Design and Manipulation of Polygonal Models in a Haptic, Stereoscopic Virtual Environment

Jing Hua
Wayne State University
jinghua@cs.wayne.edu

Ye Duan
University of Missouri at Columbia
duanye@missouri.edu

Hong Qin
Stony Brook University
qin@cs.sunysb.edu

Abstract

This paper presents a flexible, scalable framework for interactive hands-on shape design in a haptic, stereoscopic virtual environment. The framework is founded upon the concept of PDE-based geometric surface flow. Given an input polygonal mesh, a user can interactively define implicit functions around regions of interest of the mesh model, and the locally or globally affected regions of the model will automatically deform according to the underlying partial differential equations and reconstruct the implicitly defined shape. During the model deformation process, the model can always maintain its regularity and can properly modify its topology when collisions between different parts of the model occur. With augmented haptics functionality and stereoscopic display, our system provides a more intuitive interface, which allows users to directly manipulate 3D polygonal objects with hands.

1. Introduction

Modeling in virtual reality environments has been quickly emerging as an indispensable tool for solving a wide variety of design problems, such as fast virtual prototyping, virtual assembly and disassembly. Direct operations on virtual objects with a 2D mouse are not as natural and intuitive as interaction via a hand-based mechanism. The advent of haptic devices enables a hand-based mechanism for intuitive, manual interactions within virtual environments towards realistic tactile exploration and manipulation. Even though there is much research work on this topic [6][16][9], they primarily focused on integrating haptics technologies with current design systems and methodologies. Less effort has been spent on finding a good computational model to facilitate haptics-based design. As for geometric modeling, polygonal models have recently become prevalent in graphics, animation, and game applications. In general, direct mesh-based shape design can be roughly classified into either static, geometric techniques, or dynamic, physics-based techniques. Essentially, static, geometric algorithms often require a lot of user interventions that are tedious and laborious, while physics-based algorithms are more intuitive for the user to manipulate. Nonetheless, conventional physical simulation (based on Lagrangian mechanics) is relatively slow and does not scale well to large-scale models.

In this paper, we propose to facilitate hands-on 3D shape design in a haptic, stereoscopic environment based on PDE surface flow. Users can manipulate a model looking at the stereoscopic image of the model and feeling its haptic sensation in the environment. In essence, PDE surface flow as a new powerful design technique can lead to a general, physically intuitive framework. The shape deformation behaviors are controlled by general PDEs. Users can directly manipulate the models without many low-level, manual operations. Another appealing property of PDE surface flow is its efficiency (i.e., all the computations can be conducted locally). In contrast to the Lagrangian mechanics, the proposed surface flow technique does not have the second-order term for elasticity behavior simulation, which is demonstrated not necessary for interactive mesh-based shape design. Hence, in principle it is very suitable for the processing of very large-scale polygonal meshes in a haptic, stereoscopic environment. Furthermore, the surface flow formulation provides a unified framework that can take advantage of both the implicit function based shape modeling and dynamic, force-based shape design. The haptic force can be easily plugged into the evolution equation of the surface flow to guide its deformation. The surface flow also has self-adaptive improvement capability, which makes the model capture users' haptics-based deformation accurately and maintain the model quality on the fly simultaneously.

We have developed a haptics and stereoscopy based hands-on shape design system, which tightly couples the principle of haptic modeling with the concept of PDE-based geometric surface flow and permits users to directly work on meshes with hands. Force feedback provides additional

sensory cues to designers. However, according to our experiments most users oftentimes have difficulties to determine the depth information of the haptics cursor through the 2D monitor screen. An implementation of stereoscopic visual feedback in an immersive VR environment 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. The use of haptics in a stereoscopic environment promises to increase the bandwidth of information exchange between designers and the virtual world. Our prototype system provides a suite of intuitive, easy-to-use toolkits that enables users to perform a variety of haptics-based mesh editing operations with stereoscopic visual feedback. This tactile exploration can afford designers to gain a richer understanding on the 3D nature of virtual solids.

2. Prior Work

Extensive literature exists in interactive mesh generation. For example, *Skin* [10] presented a particle-based surface representation with which a user can interactively sculpt free-form surfaces. It resembled blobby modeling in the constructive approach. Zeleznik, Herndon, and Hughes [20] showed how a gesture-based modeler could be used to simplify conventional CSG-like shape creation and how sketch-based methods can facilitate the rapid creation of approximated shapes. Teddy [8] further extended this to more general, free-form models, receiving much of its power from its "inflation" operation and from an elegant collection of gestures for attaching additional parts to a shape, cutting a shape, deforming it, etc. However, these approaches can only provide very rough shapes. Our approach allows users to take advantage of current existing mesh models and create new models via the available, easy-to-use haptic toolkits in our system.

As for haptics-based computing, a good introduction to haptic rendering can be found in [16]. Salisbury and his colleagues developed the PHANToM haptic interface, which has resulted in many haptic rendering algorithms. Morgenbesser and Srinivasan [11] pioneered the concept of force shading. Salisbury and Tarr [15] presented the research work for haptic rendering of simple implicit surfaces. Kim *et al.* [9] presented a rather different implicit-based haptic rendering technique. Avila and Sobierajski [1] used the PHANToM in a haptic scientific visualization process. Thompson *et al.* [19] derived efficient intersection techniques that permit direct haptic rendering of NURBS surfaces.

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. [6] presented a touch-enabled 3D model design and texture painting system based on subdivision surfaces. Hua and Qin [7] developed a haptic interface that permits direct manipulation of volumetric objects. Balakrishnan et al. [2] developed *ShapeTape*, a curve and surface manipulation technique that can sense user-steered bending and twisting motions of the rubber tape. The *FreeForm* modeling system presented by SensAble Technologies enables users to easily create products with touch. However, this is purely a haptic enhancement for traditional system. The haptic feedback is not based on the real dynamics of modeled objects. Duan *et al.* [5] presented a haptic sculpting system based on the PDE-based surface flow. Our paper is based on the previous work and further extends the framework, enhancing it with new stable solver and further developing a new VR-based interface.

Different from the existing published results, our current work is directly based on mesh representations that are extremely popular. The specific goal of our research is to integrate the principle of haptic modeling with the concept of PDE-based geometric surface flow and provides users an intuitive system for directly designing polygonal objects in virtual environments.

3. Overview

3.1. User Interface

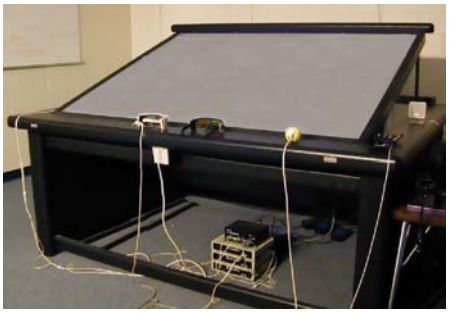


Figure 1. A haptic, stereoscopic working environment.

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 a NVIDIA's GeForce4 graphics card is used for simulation. An immersive workbench from Fakespace Inc. is used for stereoscopic display. Figure 1 shows the haptic user interface. The illustration of hardware configuration is shown in Figure 2.

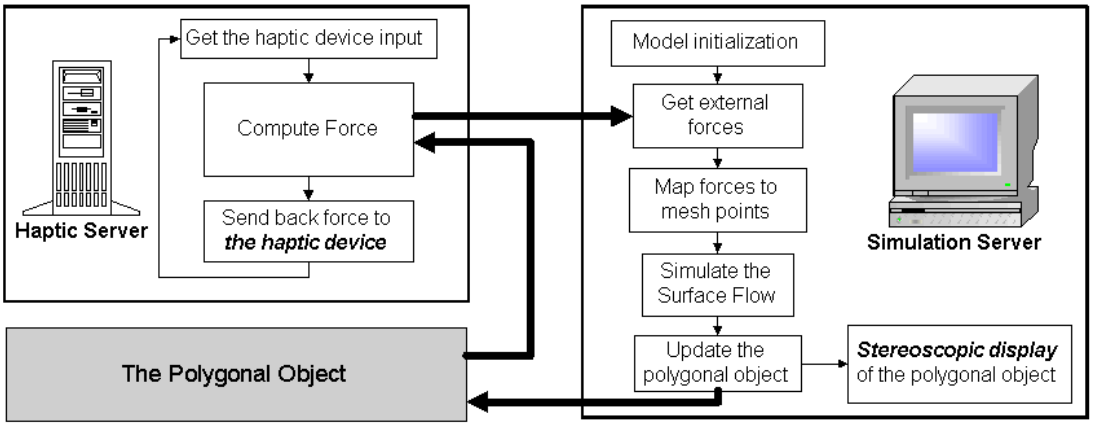


Figure 2. The system configuration.

3.2. System Overview

Our prototype system allows the user to directly edit the polygonal objects by sculpting and deforming with a haptic interface. The user sees the model being edited, the tool being used, and a menu that can be operated using either the haptic tool or a mouse with stereoscopic display. Each type of model manipulation is indicated by a different tool. When a sculpting tool is used to interact the object, either a force field or a scalar field will be generated accordingly, which will evolve the polygonal surface to achieve appropriate deformations according to the simulation of PDE-based geometric surface flow. In case of using force-based tools, the computed force will feedback to users simultaneously to obtain haptic feeling, which allows users to reach toward an object, feel the physical presence of its shape, and sculpt it with force feedback. The feedback forces are computed directly based on the object representation and the user's actions. Figure 2 shows the flow of multithreads for surface evolution simulation and haptics computation, respectively, where thick arrows represent data flow. The surface evolution simulation and haptics computation are weakly synchronized 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 synchronization is implemented through using the same object representation.

4. PDE-Based Surface Flow

The deformation of the model is governed by a nonlinear initial-value partial differential equation (PDE):

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

where F is the speed function, t is the time parameter, $\mathbf{k}$ and $\mathbf{k}'$ are the surface curvature and its derivative at the point $\mathbf{p}$, and $\mathbf{f}$ is the external force. $\mathbf{S}_0(\mathbf{p})$ is the initial shape of the model. $\mathbf{U}$ is the unit direction vector. In this paper, Equation 1 is explicitly simulated using the following iterative equation:

$$\mathbf{S}(\mathbf{p},t+\Delta t) = \mathbf{S}(\mathbf{p},t) + F(\mathbf{p},t)\mathbf{U}(\mathbf{p},t)\Delta t, \qquad (2)$$

where $F(\mathbf{p},t)$ is the speed function in Equation 1. Comparing with implicit level-set based simulation, explicit surface flow simulation allows the user to directly interact with the polygonal models without any intermediate conversion steps. In order to ensure the robust simulation of the PDE-based surface, we have to consider issues such as, model regularity, simulation step size, etc. Refer to [5] for more detail.

In order to represent shapes of arbitrary topology, the model must be able to change its topology properly whenever a collision with other parts of the model is detected. In this paper, we use a simple distance-based collision detection algorithm. Since the user is always actively involved during the deformation process, we assume that the topology modification occurs only under "good" conditions (i.e. there is no singularities). Our simple algorithm proves to be sufficient for our experiments. Collision detection is done hierarchically in two different levels: coarser-level and finer-level. Coarser-level collision detection is mainly for the purpose of collision exclusion.

We employ a novel method called "*lazy merging*" to handle topology modification. The basic idea is that whenever a collision occurs between two non-neighboring vertices (i.e., they become too close to each other), the two vertices will be freezed immediately (i.e., not allowed to move). Topology modification will happen later when all the vertices of the model become non-active, or a merge operation is en-

forced by the user. There are two steps in the topology merging operation: (1) *Merging-Vertices Clustering*, and (2) *Contour Stitching*. Figure 3 shows an illustration.

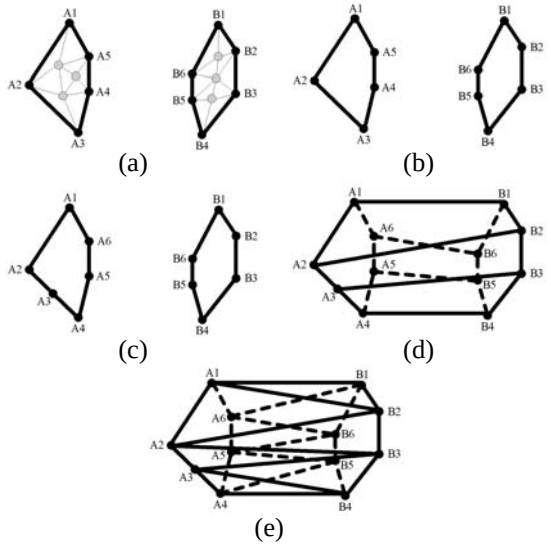


Figure 3. Topology modification. (a) Collisions are detected between two clusters of vertices, the interior edges are shown as gray lines and the interior vertices are shown as gray circles, the boundary edges are shown as dark lines and the boundary vertices are shown as dark circles. (b) The interior regions of the two merging clusters of vertices are removed, and the remaining boundary vertices are put into two separate linked lists A and B, respectively. (c) The two linked lists of merging vertices are put into correspondence. The longest edge A_2A_3 of linked list A is subdivided so that there are equal numbers of nodes in the two lists of merging vertices. (d) The corresponding vertices between the two lists of vertices are connected. (e) Each of the newly created quadrilaterals is split into two triangles.

The model-relaxation operations can quickly smooth out any artifacts that may result from the matching procedure once the topology merge has been completed.

In our system the basic force can be used by the user to grab the surface point on the polygonal object and an input force will be generated based on the user's action. We employ the Hooke's law to generate the force,

$$\mathbf{f} = k(\mathbf{p}_{curs} - \mathbf{p}_{surf}), \tag{3}$$

where $\mathbf{p}_{surf}$ is the surface point on the mesh which the user initially picks up, $\mathbf{p}_{curs}$ is the haptics cursor that the user controls to deform the mesh, and k is a positive spring constant. Usually the longer force vector, $(\mathbf{p}_{surf} - \mathbf{p}_{cursor})$, the user's action introduces, the larger external force will be generated. The generated force will be used in the simulation of surface flow (Equation 1) to evolve the mesh surface. Simultaneously, an equal but opposite force, $-\mathbf{f}$, will feed back to the user through the haptic device to get the haptic feeling.

Furthermore, our system provides a wide range of filtering functions such as Gaussian, spherical function, $\ell(d)$, to distribute the force into a set of mesh points in the nearby region. $\ell(d)$ also controls the region of influence, where the surface evolution occurs. Therefore, the force at a surface point $\mathbf{q}$ is

$$\mathbf{f}_\mathbf{q} = \ell(\|\mathbf{q} - \mathbf{p}_{surf}\|)\mathbf{f}. \tag{4}$$

The computed force $\mathbf{f}_\mathbf{q}$ is used in Equation 1. Together with geometric measures, such as the curvature, and the derivative of the curvature, it can produce a local motion that explicitly creates a desired global or regional behavior of the surface.

We implement the compressive force to help rendering and modify the surface properties. The compressive force at $\mathbf{q}$ is along the surface normal $\mathbf{n_p}$ at the closest surface point $\mathbf{p}$,

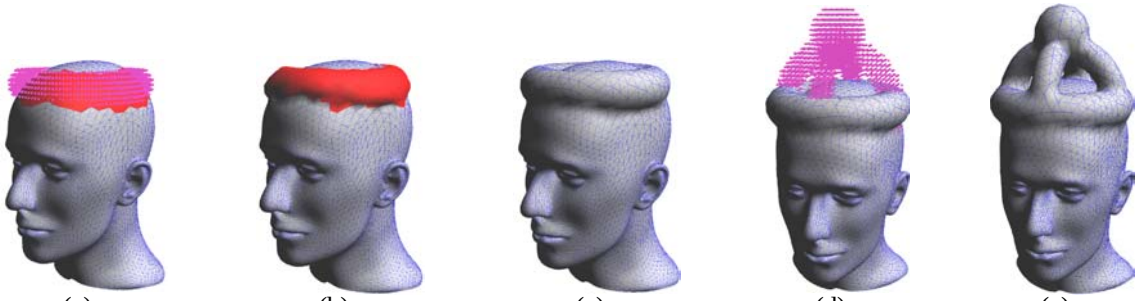
$$\mathbf{f_n} = \begin{cases} \lambda(\mathbf{q}-\mathbf{p})\cdot\mathbf{n_p}, & \text{if}(\mathbf{q}-\mathbf{p})\cdot\mathbf{n_p} < 0, \\ 0, & \text{otherwise}, \end{cases} \tag{5}$$

where $\mathbf{q}$ denotes the position of the haptics cursor, and $\mathbf{p}$ denotes the closest point $\mathbf{p}$ on the polygonal surface to $\mathbf{q}$. 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. Similar to the above basic force, an equal but opposite force, $-\mathbf{f_n}$ 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.

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

$$\mathbf{f}_s = -\mu\|\mathbf{f_n}\|\frac{\mathbf{v_p}}{\|\mathbf{v_p}\|}, \tag{6}$$

where $\mathbf{f_n}$ is the compressive force when the haptics cursor is at position $\mathbf{s}$, $\mathbf{v_p}$ is the projection of the velocity of the haptics cursor at $\mathbf{s}$ onto the tangent plane of the closest surface point $\mathbf{p}$. The friction force is a passive force purely for haptic rendering purpose. Therefore, it only feed back to the user and it is not considered in the surface evolution process.



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

Figure 4. The crowned mannequin.

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

Figure 5. The mannequin holed and wired.

5. Mesh Editing Operations

5.1. Blending and Drilling

Blending operation is conducted by doing a Boolean-like union operation between the embedding distance field of the mesh model and the user-defined distance field. The distance field can be created by the combination of simple primitives such as cylinders, spheres, etc, or it can be directly defined by volumetric datasets. For example, the input of Figure 4 (see also the color page in the appendix) is a polygonal model of a mannequin head. The user placed an implicit torus (shown in cyan color) on the top of the head (Figure 4(a)) and a union operation is selected. The locally affected regions of the head are marked as active (shown in red in Figure 4(a)), and will grow (Figure 4(b)) and finally stop (Figure 4(c)). Figure 5(e) shows another example. Here, a volumetric dataset of a spring is placed on the back of the mannequin head model, and the affected region of the model will deform according to the distance field defined by the volumetric spring dataset. Figure 5(f) shows the modified shape. Note that the mannequin head model is not a closed model (it has a big opening in the neck), level-set methods can not directly work on such kind of model.

Drilling operation is implemented in a similar fashion as the blending operation. Here, instead of a union operation, a subtraction is conducted between the embedding distance field of the mesh model and the user-defined distance field. For example, in Figure 5 (see also the color page in the appendix), the user wants to subtract an implicitly defined cylinder from the top of the head (Figure 5(a)). The locally affected region of the head (shown in red in Figure 5(a)) will shrink (Figure 5(b)) and change the topology (Figure 5(c)). Finally the normal diffusion flow proposed by Ohtake *et al.* [12] is applied to recover the sharp edges of the model (Figure 5(d)).

5.2. Force-Based Shape Manipulation

The user can apply forces through haptic device to directly manipulate the polygonal objects. The force is the basic force of Equation 3 and Equation 4 defined in the previous Section. The computed force $\mathbf{f_q}$ 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 $\ell(\|\mathbf{q}-\mathbf{p}_{surf}\|)\|\mathbf{f}\|$, the unit direction vector is $\frac{\mathbf{f}}{\|\mathbf{f}\|}$.

There are five main steps during a typical interactive design process:

1. The user selects a region of the model to be deformed by placing the haptics cursor around it.
2. The user applies force through the haptic device.
3. The corresponding region of the model deforms according to Equation 1, Equation 3, and Equation 4.
4. The system automatically conducts the collision detection operations. If there is a collision occurring between different parts of the model, the user will need

to decide whether or not to allow the system to change the topology of model.

5. The system automatically performs the model relaxation operations [5] to maintain the model regularity and smoothness.

These five steps will keep repeating until a user-desired shape has been obtained. Force-based shape manipulation enables the user to easily conduct a variety of mesh editing tasks such as extrusion, carving, drilling, etc.

5.3. Free-Form Sketching

Our system supports haptics-based free-form sketching for mesh manipulation. The user can draw some free hand strokes using the haptic device, either directly on the mesh or stem from the mesh. Strokes are then densely sampled by the system as a combination of Gaussian blobs that are assigned evenly at each point or as a collection of points and are converted to distance functions by methods such as the Fast Tagging algorithm [21]. Simultaneously, the affected regions of the underlying mesh model will automatically deform according to the corresponding scalar fields generated by the strokes. During these operations, the user can feel the force feedback and observe the shape deformation at the same time.

The PDE used here is the simplified version of the weighted minimal surface flow proposed by Caselles et al. [3]:

$$\frac{\partial \mathbf{S}}{\partial t} = (gv + g\|\mathbf{H}\|)\mathbf{N}, \tag{7}$$

where, $\mathbf{H}$ is the mean curvature of the surface, $\mathbf{N}$ is the unit normal of the surface, and v is a constant speed. g is the non-increasing, non-negative weight function that will stop the deformation of the model when it reaches the object boundary, and is defined as the commonly used 3D edge detector:

$$g(\mathbf{S}) = \frac{1}{1 + \|\nabla(I(\mathbf{S}))\|^2}, \tag{8}$$

where, I is the volumetric density function and ∇ is the gradient function.

To calculate the mean curvature $\mathbf{H}$ of the surface, we employ the discrete curvature estimator proposed by Desbrun et al. [4]:

$$\mathbf{H} = \frac{\sum_{j \in N_1(i)} (cot\alpha_j + cot\beta_j)(\mathbf{x}_i - \mathbf{x}_j)}{\sum_{j \in N_1(i)} (cot\alpha_j + cot\beta_j)}, \tag{9}$$

where $\mathbf{x}_j$ is one of the vertex at the one-neighborhood of $\mathbf{x}_i$. α_j and β_j are the two angles opposite the edge connecting the two vertices $\mathbf{x}_i$ and $\mathbf{x}_j$, and $\mathbf{H}$ is the mean curvature vector at vertex $\mathbf{x}_i$.

For example, in Figure 4(d), the user iteratively sketching two strokes on the mesh using the haptic device. The Gaussian blobs are assigned evenly at each sampling point. The blob field is shown in cyan color. Figure 4(e) shows the final shape of the model. Note that the non-trivial topology has been correctly represented.

5.4. Mesh cutting and pasting

5.4.1. Mesh cutting and blending Our system supports implicit function aided mesh cutting, blending and pasting operations. In this paper, mesh cutting is conducted by implicit primitives because of their convenient inside/outside properties. For example, the head of the cow (Figure 6(a)) and the head of the pig (Figure 6(b)) are cut by two implicit spheres, respectively. The body of the cow and the head of the pig are then aligned together by scaling, translation and rotation (Figure 6(c)), and the two parts are connected together by the contour stitching method (see also the color page in the appendix). Finally, the reconnected regions are smoothed out by the following mean curvature flow [4, 13, 17, 18]:

$$\begin{aligned} &\frac{\partial s}{\partial t} = H\mathbf{N}, \\ &S(0) = S_0. \end{aligned} \tag{10}$$

H is the mean curvature of the surface, and $\mathbf{N}$ is the unit normal of the surface S. Only the reconnected regions of the model will deform, and the deformation will stop when the mean curvature becomes zero (i.e., the velocity is zero). The mesh relaxation operators such as the tangential Laplacian operator and the three mesh optimization operations are also employed during the deformation. The final smoothed shape is shown in wireframe view and in rendered view in Figure 6(e) and Figure 6(f), respectively.

5.4.2. Mesh pasting Mesh pasting means directly pasting a mesh (source mesh) on the interior regions of another mesh (target mesh), and reconnecting them. This is different from mesh blending, where two meshes with open contours are connected together. For example, in Figure 7, the source mesh is the head of the dog (Figure 7(a)), the target mesh is the ellipsoid. The user first put the source mesh in the proximity of regions of interest of the target mesh. The boundary vertices of the source mesh are then projected onto the target mesh along its tangential normal direction (Figure 7(b)). This is done by the ray-triangle intersection method frequently used in the computational geometry community. In the interest of the space, we will omit the details here, please refer to the book written by O'Rourke [14] for more details. To reconnect the source mesh with the target mesh, we locally refine the target mesh several times by triangle quadrisection several times so that each triangle of the target mesh contains at most one source mesh vertex (Figure 7(c)). Then, the boundary vertices of the source mesh are connected with the target mesh (Figure 7(d)) by a

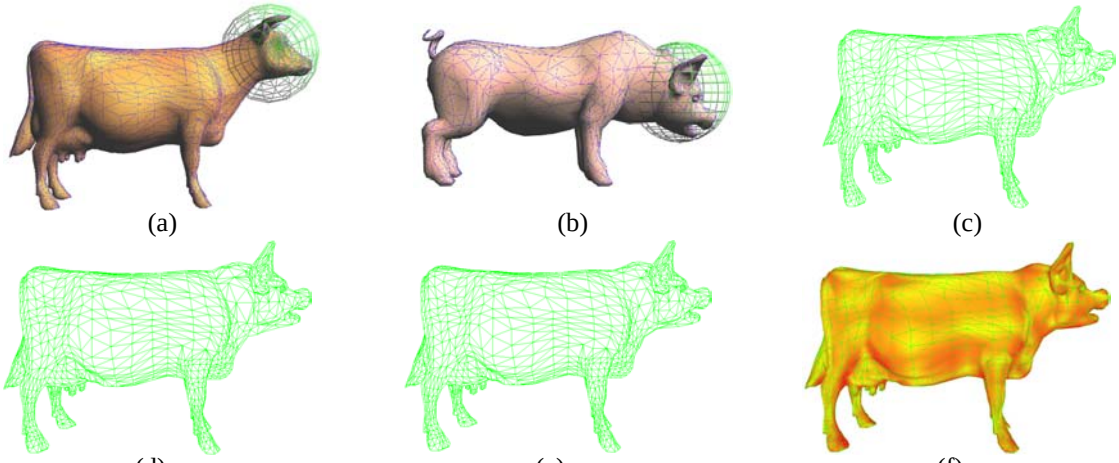


Figure 6. Mesh cutting and blending: (a)-(b) The head of the cow and the head of the pig are cut by implicit spheres. (c) The head of the pig and the body of the cow are aligned together. (d) The two parts are blended together by contour stitching. (e)-(f) The reconnected regions are smoothed out by mean curvature flow.

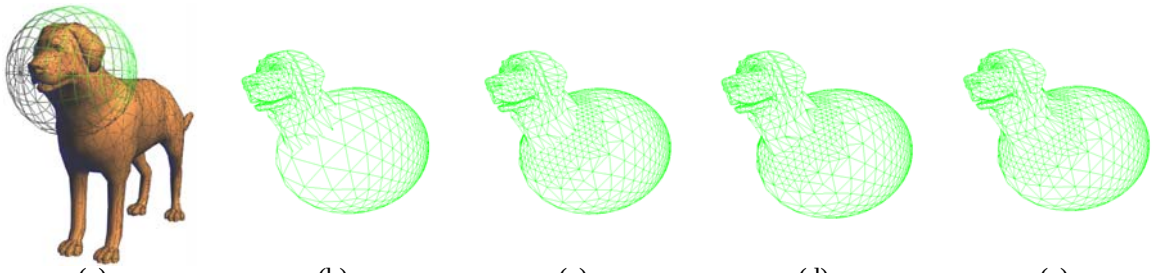


Figure 7. Dog-head egg.

new mesh snapping method that we propose in this paper. We will explain the new mesh snapping method in the following paragraph. To maintain the manifold geometry, the regions of the target mesh that are enclosed by the boundary vertices of the source mesh will be removed. Finally, the reconnected region will be smoothed out (Figure 7(e)) by the aforementioned mean curvature flow (Section 5.4.1) and mesh relaxation operators.

5.4.3. Mesh snapping In this paper, we proposed a new mesh snapping method to reconnect the boundary vertices of one mesh (the source mesh) with the corresponding vertices of another mesh (the target mesh). Before the reconnection, all the boundary vertices of the source mesh need to be projected onto the target mesh and are put into a linked list. In addition, the target mesh may need to be locally refined several times through triangle quadrisection so that each triangle of the target mesh contains at most one source mesh vertex.

Figure 8 shows an illustration of the mesh snapping method. Here, the vertices of the source mesh are shown in dark circles. The edges of the source mesh are shown in dark lines (both solid and dotted). The edges of the target mesh are shown as gray solid lines. Starting from the first vertex (shown in gray-colored circle in Figure 8(a)) in the linked list of the source mesh vertices, find its closest vertex on the target mesh (shown in small white circle), snap it to the position of the first source mesh vertex and merge these two vertices. Now (Figure 8(b)) the second vertex (shown in gray-colored circle) in the linked list becomes the current vertex. Check whether it is located in the 2-neighborhood of the previous source mesh vertex. If yes, then snap the closest one-neighborhood vertex (shown in small white circle) of the previous source mesh vertex to the current vertex and merge them (Figure 8(c)). Otherwise (Figure 8(c)-(d)), insert a new source mesh vertex in the middle between the current source mesh vertex and the previous source mesh vertex. Repeat the above steps (Figure 8(e)) until all the

source mesh vertices on the linked list are merged with the target mesh (Figure 8(f)). The linked list of the boundary vertices of the source mesh can be either open or closed. If the linked list is open (as shown in Figure 8), then a non-manifold patch is attached onto the target mesh. Otherwise, if the linked list is a closed list, then after the source mesh is attached onto the target mesh, either the regions of the target mesh that are enclosed by the boundary vertices of the source mesh or the regions of the target mesh that are outside the target mesh will be removed (e.g. Figure 7(e)).

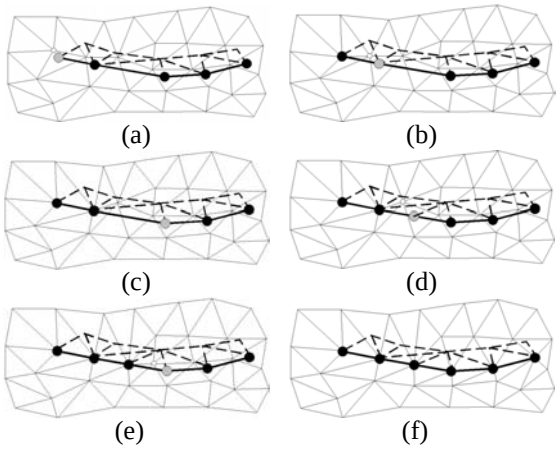


Figure 8. Merge the boundary vertices of the source mesh vertices with the corresponding vertices on the target mesh by mesh snapping. The source mesh vertices are shown in shaded circles, the source mesh edges are shown in dark colors (both solid and dotted lines), and the target mesh edges are shown in gray-colored lines. In each subfigure, the gray-colored circle is the current source mesh vertex that is merging with its corresponding vertex (small white circle) on the target mesh. (a) A chain of source mesh vertices has been projected onto the target mesh. (b)-(e) Merge the chain of source mesh vertices with the target mesh iteratively by mesh snapping. (f) The chain of source mesh vertices has been merged with the target mesh.

We can also paste a source mesh with the detailed visual information onto a target mesh. For example, in Figure 9, the source mesh is generated from a color image (Figure 9(a)) by treating the intensity function of the image as the height function of the mesh. The target mesh is a polygonal model of an apple (Figure 9(b)). To preserve the details, the source mesh is first decomposed into a base mesh plus a detail (height information) stored at each vertex. Only the base mesh will be projected onto the target mesh, and the details will be added back on each vertex along its current vertex normal direction (Figure 9(c)). Note that the target mesh is locally dense-sampled to accommodate the high resolution of the source mesh. Figure 9(d) shows the rendered view of the final pasted model.

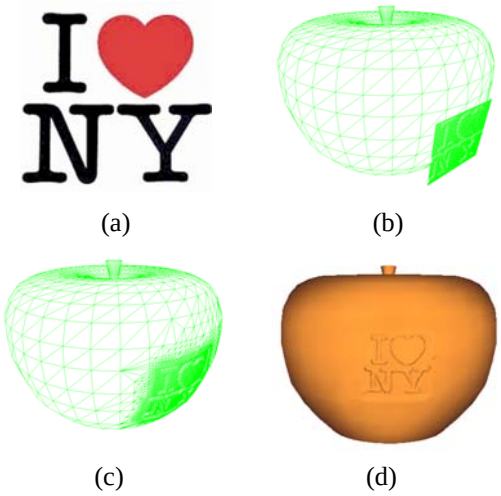


Figure 9. The big apple: I love New York.

6. Implementation and Results

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 (see Figure 2). The haptics, graphics, and simulation computations are each assigned one thread. Haptics computations usually do not require more than a single processor because the calculations are simple and must be performed approximately 1000 times per second. By having a dedicated CPU, the thread that controls the haptics needs not to compete with the computationally intensive simulation thread. By contrast, if the haptics calculations and physical simulations are performed on the same CPU, it is very likely that the haptics thread will not be allocated enough CPU time to guarantee the desired update rate of 1000 Hz. This results in buzzing, jerking, I/O latency, and other phenomena that interfere with tactile-based input/output. Therefore, we separate the force computation from the evolution of the polygonal model. By spreading the computational and graphics loads to separate processors, we free the haptics station to process user input/output exclusively. Graphical computations are also

normally assigned a single thread. Many computations related to display are now implemented in hardware and can be performed very quickly. The software is often responsible only for sending the instructions to the hardware to execute. In addition, since most graphics hardware boards are single-pipelined, in practice it may be less effective for more than a single thread of execution to access the graphics hardware at any given moment.

Evaluating the usefulness of haptic feedback is a very important step to provide helpful simulated sense of touch to users. Based on a qualitative approach, we perform a series of perceptual experiments to evaluate performance of those haptic tools in virtual sculpting tasks. After experiencing different haptic tools, the users all gave positive responses. As we all know, the polygonal objects do not have associated physical properties that can be used to derive faithful haptic feedback. However, the haptic tools that we provide is found useful and allows users to easily gain richer understanding of the 3D nature of the modeled polygonal objects. The interactions between the tools and models are much easier to control comparing with those interactions without haptics. The desired deformation of the polygonal object is easy to achieve with the available tools. The interface is found to be intuitive and easy to understand.

7. Conclusion

We have presented hands-on shape design in a haptic, stereoscopic environment founded upon PDE-based geometric surface flow. By facilitating haptics functionality and stereoscopic display, our system maximizes the potentials offered by PDE based surface flow, haptic interactions, and stereoscopic rendering. It provides an intuitive interface and allows users to directly manipulate 3D polygonal objects with hands. 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. Therefore, we plan to further extend current system into a network-based collaborative design framework in the future.

References

[1] R. S. Avila and L. M. Sobierajski. A haptic interaction method for volume visualization. In *Proceedings of the 7th IEEE Visualization '96*, pages 197–204, 1996.

[2] 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.

[3] 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.

[4] 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.

[5] Y. Duan, J. Hua, and H. Qin. Hapticflow: PDE-based mesh editing with haptics. In *Proceedings of IEEE Computer Animation and Social Agents*, pages 193–200, 2004.

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

[7] J. Hua and H. Qin. Haptics-based volumetric modeling using dynamic spline-based implicit functions. In *Proceedings of IEEE Symposium on Volume Visualization and Graphics 2002*, pages 55–64, 2002.

[8] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A skeching interface for 3D freeform design. In *SIGGRAPH '99 Proceedings*, pages 409–416, 1999.

[9] 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.

[10] L. Markosian, J. M. Cohen, T. Crulli, and J. Hughes. Skin: A constructive approach to modeling free-form shapes. In *SIGGRAPH '99 Proceedings*, pages 393–400, 1999.

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

[12] Y. Ohtake, A. Belyaev, and A. Pasko. Dynamic meshes for accurate polygonization of implicit surfaces with sharp features. In *Proceedings of Shape Modeling International 2001*, pages 74–81, 2001.

[13] Y. Ohtake, A. G. Belyaev, and I. A. Bogaevski. Mesh regularization and adaptive smoothing. *Computer Aided Design*, 33(4):789–800, 2001.

[14] J. O'Rourke. *Computation geometry in C*. Cambridge University Press, 1998.

[15] J. K. Salisbury and C. Tarr. Haptic rendering of surfaces defined by implicit functions. In *Proceedings of the ASME Sixth Annual Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 15–21, 1997.

[16] 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.

[17] G. Sapiro. *Geometric Partial Differential Equations and Image Analysis*. Cambridge University Press, 2001.

[18] J. A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, second edition, 1999.

[19] T. V. Thompson, D. E. Johnson, , and E. Cohen. Direct haptic rendering of sculptured models. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 167–176, 1997.

[20] R. C. Zeleznik, K. Herndon, and J. Hughes. Sketch: An interface for sketching 3D scences. In *SIGGRAPH '96 Proceedings*, pages 163–170, 1996.

[21] H. Zhao, S. Osher, and R. Fedkiw. Fast surface reconstruction and deformation using the level set method. In *Proc. Of the IEEE Workshop on Variational and Level Set Methods in Computer Vision*, pages 194–201, 2001.