Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Graphics 30 (2006) 610–618
COMPUTERS & GRAPHICS
www.elsevier.com/locate/cag

Technical Section

3D warp brush modeling

Yong Joo Kil[a,b], Pietro Renzulli[c], Oliver Kreylos[b], Bernd Hamann[a,b], Guiseppe Monno[c], Oliver G. Staadt[a,b,*]

[a]*Department of Computer Science, University of California, Davis, One Shields Avenue, Davis CA 95616, USA*
[b]*Institute for Data Analysis and Visualization, University of California, Davis, One Shields Avenue, Davis CA 95616, USA*
[c]*Department of Mechanical Engineering, Politecnico di Bari, Viale Japigia 182, 70126 Bari, Italy*

Abstract

We introduce 3D warp brush, a new method for interactive shape modeling in an immersive virtual reality environment. 3D warp brushes are implicitly defined tools that operate on triangle meshes. We combine the efficiency of explicit mesh representations with intuitive implicit modeling operators. The area of influence of a 3D warp brush can be of arbitrary shape since it has an associated distance field. We define different warp functions including drag, explode, and whittle. A unique feature of our framework is the ability to convert meshes into 3D warp brushes at run time. Thus, we can easily expand our set of brushes based on a small set of base brushes, such as spheres or ellipsoids. Our underlying split-edge mesh data structure supports adaptive refinement and efficient rendering with on-the-fly triangle strip generation. 3D warp brushes only operate on mesh vertices, hence, underlying mesh processing is transparent to the modeling operations. The use of a Responsive Workbench and two-handed interaction allows the user to exploit the full potential of the modeling system by intuitive and easy modification of a base surface into a desired shape. We present several models, which have been created and modified using 3D warp brushes, to demonstrate the usefulness of our framework.

Keywords: Free-form modeling; Virtual reality; Human-computer interaction

1. Introduction

Three-dimensional surface modeling is a research area of particular interest in computer graphics, computer-aided design (CAD), and computer-aided geometric design (CAGD). 3D surface models are being used in diverse application areas ranging from industrial and car-body design to 3D character animation and computer games. Traditional surface modeling methods—including parametric surface representations such as Bézier and NURBS surfaces—achieve high-quality results, but require manipulation of control meshes to influence surface shape. This indirect manipulation metaphor requires the designer to change control vertices using either a mouse or a tablet. Even though input devices with higher degrees of freedom have become more widely available, it is unclear how they can be successfully utilized in such settings.

Free-form deformation (FFD) techniques support natural interaction with the surface object. They support the use of powerful tools to manipulate the model in a direct fashion, without using control meshes. Especially in the context of implicit (i.e., volumetric) representations, tools can easily be defined to operate on the surface. Unfortunately, memory requirements of complex implicit surface representations are high, and rendering involves extraction of the surface using marching-cubes-like methods or direct volume visualization. On the other hand, explicit surface representations (i.e., meshes) can be rendered efficiently using commodity graphics hardware.

Virtual reality (VR) has over the years proved to be an excellent medium for the above-mentioned modeling techniques. Specifically the large amount of research on

*Corresponding author. Tel.: +1 530 752 4821; fax: +1 530 752 4767.
E-mail addresses: kil@cs.ucdavis.edu (Y.J. Kil), rufusgufus@libero.it (P. Renzulli), kreylos@cs.ucdavis.edu (O. Kreylos), hamann@cs.ucdavis.edu (B. Hamann), gmonno@poliba.it (G. Monno), staadt@cs.ucdavis.edu (O.G. Staadt).

0097-8493/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2006.03.014

VR interaction, such as [1–3] carried out in the past and the technical improvement of the relative hardware, make VR suitable for applications such as conceptual shape modeling, where the real-time 3D visual feedback and two-handed interaction are fundamental for the success of the creative and iterative mental processes involved.

We propose a hybrid free-form modeling framework that uses a VR interface and combines implicitly defined tools—3D warp brushes—that operate on adaptively refined triangle meshes. We present data structures and algorithms that support 3D warp brushes with arbitrary shape and a variety of associated warp functions that determines how the shape of the model is changed. Our underlying mesh representation is based on a split-edge data structure that allows for dynamic refinement and fast rendering. A powerful feature of our framework is the ability to convert a surface model to a 3D warp brush. In other words, we can model complex tools within our system and extend the library of warp brushes. Basic warp brushes can be created from pre-existing distance fields, e.g., procedurally defined ones such as spheres or cylinders.

The result is a modeling system that in conjunction with the 3D VR interface, makes conceptual free-form shape modeling simple and intuitive. In particular, this system provides the user with a means of rapidly generating, modifying, and visualizing conceptual shapes in an iterative manner until the desired result is achieved. This result may then be used as a starting point for the aesthetic, ergonomic, and functional evaluations typical of the later stages of the design process.

The capabilities of our modeling framework are illustrated in Fig. 1. It depicts a sequence of models created using 3D warp brushes, starting from a spherical base mesh.

This paper is a detailed and expanded description of the 3D warp brush modeling framework that has been presented as a poster in [4].

2. Related work

Shaw et al. [5] introduced a rectangular-mesh editor that uses 6 degree-of-freedom (6DOF) input devices. By selecting different tools the mesh can be refined, can have other quads added to it, and vertices can be moved. Various refinement levels can be used to subdivide the mesh. The user can restrict the selection to certain quads representing a certain level of refinement. This forces the user to think in terms of the underlying mesh structure, which may make the modeling process non-intuitive.

The 3D styling system described by Wesche and Droske [6] and Wesche and Seidel [7] features a Responsive Workbench display and two-handed interaction. They support curve and surface deformation using a variational approach based on an underlying B-spline representation. The modeling system developed by Matsumiya et al. [8] is based on implicit surface representations. The user wears a data glove and deforms the surface using a finger, but the system does not support more complex deformation tools.

Schkolne et al. [9] describe a free-form modeling technique for VR environments. Hand movements are used to generate free-form triangulated surfaces in 3D space. Cybergloves are used to get the information regarding the hand position and orientation. The authors mention that their mesh creation algorithm generates cracks in the mesh and non-manifold topology about one percent of the time. A Laplacian smoothing operator is

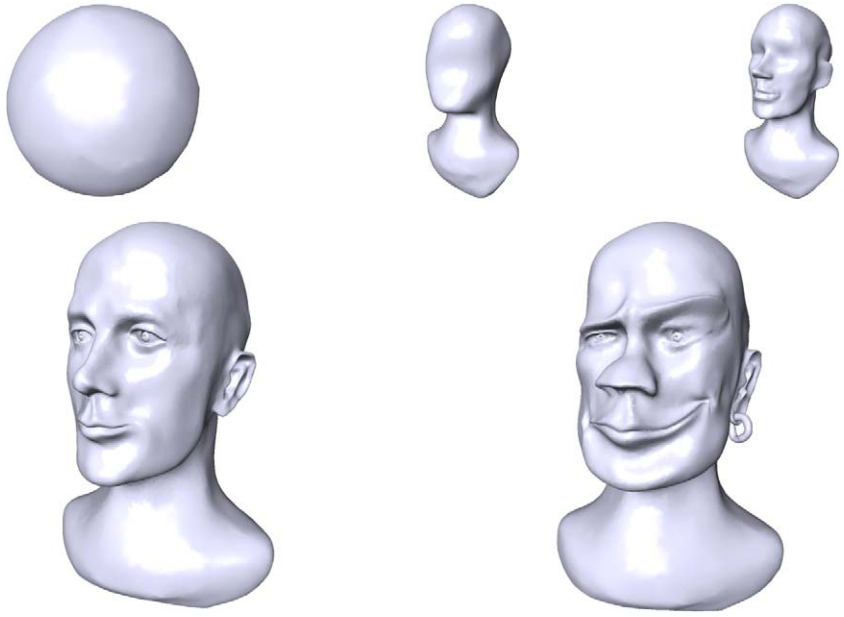

Fig. 1. Sequence of snapshots from an interactive modeling session. An initial surface (a sphere) is warped into a complex bust using several brushes and warp operations.

applied in a post-process to eliminate of unwanted "bumps" on the surface.

Bill and Lodha's [10] sculpting system uses geometric objects called virtual tools, described by super-quadratic equations, which define an area of influence which when in contact with the mesh can adaptively refine, smooth or deform it in different ways. Decay functions define how the influence of a tool applied to a point in the mesh decays and effects the neighboring points. The size of the tools can be changed by scaling along the principal axis.

Based on the kind of geometric representation used to describe the modeled object, various shape modification techniques can be used. In general, the simplest way of editing explicit polygonal meshes, parametric representations, and subdivision surfaces entails inserting, moving, and deleting single control vertices or groups of these. A more elegant technique of deforming surfaces is via the use of FFD [11,12], which involves warping a space that contains the object to be modeled. FFD methods are independent on the underlying data structure and, hence, can be used with any parametric, polygonal, or point-based representation. However, this approach to modeling has the problem that the control lattice used to manipulate space is not directly related to the object.

An approach to FFD in a more direct manner has been proposed in [13]. An interesting alternative to volumetric FFD is the one presented by Botsch et al. [14], which is based on constrained shape optimization. This modeling framework allows users to choose between different abstract basis functions, controlled by virtual 9DOF manipulator objects, which are used to perform shape editing of unstructured triangle meshes. A very effective way to generate new shapes with implicit surfaces or distance field representations is by blending one or more of these representations via Boolean operations [15]. Another commonly used technique for shape modeling using implicit representations involves generating a skeleton [16] of the intended shape around which a smooth implicit surface is generated. Llamas et al. [17] have recently presented a two-handed editing system that supports FFD by using point-displacement constraints that must be interpolated by the deformation.

In general, explicit surface modeling [5,9,10,18] using polygonal meshes has the advantage that polygon rendering is supported efficiently in modern graphics hardware. However, when modeling complex surfaces, a large number of polygonal primitives may be necessary to represent fine surface details, making real-time interaction no longer possible.

3D warp brush, the free-form modeling system described in this paper, is similar in nature to the polygonal sculpting system described in [10]. However, to overcome the limitation of a 2D environment and to make the interaction as intuitive and easy as possible, we use a VR environment with 6DOF input devices. Moreover, we allow the area of influence of the tools to be of any shape. In fact, since each tool has a distance field [15,19] associated with it, any modeled shape, represented as a mesh, can be turned into a tool and used to deform another mesh. In other words, the area of influence of a tool can be changed via another tool into any desired shape, hence greatly enhancing the modeling capabilities of the system.

A similar approach to ours that uses distance fields is the one described by Angelidis et al. [20]. They focus on guaranteeing a surface from self intersection by integrating the function defining the deformation over small time steps. Although the results are very effective, user interaction is limited to a 2D interface.

3. System overview

We use a similar setup to the one described in [2] with a Responsive Workbench display (see Fig. 2) that utilizes active stereo shutter glasses. A Polhemus Fastrak 6DOF tracking sensor is attached to the shutter glasses, allowing the system to display correct perspective image for the viewer at any location. For two-handed interaction we employ two Fakespace Pinch Gloves with 6DOF sensors and one Polhemus stylus.

A high-level overview of our software architecture is depicted in Fig. 3. At the core of our system is a self-adapting triangle mesh (see Section 5) representing the surface of the object currently being modeled. Modeling is performed by direct free-form manipulation of the represented surface, using one or more surface manipulators associated with the 6DOF input device(s) present in the virtual environment. A surface manipulator combines the effects of a *brush* and a *warp operation* (see Section 6). Brushes define the spatial shape of a manipulator and are defined by distance fields. Warp operations define how a manipulator modifies surface vertices and are defined by vector fields.

An important contribution of our modeling system is the ability to extend the set of available brush shapes from within the system. The brush generator can "clone" a part

Fig. 2. Setup of the 3D warp brush modeling system.

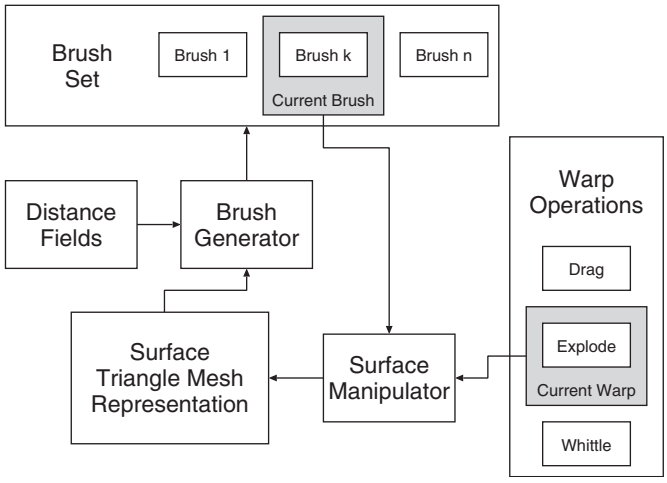

Fig. 3. Overview of the system architecture.

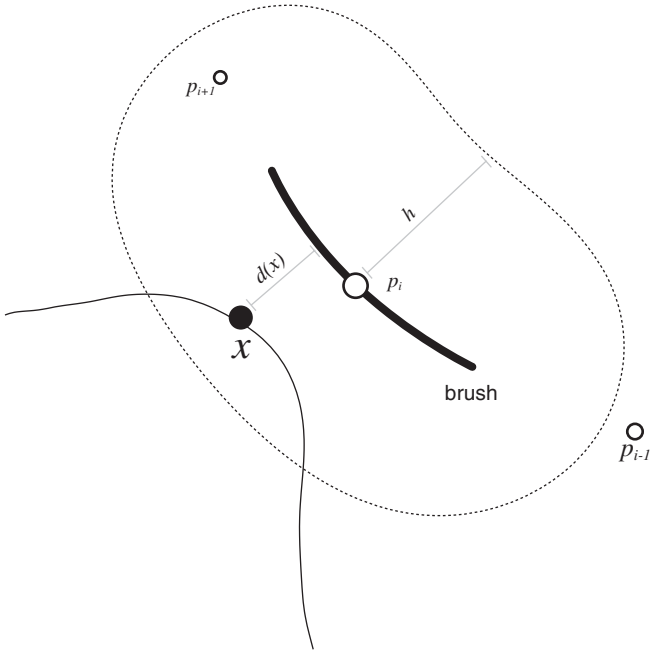

Fig. 4. A brush at position p_i interacts with a model. Vertex x on the model's surface is within the region of influence h of the brush. As the brush moves through space, the surface vertices within this area are translated according to the warp function.

of the current surface and convert it into a distance field to be added to the set of available brushes. This allows starting from a set of pre-defined brushes loaded as existing distance fields or generated from analytical representations, e.g., spheres and ellipsoids, and generating successively more complex brushes which are finally used to model a desired shape.

4. The 3D warp brush

Our core modeling tool, the 3D warp brush, employs a brush-like paradigm with a limited area of influence in a local region of space, defined by a distance field and several user-defined parameters such as region of influence and pressure. As a warp brush moves through space, a vector field is defined locally around the brush. This is realized through warp operators $f: \mathbf{R}^3 \rightarrow \mathbf{R}^3$. As the brush comes into contact with the model, the warp operator is modified by weight functions calculated from the brush's distance field and applied to the vertices of the model.

4.1. Input data

During interactive modeling, we focus on the model and the brush. Input to our system is a sequence of positions and orientations $((p_i, o_i))$, which is updated by the 6DOF tracking system at every frame (see Fig. 4). The brush is associated with a tracking device and the current measurement (p_i, o_i) defines the relative position of the brush and the current model. Some warp operations are based on motion. Thus, we derive a sequence of measurement increments $((\Delta p_i, \Delta o_i))$ using backward differencing: $\Delta p_i p_i = p_{i-1}$, and $\Delta o_i = o_i / o_{i-1}$.

To give users better control, we create an interface for parameters such as scale and pressure. Scaling factors for models and brushes are maintained separately, allowing the user to change their size relative to each other. This simple but important feature allows the user to work at different levels of detail. Pressure is utilized to define the overall sensitivity of the warp operators.

4.2. Distance field

To define the effect of a brush through warp operators, we utilize the concept of *distance fields*. A distance field $d_S(x)$ is a scalar field that represents the minimum distance between a point x and a given surface S. In our case, S is known as the current brush's surface, and we express the distance field as $d(x)$. In Euclidean space, the value of $d(x)$ can be interpreted as the Euclidean distance between x and the point s on the surface that is closest to x: $d(x) = \|x - s\|$. A *signed* distance field can be defined for oriented surfaces or solid models, and assigns negative distances to points inside the surface or model.

Our warp operators also use the gradient ∇d of the distance field, which is a vector in the direction of maximum change:

$$\nabla d = \left(\frac{\partial d}{\partial x}, \frac{\partial d}{\partial y}, \frac{\partial d}{\partial z} \right).$$

In Euclidean space, $\|\nabla d\| = 1$, and $\nabla d(x)$ is parallel to the difference vector $x - s$, where s is again the point on the surface closest to x. Thus,

$$\nabla d(x) = \frac{x - s}{\|x - s\|}.$$

For fast access to the distance field and its associated gradient field, we pre-compute both on a regular grid. This is efficiently calculated using the closest point transform

[21]. For higher accuracy, but slower acquisition, adaptive distance field can be used [15], or a hierarchical decomposition of the triangle mesh can be pre-computed [22].

4.3. Weight functions

We use distance fields to calculate weight functions for the application of warp operators to surface vertices. Weight functions have a value of one on the brush's surface ($d(x) = 0$), and decrease to zero as $d(x)$ approaches the brush's user-defined influence region size denoted by h (see Fig. 5). Writing *normalized* distance as $d_n(x) = d(x)/h$, we can define a linear weight function (see Fig. 5, left)

$$w_1(x) = 1 - d_n(x),$$

or a smooth third-degree polynomial weight function (see Fig. 5, right)

$$w_2(x) = 1d_n^2(x)(3 - 2d_n(x)).$$

The linear weight function w_1 has the advantage of more intuitive warp behavior, especially for rotational transformations, while the smooth weight function w_2 better preserves surface smoothness. Additionally, weight functions are defined to be one for negative distance values (in the case of signed distance fields), and to be zero for distance values larger than h. This ensures that surface vertices outside a brush's influence region are not affected by warp operators.

To give a user overall control over the effect of a warp operator, we add a *pressure* parameter $\rho \in [0, 1]$ to either weight function to calculate the final $w(x) = w_i(x) \cdot \rho$, for $i \in \{1, 2\}$.

4.4. Warp operators

Warp operators are defined as vector fields that impose movements on surface vertices inside the influence region of a brush. At each frame, the position x of a surface vertex v is adjusted by multiplying the weight $w(x)$ with the warp operator value $f(x)$, and adding the result to the current value x:

$$x' = x + w(x) \cdot f(x).$$

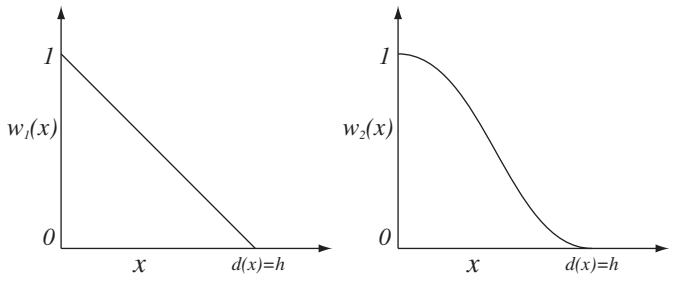

Fig. 5. Weight functions are used to modulate the effect of warp operators on surface vertices and to localize brush effects. Vertices close to a brush's surface have high weights, and weight values decrease to zero as vertices move towards the border of a brush's user-defined influence region size h.

Warp operators can be arbitrary vector fields, and they can depend on a vertex' position, on the brush position, on the brush motion, or on any combination of these. Our current modeling system comprises three different warp operators: *drag*, *explode and collapse*, and *whittle*.

(1) *Drag*: Drag is a simple yet powerful way of manipulating the model by allowing pushing, pulling, and twisting of regions of its surface. This operator's vector field is based on brush motion only.

A measurement increment $(\Delta p_i, \Delta o_i)$ generated from the stream of 6DOF tracker measurements is converted into a (implicit) vector field by calculating the movement of each point in space as if it were rigidly attached to the moving tracker coordinate frame. A position increment Δp_i is a vector, and an orientation increment Δo_i is a rotation, to be performed around the current position p_i (Δo_i can be represented, for example, as a matrix or a quaternion). Thus, the new position of a point x under the incremental transformation is

$$x' = p_i + \Delta o_i \cdot (x - p_i) + \Delta p_i,$$

and its offset vector is

$$f_{drag}(x) = p_i + \Delta o_i \cdot (x - p_i) + \Delta p_i - x.$$

(2) *Explode and collapse*: *Explode and collapse* has the effect of moving the model's surface towards or away from the brush's surface. The effect can be thought of as blowing up, or sucking the air out of, a balloon containing the brush; or as stamping the brush into the model's surface. The explode and collapse operator always moves vertices in the direction of the gradient field of the brush's distance field. Its behavior is modified by an additional parameter δ: if $\delta < 0$, surface vertices move towards the brush (collapse), if $\delta > 0$, surface vertices move away from the brush (explode). Thus, the explode and collapse operator is defined as

$$f_{explodeCollapse}(x) = h \cdot \nabla d(x) \cdot \delta.$$

(3) *Whittle*: The whittle operator is used to locally smooth the surface of a model. This is important since the 6DOF tracker measurements typically contain some amount of noise, which introduces noise into the model's surface while editing. Our system uses Laplacian smoothing to modify the positions of surface vertices inside the brush's influence region.

Laplacian smoothing is a discrete operation that replaces the position of each vertex in a mesh with the centroid of its neighboring vertices. In other words, if v is a mesh vertex with neighbors $\{v_1, v_2, \ldots, v_n\}$, then v's new position x' is defined as $x' = 1/n \sum_{i=1}^{n} x_i$, where the x_i are the positions of the neighbors v_i. To apply Laplacian smoothing locally, and in a more controlled fashion, we convert it into a warp operator by defining a vector at each mesh vertex's position as the offset from the vertex's current position to its new

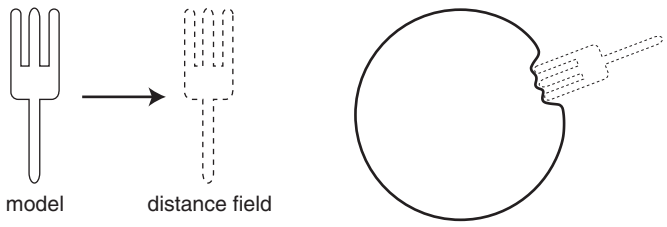

Fig. 6. Custom brushes can be created by converting existing or modified models into distance fields. The custom brush can then be used to manipulate other surfaces.

position if Laplacian smoothing were applied:

$$f_{whittle}(x) = \left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) - x.$$

This discrete set of vectors can be extended to a vector field in an arbitrary way, since it will only be evaluated at the positions of mesh vertices.

4.5. Brush generator

As described above, our system represents brush shapes as distance fields $d(x)$. These distance fields can be generated in several ways. They can be given analytically, e.g., for spherical or ellipsoidal distance fields, or they can be loaded as pre-computed regular grids. This generates distance fields for brushes directly from the surface of our models, which allows intuitive creation of complex brush shapes using the same modeling tools that are used for surface modeling. This concept is illustrated in Fig. 6, where we used a fork-shaped model to generate a fork-shaped brush which is subsequently used to manipulate a spherical surface.

5. Surface representation

The main goal in developing our surface representation was to strike a compromise between the ease of rendering of polygon meshes, and the ease of manipulation of point set surfaces, especially when inserting/removing points. The result is a triangle mesh data structure, with built-in operations for point insertion and point removal. The surface is rendered just like a triangle mesh, but it behaves like a point set under manipulation. When the mesh is stretched, and local point density becomes too low, new points are inserted automatically; when the mesh is condensed, and local point density becomes too high, points are removed automatically. Additionally, the point insertion/removal operations are implemented such that the shape of triangles in the mesh is as uniform as possible, preferring isosceles triangles over long and skinny ones (see Fig. 7).

5.1. Triangle mesh data structure

Our surfaces are represented as triangle meshes, implemented using a *split-edge* data structure [23]. We chose this data structure for its straightforward representation of mesh connectivity, its $O(1)$ traversal time between adjacent triangles, and its ability to (temporarily) represent meshes containing arbitrary polygons during point insertion/removal.

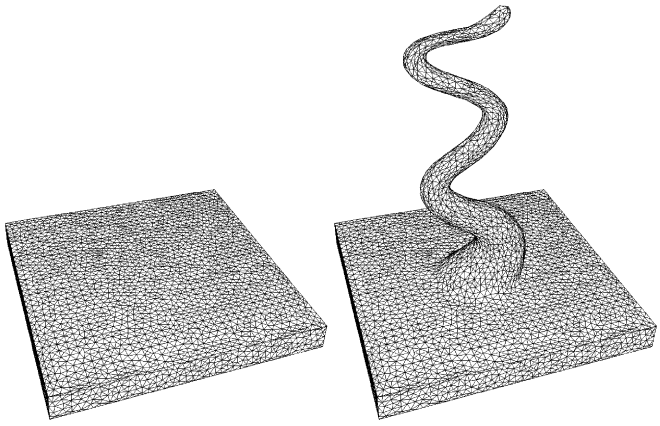

Fig. 7. Part of a surface before and after manipulation. Left: initial mesh. Right: mesh after local deformation. Note the uniform triangle shapes even in regions where large stretching occurred.

In a split-edge data structure, internal mesh edges are represented as pairs of oriented *half-edges* originating at one mesh vertex and ending at another. Each half-edge contains a pointer to its opposingly oriented sibling half-edge. Mesh faces (in our case, triangles) are represented implicitly, as closed loops of half-edges. For this purpose, each half-edge also contains a pointer to the next half-edge around the same mesh face.

5.2. Mesh rendering

In the most straightforward implementation, a triangle mesh is rendered by traversing the list of triangles twice. In the first pass, each triangle's plane equation is computed, and a triangle's normal vector is accumulated in each of the triangle's vertices. In the second pass, each triangle's half-edge loop is traversed exactly once, and each half-edge's start vertex (with its previously computed normal vector and other attributes) is passed to the OpenGL pipeline in *GL_TRIANGLES* rendering mode.

The $O(1)$ traversal of neighboring triangles provided by the split-edge data structure enables a more efficient rendering technique that generates triangle strips on the fly during the second traversal pass. While traversing the list of triangles, each triangle that has not yet been rendered serves as a seed for a triangle strip. A strip is extended until it runs into a boundary edge, or a triangle that has been rendered already. Triangle strip vertices are passed to the OpenGL pipeline as they are encountered. Even with this unoptimized, "random" mesh traversal, the generated triangle strips are typically long enough to improve rendering performance compared to the naive rendering method.

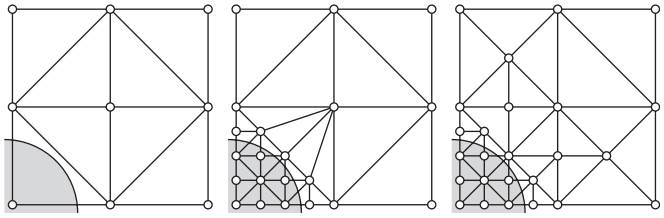

Fig. 8. Increasing mesh resolution inside an area of interest using longest-edge bisection. Left: initial mesh and area of interest (shaded). Center: refined mesh with longest-edge bisection. Right: refined mesh with recursive longest-edge bisection.

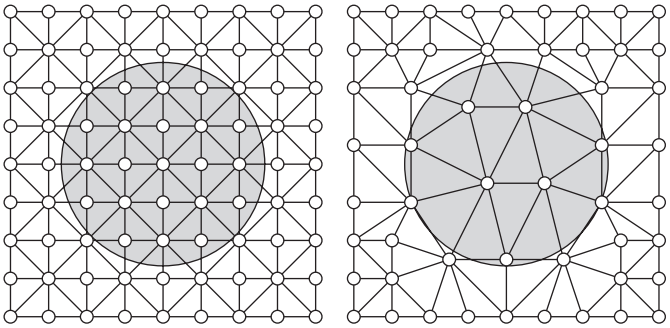

Fig. 9. Reducing mesh resolution inside an area of interest using shortest-edge collapse. Left: initial mesh and area of interest (shaded). Right: coarsened mesh.

5.3. Mesh resolution adaptation

One of the main benefits of point set surfaces is the ease which with one can insert additional points to locally increase the resolution of a surface representation in areas of manipulation (see Fig. 7). To achieve the same flexibility for triangle meshes, we implemented automatic maintenance of mesh resolution into the split-edge data structure. Whenever a surface is manipulated, all triangles that changed shape in the manipulation are checked against a pair of resolution thresholds: if a triangle's longest edge is longer than the *max-edge-length*, that edge will be split, thereby inserting a new point into the mesh. If a triangle's shortest edge is shorter than the *min-edge-length*, that edge will be collapsed, thereby removing a point from the mesh.

(1) *Edge split*: Insertion of new points into a triangle mesh is achieved by *recursive longest-edge bisection* [24] (see Fig. 8). If a triangle's longest edge is longer than the max-edge-length, the edge is split by inserting a new point at its center. To avoid hanging nodes, the other triangle sharing the same (interior) edge is split as well. Additionally, to maintain desirable triangle shapes, we first recursively split the other triangle's longest edge if the shared edge is not the other triangle's longest edge (see Fig. 8, right).

(2) *Edge collapse*: Removal of points from a triangle mesh is achieved by *shortest-edge collapse* (see Fig. 9). If a triangle's shortest edge is shorter than the min-edge-length, the edge is collapsed by moving both of its endpoints to the edge's center, and then removing one point and two triangles from the mesh. It is worthy of note that our used edge collapsing method does not invert a previously performed edge bisection, and vice versa. It turns out that the combination of the two methods, when applied consecutively to a mesh under manipulation, results in uniform point density and desirable triangle shapes (see Fig. 7).

6. User interaction

When immersed in the virtual environment, a user can use toolboxes (arrays of 3D boxes containing an icon of a brush) and menus that can be positioned anywhere in the virtual workspace (see Fig. 10), to select a deforming brush

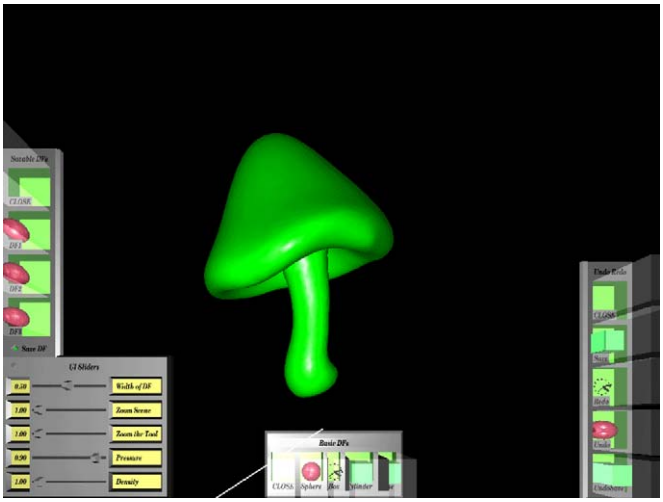

Fig. 10. The interaction interface in the virtual environment. At the center is the shaded mesh representing the model. Around this is a series of toolboxes and sliders to control the interaction.

with the relative actions to be applied to the mesh. Other widgets such as sliders, which are accessible via either the stylus or the Pinch Gloves, are used to set the properties of the brushes such as size of the distance field and the scale of the shape of the brush.

Once a brush is instantiated it is associated to the button of the input device that was used to select it from the menu or toolbox. In the case of the gloves, a different brush or deformation can be associated to each of the four fingers. The movement of the input device (with the respective button pressed) can be used to bring the distortion field associated with the brush in contact with the mesh to deform it.

In order to best utilize two input devices, we employ the two-handed interaction paradigm [2] and use the left hand for orienting the model and the right hand for modifying the model; scaling of the model and of the brush is done similarly by measuring the relative distance between the two hands. This is an invaluable tool which frees us from the constraints of the physical world, and allows us to freely orient the model and at different scales.

7. Results

We have implemented the software components of our system in C++ on a Pentium 4 2.8 GHz, with 2 Gbyte memory and NVIDIA Quadro FX 1000 graphics. The system has been informally tested with users who had no technical knowledge of the application. The feedback has been very positive and the ease with which the users learned to use the application indicates that the aim of creating an easy to use conceptual shape design application has been achieved. We implemented a way of undoing modeling actions and a means to recreate such actions so that the user could feel free to change the model, preview the change and go back to the previous step if the result was not the desired one. Stereoscopic vision and the 3D input devices have proven to be very important in making the application intuitive and easy to use. In fact when the application is run on a desktop computer with a monitor, and mouse and keyboard as input devices, the creative process slows down considerably. The possibility of modeling the shape of the brushes on the fly has proven very useful since the user can potentially generate any shaped brush to deform the mesh.

Fig. 11 show simple examples of a spherical brush. Fig. 12 shows tools that were generated and then used to modify the surface. The bust seen in Fig. 1 bottom left was made by visually referencing a low res model of a head. This allows an artist to easily reference relative positions of features. The modeling of the sphere to the bust took less than two hours. Fig. 13 depicts additional steps of modeling the bust from Fig. 1, which took about 10 min.

8. Conclusions and future work

We developed a new system for free-form modeling of triangle meshes using 3D warp brushes. Defining the shape of warp brushes using signed distance fields along with different warp function allows us to create new brushes in an efficient and flexible manner. The use of this system in immersive VR environments together with two-handed interaction using 6DOF input devices, enables direct manipulation of surface shapes in an intuitive and creative

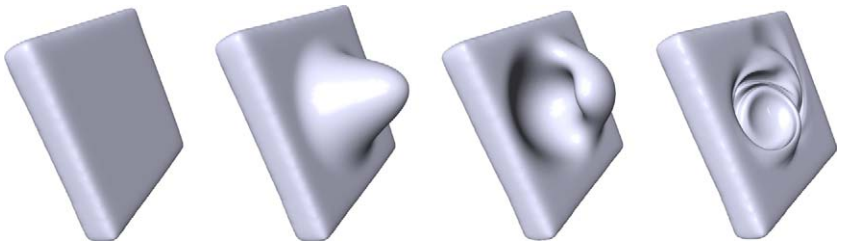

Fig. 11. Effects of a spherical brush and the drag warp operator applied to a square slab. From left to right: original slab, simple translation, simple rotation, complex combination.

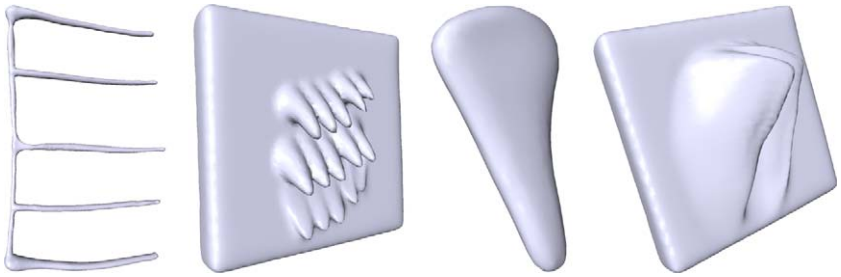

Fig. 12. A "fork" tool (a) and the result of applying it to a square slab (b), and a "triangle" tool (c) and the result of applying it to a square slab (d).

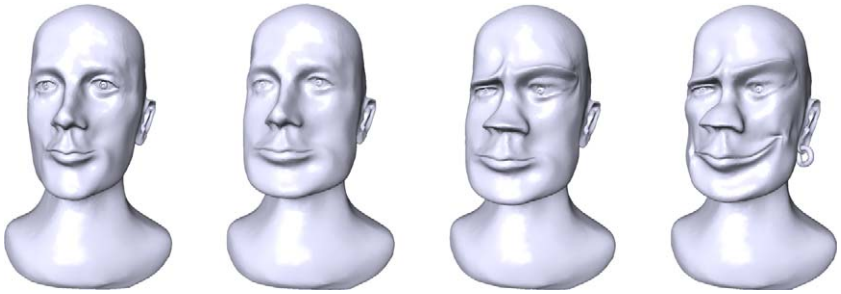

Fig. 13. Sequence of snapshots from an interactive modeling session from the left model to the resulting right model. The process took less then 10 min.

fashion. Our results demonstrate the capabilities of our system with several example models.

We plan to extend the current system with the ability to apply interactively textures to a model's surface. This will allow us to manipulate not only the shape, but also the appearance of meshes. In addition, we will make available this system to artists and designers to further explore its capabilities. We expect valuable feedback from them that will guide us to improve the framework.

Acknowledgements

This work was supported by the National Science Foundation under contracts ACI 9624034 (CAREER Award), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, and a large Information Technology Research (ITR) grant. We thank the members of the Visualization and Computer Graphics Research Group at the Institute for Data Analysis and Visualization (IDAV) at the University of California, Davis.

References

[1] Mine M. Exploiting proprioception in virtual-environment interaction. Technical Report TR97-014, UNC Chapel Hill, CS Department; 1997.

[2] Cutler LD, Fröhlich B, Hanrahan P. Two-handed direct manipulation on the responsive workbench. In: Symposium on interactive 3D graphics; 1997. p. 107–14.

[3] Bowman D, Johnson D, Hodges L. Testbed evaluation of virtual environment interaction techniques, 2001.

[4] Kil Y, Renzulli P, Kreylos O, Hamann B, Monno G, Staadt O. 3D warp brush: interactive free-form modeling on the responsive workbench. In: Proceedings of IEEE virtual reality 2005 (Poster); 2005.

[5] Shaw C, Green M. THRED: a two-handed design system. Multimedia Systems 1997;5(2):126–39.

[6] Wesche G, Droske M. Conceptual free-form styling on the responsive workbench. In: Proceedings of the ACM symposium on virtual reality software and technology. New York: ACM; 2000. p. 83–91.

[7] Wesche G, Seidel H. FreeDrawer: a free-form sketching system on the responsive workbench. In: Proceedings of the ACM symposium on virtual reality software and technology (VRST), Banff, Alberta, Canada, November 2001. p. 167–74.

[8] Matsumiya M, Takemura H, Yokoya N. An immersive modeling system for 3d free-form design using implicit surfaces. In: Proceedings of the ACM symposium on virtual reality software and technology. New York: ACM Press; 2000. p. 67–74.

[9] Schkolne S, Pruett M, Schröder P. Surface drawing: creating organic 3D shapes with the hand and tangible tools. In: Proceedings of the SIGCHI conference on human factors in computing systems, Seattle, WA, March 2001. p. 261–68.

[10] Bill JR, Lodha S. Computer sculpting of polygonal models using virtual tools. In: Technical Report UCSC-CRL-94-27, Baskin Center for Computer Engineering and Information Sciences, University of California, Santa Cruz, US, July 1994.

[11] Sederberg TW, Parry SR. Free-form deformation of solid geometric models. Computer graphics (Proceedings of SIGGRAPH 86), vol. 20, no. 4, 1986. p. 151–60.

[12] Singh K, Fiume EL. Wires: a geometric deformation technique. In: Proceedings of SIGGRAPH 98, Computer graphics proceedings, annual conference series, 1998. p. 405–14.

[13] Kobayashi K, Ootsubo K. t-FFD: free-form deformation by using triangular mesh. In: Proceedings of symposium on solid modeling and applications; 2003. p. 226–34.

[14] Botsch M, Kobbelt L. An intuitive framework for real-time freeform modeling. In: Proceedings of ACM SIGGRAPH 2004. New York: ACM Press; 2004.

[15] Frisken S, Perry R, Rockwood A, Jones T. Adaptively sampled distance fields: a general representation of shape for computer graphics. In: Proceedings of SIGGRAPH; 2000. p. 249–54.

[16] Markosian L, Cohen JM, Crulli T, Hughes JF. Skin: a constructive approach to modeling free-form shapes. In: Siggraph 1999, computer graphics proceedings, Los Angeles, US; 1999. p. 393–400.

[17] Llamas I, Kim B, Gargus J, Rossignac J, Shaw C. Twister: a space-warp operator for the two-handed editing of 3D shapes. ACM Transactions on Graphics 2003;22(3):663–8.

[18] Parent RE. A system for sculpting 3-D data. Computer graphics (Proceedings of SIGGRAPH 77), vol. 11, no. 2, July 1977. p. 138–47.

[19] Perry R, Frisken S. Kizamu: a system for sculpting digital characters. In: Proceedings of the 28th annual conference on computer graphics and interactive techniques, Los Angeles, CA, August 2001. p. 47–56.

[20] Angelidis A, Wyvill G, Cani M-P. Sweepers: swept user-defined tools for modeling by deformation. In: Proceedings of shape modeling applications; 2004. p. 63–73.

[21] Mauch S. Efficient algorithms for solving static Hamilton-Jacobi equations. PhD thesis, California Institute of Technology, Pasadena, CA; 2003.

[22] Guéziec A. 'meshsweeper': dynamic point-to-polygonal-mesh distance and applications. IEEE Transactions on Visualization and Computer Graphics 2001;7(1):47–61.

[23] Joy KI, Legakis J, MacCracken R. Data structures for multiresolution representation of unstructured meshes. In: Farin G, Hagen H, Hamann B, editors. Hierarchical approximation and geometric methods for scientific visualization. Heidelberg, Germany: Springer-Verlag; 2002. p. 143–70.

[24] Duchaineau M, Wolinskey M, Sigeti DE, Miller MC, Aldrich C, Mineev-Weinstein MB. ROAMing terrain: real-time optimally adapting meshes. In: Proceedings of IEEE visualization '97. Silver spring, MD: IEEE Computer Society; 1997. p. 81–8.