HapticFlow: PDE-Based Mesh Editing with Haptics

Ye Duan[†], Jing Hua [‡], and Hong Qin [‡]
Department of Computer Science[†]
University of Missouri at Columbia[†]
Department of Computer Science[‡]
State University of New York at Stony Brook[‡]
email: duanye@missouri.edu, {jinghua|qin}@cs.sunysb.edu

This paper presents *HapticFlow*, a haptics-based direct mesh editing system founded upon the concept of PDE-based geometric surface flow. The proposed flow-based approach for direct geometric manipulation offers a unified design paradigm that can seamlessly integrate implicit, distance-field based shape modeling with dynamic, physics-based shape design. *HapticFlow* provides an intuitive haptic interface and allows users to directly manipulate 3D polygonal objects with ease. To demonstrate the effectiveness of our new approach, we developed a variety of haptics-based mesh editing operations such as embossing, engraving, sketching as well as force-based shape manipulation operations.

Keywords: PDE, surface flow, distance field, sketching, haptics, interaction techniques.

Introduction

Polygonal models have become prevalent in graphics, animation, and game applications. The advancement of 3D laser range scanning technology has generated a huge demand for efficient algorithms to manipulate meshes that consist of millions or even billions of vertices. Commonly used mesh editing systems often rely upon 2D mouse-based interfaces for 3D interaction. Direct operations on virtual objects with a 2D mouse are not as natural and intuitive as interaction via a 3D interface. The advent of haptic devices enables a hand-based mechanism for intuitive, manual interactions within virtual environments towards realistic tactile exploration and manipulation.

In this paper, we propose to use PDE-based surface flow as a new haptic modeling paradigm. The surface flow formulation provides a unified framework that can take advantage of both the implicit, distance-field based shape modeling and dynamic, force-based shape design, and overcome some of their limitations. For example, unlike implicit-based haptic editing system such as [1], no intermediate conversion step is necessary in our system. Besides closed shapes, our system can also work with mesh models with openings and boundaries. In contrast to the classical Lagrange-mechanics based dynamic modeling techniques such as [2], the proposed surface flow technique does not have the second-order term for elasticity behavior simulation. Hence, in principle it is very suitable for the processing of very large-scale polygonal meshes in a haptics-based environment.

The integration of the haptic interface and PDE-based surface flow technique maximizes all the potentials from each individual components. The haptic interface is very valuable and intuitive for users to interact with our models since both haptics and dynamic models depend on the same physical laws to govern the interaction of dynamic objects and their realistic simulation. To demonstrate the effectiveness of our approach, we have developed a prototype mesh editing system–*HapticFlow* that enables users to perform a variety of haptic editing toolkits, including both distance-field based operations as well as force-based operations.

0.1 Prior Work

A distance field is a scalar function that specifies the minimum distance to a shape, where the distance may be signed to distinguish between the inside and outside of the shape. Distance field has been used to generate swept volumes [3], offset surfaces [4], and to morph between surface models [5, 4]. Recently, Adaptively Sampled Distance Fields (ADF) is proposed by Frisken *et al.* [6]. ADFs consists of adaptively sampled distance values organized in a spatial hierarchy of data structures, and were later incorporated into a prototype sculpting system called “Kizamu” [7] developed by Perry and Frisken.

As for haptics-based computing, a good introduction to haptic rendering can be found in [8]. Salisbury and his colleagues developed the PHANToM haptic interface, which has resulted in many haptic rendering algorithms. Morgenbesser and Srinivasan [9] pioneered the concept of force shading. Kim *et al.* [10] presented a rather different implicit-based haptic rendering technique. Despite the widespread application of haptics in visual computing areas, haptics-based interaction was mainly applied to touching compliant objects (i.e., haptic rendering). Whereas, haptic modeling allows designers to directly manipulate objects with force feedback for the purpose of modeling or deforming objects. Foskey *et al.* [11] presented a touch-enabled 3D model design and texture painting system based on subdivision surfaces. Balakrishnan et al. [12] developed *ShapeTape*, a curve and surface manipulation technique that can sense user-steered bending and twisting motions of the rubber tape.

PDE-Based Surface Flow

The general formulation of geometric surface flow is a non-linear initial-value partial differential equation (PDE):

$$\begin{aligned} \frac{\partial \vec{S}(\vec{p})}{\partial t} &= F(t, \vec{k}, \vec{k}', \vec{f} \cdots)\vec{U}(\vec{p}, t), \\ \vec{S}(p, 0) &= \vec{S}_0(\vec{p}), \end{aligned} \tag{1}$$

where F is the velocity function, t is the time parameter, $\vec{k}$ and $\vec{k}'$ are the surface curvature and its derivative at the point $\vec{p}$, and $\vec{f}$ is the external force. $\vec{S}_0(\vec{p})$ is the initial shape of the model. $\vec{U}$ is the unit direction vector and oftentimes it is the surface normal vector.

In general, there are two approaches to numerically simulate PDEs such as Equation 1: explicit Lagrangian approach or implicit level-set approach. In this paper, we take the Lagrangian approach, i.e., the geometry and topology of the model are always explicitly represented throughout the simulation process. In particular, Equation 1 is numerically approximated using the following explicit iterative equation:

$$\vec{S}(\vec{p}, t + \Delta t) = \vec{S}(\vec{p}, t) + F(\vec{p}, t)\vec{U}(\vec{p}, t)\Delta t. \tag{2}$$

The advantage of explicit simulation of surface flow is that the user can directly interact with the polygonal models without any intermediate conversion steps, while the challenge is that we need to explicitly maintain the model regularity and to be able to handle collision detection and topology modification accurately during the deformation process. We will discuss issues related to the explicit simulating of surface flow such as mesh regularity in the next section. Our approaches of handling collision detections and topology modifications are described in later parts of the paper.

The velocity function F in Equation 1 is application-dependent, it can be either directly provided by the user, or more generally, obtained as a gradient descent flow by the Euler-Lagrange equation of some underlying energy functionals based on the calculus of variation. One of the important PDE we used for distance-field based shape manipulation (Section 0.5) is the simplified version of the weighted minimal surface flow [13] as the PDE of Equation 1 :

$$\frac{\partial \vec{S}}{\partial t} = (gv + g\|\vec{H}\|)\vec{N}, \tag{3}$$

where, $\vec{H}$ is the mean curvature of the surface and is acting as a smoothing constraint. v is the constant velocity, which will enable the convex initial shape to capture non-convex, arbitrarily complicated shapes. $\vec{N}$ is the unit normal of the surface. g is the non-increasing, non-negative weight function that will stop the deformation of the model when it reaches the shape boundary. We define the stopping function g as the commonly used 3D edge detector:

$$g(\vec{S}) = \frac{1}{1 + \|\nabla(I(\vec{S}))\|^2}, \tag{4}$$

where, I is the distance field function and ∇ is the gradient function. The mean curvature vector $\vec{H}$ used in Equation 3 is calculated by the discrete curvature estimator proposed by Desbrun *et al.* [14]:

Surface Flow Simulation

0.2 Model Regularity

To ensure the numerical simulation of the surface flow to proceed smoothly, we must maintain the regularity of the model such that the model has a good node distribution, a proper node density, and a good aspect ratio of the triangles. This is achieved by the incorporation of mesh optimization and Laplacian smoothing. Note that, these techniques are applied only to regions of the polygonal model that are deforming at the current time step.

0.2.1 Mesh optimization

There are three commonly used mesh optimization operations [15]: edge split, edge collapse, and edge swap. Edge split and edge collapse are used to keep an appropriate node density. An edge split is triggered if the edge length is bigger than the maximum edge length threshold. Similarly, an edge will be collapsed if its length is smaller than the minimum edge length threshold. Edge swapping is used to ensure a good aspect ratio of the triangles. This can be achieved by forcing the average valence to be as close to 6 as possible [16]. An edge is swapped if and only if the quantity $\sum_{\vec{p}\in\Delta}(valence(\vec{p})-6)^2$ is minimized after the swapping, where Δ represents the four vertices in the two adjacent triangles of the current edge.

0.2.2 Laplacian smoothing

Laplacian operator, in its simplest form, moves repeatedly each mesh vertex by a displacement which equals to a positive scale factor times the average of the neighboring vertices. Consider a mesh vertex $\vec{p}$ and its neighbors $\vec{Q}_1,\cdots,\vec{Q}_n$, the Laplacian operator $\vec{U}$ is:

$$\vec{U}(\vec{p})=\frac{1}{n}\sum_{i=1}^{n}(\vec{Q}_i-\vec{p}). \tag{5}$$

The tangential Laplacian operator is used to maintain a good node distribution and is defined as:

$$\vec{T}(\vec{p})=C[\vec{U}-(\vec{U}\cdot\vec{n})\vec{n}], \tag{6}$$

where $\vec{n}$ is the normal vector at vertex $\vec{p}$ and C is a positive constant.

0.3 Step Size Estimation

When advancing the model using surface flow, we must enforce a constraint on Δt. In particular, the time step must satisfy the *CFL condition* (a.k.a., Courant-Friedrichs-Lewy stability criterion), i.e., the velocity of change must be strictly restrained by the minimum detail in the system. In our system, this condition is

$$\Delta t\leq\frac{m_e}{M_F}, \tag{7}$$

where m_e is the minimum edge length and M_f is the maximum magnitude of the velocity F. Before each deformation step, we will calculate the velocity F at each vertex point and determine the maximum magnitude of the velocity M_F. Thus, a proper time step can be obtained from (7).

Surface Flow Based Haptic Editing

Our system supports two types of haptic editing operations. The user can either directly manipulate the mesh model by exerting forces through the haptic device, or alternatively, define a local distance field using the haptic device, and evolve the surface according to the user-defined distance field.

In general, there are four main steps during a typical interactive design process:

1. The user specifies a region of the model to be deformed by using the haptics cursor.
2. The user exerts forces or defines a local distance field through the haptic device.
3. The corresponding region of the model deforms according to the PDE-based surface flow (Equation 1).
4. During the deformation, the system automatically detects the potential collisions between different parts of the model and change the topology accordingly.

0.4 Force-Based Shape Manipulation

In our system, the user can directly manipulate the polygonal objects by applying forces through haptic device. We employ the Hooke's law to generate the force, $\vec{f} = k(\vec{p}_{curs} - \vec{p}_{surf})$, where $\vec{p}_{surf}$ is the surface point on the mesh which the user initially picks up, $\vec{p}_{curs}$ is the haptics cursor that the user controls to deform the mesh, and k is a positive spring constant. Usually the longer force vector, $(\vec{p}_{surf} - \vec{p}_{cursor})$, the user's action introduces, the larger external force will be generated. Simultaneously, an equal but opposite force, $-\vec{f}$, will feed back to the user through the haptic device to get the haptic feeling.

To control the region of influence of the applied force, we use a finite supported Gaussian function, $S(d)$, to distribute the force into a set of mesh points in the nearby region. The force at a neighboring surface point $\vec{q}$ is calculated as: $\vec{f_{\vec{q}}} = S(||\vec{q} - \vec{p}_{surf}||)\vec{f}$.

The computed force $\vec{f_{\vec{q}}}$ is then plugged into the right hand side of Equation 1 to guide the deformation of the model. Here, the speed function F of Equation 1 becomes $S(||\vec{q} - \vec{p}_{surf}||)||\vec{f}||$, the unit direction vector is $\frac{\vec{f}}{||\vec{f}||}$.

Figure 5(b) shows an example created using our force-based manipulation tools. Here, the nose is created by dragging the mesh outwards, the mouth is created by pushing the mesh inwards. The two eyes are created by haptic drilling, i.e. keep pushing the mesh inwards until it collides with the other part of the mesh, and a hole is created by topology merging. The stones in Figure 3 are also created using force-based manipulation, there the region of influence is much larger, hence a more global deformation effect is achieved.

0.5 Distance-Field Based Shape Manipulation

Our system also supports distance-field based mesh editing. Using the haptic device, the user interactively draws some 2D or 3D strokes, either directly on the mesh or stem away from the mesh. Strokes are then densely sampled by the system as a combination of Gaussian blobs that are assigned evenly at each point. Simultaneously, the affected regions of the underlying mesh model will automatically deform according to the corresponding distance functions generated by the strokes.

During the editing process, the user may feel force feedback from the haptic device generated by the following forces: compressive force and friction force. The compressive force will prevent the user from breaking through the surface, while the friction force will give the user a realistic feeling of the bumpiness of the surface. The compressive force can also be used to define the strength of the assigned blobs. Figure 1 shows the haptic user interface.

0.5.1 Compressive force

The compressive force at $\vec{q}$ is along the surface normal $\vec{n}_{\vec{p}}$ at the closest surface point $\vec{p}$,

$$\vec{f_{\vec{n}}} = \begin{cases} \lambda(\vec{q} - \vec{p}) \cdot \vec{n}_{\vec{p}}, & \text{if}(\vec{q} - \vec{p}) \cdot \vec{n}_{\vec{p}} < 0, \\ 0, & \text{otherwise}, \end{cases} \tag{8}$$

where $\vec{q}$ denotes the position of the haptics cursor, and $\vec{p}$ denotes the closest point $\vec{p}$ on the polygonal surface to $\vec{q}$. From the above equation, we can see that if the haptics cursor is running out of the polygonal object, the compressive force is equal to zero. An equal but opposite force, $-\vec{f_{\vec{n}}}$ will be fed back to the user through the haptic device to let the user feel the resistance when the haptics cursor is trying to running inside the polygonal object.

Inspired by the Chinese brush painting and calligraphy, in our system, we associate the magnitude of the compressive force $||\vec{f_{\vec{n}}}||$ to the strength of the assigned blobs. The bigger compressive force the user exerts on the surface, the larger strength of a blob will be introduced. This allows the user to interactively and locally control the size of to-be-embossed shape. The variation of the width of the stroke is created by the variation of the compressive force the user applied through the haptic device.

0.5.2 Friction force

Using the compressive force, we can easily define the friction force as follows,

$$\vec{f_s} = -\mu||\vec{f_{\vec{n}}}||\frac{\vec{v}_{\vec{p}}}{||\vec{v}_{\vec{p}}||}, \tag{9}$$

where $\vec{f_{\vec{n}}}$ is the compressive force when the haptics cursor is at position $\vec{s}$, $\vec{v}_{\vec{p}}$ is the projection of the velocity of the haptics cursor at $\vec{s}$ onto the tangent plane of the closest surface point $\vec{p}$. The friction force is a passive force purely for haptic rendering purpose. It only feed back to the user and is not participating in the surface evolution process.

Figure 2 shows an example. Here, a $ sign is embossed on the bunny-bank model by directly sketching two strokes on the mesh using the haptic device. The Gaussian blobs are assigned evenly at each sampling point. During the editing process, the user will feel force feedback from the haptic device generated by the aforementioned friction force and compressive force. Engraving effect can also be achieved by simply set the deformation direction as the opposite of the normal vector of each vertex in the affected region. The coin slot is created by placing an implicit cube on the back of the bunny and removing all the vertices that are inside the implicit cube.

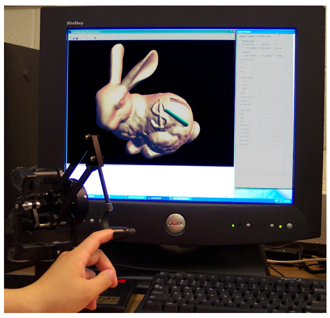
Figure 1: The haptic user interface.

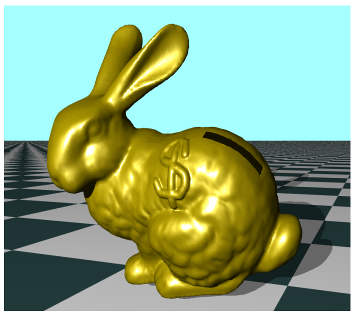
Figure 2: The bunny-bank.

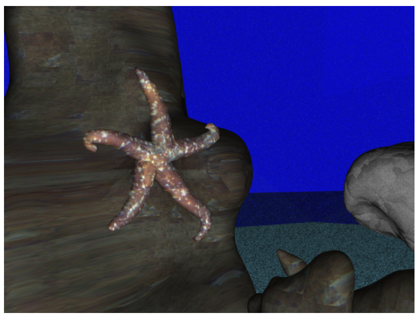
Figure 3: The starfish in the sea.

The user can also define distance field through free-hand sketching, i.e. drawing strokes away from existing mesh. Figure 4 shows an example. The input is a polygonal model of a goblet (Figure 4(a)). Now, the user wants to add two handles on the goblet. So he/she simply draws two curved strokes at both sides of the goblet. A distance field is then created by summing up the Gaussian functions that are assigned at each point (shown as red dots and green dots in Figure 4(a)) on the two strokes. The original goblet mesh will grow simultaneously along the track of the strokes to form a champion trophy as show in Figure 4(b).

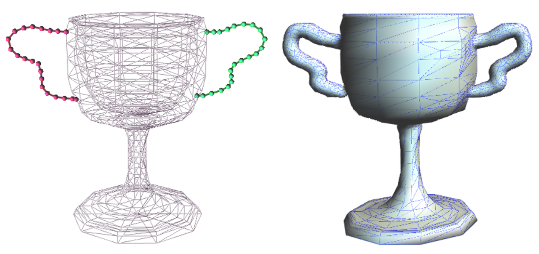
(a) (b)

Figure 4: The champion cup.

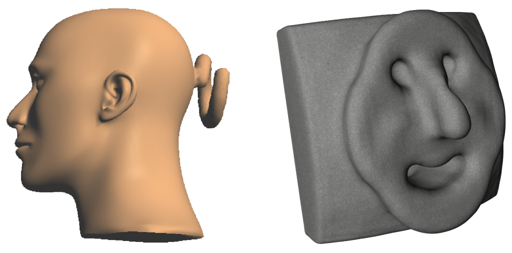
(a) (b)

Figure 5: (a) The spiral tail of the mannequin. (b) The happy monster head.

The starfish in Figure 3 and the spiral tail on the back of the mannequin in Figure 5(a) are both created in this fashion as well. For example, in Figure 3, starting from a simple sphere-like polygonal model in the center, the user iteratively draws five curved strokes away from the original mesh. The original mesh will then grow along the newly defined distance functions and create the five legs of the starfish. Note that, since the strokes are created in the “air”, there are no force feedback. The haptic device is used as a 3D pointing device only. The user can either assign a constant density value for the Gaussian blobs (Figure 5(a)), or linearly interpolate the density between a start value and a end value (Figure 3).

Particle-Based Collision Detection

One key challenge in simulating continuous surface evolution of an explicit deformable model is in performing collision detection, so that surface interpenetrations can be detected and handled properly. There has been considerable research on the problem of collision detection; in general, existing methods are either object-oriented bounding volume method or domain-oriented spatial decomposition method. The idea behind these approaches is to approximate the objects (with bounding volumes) or to decompose the space they occupy (using decomposition), to reduce the number of pairs of objects or primitives that need to be checked for contact. We propose a hybrid approach that can detect potential collisions between different parts of the surface both accurately and efficiently by combining the advantages of both the spatial decomposition method and the bounding volume method. A spatial decomposition method (a uniform occupancy grid) is used for fast collision-rejection between vertices that are located at non-neighboring grid cells. A bounding volume method (bounding spheres) is used for detecting potential collisions between vertices that are located within the same and neighboring grid cells.

We consider the object as a particle system (connected by edges) that is bounded by partially overlapping spheres of radius r centered at each particles (i.e., vertices of the object). Potential collisions between different regions of the object are then detected by potential collisions between particles. Since our model always maintains explicit maximum/minimum thresholds for the edge length, the radius r can be pre-calculated. A potential collision is detected if the distance between any two non-adjacent vertices is smaller than r. To further speed up the performance, a uniform occupancy grid is superimposed on the domain space for fast collision-rejection. Each vertex of the object will belong to a grid cell, and each grid cell will store the index/pointer of the vertices that belong to the current grid cell. The size of the grid cell is decided by the radius r of the bounding sphere so that collisions can only occur between vertices in the same or neighboring cells. At the beginning of each deformation step, the occupancy grid need to update its vertices information. This can be done locally since only a few vertices will move at each deformation step, and it usually takes constant time, or at most O(n). Figure 6 shows a 2D illustration of the collision detection scheme. Here, the two moving curves are bounded by partially overlapping circles of radius r (shown in dark circles in Figure 6 (a)) centered at each particles. Several time steps later, the distance $||P - Q||$ between particle P and Q becomes smaller than r (Figure 6 (b)) and a collision is detected. These two vertices (P and Q) will be deactivated and sent to the following topology modification step.

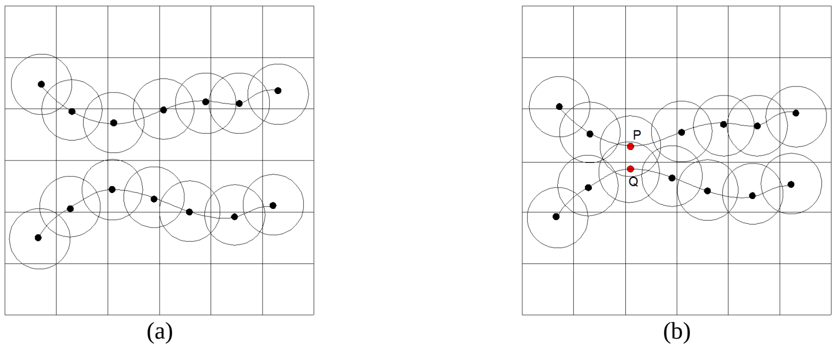

Figure 6: Particle-based collision detection.

Topology Modification

There are two types of topology modifications: topology merging (i.e. axial melting) and topology splitting (i.e. axial constriction). In our current system, only topology merging is implemented and is conducted in a sequential fashion, i.e. at most one topology merge operation can occur at any time. Moreover, to ensure the correctness of the algorithm, a topology merge operation can occur only after all the vertices of the model become deactivated (i.e. not move anymore). There are three steps in the topology merging operation:

1. Find a pair of merging vertices and align their one-neighborhoods to face each other.
2. Put the two one-neighborhoods into correspondence.
3. Reconnect the two one-neighborhoods.

0.5.3 Find a pair of center vertices and align their one-neighborhoods

The first step of the algorithm is to pick the best pair of merging vertices to serve as the center vertices. Specifically, we will calculate the inner products of the normal vectors of all pairs of merging vertices and chose the pair with the smallest inner product. If the angular deviation of the normal vectors of this pair of vertices is smaller than a certain threshold (e.g. 30 degrees), they will be picked as the two center vertices, otherwise, no topology merge operation is allowed at the current time step. Next, all the one-neighborhood vertices of these two center vertices will be projected onto the plane (passing the center vertex) that is perpendicular to the vector connecting these two center vertices. This way, the two one-neighborhoods will face exactly towards each other (Fig. 7(a)-(b)).

0.5.4 Put the two one-neighborhoods into correspondence

After the two one-neighborhoods are aligned to face each other, they will be put into correspondence by the following procedure: Iteratively refine the one-neighborhood who has fewer vertices by splitting its longest edge until both of the two one-neighborhoods have the same number of vertices, then choose an alignment that minimizes the sum of squared distances between corresponding vertices of the two one-neighborhoods. For example, in Fig. 7(b), originally the one-neighborhood of vertex A has five nodes: $\{A_1, A_2, A_3, A_4, A_5\}$, the one-neighborhood of vertex B has six nodes: $\{B_1, B_2, B_3, B_4, B_5, B_6\}$. To make these two one-neighborhoods have the same number of nodes, we first find the longest edge of the one-neighborhood of vertex A, which is the edge between nodes A_1 and A_2, and then split this edge into two edges and insert a new node in between. Finally, we put these two sets of vertices into correspondence by finding the alignment that minimizes the sum of squared distances between nodes. In Fig. 7(c), vertices $\{A_1, A_2, \cdots, A_6\}$ are corresponding to $\{B_1, B_2, \cdots, B_6\}$, respectively.

0.5.5 Reconnect the two one-neighborhoods

After the two sets of one-neighborhood vertices are put into correspondence, each vertex is connected with its corresponding vertex in the opposite one-neighborhood. The two center vertices and all its incident edges are removed (Fig. 7(d)). The newly created quadrilaterals are further triangulated by splitting each quadrilateral into two triangles along one of its diagonals (Fig. 7(e)). The aforementioned model-relaxation operations (Section 0.2) can quickly smooth out any artifacts that may result from the matching procedure once the topology merging operation has been completed.

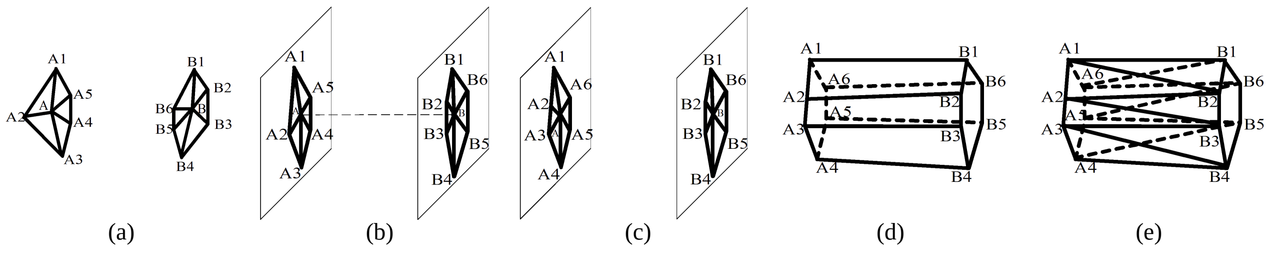

Figure 7: Topology merge.

System Implementation

Haptics-based applications demand high update rates, therefore it is both desirable and necessary to employ multiprocessor computers to accelerate computation. We develop multi-threaded implementation and parallel algorithms in order to take advantage of parallel computational architecture for performance improvement. The haptics, graphics, and simulation computations are each assigned one thread. A SensAble Technologies's PHANToM is employed as a haptic device for haptic input and force feedback. The haptic device is attached to a low-end PC. Another dual-processor XEON PC with NVIDIA's GeForce4 graphics card is used for simulation and display of polygonal objects. Figure 1 shows the haptic user interface. The surface evolution simulation and haptics computation are weakly coupled since the haptics computation is much faster than the surface evolution simulation and the force update rate has to run at above 1kHz. The weak coupling is implemented through using the same object representation for both computations. Therefore, the visual feedback and haptic feedback are consistent with each other.

Conclusion

We have presented *HapticFlow*, a haptics-based interactive mesh editing system founded upon PDE-based geometric surface flow. *HapticFlow* maximizes the potentials offered by both the surface flow and haptic interactions. It provides an intuitive haptic interface and allows users to directly manipulate 3D polygonal objects with intuitive force feedback.

During this research work, we have also observed that the flow-based approach has some very appealing potentials on accomplishing parallel design tasks that are simultaneously performed by several designers. So we plan to further extend *HapticFlow* into a network-based collaborative design framework. Through various experiments, we found that naive users oftentimes have difficulties to determine the depth information of the haptics cursor through the 2D monitor screen. An implementation of stereoscopic visual feedback on a semi-immersive VR environment such as workbench would definitely help those users to gain a much better understanding of the 3D shape geometry and perform the direct geometric deformation through user immersion.

References

[1] L. Kim, G. S. Sukhatme, and M. Desbrun. Haptic editing for decoration and material properties. In *Proceedings of the ASME 11th Annual Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 2003.

[2] Chandomay Mandal, Hong Qin, and Baba C. Vemuri. Dynamic modeling of butterfly subdivision surfaces. *IEEE TVCG*, 6(3):265–287, 2000.

[3] W. Schroeder, W. Lorensen, and S. Linthicum. Implicit modeling of swept surfaces and volumes. In *IEEE Visualization '94 Proceedings*, pages 40–45, 1994.

[4] B. Payne and A. Toga. Distance field manipulation of surface models. *IEEE Computer Graphics and Applications*, pages 65–71, 1992.

[5] D. Coher-Or, D. Levin, and A. Solomovici. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 1997.

[6] S. Frisken, R. Perry, A. Rockwood, and T. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *SIGGRAPH '00 Proceedings*, pages 249–254, 2000.

[7] R. Perry and S. Frisken. Kizamu: A system for sculpting digital characters. In *SIGGRAPH '01 Proceedings*, pages 47–56, 2001.

[8] K. Salisbury, D. Brocki, T. Massiet, N. Swarupf, and C. Zillest. Haptic rendering: programming touch with virtual objects. In *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, pages 123–130, 1995.

[9] H. B. Morgenbesser and M. A. Srinivasan. Force shading for haptic perception. In *Proceedings of the 1996 ASME International Mechanical Engineering Congress and Exposition, Dynamic Systems and Control Division*, pages 407–412, 1996.

[10] L. Kim, A. Kyrikou, G. S. Sukhatme, and M. Desbrun. An implicit-based haptic rendering technique. In *Proceeedings of the IEEE/RSJ International Conference on Intelligent Robots*, 2002.

[11] Mark Foskey, Miguel A. Otaduy, and Ming C. Lin. Artnova: Touch-enabled 3d model design. In *Proceedings of IEEE Virtual Reality*, pages 119–126, 2002.

[12] R. Balakrishnan, G. Fitzmaurice, G. Kurtenbach, and K. Singh. Exploring interactive curve and surface manipulation using a bend and twist sensitive input strip. In *Proceedings of the 1999 ACM Symposium on Interactive 3D Graphics*, pages 111–118, 1999.

[13] V. Caselles, R. Kimmel, G. Sapiro, and C. Sbert. Minimal surfaces based object segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19:394–398, 1997.

[14] M. Desbrun, M. Meyer, P. schroder, and A. H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH '99 Proceedings*, pages 317–324, 1999.

[15] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *SIGGRAPH '93 Proceedings*, pages 19–26, 1993.

[16] L. Kobbelt, T. Bareuther, and H.-P. Seidel. Multiresolution shape deformations for meshes with dynamic vertex connectivity. In *Eurographics '00 Proceedings*, pages 249–260, 2000.