The Visual Computer manuscript No.
(will be inserted by the editor)

Andrei Sharf · Marina Blumenkrants · Ariel Shamir · Daniel Cohen-Or

SnapPaste: An Interactive Technique for Easy Mesh Composition

Abstract Editing and manipulation of existing 3D geometric objects are means to extend their repertoire and promote their availability. Traditionally, tools to compose or manipulate objects defined by 3D meshes are in the realm of artists and experts. In this paper, we introduce a simple and effective user interface for easy composition of 3D mesh-parts for non-professionals. Our technique borrows from the cut-and-paste paradigm where a user can cut parts out of existing objects and paste them onto others to create new designs. To assist the user attach objects to each other in a quick and simple manner, many applications in computer graphics support the notion of "snapping". Similarly, our tool allows the user to loosely drag one mesh part onto another with an overlap, and lets the system snap them together in a graceful manner. Snapping is accomplished using our Soft-ICP algorithm which replaces the global transformation in the ICP algorithm with a set of point-wise locally supported transformations. The technique enhances registration with a set of rigid to elastic transformations that account for simultaneous global positioning and local blending of the objects. For completeness of our framework, we present an additional simple mesh-cutting tool, adapting the graph-cut algorithm to meshes.

Keywords Interactive Tools · User-Interface · Cut-and-Paste · Snapping · Meshes

A. Sharf
Tel Aviv University
Tel.: +972-3-6405360
E-mail: asharf@tau.ac.il

M. Blumenkrants
The Interdisciplinary Center Herzliya

A. Shamir
The Interdisciplinary Center Herzliya

D. Cohen-Or
Tel Aviv University

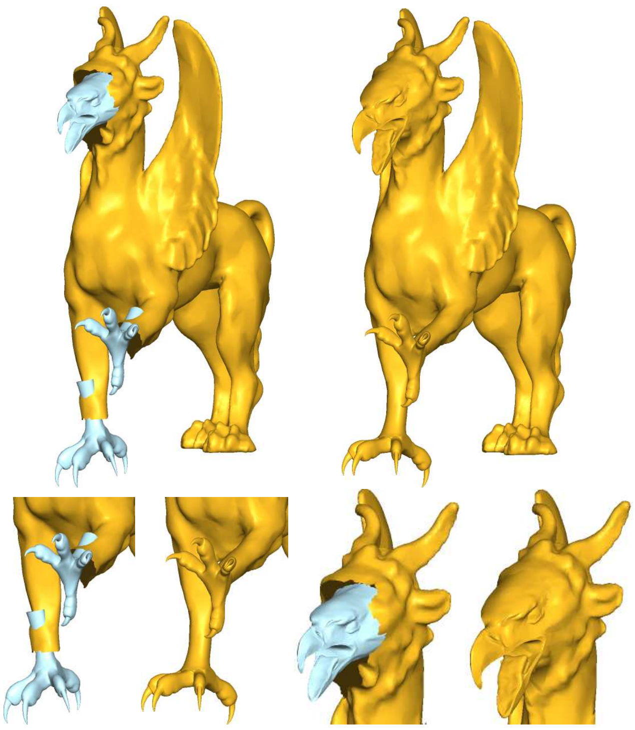
Fig. 1 Snapping allows easy pasting of mesh parts by loosely positioning them close to their target, and letting the system snap them together.

1 Introduction

The creation of digital geometric objects for graphics, engineering, games or motion pictures is a difficult and often expensive process. Despite recent impressive progress in automatic modeling and reconstruction, a fair amount of geometric processing is always done manually. Experts use modeling software for the design and creation of 3D models. However, such software require meticulous work, and must be driven by a professional skilled artist or engineer to produce correct and visually pleasing results. Recently, a growing number of works have been targeted at simplifying the interaction and manual manipulation of 3D objects to enable simple and interactive editing by non-professionals [15, 13, 14, 22].

In this work we focus on a user interface operation borrowed from computer editing applications – the *cut-and-paste* operation. In the context of direct manipulation, cut-and-paste is used as a metaphor for cutting and connecting mesh parts and surfaces. However, in current mesh editing application this can become a complicated procedure. The user is required to carefully cut a designated surface-part, place it near the other surface in an exact position, define the correspondence between boundary parts, stitch them together, and possibly add or remove outliers manually. These operations may need to be repeated several times to generate a natural merge between the two surfaces. Our goal is to define an easy-to-use tool which supports cut-and-paste of surface meshes as a simple, natural and intuitive operation.

To support simple cutting, we present a simple mesh cutting tool based on a graph-cut procedure. This tool enables the user to separate parts from a mesh simply by drawing a cutting stroke even from a single viewpoint. The stroke path is projected on the mesh and automatically closed to form a loop. This loop is used to define the approximated area of the cut. The final cut is created using a feature-sensitive graph-cut algorithm.

To support simple pasting, we turn to a very useful and effective notion in graphics applications - the notion of *snapping*. Snapping is often used to alleviate the user's need to exactly position shapes before connecting them. Using snapping, shapes are automatically attracted to specific locations (grid) or to other shapes. By extending this notion to 3D surfaces, our tool snaps two mesh parts together with a local graceful warp that respects their initial relative position. Our snapping tool allows even little kids to connect 3D mesh parts easily (Figure 2): the user chooses a mesh part, loosely drags and rotates it to an approximate position close to the target mesh and releases. The part automatically *snaps* to the target mesh. There is no need for precise positioning of parts in 3D, no need to delicately separate mesh parts and define exact boundaries and no need to set or tune any parameters. Snapping merges gracefully any two valid mesh parts with boundaries, as long as they have an overlapping region. We refer to such overlapping region as the *snapping region*.

In effect, snapping involves two parts: relative global positioning and local blending of the two surface meshes. We use point-wise local rigid transformations with varying support to achieve both tasks. We base our technique on the classic iterative closest point (ICP) algorithm which supports rigid body transformations, and extend it toward a more general transformation by defining a novel *soft-ICP* algorithm. While the original ICP algorithm searches for a global transformation to register two point sets, we use a series of global-to-local transformations with differing supports for positioning and blending the two objects.

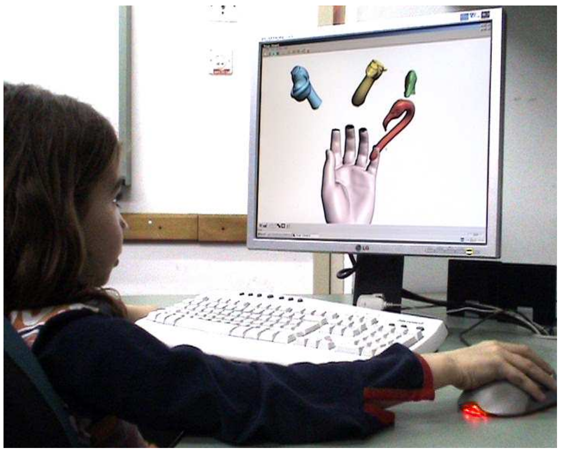
Fig. 2 Mesh-snapping allows even eight years old children to compose 3D mesh parts easily.

The support neighborhood size, defining the local transformation of each point, changes depending both on the position of the point on the surface and on time (i.e. the iteration number). Consequently, the transition between global to local, thus from rigid to soft is defined in a gradual manner across the surface and through the iterations. Points which are far from the snapping region undergo a more rigid displacement common to larger neighborhoods. Points which are closer to the snapping region move with locally rigid transformations that together form a soft-warp of the surface at the region where the two surfaces overlap. This series of transformations creates a smooth transition that scales, aligns and merges the two surfaces together.

Once the two surfaces are snap together, the overlapping region connecting the two parts is re-meshed based on [25] to create one valid mesh. Examples of snapping of general cylindrical shapes can be seen in Figure 1, where the *Feline* is transformed into a *Griffin* by loosely pasting parts of an eagle's head and legs. Disk-like shapes are seen in Figure 9, where the sphinx face is replaced. More examples can be found in the results section and in the video attached.

The main contributions of this paper are as follows:

- We extend the cut-and-paste notion to 3D meshes.
- We define a snapping method for two mesh parts to assist easy pasting and composition of 3D objects. The snapping solves both positioning and blending of the two surfaces by defining a novel soft-ICP algorithm.
- We define an easy mesh cutting tool based on graph cuts.

2 Background and Related Work

Cut-and-paste is a user-interface paradigm for transferring data pieces of any kind from a source to a destination. The cut-paste notion is derived from the traditional practice in manuscript editing in which paragraphs were literally cut from a page with scissors and physically pasted onto another page. In software system this paradigm has become so universal and general, that most users expect to find it in any editing and modeling application. Although the basic idea of extracting data from one part and inserting it back into another is simple, there is a need to define the meaning of cutting and pasting depending on the data and its representation. This is especially true when the data is as complex as 3D objects in mesh representation.

Constructive solid geometry (CSG) representation uses Boolean operations such as subtraction to support cutting and union to support pasting. Several works have extended these operations to boundary surfaces using different implicit representations. [29] use analytic primitives or skeletal implicit representation. Point-based representations and level-sets were used in [21], moving least square surfaces in [31], and partition of unity in [23]. The approach presented in [12] is based on voxelization, volumetric minimal cuts and stitching to compose meshes. Nevertheless, most available graphics objects are still defined using explicit meshes where boolean operations and cut-and-paste are more difficult. Moreover, the relative positioning of the two blended parts still remains a difficult task, regardless of the representation.

In [4] boolean operations on explicit multi-resolution surfaces are presented, but with no blending or smoothing between parts. Later, in [5] a cut-and-paste operator is defined mostly for details of implanting and patch embedding, rather than composing whole mesh parts. Other works on mesh editing deal with direct manipulation and mesh-deformation operators [18,2,1,26]. Nevertheless, in these works mesh composition usually needs delicate manual intervention. For example in [30] manual intervention is needed to achieve correspondence between boundaries. In [27], additional steps of registration, possibly local re-meshing and zipping (filling in gaps) are required. Similarly, in the system for modeling-by-example [11] the two major challenges in their composition, positioning and attachment, are dealt with by global rigid positioning using volumetric ICP, and stitching between two predefined conforming (corresponding) contours.

Other approaches are based on joint-parameterization of local regions on the source and target meshes. In [10], pasting of a source mesh (possibly with high genus) is performed by defining a joint base parameterization to the source and target regions, replacing the base mesh region in the target with the source base region and then transferring the details. In [16] a combined parameterization is built, when the correspondence is defined using ICP, and the pasting operation is implemented by blending the two meshes using morphing between the source and target on the target mesh.

For interactive mesh cutting, modeling applications either allow explicit specification of the cut position or the use of a lasso tool which needs specific view directions to succeed. Intelligent scissoring was presented in modeling-by-example [11] where a stroke is painted directly on the mesh. This stroke is used to define a neighborhood where the real cut is then completed to form a loop using a cost function and a variant of Dijkstra algorithm. Our cutting algorithm uses a similar method to close the loop between the two end-points of the initial stroke. However this is only done to define two separate regions on the mesh. The actual cut is defined using a graph cut between the two regions, adhering to surface features. Graph cuts have been used extensively on images for segmentation and feature extraction [6,20] and in [17] for automatic partitioning of meshes. A different intelligent scissoring tool is proposed in [19], where a 2D line is used to define a plane that cuts the mesh to create an initial loop. The position of this loop is then refined using a 3D snake. However, sketching a stroke directly on the mesh has more flexibility than a 2D plane, and advancing an active contour can be more complicated than finding a graph cut

Our work seeks to define a simpler and more effective interactive tool for easy cut-and-paste in the spirit of a smart and simple user interface for modeling. Such a tool must support easy mesh cutting and address both the positioning and the blending problems when pasting two surface parts together. We focus our efforts on the ease-of-use of such a tool, where snapping replaces the usual meticulous work of positioning and connecting two meshes, with an interactive technique for pasting.

The rest of the paper is composed as follows. In Section 3 we present an overview of our Soft-ICP algorithm for snapping, followed by Sections 4 to 6 which describe the Soft-ICP algorithm details. Section 7 describes our simple cutting tool and in Section 8 we present our data-structure. We show some results and conclude in Sections 9 and 10.

3 Soft-ICP Algorithm Overview

The definition of our easy pasting tool is based on the ability to automatically snap two mesh parts and compose a single one. Snapping involves both calculating the relative position (global alignment) of the composed parts and creating a blend (local deformation) between them. Both of these tasks are addressed by using a new soft-ICP algorithm (Figure 3). The goal of the original Iterative Closest Point (ICP) algorithm is to find a registration between two point sets [7,3]. The ICP algorithm proceeds in an iterative manner using the following steps:

1. Find a point-wise correspondence between the sets.
2. Compute and apply registration on the two sets.
3. Check error threshold, and return to first step if needed.

The correspondence and registration steps search for a rigid transformation, including translation and rotation, that reduces the sum-of-squared distances between point pairs. The many variants of the basic algorithm differ in the method for selecting and matching the points, the weighting of the corresponding pairs, the use of pair-rejection criteria, the error metric used and the minimization technique of the er-

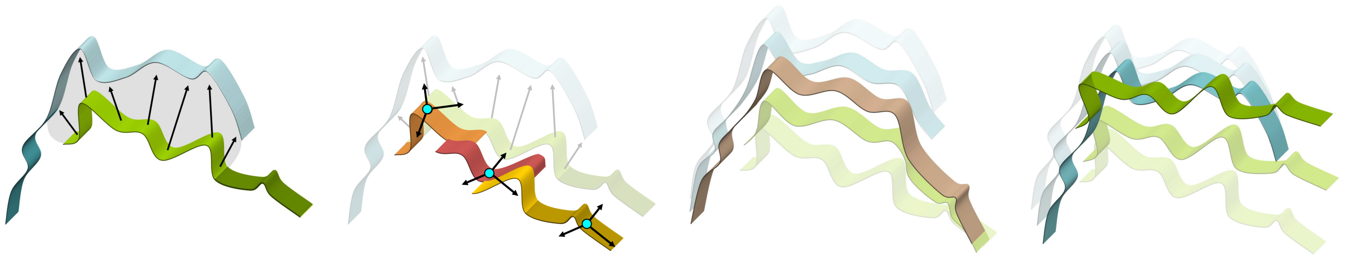

Fig. 3 The Soft-ICP algorithm: (a) Starting with one shape placed near another with an overlap, correspondence is computed in the *snapping region* (gray region). (b) Transformations are computed point-wise (example shows for 3 points), based on local supporting neighborhoods (colored stripes). (c) The algorithm iteratively computes correspondence and transformations on both shapes until they gracefully blend together. Note the elastic deformation inside the snapping region while details outside rigidly align. (d) For comparison we show the result of applying the original ICP algorithm on the two shapes.

ror metric [24]. Nevertheless, most methods still look for one global transformation that registers the two point sets. Our approach is similar to the methods introduced in [28, 9]. These methods use a local ICP algorithm for the registration of anatomical 3D data-sets. However, in these works two full surfaces are registered and not sub-parts, and static Euclidean neighborhoods are used. Using such algorithms for combining mesh parts will not solve both positioning and blending problems.

The Soft-ICP algorithm solves both global positioning and blending of the two objects simultaneously. The basic idea is that at each iteration, each point p_i of the mesh is transformed by its own individual transformation T_i. This transformation is computed using similar calculations as ICP, but on a local support around p_i denoted: $N(p_i)$ (Figure 3(b)). The overall effect of applying a different transformation to each point individually is, that the surface defined by the points can undergo non-rigid (elastic) deformations. Since neighboring points have overlapping local support, they undergo similar transformations, creating a smooth deformation across the surface ((Figure 3(b-c))). We utilize this type of deformation for snapping the two surfaces together.

The pseudocode of the soft-ICP algorithm is presented in Figure 4. Note that the algorithm is defined for snapping one mesh M_A to another M_B. In practice, we exchange the roles of M_A and M_B after each iteration, creating an almost symmetric snapping procedure.

```
Procedure: snap $M_A$ to $M_B$
Let $S_A$ and $S_B$ be the snapping regions
   of $M_A$ and $M_B$ respectively
Loop until convergence {
   Find correspondence $\phi$ of $S_A$ to $S_B$
   For each point $p_i$ in $M_A$ {
      Find the local neighborhood $N(p_i)$
      Calculate the transformation $T_i$
         based on $\phi|_{N(p_i)}$
   }
   For each point $p_i$ in $M_A$ {
      Apply $T_i$ to $p_i$
   }
}
```

Fig. 4 The Soft-ICP pseudocode.

4 Snapping Region and Correspondence

The Soft-ICP technique depends on the correspondence among points on both surfaces. To find correspondence, it is important that the surfaces overlap since the correspondence is defined in the overlapping region (Figure 3(a)). If there is too little or no overlap between the meshes, i.e. not enough correspondence, snapping is not performed. This in turn, will drive the user to continue dragging or rotating the mesh-parts until they overlap and snap.

To define a snapping region, we assume that the two meshes are valid manifolds and contain two designated boundary loops. Note that in the case of closed meshes with no boundaries an additional cutting step should be performed. Furthermore, for meshes with multiple boundaries, we take the closest pair (one from each mesh) to be the designated boundary loops.

For each mesh, we find the set of closest points to the other shape boundary loop. Next, we compute the geodesic distance for each point in the set to their own boundary loop. We define the snapping region size R as the maximum of the geodesic distances. Thus, the sub-mesh within R geodesic distance from the boundary loop is defined as the snapping region. Only points within the snapping regions of the two meshes are used for correspondence computation (see Figures 3(a) and 6).

Let S_A and S_B be the two snapping regions on the two meshes. For each point $p_a \in S_A$ the corresponding point $p_b \in S_B$ is defined as: $(\forall q \in S_B, D(p_a,p_b) < D(p_a,q))$. The distance $D(p_a,p_b)$ is a weighted sum of three terms:

$$w_1||p_a - p_b|| + w_2 \arccos(N_a \cdot N_b) + w_3(c_a - c_b)$$

The first term is the Euclidean distance between the points. The second term represents the angle difference of the nor-

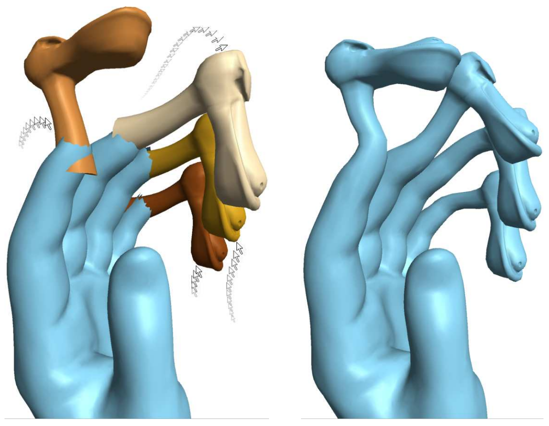

Fig. 5 Differences in the speed of the pasting gesture: colors of the heads denote the dragging speed, the darker the color, the more slowly the positioning is. The snapping process has more freedom in relative positioning as the pasting is faster. Dragging the shapes slowly causes the shapes to merge with little change in the global position.

mals N_a and N_b at the points, and the third is the difference in the local Gaussian curvatures c_a, c_b at the points. The weights of the three terms are user defined and in practice we use $w_1 = 0.6, w_2 = 0.2, w_3 = 0.2$. This means that only point-pairs which are both close and similar in local shape features are matched.

5 Supporting Neighborhoods

The transformation T_i of each point p_i is computed based on a local geodesic neighborhood around p_i. We define the geodesic neighborhood of p_i: $N_r(p_i)$, as the set of mesh vertices that are within r geodesic distance from p_i. Thus, for computation of T_i, we consider only the local correspondence ϕ that is inside the geodesic neighborhood N_r of p_i: $\phi|_{N_r(p_i)}$ (Figure 3(b)).

These supporting neighborhoods $N_r(p_i)$ change their size r depending on the distance of the points from the snapping region and on the iteration. For each point p_i we start from a maximal neighborhood that includes all ϕ. As the iterations progress, the neighborhood of the point is reduced as a function of both iteration t and the proximity of the point to the snapping region (its boundary loop). For a point p_i the size r of $N_r(p_i)$ is calculated as follows:

$$r_i = R \cdot e^{-[(t \cdot k/d_i)^2]}$$

Where R is the snapping region size, t is the iteration ($1 \leq t \leq t_{max}$) and d_i is the geodesic distance of p_i to the snapping region (its boundary loop). Note that for points outside the snapping region, we additionally offset r_i with their distance to the snapping region. Thus, the supporting neighborhood is always contained (fully or partially) inside the snapping region (in Figure 3(b) the yellow stripe).

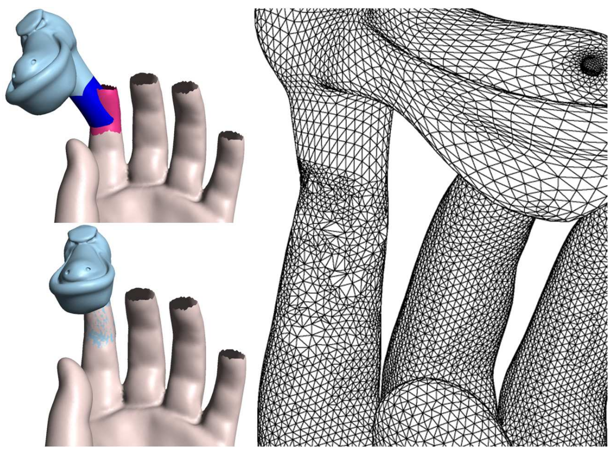

Fig. 6 Snapping region (left-top before and left-bottom after snapping), is remeshed (right) while constraining its boundaries to conform to connectivity of each mesh, creating one valid mesh after the snap.

Points which are far from the snapping region have larger neighborhoods, hence their transformation is mostly a global rigid one. The closer the point is to the snapping region, the smaller its neighborhood, thus resulting in more locally-rigid transformation. The displacements of the meshes are a series of rigid-to-elastic transformations which depend on time (iterations) and the geodesic distance to the snapping region (Figure 3(c)).

The elasticity constant k ($1 \leq k \leq k_{max}$) governs the trade-off between movement and accuracy. It is used to distinguish between slow pasting gestures, when high accuracy is needed ($k = k_{max}$) and fast pasting gestures, when large movement is allowed ($k = 1$). The interactive snapping process has more freedom in the relative positioning (rigid alignment) as the interactive pasting is faster. Aligning the shapes slowly causes the shapes to merge with little change in the global position, as indicated by the brown heads in Figure 5.

6 Transformation Calculation

Once correspondence ϕ and local neighborhoods $N_r(p_i)$ are defined, we compute the transformation $T_i = (\mathscr{R}, \mathscr{T}, \mathscr{S})$ based on $\phi|_{N_r(p_i)}$; First, scale $\mathscr{S}$ is found using the oriented bounding boxes of $N_r(p_i)$ and its corresponding points. Translation $\mathscr{T}$ among the two point sets is found using center-of-mass alignment. Finally rotation $\mathscr{R}$ is determined using standard SVD minimization:

$$\sum_{<p_a,p_b> \in \phi|_{N_r(p_a)}} ||(\mathscr{R}(p_a) + \mathscr{T}) - p_b||^2$$

To create a softer transition and find a better fit between the surfaces we apply the soft-ICP transformations in a gradual manner. In the first iterations we scale, translate and rotate

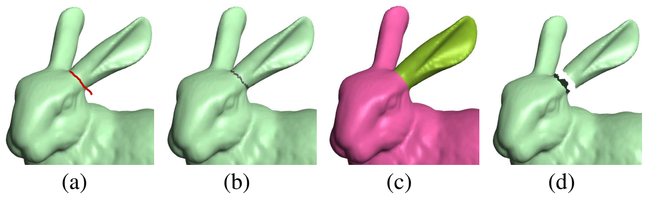

Fig. 7 Cutting a mesh: (a) The user marks a path on the mesh (b) The path is automatically closed (c) The min-cut separates the mesh into two parts (d) The mesh is cut.

(using quaternions) only a portion (l_i) of the way. As iterations progress (and the surfaces get closer), this portion increases locally until we use the full transformation:

$$l_i = t/t_{max}$$

The snapping algorithm results in the two shape meshes fully overlayed in the snapping regions. To create a valid mesh, a post processing step of local re-meshing is required for the two overlayed meshes. We used the re-meshing algorithm with boundary constraints from [25] (Figure 6). Nevertheless, any local re-meshing algorithm can be used to re-triangulate the snapping regions connecting the two meshes.

7 Mesh Cutting

To complete our cut-and-paste framework for 3D meshes we have developed a simple and effective cutting tool. The main idea behind our approach, is to assist the user to create correct and effective cuts while still preserving a simple stroke-based natural user interface.

Our mesh cutting algorithm has three major parts (Figure 7): 1. Loop completion, 2. Graph cut, and 3. Cut smoothing. To start a new cut, the user “paints” a stroke near the desired position of the cut. The positions of the mouse are projected onto the surface and a set of faces are created to represent the stroke.

Using the longest component of consecutive faces of the stroke denoted by S, the first step is to complete a whole loop from its two end-faces. We use Dijkstra shortest path algorithm using the following cost function between neighboring faces f_i and f_j:

$$c(f_i, f_j) = g(f_i, f_j) + ID(f_j, S) + ND(n_j, S)$$

Where $g(f_i, f_j)$ is the geodesic distance between f_i and f_b.

$ID(f_j, S)$ is the inverse geodesic distance from $f_j \notin S$ to the stroke S:

$$ID(f_j, S) = \sum_{f \in S} \frac{1}{g(f, f_j)}$$

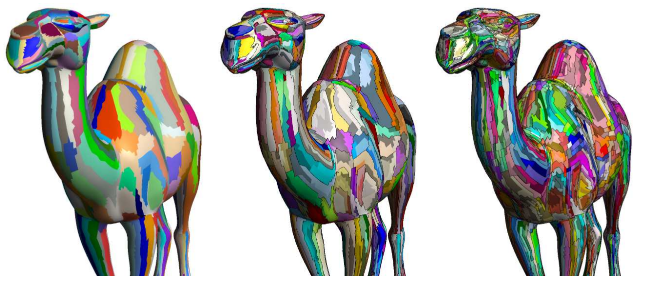

Fig. 8 Three levels of the GeoTree anisotropic patch hierarchy on the camel.

The inverse distance favors faces that are distant from the stroke.

$ND(f_j, S)$ is the normal distance from a face $f_j \notin S$ with normal n_j to the stroke S. We compute the normal cone of all faces in S denoted by its central vector v and opening angle α.

$$ND(f_j, S) = \begin{cases} 1 & \text{if } n_j \cdot v \geq \cos(\alpha) \\ \frac{n_j \cdot v + 1}{\cos(\alpha) + 1} & \text{otherwise} \end{cases}$$

The normal distance favors faces whose normal is generally facing in the other direction from the stroke.

The computed loop is used to designate two distinct mesh sides and a “fuzzy region”, inside which the real cut will pass. Thus, we extend the initial loop of faces to a width of n triangles to both sides (n is the user defined stroke width). The faces at the boundaries of the fuzzy region in each side are marked as source and sink respectively.

Next, a weighted graph on the faces is built to represent the fuzzy region. The edge weight to the source and sink is the geodesic distance to them, and the edge weight between adjacent faces is the geodesic and normal difference of these faces. Finding a minimum cut in this graph [6] separates the mesh into two parts. Finally, before cutting we apply a simple smoothing step which locally switches the membership of triangles that create sharp angle edges in the cut.

8 The GeoTree Data Structure

Both cut and paste operations use queries that need to retrieve geodesic neighborhoods on the mesh. In a naive implementation, finding the geodesic neighborhood of a vertex can take an order of the number of vertices. Using many such queries may lead to quadratic algorithm complexity. For a more efficient implementation, we define a hierarchical search structure, which we call the *GeoTree*, supporting fast neighborhood retrieval given a geodesic radius.

The GeoTree is a hierarchical patch structure built on the mesh. To create the coarsest level of the hierarchy we use a variant of Lloyd algorithm (e.g., k-means) similar to [8] to

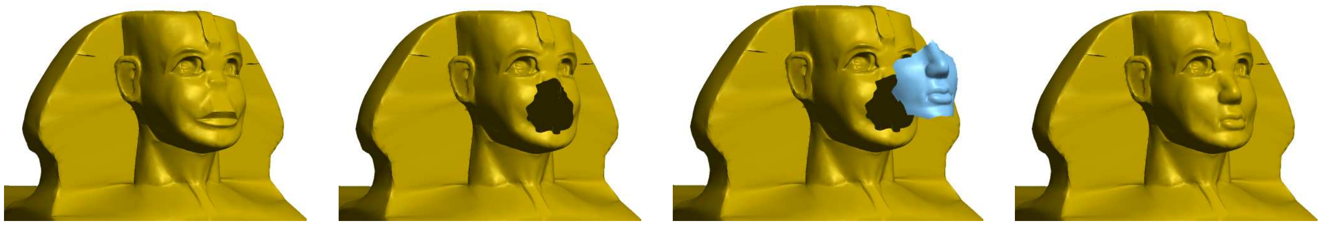

Fig. 9 Sphinx's face lift. A part of the Sphinx's face is loosely cut out and replaced (left). Snapping aligns and merges the replacing part in a natural manner (right).

partition the mesh into k anisotropic patches of similar size. This creates *anisotropic* feature-sensitive neighborhoods which are more coherent to surface attributes. As will be shown, both cutting and snapping gain from this feature.

To build a full matrix of geodesic distances between the patches in the coarsest level we compute all-pairs shortest paths between the centers of the patches. Each patch is then partitioned recursively to four, creating a quadtree structure on the surface (Figure 8). At each level, we compute for each node (i.e., each patch) the distances to the m closest patches only, and store them in the node.

The geodesic neighborhood of a point p is approximated by the union of patches collected from the GeoTree. Given a radius r, and a point p, we find the leaf node in the GeoTree that contains p. If the radius r is smaller than the m-furthest patch distance, we collect all patches whose distance is smaller than r to compose the geodesic neighborhood of p. If the radius r is larger than the m-furthest patch, we go up one level in the hierarchy and repeat the process. At topmost level, we use the full matrix of the patch distances.

We utilize our GeoTree hierarchy to accelerate the cutting process. Instead of using all triangles of the mesh we use an adaptive cut in the GeoTree. Far from the fuzzy region we use the coarse levels of the GeoTree and build the weighted graph using GeoTree patches. As we get closer to the stroke we use finer levels of the GeoTree patches and finally very near and inside it we use the triangles.

In the Soft-ICP algorithm, in each iteration we calculate ϕ once, but for each point we need to find $\phi|_{N_r}$ (Section 5). Using the GeoTree, this neighborhood search is supported in an efficient manner. Furthermore, since anisotropic neighborhoods are more coherent to surface attributes, $\phi|_{N_r}$ has a bias towards similar features matchings-pairs.

9 Results

We have applied our snapping algorithm to various types of 3D objects including scanned meshes, CAD and artificial meshes. We used a Pentium-4, 2.4 GHz, 1G memory with Nvidia GeForce-4800 card. Results show high visual quality with an interactive rate of not more than a few seconds per operation (see Table 1, and the supplied video).

Model	Number faces	Number vertices	Snapping time
hand+camel	96,394	41,856	3.5 sec
hand+flamingo	79,498	37,954	2.7 sec
hand+duck	71,594	35,956	2.4 sec
hand+dinopet	67,870	34,082	2.2 sec

Table 1 Timing results of snap-paste for the puppet show example. The differences depend on the snapping region size and number of soft-ICP iterations. Other pasting examples have similar timing of order of only few seconds.

For very large meshes (millions of triangles), we have developed two efficient accelerations: The first, uses GeoTree patches instead of triangles far from the snapping region. The second, restricts the Soft-ICP computation within a constant distance from the snapping region. Mesh parts that are further away from this distance will transform using one global rigid transformation.

Due to our anisotropic GeoTree structure (Section 8), our algorithm works in a feature sensitive manner, guaranteeing that the shapes are snapped along similar features. Figure 10 shows the snapping of the *Gargoyle* to the bow of a ship. While the squared like base of the *Gargoyle* is matched and snapped to the bow, the features of the models itself undergo only little deformation due to nearly global transformations.

Figure 11 shows some results of snapping multiple parts together to create new diverse models. In the creation process, no fine cutting or exact positioning of the shapes is required. The objects were loosely dragged in proximity to one another and released, and the snapping process took only a few seconds each time.

Our technique can also handle complex topological shapes and snap multiple boundaries at once. For example, in Figure 12, we compose the tail of the *Feline*, which contains a hole and two boundaries, to its neck. This simultaneous snapping can only be achieved by a technique that treats the surface locally as a soft body and does not require exact positioning. The reason for this is that it is unlikely that the two boundaries can be fit simultaneously.

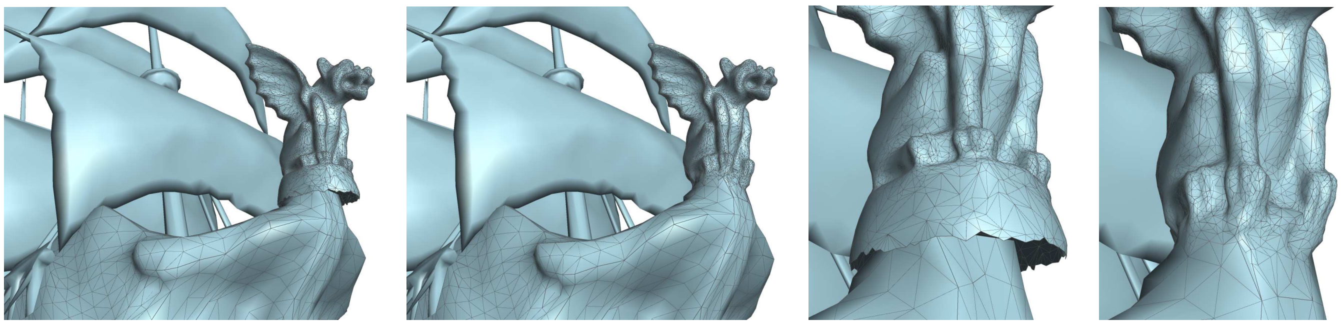

Fig. 10 Snapping the *Gargoyle* to the bow of a ship.

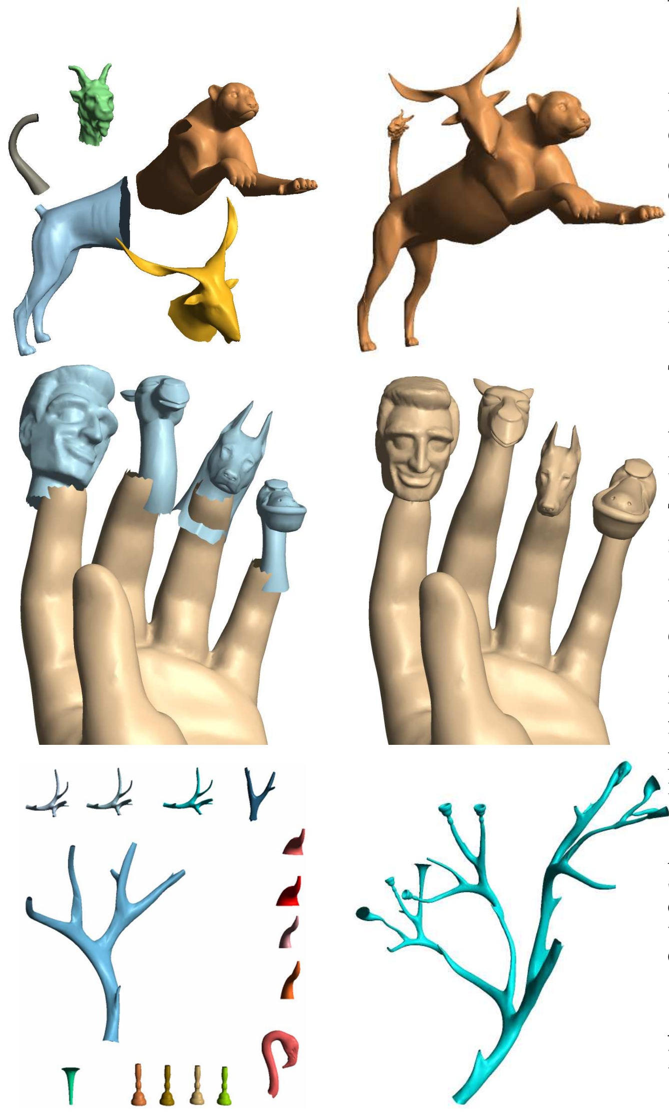

Fig. 11 Snapping multiple parts together creates a repertoire of new models in a matter of seconds. The mythological Chimera, a puppet show and a tree created of various parts.

10 Conclusions

In this paper we introduced a cut-and-paste tool which is designed for non-professionals. In our framework the user does not need to cut nor to position the parts precisely to compose them. Instead, the system automatically snaps together parts which are overlapping in a graceful manner. We presented numerous results obtained using these tools with very little user effort. The core of our tool is a snapping technique based on a soft-ICP algorithm. Soft-ICP consists of a series of point-wise locally supported rigid transformations. The local support varies both in time and in space, defining a series of global to local, and a rigid to elastic transformation that simultaneously solve both the global positioning and the local blending problems.

There are several possible extensions to this work. The definition of a snapping operation when no overlap exists is a possible extension. Another, is the definition of an operation semantics when one or both of the snapped parts do not contain boundary loops. We also believe that the soft-ICP algorithm can be beneficial for other related applications. This includes hole-filling, repairing polygonal models, and surface registrations. Additionally, we would like to continue pursuing easy-to-use tools for interactive modeling, to facilitate modeling for both professional and amateurs.

Acknowledgements We would like to thank Carlos E. Scheidegger, Shahar Fleishman and Claudio Silva for providing us the code for local triangulation with boundary constraints in our remeshing step. This work was supported partly by a grant from the Israeli Ministry of Science.

References

1. Angelidis, A., Wyvill, G., Cani, M.P.: Sweepers: Swept user-defined tools for modeling by deformation. In: Shape Modeling International, pp. 63–73 (2004)
2. Bendels, G.H., Klein, R.: Mesh forging: editing of 3d-meshes using implicitly defined occluders. In: Proceedings on ACM Symposium on Geometry Processing, pp. 207–217 (2003)
3. Besl, P.J., McKay, N.D.: A method for registration of 3-d shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence **14**(2), 239–256 (1992)

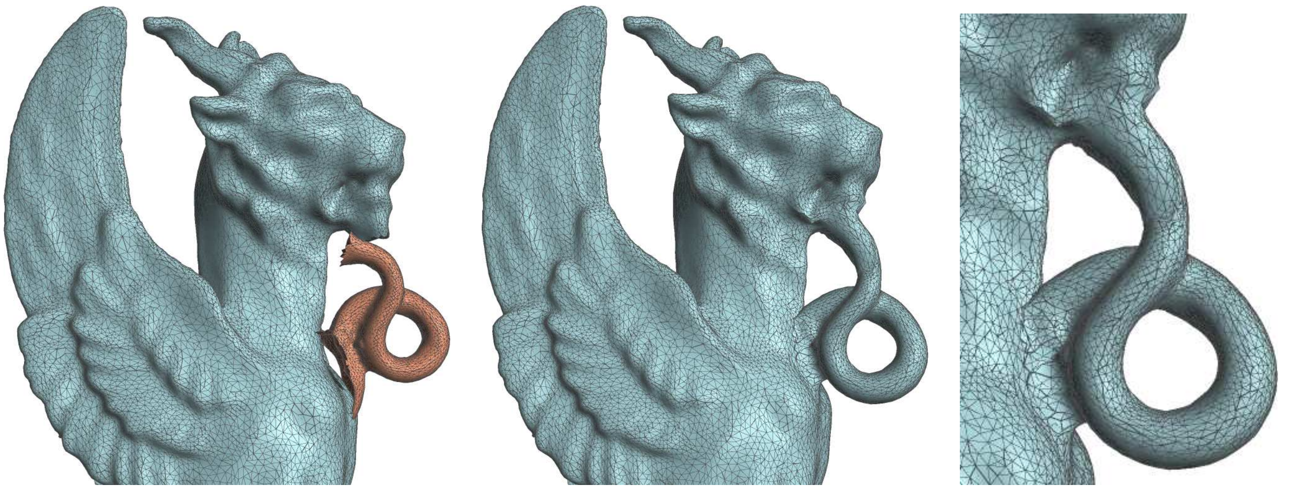

Fig. 12 Snapping multiple boundaries simultaneously can only be achieved by a technique that treats the surface locally as a soft body and does not require exact positioning.

4. Biermann, H., Kristjansson, D., Zorin, D.: Approximate boolean operations on free-form solids. In: Proceedings of ACM SIGGRAPH 2001, pp. 185–194 (2001)
5. Biermann, H., Martin, I., Bernardini, F., Zorin, D.: Cut-and-paste editing of multiresolution surfaces. In: Proceedings of ACM SIGGRAPH 2002, pp. 312–321 (2002)
6. Boykov, Y., Jolly, M.: Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. In: International Conference on Computer Vision (ICCV), pp. 105–112 (2001)
7. Chen, Y., Medioni, G.: Object modelling by registration of multiple range images. Image and Vision Computing **10**(3), 145–155 (1992)
8. Cohen-Steiner, D., Alliez, P., Desbrun, M.: Variational shape approximation. ACM Transactions on Graphics pp. 905–914 (2004)
9. Feldmar, J., Ayache, N.: Rigid, affine and locally affine registration of free-form surfaces. The International Journal of Computer Vision **18** (1996)
10. Fu, H., Tai, C.L., Zhang, H.: Topology-free cut-and-paste editing over meshes. In: Proceedings of the 3rd International Conference on Geometric Modeling and Processing (2004)
11. Funkhouser, T., Kazhdan, M., Shilane, P., Min, P., Kiefer, W., Tal, A., Rusinkiewicz, S., Dobkin, D.: Modeling by example. ACM Transactions on Graphics (SIGGRAPH 2004) pp. 652–663 (2004)
12. Hassner, T., Zelnik-Manor, L., Leifman, G., Basri, R.: Minimal-cut model composition. In: International Conference on Shape Modeling and Applications (SMI' 05), pp. 72–81 (2005)
13. Igarashi, T., Matsuoka, S., Tanaka, H.: Teddy: a sketching interface for 3d freeform design. In: Proceedings of ACM SIGGRAPH, pp. 409–416 (1999)
14. Igarashi, T., Moscovich, T., Hughes, J.F.: As-rigid-as-possible shape manipulation. ACM Trans. Graph. **24**(3), 1134–1141 (2005)
15. James, D.L., Pai, D.K.: Artdefo: accurate real time deformable objects. In: Proceedings of ACM SIGGRAPH, pp. 65–72 (1999)
16. Kanai, T., Suzuki, H., Mitani, J., Kimura, F.: Interactive mesh fusion based on local 3d metamorphosis. In: Graphics Interface, pp. 148–156 (1999)
17. Katz, S., Tal, A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. ACM Transactions on Graphics (Proceedings SIGGRAPH 2003) **22**(3), 954–961 (2003)
18. Kobbelt, L., Campagna, S., Vorsatz, J., Seidel, H.P.: Interactive multi-resolution modeling on arbitrary meshes. In: proceedings ACM SIGGRAPH 98, pp. 105–114 (1998)
19. Lee, Y., Lee, S., Shamir, A., Cohen-Or, D., Seidel, H.P.: Mesh scissoring with minima rule and part salience. Computer Aided Geometric Design **22**(5), 444–465 (2005)
20. Li, Y., Sun, J., Tang, C.K., Shum, H.Y.: Lazy snapping. ACM Trans. Graph. **23**(3), 303–308 (2004)
21. Museth, K., Breen, D.E., Whitaker, R.T., Barr, A.H.: Level set surface editing operators. In: Proceedings of ACM SIGGRAPH 2002, pp. 330–338 (2002)
22. Nealen, A., Sorkine, O., Alexa, M., Cohen-Or, D.: A sketch-based interface for detail-preserving mesh editing. ACM Trans. Graph. **24**(3), 1142–1147 (2005)
23. Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., Seidel, H.P.: Multi-level partition of unity implicits. ACM Transaction on Graphics **22**(3), 463–470 (2003)
24. Rusinkiewicz, S., Levoy, M.: Efficient variants of the icp algorithm. In: Third International Conference on 3D Digital Imaging and Modeling (3DIM) (2001)
25. Scheidegger, C., Fleishman, S., Silva, C.: Triangulating point set surfaces with bounded error. In: Eurographics Symposium on Geometry processing, pp. 63–72 (2005)
26. Singh, K., Fiume, E.: Wires: a geometric deformation technique. In: Proceedings of SIGGRAPH, pp. 405–414 (1998)
27. Sorkine, O., Lipman, Y., Cohen-Or, D., Alexa, M., Rössl, C., Seidel, H.P.: Laplacian surface editing. In: Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing, pp. 179–188 (2004)
28. Thirion, J.P.: Fast non-rigid matching of 3d medical images. In: Proceedings of the Conference on Medical Robotics and Computer Assisted Surgery (MRCAS'95) (1995)
29. Wyvill, B., Galin, E., Guy, A.: Extending the csg tree. warping, blending and boolean operations in an implicit surface modeling system. Computer Graphics Forum **18**(2), 149–158 (1999)
30. Yu, Y., Zhou, K., Xu, D., Shi, X., Bao, H., Guo, B., Shum, H.Y.: Mesh editing with poisson-based gradient field manipulation. ACM Trans. Graph. **23**(3), 644–651 (2004)
31. Zwicker, M., Pauly, M., Knoll, O., Gross, M.: Pointshop 3d: an interactive system for point-based surface editing. In: Proceedings of SIGGRAPH, pp. 322–329 (2002)