Modeling by Example

Thomas Funkhouser,[1] Michael Kazhdan,[1] Philip Shilane,[1] Patrick Min,[2]
William Kiefer,[1] Ayellet Tal,[3] Szymon Rusinkiewicz,[1] and David Dobkin[1]

[1]Princeton University [2]Utrecht University [3]Technion - Israel Institute of Technology

Abstract

In this paper, we investigate a data-driven synthesis approach to constructing 3D geometric surface models. We provide methods with which a user can search a large database of 3D meshes to find parts of interest, cut the desired parts out of the meshes with intelligent scissoring, and composite them together in different ways to form new objects. The main benefit of this approach is that it is both easy to learn and able to produce highly detailed geometric models – the conceptual design for new models comes from the user, while the geometric details come from examples in the database. The focus of the paper is on the main research issues motivated by the proposed approach: (1) interactive segmentation of 3D surfaces, (2) shape-based search to find 3D models with parts matching a query, and (3) composition of parts to form new models. We provide new research contributions on all three topics and incorporate them into a prototype modeling system. Experience with our prototype system indicates that it allows untrained users to create interesting and detailed 3D models.

Keywords: databases of geometric models, 3D shape matching, interactive modeling tools

1 Introduction

One of the most significant obstacles in computer graphics is providing easy-to-use tools for creating detailed 3D models. Most commercial modeling systems are difficult to learn, and thus their use has been limited to a small set of trained experts. Conversely, 3D sketching programs are good for novices, but practical for creating only simple shapes. Our goal is to provide a tool with which almost anybody can create detailed geometric models quickly and easily.

In this paper, we investigate "modeling by example," a data-driven approach to constructing new 3D models by assembling parts from previously existing ones. We have built an interactive tool that allows a user to find and extract parts from a large database of 3D models and composite them together to create new 3D models. This approach is useful for creating objects with interchangeable parts, which includes most man-made objects (vehicles, machines, furniture, etc.) and several types of natural objects (faces, fictional animals). Our current implementation employs a database of more than 10,000 models, including multiple examples of almost every type of household object.

The main motivation for this approach is that it allows untrained users to create detailed geometric models quickly. Unlike previous interactive modeling systems, our users must only search, select, and combine existing parts from examples in the database – i.e., they rarely have to create new geometry from scratch. As a result, the user interface can be simpler and accessible to a wider range of people. For example, when making the rocking chair shown in Figure 1, the user started with a simple chair (top-left), and then simply replaced parts. The commands were very simple, but the result has all the geometric details created by the expert modelers who populated the database. This approach provides a new way to make 3D models for students, designers of virtual worlds, and participants in on-line 3D games.

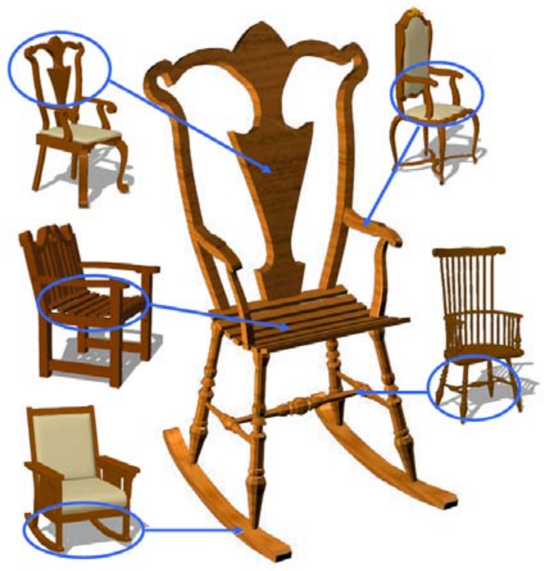


Figure 1: Modeling by example: geometric parts extracted from a database of 3D models can be used to create new objects. The large brown chair was built from the circled parts of the others.

In the following sections, we address the main research issues in building such a system: segmenting 3D surfaces into parts, searching a database of 3D models for parts, and compositing parts from different models. Specifically, we make the following research contributions: (1) an intelligent scissors algorithm for cutting 3D meshes, (2) a part-in-whole shape matching algorithm, (3) a method for aligning 3D surfaces optimally, and (4) a prototype system for data-driven synthesis of 3D models. Experience with our prototype system indicates that it is both easy to learn and useful for creating interesting 3D models.

2 Related Work

This paper builds upon related work in several sub-fields of computer graphics, geometric modeling, and computer vision.

Geometric modeling: Our system is a 3D modeling tool. However, its purpose is quite different than most previous modeling systems (e.g., [Wavefront 2003]). It is intended for rapidly combining existing geometry into new models, and not for creating new geometry from scratch. As such, it has a synergistic relationship with other modeling systems: our tool will benefit from improvements to existing modeling systems, since there will then be larger/better databases of 3D geometry, while other modeling systems will likely benefit from including the methods described in this paper to provide better utilization of existing models.

Sketch modeling tools: Our system shares many ideas with 3D sketching systems, such as Sketch [Zeleznik et al. 1996] and Teddy [Igarashi et al. 1999]. Like these systems, we follow the general philosophy of keeping the user interface simple by inferring the intention of a few, easy-to-learn commands, rather than providing an exhaustive set of commands and asking the user to set several parameters for each one. However, previous systems have achieved their simplicity by limiting the complexity and types of shapes that can be created by the user. We achieve our simplicity by leveraging existing geometry stored in a database.

Data-driven synthesis: Our work is largely inspired by the recent trend towards data-driven synthesis in computer graphics. The general strategy is to acquire lots of data, chop it up into parts, determine which parts match, and then stitch them together in new and interesting ways [Cohen 2000]. This approach has been demonstrated recently for a number of data types, including motion capture data (e.g., [Lee et al. 2002]). However, to our knowledge, it has never been applied to 3D surface modeling. Perhaps this is because 3D surfaces are more difficult to work with than other data types: they are harder to "chop up" into meaningful parts; they have more degrees of freedom affecting how they can be positioned relative to one another; they have no obvious metric for identifying similar parts in the database; and, they are harder to stitch together. These are the issues addressed in this paper.

Shape interpolation: Our work shares many ideas with "shape by example" [Sloan et al. 2001] and other blending systems whose goal is to create new geometric forms from existing ones (e.g., [Lazarus and Verroust 1998]). However, our approach is quite different: we focus on recombining parts of shapes rather than morphing between them. We take a combinatorial approach rather than an interpolative one. Accordingly, the types of shapes that we can create and the research issues we must address are quite different. We believe that our approach is better suited for creating shapes composed of many parts, each of which has a discrete set of possible forms (e.g., cars, tables, computers, etc.), while interpolation is better for generating new shapes resulting from deformations (e.g., articulated motions).

Geometric search engines: Our system includes the ability to search a large database of 3D models for matches based on keyword and/or shape similarity. In this respect, it is related to 3D search engines that have recently been deployed on the Web (e.g., [Chen et al. 2003; Corney et al. 2002; Funkhouser et al. 2003; Paquet and Rioux 1997; Suzuki 2001; Vranic 2003]). Several such systems have acquired impressive databases and allow users to download 3D models for free. In our current implementation, we use the data of the Princeton 3D Model Search Engine [Min et al. 2003]. That system and ones like it employ text-based search methods similar to ours. However, their shape-based matching algorithms consider only whole-object shape matching. In this paper, we address the harder problem of part-in-whole shape matching.

To our knowledge, this is the first time that a large database of example 3D models and shape-based retrieval methods have been integrated into an interactive modeling tool.

3 System Overview

The input to our system is a database of 3D models, and the output is a new 3D model created interactively by a user. The usual cycle of operation involves choosing a model from the database, selecting a part of the model to edit, executing a search of the database for similar parts, selecting one of the models returned by the search, and then performing editing operations in which parts are cut out from the retrieved model and composited into the current model. This cycle is repeated until the user is satisfied with the resulting model and saves it to a file. The motivation for this work cycle is that it requires the user to learn very few commands (open, save, select, cut, copy, paste, undo, search, etc.), all of which are familiar to almost every computer user.

A short session with our system is shown in Figure 2. Imagine that a school child wants to investigate what the Venus de Milo sculpture looked like before her arms were broken off. Although there are several theories, some believe that she was holding an apple aloft in her left hand, and her right arm was posed across her midsection [Curtis 2003]. Of course, it would be very difficult for a child to construct plausible 3D models for two arms and an apple from scratch. So, we investigate extracting those parts from other 3D models available in our database.

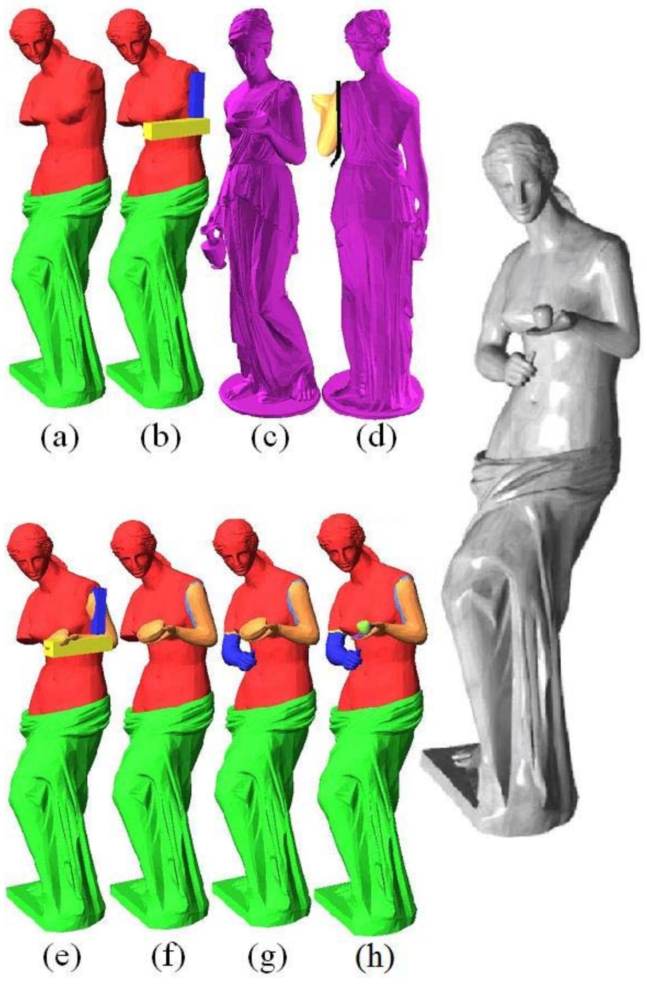


Figure 2: Screenshots of a ten-minute session demonstrating the main features of our system being used to investigate what Venus looked like before she lost her arms.

In this example, a text search yields a 3D model of the armless Venus (a). Then, two boxes are drawn representing the desired pose of a new left arm (b), and a part-in-whole shape search yields a sculpture of Hebe with an arm in a similar pose (c) as the top match. The matching arm is cut off Hebe using intelligent scissors with a single, approximate mouse stroke (black line in (d)), producing a cut along the natural seam over her shoulder and through her arm pit separating her arm (yellow) from her body (purple). The arm is then copied to the clipboard and pasted into the window with the Venus sculpture. A good initial placement for the arm is found with automatic optimal alignment to the two boxes (e), saving the user most of the work of interactively moving, rotating, and scaling it to the right place. Next, a hole is cut in Venus' left shoulder and stitched to the open boundary of the pasted arm to make a watertight junction with smooth blending (f). The second arm is found with text search, and similar cut, copy, paste, and blend operations transfer it onto Venus' body (g). Finally, cutting the bowl out of her left hand and replacing it with an apple found with text keyword search yields the final model (h). While the result is certainly not historically accurate, it took less than 10 minutes to make, and it provides a plausible rendition suitable for school and entertainment applications.

Although this example is mostly for pedagogy and fun, it demonstrates the main features of our system: (1) segmenting surface models into parts, (2) searching a 3D model database for parts, and (3) compositing multiple parts into a new model. The following sections detail our research contributions and design decisions in each of these areas.

4 Intelligent Scissoring of 3D Meshes

The first issue we address is segmenting 3D models into parts. Our goal is to provide simple and intuitive tools with which the user can quickly and robustly specify a meaningful subset of a 3D model (a *part*) to form a selection, a query for a search, and/or the target for a surface editing operation.

Of course, many models come already decomposed into scene graph hierarchies or multiple disconnected components, and several approaches have been proposed for automatic mesh segmentation (e.g., [Katz and Tal 2003]). We maintain these segmentation(s) when they are available. However, often the provided segmentation is not what the user needs for a particular editing operation, and thus we must also consider interactive methods.

There are several possible approaches to interactive segmentation of 3D meshes. First, most commercial modeling tools allow the user to draw a screen-space split line or "lasso" and then partition the mesh vertices according to their screen space projections (Figure 3). This approach is only able to make cuts aligned with the camera view direction, which is often not what the user wants (top row of Figure 3). Second, some systems allow the user to select a sequence of vertices on the surface mesh and cut along the shortest paths between them [Gregory et al. 1999; Wong et al. 1998; Zöckler et al. 2000]. Although this method supports cuts of arbitrary shape, it requires great care when selecting points (because the cut is constrained to pass exactly through those points), points must be specified in order (which makes fixing mistakes difficult), and the camera view must be rotated to get points on multiple sides of the object (otherwise, the shortest path will not encircle the object). Finally, other systems (e.g., [Lee and Lee 2002]) have used "active contours" to adjust a user-drawn cut to follow the closest seams of the mesh. While this approach allows the user to specify the cut less precisely, we find active contours hard to control – i.e., it is difficult to find a set of parameters that prevent the snakes from getting stuck on small bumps and irregularly sampled regions of the mesh while ensuring that they find the natural seams of the mesh nearby the user's cut. Perhaps more sophisticated variational approaches (e.g., coarse-to-fine refinement) can address this issue. Rather, we decided to provide a more explicit method for the user to control the placement of the cut while guaranteeing that an optimal seam is found.

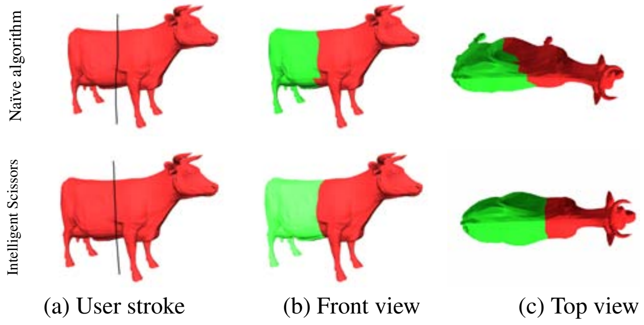


Figure 3: A screen-space "lasso" (top row) produces an unexpected segmentation when the camera view is not perfectly aligned with the desired cut. In contrast, our intelligent scissors (bottom row) finds the optimal cut through the stroke, which may or may not be orthogonal to the view direction.

4.1 Painting Strokes

We allow the user to paint "strokes" on the mesh surface to specify where cuts should be made (Figure 4a). Each stroke has a user-specified width (r) representing a region of uncertainty within which the computer should construct the cut to follow the natural seams of the mesh (our current system considers only cuts along edges of the mesh). From the user's perspective, the meaning of each paint stroke is "I want to cut the surface along the best seam within here." From the system's perspective, it specifies a constraint that the cut *must* pass within r pixels of every point on the stroke, and it provides parameters for computing the cost of cutting along every edge (e) in the mesh:

$$cost(e) = c_{len}(e) \times c_{ang}(e)^{\alpha} \times c_{dist}(e)^{\beta} \times c_{vis}(e)^{\gamma} \times c_{dot}(e)^{\delta}$$

where $c_{len}(e)$ is the edge's length, and α, β, γ, and δ trade off between factors based on the dihedral angle (θ_e) of its adjoining faces ($c_{ang}(e) = \theta_e/2\pi$), its visibility at the time the stroke was drawn ($c_{vis}(e) = 1$ if e was visible, and 0.5 otherwise), the orientation of its surface normal ($\vec{N}$) with respect to the view direction ($\vec{V}$) when the stroke was drawn ($c_{dot}(e) = (1 + \vec{V} \cdot \vec{N})/2$), and the maximum distance (d) from the centerline of the stroke to the screen space projection of the edge ($c_{dist}(e) = \frac{r-d}{r}$). Default values for α, β, γ, and δ are all one.

Intuitively, this definition imposes less cost for edges that are shorter, along more concave seams of the mesh, closer to the center of the stroke, and invisible or back-facing when the stroke was drawn. The first three terms are self-explanatory. The fourth term ($c_{vis}(e)$) is motivated by the observation that a user probably would have painted on a visible edge if a cut were desired there – i.e., not drawing on a visible portion of the mesh implies "don't cut here." This term combines with $c_{dot}(e)$ to encourage the least cost closed contour to traverse the "back-side" of the mesh. Without these two terms, the least cost closed contour through the stroke would most likely fold back along itself, traveling from the end of the stroke back to the beginning along a path close to the stroke on the front side of the mesh. However, with either of these terms, the cost of traversing the back side of the mesh diminishes (Figure 4f), and the closed contour usually encircles the mesh, making it possible to cut the mesh with one stroke (Figure 4g-h). In particular, with sufficient weighting of these cost terms ($\gamma >> \alpha$ and/or $\delta >> \alpha$), the

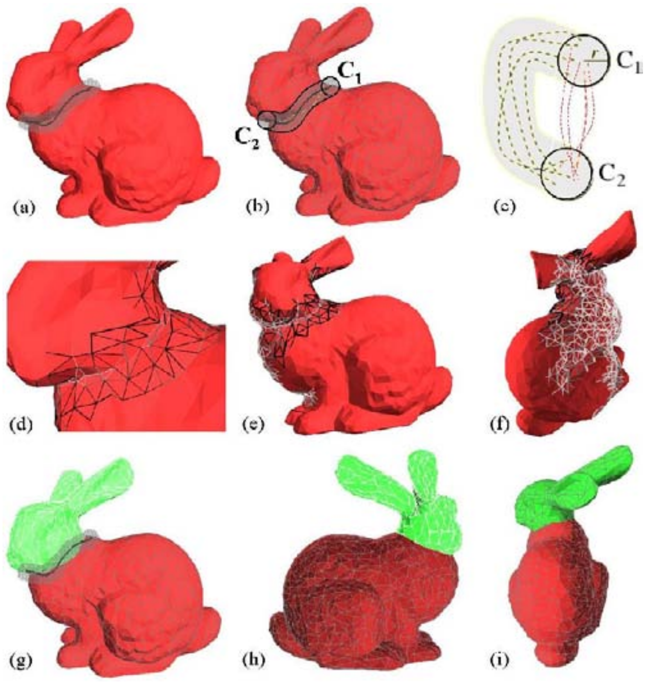


Figure 4: Cutting the bunny with intelligent scissoring: (a) the user draws a wide paint stroke; (b) the system identifies all vertices in the *caps* of the stroke, C_1 and C_2; (c) it then finds the least cost paths from every vertex in C_1 to every vertex in C_2 twice, once constrained to lie within the stroke (yellow dotted lines) and once without any constraints (red dotted lines), and forms the proposed cut out of the pair of paths with least total cost; (d-f) Since the edges traversed by the algorithm (wireframe gray) have less cost (lighter gray values) in concave seams and on the back-side of the mesh, (g-f) the least cost cut partitions the mesh into two parts (red and green) along a natural seam of the mesh.

closed contour is guaranteed to traverse the back side of the mesh whenever the sum of costs to travel from the stroke endpoints to the closest silhouette boundaries is less than the cost to travel between the endpoints along the front side of the mesh.

4.2 Finding the Cut

We solve a constrained least cost path problem to find the optimal cut specified by the user's stroke. More formally, we find the sequence of mesh edges with least cost that passes within r pixels of every point on the user's stroke in sequence. This is a graph version of the "safari problem," where the "cages" to visit are the sequence of overlapping circular neighborhoods defined by the centerline and width of the user's stroke. Since we do not have explicit start and stop vertices, we cannot find the least cost path directly with Dijkstra's algorithm. Rather, we must search over many possible start and stop vertices to find the least cost path anywhere within the stroke. While this approach is computationally more expensive, it allows the user to cut meshes quickly with partial and approximate strokes, while the computer fills in the details of the cut automatically.

The key to solving this least cost path problem efficiently is observing that the cut must pass through at least one vertex in the "cap" at each end of the stroke, C_1 and C_2 – i.e., the set of vertices projecting within r pixels of the first and last points on the stroke (Figure 4b). Thus, we can divide-and-conquer by solving two sub-problems: (1) find the least cost path from all vertices in C_1 to all vertices in C_2 (or vice-versa) while constrained to traverse only edges within the stroke (yellow dotted lines in Figure 4c), and (2) find the least cost path connecting all vertices in C_2 back to all vertices in C_1 without any constraints (red dotted lines in Figure 4c). The optimal cut is the pair of paths, one from each sub-problem, that forms a closed contour with least total cost [Mitchell 2003]. These sub-problems are solved with an execution of Dijkstra's algorithm for each vertex in C_1.

The computational complexity of this algorithm for a single stroke is $O(k_1 \cdot n \log n)$, where n is the number of edges in the mesh, and k_1 is the number of vertices in C_1. Since k_1 is usually small, and least cost path searches usually cover a subset of the mesh (Figure 4d-f), running times are interactive in practice. This algorithm took under a second for all examples shown in this paper.

4.3 Refining the Cut

Our system also allows the user to refine the cut interactively with "over-draw" strokes. Immediately after the first stroke, the system can partition the mesh according to the computed optimal cut (the default). Alternatively, it can display a "proposed cut" for verification. If the user is not satisfied, she can draw new strokes that refine the cut incrementally (Figure 5). This feature encourages the user to draw broad strokes quickly, in any order, and then iteratively refine the details only where necessary.

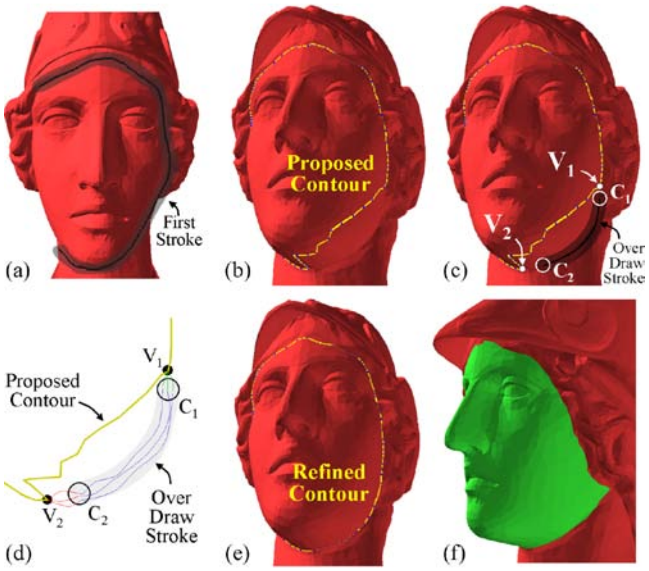


Figure 5: Cutting the face of Athena with intelligent scissoring: (a) the user draws an imprecise first stroke (gray); (b) the system proposes a cut (yellow curve); (c) the user draws an overdraw stroke (gray) to refine the cut; (d) the system splices in the least cost path traveling from V_1 first to C_1 (red) then to C_2 within the stroke (blue) and finally to V_2 (green); (e) the proposed cut contour is updated; (f) the final result is a segmentation of the mesh into two parts (green and red) separated by natural seams of the mesh.

For each over-draw stroke, S, the system splices a new, locally optimal path through S into the proposed cut contour. The system splices out the portion of the proposed contour between the previously painted vertices, V_1 and V_2, closest to the two endpoints of the over-draw stroke (Figure 5c). It then splices in a new contour with least cost from V_1 to V_2 traveling within r of every pixel in the over-draw stroke. This path is found in three stages (Figure 5c). First, a single execution of Dijkstra's algorithm finds the least cost paths from V_1 to all vertices in the over-draw stroke's first cap, C_1.

Then, those paths are augmented with the least cost paths within the stroke from all vertices within C_1 to all vertices in C_2, the second cap of the over-draw stroke. Finally, those paths are augmented with the least cost paths from all vertices in C_2 to V_2, the point at which the new cut contour connects back to the original proposed cut (Figure 5d). The path found with overall least cost is spliced into the proposed cut (Figure 5e).

This incremental refinement approach has several desirable properties. First, it provides local control, guaranteeing that previously drawn strokes will not be overridden by new strokes unless they are in close proximity. Second, it is fast to compute, since most of the least cost path searches are constrained to lie within the stroke. Finally, it allows the user to specify precisely and naturally where the splice should be made both by simply starting and stopping the over-draw stroke with the cursor near the proposed contour.

5 Search for Similar Parts

The second issue we address is searching a large database of 3D models. The challenge is to develop effective mechanisms whereby users can enter a query specifying the objects they want to retrieve from the database and the system quickly returns the best matches and suggests them as candidates for future editing operations. In general, we would like to support two types of queries. Initially, we expect the user to search for whole objects that represent the general shape they would like to construct – requiring an interface supporting whole-object matching. As the user progresses through her editing operations, we expect that she will want to replace individual parts of the model with parts from other models – requiring an interface supporting partial object matching. In either case, it is important that the interface be easy to use and capable of finding similar parts efficiently.

Perhaps the simplest and most common query interface is textual keywords. Most users are familiar with this type of query and it is well suited for finding 3D models based on their semantics if models are well-annotated. However, recent research [Min 2004] has indicated that many models are poorly annotated. This problem is further exacerbated if we are interested in finding parts of models, which tend to have even less annotation. For this reason, our system augments text search with shape-based methods.

Traditional methods for matching shapes [Besl and Jain 1985; Loncaric 1998; Tangelder and Veltkamp 2004] have focused on whole-object matching, providing methods for finding models whose overall shape is similar to a query. We provide this feature in our system, but would also like to support "part-in-whole" shape searches. This type of query matches whole objects, but with special emphasis on a selected part. It is very useful for finding specific parts within a class of objects. For instance, consider a situation in which a user has a chair and would like to replace some of the parts (Figure 6). If she performs a shape similarity query with only the legs of the chair selected, then the system should retrieve chairs with similarly shaped legs, independent of the presence or shapes of their arms. On the other hand, if she selects only the arms of the chair and performs a shape similarity query, the system should find chairs with arms. Figure 6 shows the results achieved by our system in this case.

Traditional shape matching methods that represent each 3D model by a multi-dimensional feature vector (a shape descriptor) and define the dissimilarity between a pair of models as the distance (L_1, L_2, L_∞, etc.) between them are not well-suited for this type of query. They consider two models the same if and only if their descriptors are equal. In this work, we would like to represent a single model by many different descriptors, depending on the features selected by the user. Since it is imperative that a model with

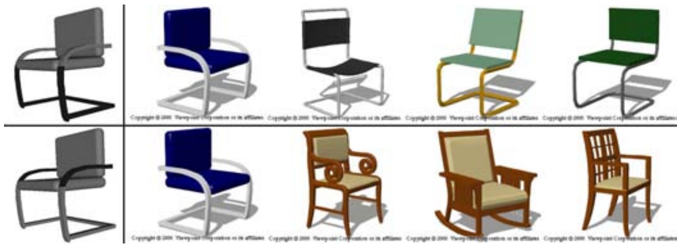


Figure 6: Results of shape similarity queries where the query provided to the system is (top) the chair with the legs selected, and (bottom) the chair with the arms selected.

features selected always match itself, we must use an alternative approach.

The notion of shape similarity that we use is based on the sum of squared distances for models aligned in the same coordinate system. Specifically, we define the distance between two models as the sum of the squares of the distances from every point on one surface to the closest point on the other, and vice-versa. This definition of shape similarity is intuitive, since it approximates the amount of work required to move points on one surface to the other (as in [Rubner et al. 2000; Tangelder and Veltkamp 2003]), and it implies that two shapes should be subsets of one another to achieve a good match. It is also well suited for feature based matching, since we can associate a weight to each point and then scale the contribution of each summand accordingly. Then, selected parts (points with higher weight) can contribute more to the measure of shape similarity than others.

While a direct approach for computing the sum of squared distances would require a complex integration over the surfaces of the models, we present a new method for computing this distance that is easy to implement (Figure 7). For each model A in the database, we represent the model by two voxel grids, R_A and E_A. The first voxel grid, R_A, is the rasterization of the boundary, with value 1 at a voxel if the voxel intersects the boundary, and value 0 if it does not. The second voxel grid, E_A is the squared Euclidean Distance Transform of the boundary, with the value at a voxel equal to the square of the distance to the nearest point on the boundary. In order to compare two models A and B we simply set the distance between the two of them to be equal to:

$$d(A,B) = \langle R_A, E_B \rangle + \langle E_A, R_B \rangle,$$

the dot product of the rasterization of the first model with the square distance transform of the second, plus the dot product of the rasterization of the second model with the squared-distance transform of the first. The dot product $\langle R_A, E_B \rangle$ is equal to the integral over the surface of A of the square distance transform of B. Thus, it is equal precisely to the minimum sum of square distances that points on the surface of A need to be moved in to order to lie on the surface of B.

For matching parts of models, the process is very similar. Given a triangulated model A with a subset of triangles, $S \subset A$, selected by the user as features with some weight w, we can compute the feature weighted descriptor of A, $\{R_{A,S}, E_A\}$, by setting $R_{A,S}$ to be the rasterization of A with value w for points on S, a value of 1 for points on A that are not in S, and a value of 0 everywhere else. When we compute the dot product of the weighted rasterization with the distance transform of another model, $\langle R_{A,S}, E_B \rangle$, we get the weighted sum of square distances, with the contribution of square distances from points on S scaled by weighting factor w. This approach allows us to compute just a single shape descriptor for each model in the database, while using it for matching queries with arbitrary feature weighting.

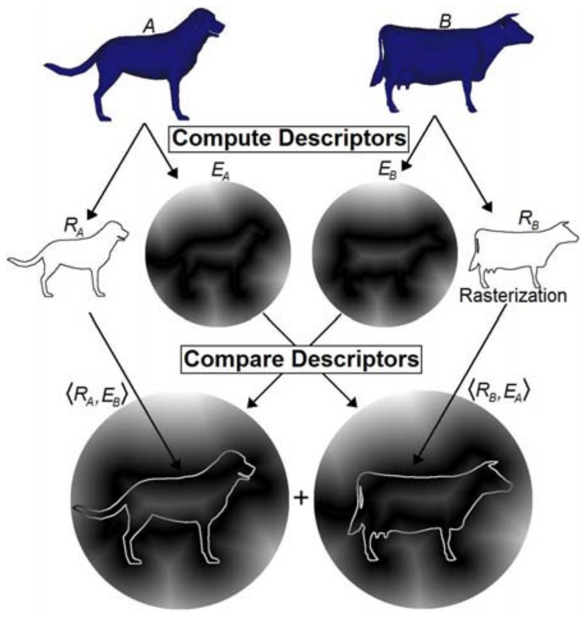


Figure 7: Two models are compared by computing the voxel rasterization and square distance transform of each one and defining the distance measure of model similarity as the dot product of the rasterization of the first with the distance transform of the second, plus the dot product of the distance transform of the first with the rasterization of the second. The resultant value is equal to the minimum sum of square distances that points on each model need to be moved in order to lie on the other model.

Overall, the advantages of this shape descriptor are two-fold. First, its underlying notion of similarity is based on the sum of squared distances between points on two surfaces, which is intuitive to users. Second, it works for matching both whole objects and objects with specific parts within the same framework, requiring just a single descriptor for every object in the database. Moreover, this descriptor has the non-trivial property that the distance between two different feature weighted representations of the same model is always equal to zero.

6 Composition of Parts

The third issue we address is how to assemble parts from different sources into a single model. Our goal is to provide interactive tools for the user to position and orient parts when they are added to a new model and possibly to stitch the surfaces together at the joints.

Commercial 3D modeling programs often provide interactive tools to perform these functions. So, our goal here is not new. However, our system is intended for novice users, and thus we aim to find simpler interfaces and more automatic methods than are typically found in commercial software. There are two challenges we must address: part placement and part attachment.

6.1 Part Placement

When inserting a part into a model, the first challenge is to find the transformation that places it into the appropriate coordinate frame with respect to the rest of the model. We have investigated several options, including most of the interactive direct manipulation (translation, rotation, anisotropic scale, etc.) and alignment commands (align centers, align tops, align anchors, align moments, etc.) commonly found in 3D commercial modeling programs. However, we find that the subtleties of 3D placement are difficult to master for most novice users. So, we are motivated to find the best possible automatic strategies.

In particular, our users are often faced with the problem of replacing one part with another. For instance, they may have created a simple version of a part and then queried the database for similar ones to replace it (Figure 2b). Or, they may have started with a plain version of an object selected from the database, and then want to replace parts from it with better versions. In either case, we would like to provide a simple automatic command with which the new part (the query) can be placed in the same coordinate frame as the one it is replacing (the target). Specifically, we would like to solve for the translation, scale, rotation, and mirror that minimizes the sum of squared distances from each point on one surface to the closest point on the other, and vice-versa.

Based on the work of [Horn 1987; Horn et al. 1988], we can solve for the optimal translation by moving the query part so that its center of mass aligns with the center of mass of the target. Similarly, we can solve for the optimal scale by isotropically rescaling the query part so that its mean variance (sum of squared distances of points from the center of mass) is equal to the mean variance of the target. However, solving for the optimal rotation using the method of [Horn 1987; Horn et al. 1988] would require the establishment of point-to-point correspondences which is often a difficult task.

A commonly used method for aligning two models is the ICP algorithm [Besl and McKay 1992] which, given an initial guess, will converge to a locally optimal solution minimizing the sum of squared distances between the two models. In our approach we provide a voxel space implementation of ICP that does not require an initial guess and is guaranteed to give the globally optimal solution. Specifically, we use the recently developed signal processing techniques of [SOFT 1.0 2003; Kostelec and Rockmore 2003] that efficiently solve for the correlation of two functions – giving a 3D array, indexed by rotations, whose values at each entry is the dot product of the first function with the corresponding rotation of the second. In particular, using the shape descriptors from Section 5 as the input functions for correlation, we obtain a 3D array whose value at a given entry is the minimum sum of squared distances between the two models, at the corresponding rotation. Thus, searching the 3D correlation array for the entry with smallest value gives the optimal rotation for aligning the two models. This process takes $O(N^4)$ for a $N \times N \times N$ voxel grid, rather than the $O(N^6)$ that would be required for brute force search.

6.2 Part Attachment

Of course, the model resulting from simply placing parts next to one another does not produce a connected, manifold surface. Although disconnected meshes are common in computer graphics, especially for man-made objects, our system provides simple methods based on [Kanai et al. 1999] with which the user can stitch parts together to form smooth, connected surfaces from multiple parts. While these methods are not a research contribution of this paper, we describe them briefly for completeness.

When a user wishes to join two parts, she first selects an open boundary contour on each mesh, C1 and C2, possibly first cutting holes with intelligent scissors (Figure 8a). Then, she executes a ”join” command, which uses heuristics to establish vertex correspondences for filleting and blending automatically. Specifically, the closest pair of vertices, $V1$ and $V2$, are found, with $V1$ on C1 and $V2$ on C2 (Figure 8b). Then, the orientation for C1 with respect to C2 is found by checking the dot product of the vector from $V1$ to $V1'$, a vertex 10% of the way around the length of C1, and the vector from $V2$ to $V2'$, defined similarly. If the dot product is negative, the orientation of C1 is switched. Then, vertex correspondences are established with a greedy algorithm that iteratively increments the ”current vertex” on either C1 or C2, choosing the one such that the Euclidean (or parametric) distance between the two current vertices is least. Finally, fillet edges are constructed between corresponding vertices (Figure 8c), and any resulting quadrilaterals are split into triangles. Additionally, the join command can smooth vertices within a user-specified distance of the fillet by averaging their positions with all their neighbors with user-specified weights for a user-specified number of iterations (Figure 8d).

Although these filleting and blending methods are not particularly sophisticated, they always produce a watertight junction, and they do make it possible to create natural objects with smooth surfaces (e.g., Figure 8e). In future work, we plan to include more general and automatic approaches based on boolean operators, level sets, and/or morphing (e.g., [Alexa 2001; Museth et al. 2002]).

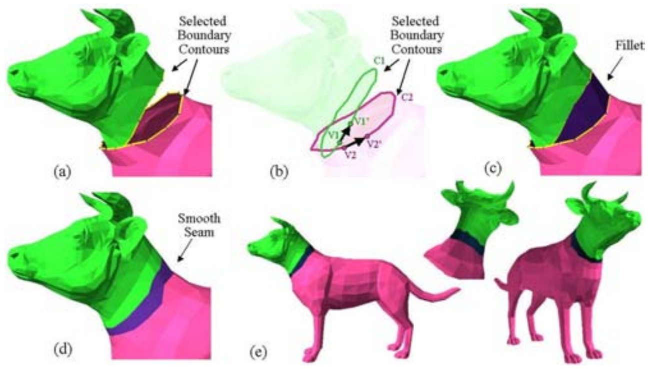


Figure 8: Attaching the head of a cow to the body of a dog: (a) a boundary contour is selected on each part ($C1$ and $C2$); (b) the pair of closest points ($V1$ and $V2$) is found and the local direction near those points is used to determine the relative orientation of the contours; (c) a fillet is constructed attaching the contours; (d) the mesh is smoothed in the region nearby the seams of the fillet. (e) the result is a smooth, watertight seam.

7 Implementation

We have implemented a prototype system with all the features described in this paper. In this section, we provide several implementation details and discuss limitations.

7.1 Hardware and Software

Our system’s architecture consists of a client running the user interface and two servers running the shape and text matcher respectively. The client PC has one 2.8 GHz Pentium IV processor, 1 GB memory, a GeForce4 graphics card, and is running Windows XP. The server PCs each have two 2.2 GHz Xeon processors, 1 GB memory, and are running Red Hat Linux 9.0.

7.2 Shape Descriptors

When computing shape descriptors, we rasterize the surfaces of each 3D model into a $64 \times 64 \times 64$ voxel grid, translating the center of mass to its center, and rescaling the model so that twice its mean variance is 32 voxels. Every model is normalized for rotation by aligning its principal axes to the x-, y-, and z-axes. Finally, the descriptors are rescaled so that the L_2-norm of both the rasterization and the Euclidean Distance Transform are equal to 1, and $w = 10$ is used for feature weighted matching.

When matching 3D models, we resolve the ambiguity between positive and negative principal axes by using the efficient axial search method described in [Kazhdan 2004]. In particular, the coefficients of the voxel grids are decomposed into eight different component vectors and the measure of similarity at each of the eight axial flips is computed by summing the dot products of the component vectors with appropriate sign. The dot product minimized over the different axial flips is used as the measure of similarity between models.

In order to facilitate efficient storage and matching of the shape descriptors, we use standard SVD techniques to project the shape descriptors onto a low-dimensional subspace. Specifically, we compute the covariance matrices of the eight component vectors of each descriptor in the database, solve for the Eigenvalues of the matrices, and project each of the component vectors onto the subspace spanned by the Eigenvectors associated with the top 100 Eigenvalues. This allows us to compress the initial $2 \times 64 \times 64 \times 64 = 524,288$ dimensional representation down to a $2 \times 8 \times 100 = 1600$ dimensional vector. This compression approach is well-suited for our application because we define the distance between models in terms of the dot product between two functions. Any part of the query descriptor that is not well represented in the compressed domain is mostly orthogonal to the subspace repesented by the top Eigenvectors and hence orthogonal to most of the target models and does not contribute much to the dot product.

We have found that these choices maintained sufficient resolution to allow for discriminating matching, while limiting the size of the descriptor and allowing for efficient computation and retrieval. In particular, the descriptor of each model is computed in two seconds, on average, and a database of over 10,000 models returns retrieval results in under one second. Our compressed descriptors provide the same retrieval precision as the full ones, though they are easier to store and more efficient to compare.

7.3 Database and Preprocessing

Our test database contains 11,497 polygonal models [Min et al. 2003], including 6,458 free models collected from the World Wide Web during crawls in October 2001 and August 2002, and 5,039 models provided by commercial vendors (Viewpoint, Jose Maria De Espona, and CacheForce). We limited our database to include only models representing objects commonly found in the real world (e.g., chairs, cars, planes, and humans). Other classes, including molecules, terrains, virtual environments, terrains, and abstract objects were not included. The average triangle count is 12,940, with a standard deviation of 33,520.

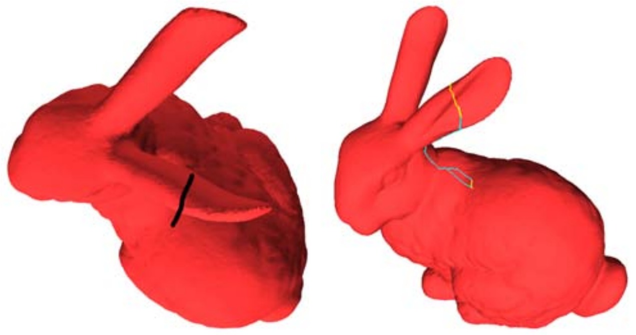
For all models in the test database, we executed a series of preprocessing steps. First, we converted all file formats to VRML 2.0 and PLY to ease processing by subsequent steps. Then, we analyzed the scene graph structure of the VRML files to create a default set of segments. This step took 8 seconds per model and produced 15.4 parts on average. Second, we extracted text keywords for each model and built an index for them, which took 0.3 seconds per model. Third, we computed shape descriptors for every model in the database, which took less than 2 seconds, on average, per model. Finally, we produced a 160×120 “thumbnail” image of each object, which took 4 seconds on average. The total preprocessing time was about 50 hours, and the cumulative storage of all data on disk requires 20 gigabytes.

7.4 Limitations

Our system is a first prototype and has several limitations. First, it includes a small subset of commands that normally would be found in a 3D modeling system. Specifically, we provide only two commands for creating faces from scratch (insert cube and join parts) and only two commands for moving vertices (affine transformation and blend). This particular design decision was made to test modeling by example in its purest form. While it adds to the simplicity of the system, it limits the types of objects that can be created. For example, without surface deformation tools, it is difficult to put the door of a Ford onto a Chevrolet. We expect that others implementing systems based on this approach in the future will include a more complete set of modeling features.

Second, it allows segmentation and attachment of parts only along existing vertices and edges of their meshes (our algorithms do not split triangles adaptively), which can prevent certain operations on coarsely tessellated models (e.g., splitting a table in half when the original mesh contains only one or two large polygons for the entire tabletop). While edges of the mesh usually occur along seams in man-made objects, and meshes representing natural objects are often highly tessellated, we plan to extend our system to allow splitting and merging triangles in the future.

Third, our intelligent scissoring algorithm is not well-suited for all types of segmentation tasks. For instance, if a surface is occluded from all viewpoints, the user cannot paint on it. Although we do provide a "laser" mode in which all surfaces under the mouse get cut (Figure 3), we think other user interface metaphors would be better (e.g., [Owada et al. 2003]). Similarly, our painting interface metaphor can produce unexpected results when the user paints over a silhouette boundary. The problem is that our system makes sure to cut through every part of the user's stroke, which may connect points adjacent in screen space but distant on the surface (Figure 9).



(a) User Stroke (b) Proposed Cut

Figure 9: The intelligent scissors algoritm ensures that the cut contour (blue and yellow line on right) visits all regions painted by the user (left), which may be problematic when the stroke crosses an interior silhouette boundary. Yellow portions of the cut are painted, and blue ones are not.

Fourth, our part-in-whole shape matching method is only able to find parts if they reside in the same relative location within their respective whole objects. Consequently, it is not able to find all possible instances of a part in the database (e.g., a bell in a church does not match a bell on a fire engine). However, in many cases, the geometric context of a part within the whole object is the defining feature of a part - e.g., searching with the wheel of an airplane selected retrieves airplanes with wheels, and not all objects that have disk shaped parts (plates, frisbees, round table tops, etc.). Further research is required to characterize the space of partial shape matching problems and algorithms.

Finally, our system provides little user feedback about how its segmentation, matching, alignment, and attachment algorithms work, and thus the user can only hit the "undo" button and try again when the system produces an unexpected result. In the future, we plan to investigate how to include fine-grained controls for expert users to handle difficult cases while maintaining a simple look and feel for novice users.

8 Results

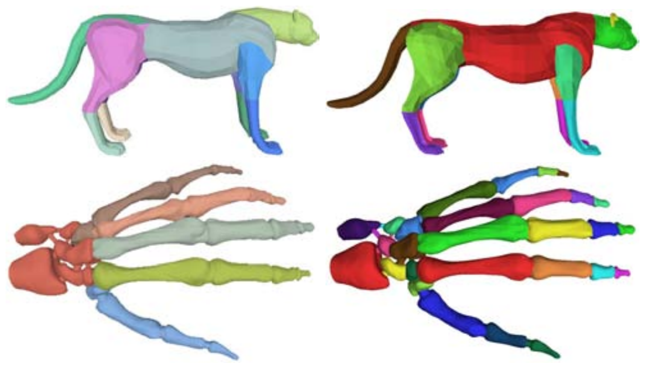
In this section, we evaluate the main research contributions of the paper, comparing our results to previous work where possible. We first present results of experiments with each component of our system, and then we show models created for a number of different applications (Section 8.4).

8.1 Scissoring Results

In order to help the reader evaluate our intelligent scissoring algorithm, we show several segmentations produced with our algorithm and compare the resulting cuts and processing times with respect to previous interactive and automatic systems.

Figure 10 shows screenshots from a session (left-to-right) during which the user segmented a statue of Mercury into parts using intelligent scissors. Each image shows a stroke drawn by the user and its segmentation result. Note how all the strokes are approximate, and yet the cuts lie along natural seams of the 3D model. Note also that every cut was made with a single stroke from the same camera viewpoint. This entire segmentation sequence can be performed in under one minute with our system. The same segmentation takes over 8 minutes with our implementation of the method described in [Gregory et al. 1999; Zöckler et al. 2000].

Figure 11 compares our intelligent scissoring results for a cheetah and a hand with results reported by [Katz and Tal 2003]. While this comparison is not altogether fair, since our algorithm is interactive and theirs is automatic, it highlights the main disadvantage of automatic approaches: they do not understand semantics. For instance, the cheetah segmentation produced by [Katz and Tal 2003] includes portions of the animal's back with the tail and neck and contains an unnatural boundary between the right-hind leg and body (Figure 11a). As a result, the parts cannot be simply pasted into another model without re-cutting them. Similarly, their hand segmentation does not separate all the bones. In contrast, our interactive approach allows users to cut the models quickly along semantic boundaries (only as needed). Our segmentation of the hand (Figure 11d) took 13 minutes (of user time), while the automatic segmentation (Figure 11c) took 28 minutes (of computer time) for the same model [Katz and Tal 2003]. Our segmentation of the cheetah took under thirty seconds.



[Katz et al, 2003] Intelligent scissors

Figure 11: Comparison of segmentations produced by the automatic algorithm of [Katz et al, 2003] (left) and the interactive intelligent scissors algorithm described in this paper (right) for a model of a cheetah and a hand (654,666 polygons). Note that the cuts are better aligned with the semantic seams of the object with intelligent scissors.

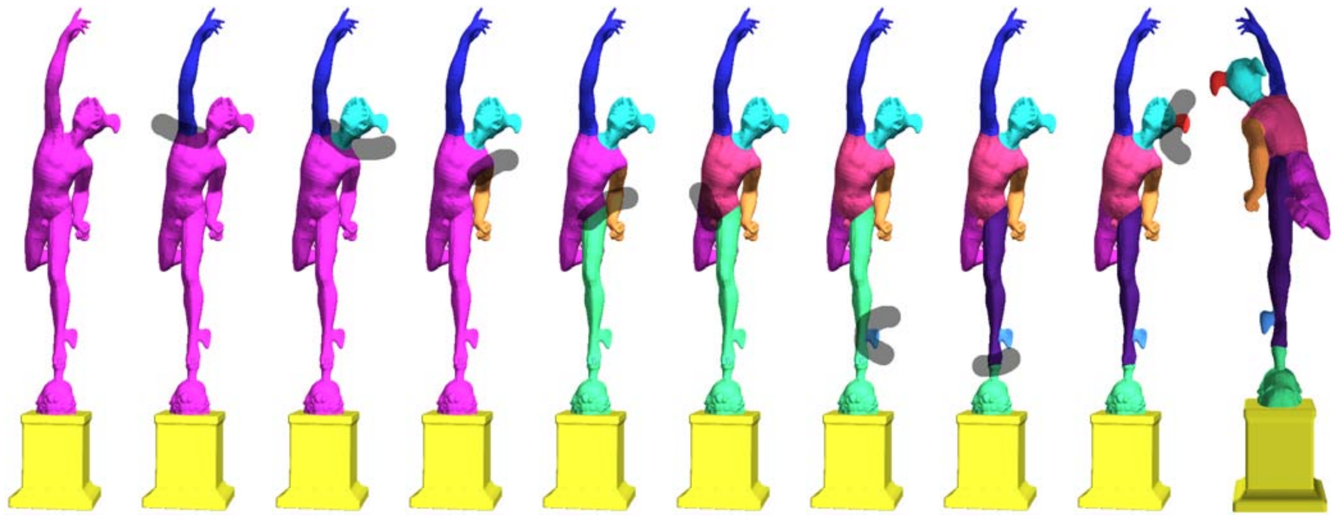


Figure 10: Example session in which a user segmented a model with intelligent scissoring. Gray strokes show the user's input. Different colored surfaces indicate separate parts computed by the intelligent scissoring algorithm. Note how the cuts between parts are smooth, well-placed, and encircle the entire object, even though the user's strokes were not very careful and all performed from the same viewpoint (the rightmost image shows the cuts as seen from the backside of the mesh). This sequence took under a minute with our system.

8.2 Search Results

In order to measure the empirical value of the shape descriptor presented in Section 5, we designed experiments aimed at evaluating the retrieval performance of the descriptor in whole-object and part-in-whole matching.

For the whole-object matching experiment, we used the Princeton Shape Benchmark test data set [Shilane et al. 2004] to compare the matching performance of our new descriptor with two state-of-the-art descriptors: the Spherical Harmonic Representation of the Gaussian Euclidean Distance Transform [Funkhouser et al. 2003] and the Radial Extent Function [Vranic 2003]. We evaluated the performance of retrieval using a precision vs. recall plot, which gives the accuracy of retrieval as a function of the number of correct results that are returned. The results of the experiment are shown in Figure 12. Note that the precision achieved with the sum of squared distances is higher than the precision of the other two methods at every recall value. This result suggests that our descriptor is well suited for whole-object matching.

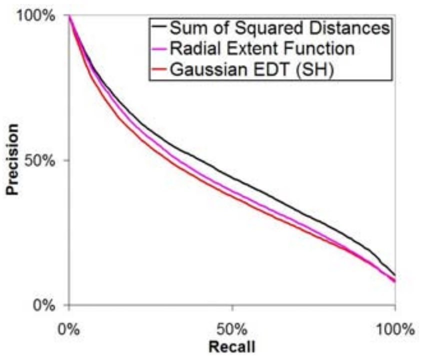


Figure 12: Whole-object matching experiment: Precision versus recall curves showing that our sum of squared distances shape matching method (black curve) compares well with two state-of-the-art methods for matching whole objects.

For the part-in-whole matching experiment, we compared the retrieval performance of feature weighted matching with whole-object matching (unweighted) using our sum of squared distances framework. Specifically, we gathered a database of 116 chairs from the Viewpoint Household Collection and labeled their pre-segmented parts (scene graph nodes) according to membership in one of 20 classes (e.g. "T-Shaped Arms", "Solid Back", etc.). There were 473 parts. We then ran two tests in which we would query with a model and, for every part in that model, measure how quickly the method would retrieve other models having parts within the same class. During the first test, we queried with the whole-object descriptor (unweighted). During the second, we weighted the descriptor ($w = 10$) to emphasize the query part. As before, we measured retrieval performance with precision-recall analysis. The results are shown in Figure 13. They show that feature weighting can help our matching algorithm retrieve models with specific parts, and suggest that the user's ability to selectively specify parts will, in general, enhance her ability to retrieve models with similar components.

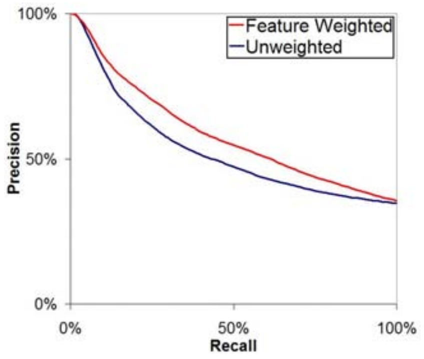


Figure 13: Part-in-whole matching experiment: Precision vs. recall curves showing that feature weighted matching (red curve) performs better than unweighted whole-object matching (blue curve) for finding chairs with particular types of parts.

8.3 Alignment Results

In order to measure the quality of our optimal alignment approach (Section 6), we ran a test to compare its performance with a traditional method (PCA) that aligns a pair of models by aligning their principal axes and resolves directions of the principal axes with heaviest axis flips [Elad et al. 2001].

For our test, we gathered 326 models and decomposed them into 45 classes. A human manually aligned all the models within a class in a common coordinate frame. We then computed the aligning rotations for every pair of models within the same class using both PCA and our optimal alignment method. We measured error as the angle of the obtained rotation, where zero angle indicates the identity rotation – a perfect match of the human alignment.

Figure 14 shows results of the experiment, expressed in terms of cumulative rotational error. For each angle of rotation α, the graph shows the percentage of models that were correctly aligned within α degrees. We show curves for the PCA method (blue) and our algorithm (red). Note that the optimal alignment approach provides much better alignments, closely matching the human's for the most model pairs. For example, if we consider alignments within 10 degrees of the human's, we find that PCA aligns only 14% of the models correctly, while the optimal alignment method aligns 75%.

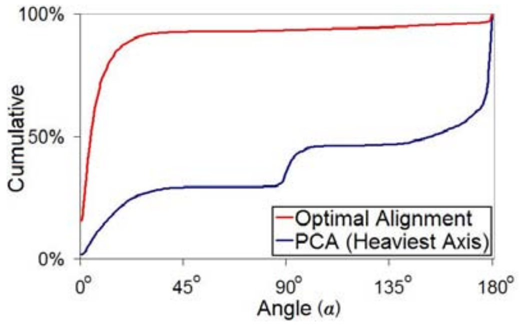


Figure 14: A plot showing the percentage of models aligned within a certain tolerance angle, α (horizontal axis). The graph shows results for PCA alignment with heaviest axis flip and our optimal alignment. Note that for a small error tolerance (e.g., within 10°), the optimal alignment algorithm succeeds for a much greater percentage of the models.

8.4 Applications

Our final challenge is to evaluate whether the "modeling by example" approach is useful: "Is it indeed easier and quicker to make interesting models with our prototype than with other systems?" Of course, this question is difficult to answer without a formal user study, which is beyond the scope of this paper. In lieu of that, we show several models created with our system by a user with no prior experience with any 3D modeling tool (Figures 15-19). For most examples, we show a photograph depicting the user's target (above/left) and describe the process by which the user proceeded to make the model. Our goal is to demonstrate that it is possible to construct new models from existing parts quickly and easily for several applications:

- **Virtual Worlds:** Figure 15 shows a 3D model created to mimic a person's home office. The user was presented with the photograph on the top and given the task of making a virtual replica. The resulting model (shown on the bottom) contains 143,020 polygons and took one hour to construct. While it is not an exact copy, we are not aware of any other way that novice users could create such a model so quickly.

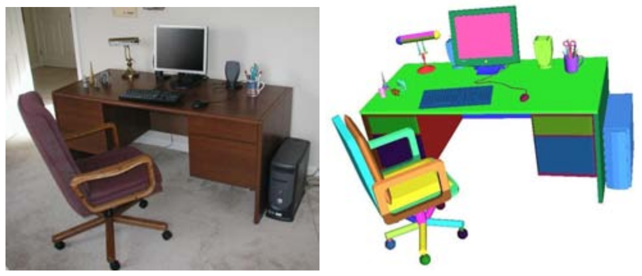


Figure 15: Virtual mockup of a home office. The user was presented with a photograph (top) and asked to make a virtual replica (bottom). Colors are used to visualize how the scene is decomposed into parts from the database.

- **Education:** Motivated by a fifth grade assignment after a field trip to the A.J. Meerwald, a schooner in the National Register of Historic Places, a user created a virtual replica mimicking what a child could have done with our system. Figure 16 shows an image of the real schooner (top-left), the resulting model (middle), and some of the parts that were used to construct it (around). Note that no boat in our database has more than one or two parts in common with our final result. This model took 90 minutes to construct, and yet contains details far beyond what our user could have created in any other tool without significant training (Figure 16).

- **Entertainment:** Figure 19 shows a 3D model of the Classic Radio Flyer Tricycle that could be incorporated in a computer game or digital video sequence. In this case, the closest tricycle in our database (left) had only one part in common with the Radio Flyer (the back plate). It was used mainly as a template for searches and alignment. All other parts are new, sometimes found in the least expected places. For instance, the curved bar connecting the handle bars to the rear wheels was extracted from the handle bar of a motorcycle, and the tassels hanging from the handgrips were cut from the string of a balloon.

- **Digital Mockup:** In several fields, it is common for people to build a quick mockup by combining parts from existing objects. Motivated by a police sketch application, a user decided to experiment with this approach for 3D modeling of faces. The result (shown in the top-row of Figure 17) is a plausible face created in 15 minutes. It combines five different parts (color-coded in the left-most image of the top row), highlighting the value of our intelligent scissoring, alignment, and composition methods. Note that the final face bears little resemblance to any of the faces from which parts were extracted.

- **Art:** Motivated by a drawing of science fiction artist Shannon Dorn (upper left of Figure 18), a user created a 3D model of a flying centaur with the upper body of a woman, the body of a horse, and wings from a bird. This example, which could be included in a video game or animation sequence, leverages all the features of our system. Yet, it also demonstrates a drawback – we do not provide the deformation tools required to adjust the horse's rear legs to match the pose in the image. This is an issue for future work.

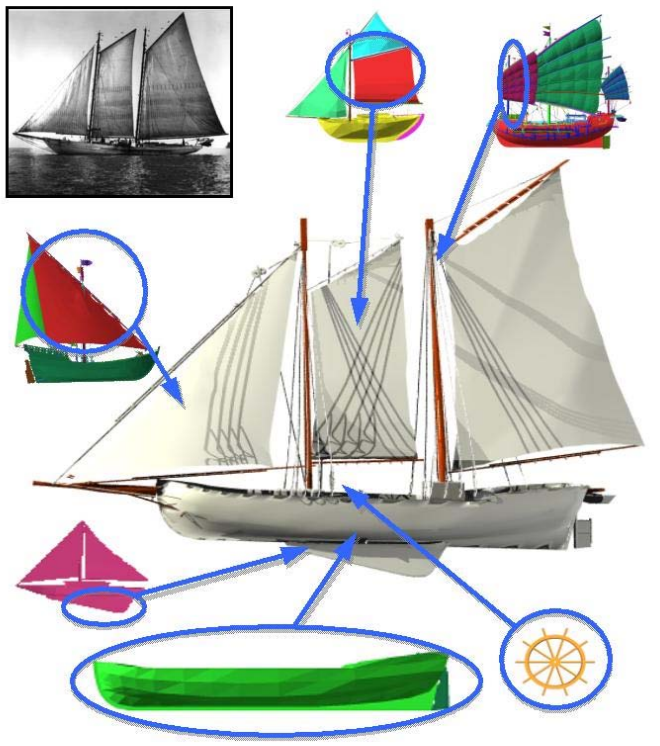


Figure 16: **A.J. Meerwald:** Creation of this model began with a text search for "ship," which produced a row boat (green, bottom) that was used for the hull. Then, to create the sail, a box was inserted and anisotropically scaled to approximate a sail over the hull, and a part-in-whole shape search resulted in the triangular sail (red, left), which was automatically aligned with the box. The keel was found through a similar shape search with a box positioned below the hull. The model with a good keel (pink, bottom left) was one big triangle mesh, so the keel was cut with the intelligent scissors. Searching with the entire shape as the query resulted in sailboats that provided the other sails and masts (top center and right). The steering wheel (yellow, bottom right) was found through a text search. The photograph shown in the top-left is courtesy of The Bayshore Discovery Project.

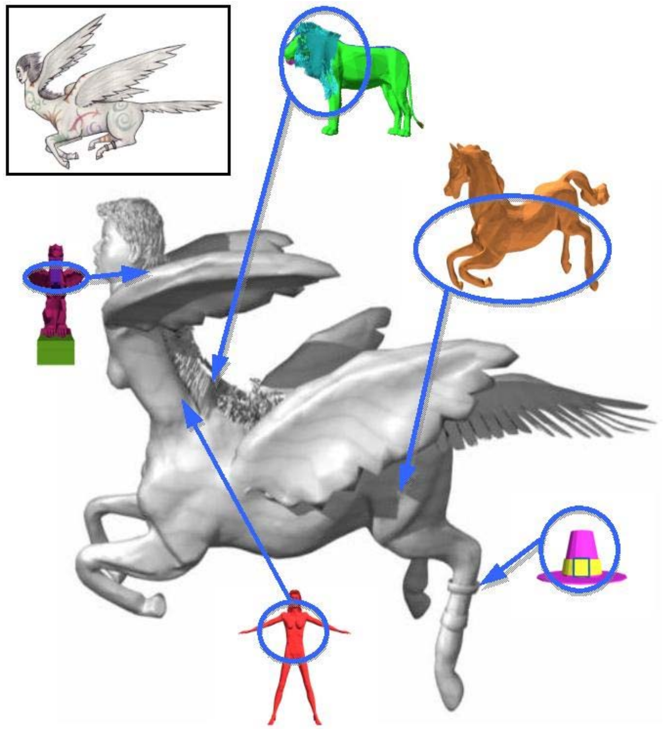


Figure 18: **Sagittarius:** We began with a text search for horse and wings. The wings were cut off of a griffin (purple, left) with the intelligent scissors, then joined to the body of the horse (yellow, top right). The head was cut off the horse, and its body was joined to the upper torso of a woman (red, bottom). Wings were also joined to the woman at the shoulder. The horse's mane came from a lion (green, top). The rings on the horse's leg were produced by searching for the term belt and selecting one off of a Pilgrim's hat (right bottom). The image in the top-left is courtesy of Shannon Dorn.

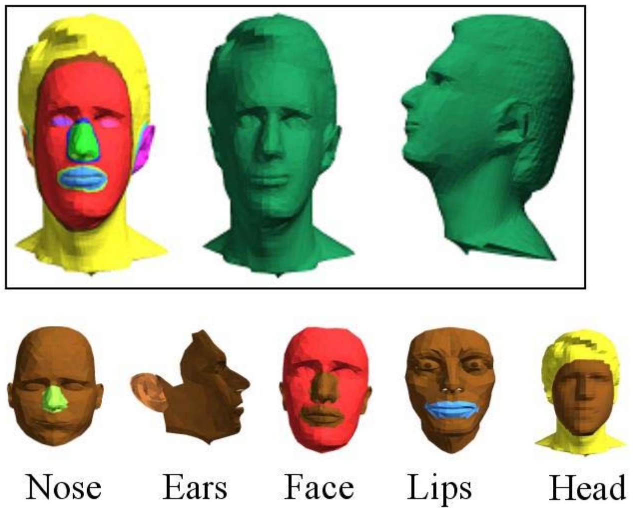


Figure 17: **Police Sketch:** Here we show the 3D model of a fictitious person's head (green) created by combining parts from other heads in the database (top-left image is color-coded by parts). Shape-based search was used to find the different models from which the lips, nose, ears, face, and head were selected. All parts were cut with the intelligent scissors and joined with blending to create a smooth, watertight model.

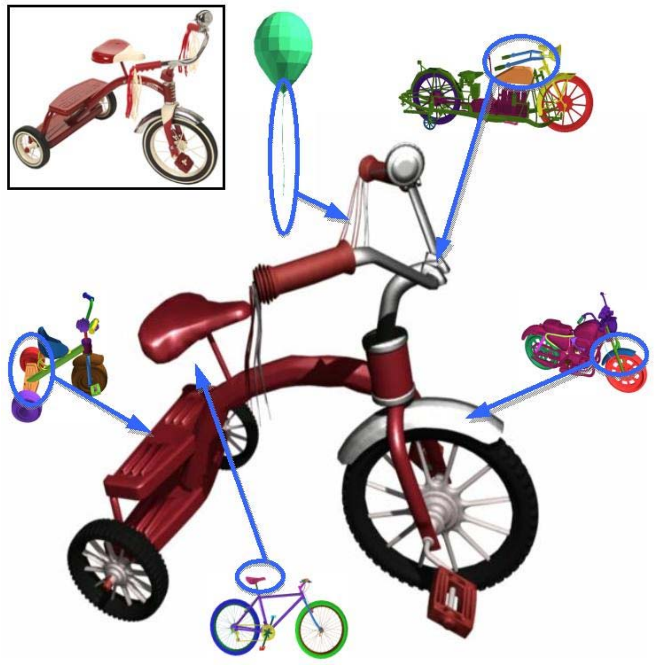


Figure 19: **Radio Flyer Tricycle:** Making this model began with a text search for "tricycle," which produced one tricycle model (left) which was used as a template. Eventually, all parts of that model were replaced except for the back plate (yellow). A bicycle seat (bottom) became the seat of the tricycle. The wheel guard came from a motorcycle (right). The handle bar region was selected and used in a shape based search, which generated motorcycle handle bars (top) that more closely matched the Radio Flyer's shape. The tassels came from the string of a balloon (top). The image in the top-left is courtesy of Radio Flyer, Inc.

9 Conclusions and Future Work

In this paper, we investigate modeling by example, a new paradigm for creating 3D models from parts extracted from a large database. We present algorithms for segmenting models into parts interactively, finding models in a database based on the shapes of their parts, and aligning parts automatically. Experience with our prototype system indicates that it is easy to learn and useful for creating interesting 3D models.

While this paper takes a small step down a new path, there are many avenues for future work. An immediate area of possible research is to investigate how sketching interfaces can be combined with our approach to produce new modeling tools for novices. In our current system, users can "sketch" new geometry only with boxes. In the future, it would be interesting to investigate more powerful sketch interfaces and study their interaction with databases of models. We imagine many types of new tools in which a user sketches coarsely detailed geometry and then the system suggests replacement parts from the database.

Another related topic of possible research is to investigate which sets of 3D modeling commands are best suited for novice users. In our system, we use a database to simplify the user interface. This leads us to ask in what other ways 3D modeling systems can become simpler while retaining or replacing their expressive power as much as possible. This is a big question whose answer will have impact on whether 3D models become a media for the masses in the future.

Acknowledgements

We would like to thank Josh Podolak, Benedict Brown, Paul Calamia, Nathaniel Dirksen, Joseph Mitchell, and the Tiggraph reviewers, who all provided assistance in the preparation of this paper. Viewpoint, Cacheforce, and Jose Maria De Espona donated commercial databases of polygonal models. The National Science Foundation funded this project under grants CCR-0093343, CCR-03-06283, IIS-0121446, and DGE-9972930. Ayellet Tal and Patrick Min were supported in part by the Information Society Technologies "AIM@SHAPE" Network of Excellence Grant (506766) by the Commission of the European Communities.

References

ALEXA, M. 2001. Local control for mesh morphing. In *Shape Modeling International*, 209–215.

BESL, P. J., AND JAIN, R. C. 1985. Three-dimensional object recognition. *Computing Surveys 17*, 1 (March), 75–145.

BESL, P., AND MCKAY, N. 1992. A method for registration of 3D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14*, 2, 239–256.

CHEN, D.-Y., OUHYOUNG, M., TIAN, X.-P., AND SHEN, Y.-T. 2003. On visual similarity based 3D model retrieval. *Computer Graphics Forum*, 223–232.

COHEN, M., 2000. Everything by example. Keynote talk at Chinagraphics 2000.

CORNEY, J., REA, H., CLARK, D., PRITCHARD, J., BREAKS, M., AND MACLEOD, R. 2002. Coarse filters for shape matching. *IEEE Computer Graphics & Applications 22*, 3 (May/June), 65–74.

CURTIS, G. 2003. *Disarmed: The Story of the Venus de Milo*. Alfred A. Knopf.

ELAD, M., TAL, A., AND AR, S. 2001. Content based retrieval of VRML objects - an iterative and interactive approach. In *6th Eurographics Workshop on Multimedia 2001*.

FUNKHOUSER, T., MIN, P., KAZHDAN, M., CHEN, J., HALDERMAN, A., DOBKIN, D., AND JACOBS, D. 2003. A search engine for 3D models. *Transactions on Graphics 22*, 1, 83–105.

GREGORY, A., STATE, A., LIN, M., MANOCHA, D., AND LIVINGSTON, M. 1999. Interactive surface decomposition for polyhedral morphing. *Visual Comp 15*, 453–470.

HORN, B., HILDEN, H., AND NEGAHDARIPOUR, S. 1988. Closed form solutions of absolute orientation using orthonormal matrices. *Journal of the Optical Society 5*, 1127–1135.

HORN, B. 1987. Closed form solutions of absolute orientation using unit quaternions. *Journal of the Optical Society 4*, 629–642.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3D freeform design. In *Proceedings of SIGGRAPH 1999*, Computer Graphics Proceedings, Annual Conference Series, ACM, 409–416.

KANAI, T., SUZUKI, H., MITANI, J., AND KIMURA, F. 1999. Interactive mesh fusion based on local 3D metamorphosis. In *Graphics Interface*, 148–156.

KATZ, S., AND TAL, A. 2003. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics (TOG) 22*, 3, 954–961.

KAZHDAN, M. 2004. *Shape Representations and Algorithms for 3D Model Retrieval*. PhD thesis, Department of Computer Science, Princeton University.

KOSTELEC, P., AND ROCKMORE, D. 2003. FFTs on the rotation group. Tech. Rep. 03-11-060, Santa Fe Institute's Working Paper Series.

LAZARUS, F., AND VERROUST, A. 1998. 3d metamorphosis : a survey. *The Visual Computer 14*, 8-9.

LEE, Y., AND LEE, S. 2002. Geometric snakes for triangular meshes. *Computer Graphics Forum (Eurographics 2002) 21*, 3, 229–238.

LEE, J., CHAI, J., REITSMA, P., HODGINS, J., AND POLLARD, N. 2002. Interactive control of avatars animated with human motion data. *Proceedings of SIGGRAPH 2002*, 491–500.

LONCARIC, S. 1998. A survey of shape analysis techniques. *Pattern Recognition 31*, 8, 983–1001.

MIN, P., HALDERMAN, J., KAZHDAN, M., AND FUNKHOUSER, T. 2003. Early experiences with a 3D model search engine. In *Proceeding of the eighth international conference on 3D web technology*, 7–18.

MIN, P. 2004. *A 3D Model Search Engine*. PhD thesis, Department of Computer Science, Princeton University.

MITCHELL, J., 2003. personal communication.

MUSETH, K., BREEN, D., WHITAKER, R., AND BARR, A. 2002. Level set surface editing operators. *Proceedings of SIGGRAPH 2002*, 330–338.

OWADA, S., NIELSEN, F., NAKAZAWA, K., AND IGARASHI, T. 2003. A sketching interface for modeling the internal structures of 3D shapes. In *3rd International Symposium on Smart Graphics*, Lecture Notes in Computer Science, Springer, 49–57.

PAQUET, E., AND RIOUX, M. 1997. Nefertiti: A query by content software for three-dimensional models databases management. In *International Conference on Recent Advances in 3-D Digital Imaging and Modeling*.

RUBNER, Y., TOMASI, C., AND GUIBAS, L. J. 2000. The earth mover's distance as a metric for image retrieval. vol. 40, 99–121.

SHILANE, P., MIN, P., KAZHDAN, M., AND FUNKHOUSER, T. 2004. The princeton shape benchmark. In *Shape Modeling International*.

SLOAN, P., ROSE, C., AND COHEN, M. 2001. Shape by example. *Symposium on Interactive 3D Graphics* (March), 135–143.

SOFT 1.0, 2003. `www.cs.dartmouth.edu/ ~geelong/soft/`.

SUZUKI, M. T. 2001. A web-based retrieval system for 3D polygonal models. *Joint 9th IFSA World Congress and 20th NAFIPS International Conference (IFSA/NAFIPS2001)* (July), 2271–2276.

TANGELDER, J., AND VELTKAMP, R. 2003. Polyhedral model retrieval using weighted point sets. In *Shape Modeling International*.

TANGELDER, J., AND VELTKAMP, R. 2004. A survey of content based 3d shape retrieval methods. In *Shape Modeling International*.

VRANIC, D. V. 2003. An improvement of rotation invariant 3D shape descriptor based on functions on concentric spheres. In *IEEE International Conference on Image Processing (ICIP 2003)*, vol. 3, 757–760.

WAVEFRONT, A., 2003. Maya. http://www.aliaswavefront.com.

WONG, K. C.-H., SIU, Y.-H. S., HENG, P.-A., AND SUN, H. 1998. Interactive volume cutting. In *Graphics Interface*.

ZELEZNIK, R., HERNDON, K., AND HUGHES, J. 1996. Sketch: An interface for sketching 3D scenes. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, 163–170.

ZÖCKLER, M., STALLING, D., AND HEGE, H. 2000. Fast and intuitive generation of geometric shape transitions. *Visual Comp 16*, 5, 241–253.