rules for Catmull–Clark surfaces to generate a volumetric model from a lattice of control vertices (MacCracken and Joy 1996). However, their goal for using subdivision techniques was to contrive a space in which an arbitrary object could be deformed (i.e., free-form deformation, FFD). In this paper, we treat the volumet-

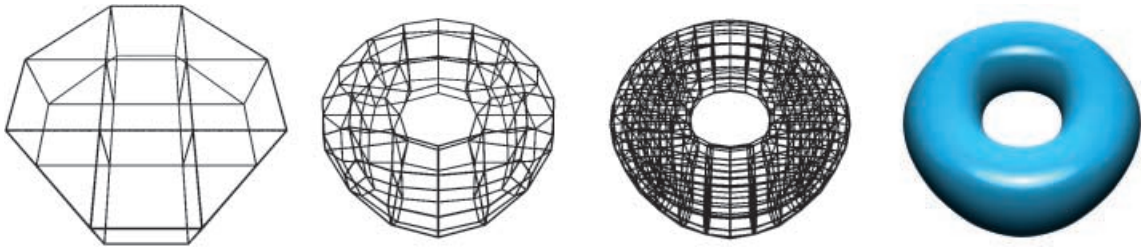


Fig. 4. A solid, torus-like object that has been subdivided a few times using the solid subdivision rules. The complex topology becomes apparent when the solid is drawn in wireframe

ric MacCracken–Joy subdivision scheme as a novel free-form spline solid that is obtained in the limit through recursive application of subdivision rules on a user-specified lattice of control vertices.

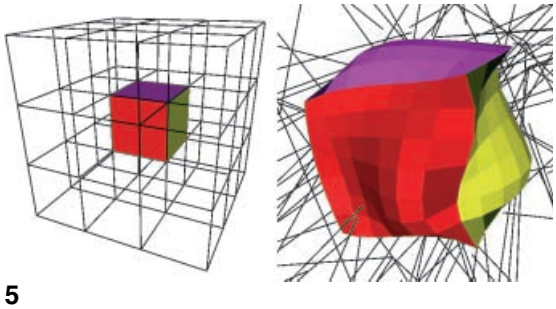
Given an initial *control lattice* in 3-space, the subdivision solid rules recursively subdivide the lattice and refine the three-dimensional space occupied by the lattice. The lattice consists of a set of closed cells that are defined by a collection of their constituent faces. The faces in the lattice are comprised of an ordered list of edges that are defined by their end vertices. Hence, there are four types of geometric entities in the lattice: cells, faces, edges and points. Figure 4 shows an example of a solid.

The subdivision scheme recursively applies a set of four rules, one for each type of element, in order to achieve successively finer representations of the original lattice. Each element in the lattice produces a new vertex that must be subsequently incorporated into the next finer level of the subdivided lattice. Although many alternative subdivision rules can be devised, we follow the original rules as described by MacCracken and Joy in our paper because this set of rules generates a B-spline solid in the limit if the initial control lattice is regular (i.e., each vertex shares six edges, and each cell is topologically cuboid). Hence, the B-spline solid explained above is a special case of our subdivision solids.

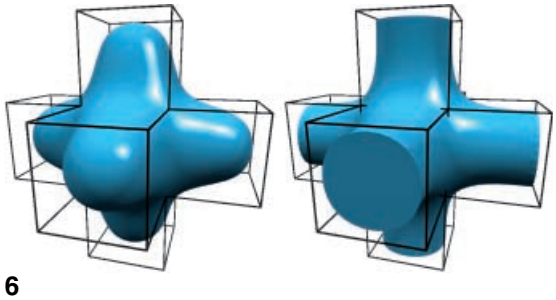
The subdivision solid rules include:

- *Cell points:* For each cell, the cell point is its centroid.
- *Face points:* For each face, the face point is the weighted average: $\mathbf{f} = \frac{\mathbf{c}_0+2\mathbf{a}+\mathbf{c}_1}{4}$, where $\mathbf{a}$ is the face's centroid and $\mathbf{c}_0$ and $\mathbf{c}_1$ are the centroids of the cells on either side of the face.
- *Edge points:* For each edge, the edge point is the weighted average: $\mathbf{e} = \frac{\mathbf{c}_{\text{avg}}+2\mathbf{a}_{\text{avg}}+(n-3)\mathbf{m}}{n}$, where $\mathbf{c}_{\text{avg}}$ is the average of the centroids of the cells that contain the edge, $\mathbf{a}_{\text{avg}}$ is the average of the centroids of the faces that contain the edge, $\mathbf{m}$ is the mid-point of the edge, and n is the number of faces that contain the edge.
- *Vertex points:* For each vertex $\mathbf{p}$, the vertex point is the weighted average: $\mathbf{v} = \frac{\mathbf{c}_{\text{avg}}+3\mathbf{a}_{\text{avg}}+3\mathbf{m}_{\text{avg}}+\mathbf{p}}{8}$, where $\mathbf{c}_{\text{avg}}$ is the average of the centroids of the cells that contain the point, $\mathbf{a}_{\text{avg}}$ is the average of the centroids of the faces that contain the point, and $\mathbf{m}_{\text{avg}}$ is the average of the mid-points of the edges that contain the point.

Note that these rules are a straightforward extension of the subdivision rules for surfaces originally described in Catmull and Clark (1978). For example, a tensor-product tri-variate B-spline solid with uniform knot sequence $[0, 1, 2, 3, 4, 5, 6, 7]$ can be obtained in the limit of the recursive subdivision process (Fig. 5) on $4 \times 4 \times 4$ control lattice, without any special rules on boundary elements (e.g., face, edge, and vertex). We can introduce a special set of rules for boundary elements in the solid lattice in order to enhance the geometric flexibility of subdivision solids. In particular, we can use Catmull–Clark subdivision rules for surface generation in which a smooth rounded boundary of the solid will be obtained instead. Alternatively, we can modify the rules to produce sharp edges on the boundary (MacCracken and Joy 1996; DeRose et al. 1998). Figure 6 illustrates the same initial control lattice for



5



6

Fig. 5. A tri-variate B-spline solid with uniform knot distribution obtained in the limit of the subdivision process. The second image shows a zoomed view of the solid after deformation. The control lattice is drawn in wireframe
Fig. 6. Different sets of rules are applied to the boundary for the same initial cross-shaped control lattice

which different surface subdivision rules have been applied on the boundary.

The new lattice is then assembled as follows: Cell points are connected to the new face points of the faces that defined the cell; face points are connected to the new edge points of the edges that defined the face; and edge points are connected to the new vertex points of the vertices that defined the edge. In the limit of this recursive subdivision process, a smooth free-form solid, **s**, can be obtained:

$$\mathbf{s}(\mathbf{x}) = \sum_{i=1}^{n} \mathbf{p}_i \hat{B}_i(\mathbf{x}) \ , \qquad (2)$$

where $\mathbf{x}$ is a parametric value whose domain is a 3-space occupied by the initial lattice, $\mathbf{p}_i$ is a control point, which is one of the original lattice points, $\hat{B}_i(\mathbf{x})$ is a basis function, and n is the number of the initial vertices defined by the original lattice. Note that the affine rules explained above naturally establish one-to-one point correspondences between lattices in two consecutive levels within the subdivision hierarchy. As with Catmull–Clark surfaces, the basis functions $\hat{B}_i(\mathbf{x})$ of subdivision solids in the limit can be defined explicitly over the original lattice. However, it is non-trivial to derive the closed-form analytic equation for $\hat{B}_i(\mathbf{x})$, given an initial control lattice of arbitrary connectivity and topology. In contrast, when the initial lattice is regular, a certain basis function, $B_h(\mathbf{x})$, can be computed from three univariate B-spline basis functions: $B_h(\mathbf{x}) = B_{i,q}(u)\, B_{j,r}(v)\, B_{k,s}(w)$, where $B_{i,q}(u)$, $B_{j,r}(v)$, and $B_{k,s}(w)$ are the piecewise B-spline polynomials. This is because the subdivision solid of a regular lattice reduces to a standard tri-cubic B-spline solid.

3.4 Physics-based modeling

Although free-form modeling approaches are powerful for representing smooth volumetric shapes, they constitute a purely geometric representation. In addition, conventional geometric modeling may be inconvenient for representing complicated solids, because modelers are faced with the tedium of indirect shape modification and refinement through time-consuming operations on a large number of control vertices. Despite the advent of advanced three-dimensional graphics interaction tools, these indirect geometric operations remain non-intuitive and laborious, in general. In contrast, physics-based models respond to externally applied forces in a very intuitive manner. The dynamic formulation marries the model geometry with time, mass, damping and constraints via a force-balance equation. Dynamic models produce smooth, natural motions that are intuitive to control. In addition, they facilitate interaction – especially direct manipulation – of complex geometries and topologies. Furthermore, the equilibrium state of the model is characterized by a minimum of the deformation energy subject to the imposed constraints. The deformation energy functionals can be formulated to satisfy local and global modeling criteria, and geometric constraints relevant to shape control can also be imposed. The dynamic approach subsumes all of the aforementioned modeling capabilities in a formulation that grounds everything in real-world physical behavior.

Free-form deformable models were introduced to computer graphics by Terzopoulos et al. (1987) and further developed by Terzopoulos and Fleischer (1988), Pentland and Williams (1989), and Metaxas

and Terzopoulous (1992). Qin and Terzopoulos introduced D-NURBS surfaces, an extension to traditional NURBS that permits more natural control of the geometry of the surface (Qin and Terzopoulos 1996; Terzopoulos and Qin 1994). Later, Qin et al. extended such ideas to dynamic subdivision surfaces (Qin et al. 1998; Mandal et al. 1999). Most recently, Dachille et al. (1999) combined haptic interaction with dynamic B-spline surfaces to provide a very natural user interface for deformation. To date, however, most physics-based modeling work has focused on dynamic surface modeling. This paper incorporates spline-based and subdivision-based solid modeling into the dynamic framework.

4 Formulation and algorithms

This section presents the dynamic modeling formulation and numerical algorithms for the both B-spline solids and subdivision solids. We begin with the formulation for dynamic B-spline solids and follow with a natural generalization to dynamic subdivision solids.

4.1 Dynamic B-spline solids

In order to incorporate physical properties into a B-spline solid, we synchronize B-spline solid geometry with a mass–spring lattice. This relationship can be expressed as the matrix multiplication

$$\mathbf{c} = \mathbf{B}\mathbf{q} \,, \tag{3}$$

where $\mathbf{q}$ collects all the positional components (i.e., x, y, and z coordinates) of the control points, $\mathbf{c}$ contains the positions of the mass points, and $\mathbf{B}$ is a transformation matrix whose entries are the basis functions evaluated at various parametric values. $\mathbf{B}$ is uniquely determined by the parametric values of $\mathbf{c}$. Suppose, for example, that $\mathbf{B}$ is of size $n \times m$, where n is the number of control points and $m = m_0 m_1 m_2$. The basis functions are uniformly sampled at m_0, m_1 and m_2 in the u, v and w directions, respectively. These weights are then stored in $\mathbf{B}$ and when multiplied with $\mathbf{q}$ give the positions of the mass points, $\mathbf{c}$.

4.2 Dynamic subdivision solids

In order to associate material properties with a subdivision solid, we begin by expressing the subdivision process using the same matrix–vector multiplication formulation as in Eq. (3):

$$\mathbf{d} = \mathbf{A}\mathbf{p} \,. \tag{4}$$

Again, we exploit the idea exhibited in Eq. (3). However, unlike B-spline solids, subdivision techniques do not offer users explicit analytic formulations of basis functions. It is a tremendous challenge to accurately evaluate these basis functions for our dynamic modeling formulation. Hence, we construct the basis function matrices in a different manner. We assume that vector where $\mathbf{p}$ contains the positions of the lattice points at the initial, coarsest level. As shown in Eq. (2), such points are called the "control points" of a subdivision solid in analogy with parametric solids. Vector $\mathbf{d}$ contains the positions of the points in the lattice at the current subdivision level. We call these vertices "data points" or "mass points," since each of them is assigned a mass. (Note that if no subdivision has been performed, $\mathbf{d} = \mathbf{p}$.) We use the discretized point set $\mathbf{d}$ to approximate the continuous subdivision solid in the interest of the simplicity and efficiency of dynamic simulation and manipulation. Note that after several levels of subdivision are conducted, $\mathbf{d}$ can be considered to be a good approximation with high precision. $\mathbf{A}$ contains weights given by the subdivision rules and defines how to obtain the data point positions in $\mathbf{d}$ from $\mathbf{p}$.

During the subdivision process, each new data point is defined by a certain affine combination of the existing points as computed by the subdivision rules. We can store this collection of weights for each point in $\mathbf{A}$. We can then perform the matrix multiplication in Eq. (3) to obtain the positions of the points in the new lattice. Below we describe how to compute $\mathbf{A}$.

Assume we wish to subdivide our solid n times. We can express this process as a right-multiplication of $n+1$ matrices, constructed as follows:

1. Run the subdivision algorithm once on $\mathbf{p}$ to obtain the first subdivided lattice. Name this new lattice $\mathbf{d}^{(1)}$ and the subdivision matrix $\mathbf{A}^{(1)}$.
2. Given lattice $\mathbf{d}^{(j)}$, compute $\mathbf{A}^{(j)}$ using the subdivision scheme, and multiply $\mathbf{A}^{(j)}$ by $\mathbf{d}^{(j)}$ to get $\mathbf{d}^{(j+1)}$: $\mathbf{d}^{(j+1)} = \mathbf{A}^{(j)}\mathbf{d}^{(j)}$.
3. Set $j \leftarrow j+1$.
4. If $j \leq n$, then go to step 2. Else stop.

After n levels of subdivision are conducted, our data point set $\mathbf{d}$ (used to approximate the continuous MacCracken–Joy solid) can be assigned to the current lattice $\mathbf{d}^{(n)}$: $\mathbf{d} = \mathbf{d}^{(n)}$. We find that the above

algorithm produces the following series of multiplications:

$$\begin{aligned}
\mathbf{d}^{(1)} &= \mathbf{A}^{(1)}\mathbf{p} \,, \\
\mathbf{d}^{(2)} &= \mathbf{A}^{(2)}\mathbf{d}^{(1)} \,, \\
&\vdots \\
\mathbf{d} = \mathbf{d}^{(n)} &= \mathbf{A}^{(n)}\mathbf{d}^{(n-1)} \,.
\end{aligned}$$

We collect all the terms and arrive at: $\mathbf{d} = \mathbf{A}^{(n)}\mathbf{A}^{(n-1)} \cdots \mathbf{A}^{(1)}\mathbf{p}$. Assigning $\mathbf{A} = \mathbf{A}^{(n)}\mathbf{A}^{(n-1)} \cdots \mathbf{A}^{(1)}$ gives us the compact form of Eq. (3), where $\mathbf{A}$ is an approximation of the discretized basis function matrix, obtained through a procedure-based subdivision algorithm using the subdivision solid rules. Note that the accuracy of this approximation depends on the subdivision level, and it can be satisfied within any user-specified tolerances. Figure 1 shows various examples of dynamic subdivision solids. We can represent objects of arbitrary topologies as well as organic shapes such as a soccer player.

4.3 Dynamics equations

Now we set the stage to define how the dynamic system evolves over time. A dynamic solid is characterized by its position $\mathbf{s}(\mathbf{x}, t)$, velocity $\dot{\mathbf{s}}(\mathbf{x}, t)$ (which stands for $\frac{\partial \mathbf{s}(\mathbf{x},t)}{\partial t}$), and acceleration $\ddot{\mathbf{s}}(\mathbf{x}, t)$ (i.e., $\frac{\partial^2 \mathbf{s}(\mathbf{x},t)}{\partial t^2}$) along with material properties including mass density, $\mu(\mathbf{x})$, damping density, $\gamma(\mathbf{x})$, and the internal energy functional, $E(\mathbf{s})$. The continuous form of the Lagrangian equations of motion can be written as

$$\mu\ddot{\mathbf{s}} + \gamma\dot{\mathbf{s}} + \frac{\partial E(\mathbf{s})}{\partial \mathbf{s}} = \mathbf{f} \,, \tag{5}$$

where $\mathbf{f}(\mathbf{s})$ is the sum of all external force distribution acting on $\mathbf{s}(\mathbf{x})$. A large variety of physical behavior of $\mathbf{s}$ (elasticity, plasticity, etc.) results from different mathematical formulations of energy functionals, E (Terzopoulos and Fleischer 1988).
Based on concepts from differential geometry, a 3×3 metric tensor function $\mathbf{G}(\mathbf{s})$,

$$\mathbf{G} = \begin{pmatrix}
\frac{\partial \mathbf{s}}{\partial u} \cdot \frac{\partial \mathbf{s}}{\partial u} & \frac{\partial \mathbf{s}}{\partial u} \cdot \frac{\partial \mathbf{s}}{\partial v} & \frac{\partial \mathbf{s}}{\partial u} \cdot \frac{\partial \mathbf{s}}{\partial w} \\
\frac{\partial \mathbf{s}}{\partial v} \cdot \frac{\partial \mathbf{s}}{\partial u} & \frac{\partial \mathbf{s}}{\partial v} \cdot \frac{\partial \mathbf{s}}{\partial v} & \frac{\partial \mathbf{s}}{\partial v} \cdot \frac{\partial \mathbf{s}}{\partial w} \\
\frac{\partial \mathbf{s}}{\partial w} \cdot \frac{\partial \mathbf{s}}{\partial u} & \frac{\partial \mathbf{s}}{\partial w} \cdot \frac{\partial \mathbf{s}}{\partial v} & \frac{\partial \mathbf{s}}{\partial w} \cdot \frac{\partial \mathbf{s}}{\partial w}
\end{pmatrix} ,$$

remains unchanged if there is no solid deformation of $\mathbf{s}(\mathbf{x})$ [i.e., rigid-body motion does not modify the tensor function $\mathbf{G}(\mathbf{s})$] during the process of dynamic sculpting and animation. Note that (u, v, w) is the parametric coordinate for $\mathbf{x}$. To minimize the volumetric variation of a solid $\mathbf{s}$, we let

$$E = \int_{\Omega} \sum_i \sum_j c_{i,j}(u, v, w) \left(g_{i,j} - g_{i,j}^{(0)}\right)^2 \mathrm{d}u\mathrm{d}v\mathrm{d}w$$

throughout this paper, where $g_{i,j}$ is the (i, j) entry of $\mathbf{G}$, the superscript of $g_{i,j}$ denotes the default value at the rest shape ($t = 0$), and $c_{i,j}$ serves as the weighting function (Terzopoulos and Fleischer 1988).
The discretization of free-form solids will transform Eq. (5) to Lagrangian dynamics of all mass points:

$$\mathbf{M}\ddot{\mathbf{d}} + \mathbf{D}\dot{\mathbf{d}} + \mathbf{K}\mathbf{d} = \mathbf{f}_{\mathbf{d}} \,. \tag{6}$$

Since the data point set of a solid is constrained by control points, we obtain a new set of motion equations whose unknowns are $\mathbf{p}$:

$$\mathbf{A}^{\mathrm{T}}\mathbf{M}\mathbf{A}\ddot{\mathbf{p}} + \mathbf{A}^{\mathrm{T}}\mathbf{D}\mathbf{A}\dot{\mathbf{p}} + \mathbf{A}^{\mathrm{T}}\mathbf{K}\mathbf{A}\mathbf{p} = \mathbf{A}^{\mathrm{T}}\mathbf{f}_{\mathbf{d}} \,. \tag{7}$$

Therefore, we can directly compute the acceleration of the control points ($\ddot{\mathbf{p}}$) based on the forces acting on the lattice of the current data points $\mathbf{d}$:

$$\mathbf{A}^{\mathrm{T}}\mathbf{M}\mathbf{A}\ddot{\mathbf{p}} + \mathbf{A}^{\mathrm{T}}\mathbf{D}\dot{\mathbf{d}} + \mathbf{A}^{\mathrm{T}}\mathbf{K}\mathbf{d} = \mathbf{A}^{\mathrm{T}}\mathbf{f}_{\mathbf{d}} \,, \tag{8}$$

$$\mathbf{A}^{\mathrm{T}}\mathbf{M}\mathbf{A}\ddot{\mathbf{p}} = \mathbf{A}^{\mathrm{T}}\mathbf{f}_{\mathbf{d}} - \mathbf{A}^{\mathrm{T}}\mathbf{D}\dot{\mathbf{d}} - \mathbf{A}^{\mathrm{T}}\mathbf{K}\mathbf{d} \,, \tag{9}$$

$$\ddot{\mathbf{p}} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{M}\mathbf{A}\right)^{-1}\left(\mathbf{A}^{\mathrm{T}}\mathbf{f}_{\mathbf{d}} - \mathbf{A}^{\mathrm{T}}\mathbf{D}\dot{\mathbf{d}} - \mathbf{A}^{\mathrm{T}}\mathbf{K}\mathbf{d}\right) \,. \tag{10}$$

4.4 Numerical integration

In order to achieve real-time, interactive performance in our sculpting system, we employ an explicit integration method to steer our physical simulation. At each time-step, the state of the system is advanced based on the preceding states. The summarized forces on the discretized lattice are applied, and then the accelerations of the control points are computed using Eq. (10).
The velocities $\dot{\mathbf{p}}$ of the control points are updated according to the applied forces and material quantities (i.e., mass, damping, and stiffness). The control points are moved to their new positions:

$$\begin{aligned}
\dot{\mathbf{p}}(t + \Delta t) &= \dot{\mathbf{p}}(t) + \ddot{\mathbf{p}}(t)\Delta t \,, \\
\mathbf{p}(t + \Delta t) &= \mathbf{p}(t) + \dot{\mathbf{p}}(t)\Delta t \,,
\end{aligned}$$

where the variable t denotes time, and Δt stands for one time-step. The updated control point positions

$\mathbf{p}(t+\Delta t)$ and their velocities are further used to update the discretized model defined by $\mathbf{d}(t+\Delta t) = \mathbf{A}\mathbf{p}(t+\Delta t)$ and $\dot{\mathbf{d}}(t+\Delta t) = \mathbf{A}\dot{\mathbf{p}}(t+\Delta t)$.

5 Sculpting system

We have developed a prototype sculpting system that allows users to interactively model, design, and deform B-spline solids and subdivision solids in real-time. This section details the implementation issues.

5.1 Forces and constraints

The discretized dynamic model has material properties such as mass, stiffness and damping distributions. Currently, these values are distributed equally throughout the solid in our sculpting system, although it would be simple to extend our implementation to permit heterogenous distributions. The mass points are connected to their immediate neighbors through springs. These springs correspond to just the diagonal entries of the metric tensor $\mathbf{G}$, which seek to maintain the Euclidean distances between all neighboring points. However, nearest-neighbor connections would not sufficiently constrain the overall deformation of the solid due to the lack of control of off-diagonal entries in $\mathbf{G}$ (which attributes to differential quantities of local angles). For this reason, we incorporate an additional set of *virtual springs* into conventional mass–spring systems. These virtual springs, whose stiffnesses may be different from normal springs, are intended to help maintain internal angles of each face, and thereby minimize the volumetric distortion of a solid object. Note that, since the stiffness matrix $\mathbf{K}$ (derived from E) is a function of $\mathbf{s}$ and is extremely complex due to the non-linear nature of E, it would be nearly impossible to achieve real-time sculpting performance if the system were to assemble it at each time-step based on the energy functional E. Therefore, our mass–spring configuration explained above only intends to achieve the objective of minimizing the volumetric distortion (characterized by $\left|\mathbf{G}-\mathbf{G}^{(0)}\right|$) in an approximate sense. More accurate implementations based on the finite-element method would be more desirable for certain applications such as engineering design and analysis. Such approaches are currently under investigation.

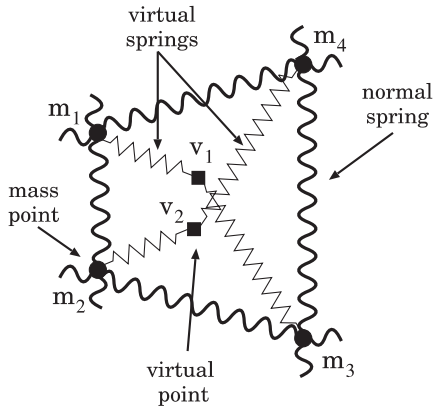
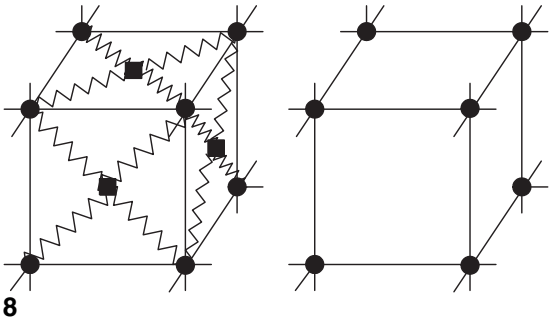


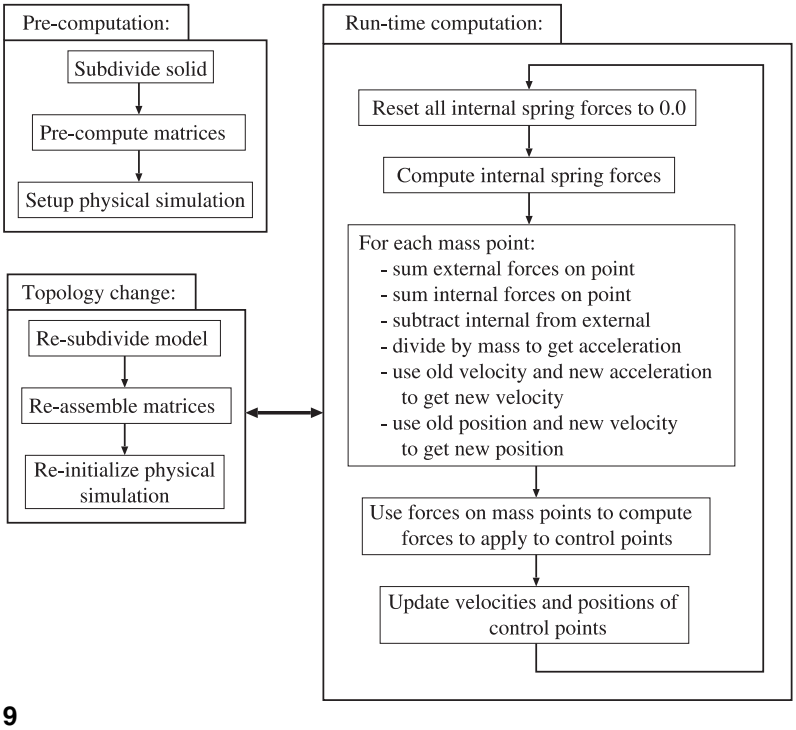
Fig. 7. The virtual and normal springs associated with a typical face of a lattice. *Solid circles*: mass points; *solid squares*: virtual points. The position of mass point $\mathbf{m_1}$'s virtual point, $\mathbf{v_1}$, is defined as the average of the positions of $\mathbf{m_1}$'s neighboring mass points: $\mathbf{v_1} = \frac{1}{2}(\mathbf{m_2} + \mathbf{m_4})$

Figure 7 illustrates an example of four virtual springs for a single face element. For each point $\mathbf{m}$ of each face in the finest lattice, we introduce a corresponding *virtual point*, $\mathbf{v}$, whose location is defined as the mid-point of the two points adjacent to point $\mathbf{m}$ in the same face. One end of the virtual spring is attached to $\mathbf{m}$ and the other end to the virtual point. The location of the virtual point is updated dynamically as the accompanying positions of the two points evolve over time. The rest length of the spring remains constant during the entire sculpting process, however. The virtual springs resist any changes in their corresponding internal angles for each face, and thereby assist in minimizing local distortions. Note that unlike traditional "diagonal" springs, one end-point of a virtual spring is a mass point and the other a massless point whose position is determined by geometry. Figure 8 shows a typical cell with and without virtual springs. In the interest of clarity, normal springs are drawn as straight lines.

Figure 9 provides a global view of the major steps in our system architecture. After an initial preprocessing stage, the simulation runs in a loop and continuously updates the state of the dynamic solid object in real-time. The simulation process starts with a preprocessing stage in which we subdivide the solid to the requested level, pre-compute several matrices and initialize the physical state. Dur-



8

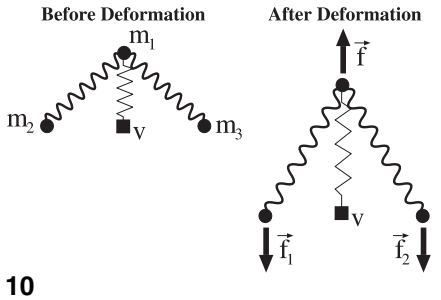


9

Fig. 8. A typical regular cell of a mass-spring lattice both with and without virtual springs. Normal springs are drawn as *straight lines* in order to avoid cluttering the figure. *Solid circles*: mass points; *solid squares*: virtual points

Fig. 9. System architecture of the physical simulation

ing the simulation, the system evolves as follows: First, all internal forces are reset to zero. Second, for each spring, the force exerted on each mass point attached to the spring is computed by Hooke's law: $\mathbf{f} = -k\,(\mathbf{l} - \mathbf{l}_0)$, where $\mathbf{l}_0$ is the spring's rest length. Third, the forces exerted by the virtual springs are taken into account, again using Hooke's law. After we compute the force exerted on the mass point that is attached to a virtual spring, exactly one-half of that force is applied in the opposite direction to each of the endpoints that define the virtual point's position (see Fig. 10). This is done in order to preserve total internal energy. The sum of all forces exerted on a mass point is then subtracted from any external user-applied forces on the point. A damping force is applied uniformly over all mass points to prevent



10

$$\begin{bmatrix}\mathbf{d}\end{bmatrix} = \begin{bmatrix}\mathbf{A}\end{bmatrix}\begin{bmatrix}\mathbf{p}\end{bmatrix} \rightarrow \begin{bmatrix}\mathbf{d}_1\end{bmatrix} = \begin{bmatrix}\mathbf{A}_1\end{bmatrix}\begin{bmatrix}\mathbf{p}_1\end{bmatrix} + \begin{bmatrix}\mathbf{A}_2\end{bmatrix}\begin{bmatrix}\mathbf{p}_2\end{bmatrix}$$

11

Fig. 10. The user exerts a force **f** on a mass point, and the virtual spring causes additional forces to be applied. (Note that $\mathbf{f}_1 + \mathbf{f}_2 = -\mathbf{f}$)

Fig. 11. Matrices **d**, **A** and **p** are partitioned to speed up the simulation

oscillations and to bring the system to an eventual rest state. We then divide the force acting on each point by the mass in order to compute the point's acceleration. The positions of the virtual points are updated. Finally, we compute the accelerations of the control points using Eq. (10) and update their positions. At any time during the simulation, the user may modify the object's topology, and the system will rebuild the necessary data structures to effect the change.

5.2 Fixed regions and local modifications

For large control lattices, our model has the potential of being computationally expensive due to a very large number of mass points. In order to improve system performance, our model can be modified to constrain a set of selected control points that are attributed to certain fixed regions. This reduces the amount of data that needs to be processed by the system and thereby increases the simulation speed. This reduces the size of all matrices, which decreases the time required to perform expensive matrix multiplications. In addition, by fixing control points, we reduce the number of mass points and springs that need to be examined during the physical simulation. That is, if a mass point's position is determined solely by fixed control points, it need not be processed by the numerical integration phase of the algorithm. Such modifications also speed up the simulation and enable the designer to make changes only in the localized region(s) of his/her interest without affecting a large portion of the solid object.

The necessary modifications to the matrices are fairly straightforward. We break the free and fixed portions of the model into two matrix–vector multiplications that we can then add together. When we fix a control point, we delete its components in **p** and its corresponding column in **A**. For each data point that becomes fixed, we remove its component in **d** as well as its corresponding rows in **A** and $\mathbf{f_d}$. We also delete its rows and columns in **M**, **D** and **K**. Note that we can consider a data point as "fixed" only when all of the control points that define its position are fixed. Specifically, we separate each of **p** and **A** into two matrices: $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{A}_1$ and $\mathbf{A}_2$, where $\mathbf{p}_1$ and $\mathbf{p}_2$ contain the positions of the free and fixed control points, respectively, and $\mathbf{A}_1$ and $\mathbf{A}_2$ contain the weights for the free and fixed control points, respectively. The familiar $\mathbf{d} = \mathbf{Ap}$ is replaced with $\mathbf{d} = \mathbf{A}_1\mathbf{p}_1 + \mathbf{A}_2\mathbf{p}_2$ so that the positions of the fixed points in $\mathbf{p}_2$ correctly influence the positions in **d**. Figure 11 illustrates this change. Vector $\mathbf{d}_1$ contains the positions of the free points from **d**. Note that there is no need to assemble a second matrix for the fixed points of **d**, since they will not appear in any matrix multiplication.

5.3 Data structures

For the dynamic B-spline solid, no complicated data structures were required to represent the model. Most information was stored in arrays and matri-

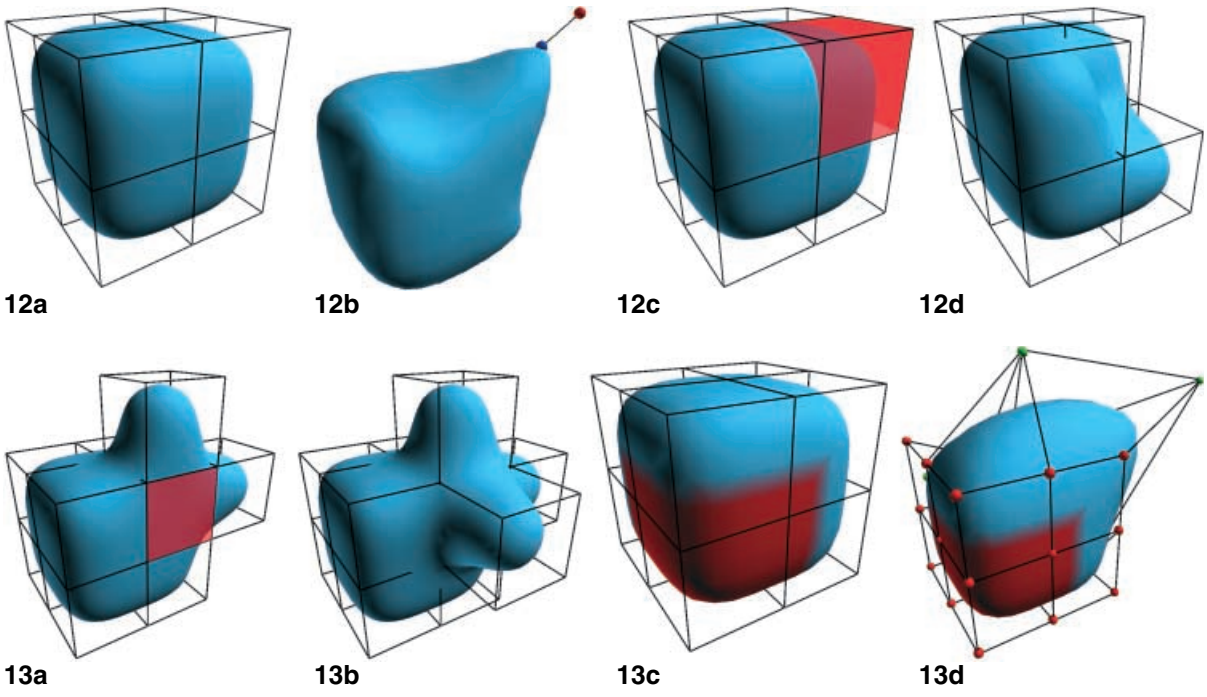


Fig. 12a–d. Original cube-like object **a** before and **b** after minor deformation caused by the rope tool. The user **c** highlights and then **d** deletes a cell of the control lattice

Fig. 13a–d. The user **a** highlights several faces and then **b** extrudes material. The user **c** fixes the left side, front side and underside of the solid and then **d** deforms it

ces. A subdivision solid requires a somewhat sophisticated data structure to store the adjacency and connectivity information of the lattice. For this purpose we used a modified version of the radial-edge data structure (Weiler 1986; Muuss et al. 1991). Given a point, edge, face or cell, this data structure can help one determine which components comprise which elements, which elements are adjacent to which elements, etc. This functionality is achieved through the use of numerous arrays of pointers.

5.4 Sculpting tools

Figures 12 and 13 demonstrate applications of some of the sculpting tools in our system. The main sculpting tool in our system is a non-compressible *rope tool* that permits the user to interactively deform the solid. Through the use of a three-dimensional input device, the user can select any vertex of the discretized lattice and drag it along any direction in three dimensions. Forces acting on the given mass point result in a solid deformation that behaves in a physically plausible manner. We undergo a simple linear search to determine which point in the lattice is the closest to the current location of the three-dimensional cursor.

We have implemented an extrusion tool for the subdivision solid that lets the user grow protrusions from the existing solid. After using the input device to select a face on the surface of the object, the user presses a button in the GUI, and the system creates a new cell in the control lattice and affixes it to the highlighted face. The size of the new cell is calculated based on the information about the existing cell to which it is attached, and it grows along the normal direction of the selected face. The use of a radial-edge data structure (see

Sect. 5.3) makes it easy to add and delete geometric primitives (points, edges, faces and cells) from the model.

Trimming is supported for the subdivision solid by allowing the user to select any cell in the control lattice and to subsequently remove it. This operation, as well as the extrusion process, entails reconstructing most of the data structures that represent both the geometry and physical properties of the object. The topology, on the other hand, can be updated quite easily because of the radial-edge data structure.

We have also implemented a useful and simple feature that permits the user to instruct the system to reset all of the rest lengths of the springs to their current lengths. This has the effect of re-defining the rest state of the system to its current state. In this way we can deform an elastic object but force it to take on a new rest shape at any time during the simulation. Essentially, this causes a normally elastic object to become temporarily plastic for the purpose of sculpting. Note that the rest lengths of the virtual springs must also be updated to their current lengths.

Control points can be fixed at run-time by moving the cursor to the desired point and pressing a button in the GUI. The matrices and other data structures are re-assembled on the fly. Other parameters, such as spring stiffness and damping factor, can be modified at run-time through the GUI.

5.5 Results and time performance

We document the various solids sculpted in our system and report their time performances in Tables 1 and 2. The test platform was a Microsoft Windows NT PC with a single Intel Pentium III 550 MHz CPU and 512 MB RAM. As one would expect, the update time grows linearly with an increase in the number of control points, since a mass–spring lattice lies at the core of the physical model.

Table 1. Various example B-spline solids deformed in our system, their sizes, and their time performance. Data points: the number of sampled points in the discretized B-spline volume. Update time: the number of milliseconds the program takes to execute one full time-step

Control points	Data points	Update time (ms)
$4 \times 4 \times 4$	$10 \times 10 \times 10$	40
$4 \times 4 \times 4$	$12 \times 12 \times 12$	61
$4 \times 5 \times 6$	$10 \times 10 \times 10$	41
$5 \times 5 \times 5$	$10 \times 10 \times 10$	61
$5 \times 5 \times 5$	$12 \times 12 \times 12$	92
$6 \times 6 \times 6$	$10 \times 10 \times 10$	61
$6 \times 6 \times 6$	$12 \times 12 \times 12$	111

6 Conclusions

We have presented a novel dynamic framework for deforming and sculpting free-form solid objects. The primary objective was to develop a physics-based manipulation approach for popular and powerful solid modeling. We have devised a dynamic algorithm and have implemented a system that permits physically plausible deformation and manipulation of *virtual clay* expressed by either B-spline or by subdivision solids. The use of a simple, efficient numerical solver enables the system to perform in real-time. The virtual springs minimize local distortion of the solid by resisting change in internal angles. Using the dynamic modeling approach, graphic modelers can naturally enforce various functional and aesthetic requirements on a free-form solid without the need to explicitly manipulate the control vertices. Therefore, general users are freed from comprehending the underlying complex geometric structure and abstract mathematical formulations. Moreover, they are provided with a set of intuitive tools that afford a natural interface for interacting with a solid object. Our experiments have demonstrated the applicability of the dynamic modeling techniques in solid modeling. Our formulations, algorithms, and system techniques are very general in the sense that they can be applied to arbitrary parametric or subdivision solids in a hierarchical and adaptive fashion. We envision several generalizations resulting from our current results in the near future, including extensions to other subdivision schemes and to other physical domains, such as fluid dynamics and heat transfer. Local or adaptive subdivision of a free-form solid would permit users to sculpt finer details and smaller features in localized regions of the limit solid. The use of an efficient implicit solver in our sculpting system would afford larger time-steps in physical simulation. More sophisticated sculpting tools would enable users to create a wider array of shape variations for free-form solids with ease.

Table 2. Various subdivision solids sculpted in our system, their sizes, and their time performance. Initial cells: the number of geometric elements in the control lattice; data cells: the number of elements in the finest subdivided lattice

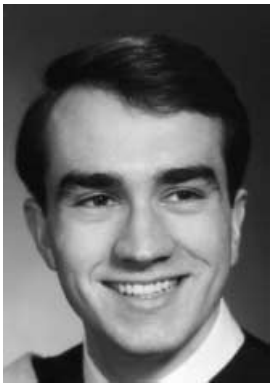
Model name	Sub. level	Initial cells	Data cells	Update time (ms)
Cube (4^3)	1	27	216	10
Cube (4^3)	2	27	1728	72
Torus	2	8	448	20
Torus	3	8	3584	122
Gear	2	16	960	42
Tetrahedron with holes	3	1	1744	61
Cube with holes	2	20	1280	61
Soccer player	2	24	1536	92
Cactus	2	26	1664	94
Spiked object	1	75	600	71
Holed object	1	135	1080	189

Acknowledgements. This research was supported in part by the NSF CAREER award CCR-9896123, the NSF grant DMI-9896170, the NSF ITR grant IIS-0082035, the GAANN grant P200A9703199, and a research grant from the Ford Motor Company.

References

1. Catmull E, Clark J (1978) Recursively generated B-spline surfaces on arbitrary topological meshes. Comput Aided Des 10:350–355
2. Chaikin G (1974) An algorithm for high speed curve generation. Comput Graph Image Process 3:346–349
3. Dachille IX F, Qin H, Kaufman A, El-Sana J (1999) Haptic sculpting of dynamic surfaces. In: 1999 Symposium on Interactive 3D Graphics. ACM Press, New York, pp 103–110
4. DeRose T, Kass M, Truong T (1998) Subdivision surfaces in character animation. Comput Graph (SIGGRAPH '98 Proc.) 32:85–94
5. Doo D (1978) A subdivision algorithm for smoothing down irregularly shaped polyhedrons. In: Proceedings on Interactive Techniques in Computer Aidied Design, Bologna, Italy. IEEE, New York, pp 157–165
6. Dyn N, Levin D, Rippa S (1990) Data dependent triangulations for piecewise linear interpolation. IMA J Numer Anal 10(1):137–154
7. Farin G (1997) Curves and surfaces for computer aided geometric design: a practical guide, 4th edn. Academic, San Diego
8. Loop C (1987) Smooth subdivision surfaces based on triangles. Master's thesis, Department of Mathematics, University of Utah
9. MacCracken R, Joy KI (1996) Free-form deformations with lattices of arbitrary topology. Comput Graph (SIGGRAPH '96 Proc.) 30:181–188
10. Mandal C, Qin H, Vemuri BC (1999) A novel FEM-based dynamic framework for subdivision surfaces. Solid Model 99:191–202
11. Metaxas D, Terzopoulos D (1992) Dynamic deformation of solid primitives with constraints. Comput Graph (SIGGRAPH '92 Proc.) 26:309–312
12. Mortenson ME (1997) Geometric modeling, 2nd edn. Wiley Computer Publishing, New York
13. Muuss MJ, Butler LA (1991) Combinatorial solid geometry, boundary representations, and n-manifold geometry. In: Rogers DF, Earnshaw RA (eds) State of the art in computer graphics: visualization and modeling. Springer, Berlin Heidelberg, pp 185–223
14. Pentland A, Williams J (1989) Good vibrations: modal dynamics for graphics and animation. Comput Graph (SIGGRAPH '89 Proc.) 23:215–222
15. Qin H, Terzopoulos D (1996) D-NURBS: a physics-based framework for geometric design. IEEE Trans Vis Comput Graph 2(1):85–96
16. Qin H, Mandal C, Vemuri BC (1998) Dynamic Catmull–Clark subdivision surfaces. IEEE Trans Vis Comput Graph 4(3):215–229
17. Requicha AAG, Rossignac JR (1992) Solid modeling and beyond. IEEE Comput Graph Appl 12(5):31–44
18. Sederberg T, Zheng J, Sewell D, Sabin M (1998) Non-uniform recursive subdivision surfaces. Comput Graph (SIGGRAPH '98 Proc.) 32:387–394
19. Terzopoulos D, Fleischer K (1988) Deformable models. Visual Comput 4(6):306–331
20. Terzopoulos D, Qin H (1994) Dynamic NURBS with geometric constraints for interactive sculpting. ACM Trans Graph 13(2):103–136
21. Terzopoulos D, Platt J, Barr A, Fleischer K (1987) Elastically deformable models. Comput Graph (SIGGRAPH '87 Proc.) 21:205–214
22. Weiler KJ (1986) Topological structures for geometric modeling. PhD thesis, Rensselaer Polytechnic Institute, Troy, New York
23. Zorin D, Schröder P, Sweldens W (1996) Interpolating subdivision for meshes with arbitrary topology. Comput Graph (SIGGRAPH '96 Proc.) 30:189–192

Photographs of the authors and their biographies are given on the next page.



KEVIN T. MCDONNELL is a Ph.D. candidate in Computer Science at the State University of New York at Stony Brook, where he is also a Research Assistant in the SUNYSB Center for Visual Computing (CVC). In 1998 he was awarded both a University Graduate Research Fellowship and a GAANN fellowship (Graduate Assistance in Areas of National Need). He graduated *summa cum laude* with a B.S. in both Computer Science and Applied Mathematics from SUNY at Stony Brook in 1998. He received an M.S. in Computer Science in 2001, also from Stony Brook. His research interests include physics-based modeling, geometric modeling, scientific visualization, and interactive three-dimensional graphics. He is a member of Phi Beta Kappa, the Golden Key National Honor Society, and ACM/SIGGRAPH. For more information see http://www.cs.sunysb.edu/∼ktm.

DR. HONG QIN is an Associate Professor of Computer Science at the State University of New York at Stony Brook, where he is also a member of the SUNYSB Center for Visual Computing. He received his B.S. degree (1986) and his M.S. degree (1989) in Computer Science from Peking University in Beijing, P.R. China. He received his Ph.D. degree (1995) in Computer Science from the University of Toronto (UofT). From 1989–90 he was a research scientist at North-China Institute of Computing Technologies. From 1990–91 he was a Ph.D. candidate in Computer Science at the University of North Carolina at Chapel Hill. From 1996–97, he was an Assistant Professor of Computer & Information Science & Engineering at the University of Florida. He received the Honor Student Award from 1983–85 and the Best Graduate Award in 1986 from Peking University. During his years at UofT, he received a UofT Open Doctoral Fellowship. In 1997, Dr. Qin was awarded the NSF CAREER Award from the National Science Foundation (NSF), and, in September 2000 was awarded a newly established NSF Information Technology Research (ITR) grant. In December 2000 he received a Honda Initiation Grant Award and in April 2001 was selected as an Alfred P. Sloan Research Fellow by the Sloan Foundation. He is a member of ACM, IEEE, SIAM, and Eurographics.