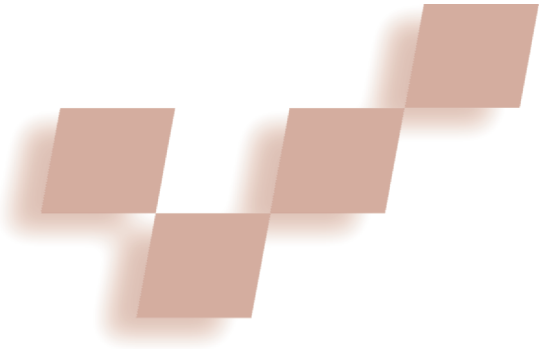


Scalar-Function-Driven Editing on Point Set Surfaces

Xiaohu Guo, Jing Hua, and Hong Qin
State University of New York at Stony Brook

A modeling paradigm and system for local and global editing on point set surfaces lets users easily modify the topology of sculpted point sets.

Three-dimensional acquisition is an increasingly popular means of creating surface models. As 3D digital photographic and scanning devices produce higher resolution images, acquired geometric data sets grow more complex in terms of the modeled objects' size, geometry, and topology. *Point-based geometry* is a form of 3D content acquisition popular in graphics and related visual computing areas. Points have unique advantages over traditional primitives such as triangle meshes. For example, they are free of connectivity concerns, especially in large-scale scanned models.

Point set surfaces are enjoying a renaissance in modeling and rendering, with many efforts focused on direct rendering techniques[1–3] and effective modeling mechanisms.[4,5] Despite these research achievements, the spectrum of surface editing and deformation types is still quite limited. For example, no current research investigates the physics-based dynamic simulation of virtual materials represented by point sets (that is, allowing the point sets to respond dynamically to users' applied force in an intuitive and natural fashion). Handling collision detection and topology changes using point information alone is another challenging problem.

We've developed a scalar-field-driven editing paradigm and system for point set surfaces that let users manipulate and sculpt point clouds intuitively and efficiently. The paradigm is primarily based on the representation of versatile, embedded scalar fields associated with any region of the point set surface. Scalar-field-driven implicit representation, such as volumetric implicit models[6] and level sets, is a powerful paradigm that not only handles arbitrary topology and complicated geometry, but also affords physics-based modeling techniques.

After constructing the surface distance field from the point clouds,[7] we incorporate the dynamic implicit volumetric model[6] into the point-based geometry deformation. Level-set editing lets us incorporate scalar-field guided free-form deformations (FFD)[8] into our surface deformation system to further expand local and global surface editing functionality. Using digital topology information, we can easily handle collision detection and the point set surface's changing topology. We've also integrated a haptics interface into our surface-modeling framework. Experiments show that the proposed paradigm complements current point set surface modeling techniques with a more powerful editing scheme.

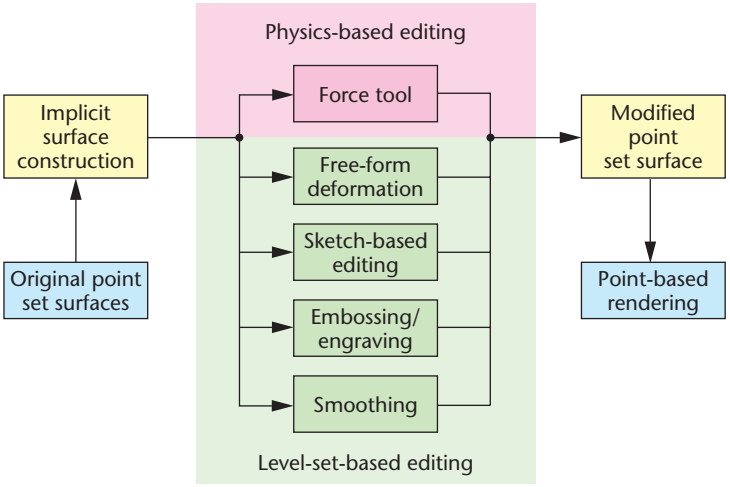
System overview

Figure 1 shows the data flow and architecture of our scalar-function-driven point set surface-editing system. The system takes any region of the original point set surface as input, using physics-based dynamic volumetric deformation and level-set-based editing techniques to deform the point set locally or globally and change its topology when needed. It then uses point-based surface-rendering techniques to render the modified surface.

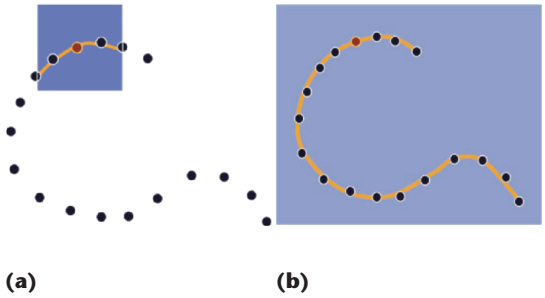
We can use several existing techniques to convert user-specified regions of the original point set surfaces into an implicit surface representation. (Even after this process, we need to keep the original point-sampled geometry for downstream procedures such as local editing, global free-form deformation, topological change, and rendering.) Dual representations facilitate the entire data-processing pipeline, as Figure 1 shows.

After constructing the implicit surface, we integrate the physics-based dynamic volumetric deformation method and level-set surface-editing techniques into our system to provide editing operations such as force-based sculpting, free-form deformation, sketch-based editing, and embossing or engraving. While incorporating the level-set operators, we integrate scalar-field free-form deformation (SFFD) techniques[8] into the level-set framework to facilitate both local and global surface editing. After the system deforms the underlying implicit surface at each iteration step, we update the point set surface accordingly. In addition to geometric deformation, we consider collision detection and topology change, as we discuss later.

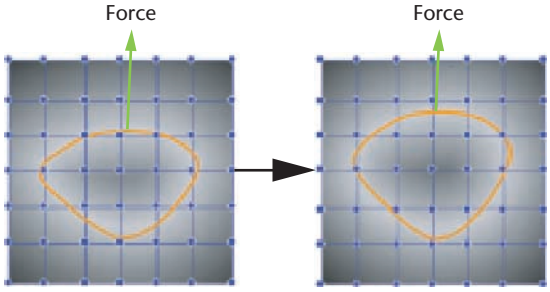
Published by the IEEE Computer Society 0272-1716/04/$20.00 © 2004 IEEE



1 System editing framework and data pipeline.



2 Distance fields: (a) Local distance field in each point sample's neighborhood and (b) global distance field.



3 We use the force tool to intuitively deform the global scalar field. The yellow curves are the zero-level sets of isosurfaces.

Dynamic point set surface

Our scalar-field-driven dynamic point set surface collectively takes advantage of the topological strength of implicit surfaces, the modeling power of physics-based simulation techniques, and the simplicity of point-sampled surfaces.

Surface distance field

The input data consists of an unstructured point cloud $\mathbf{P} = \{\mathbf{p}_i \mid 1 \le i \le N\}$ that uniquely characterizes the geometry of an underlying manifold surface $\mathbf{S}$.

In previous work,[9] we fit a volumetric implicit function to the local distance field in the neighborhood of the point sample $\mathbf{p}_i$, as Figure 2a shows. We used scalar trivariate B-spline functions as the underlying shape primitives. Implicit B-spline functions have several attractive qualities: simplicity, generality, local support, ease of evaluation, and so on. These trivariate functions take the following form:

$$s(u,v,w) = \sum_{i=0}^{l-1}\sum_{j=0}^{m-1}\sum_{k=0}^{n-1} P_{ijk} B_{i,r}(u) C_{j,s}(v) D_{k,t}(w)$$

where $B_{i,r}(u)$, $C_{j,s}(v)$, and $D_{k,t}(w)$ are the uniform B-spline basis functions of degrees $r-1$, $s-1$, and $t-1$. P_{ijk} is the scalar coefficient in a volumetric mesh of size $l \times m \times n$, and $s(u, v, w)$ is a scalar function at position (u, v, w) in the parametric domain. In this case, the scalar function defines the distance of position (u, v, w) to the surface. The local reference parameter domain can enclose the k-nearest neighbors of $\mathbf{p}_i$. Blending the local implicit primitive associated with each point sample using Shepard's[7,10] blending techniques produces the global continuous distance field shown in Figure 2b:

$$s(x,y,z) = \frac{\sum_{i=1}^{N} s_i(x,y,z)\phi_i(x,y,z)}{\sum_{i=1}^{N} \phi_i(x,y,z)}$$

If (x, y, z) is inside the local region of point sample $\mathbf{p}_i$, we use the trivariate implicit function associated with $\mathbf{p}_i$ for a distance value of $s_i(x, y, z)$; if it is outside this region, we simply set the value to zero. Also, $\phi_i(x, y, z)$ is a smooth, positive, and monotonically decreasing weighting function associated with $\mathbf{p}_i$. More detailed information on our surface distance field construction is available elsewhere.[9]

Other researchers have also used locally fitted implicit function methods.[7,10] Ohtake et al. blended piecewise quadratic functions using an adaptive octree subdivision method, an efficient approach to constructing an implicit surface from large point sets.[7] Xie at al. applied Shepard's method to blend locally fitted quadric fields, using a prioritized front growing scheme to handle highly noisy point clouds.[10]

Physics-based dynamic local editing

Our approach uses dynamic volumetric models[6] to allow intuitive sculpting of the point-based distance field. We resample the surface scalar field in a global region of interest, and then discretize the global scalar field into a voxel raster. Every voxel contains a scalar value sampled at each grid position. While sculpting or deforming the scalar field, we simulate the dynamics in this global region. In each simulation step, if we change the global scalar field on the global grids, we must update each point sample's positions and perform dynamic sampling on the point set surface. We model the discretized scalar field as a collection of mass points connected by a network of special springs across nearest-neighbor voxels. We use the straightforward density spring model (a special type of spring that attracts or repels its neighbors' density values) to simulate the model's dynamics.[6] Figure 3 shows how we can apply the force tool to intuitively deform the global scalar field.

We formulate the discretized scalar field's motion equation as a discrete simulation of Lagrangian dynamics:

$$\mathbf{M}\ddot{\mathbf{d}} + \mathbf{D}\dot{\mathbf{d}} + \mathbf{K}\mathbf{d} = \mathbf{f}_d$$

where **M** is the mass matrix, **D** is the damping matrix, **K** is the stiffness matrix, and $\mathbf{f}_d$ is the external force vector. We use d to denote the scalar grid value. The connecting springs—each with force $f = k\,(I - I_0)$, according to Hooke's law, which states that a spring's restoring force is proportional to how far it is stretched—generate the internal forces. Unlike other pure geometric surface-sculpting methods (see the "Background on Point Set Geometry" sidebar), our method synchronizes the geometric and physical representations of modeled objects and lets users deform the surface quickly and easily in a physically plausible fashion. More details on dynamic volumetric models, such as applied-force computation and numerical integration, are available elsewhere.[6]

In our haptic implementation, users can perform the deformation just inside any sculpting region of the object. The region is independent of the surface definition and can be much smaller than the entire surface. Limiting the deformation to the inside of the surface region of interest can thus help the system achieve real-time performance.

Level-set formulation for point geometry

Deformable isosurfaces implemented through level-set methods have great potential in fields such as visualization, scientific computing, and computer graphics. An implicit surface consists of all points $S = \{\mathbf{x}(t) \,|\, \phi(\mathbf{x}, t) = k\}$, where $\phi(\mathbf{x}, t)$ is a time-varying scalar function embedded in 3D. Level-set methods relate the implicit surface's motion to a partial differential equation defined on the associated volume by

$$\frac{\partial \phi}{\partial t} = -\nabla\phi \cdot \frac{d\mathbf{x}}{dt} \tag{1}$$

where $\nabla\phi$ denotes the implicit surface's gradient and $d\mathbf{x}/dt$ describes the level-set surface's motion. We can rewrite Equation 1 as

$$\frac{\partial \phi}{\partial t} = \left|\nabla\phi\right| F\left(\mathbf{x}, \phi, \nabla\phi, \ldots\right)$$

where $F(\mathbf{x}, \phi, \nabla\phi, \ldots) \equiv -\nabla\phi / |\nabla\phi| \times d\mathbf{x}/dt$ is the speed function.

Using the level-set method, we can implement a wide range of deformations by defining an appropriate speed function using a combination of data-dependent terms and geometric measures (curvature, for example) of the implicit level-set surface[11]: $F = F_A + F_G$. F_A is the advection, which expands or contracts the front with speed F_A depending on its sign (the advection independent of the moving front's geometry). F_G depends on the front's geometry, such as its local curvature, and can be used to smooth high curvature regions in the front. We not only use this representation to perform the level-set-based editing operations on the point set surfaces, but also integrate SFFD[8] into our level-set framework.

Background on Point Set Geometry

In recent years, considerable research has sought to efficiently represent, model, process, and render point-sampled geometry.

Rusinkiewicz and Levoy[1] introduced QSplat, a technique that uses a hierarchy of spheres of different radii to approximate and display a high-resolution model with level-of-detail control. Zwicker et al.[2] introduced *surface splatting* for directly rendering opaque and transparent surfaces from point clouds without connectivity. In later work, they present Pointshop 3D,[3] a system for interactive shape and appearance editing of 3D point-sampled geometry. The system's key ingredients are a point cloud parameterization and a dynamic resampling scheme based on continuous reconstruction of the surface. Their geometry editing is limited to normal displacement, however.

Alexa et al.[4] use the moving least squares (MLS) projection framework to approximate a smooth surface defined by a set of points, and they introduce associated resampling techniques to generate an adequate representation of the surface.

Pauly et al.[5] present a free-form shape-modeling framework for point-sampled geometry using the implicit surface definition of the MLS approximation, but their approach is limited to free-form deformation and doesn't offer physics-based surface-modeling techniques. Most recently, Guo and Qin[6] present a physics-based dynamic local sculpting paradigm for point-sampled surfaces using volumetric implicit functions.

References

1. S. Rusinkiewicz and M. Levoy, "QSplat: A Multiresolution Point Rendering System for Large Meshes," *Proc. Siggraph*, ACM Press, 2000, pp. 343-352.
2. M. Zwicker et al., "Surface Splatting," *Proc. Siggraph*, ACM Press, 2001, pp. 371-378.
3. M. Zwicker et al., "Pointshop 3D: An Interactive System for Point-Based Surface Editing," *Proc. Siggraph*, ACM Press, 2002, pp. 322-329.
4. M. Alexa et al., "Computing and Rendering Point Set Surfaces," *IEEE Trans. Visualization and Computer Graphics*, vol. 9, no. 1, Jan.-Mar. 2003, pp. 3-15.
5. M. Pauly et al., "Shape Modeling with Point-Sampled Geometry," *Proc. Siggraph*, ACM Press, 2003, pp. 641-650.
6. X. Guo and H. Qin, "Dynamic Sculpting and Deformation of Point Set Surfaces," *Proc. Pacific Graphics*, IEEE CS Press, 2003, pp. 123-130.

Because we're only interested in surface editing the point set that relies on the isosurfaces' zero level-set, we need only compute the parts of the solution that are adjacent to the moving surface. In addition, we directly apply our operations to the point set, so the level-set embedding occurs wherever the deformation is needed. Users interactively control the level-set space and the deformation operators within the space.

We use Whitaker's sparse-field method[12] to update the wavefront and several layers around it via a simple city block distance metric at each iteration. We call the set of grid points adjacent to the level set the *active set*, denoting it by L_0, and defining its neighborhoods in layers, $L_{+1}, \ldots, L_{+l}, \ldots, L_{+N}$ and $L_{-1}, \ldots, L_{-l}, \ldots, L_{-N}$, where l indicates the city block distance from the nearest active grid point (negative numbers represent outside layers). In this article, we use up to the second-order derivatives of ϕ, so we need only five lay-

ers: L_2, L_1, L_0, L_{-1}, and L_{-2}. The sparse field method updates the level-set geometry by computing and manipulating the layers' data structure at each iteration. Computational time complexity grows proportionally to the surface region area undergoing the deformation. As described earlier, local level-set embedding and point-set geometry simplicity make deformation extremely fast. (Conventional level-set methods often build a global grid structure for the entire volumetric space where the surface resides.)

Scalar-field free-form deformation

Free-form deformation is used extensively in shape modeling and animation because it doesn't require knowledge about the embedded object's underlying geometric representation and topological structure. We could perform free-form deformation directly on the point set surface fairly easily. However, we'd then have to rely on the moving least squares surface projection for the exact positions of the newly inserted points. Point insertion is unavoidable in large deformations and expands the model significantly. Consequently, gaps occur at the current resolution. To address this problem, we perform free-form deformation directly on the global scalar field, which we can then use to guide both point movement and point insertion in a single, unified fashion.

Our approach uses SFFD.[8] Because SFFD is founded on flow constraints based on a partial differential equation, we can easily use the velocity field obtained by SFFD to update our point surface's global scalar field. Furthermore, introducing intermediate steps during deformation lets us perform dynamic sampling on the point set surface to maintain a good surface quality throughout the model-sculpting session. Our representation consists of two classes of scalar fields:

- s_s, or surface representation, is the global scalar field we acquire from point clouds using the multilevel partition of unity (MPU) implicit surface construction method[6]; and
- s_t is the scalar field of the deformation tool,[8] such as a sketched point or curve skeleton.

We use the tool scalar field s_t to perform deformations on the point surface scalar field s_s. In our global free-form deformation, discretized voxel grids store both s_s and s_t, which we update using the sparse-field method at each iteration step.

During deformation of the tool scalar field s_t, we assume that the vertices on the discretized voxel grids are constrained on their original level set. We therefore represent the trajectory of these grid vertices as $\{\mathbf{x}(t) \mid s_t(\mathbf{x}(t), t) = c\}$. The derivative of $s_t(\mathbf{x}(t), t)$ with respect to time yields

$$\nabla_{\mathbf{x}} s_t \cdot \frac{d\mathbf{x}}{dt} + \frac{\partial s_t}{\partial t} = 0$$

where $\nabla_{\mathbf{x}} s_t$ is the gradient of s_t at $\mathbf{x}$. To get the general velocity along the three coordinate axes of the 3D space (v_x, v_y, v_z), taking into account the deformation motion's smoothness, we add a smoothness constraint on the underlying level-set surface s_s by minimizing the objective function:

$$E = \int \left(\nabla_{\mathbf{x}} s_t \cdot \mathbf{v} + \frac{\partial s_t}{\partial t} \right)^2 + \lambda \left(\left| \nabla \mathbf{v} \right| \right)^2 d\mathbf{x} \qquad (2)$$

where λ is a Lagrange multiplier. To discretize the objective function in Equation 2, we consider a voxel grid vertex i, where $i \in L_{-2} \cup L_{-1} \cup L_0 \cup L_1 \cup L_2$ (the sparse-field layers), and its adjacent neighboring voxel grids N_i, where

$$N_i = \left\{ j \left| \overline{ij} \in C_6, j \in L_{-2} \cup L_{-1} \cup L_0 \cup L_1 \cup L_2 \right. \right\}$$

and C_6 is the set of six connected voxel grid pairs. This lets us transform the objective function (Equation 2) into

$$E = \sum_i (c(i) + \lambda s(i))$$

where $c(i)$ is the flow constraint approximation error:

$$c(i) = \left(\frac{\partial s_t}{\partial x} v_x(i) + \frac{\partial s_t}{\partial y} v_y(i) + \frac{\partial s_t}{\partial z} v_z(i) + \frac{\partial s_t}{\partial t} \right)^2$$

and $s(i)$ is the discretized smoothness factor computed as the velocity difference between the voxel grid vertex and its adjacent neighbors:

$$s(i) = \frac{1}{|N_i|} \sum_{j \in N_i} \begin{bmatrix} (v_x(i) - v_x(j))^2 + (v_y(i) - \\ v_y(j))^2 + (v_z(i) - v_z(j))^2 \end{bmatrix}$$

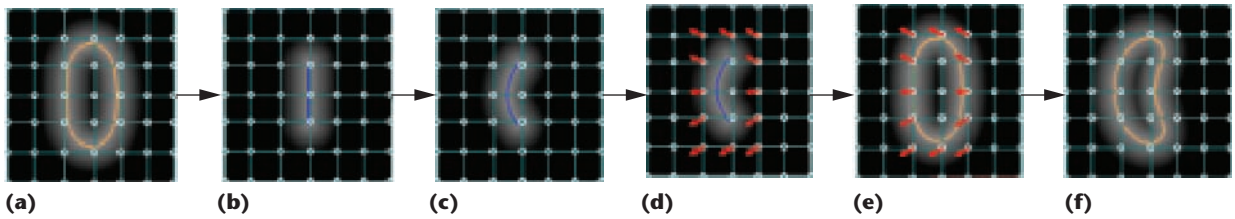
Here, $|N_i|$ is the number of voxel grid vertices in N_i. Satisfying $\partial E/\partial v_x(i) = 0$, $\partial E/\partial v_y(i) = 0$, and $\partial E/\partial v_z(i) = 0$ lets us minimize the objective function E and obtain the iterative solution:

$$\left[v_x, v_y, v_z \right]^{\mathrm{T}} = \left[\bar{v}_x, \bar{v}_y, \bar{v}_z \right]^{\mathrm{T}} - \mu \left[\frac{\partial s_t}{\partial x}, \frac{\partial s_t}{\partial y}, \frac{\partial s_t}{\partial z} \right]^{\mathrm{T}} \qquad (3)$$

where $(\bar{v}_x, \bar{v}_x, \bar{v}_x)$ is the average velocity of the adjacent voxel grids, and

$$\mu = \frac{\frac{\partial s_t}{\partial x} \bar{v}_x + \frac{\partial s_t}{\partial y} \bar{v}_y + \frac{\partial s_t}{\partial z} \bar{v}_z + \frac{\partial s_t}{\partial t}}{\lambda + \left(\frac{\partial s_t}{\partial x} \right)^2 + \left(\frac{\partial s_t}{\partial y} \right)^2 + \left(\frac{\partial s_t}{\partial z} \right)^2}$$

Although we obtain the velocity field associated with each voxel grid according to changes in the tool scalar field s_t, we don't change their positions. Instead, we use the velocity field $\mathbf{v}$ to update the surface scalar field s_s by defining the speed function as



4 Scalar fields: (a) The surface scalar field s_s representing the point set surface; (b) the initial tool scalar field s_t; (c) the deformed tool scalar field; (d) the velocity (red arrow) field computed according to the tool scalar field's deformation; (e) the velocity field, which we use to deform the surface scalar field; and (f) the deformed surface scalar field.

$$F = -\frac{\nabla s_s}{|\nabla s_s|} \cdot \mathbf{v} \tag{4}$$

Using these formulations, we design our level-set-based global free-form deformation as an evolution process:

- We generate a tool scalar field using skeletons or sketches.
- We track down the original tool scalar values s_{to} at the voxel grid positions.
- After the user alters the tool scalar field, we find their final tool scalar values s_{tf}.
- We then have $s_{tf} = s_{to} + k\Delta s_t$, where k is a user-specified number of steps taken to deform s_t, and Δs_t is the evolution step size of the tool scalar field.

The SFFD algorithm for point geometry is as follows: At each time step m ($m = 0 \ldots k$),

1. Update the tool scalar field by: $s_t = s_{to} + m\Delta s_t$, then $\partial st/\partial t = \Delta s_t$;
2. Calculate $\partial st/\partial x$ using finite differences;
3. Initiate the sparse-field layer voxel grid velocities to be 0;
4. Deduce the current velocity field by iteration using Equation 3 (converging the velocities typically requires three to four iterations);
5. Deduce the surface scalar field updating rate using Equation 4 and obtain the updated surface scalar field s_s;
6. Update points' positions and perform dynamic sampling.
7. If $m = k$, terminate the deformation process; otherwise, proceed to the next time step $m + 1$ and repeat steps 1 through 6.

Figure 4 shows how we deform the surface scalar field using the tool scalar field deformation. In the figure, yellow curves denote the isosurface of the surface scalar field associated with the point set surface; blue curves denote the skeleton-based tools for defining the tool scalar field; and red arrows denote the velocities evaluated at the sparse-field layer voxel grids.

Dynamic update and sampling

After user interaction modifies the global distance field, we must change the points' locations because the point samples are assumed to be on the underlying implicit function's zero level set. When deforming the distance-field space, we represent the point sample trajectory as $\{\mathbf{x}(t) \,|\, s_s(\mathbf{x}(t), t) = 0\}$, where $\mathbf{x}(t)$ is the point's parameter position and s_s is the point surface's global scalar field. The derivative of $s_s(\mathbf{x}(t), t)$ with respect to time yields:

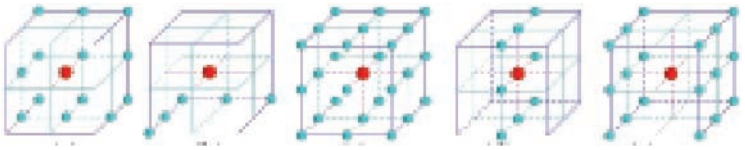
$$\nabla_{\mathbf{x}} s_s \cdot \mathbf{v} + \frac{\partial s_s}{\partial t} = 0 \tag{5}$$

where $\nabla_{\mathbf{x}} s_s$ denotes the gradient $\partial s_s(\mathbf{x}(t), t)/\partial \mathbf{x}$. Because both $\nabla_{\mathbf{x}} s_s$ and $\mathbf{v}$ are vectors, no unique solution for the point velocity exists. If we divide $\mathbf{v}$ into three components $\mathbf{v} = \mathbf{v_n} + \mathbf{v_t} + \mathbf{v_w}$ along three orthogonal directions $\mathbf{n}, \mathbf{t}, \mathbf{w}$, where $\mathbf{n} = -\nabla_{\mathbf{x}} s_s / |\nabla_{\mathbf{x}} s_s|$ represents the unit principle normal vector of the distance field's isosurface, $\mathbf{t}$ represents the unit tangent vector, and $\mathbf{w}$ represents the unit binormal vector, the dot product $\nabla_{\mathbf{x}} s_s \cdot \mathbf{v}$ in Equation 5 retains only the component containing $\mathbf{v_n}$. Therefore, if we assume a point only moves in its normal direction, we get its normal velocity by

$$\mathbf{v_n} = \frac{\frac{\partial s_s}{\partial t}}{|\nabla_{\mathbf{x}} s_s|} \mathbf{n} \tag{6}$$

We use this velocity to update the point position by advancing to the next time step through the forward Euler method $\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v_n}\Delta t$.

The point sampling density changes as users perform surface sculpting or deformation. To maintain surface quality, we insert new sample points when the surface density becomes too low. Alternatively, we simplify the surface by eliminating points when the surface is squeezed. We use the farthest point sampling scheme[5] for point insertion. In each modeling step, each point checks its neighboring density by projecting its neighbor points onto its tangent plane. Next, we compute the points' Voronoi diagram using the Voronoi vertex with the largest circle radius on the tangent plane. If the radius is larger than a specified threshold, we project the vertex onto the global scalar field's isosurface. This gives us a surface density that is locally near uniform. Meanwhile, we can reduce the sampling density using various methods.[4] We implement an iterative simplification method similar to one introduced in ear-



(a) (b) (c) (d) (e)

5 Characterization of the center grid (in red) with respect to its neighborhood: **(a)** $n_{int} = 1$, $n_{ext} = 1$; **(b)** $n_{int} = 0$, $n_{ext} = 1$; **(c)** $n_{int} = 1$, $n_{ext} = 0$; **(d)** $n_{int} = 1$, $n_{ext} = 2$; **(e)** $n_{int} = 2$, $n_{ext} = 1$.

lier work.[4] However, instead of using a quadric error metric, we simply replace two points with their middle point and project this new point onto the global scalar field isosurface. A relaxation stage can follow the up- and down-sampling process, using a simple point repulsion scheme to obtain a more uniform sampling pattern.

Topology change handling

For our level-set-based surface-editing tools to change the shape and topology of a point set surface, the surface must be able to change its topology properly whenever it detects a collision. The underlying level-set surface contour automatically changes topology (for example, by merging or splitting) without an elaborate mechanism. However, the topology of the point set surface requires explicit handling. For example, unless we delete the points in the intersection region, we will have little control over the movement of points in the intersection region. In the topology change region, for example, the surface scalar field gradient $\nabla_{\mathbf{x}} s_s$ might change to 0 during the iterations. In addition, because Equation 6 guides point sample movement, we need a robust method to predict where topology changes will occur, and then delete the point samples in those regions.

Bischoff and Kobbelt control the level-set contour topology by adding topological information to the volume representation with the clear goal of locally resolving topological ambiguities.[13] Their novel method is useful for collision detection of the level-set contour. Tasks in our current work differ from theirs, and we focus mainly on the robust and proper topological modification of the point-sampled geometry undergoing deformation. We therefore simply change the topology when the level-set surface detects a collision, even though integrating Bischoff and Kobbelt's setting into our framework to preserve the topology of the underlying point set surface is quite straightforward.[13]

Digital topology. We use fundamental digital topology techniques to detect collisions in a level-set context.

A grid point $\mathbf{u} \in Z^3$ is a 6-, 18-, 26-neighbor of another grid $\mathbf{v} \in Z^3$ if $\|v - u\|_2 \le 1, \sqrt{2}, \sqrt{3}$, respectively. We denote the n-neighborhood ($n \in \{6, 18, 26\}$) of $\mathbf{v}$ by $N_n(\mathbf{v}) = \{\mathbf{u} | \mathbf{u}$ is an n-neighbor of $\mathbf{v}\}$, and set $N_n^*(\mathbf{v}) = N_n(\mathbf{v}) \backslash \{\mathbf{v}\}$.

Let $\mathbf{v} \in Z^3$ be a grid point and $V \subset Z^3$ be a grid set. V is an interior grid; its complement, $\bar{V} = Z^3 \backslash V$ is exterior. We define the geodesic n-neighborhood of order k of a grid $\mathbf{v}$ with respect to $V \subset Z^3$ recursively by

$$N_n^1(\mathbf{v}, V) = V \cap N_n(\mathbf{v}),$$

$$N_n^{k+1}(\mathbf{v}, V) = V \cap \{N_n(\mathbf{u}) \,|\, \mathbf{u} \in N_n^k(\mathbf{v}, V)\}$$

By denoting the number of n-connected components of a grid set $V \subset Z^3$ by $c_n(V)$, we explicitly define the topological numbers n_{int} and n_{ext} as

$$n_{int}(\mathbf{v}, V) = c_6\left(N_6^2(\mathbf{v}, V) \cap N_{26}^*(\mathbf{v})\right)$$

$$n_{ext}(\mathbf{v}, V) = c_{26}\left(\bar{V} \cap N_{26}^*(\mathbf{v})\right)$$

where $n_{int}(\mathbf{v}, V)$ is the number of interior components of V that touch $\mathbf{v}$, and $n_{ext}(\mathbf{v}, V)$ the number of exterior components.

A grid point $\mathbf{v}$ is simple with respect to $V \subset Z^3$ if V and $V \cup \{\mathbf{v}\}$ have the same number of components, handles, and cavities[13]; otherwise, $\mathbf{v}$ is complex. Furthermore, $\mathbf{v}$ is simple with respect to V if and only if

$$n_{int}(\mathbf{v}, V) = n_{ext}(\mathbf{v}, V) = 1^{14} \quad (7)$$

More theoretical results on digital topology are available elsewhere.[13,14]

Collision detection. For clarity, we only consider situations in which the surface advances outward; when a surface moves inward, we simply switch the inside and outside of the surface region. We call a grid point on the sparse field's active layer L_0 or inside the surface region *conquered*. When a grid point is about to be conquered (that is, about to change its status from outside layer L_{-1} to active layer L_0), we test it for simplicity using Equation 7. If it's simple, the sparse-field algorithm proceeds as usual. If it isn't, we must explicitly resolve the point set surface topology change—that is, delete the point samples residing in the grid's near vicinity.

Similar to Bischoff and Kobbelt,[13] we distinguish five cases, shown in Figure 5. We handle the topological change and resolve the topological ambiguity only when $n_{int} \ge 2$—that is, when two parts of the front collide. Figure 6 illustrates collision detection and topological modification. Grid $\mathbf{v}$ is added to L_0 in the current iteration step. The topology number n_{int} for $\mathbf{v}$ equals 2. We therefore delete the point samples residing in the prescribed vicinity of grids $\mathbf{u}$ and $\mathbf{w}$ (users can identify the affected region interactively or automatically depending on their specified parameters, such as each grid's influence factor). We also place a tag between $\mathbf{v}$ and $\mathbf{w}$, so the system will consider them disconnected when checking topology numbers for nearby grid points.

Editing toolkits

Our sculpting system provides various tools for flexibly performing many modeling operations on point set surfaces.

Force-based tools

Using our force-based tools, users can select any point location inside the sculpting region and simulate the dynamics on the mass points inside the region. A user-defined function, which can be Gaussian, constant, spherical, or any other distribution, distributes the force among nearby mass points. In addition to the point-based rope tool,[15] our system offers a curve-based rope tool for applying force along any user-defined curve with any distribution mode. Figure 7a shows how we use the point-based rope tool to deform the rabbit model's surface. Figure 7b shows a curve-based force tool for sculpting the rabbit's head.

Free-form deformation tools

Our system lets users perform free-form deformations on point set surfaces by interactively sketching skeletons using a mouse or 3D pointing device. The system generates the tool scalar field as the blending of field functions g_i of a set of skeletons $t_i (i = 1, \ldots, N)$:

$$s_t(x,y,z) = \sum_{i=1}^{N} g_i(x,y,z)$$

where skeletons t_i can be any geometric primitive, such as blobs and curves. The field functions g_i are decreasing functions of the distance to the associated skeleton $g_i(x, y, z) = F_i(d(x, y, z, t_i))$, where $d(x, y, z, t_i)$ is the distance between (x, y, z) and t_i, and we can define F_i using polynomials or more sophisticated anisotropic functions.

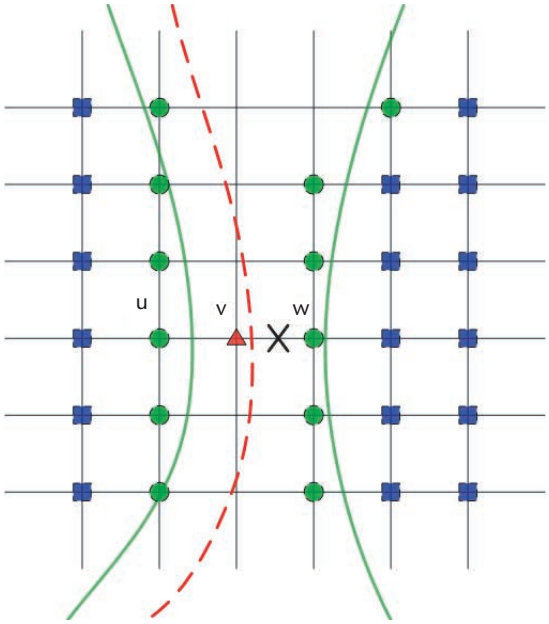
After constructing the tool scalar field s_t, designers can enforce global control of s_t in two ways:

- adjusting the coefficients of the implicit functions defined for each skeletal element, or
- manipulating or moving the skeletons.

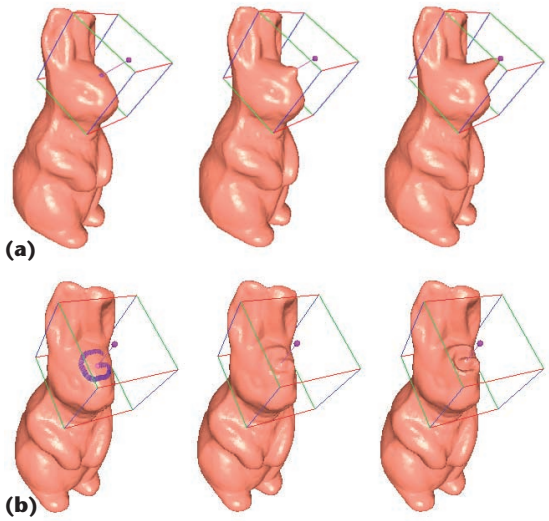
When designers modify the tool scalar field, the embedded surface scalar field and the point set surfaces are deformed according to the SFFD algorithm. Figure 8 shows how adjusting the influence radius of the underlying blob skeletons achieves free-form deformations on a rabbit model. Figure 9 (next page) shows how bending the underlying curve skeleton bends the rabbit model. Bending the underlying curve skeleton first changes the tool scalar field, and then deforms the embedded point geometry according to the SFFD algorithm. Users can taper the model by sketching source and target strokes, as Figure 10 illustrates.

Sketch-based editing tools

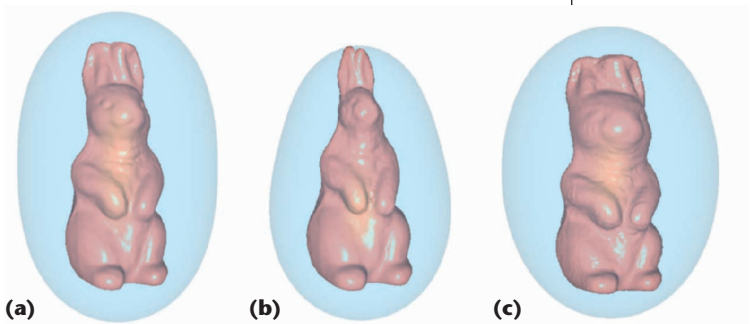
We also developed several simple sketching tools to let users edit point set surfaces using hand strokes. The system gathers strokes from the mouse as a set of curves or collection of points, and assigns Gaussian blobs evenly along the curve or at each point. The surface grows along this implicitly defined region, either outside or inside of it depending on the surface motion's direction. These tools use the well-known speed function, which is a combination of two terms[11]:



6 Collision between two parts of the front of the surface (green curves). The red dashed curve advances in the current iteration step, while the red grid changes status from layer L_{-1} to L_0. Green grids denote active grid points in L_0 from the previous step. Blue grids are inside the active surface region.



7 Force-based tools: (a) A point-based rope tool for deforming the rabbit model's surface and (b) a curve-based rope tool for sculpting the rabbit's head.

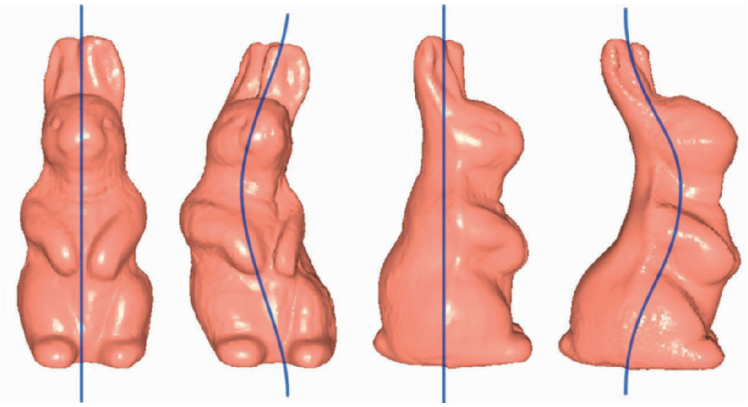


8 Shrinking and inflating a rabbit model using blob skeleton: (a) defining the blob skeleton on the original rabbit model (blue indicates the skeleton-based scalar field's isosurface); (b) shrinking the rabbit model's head; and (c) inflating the middle part of the rabbit model.

$$F = \alpha D + (1-\alpha)\nabla \cdot \frac{\nabla \phi}{|\nabla \phi|}$$

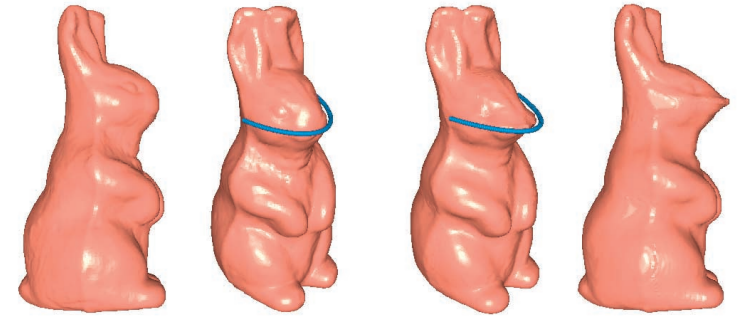
where D depends on user input strokes, forcing the surface to expand or contract toward the stroke region boundary. The term $\nabla \cdot \nabla\phi/|\nabla\phi|$ is the surface's mean curvature H, which forces the surface to remain smooth. We use the topology change handling techniques addressed earlier for collision detection and point sample deletion.

Figure 11 shows the sketch-based surface growing method. The user simply draws a sketched curve on the rabbit's back (Figure 11a). The original point set surface grows along the curve (Figure 11b), automatically resolving topology change when it detects a collision (Figure 11c). Finally, it forms a handle on the rabbit's back (Figure 11d). By inverting the growing direction of level-set surfaces, we achieve drilling operations based on user input strokes.



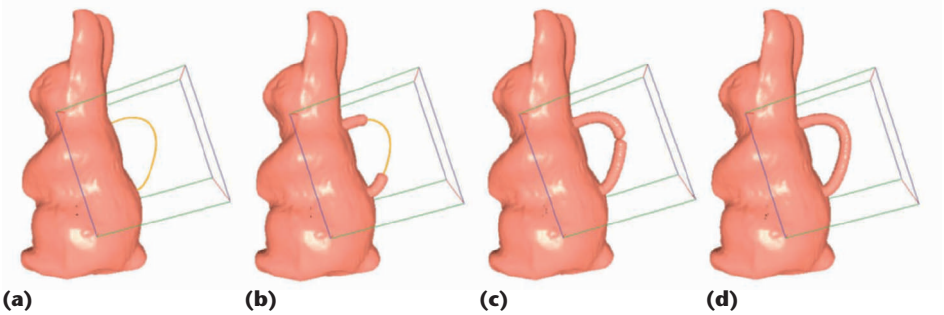
(a) (b) (c) (d)

9 Bending a rabbit model using curve skeleton: (a) original frontal view of the model, (b) frontal view of the model after bending the underlying curve skeleton, (c) original profile of the rabbit model, and (d) profile after bending the curve skeleton.



(a) (b) (c) (d)

10 Tapering the rabbit mouth using sketched strokes: (a) original profile of the rabbit model and (b) the sketched source stroke, and (c) the sketched target stroke and (d) profile after applying the tapering operation.



(a) (b) (c) (d)

11 Sketch-based surface growing: (a) the user draws a curve on the rabbit model's back, (b) the point set surface grows along the curve, (c) the surface resolves topology change on detecting a collision, and (d) the point set surface forms a handle.

Embossing and engraving tools

Attracting and repelling the surface toward and away from a set of curves or collection of points produces an embossing and engraving effect. In our system, after the user draws curves or places points near the surface, we emboss or engrave the surface with the shape of those curves or point sets. We use Museth et al.'s speed function[16] $F = D_q(d)C(\gamma)G(\gamma)$. Here D_q is a distance-based cut-off function that depends on a distance measure d to a geometric region of influence (ROI) primitive such as a superellipsoid. $C(\gamma)$ is a cut-off function that controls the contribution of $G(\gamma)$ to the speed function, whereas $G(\gamma)$ depends on geometric measures γ of the level-set surface.

For the embossing and engraving tool, we formulate the speed function as $F(\mathbf{x}, \mathbf{n}, \phi) = -\alpha D_q(d)C(-sign[\phi(\mathbf{p}_i)]\mathbf{n} \cdot \mathbf{u}_i)\,\phi(\mathbf{p}_i)$, where $\mathbf{x}$ is the current grid's location, $\mathbf{n}$ is the surface normal direction, d is a signed distance measure to an ROI primitive evaluated at $\mathbf{x}$, $\mathbf{p}_i$ is the closest point in the curve (or point set) to $\mathbf{x}$, and $\mathbf{u}_i = (\mathbf{p}_i - \mathbf{x})/|(\mathbf{p}_i - \mathbf{x})|$. Figure 12 is an example of surface embossing.

Smoothing tools

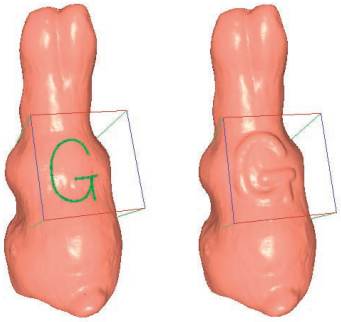
Applying motion to reduce local curvature can locally smooth the surface. We use Museth et al.'s formulation[16]:

$$F = \alpha D_q(d)C(\kappa)\kappa \tag{8}$$

where κ is either mean curvature or Gaussian curvature of the level-set surface. The point set surface models produced by the constructive solid geometry (CSG) Boolean operations can contain sharp and jagged creases at surface intersections. Figures 13a and 13b show the effects of applying the smoothing operator to the intersection region of the point set surfaces constructed from the CSG union operations. Inverting α in Equation 8 and applying an upper cut-off in $C(\kappa)$ to maintain numerical stability achieves the sharpening operator.

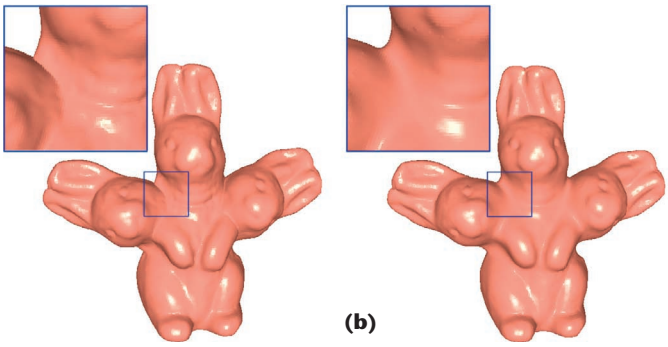
Implementation and results

We implemented the simulation and rendering parts of our system on a Microsoft Windows XP PC with dual Intel Xeon 2.4-GHz CPUs, 2-Gbyte RAM, and an Nvidia GeForce Fx 5800 Ultra GPU. We wrote the entire system in Microsoft Visual



12 Surface embossing based on a set of user-sketched curves near the surface.

14 Rabbit king, rabbit teapots, and CGA 2004 logo created using our scalar-function-driven editing framework and rendered using QSplat.



13 Curvature-based smoothing operations: (a) intersection region of the point set surface constructed from the constructive solid geometry (CSG) union operations and (b) smoothed intersection region.

C++ and built the graphics-rendering component on OpenGL. We used Sensable Technology's Phantom device for haptic input and force feedback during local surface sculpting.

We attach the haptic device to a low-end PC. A parallel technique multithreads the haptics, graphics, and sculpting processes with weak synchronization, reducing latency and maximizing throughput. This technique significantly improves performance and enables a parallel implementation of haptic sculpting given high-end multiprocessor computing resources. Having a dedicated low-end PC process haptic input and output exclusively saves this thread from competing with the computationally intensive simulation thread. The haptic thread gets enough CPU time to guarantee the desired update rate of 1,000 Hz. We implement rendering and simulation on different threads as well. Therefore, when we run the system on a dual-processor board, rendering and simulation don't interfere with each other from the CPU load's viewpoint.

We've conducted many experiments and recorded update times for force-based sculpting operations and various level-set-based editing operations. Tables 1 and 2 detail the results. We didn't include dynamic sampling (that is, up-sampling and down-sampling) time into the update times in Tables 1 and 2 because it depends on the number of points inserted or deleted at each simulation step. The haptic loop always completes within 1 millisecond on the separate low-end PC.

Using our scalar-function-driven point set surface modeling framework, we've created several interesting objects, shown in Figure 14. We generated the rabbit king's crown by dragging the rabbit's head with the force tool for a bumpy effect, and the rabbit king's bow tie using embossing operations. We created the rabbit teapots from rabbit models and the teapot spouts using curve-based bending on the rabbit models, tapering for the rabbits' mouths, sketch-based editing for the handles on the back of the rabbits, and smoothing for the intersection of the rabbits and spouts after the CSG union operation. We created the CGA 2004 logo using sketch-based editing techniques by sketching the logo-shaped curves on the rabbit point set

Table 1. Update time for force-based dynamic simulation.

No. of mass points	No. of inside points	Time (seconds)
$20 \times 20 \times 20$	6,060	0.018821
$30 \times 30 \times 30$	6,060	0.030717
$40 \times 40 \times 40$	6,060	0.050563
$20 \times 20 \times 20$	16,071	0.029026
$30 \times 30 \times 30$	16,071	0.042454
$40 \times 40 \times 40$	16,071	0.066969

Table 2. Update time for various level-set-based editing operations.

Editing tool	No. of grids	No. of points	Time (seconds)
Shrinking/inflation	128^3	67,038	4.652733
Bending	128^3	67,038	3.009914
Sketch editing	64^3	9,348	0.183589
Embossing	64^3	9,191	0.214501
Smoothing	64^3	9,826	0.535967

surface models and letting the surface grow along the user input curves.

Conclusion

Several improvements to our current work are possible in the near future. Currently the dynamic sampling and the underlying level-set approach limit the speed of our free-form deformation. Our ultimate goal is to enhance the deformation operations with haptics so users can have realistic force feedback when performing deformations on point set surfaces. ■

Acknowledgments

This research was supported in part by the US National Science Foundation through grants IIS-0082035 and IIS-0097646 and an Alfred P. Sloan Fellowship. The authors thank Christopher Carner for proofreading the draft. Special thanks to Mitsubishi Electric Research Laboratories and Liu Ren from Carnegie Mellon University for providing us the vertex/pixel shader codes of the hardware-accelerated elliptical weighted average surface splatting. The rabbit model is courtesy of Cyberware.

References

1. L. Ren, H. Pfister, and M. Zwicker, "Object Space Ewa Surface Splatting: A Hardware Accelerated Approach to High Quality Point Rendering," *Proc. Eurographics*, Blackwell, 2002, pp. 461-470.
2. C. Dachsbacher, C. Vogelgsang, and M. Stamminger, "Sequential Point Trees," *Proc. Siggraph*, ACM Press, 2003, pp. 657-662.
3. H. Pfister et al., "Surfels: Surface Elements as Rendering Primitives," *Proc. Siggraph*, ACM Press, 2000, pp. 335-342.
4. M. Pauly, M. Gross, and L. Kobbelt, "Efficient Simplification of Point-Sampled Surfaces," *Proc. IEEE Visualization*, IEEE CS Press, 2002, pp. 163-170.
5. M. Alexa et al., "Computing and Rendering Point Set Surfaces," *IEEE Trans. Visualization and Computer Graphics,* vol. 9, no. 1, Jan.-Mar. 2003, pp. 3-15.
6. J. Hua and H. Qin, "Haptics-Based Volumetric Modeling Using Dynamic Spline-Based Implicit Functions," *Proc. IEEE Symp. Volume Visualization and Graphics*, IEEE Press, 2002, pp. 55-64.
7. Y. Ohtake et al., "Multilevel Partition of Unity Implicits," *Proc. Siggraph,* ACM Press, 2003, pp. 463-470.
8. J. Hua and H. Qin, "Free-Form Deformations via Sketching and Manipulating Scalar Fields," *Proc. 8th ACM Symp. Solid Modeling and Applications*, ACM Press**,** 2003, pp. 328-333.
9. X. Guo and H. Qin, "Dynamic Sculpting and Deformation of Point Set Surfaces," *Proc. Pacific Graphics*, IEEE CS Press, 2003, pp. 123-130.
10. H. Xie et al., "Piecewise C1 Continuous Surface Reconstruction of Noisy Point Clouds via Local Implicit Quadric Regression," *Proc. 14th IEEE Visualization Conf.*, IEEE CS Press, 2003, pp. 91-98.
11. R.T. Whitaker, "Volumetric Deformable Models: Active Blobs," *Proc. Conf. Visualization in Biomedical Computing*, ACM Press, 1994, pp. 122-134.
12. R.T. Whitaker, "A Level-Set Approach to 3D Reconstruction from Range Data," *Int'l J. Computer Vision*, vol. 29, no. 3, 1998, pp. 203-231.
13. S. Bischoff and L. Kobbelt, "Sub-voxel Topology Control for Level Set Surfaces," *Proc. Eurographics*, Blackwell, 2003, pp. 273-280.
14. G. Bertrand, "Simple Points, Topological Numbers and Geodesic Neighborhoods in Cubic Grids," *Pattern Recognition Letters*, vol. 15, no. 10, 1994, pp. 1003-1011.
15. F. Dachille, H. Qin, and A. Kaufman, "A Novel Haptics-Based Interface and Sculpting System for Physics-Based Geometric Design," *Computer-Aided Design*, vol. 33, no. 5, 2001, pp. 403-420.
16. K. Museth et al., "Level-Set Surface Editing Operators," *Proc. Siggraph*, ACM Press, 2002, pp. 330-338.

Xiaohu Guo *is a PhD candidate in the Department of Computer Science at the State University of New York at Stony Brook. His research interests include geometric and physics-based modeling, computer animation and simulation, interactive 3D graphics, scientific visualization, and virtual reality. Guo has a BS in computer science from the University of Science and Technology of China. For more information see http://www.cs.sunysb.edu/~xguo.*

Jing Hua *is a PhD candidate in computer science at the State University of New York at Stony Brook, where he is also a research assistant in the Center for Visual Computing. His research interests include computer graphics, geometric and physics-based modeling, scientific visualization, human–computer interaction, and computer vision. He has a BS in electrical engineering from the Huazhong University of Science and Technology, China, and an MS in pattern recognition and artificial intelligence from the Institute of Automation, Chinese Academy of Sciences. For more information see http://www.cs.sunysb.edu/~jinghua.*

Hong Qin *is an associate professor of computer science at the State University of New York at StonyBrook. His research interests include computer graphics, geometric and physics-based modeling, computer-aided design, virtual environments, animation, simulation, and robotics. He has a PhD in computer science from the University of Toronto. Qin received an NSF CAREER Award, a Honda Initiation Grant Award, and was an Alfred P. Sloan Research Fellow. He is on the editorial board of* The Visual Computer. *For further information, please visit http://www.cs.sunysb.edu/~qin.*

Readers may contact Xiaohu Guo at the Center for Visual Computing, Dept. of Computer Science, State Univ. of New York at Stony Brook, Stony Brook, NY 11794-4400; xguo@cs.sunysb.edu.