Touch-Based Haptics for Interactive Editing on Point Set Surfaces

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

Over the past several years, point-sampled geometry has gained popularity in graphics and related visual computing areas. Point set surfaces, in particular, have enjoyed a major renaissance in both modeling and rendering. Points have clear advantages over traditional primitives such as triangle meshes—for example, points can be rendered and updated efficiently with no connectivity concerns, especially in large-scale, scanned models from laser range and image-based scanning devices. Many recent efforts have focused on direct rendering techniques and effective modeling mechanisms (see the "Related Work" sidebar) for point-sampled geometry without connectivity. However, such achievements are inadequate for interactive shape control of the point-based geometry when objects' dynamic behavior and user interactivity are of prime significance to graphical modelers and animators. Among the remaining challenges, perhaps most important for point-based modeling techniques is the ability to perform fast, interactive modeling tasks, allowing users to manipulate and sculpt point clouds intuitively and efficiently.

Unlike pure point-based modeling, implicit modeling can not only handle arbitrary topology and complicated geometry, but also affords powerful physics-based modeling.[1] Hence, integrating point-based geometry with implicit functions and introducing physics-based modeling and haptics into this under-explored field is both appealing and important.

We propose a point-based geometry representation that we initially designed for dynamic physics-based sculpting, but can easily generalize to other relevant applications such as data modeling and human–computer interaction. By extending the idea of the local reference domain in the moving least square (MLS) surface model[2] to the construction of a local and global surface distance field, we naturally incorporate Hua and Qin's dynamic implicit volumetric model[1] into our deformation of the point-based geometry, which not only facilitates topology change but also affords dynamic sculpting and deformation.

To exploit this unified scheme, we further integrate haptics-based dynamic sculpting and Boolean operations into our surface-modeling framework and use haptics painting to enhance the appearance of the point set surface. Our sculpting system offers virtual sculpting tools with various types of haptic interaction and supports real-time manipulation of the dynamic point set surface. Our long-term goal is to bring haptics, dynamics, and physics into the realm of point-based modeling and advance knowledge in this underdeveloped area.

A modeling paradigm for haptics-based editing on point set surfaces exploits implicit surfaces, physics-based modeling, point-sampled surfaces, and haptic feedback.

Dynamic point set surface

Our point-based geometry representation bears some similarity to Pauly et al.'s hybrid surface representation (see the "Related Work" sidebar on the next page), which combines explicit point samples with implicit surface functions. In our representation, we acquire the global surface distance field by blending the local trivariate distance function associated with each point sample. Our basic assumption is that the point samples always reside on the global surface distance field's zero-level set. Thus, manipulating the global scalar field, which we model as a global mass-spring system (described later), updates the point samples' positions, resulting in the deformation of the point-sampled surfaces. The difference between our system and Pauly et al.'s free-form shape-modeling system is that ours is designed for dynamic physics-based sculpting and deformation, and thus depends on real-world physical laws to govern the interaction of dynamic objects and their realistic simulation.

Local and global surface distance fields

The MLS surface model[2] stimulated our idea of the *local surface scalar field,* in which the input data is an unstructured point cloud, $P = \{\mathbf{p}_i \mid 1 \le i \le N\}$, describing an underlying manifold surface S. Each point sample

Related Work

Since Levoy and Whitted's[1] pioneering report, considerable research has been devoted to the efficient representation, modeling, processing, and rendering of point-sampled geometry.

Rusinkiewicz and Levoy[2] introduced QSplat, a technique that uses a hierarchy of spheres of different radii to render a high-resolution model. Zwicker et al.[3] introduced a surface-splatting technique that directly renders opaque and transparent surfaces from point clouds without connectivity. Later, they presented Pointshop 3D,[4] a system for interactive shape and appearance editing of 3D point-sampled geometry. Alexa et al.[5] used the moving least square (MLS) projection framework to approximate a smooth surface defined by a set of points and introduced associated techniques for resampling the surface to generate an adequate surface representation.

Most recently, Pauly et al.[6] presented a freeform shape-modeling framework for point-sampled geometry to handle both Boolean operations and freeform deformations.

As for haptics-based computing, Salisbury and his colleagues developed the Phantom haptic interface, which has resulted in many haptic-rendering algorithms. Salisbury and Tarr[7] presented research on haptic rendering of simple implicit surfaces. Kim et al.[8] presented a rather different implicit-based haptic-rendering technique.

Implicit functions are well suited for both scientific visualization and graphics modeling tasks. Igarashi and Hughes,[9] for example, describe a framework for introducing visually smooth surfaces into sketch-based freeform modeling systems. They compute a smooth interpolative surface via implicit quadratic surfaces that best fit the mesh locally in a least-squares sense. Ohtake et al.[10] present the multilevel partition of unity (MPU) implicit surface for constructing surface models from very large point sets.

Our construction method of a global surface distance field by blending local trivariate implicits is fundamentally similar to Igarashi and Hughes'[9] and Ohtake et al.'s[10] approaches. Hua and Qin developed a haptic interface that permits direct manipulation of volumetric objects (see the main article). Our current work is a direct extension of that work.

References

1. M. Levoy and T. Whitted, *The Use of Points as a Display Primitive*, tech. report 85-022, Univ. of North Carolina at Chapel Hill, 1995.
2. S. Rusinkiewicz and M. Levoy, "Qsplat: A Multiresolution Point Rendering System for Large Meshes," *Proc. 27th Ann. Conf. Computer Graphics and Interactive Techniques*," ACM Press/Addison-Wesley, 2000, pp. 343-352.
3. M. Zwicker et al., "Surface Splatting," *Proc 28th Ann. Conf. Computer Graphics and Interactive Techniques*, ACM Press, 2001, pp. 371-378.
4. M. Zwicker et al., "Pointshop 3D: An Interactive System for Point-Based Surface Editing," *Proc. 29th Ann. Conf. Computer Graphics and Interactive Techniques*, ACM Press, 2002, pp. 322-329.
5. M. Alexa et al., "Computing and Rendering Point Set Surfaces," *IEEE Trans. Visualization and Computer Graphics,* vol. 9, no. 1, 2003, pp. 3-15.
6. M. Pauly et al., "Shape Modeling with Point-Sampled Geometry," *ACM Trans. Graphics*, vol. 22, no. 3, 2003, pp. 641-650.
7. J.K. Salisbury and C. Tarr, "Haptic Rendering of Surfaces Defined by Implicit Functions," *Proc. ASME 6th Ann. Symp. Haptic Interfaces for Virtual Environment and Teleoperator Systems*, ASME, 1997, pp. 61-68.
8. L. Kim et al., "An Implicit-Based Haptic Rendering Technique," *Proc. IEEE/RSJ Int'l Conf. Robotics and Intelligent Systems,* IEEE CS Press, 2002, pp. 2943-2948.
9. T. Igarashi and J.F. Hughes, "Smooth Meshes for Sketch-Based Freeform Modeling," *Proc. 2003 Symp. Interactive 3D Graphics*, ACM Press, 2003, pp. 139-142.
10. Y. Ohtake, et al., "Multi-Level Partition of Unity Implicits," *ACM Trans. Graphics,* vol. 22, no. 3, 2003, pp. 463-470.

stores a geometric position as well as a set of other attributes, such as normal or color. Whereas a continuous MLS surface consists of points projected onto the MLS surface, our model representation defines a set of local surface scalar fields associated with each point sample.

In MLS, given a point set P, a projection operator Ψ defines the MLS surface $S_{MLS}(P)$ as the points projecting onto themselves—that is, $S_{MLS}(P) = \{\mathbf{x} \in \Re^3 \mid \Psi(P, \mathbf{x}) = \mathbf{x}\}$. A two-step procedure defines the projection $\Psi(P, \mathbf{r})$ of any point $\mathbf{r}$ near the surface. First, we compute a local reference plane, $H = \{\mathbf{x} \langle \mathbf{n}, \mathbf{x}\rangle - D = 0, \mathbf{x} \in \Re^3\}$, $\mathbf{n} \in \Re^3$, $\|\mathbf{n}\| = 1$, by locally minimizing

$$\sum_{i=1}^{N} \left(\left\langle \mathbf{n}, \mathbf{p}_i \right\rangle - D\right)^2 \theta\left(\left\|\mathbf{p}_i - \mathbf{q}\right\|\right) \qquad (1)$$

where $\mathbf{q}$ is the projection of $\mathbf{r}$ onto H; and θ is a smooth, monotone-decreasing function—for example, a Gaussian function $\theta(d) = e^{-d^2/h^2}$ with anticipated spacing h between neighboring points.

We then fit a bivariate polynomial $g(u, v)$ to the points projected onto the reference plane H using a similar weighted-least-squares optimization. To perform this process, we minimize

$$\sum_{i=1}^{N} \left(g\left(u_i, v_i\right) - h_i\right)^2 \theta\left(\left\|\mathbf{p}_i - \mathbf{q}\right\|\right)$$

where (u_i, v_i, h_i) are the coordinates of $\mathbf{p}_i$ in the local coordinate system induced by H. We can obtain the normals at the points from the reference plane, or by evaluating the gradient of the polynomial g and then using the minimal spanning-tree method to achieve a consistent normal orientation.

Further details on MLS point set surfaces are available elsewhere.[2]

In our model, instead of locally fitting a bivariate polynomial $g(u, v)$ to the height function in the reference plane H, we fit a volumetric implicit function to the local distance field in the neighborhood of the point sample $\mathbf{p}_i$.

Throughout this article, we use scalar trivariate B-spline functions as the underlying shape primitives.[3] These trivariate functions are of the form:

$$s(u,v,w)=\sum_{i=0}^{l-1}\sum_{j=0}^{m-1}\sum_{k=0}^{n-1}P_{ijk}B_{i,r}(u)C_{j,s}(v)D_{k,t}(w) \quad (2)$$

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 $t-1$, respectively. P_{ijk} are the scalar coefficients 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. We choose the local reference parameter domain size to enclose all the K-nearest neighbors of $\mathbf{p}_i$. We use the K-nearest neighbors method to accelerate the weighted-least-square fitting process and simplify the local reference domain.

In our distance field fitting process, we generate two off-surface points associated with each point sample $\mathbf{p}_i$, one outside and another inside. We then use the weighted-least-square fitting to get the volumetric implicit function, whose zero-level set fits the given point samples. We compute the scalar coefficients P_{ijk} to minimize the weighted-least-squares error:

$$\sum_{j=1}^{K}\left(s\left(u_j,v_j,w_j\right)-d_j\right)^2\theta\left(\left\|\mathbf{p}_j-\mathbf{p}_i\right\|\right)$$

Here we focus on $\mathbf{p}_i$; $\mathbf{p}_j$ is one of its K-nearest neighbors. The local parameter coordinate (u_j, v_j, w_j) of $\mathbf{p}_j$ has a distance value d_j to the surface. θ is a smooth, monotone decreasing weighting function, such as in Equation 1.

To achieve the global continuous distance field, we blend the local implicit primitive associated with each point sample using established implicit blending techniques. In fact, we can use a weighting function with finite support that blends individual implicit primitives:

$$s(x,y,z)=\frac{\sum_{i=1}^{N}s_i(x,y,z)\phi_i(x,y,z)}{\sum_{i=1}^{N}\phi_i(x,y,z)} \quad (3)$$

If (x, y, z) is inside the local region of point sample $\mathbf{p}_i$, we evaluate the distance value $s_i(x, y, z)$ using the trivariate implicit function associated with $\mathbf{p}_i$; otherwise, we simply set it to zero. We associate $\phi_i(x, y, z)$, a smooth, positive, and monotonously decreasing weighting function with local support, with $\mathbf{p}_i$ (for example, to approximate the distance field, we use the quadratic B-spline to generate weight functions ϕ_i). In our representation, support of $\phi_i(x, y, z)$ shouldn't exceed the local reference region of $\mathbf{p}_i$. We can interpolate the distance value by letting $\phi_i(x, y, z) \to \infty$, when $(x, y, z) \to \mathbf{p}_i$.

In our surface-editing system, user manipulation locally sculpts or deforms a point-sampled surface, including the points' local shapes. This can cause some of the original surface's point samples to move from their original positions. Therefore, after user manipulations, the system must update the local distance fields for those points undergoing local shape deformations. To quickly reconstruct the local distance field, we sample the trivariate function (see Equation 2) at fixed local grids $\mathbf{G} = \{(u_i, v_i, w_i) \mid i \in [0, g]\}$, where g is the number of sampling grid positions (u_i, v_i, w_i). This lets us simplify Equation 2 as the matrix form[3]:

$$\mathbf{s} = (\mathbf{B} \otimes \mathbf{C} \otimes \mathbf{D})\mathbf{p} \quad (4)$$

where ⊗ denotes the Kronecker Product, $\mathbf{s}$ is the vector of the distance values at the grid positions, and $\mathbf{p}$ is the control coefficients vector. If we modify the distance field, we can easily reconstruct the trivariate function (see Equation 4) using

$$\mathbf{p} = [(\mathbf{B} \otimes \mathbf{C} \otimes \mathbf{D})^T(\mathbf{B} \otimes \mathbf{C} \otimes \mathbf{D})]^{-1}(\mathbf{B} \otimes \mathbf{C} \otimes \mathbf{D})^T\mathbf{d} \quad (5)$$

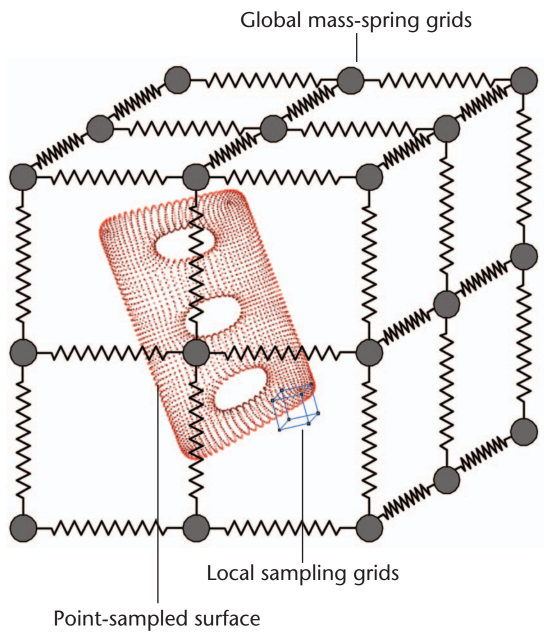
where $\mathbf{d}$ is the vector of the new distance values at the $\mathbf{G}$ grids. Because the grid positions (u_i, v_i, w_i) are fixed in the local domain during our simulation, we can improve performance by precomputing the matrix $(\mathbf{B} \otimes \mathbf{C} \otimes \mathbf{D})$.

Dynamic models

To introduce physics into our point-based local distance field, we use Hua and Qin's dynamic volumetric models approach.[1] We resample the distance field defined by Equation 3 in a region of our interest, which we denote as a global region, as compared to the point-based local region. We then discretize the global distance field into a voxel raster. Every voxel contains a scalar value, in our case a distance value, which we sample at each grid position using Equation 3. (Note that we define Equation 3 only around the surface region's thin shell.) To evaluate a location (x, y, z) outside any point sample's local definition domain, we simply approximate the implicit value as its distance to the nearest point in the point set. When users perform sculpting operations, we simulate the dynamics in this global region. After user manipulations (such as pushing or dragging the point-sampled surfaces locally), the user deforms the global distance field based on the Lagrangian dynamics (described in the following section). Assuming the points always rely on the global distance field zero-level set, we can deform the underlying point-sampled surfaces.

After deformation, the scalar value associated with each voxel might not strictly be the distance value to the isosurface. We therefore call the scalar value a *density value*.[1] However, to be consistent with our implicit construction method, we call it a distance value in the following section.

Dynamic global volumetric model. We chose the mass spring model to simulate the model's dynamics[1] because of its simplicity. We assign the discretized global distance field material quantities such as mass, damping, and a stiffness distribution, which are often considered constant. However, users can modify these material distributions interactively or directly. We model the discretized distance field as a collection of mass points connected by a network of springs across nearest-



1 Point-sampled surface (red), mass-spring grids (black) in the global sculpting domain, and sampling grids (blue) in a point's local domain.

neighbor voxels. Mass points are located at the sampled grid points over the entire global scalar field working space. The special springs for our implicit function don't change the geometric positions of the voxel mass points. Instead, they only change the magnitude of the densities located at the mass points. Essentially, this new type of spring will only attract or repel neighbors' density values. When users manipulate the implicit solids, the mass-spring system changes the density values, producing the deformable behavior of the object's shape modeled by the zero-level set of the discretized global distance field. We've therefore introduced elasticity into our volumetric implicit objects, which can thus be considered deformable models.

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

$$\mathbf{M_g}\ddot{\mathbf{d}}_\mathbf{g} + \mathbf{D_g}\dot{\mathbf{d}}_\mathbf{g} + \mathbf{K_g}\mathbf{d_g} = \mathbf{f_d}$$

where $\mathbf{M_g}$ is the mass matrix, $\mathbf{D_g}$ is the damping matrix, $\mathbf{K_g}$ is the stiffness matrix, and $\mathbf{f_d}$ is the external force vector. We use the notion $\mathbf{d_g}$ to denote the global grids' value compared with the local grid value. The connecting springs generate the internal forces, with each spring having force $f = k(I - I_0)$ according to Hooke's law. Hua And Qin's force-mapping mechanism[1] is another interesting method for computing the applied sculpting force according to the global distance field deformation:

$$f = -\int_\mathbf{C} s\,(u, v, w)\,d\mathbf{C}$$

where $\mathbf{C}$ is any force vector and $s(u, v, w)$ is the distance distribution function in 3D space.

Figure 1 shows the point-sampled surface (in red) residing in the mass-spring grids of the global distance field (in black). The blue grids are the local sampling grids $\mathbf{G}$ of one point sample, which we use to reconstruct the local trivariate function (see Equation 5) after deforming the point-sampled surface.

Simulating the behavior of dynamic implicit solids in the haptics-based environment requires less costly yet stable time-integration methods that take modest time steps. Our current implementation uses the forward Euler method to compute the modified distance values and their velocity:

$$\dot{\mathbf{d}}_{\mathbf{g}_{i+1}} = \dot{\mathbf{d}}_{\mathbf{g}_i} + \ddot{\mathbf{d}}_{\mathbf{g}_i}\Delta t$$

$$\mathbf{d}_{\mathbf{g}_{i+1}} = \mathbf{d}_{\mathbf{g}_i} + \dot{\mathbf{d}}_{\mathbf{g}_i}\Delta t$$

Although the more robust, implicit Euler solver is readily available in our system, we chose a simpler, forward method for real-time haptic interaction. Adaptively reducing the integration time step size or increasing the damping coefficients can mitigate instability due to large transient applied forces. For example, if $\ddot{\mathbf{d}}_{\mathbf{g_i}}$ is greater than a specified threshold, we use half of the previous time step as the current simulation time step.

In our implementation, users select any sculpting region of the object and perform the deformation just inside this specified global region. This region is independent of the surface definition, and limiting the deformation to the region can help achieve interactive performance.

Dynamic update of point samples. After user interaction alters the global distance field, we change the points' locations because we assume the point samples are on the implicit function's zero-level set. When we deform the distance-field space, we represent the point sample's trajectory as $\{\mathbf{x}(t)\,|\,s(\mathbf{x}(t), \mathbf{p}(t)) = 0\}$. Here, $\mathbf{x}(t)$ is the point's parameter position in its local reference domain, and $\mathbf{p}(t)$ is the control coefficients vector. The derivative of $s(\mathbf{x}(t), \mathbf{p}(t))$ with respect to time yields

$$\frac{ds(\mathbf{x}(t),\mathbf{p}(t))}{dt} = \frac{\partial s(\mathbf{x}(t),\mathbf{p}(t))}{\partial \mathbf{x}}\frac{d\mathbf{x}}{dt} + \frac{\partial s(\mathbf{x}(t),\mathbf{p}(t))}{\partial \mathbf{p}}\frac{d\mathbf{p}}{dt} = 0 \tag{6}$$

To simplify the notation, we use $\nabla_\mathbf{x} s$ to represent the gradient $\partial s(\mathbf{x}(t), \mathbf{p}(t))/\partial \mathbf{x}$, and $\nabla_\mathbf{p} s$ to represent $\partial s(\mathbf{x}(t), \mathbf{p}(t))/\partial \mathbf{p}$. We can then rewrite Equation 6 as

$$\nabla_\mathbf{x} s \cdot \dot{\mathbf{x}} + \nabla_\mathbf{p} s \cdot \dot{\mathbf{p}} = 0 \tag{7}$$

Because $\nabla_\mathbf{x} s$ and $\dot{\mathbf{x}}$ are both vectors, no unique solution for the point velocity exists. We divide $\dot{\mathbf{x}}$ into $(\mathbf{v_n}, \mathbf{v_t}, \mathbf{v_w})$, where $\mathbf{n} = -\nabla_\mathbf{x} s / \|\nabla_\mathbf{x} s\|$ represents the unit principle normal vector of the distance field isosurface, $\mathbf{t}$ represents the unit tangent vector, and $\mathbf{w}$ represents the unit binormal vector. The dot product in Equation 7 then retains only the item containing $\mathbf{v_n}$. Therefore, if we assume that the points are only moving in their normal direction, we can get their normal velocity:

$$\mathbf{v_n} = \frac{\nabla_{\mathbf{p}} s \cdot \dot{\mathbf{p}}}{\left\| \nabla_{\mathbf{x}} s \right\|} \mathbf{n}$$

We use this velocity to update the point position by advancing to the next time step through the forward Euler method.

Because we're approximating the surface at each point sample's local domain, after user manipulations we might need to reconstruct the local trivariate functions of the points undergoing deformation. We update the local grids' scalar value $\mathbf{d}$ and its velocity $\dot{\mathbf{d}}$ by interpolating the eight corner values $\mathbf{d_g}$ and $\dot{\mathbf{d}}_\mathbf{g}$ stored in each global cell using Gaussian blending. After evaluating the new value for $\mathbf{d}$, we use Equation 5 to update the control coefficients $\mathbf{p}$. We don't need to update the coefficients' local trivariate functions in each simulation step. We can update them together after completion of the haptic sculpting.

Dynamic sampling. User manipulations (for example, sculpting or deformation) on the surface change the point-sampling density. To maintain a good surface quality, we insert new sample points when surface density becomes too low or simplify the surface by eliminating points when the surface is squeezed otherwise. We use an up-sampling scheme[2] for point insertion. In each modeling step, each point checks its neighboring density by projecting its neighbor points onto its tangent plane. We compute the Voronoi diagram of these points and choose 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 distance field isosurface. Using this approach, we achieve a locally near-uniform surface density. In the meantime, we also reduce the sampling density using one of several other methods. To save computation overhead, we didn't implement the surface simplification scheme (for down-sampling) in our interactive simulation.

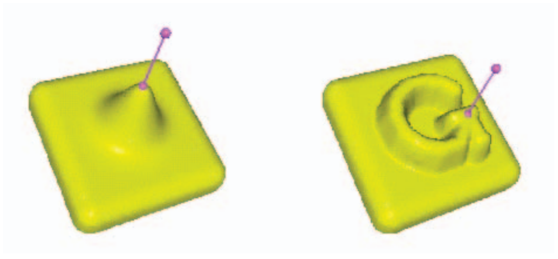
Editing toolkits

Our sculpting system has two classes of tools: haptic and geometric or topological.

Haptic tools

Haptic tools, such as rope and painting tools, not only let users perform direct sculpting and deformation with force feedback, but also enhance appearance because users can paint directly on the point set surface.

Rope tool. Our rope tool lets users select any point location inside the sculpting region and simulate the dynamics on the mass points inside the region. A user-defined function distributes the force among nearby mass points. The function can be Gaussian, constant, spherical, or any other distribution. In addition to the point-based rope tool, our system provides a curve-based rope tool with which users can apply force along the user-defined curve using any distribution mode. Figure 2a shows how we can use the point-based rope tool to drag a plate's surface; Figure 2b shows how we apply a curve-based force to sculpt a letter "G" on the plate.



2 **Rope tool functions: (a) applying a point-based rope tool deform a plate's surface; (b) using a curve-based rope tool to sculpt a letter "G" on the plate.**

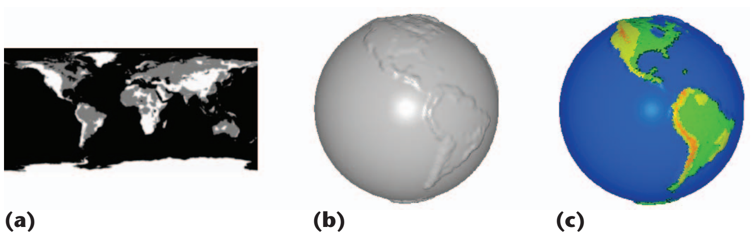
Painting tool. Given the haptic interaction, we can paint directly onto the model's surface without the cumbersome mapping schemes Zwicker et al. describe (see the "Related Work" sidebar). Our painting scheme is similar to that of Gregory et al.[4] We use the haptics cursor as a virtual paintbrush with the user-specified brush color, brush size, and brush fall-off. The brush size is proportional to the amount of force the user applies—much like a real paintbrush, which applies more paint if it's pressed against the surface harder. In our implementation, the brush colors a point within its size range according to the brush function by blending with the point's original color. We used the following brush function:

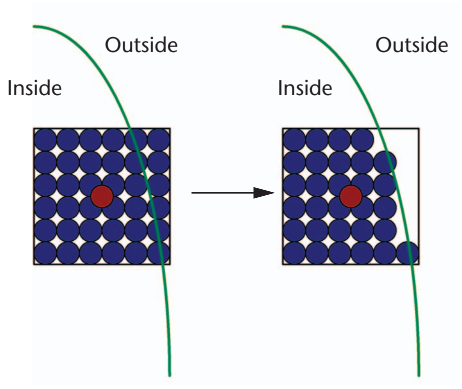
$$I = \left(\left(\frac{R_p}{R_b} \right)^f - 1 \right)^2;$$

$$C = C_b * I + C_p * (1 - I)$$

where I is paint intensity, R_p is the point's distance to the haptics cursor, R_b is brush size, f is a user-specified fall-off rate, C_b is brush color, C_p is the point's original color, and C is the resulting color that smoothly blends C_b and C_p. If the brush only partly covers a point sample (a surfel), we can up-sample more points in the partially covered point sample's vicinity[2] and then perform the covering test again on the up-sampled points. This process proceeds recursively until the partially covered point sample becomes so small that we can project it directly onto the screen as a single pixel.

While users are performing direct painting on 3D models, we let them sense the painting tool's force feedback. Because our point set surfaces are embedded in a global surface distance field, we can easily perform collision detection based on the scalar field's inside/outside property. In our current implementation, we use Kim et al.'s force-generation scheme[5] to decide both the direction and magnitude of the feedback force. We determine the direction of the haptic device's force feed-


3 Embossing and engraving tool: (a) grayscale image of the global map, which we use to define the local distance field; (b) embossed sphere; and (c) virtual earth painted using the haptic device.



4 Rendering of point samples (red) near the intersection curve (green) of two isosurfaces.

Table 1. Classification of our implicit Boolean operations.

Operation	Point Subset Q_1 with $Q_1 \subseteq P_1$	Point Subset Q_2 with $Q_2 \subseteq P_2$
$S_1 \cup S_2$	$\{p \in P_1 \mid s_2(p) < 0\}$	$\{p \in P_2 \mid s_1(p) < 0\}$
$S_1 \cap S_2$	$\{p \in P_1 \mid s_2(p) \geq 0\}$	$\{p \in P_2 \mid s_1(p) \geq 0\}$
$S_1 - S_2$	$\{p \in P_1 \mid s_2(p) < 0\}$	$\{p \in P_2 \mid s_1(p) \geq 0\}$

back by interpolating the gradient of eight cell points of the global grid around the tool tip. We base the force's magnitude on the surface's virtual contact point (VCP), which is the intersection point between the surface and the ray along the force direction. Having the VCP lets us compute the penetration force as proportional to the distance between the VCP and the tool tip. We can also simulate friction force by limiting the VCP's movement, which accounts for penetration depth.

Additional tools

Our geometric and topological tools use embossing and engraving and Boolean operations to change the underlying object's shape.

Embossing and engraving. Users can easily perform embossing or engraving operations on an object using our implicit scheme. They can use an existing image to define the distance value at the point samples. Figure 3a is an image of global map. We construct the implicit surface by fitting the distance value associated with each point sample. We then displace the points onto the isosurface of the local distance field, giving us embossing or engraving effects on the point set surface. Figures 3b and 3c show a global map embossed onto a sphere shape and painted using the haptic device. We can easily perform engraving operations on the surfaces by reversing the distance field along the opposite direction.

Boolean operations. Another major advantage of the implicit surface modeling system is that users can easily perform constructive solid geometry (CSG) Boolean operations such as union, intersection, and difference between half-space primitives because computing surface–surface intersection only requires evaluating the implicit function. For point-sampled geometry, Pauly et al. proposed using the MLS projection operator to conduct inside/outside classification; however, the projection operation can be rather time consuming. In our implicit scheme, we perform the inside/outside classification by simply evaluating the implicit function (see Equation 3).

At present, our framework doesn't let us represent objects with sharp features. Therefore, when performing Boolean operations, if we want to retain the sharp intersection of two original point set surfaces, we treat the resulting surface as two different patches of an implicit surface. Otherwise, we treat the resulting surface as a single patch for a relatively smooth intersection.

To render a sharp corner, we use the rendering scheme Alexa et al.[2] propose—that is, we sample additional points in an existing point sample's neighborhood at a resolution sufficient to conform to the screen space resolution. Using this rendering method, illustrated in Figure 4, it's simple to render two surfaces' sharp intersection. We sample additional points (in blue) near the red point, evaluating the implicit function and discarding the points outside the surface.

Using CSG Boolean operations, users can create objects of complicated geometry and arbitrary topology. More specifically, given two closed surfaces, S_1 and S_2, represented by two point sets, P_1 and P_2, we obtain a new point set P that defines the resulting surface S. Apparently, P consists of two subsets: $Q_1 \subseteq P_1$ and $Q_2 \subseteq P_2$. We can thus perform the operations by computing Q_1 and Q_2 using Table 1. In the table, s_1 and s_2 are the implicit surface functions corresponding to S_1 and S_2, and we assume that they have negative values outside the surface and positive values inside.

Figure 5 lists the results of performing different Boolean operations on two rabbit models, with sharp corners preserved.

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 Gbytes of RAM, and an Nvidia GeForce Fx 5800 Ultra GPU. We wrote the system entirely in Microsoft Visual C++ and built the graphics-rendering component on OpenGL. A SensAble Technologies Phantom, attached to a low-end PC, provided haptic input and force feedback. Figure 6 shows the haptic user interface.

System configuration

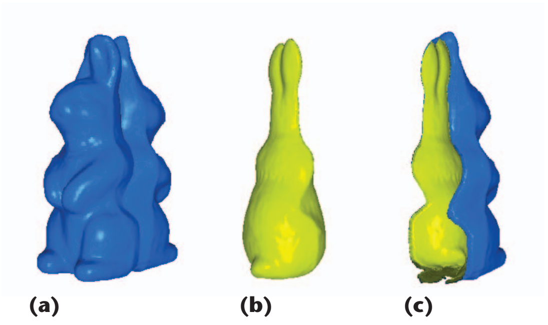
To reduce latency and maximize throughput when using haptic tools, we resort to a parallel technique that multithreads the haptics, graphics, and sculpting processes with weak synchronization (see Figure 7). This technique significantly improves performance and permits a parallel haptic sculpting implementation given high-end multiprocessor computing resources.

Because the haptics server processes haptic input and output exclusively, it need not compete with the computationally intensive simulation thread implemented in a high-end workstation.

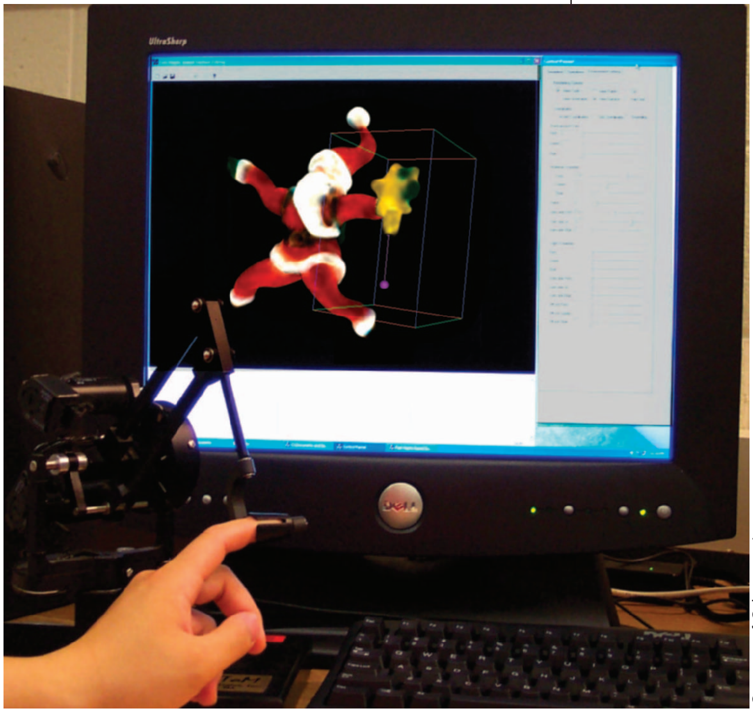
We implement the force-mapping thread in the high-end workstation as a single thread with highest priority. The force-mapping thread takes the haptics cursor position and the implicit point set surface information and computes the necessary force feedback, which it sends to the haptics server via network communication. Because our force-mapping thread has the highest priority, it can maintain the haptic refresh rate, which is no less than 1 KHz. Typically, the simulation process is much slower than 1 KHz, so our force-mapping thread must estimate the resultant force several times between updates. If we computed the force using only a piecewise constant estimation between updates, the user would feel a snapping, as if pulling the surface over ridges.

Our current implementation uses the exponential weighted average extrapolation scheme to smooth the resulting force. We formulate the underlying mathematics as $E_{i+1} = E_i \alpha + F_i(1 - \alpha)$, where E_i is the expected force value at time i, $\alpha \in (0, 1)$ is the smoothing factor, and F_i is the value of the constant force between updates. This force-extrapolation scheme smoothes the resulting force while anticipating large changes.

Because we also implement the rendering and simulation on different threads, when we run the system on a dual-processor board, the two components won't interfere with each other from a CPU load viewpoint. For simple colored point surfaces, we simply render the surfaces

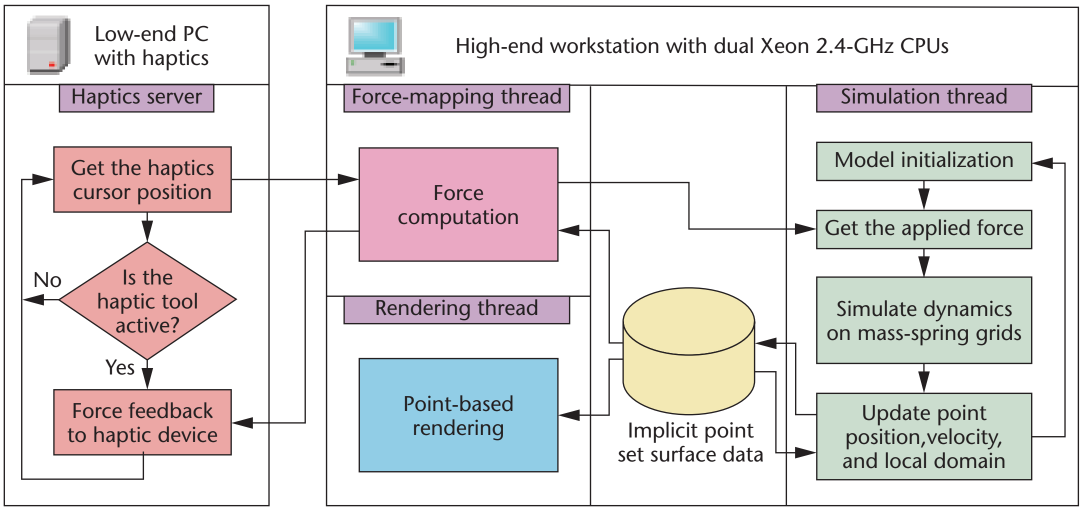


5 Results of implicit Boolean operations: (a) union (rabbit ∪ rabbit); (b) intersection (rabbit ∩ rabbit); and (c) difference (rabbit − rabbit).



Courtesy of Cyberware Inc.

6 Haptics-based user interface. The user is using a rope tool to sculpt the hand of a Santa Claus model.


Low-end PC with haptics
High-end workstation with dual Xeon 2.4-GHz CPUs
Haptics server
Force-mapping thread
Simulation thread
Get the haptics cursor position
Is the haptic tool active?
No
Yes
Force feedback to haptic device
Force computation
Rendering thread
Point-based rendering
Implicit point set surface data
Model initialization
Get the applied force
Simulate dynamics on mass-spring grids
Update point position,velocity, and local domain

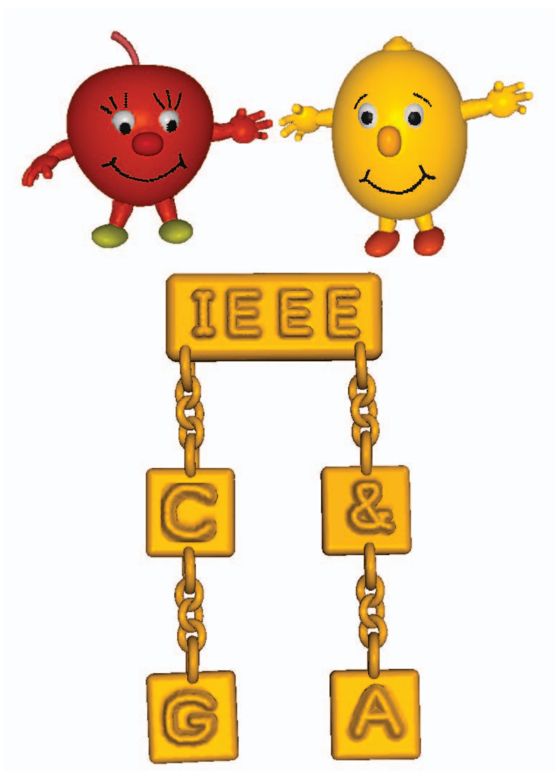
7 System configuration. We use a parallel technique to multithread the haptics, graphics, and simulation processes.

Table 2. Running time for the implicit surface construction.

Point Samples	Construction Time (seconds)
1,442	2.575340
4,500	8.177936
8,000	15.378002
16,002	30.161726
67,038	134.427521

Table 3. Updating time for the dynamic simulation.

Mass Points	Points Inside	Updating 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



8 Cartoon images created using haptics-based rope tool, Boolean operations, and haptics painting.

as point sprites, small disks, or spheres using the QSplat structure. For high-quality antialiased point rendering, we implement the rendering component using hardware-accelerated splatting.[6] Our experiments have shown that Ren et al.'s hardware-accelerated EWA splatting is fast enough for moderate-sized (fewer than 100,000 points) point set surfaces compared with the simulation speed, so the splatting technique will not become a bottleneck of our overall performance.

During the sculpting simulation, we set the number of free control coefficients and local grids to $3 \times 3 \times 3$. Results showed that this number is sufficient for each point sample in our experiments. To achieve interactive simulation, our dynamic sculpting only handles relatively small data sets (discretizing the global distance field is space consuming, and simulating the dynamics on large data sets is time consuming).

Currently, we don't consider the self-intersection problem for our interactive system because we assume users will always try to avoid self-intersection when manipulating the surfaces. We're planning to consider this issue in our future work, however.

Time performance

We've performed many experiments and have recorded the running time for constructing implicit surfaces and for updating the sculpting operations without dynamically updating the local domains. Tables 2 and 3 detail the results. We don't include the dynamic sampling time in Table 3 because it greatly depends on the number of points inserted on each simulation step. (Note that the haptics loop is always performed within 1 microsecond on the low-end haptics server.) Table 3 shows that the number of point samples inside the sculpting region is fundamental to system performance. Thus, it's important to limit the sculpting region to only the surface of interest.

Results

Using our implicit point set surface modeling framework, we've created several objects. Figure 8 shows cartoon-style characters created from some simple data sets such as spheres, ellipses, and cylinders. To create the apple and lemon characters, we used the haptics-based rope tool to pull out the arms and legs from both bodies as well as a bump on the lemon's head, and to push in a dent on the apple's head. We used Boolean operations to add the hands, feet, noses, and eyes. Finally, we used haptics painting to make the images in Figure 8 attractive. We created the *IEEE CG&A* logo using a haptics-based curve tool and Boolean operations. We began with five cubic blocks and drew the characters as curves on the surfaces. We then applied the curve-based rope tool on the curves to sculpt the logo on the plates. Experienced users completed similar examples in less than 15 minutes.

Conclusion

Several extensions to our work are possible in the near future. For example, to adapt our system to support global deformation of the point set surface, we could integrate the scalar-field-based freeform deformation which is performed by manipulating the underlying scalar field. In addition, we'll further explore the dynamic resampling scheme so we can dynamically change point set surface topology such as with sketch-based editing. ■

Acknowledgments

We thank the guest editors, Ming Lin and Kenneth Salisbury, and the anonymous reviewers for their careful and constructive reviews that have greatly helped us to improve the article's quality. We also thank Christopher Corner for proofreading the draft. Special thanks to Mitsubishi Electric Research Laboratories and Liu Ren from Carnegie Mellon University for providing the vertex/pixel shader codes of the hardware-accelerated EWA surface splatting. US National Science Foundation grants IIS-0082035 and IIS-0097646 and an Alfred P. Sloan Fellowship partially funded this research. The rabbit and Santa Claus models are courtesy of Cyberware.

References

1. J. Hua and H. Qin, "Haptics-Based Volumetric Modeling Using Dynamic Spline-Based Implicit Functions," *Proc. 2002 IEEE Symp. Volume Visualization and Graphics*, IEEE Press, 2002, pp. 55-64.
2. M. Alexa et al., "Computing and Rendering Point Set Surfaces," *IEEE Trans. Visualization and Computer Graphics*, vol. 9, no. 1, 2003, pp. 3-15.
3. J. Hua and H. Qin, "Haptic Sculpting of Volumetric Implicit Functions," *Proc. 9th Pacific Conf. Computer Graphics and Applications*, IEEE CS Press, 2001, pp. 254-264.
4. A.D. Gregory, S.A. Ehmann, and M.C. Lin, "Intouch: Interactive Multiresolution Modeling and 3D Painting with a Haptic Interface," *Proc. IEEE Virtual Reality 2000 Conf.*, IEEE CS Press, 2000, pp. 45-52.
5. L. Kim et al., "An Implicit-Based Haptic Rendering Technique," *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems*, IEEE CS Press, 2002, pp. 2943-2948.
6. 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.

Xiaohu Guo *is a PhD candidate in the Department of Computer Science at the State University of New York (SUNY) 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 an assistant professor of computer science at Wayne State University. His research interests include computer graphics, geometric and physics-based modeling, scientific visualization, human–computer interaction, virtual reality, and computer vision. Hua has a PhD in computer science from SUNY Stony Brook. For more information, see http://www.cs.wayne.edu/~jinghua.*

Hong Qin *is an associate professor of computer science at SUNY Stony Brook. His research interests include computer graphics, geometric and physics-based modeling, computer-aided design, virtual environments, animation, simulation, and robotics. Qin has a PhD in computer science from the University of Toronto. He is on the editorial board of* The Visual Computer. *For further information, see http://www.cs.sunysb.edu/~qin.*

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

For more information on this or any other computing topic, visit our Digital Library at www.computer.org/pubs/dlib.