Minimal-Cut Model Composition

T. Hassner[1] L. Zelnik-Manor[2] G. Leifman[3] R. Basri[1]

1. Dept. of Computer Science & Applied Math
The Weizmann Institute of Science
Rehovot, 76100 Israel

2. Faculty of Electrical Engineering
California Institute of Technology
Pasadena CA, 91125 USA

3. Faculty of Electrical Engineering
Technion - Israel Institute of Technology
Haifa 32000, Israel

Abstract

Constructing new, complex models is often done by reusing parts of existing models, typically by applying a sequence of segmentation, alignment and composition operations. Segmentation, either manual or automatic, is rarely adequate for this task, since it is applied to each model independently, leaving it to the user to trim the models and determine where to connect them. In this paper we propose a new composition tool. Our tool obtains as input two models, aligned either manually or automatically, and a small set of constraints indicating which portions of the two models should be preserved in the final output. It then automatically negotiates the best location to connect the models, trimming and stitching them as required to produce a seamless result. We offer a method based on the graph theoretic minimal cut as a means of implementing this new tool. We describe a system intended for both expert and novice users, allowing easy and flexible control over the composition result. In addition, we show our method to be well suited for a variety of model processing applications such as model repair, hole filling, and piecewise rigid deformations.

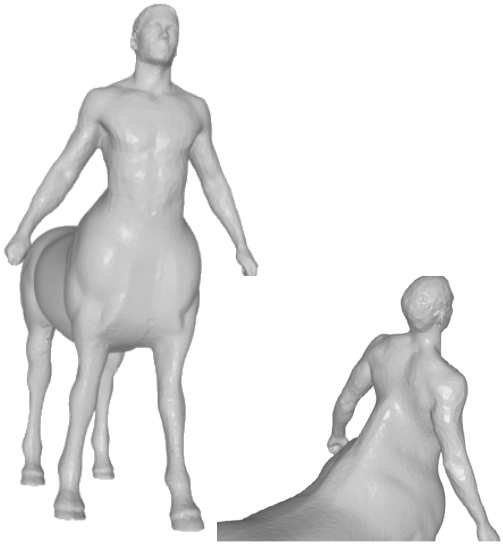

Figure 1. Centaur. An output of our model composition framework.

1. Introduction

Computer graphics systems are becoming ubiquitous, posing a growing demand for both realistic and fictitious 3D models. Constructing new models from scratch is a tedious process, requiring either a careful scan of real objects or the artistic skills of trained graphics experts. This process can potentially be enhanced, as more and more models become available, by reusing parts of existing models. With current methods, however, the process of composing new models from existing ones is still laborious, requiring a user to manually segment the input models, align them, and determine where to connect their parts. Automatic segmentation tools exist, but they are largely inadequate for this task. Segmentation tools are applied to each of the input models independently, and so they often produce results that require the user to further trim the parts to eliminate undesired protrusions or to significantly extend the parts so that they can be connected properly. In this paper we present a new, model composition tool. This tool, which is intended for use by both novice and expert modelers, automates much of the manual labor often associated with creating complex models. We further show the new operation to be particularly suitable for an assortment of model processing applications.

To illustrate our motivation consider for example the centaur in Fig. 1. Using our system, creating such a model from those of a horse and a man (Fig. 2) is as easy as (1) stating how the two models are to be positioned with respect to each other and (2) indicating to the system that the man's head and the horse's legs must be included in the final composition. The computer then automatically negotiates the best location for cutting and stitching the two models, in order to produce a seamless result. With existing methods, creating such a model is rarely that easy. Cur-

Proceedings of the International Conference on Shape Modeling and Applications (SMI'05)
0-7695-2379-X/05 $20.00 © 2005 **IEEE**

rent methods can produce excellent model composition results by Boolean operators, mesh stitching, morphing and more. However, none of these methods addresses the question where would be the best place to connect two models, under user specified constraints, without prior trimming (e.g., [12]). This is particularly important when the two models overlap and can connect in many places.

In designing our tool, we are faced with the following challenges: 1) Creating an intuitive interface to aid the user in positioning the models and influencing the appearance of the final composition. 2) Automatically finding the best place to connect the models, under the user given constraints, such that the result is smooth. 3) Keeping the design fast and simple, with processing kept at interactive speeds.

At the heart of our system is the graph theoretic minimal-cut operation [7]. This operation has been used in the past for both image [2, 17] and video [17] composition, and 3D model segmentation [16]. In fact, our work is inspired by [17]. Here we apply the minimal-cut operation to the task of 3D model composition. We present a system providing the user with an easy interface for defining how the output should be composed. Then, a graph representing local differences between the two models is automatically created. The minimal cut in this graph indicates the location where the two models are closest and most similar. Clipping the two models at this location and then stitching the obtained parts across the cut produces the final result.

Our work claims the following contributions: First, we describe a new model composition tool: Finding the best place to clip and connect overlapping models under user specified constraints. Second, we describe a novel algorithm based on graph cuts, as a means of implementing the new tool. Next, we detail a system designed for easy manipulation of this tool. Lastly, we show novel solutions to existing applications, based on our proposed framework. These applications include model restoration (Fig. 5), hole filling (Fig. 6), and rigid model deformations (Fig. 7).

The rest of this document is organized as follows. We review related work in Section 2. We then give an overview of our system in Section 3 and provide a detailed description of its components in Section 4. Having defined the new composition framework we show in Section 5 how it can be applied to other problems besides model composition. Implementation issues and a summary of results are given in Section 6. We conclude in Section 7.

2. Related work

A framework for constructing new models by reusing parts of existing models has been recently proposed in [12]. In their "Modeling by Example" system, new models are created by cutting each input model separately, using semi-automatic segmentation tools (i.e., "Intelligent scissors"), then stitching them to form the composition result. As models are cut independently of each other, large gaps often separate their borders. These gaps must then be interpolated by the stitching algorithm. Our system, on the other hand, clips models against each other, leaving only the smallest of gaps between their borders, allowing them to be connected with little or no visible artifacts. In addition, our interface does not rely on prior segmentation of the models (although we do provide semi-automatic model alignment which can use prior segmentation when available) and is simple enough to allow for automation, to a large extent, for certain applications (see Section 5).

In Constructive Solid Geometry (CSG) model primitives are combined using Boolean operations including union, intersection and subtraction (see overview in [13]). Such operations were adapted for numerous model representations [1, 5, 18, 19]. Although very intuitive, our proposed tool cannot generally be specified as a sequence of Boolean operations. Specifically, reproducing our results using Boolean operations requires preprocessing of the input models in order to remove all undesired parts (see for example the centaur result of [1] where the horse's head and the man's waist were removed before union.)

The problem of mesh stitching has enjoyed much attention, ranging from the Zippering system of [24], to more recent methods such as [6, 15, 26]. These produce seamless model compositions even if the input models have significantly different borders, separated by large gaps. These methods, however, assume that there is no ambiguity about where the two models should connect (e.g., [15, 26]), or alternatively blend the two models in all locations where they connect(e.g., [24]). Our method, on the other hand, handles practical cases where two overlapping models can connect smoothly in many places. Given user constraints, it selects a single location through which the two models should connect, cutting and stitching them when necessary.

In addition to model composition, we propose our tool as a new solution to problems such as hole filling, restoration, and piecewise rigid deformations. Existing methods solve these problems in a variety of ways. Hole filling and restoration are often performed using diffusion based methods (e.g., [10, 25]). Diffusion, however, can only complete smooth surfaces. For example, if a nose were missing from a face, a diffusion based approach would fill the hole smoothly, generating a nose-less face. Two recent methods, [4] and [22], were shown to produce excellent results even for non-smooth models. The former morphs an ideal model prototype onto the flawed model. The latter covers flaws in the input model with surface patches taken from un-flawed parts of the same model. Our method shares some ideas with [22], however, all the methods mentioned above are specific to hole filling, and it is unclear how to extend them

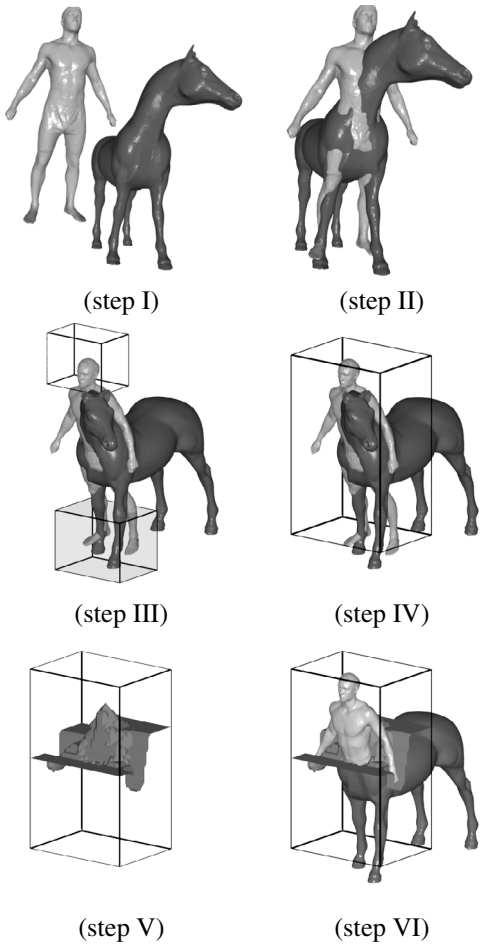

Figure 2. A typical composition session. The stages involved in creating the centaur model from those of a man and horse. (I) Input. (II) Placement (semi-automatic or manual). (III) Constraint selection (manual). (IV) Transition volume selection (manual or automatic). (V) The recovered *transition surface* (automatic). (VI) Clipped models (automatic). Final result after stitching (automatic) is given in Fig. 1.

to other applications, such as general model composition.

There are multitudes of existing methods for deforming models. These include morphing (e.g., [3]), skeleton based methods (see survey in [8]), example based methods (e.g., [23]), and more. We do not presume to replace these sophisticated methods, however, we have found that often our tool offers a simple, "quick and dirty" alternative for applying piecewise rigid deformations to models. The simplicity of our tool can be highly beneficial to unskilled users as it can easily deform models using a trivial interface (e.g., it does not require assigning skin elasticity properties etc.).

3. System overview

A typical model composition session, using our system, would proceed as follows (see also Fig. 2): The user starts by (step I) selecting two models to be composed, and (step II) placing them according to the required output, aided by our semi-automatic alignment tool (details in Section 4.1). We assume here that models are represented as meshes.

The user then specifies the composition *constraints* (step III). These are locations on each of the two input models that are desired to be included in the output model. This can be done, for instance, by selecting a cube in space that contains the constraint region, selecting a segment from a segmentation output, or even marking a single point. The constraints for each input model are thus a subset of its surface and as such are independent of the global position of the model. To constrain our algorithm to produce the centaur in Fig. 1, for example, we selected the man's head and the horse's legs (Fig. 2, step III). The rest of the centaur was extracted automatically by the algorithm.

A *transition volume* can also be specified (step IV). This is an additional means of controlling the final output, by confining where the models are allowed to connect. By default models can connect anywhere, which is to say the default transition volume is the bounding box of the models' union. The user may specify a different transition volume by drawing an appropriate box. The models would then be allowed to connect only within this user defined volume.

The system then proceeds to automatically cut and stitch the models. First, a weighted graph is constructed, (details in Sections 4.2 and 4.3) reflecting local differences between the two models in the transition volume. This graph is then cut using a minimal cut method. The graph cut represents a surface separating the transition volume into disjoint subvolumes (step V). We call this the *transition surface*, as it determines where the models should be cut and stitched, or rather, where the transition from one model to the other occurs. The weights associated with each edge in the graph ensure that this surface passes where the models are closest and most similar, which in return ensures that the resulting composition will be smooth. Both models are then clipped (Section 4.4) at the transition surface (step VI) and stitched (Section 4.5) across it to produce the final result (Fig. 1).

The user is now free to accept the composition or make further changes to either position, constraints or transition volume. Note that in making subsequent changes to any of the positions, there is no need to reselect constraints. The constraints, being parts of input models' surfaces, are unaffected by the particular position of each model. This allows quick review of different model arrangements.

Our interface is trivial, requiring only a few intuitive boxes to be roughly drawn around parts of the models. As a consequence, this tool can easily be automated, opening up

the possibility of using it for a variety of applications (a few are suggested in Section 5). We next offer a detailed look at the different stages in our system.

4. Composition framework

4.1. Part-in-whole model placement

We provide the modeler with a graphical interface capable of applying rigid transformations (i.e., translation, rotation, scale and mirror) to each model, allowing anyone with passing knowledge of modeling software to arrange them as required in a few short minutes. In addition, having designed our system for both expert and novice users, we provide a tool for semi automatic model alignment.

Most work on alignment is based on searching for an optimal alignment of complete models [21]. However, for our purposes we found this often to be inappropriate. For example, aligning the man and horse from Fig. 2 will place the man horizontally along the horse's back. Our goal is therefore different. We wish to find an optimal "part-in-whole" alignment. In other words, we require an optimal alignment of "emphasized" parts (e.g., the heads of the cow and dog in Fig. 3), and not of whole models. Models thus aligned, share overlapping surfaces in the selected parts, which can then be smoothly connected (Fig. 3).

Unlike alignment methods proposed in computer vision which use mostly feature points and lines, we use segmentation [16] as an aid. This segmentation can be semi-manual or automatic. All the user needs to do is select the emphasized parts and the system then aligns the whole models automatically. For the actual part-to-part alignment algorithm, we use PCA [14] to obtain a coarse guess for the rigid transformation between the selected parts. Standard ICP [21] is then used to refine the guess and recover the final part to part alignment.

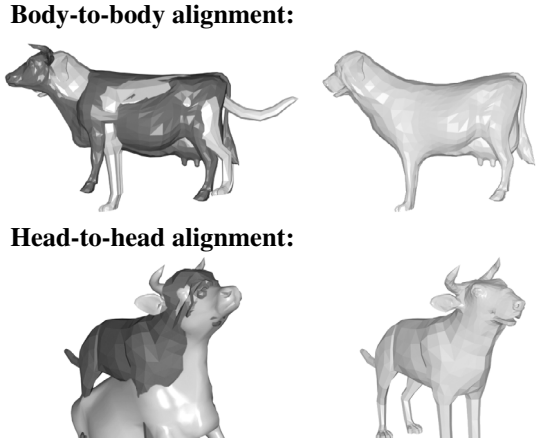

Figure 3. Part-in-whole model placement. Composition after part-in-whole alignment. Left column shows input models overlayed after alignment. Right column shows final results.

4.2. The minimal-cut of models

Once the user arranges models A and B (either manually or by using an alignment tool) and selects the constraints, the system proceeds automatically to cut and stitch them. This is achieved by running an optimization procedure to find the best location for a transition from one model to another, on each of the two models (within the transition volume). To implement this optimization procedure we construct a weighted graph $G = (V, E)$ with nodes in V representing locations, edges in E encoding neighborhoods, and weights associated with the edges, expressing the (inverse) likelihood of transitions (i.e., low cost implies smooth transition from one model to another). Auxiliary source and target nodes are added to this graph and are connected to the constrained regions in each model. The best transition is then found by computing the minimal cut in the graph (using a max flow algorithm). A detailed description of our graph is provided in Section 4.3.

A graph cut can be considered as a labeling L of all nodes in V, where in our case L_i, the label associated with node i, can either be "Model A" or "Model B". A cut passes between two neighboring nodes p and q, and is said to separate them if $L_p \neq L_q$. We call a cut minimal if the sum of all weights $w(p, q)$ for all nodes p and q separated by the cut, is minimal.

Computing a minimal cut has long since been known to have polynomial time worst case algorithms, such as the Ford and Fulkerson type methods [11]. Recently, Boykov et. al [7] have developed a variant of these methods which has been shown to have linear running time in practice for regular lattice graphs. This has made this method popular in applications involving images and video. We use this same algorithm in our implementation.

4.3. Weighted graph representation

There are several ways to implement an automatic, graph-based compositing scheme. We chose the following simple approach (see also Fig. 11). We jointly rasterize the boundaries of the models within the transition volume. Each voxel in the joint rasterization is represented by a node in the graph and two N_6 neighboring voxels are connected by an edge (we use the terms "voxel" and "node" interchangeably as they correspond). Our graph also contains two special nodes, the source and sink nodes, s and t.

If constrained faces from model A pass through voxel i we add an edge connecting node i to the source s (Fig. 11.b). Similarly, if constrained faces in B pass through a voxel, we connect it to the sink t.

Edges connected to either source or sink nodes (s or t) are assigned infinite weights as they are not allowed to be cut. Otherwise, the weight $w(p,q)$ associated with the edge connecting nodes p and q is defined as:

$$w(p,q) = \min\{dist(A_p, B_p), dist(A_q, B_q)\} \qquad (1)$$

Where A_i and B_i represent the parts of the surfaces of A and B respectively, in volume voxel i, and $dist(A_i, B_i)$ measures their distance.

The notion of a best place to connect two models is captured by the choice of a function $dist(A_i, B_i)$. Different functions reflect different user preferences for a best transition location (e.g., based on local surface curvature, texture etc.). We have tested several such functions, but found the following function particularly useful. Intuitively, we seek to cut and stitch the models where the least amount of "glue" is needed to connect them. Specifically, we attempt to cut the two models where they are closest (approximately intersecting), while at the same time minimizing the cut itself. To this end, we consider three types of voxels. A *boundary* voxel contains a boundary of only one of the two models; an *empty* voxel contains no boundary at all; an *intersection* voxel contains boundaries from both models. These three types are color coded in Fig. 11.c.

A smooth cut should connect the two models approximately through their intersection and avoid cutting where only one surface passes. Therefore, high distances are assigned to boundary voxels and low distances are assigned to intersection voxels. Moreover, intersection voxel distances are chosen such that the accumulative distance of all intersection voxels is still smaller than the distance assigned to a single boundary voxel. Since we prefer not to cut any boundary at all, we assign an even lower distance to empty voxels such that the accumulative distance of all the empty voxels is smaller than any intersection voxel. Our assignment of values is:

$$dist(A_i, B_i) = \begin{cases} 1 & \text{i is a boundary voxel} \\ \frac{1}{10k} & \text{i is an intersection voxel} \\ \frac{1}{100nk} & \text{i is an empty voxel} \end{cases} \qquad (2)$$

where k is the total number of intersection voxels and n is the total number of voxels.

The minimal cut provides us with a partition of the voxels to those labeled "Model A" and those labeled "Model B". Our composition result contains those parts of the boundary of model A located in "Model A" labeled voxels, and similarly parts of model B in "Model B" labeled voxels (Fig. 11.d-f). Note that this guarantees that the result will contain no self intersecting surfaces, as long as there were none in the original models.

The transition surface is defined as the surface made up of all voxel sides shared by voxels p and q, separated by the cut (i.e., voxels p and q for which $L_p \neq L_q$). Note that this does not have to be a single connected surface.

4.4. Mesh clipping

Given the transition surface, we clip the two meshes and stitch them into a single model, by improving on the Zippering method of Turk et al. [24]. Their system clips the faces of one model against the other's by intersecting them. Searching for intersecting faces can be very expensive computationally. To avoid this, Turk et al. assume that both models have regularly spaced, dense point clouds, which is often true for models acquired by a range scanner. We avoid this assumption while still maintaining a low computational cost by using additional information available to us - the transition surface. We clip both models not against each other, but rather against the transition surface. As the transition surface passes where the two models are closest (approximately intersecting), we are guaranteed that models clipped against it will have tightly matching borders.

Our clipping procedure is illustrated in Fig. 4. A face belonging to model A is called a *border face* (Fig. 4.a) if it contains at least one vertex in a voxel labeled A (an *inside vertex*) and at least one in a voxel labeled B (*outside vertex*). We clip all such faces by traversing edges leading from inside to outside vertices. This is done using the fast, integer based voxel traversal algorithm of Cohen [9]. The traversal terminates once a voxel label changes (i.e., the traversal along the face edge crosses the transition surface). We intersect the mesh edge with the last voxel side through which it passed, obtaining a new end vertex (Fig. 4.b). Border faces having one inside vertex are thus cropped by replacing their outside vertices with new vertices on the transition surface. Border faces having two inside vertices produce tetrahedrons which we then triangulate (Fig. 4.c). This procedure is then repeated for model B. Some clipping results are displayed in Fig. 5.c and Fig. 12.

4.5. Mesh stitching

Having clipped the two meshes, we now stitch them into a single model. Stitching can proceed by applying any standard method (e.g., [15, 26]). In fact, as the two models now share close matching borders, even a simple method would most likely do well. In practice we adopt the stitching method used in [12], based on [15]. This is not a contribution of our work, but for completeness, we now quickly review this method.

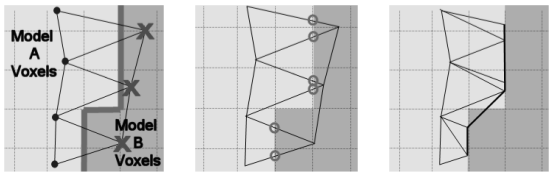

Figure 4. Mesh clipping. (a) Model A border faces crossing the transition surface (thick line). Inside vertices as circles; outside vertices as "X"s (displayed as a 2D sketch). (b) Mesh edges crossing the transition surface are clipped against it. (c) The clipped mesh.

Let C_A and C_B be two matching border contours of the two models A and B. We start by selecting two vertices n_A and n_B, one from C_A and the other from C_B, which are the closest of all such pairs. Let n'_A be the vertex 10% of the way around C_A starting at n_A, and n'_B be similarly defined on C_B. The dot product of the two vectors, the one from n_A to n'_A and the other from n_B to n'_B, gives us the orientation around C_A and C_B. Vertex correspondences are then set between C_A and C_B iteratively. Starting at n_A and n_B and proceeding along the curve for which the next vertex is closest, we match vertices by adding edges between them, creating new faces. Having thus sealed the gap between the two models, we allow the user to further smooth the new boundary by averaging vertex positions by their neighbors a user specified number of iterations, applied to vertices at a distance no larger than a user specified threshold, using user defined weights.

Figures 12, 9 and 10 illustrate various charasteristics of the suggested composition framework.

5. Applications

The system we described thus far has a trivial interface. It can therefore be easily automated, to a large extent, and applied as a unified solution to a variety of model processing tasks. We next describe a few such examples.

5.1. Model restoration

We allow a modeler easy means of repairing flawed models (e.g., scarred models) by replacing its defective boundary patches with perfect surfaces obtained from a model database. The user selects the flawed boundary (the *query*) by drawing a box around it (Fig. 5.a). Our system then searches a database of models for a surface most similar to the one selected by the user (we describe our search method below). Once found, the recovered surface patch is then automatically cut and pasted into the original model using the minimal cut tool . Our results in Fig. 5 can be compared to those of [22] and [18] obtained on the same bust model.

To automate the process, we take the user drawn box around the flaw as the transition volume. Constraints are selected by the system as follows: We assume that the flaw in the input model is roughly at the center of the user drawn box (i.e., the center of the query). Constraints on the flawed

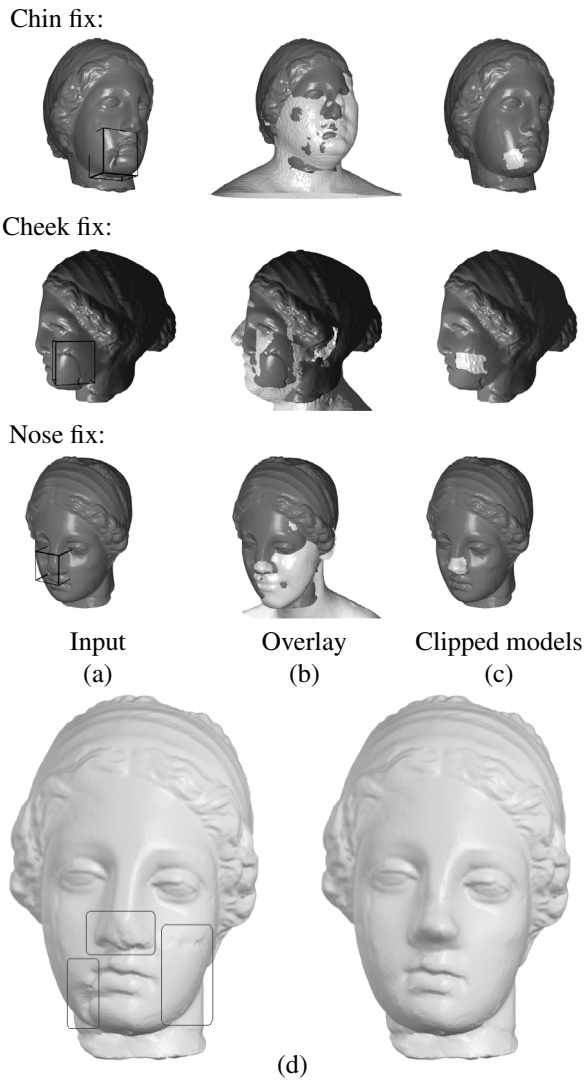

Figure 5. Model restoration. Fixing the scars and broken nose on the Igea model, in three steps. (a) Input model and the user drawn boxes around the flaws. (b) Overlay of the input model and the aligned database model chosen to fix each flaw. (c) Clipped models. (d) Input (left) and result (right).

model are therefore selected to be faces furthest from the flaw, i.e., faces closest to the sides of the transition volume. Constraints on the database model chosen to repair the flaw are selected closest to the flaw, in other words, closest to the center of the transition volume. Both constraints are illustrated in Fig. 13.a. The actual distances from the center of the volume, and its sides are governed by a user defined parameter. We found the results to be robust to these values.

Searching for the best surface patch. We search the database for the best fitting surface patch in progressively finer and finer resolutions. Resolution defaults are set for the whole database in advance and do not require changing from one query to the next. At each resolution we rasterize both the query surface and each database model. We then perform a weighted sum of squared distances search for the sub-volume most similar to the query volume. Weights equal the number of occupied voxels in each search site.

From one scale to the next we limit the search in two ways. First, we remove half the models searched in the previous scale, for which the best score was lowest. Second, searching finer scales is performed only in the area of the best match from the previous coarse scale.

Having selected a best match we then translate the whole model to align the match's position with that of the user's query. Part-in-whole alignment (section 4.1) can now be used to further refine the surface to surface alignment, taking the query and the selected database surface as emphasized parts. Fig. 5.b displays overlays of the input models (in blue) and the obtained database models (in red) after alignment.

5.2. Hole filling

Holes in meshes are a common phenomena often the product of using 3D range scanners. Given a model with holes, the user can select the area around the hole by drawing a box around it. The system then attempts to fill the hole using a database of complete models. We use the same implementation as the one used for model restoration (Section 5.1) but with a different constraint selection.

Unlike model restoration, here we have additional information about the model selected by the user: We know that parts of it are missing (i.e., it contains a hole). We therefore select constraints on the database model, to be surface patches that are furthest from the surface of the input model. This idea is illustrated in Fig. 13.b. Actual face selection is performed by calculating the discrete distance transform, D_Q, of the user's query. We further obtain the binary rasterization, R_S, of the selected database surface. The componentwise multiplication of D_Q and R_S gives us an estimate of the distance of each face in the database model from the query surface. We then choose as constraints, faces passing through voxels whose distance is larger then a user specified distance.

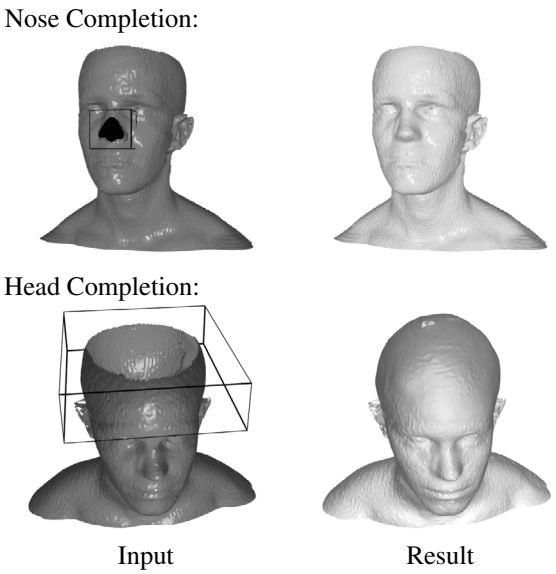

Figure 6. Hole filling. Artificial holes opened in a bust model by removing both the nose and the top of the head. These were automatically repaired in two steps. In each row on the left is the input model (with the user drawn box around the hole) and on the right is the final result (the little lump on the man's head is *not* an artifact. It is the tip of the cap worn by the scanned subjects in the database).

In practice the user can choose between the two methods of constraining the database surface, depending on the application. Fig. 6 displays two hole filling results.

5.3. Model deformations

In this Section we suggest our tool as a simple, "quick and dirty" method for applying piecewise rigid deformations to models and for generating simple 3D animations. Our idea is to take a straightforward approach to deformation. That is, we clone the model and allow the user to change the clone's position with respect to the original model (i.e., apply rotation, translation etc.) Our system then automatically cuts and stitches the two models, producing the desired deformation.

See for example the arm model in Fig. 7. Having cloned it, the user is only required to select the shoulder of one clone, and the hand of the other, as constraints. Rotating the cloned arm results in a bent arm model. Note, that once constraints are selected, there is no need to reselect them for subsequent deformations. We are therefore able to quickly create models of arms bent at different angles,

or heads looking in different directions (Fig. 8). Although limited in comparison to more sophisticated methods, our tool does allow even unskilled users to easily deform models. Compare, for example, our arm results (Fig. 7) to those obtained by interpolating models in [23].

Figure 7. Model deformations - arms. The right arm is the original. The center and left were obtained by our system.

Figure 8. Model deformations - head. The center model is the original. Left and right were obtained by our system.

6. Implementation and results

Our system is currently implemented in C++ and MATLAB. The minimal cut is obtained using the code made available by Boykov et. al [7]. The user interface for model arrangement, constraints and transition volume selection is 3D Studio Max.

Model composition. Figures 1, 3, 11, 12, 9 and 10 all display composition results. Transition volume dimensions and running times for some examples are reported in table 1. We set transition volume dimensions by hand, although the system provides default values based on available memory. We have found our results to be robust to the resolution scale, and thus, often changed it to coarser resolutions to reduce running times.

Model restoration. Fig. 5 displays three steps in restoring the scars and nose of Cyberware's Igea bust model. Our search database for completion consisted of 18 male and female models available as free samples from the CEASAR database and aligned using the *Scanalyze* software [20].

Hole filling. We removed one model from the database used for restoration, and manually cut off its nose and the top of its head. We then used the other 17 database models to repair both holes. The results are presented in Fig. 6.

Figure 9. Minimizing user intervention. Each row shows a different example. In all examples we constrained the glasses and the hat to be included as a whole in the result. We only constrain the center of the bush allowing the system to trim it freely. As constraints are common to all three examples they were set only once, thus eliminating the need for repeated user intervention. Stitching was not applied as we required separate output models.

7. Conclusions and future work

We described a new model composition tool and showed it to be flexible enough to be used as a unified solution to various modeling problems, including hole filling, restoration and rigid deformations. In addition, this tool can be plugged into such systems as the Modeling By Example framework [12], providing an alternative to mesh segmentation prior to composition.

We have plans to extend this work in several directions.

Example	*Dim.*	*Time*
Centaur (Fig. 1)	34×56×26	1 sec
Dog-cow (Fig. 3)	84×61×37	4 sec
Shades (Fig. 9)	155×71×142	16 sec
Top hat (Fig. 9)	104×76×92	40 sec
Kerberus (Fig. 10)	29×32×29	0.3 sec

Table 1. Transition volume dimensions and minimal cut average run times for various results. All run times were obtained on a standard 2.6MHz processor PC running WinXP.

These include alternative graph representations. For example, one method would be to define a graph representing the models' meshes rather than their rasterizations. Such a graph is often more compact and does not suffer the drawbacks of our sampling method.

A different direction is to investigate using our composition tool for additional applications (e.g., super resolution of models, model synthesis, etc.), and improving its interface. This can include addition of morphing methods (e.g., [3]), a better part-in-whole search method etc.

8. Acknowledgements

The bust models used for object composition, the horse, and the man models were all courtesy of Cyberware Inc. Our bust database contains freely available samples from the CAESAR database which can be found at http://www.hec.afrl.af.mil/HECP/Card1b.shtml. T.H., G.L., and R.B. were supported in part by the European Community grant IST-2002-506766 Aim Shape. The vision group at the Weizmann Institute is supported in part by the Moross Foundation. L.Z.M. was supported by the MURI award number SA3318 and by the Center of Neuromorphic Systems Engineering award number EEC-9402726. G.L. was also supported by the Israeli Ministry of Science, grant 01-01-01509. The authors also thank Ayellet Tal and Daniel Cohen-Or for helpful comments and discussions.

References

[1] B. Adams and P. Dutre. Interactive boolean operations on surfel-bounded solids. In *Proc. of SIGGRAPH*. ACM, 2003.

[2] A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen. Interactive digital photomontage. In *Proc. of SIGGRAPH*. ACM, 2004.

[3] M. Alexa. Local control for mesh morphing. In *Proc. of SMI*, 2001.

[4] B. Allen, B. Curless, and Z. Popović. The space of all body shapes: Reconstruction and parameterization from range scans. In *Proc. of SIGGRAPH*. ACM, 2003.

[5] H. Biermann, D. Kristjansson, and D. Zorin. Approximate boolean operations on free-form solids. In *Proc. of SIGGRAPH*. ACM, 2001.

[6] H. Biermann, I. Martin, F. Bernardini, and D. Zorin. Cut-and-paste editing of multiresolution subdivision surfaces. In *Proc. of SIGGRAPH*. ACM, 2002.

[7] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in computer vision. In *EMMCVPR*, 2001.

[8] M. Cavazza, R. Earnshaw, N. Magnenat-Thalmann, and D. Thalmann. Survey: Motion control of virtual humans. *IEEE Computer Graphics & Applications*, 18(5), 1998.

[9] D. Cohen. Voxel traversal along a 3d line. *Graphics gems IV*, pages 366–369, 1994.

[10] J. Davis, S. Marschner, M. Garr, and M. Levoy. Filling holes in complex surfaces using volumetric diffusion. In *Proc. Int. Symp. on 3D Data Processing, Visualization*, 2002.

[11] L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[12] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin. Modeling by example. In *Proc. of SIGGRAPH*. ACM, 2004.

[13] C. M. Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann, 1989.

[14] I. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New-York, 1988.

[15] T. Kanai, H. Suzuki, J. Mitani, and F. Kimura. Interactive mesh fusion based on local 3d metamorphosis. In *Graphics Interface*, 1999.

[16] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. In *Proc. of SIGGRAPH*. ACM, 2003.

[17] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. In *Proc. of SIGGRAPH*. ACM, 2003.

[18] K. Museth, D. Breen, R. Whitaker, and A. Barr. Level set surface editing operators. In *Proc. of SIGGRAPH*. ACM, 2002.

[19] M. Pauly, R. Keiser, L. Kobbelt, and M. Gross. Shape modeling with point-sampled geometry. In *Proc. of SIGGRAPH*. ACM, 2003.

[20] K. Pulli and M. Ginzton. Scanalyze. http://graphics.stanford.edu/software/scanalyze, 2002.

[21] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Int. Conf. on 3D Digital Imaging and Modeling (3DIM)*, 2001.

[22] A. Sharf, M. Alexa, and D. Cohen-Or. Context-based surface completion. In *Proc. of SIGGRAPH*. ACM, 2004.

[23] P.-P. Sloan, C. Rose, and M. Cohen. Shape by example. In *Symp. on Interactive 3D Graphics*, 2001.

[24] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proc. of SIGGRAPH*. ACM, 1994.

[25] J. Verdera, V. Caselles, M. Bertalmio, and G. Sapiro. Inpainting surface holes. In *IEEE Int. Conf. on Image Processing*, 2003.

[26] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum. Mesh editing with poisson-based gradient field manipulation. In *Proc. of SIGGRAPH*. ACM, 2004.

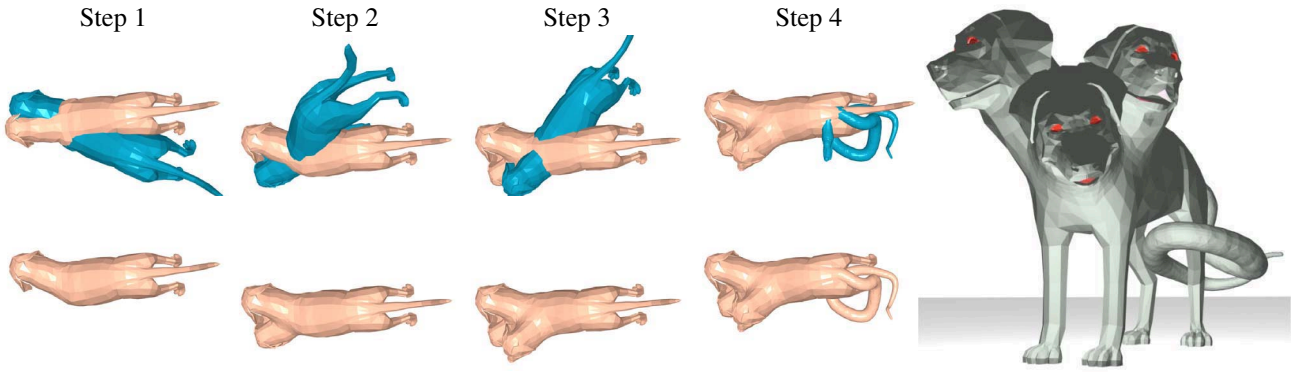

Figure 10. Multi-step composition. Kerberus, the mythical guardian of the gates of hell, is created here in four steps. The top row shows overlays of the models used to create the result in each step (shown in the bottom row). In step 1 we deform the dog's head (Section 5.3). Additional heads were created by cloning and rotating the whole dog. The tail was added by composing the existing tail with a serpent model (in blue).

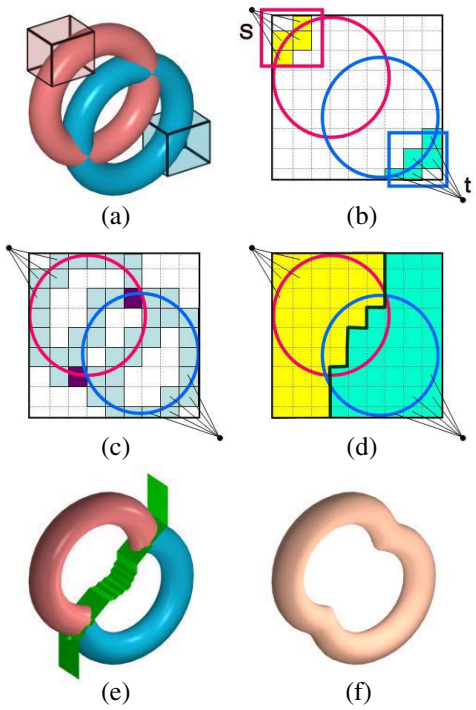

Figure 11. Weighted graph construction. (a) Input models and constraints. The transition volume is the bounding box of the models' union. (b) Voxel representation (displayed as a simplified 2D sketch). Source and sink nodes connected to constrained voxels. (c) Colors represent voxel types. Empty voxels in white, boundary voxels in light blue and intersection voxels in purple. (d) The minimal cut. Voxel labels are color coded. (e) Actual transition surface for the two tori. (f) Result.

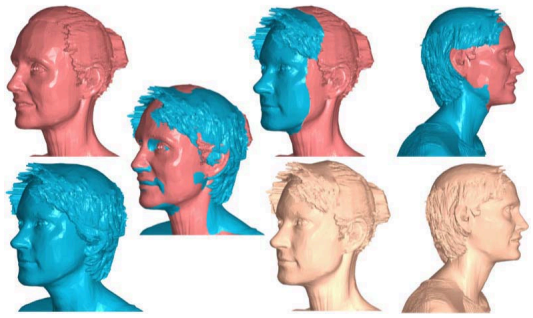

Figure 12. Reversing constraints. From left to right: The two busts; The models overlayed; First example, taking the blue face and red head; Second example, reversing constraints, now choosing the red face and blue head. In both, top is the cut result (clipped models) bottom is the final result.

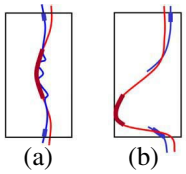

Figure 13. Automatic constraints. A 2D sketch of the automatically selected constraints. (a) The scarred surface from the Igea bust (Fig. 5). (b) The nose hole filling example (Fig. 6). The query surface is in blue; the selected database surface is in red. Query constraints are the faces closest to the transition volume sides (thick blue). Dark red patches are constraints on the selected database model. For restoration, these are faces closest to the center of the volume. For hole filling, these are faces furthest from the query (blue) surface.