RigMesh: Automatic Rigging for Part-Based Shape Modeling and Deformation

Péter Borosán[1]* Ming Jin[1]† Doug DeCarlo[1] Yotam Gingold[1,2] Andrew Nealen[1]
[1]Rutgers University [2]Columbia University

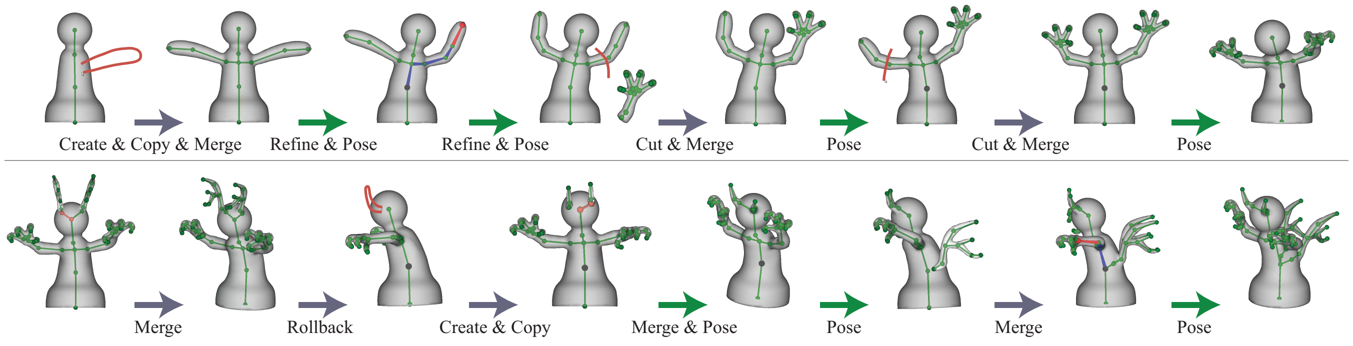

Figure 1: *An example RigMesh modeling sequence: the artist experiments with different ideas for a shape by modeling (blue arrows) and posing (green arrows) using our skeletal rig. In the traditional pipeline, modeling and rigging are separate procedures, so the model must be re-rigged every time the shape changes (i.e. after each modeling operation). RigMesh unifies modeling and rigging; the model is always rigged, and the artist can pose freely, allowing for iterative modeling, deformation, and key-frame animation with real-time response.*

Abstract

The creation of a 3D model is only the first stage of the 3D character animation pipeline. Once a model has been created, and before it can be animated, it must be *rigged*. Manual rigging is laborious, and automatic rigging approaches are far from real-time and do not allow for incremental updates. This is a hindrance in the real world, where the shape of a model is often revised after rigging has been performed. In this paper, we introduce algorithms and a user-interface for sketch-based 3D modeling that unify the modeling and rigging stages of the 3D character animation pipeline. Our algorithms create a rig for each sketched part in real-time, and update the rig as parts are merged or cut. As a result, users can freely pose and animate their shapes and characters while rapidly iterating on the base shape. The rigs are compatible with the state-of-the-art character animation pipeline; they consist of a low-dimensional skeleton along with skin weights identifying the surface with bones of the skeleton.

Keywords: rigging, skeletonization, skinning, animation, sketch-based modeling

Links: DL PDF

*e-mail: pborosan@cs.rutgers.edu
†e-mail: mjin@cs.rutgers.edu
Péter Borosán and Ming Jin contributed equally to this work and are placed in alphabetical order.

1 Introduction

The task of creating ready-to-animate 3D models is fundamentally hard; designers and artists spend years becoming proficient at commercial, state-of-the-art tools such as Maya and 3ds Max [Autodesk 2012c; Autodesk 2012a]. In an effort to make freeform 3D modeling more accessible to novices and to enable rapid iterative prototyping, sketch-based tools such as Teddy [Igarashi et al. 1999] and a large body of follow-up work [Tai et al. 2004; Cherlin et al. 2005; Schmidt et al. 2005; Alexe et al. 2005; Karpenko and Hughes 2006; Nealen et al. 2007; Bernhardt et al. 2008; Sugihara et al. 2008; Pihuit et al. 2010; Cordier et al. 2011] introduced techniques to create plausible 3D models from 2D freeform strokes. (See [Olsen et al. 2009] for a recent survey.) While these tools greatly simplify shape modeling, modeling is only the first stage of the 3D animation pipeline. Once a model has been created, and before it can be animated, it must be *rigged*. In the state-of-the-art character animation pipeline, a rig takes the form of a *skeleton*, a cycle-free graph whose nodes are called joints and whose edges are called bones, and *skin weights* identifying the surface of the model with the bones of the skeleton [Magnenat-Thalmann et al. 1988; Lewis et al. 2000]. Rigging can be performed manually, by designing the skeleton and then laboriously painting the surface with skin weights for each bone. Various computational methods exist for automatic skeleton extraction [Sharf et al. 2007; Cornea et al. 2007; Au et al. 2008; Pan et al. 2009] and skinning [Lewis et al. 2000; Bloomenthal 2002; Kry et al. 2002; Mohr and Gleicher 2003; Weber et al. 2007; Baran and Popović 2007; Wang et al. 2007; Yang and Wünsche 2010; Miller et al. 2010].

In this paper, we take advantage of these recent advances in the literature, and further simplify the process of creating ready-to-animate 3D models by combining shape creation, modification and posing into a single coherent framework. However, this cannot be achieved trivially by integrating existing tools. Sketch-based modeling tools like Teddy [1999] and FiberMesh [2007] do not explicitly maintain a skeleton. Automatically adding the skeleton (e.g. [Au et al. 2008]) and rigging (e.g. [Baran and Popović 2007]) as a post-process typically does not allow for real-time interactions. Furthermore, in our discussions with professional animators [Shukan 2012], we have learned that they often need to revise the geometry of their models after they have been rigged (Fig. 1 illustrates this process). Such incremental updates to the shapes are especially difficult with the aforementioned tools. We believe the fundamental problem lies in the conventional pipeline where modeling and rigging are treated as two entirely separate processes. Shapes are modeled before they can be rigged; if the base shape requires any modifications at a later time, the skeleton has to be re-computed, and the rig re-designed.

Thus, in contrast with the traditional, sequential view of modeling and rigging, we propose to unify these steps and introduce algorithms for sketch-based modeling by parts that **maintain a rigged mesh at all times**. Our algorithms generate a rigged mesh—a surface, a skeleton, and skin weights—for each sketched part and update it as parts are merged or cut. *Modeling by parts* is consistent with human recognition of shapes by components [Biederman 1987], and has been employed by modeling approaches such as ShapeShop [Schmidt et al. 2005] for CSG operations on individual sketched parts; Gingold et al. [2009] for single-view modeling from existing sketches; and the SPORE Creature Creator [Maxis 2008], as well as SnapPaste [Sharf et al. 2006], Shuffler [Kraevoy et al. 2007], meshmixer [Schmidt and Singh 2010], the works of Chaudhuri et al. [2011], and Funkhouser et al. [2004] for re-using parts of existing models.

By unifying modeling and rigging, artists can freely pose the shape while modeling to adjust it, or to examine the motion of the model by using poses as key-frames. Furthermore, the unification removes friction from incremental and iterative character modeling and animation (Fig. 1). Our algorithms are efficient and execute in real-time, far below the one second delay that interrupts mental flow [Nielsen 1993]. Our work builds on Pinocchio [Baran and Popović 2007], adapting its skin weight algorithm to support incremental merging and cutting of shapes, and revisits the Chordal Axis Transform [Prasad 1997] used by Teddy [Igarashi et al. 1999] for skeleton generation. Specifically, our contributions are:

- A unified approach to modeling and rigging, eliminating a recurring step from the professional 3D animation pipeline.
- A modification of the Douglas-Peucker algorithm [1973] for creating a high-quality skeleton from the sketched outline of a shape.
- An efficient method for locally re-computing skin weights to maintain a valid, high-quality rig when merging or cutting shapes.

Note that a variety of hierarchical approaches to smooth surface modeling exist in the literature, such as the seminal BlobTrees [Wyvill et al. 1999] and aforementioned ShapeShop [Schmidt et al. 2005], as well as other approaches based on convolution surfaces [Sherstyuk 1999; Alexe et al. 2005; Allègre et al. 2006]. While a convenient representation for modeling and animation, CSG-based hierarchies are quite different from the rigged models preferred in professional character animation pipelines, which are based on surfaces, skeletons, and skin weights. Also notably, a system that shares some similarity with our work flow is the ZSpheres tool of ZBrush [Pixologic 2012], in which a base model is created by connecting spheres of varying radii with tapered cylinders. While ZSpheres allows for base model creation and posing at the same time, this base model only serves as a bone skeleton and rig, from which the surface mesh needs to be sculpted as a separate step. Furthermore, our system differs from ZSpheres in being sketch-based and by supporting modeling by parts, which enables part re-use.

2 Initial Shape Creation

To create a new shape, the user sketches its silhouette. From the sketched curve, our system generates a surface composed of smoothly joined generalized cylinders, a skeleton, and skin weights attaching the surface to the skeleton. First, the user's sketched curve is closed and treated as a planar polygon (Fig. 2 (a)). The polygon is decomposed into *cylindrical regions* with well defined local symmetry, and *connecting regions* lacking such symmetry (Section 2.1). The surface and the skeleton are then each generated (Sections 2.2 and 2.3), and, as a final step, the surface is attached to the skeleton by automatically computing skin weights (Section 2.4). These steps are all computed instantaneously as soon as the user completes the stroke by releasing the mouse button.

2.1 Decomposition and Symmetry Axes

Our use of generalized cylinders benefits from an extensive literature on 2D shape analysis. We use the Chordal Axis Transform (CAT) [Prasad 1997], and refine its result to obtain the decomposition and local symmetry axes of the polygon (Fig. 2).

The CAT relies on the Constrained Delaunay Triangulation (CDT) [Chew 1989] of the polygon. The new (internal) edges of the polygon, added by the CDT, are referred to as *chords*. Triangles defined by three chords are called *junction* triangles (green triangles in Fig. 2 (b)). Local symmetries of the polygon are approximated by the *chordal axis* — a polyline connecting the chord midpoints. As in Prasad [1997], we prune the chordal axis to remove *insignificant branches* — features of the chordal axis not contributing significantly to the characterization of the polygon. After pruning, we apply Laplacian smoothing to both the extracted chordal axis and the orientation of the chords.

The result of these steps is a decomposition of the polygon with smooth approximate symmetry axes ready for generating the surface and the skeleton. Note that junction triangles connect three parts of a polygon that have local symmetry, so we consider them to

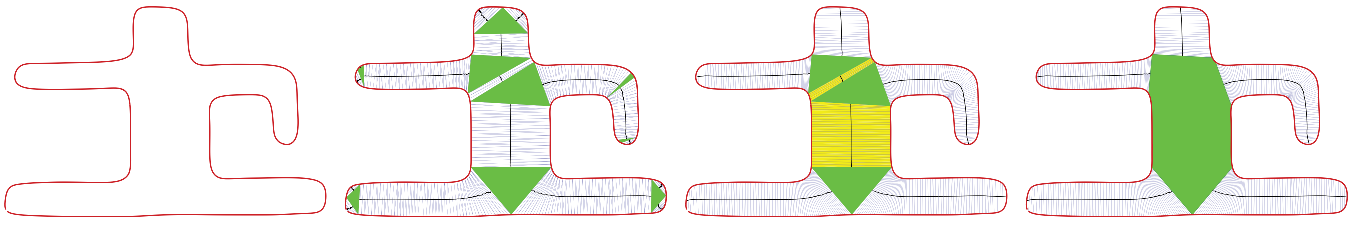

Figure 2: *(a) The input user sketch. (b) The CAT of the polygon in (a). Chordal axes are shown as black polylines and junction triangles are shaded green. (c) Pruned and smoothed chordal axes. Flat cylindrical regions (yellow) still exist between junction triangles. (d) The final decomposition: cylindrical regions (white) and the connecting region (green).*

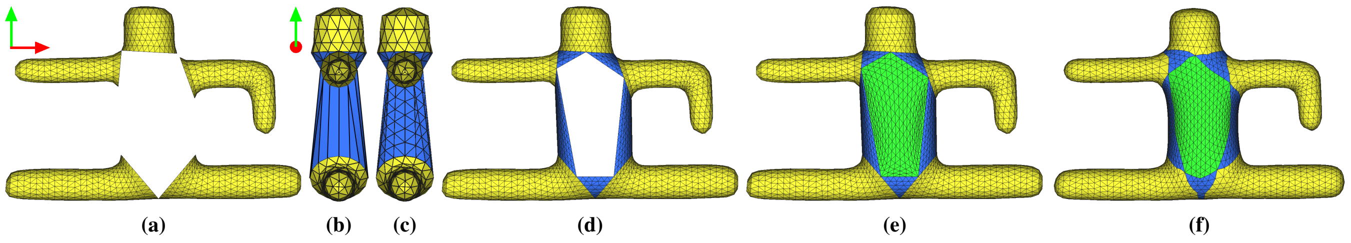
Figure 3: *Generating the 3D surface mesh. (a) Generalized cylinders (yellow) are created from cylindrical regions. (b) and (c) are side-views shown with coarser tessellation for illustration purposes. (Orientation is indicated by the red/green coordinate axes.) (b) The semi-circles of neighboring generalized cylinders are connected by lines. (c) Uniformly sampled points along these lines are stitched together to form triangle strips. (d) Frontal view after steps in (b) and (c). (e) Using the boundary vertices, GridMesh [Nealen et al. 2009] covers the holes (green) and provides a water-tight mesh. (f) The surface is smoothed using Least-Squares Meshes [Sorkine and Cohen-Or 2004].*

be *connecting regions*. The remainder of the polygon can be treated as *cylindrical regions*. There may also be short-yet-wide cylindrical regions between junction triangles (yellow in Fig. 2 (c)), which would lead to the creation of flat, disc-like surfaces. Such regions are merged with the neighboring junction triangles to form larger connecting regions (green in Fig. 2 (d)), alongside the elongated cylindrical regions (remainder of polygon in Fig. 2 (d)).

The use of these techniques is inspired by Teddy. However, Teddy only uses the CAT for surface inflation from a 2D silhouette and discards it afterwards; we, however, further process the chordal axis to generate the skeleton (Section 2.3).

2.2 Surface generation

To create the surface of the 3D shape, generalized cylinders are constructed from the cylindrical regions. The smoothed chordal axis is used as the cylinder axis, and circular cross sections are erected perpendicular to the sketch plane, using the smoothed chords as their respective diameters. The circumference of each cross section is uniformly sampled in arc length, and neighboring cross sections are stitched together by triangle strips. Cross sections are added at a frequency such that the maximum distance between two adjacent cross sections is as close to a predefined constant (target edge length) as possible, without the two cross sections intersecting each other. The resulting generalized cylinder surfaces are shown in Fig. 3 (a), while subsequent steps for constructing connecting regions are shown in Fig. 3 (b)–(e).

As a final step in surface generation, the entire surface is smoothed using Least-squares Meshes [Sorkine and Cohen-Or 2004] (Fig. 3 (f)): connecting region vertices (blue and green) are unconstrained, and vertices in cylindrical regions (yellow) are used as positional constraints. The weights on these positional constraint vertices are proportional to the distance (along the chordal axis) from their cross sections to the closest connecting region. In simpler terms, the farther from the (unconstrained) connecting region, the higher the weight. Connecting region vertices are unconstrained because, as junctions, it is paramount that they provide a smooth transition between cylindrical regions; moreover, they typically share only a short boundary with the user sketch. The result is a smooth, watertight mesh with well-shaped triangles.

2.3 Skeleton generation

Cylindrical regions. The (smooth) chordal axis is down-sampled into a skeleton composed of *joints* and *bones*. Intuitively, choosing the skeletal joints is similar to identifying the *salient* points of the chordal axis. The popular Douglas-Peucker algorithm (DP) [Douglas and Peucker 1973] finds such points in a greedy manner; a dynamic programming approach was proposed by Lewis and Anjo [2009].

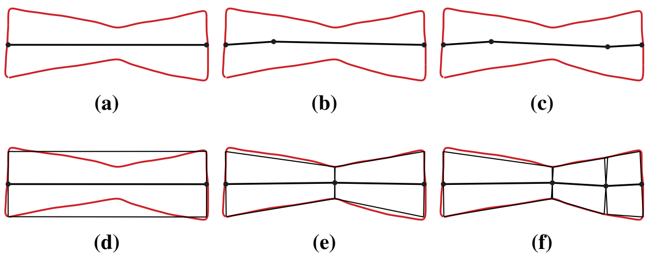
Figure 4: *Applying DP and CDP to a cylindrical part with roughly straight symmetry axis. The user sketch is shown in red. The fitted trapezoids are shown in black. Top row: the DP algorithm. Bottom row: our CDP algorithm. Each image in the left-to-right sequence shows one step in the divide-and-conquer algorithm.*

Unfortunately, simply identifying salient points along the chordal axis polyline does not suffice for determining skeletal joints in positions where users would expect them. For example, in Fig. 4, the polygon has a nearly straight chordal axis, while the cylinder thickness varies substantially along the chordal axis. Applying DP (or any other line simplification method) would result in a skeleton with only one bone (Fig. 4 (a)), while reducing the error threshold might place the joints at undesirable locations (Fig. 4 (b) and (c)) due to noise or slight perturbations of the chordal axis.

The problem with such approaches is that the measure of saliency considers only the chordal axis, rather than the entire shape. To alleviate this, we propose a modified, cylindrical version of the DP algorithm (Cylindrical Douglas-Peucker or CDP) that also considers the variation in shape thickness along the chordal axis. Recall the original DP algorithm: at each step, a range $[i, j]$ of the chordal axis $(v_1, v_2, \ldots, v_n)$ is considered. If the distance of every vertex in the range to $\overline{v_i v_j}$ (the imaginary line between the first and last vertices of the range) is less than a predefined threshold, the algorithm outputs $\overline{v_i v_j}$. Otherwise, the algorithm proceeds by dividing the range into two: $[i, m]$ and $[m, j]$, where m is the index of the vertex in the range whose distance was greatest. Fig. 5 illustrates the error measure (distance): the DP error for vertex v is e_{DP}, the distance to the line segment $\overline{v_1 v_n}$. In our proposed CDP, each line segment induces a symmetric trapezoid (the shaded quadrilateral $ABCD$), with the line segment $(\overline{v_1 v_n})$ as its symmetry axis. The bases of the trapezoid ($\overline{AB}$ and $\overline{CD}$) are deter-

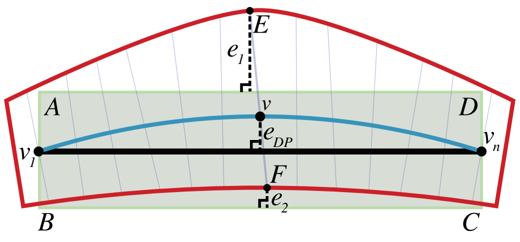

Figure 5: *Error terms for vertex v in DP and CDP algorithms. The chordal axis is shown in blue. The user sketch is shown in red. $\overline{v_1 v_n}$ is the line segment added in one step of DP; In CDP, besides $\overline{v_1 v_n}$, symmetric trapezoid ABCD (shaded) is also added to evaluate the error term.*

mined by projecting the chords passing through v_1 and v_n, the end points of the range, onto the direction perpendicular to $\overline{v_1 v_n}$. The error term for vertex v in CDP is defined as:

$$e_{CDP} = e_1^2 + e_2^2$$

where e_1 and e_2 are the distances from the two end points of the chord passing through v ($\overline{EF}$) to the two sides of the trapezoid, respectively.

Fig. 4 compares the results of DP and CDP. Note how the CDP algorithm captures the kink in the boundary of the input polygon, while the standard DP algorithm chooses potentially undesirable skeletal joints.

Connecting regions. To determine the skeletal structure of connecting regions, a joint is placed at the center of each junction triangle and a bone is added across each of its three sides (Fig. 6(a)). If a side is adjacent to a cylindrical region, the bone is connected to the end of its skeleton; otherwise, the side must be adjacent to another junction triangle, and the bone connects both junction triangles' centers. This may lead to short bones within connecting regions, resulting in a rig with redundant degrees of freedom for controlling the shape. Thus, as a last step, short bones inside "thick" regions are collapsed. Fig. 6(b) shows the final skeleton for the shape.

2.4 Skin weight computation

To create (and maintain) a rigged model, we must attach the surface to the skeleton through a process that assigns to each vertex a set of skin weights for each bone. We calculate skin weights using the heat diffusion method of Pinocchio [2007]. In this method, bones

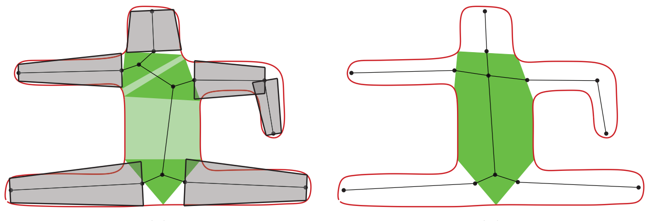

Figure 6: *Determining the skeletal structure. The connecting regions are shown in green. The skeleton is shown in black. (a) Trapezoids (gray) are fit to cylindrical regions using our CDP algorithm; each trapezoid corresponds to a bone. A joint is also added at the center of each junction triangle, and a bone is added across each of its three sides. (b) Short bones inside connecting regions are collapsed.*

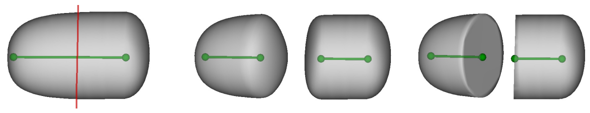

Figure 7: *Cutting an existing 3D shape into two parts. The cutting stroke is shown as a red open curve. Left: cut placement. Middle: result of a smooth cut. Right: result of a sharp cut*

radiate heat onto the surface, and the equilibrium temperature at each vertex is computed by solving a linear system of equations. The performance bottleneck for Pinocchio's skinning algorithm is in determining the closest visible bone for each vertex. In our system, for vertices on cylindrical regions, the set of closest visible bones and their distances are trivially known at creation time. Consequently, visibility and distance computations only need to be performed for vertices in connecting regions. The number of such vertices is typically small; as a result, skin weights are computed instantaneously for all of our examples.

3 Modeling Tools

Our interface includes two part-based modeling operations, cutting and merging, as well as two operations based around manipulating the rig, posing and skeleton refinement. With the aid of the rig, posing is naturally performed using forward and inverse kinematics (IK) [Watt and Watt 1991]. (We use dual quaternion skinning [Kavan et al. 2008] to compute the deformed surface.) Our algorithms for the other operations are described as follows.

3.1 Cutting

The user can perform a smooth or sharp cutting operation on the shape by drawing an open curve across the model, as shown in Fig. 7 (left). The surface is split into two parts along this curve, and GridMesh [Nealen et al. 2009] is used to generate the hole-filling patches, resulting in two watertight meshes. A smooth cutting operation creates a smooth boundary (Fig. 7 (middle)), while a sharp cut does not (Fig. 7 (right)). Vertex positions for the glued patch of surface are computed using Least-Squares Meshes [Sorkine and Cohen-Or 2004], with appropriate smooth or sharp boundary constraints.

The skeleton is also cut by the user's stroke, resulting in two independent skeletal structures. In order to maintain valid skin weights on the two resulting shapes, weights corresponding to bones no longer in a shape are removed and weights for vertices on the newly created surface patches are computed as follows. Following the heat diffusion analogy in Pinocchio, we wish to find the temperatures at the locations of the new vertices. Since temperature varies smoothly, we can compute the skin weights by diffusion. So, skin weights for the new vertices are computed by minimizing, for each bone, $\|\Delta w^i\|^2$, where w^i are the skin weights for bone i. [1] Skin weights for old vertices are kept fixed. Finally, since the skin weights at both new and old vertices may no longer sum to one, they are normalized.

3.2 Merging

A merging operation is initiated by dragging one shape into close proximity of another. Fig. 8 shows the three types of merging: In the case of *snapping*, the entire dragged shape translates so that the two joints indicated by the user (highlighted in blue) become coincident, and then the skeletons are joined. In the case of *splitting*, a

[1]The matrix factorization from the computation of vertex positions for the glued surface patch can be re-used.

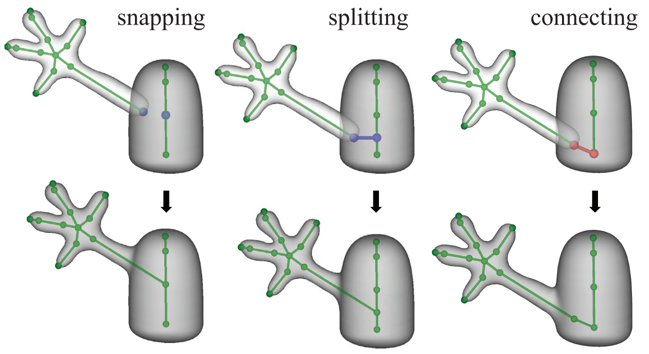

Figure 8: *Merging two existing 3D shapes. The shape being dragged has a lighter color. Top row: Before merging; Bottom row: Results of merging operations. See text for details.*

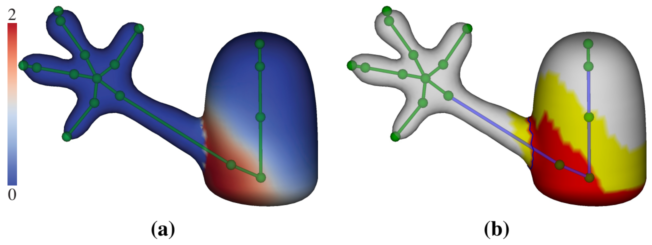

Figure 9: *Merging two shapes by inserting a bone (the connecting merge from Fig. 8). (a) Global skin weight re-computation results in a localized change. The value visualized at each vertex is the L_1 magnitude of the vector difference between skin weights of the two parts before the merge, and after the merge and global skin weight re-computation (pre-merge skin weights for any newly inserted bone are 0). (b) Bones and vertices included in our local skin weight re-computation algorithm (see text for details).*

new joint is inserted at the indicated location on the second shape's bone and then snapping is performed with this joint (the new joint and the projection line are highlighted in blue). In the case of *connecting*, a new bone (highlighted in red) is inserted to connect the two skeletons at the joints indicated by the user. After releasing the dragged shape, skeletons and surfaces are merged (Section 3.2.1). Then, we prune bones that are not the most influential for any vertex of the merged surface (i.e. bones that do not correspond to any vertex's maximum skin weight). Finally, the skin weights are updated (Section 3.2.2).

3.2.1 Merging the surfaces

When merging two shapes, our system creates a new, watertight surface from their union (Fig. 8). The intersection of two surfaces results in a closed intersection loop.[2] First, each surface's mesh is adjusted to accommodate the intersection loop, using the method from [Nealen et al. 2005]. Then, any triangles from one surface inside the other are discarded, and the resulting meshes are glued together along the shared boundary. If necessary, new vertices are created by splitting boundary edges. An example intersection loop can be seen as a blue line in Fig. 9 (b). To smooth the geometric transition between the two shapes, local Laplacian smoothing is performed around the intersection loop. The vertices included in this smoothing step are every vertex whose graph distance from the intersection loop—let v_i be the closest point on the intersection loop—is closer than $k = 0.6$ times the Euclidean distance from v_i to the intersection loop's centroid.

[2]Without loss of generality, we describe the simple case where the intersection results in a single closed loop. Our system can handle more complicated cases with multiple closed loops.

3.2.2 Local skin weight re-computation

Once two shapes' surfaces and skeletons have been merged, skin weights must be updated. While we could simply perform the same global skin weight computation as in Section 2.4, the algorithm slows down considerably as the complexity of a shape increases following a series of merge operations. We make the crucial observation that only the skin weights of vertices near the merge boundary—a small fraction of all vertices—are noticeably affected (see Fig. 9 (a)). This allows for a local skin weight re-computation that greatly accelerates the process. In this local re-computation, we run the same skinning algorithm used for the global computation (see Section 2.4) on a subset of the shape's vertices and bones.

In deciding which vertices and bones should be included, we first identify vertices for which the original (pre-merge) weights are clearly not correct anymore. We rely on the simple intuition that the greatest influence on a vertex should come from the bone closest to and visible from it. Alas, this is not always the case, since skin weights vary smoothly over the surface. Yet it is the case that a vertex's closest visible bone almost always has above average weight (taking the average over all positive bone weights associated with the vertex). This suggests the following inclusion test: for a given vertex, if the closest visible bone (or bones if there are more than one equally close) in the new shape does not have above average weight, then the vertex, and obviously this closest bone, should be included in the re-computation. We perform this test beginning with vertices along the merge boundary, expanding outward in a breadth-first manner until the test fails. Fig. 9 (b) depicts the merge boundary in blue, and vertices that pass the inclusion test in red. Finally, to ensure skin weight smoothness between recomputed and not re-computed vertices, the following additional vertices and bones are included: every vertex (and bones having higher than average weight for the vertex) whose graph distance from a red vertex v_i is less than c times the Euclidean distance between v_i and the bone closest to v_i, where $c = 1.5$. This prevents folding artifacts around joints by ensuring a transition width proportional to the distance from the joint to the surface. Fig. 9 (b) depicts these additional vertices in yellow (using $c = 0.8$ for illustration purposes) and the added bones in blue. Note that all vertices whose skin weights would be significantly affected by a global recomputation are included (Fig. 9 (a)–(b)).

To generate the new skin weights, the Pinocchio skinning algorithm is run on the included vertices and bones; the skin weights of excluded immediate neighboring vertices are used as boundary conditions. Afterwards, the updated skin weights are normalized to sum to one, and the merge is complete.

3.3 Skeleton refinement

The user can refine the skeleton by splitting an existing bone and adding a new joint along it, or by deleting a bone and merging the two adjacent joints. For the purposes of updating skin weights, splitting a bone is treated as a cutting operation followed by a merge: the shape is cut with a plane perpendicular to the bone and through the position of the new joint; the split shape is then merged back together. Deleting a bone b is handled as several cutting operations, followed by a skeleton modification, followed by a merge: the skeletal modification collapses b (its joints move to the midpoint), the cuts are through the *far* joints of all bones connected to b (and perpendicular to those bones), and the merge reattaches all the cut pieces back together again (**N.B.** every vertex and bone in collapsed-bone piece are included in the skin weight re-computation).

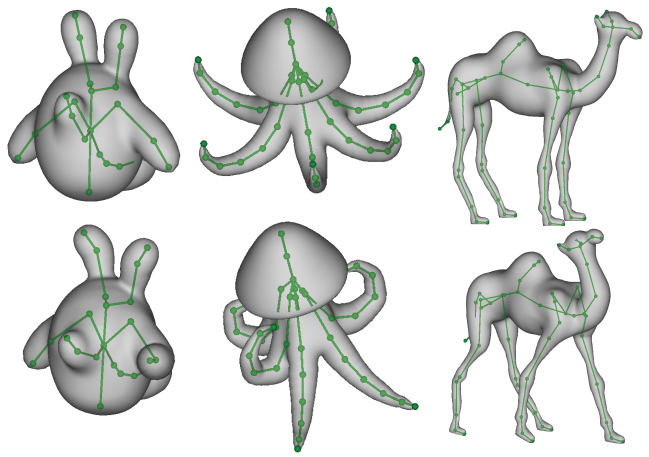

Figure 10: *Models created using RigMesh. Models took an average of 20 minutes to create. Top: result shapes; Bottom: posed shapes.*

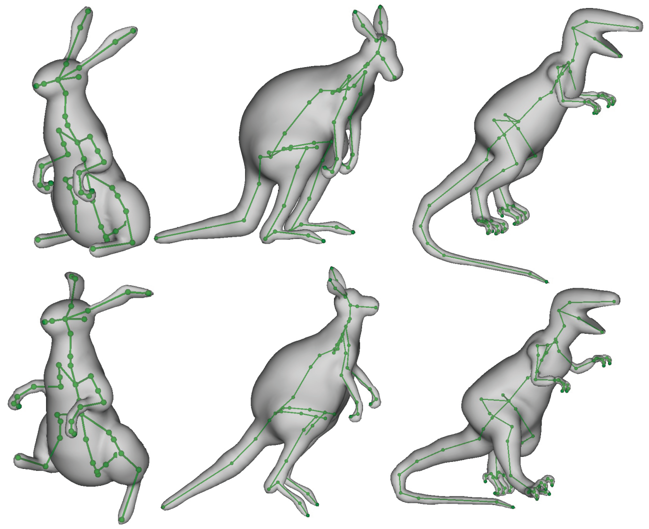

Figure 11: *More modeling results. Models were created in 30–40 minutes each. Top: result shapes; Bottom: posed shapes.*

4 Results

Our current implementation is written in C++, and was tested on a 2.8GHz Intel i7 under Windows 7. We use OpenMesh for mesh processing and CHOLMOD for numerical computations. Our modeling software runs at interactive rates (see the accompanying videos).

4.1 Modeling Results

When modeling with an ever-present rig, it is natural to interleave posing with modeling (Fig. 1). In Fig. 10 and 11, we show complex animal shapes created and posed using our modeling tool. Such poses, along with the possibility to make modeling changes on-the-fly, are ideal for key-frame animation. Because our rig is created in tandem with the model itself and updated locally, there are no pauses in the modeling process due to re-rigging.[3]

[3]A complete modeling session can be found in the video materials.

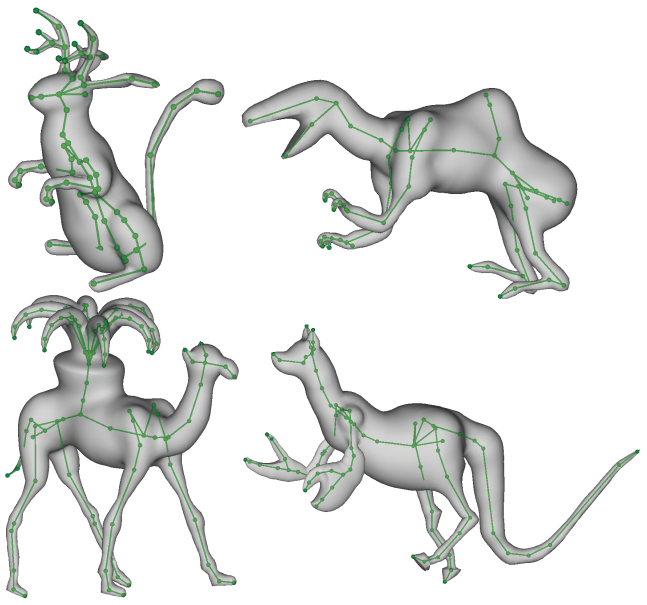

Figure 12: *Imaginary creatures. These models took approximately 5 minutes to make by reusing existing models or parts through simple cut-and-merge operations. Top row: Jackalope; Dino-camelaroo. Bottom row: Octocamel; Crabadogotaurus.*

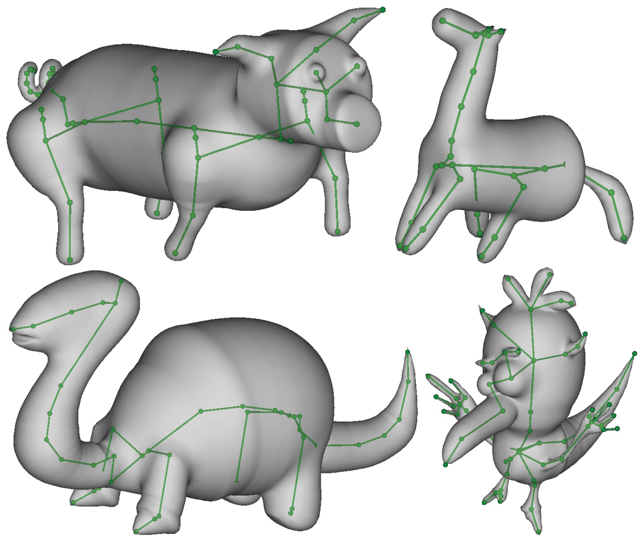

Figure 13: *Models created by first-time users. These models took an average 30 minutes to make.*

The idea of modeling by parts also enables re-use and collaboration amongst users. For the tyrannosaur model in Fig. 11, the head and torso were created by a first-time user; an experienced user created the front-arms, feet, and tail; and a second experienced user performed detailed refinement by adding the claws. Fig. 12 shows the results of repurposing existing models and parts. Simple cut-and-merge operations proved efficient for creating complex shapes.

We have conducted an informal user study by training first-time users on our system for approximately 20 minutes, and then allowing them to model without a time limit. None of the subjects had significant prior 3D modeling experience. One subject had 2D artistic experience and another had research experience in computer graphics. Some of their modeling results are shown in Fig. 13. Fig. 14 depicts two very young users.

	Step 1		Step 2		Step 3		Step 4		Step 5		Step 6	
	Local	Global	Local	Global	Local	Global	Local	Global	Local	Global	Local	Global
#Bones involved	10	17	11	25	8	33	10	41	9	45	4	49
#Vertices involved	1685	3270	1567	4071	1771	4867	1639	5643	937	6284	352	6582
Time (ms)	102	181	115	259	142	333	151	428	96	529	29	547
Difference in weight	0.0632		0.0396		0.0281		0.0228		0.0061		0.0023	

Table 1: *Comparing local and global skin weights for the camel model from Fig. 10.*

Figure 14: *Two very young users trying out RigMesh.*

We have also publicly released RigMesh to professional artists. Artists expressed enthusiasm regarding RigMesh's simple rigging process and have remarked that it "put[s] the fun back into rigging" [Thacker 2012]; they also find the idea of using the rig as part of the modeling process very helpful, which allows for quickly posing while trying out ideas. This is especially valuable for artists who frequently begin modeling without a specific goal in mind.

4.2 Local skin weight re-computation

In computing the skin weights, we use a KD-tree to accelerate the test to determine whether a line segment is contained entirely inside the shape volume. When combined with our local skin weight re-computation algorithm, the result is that merging shapes never requires a global segment/volume containment test, which is the performance bottleneck of the (global) Pinocchio skin weight algorithm [Baran and Popović 2007].

We compared our local skin weight re-computation algorithm to the global algorithm. Fig. 15 visualizes the difference (L_1 vector magnitude) between skin weights (which are all positive and sum to 1 on each vertex) computed by our incremental algorithm and weights computed globally on the finished models. Table 1 compares the running time and the 90th percentile of difference in skin weights between the two algorithms from the camel modeling session (Fig. 10).[4] Our local re-skinning method shows considerably improved performance over the global method without noticeable loss in deformation fidelity; as the number of vertices increases due to merging, the difference in running time between the local and global re-skinning becomes more pronounced.

5 Conclusion, limitations and future work

Our modeling tool provides an easy-to-use interface for modeling relatively complex characters, following the idea of *modeling by parts*. In addition to being accessible to novices, our tool is advantageous for rapid prototyping and base mesh creation (and the created models can be further refined in other tools). Unlike the traditional, sequential approach to modeling and rigging, our system maintains a rigged mesh throughout the modeling process. With the skeleton and skin weights, artists can freely pose their

[4]Specifically, the steps shown are the merging of the left back leg, right back leg, left front leg, right front leg, neck, and head.

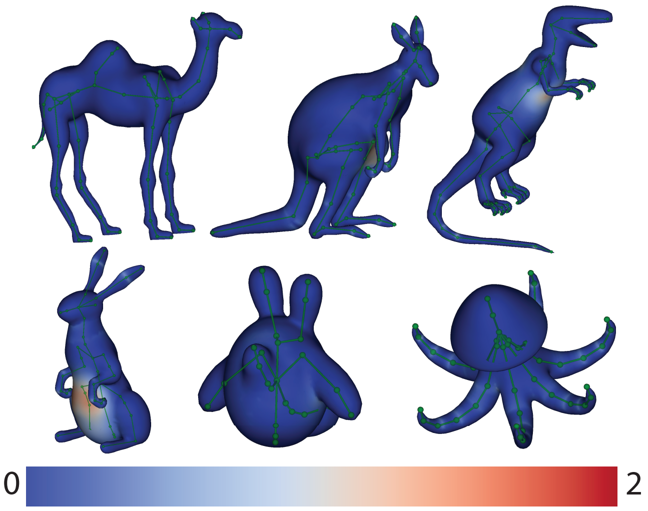

Figure 15: *Visualizing the L_1 magnitude of the vector difference in skin weights computed by our incremental algorithm and weights computed globally using the Pinocchio algorithm [Baran and Popović 2007].*

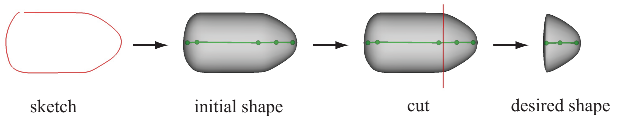

Figure 16: *A possible work-around to create the squid head.*

models-in-progress and more rapidly iterate through the entire character animation pipeline. We believe our work is important for connecting sketch-based modeling approaches from the literature with professional uses. (Our current implementation supports exporting to the industry-standard FBX format [2012b] for further refinement. In the future we would like to support exporting to additional formats, allowing artists to model using parts created or refined in other tools.)

Our system is designed for models that merit a skeleton-based rig, such as characters, and can handle any tubular organic shape or shapes that can be cut therefrom (limited to spherical topology). However, when the input sketch does not have an obvious symmetry axis, our system does not generate a desirable shape with a reasonable skeleton. For example, to make the squid head (Fig. 16), the user cannot simply sketch its silhouette. Instead, a work-around is introduced by sketching a longer shape and cutting it apart afterwards so the skeleton has the desired orientation. In addition, it can be unfamiliar to novice users to consider the skeletal structure during the creative process. For example, the novice user who created the bird character in Fig. 13 commented that he had to resort to unnatural skeletons to achieve the details on the face.

The CAT occasionally produces unstable chordal axes near regions of the stroke with a close-to-circular shape; this can lead to somewhat unexpected 3D shapes. Mi et al. [2009] describe techniques that extend and extrapolate the chordal axes to mitigate such problems.

Our local skin weight re-computation follows the heat diffusion method in Pinocchio [2007], which can lead to incorrect bone weight assignment in certain cases. Wareham and Lasenby [2008] discussed this issue in details and proposed the bone glow method to address it. Our skin weights can thus be improved by adapting for their algorithm.

The current system employs inverse kinematics for posing and deformation, which is oblivious to the semantics of the model's parts. It would be interesting to explore the possibilities of posing the shapes in meaningful ways, as in the work of Grochow et al. [2004] and Hecker et al. [2008].

Finally, an obvious, yet non-trivial extension of our tool is towards simplified animation controls. Specifically, we are interested in including animation in our unified pipeline, such that modeling operations are immediately reflected in the animation sequence. We envision that this will simplify the work of animators, as well as bring more practitioners to the craft.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments and suggestions. We also want to thank Tim Gerstner, Kevin Sanik and Polina Yanovich for their helpful comments on the write-up of the paper. We are grateful to Zach Shukan and Omega Darling for helpful discussions. We greatly appreciate the efforts of our lab members for the models they made during the informal user study. This research is supported in part by the Sloan Foundation, the NSF (CAREER Award CCF-06-43268 and grants IIS-09-16129, IIS-10-48948, IIS-11-17257, CMMI-11-29917, IIS-09-16845, CCF-0541185), and generous gifts from Adobe, Autodesk, Intel, mental images, Microsoft, NVIDIA, Side Effects Software, and the Walt Disney Company.

References

ALEXE, A., BARTHE, L., CANI, M.-P., AND GAILDRAT, V. 2005. Shape modeling by sketching using convolution surfaces. In *Proceedings of Pacific Graphics*, Short paper.

ALLÈGRE, R., GALIN, E., CHAINE, R., AND AKKOUCHE, S. 2006. The hybridtree: mixing skeletal implicit surfaces, triangle meshes, and point sets in a free-form modeling system. *Graph. Models 68*, 1 (Jan.), 42–64.

AU, O. K.-C., TAI, C.-L., CHU, H.-K., COHEN-OR, D., AND LEE, T.-Y. 2008. Skeleton extraction by mesh contraction. *ACM Trans. Graph. 27*, 3 (Aug.), 44:1–44:10.

AUTODESK, 2012. 3ds Max. http://www.autodesk.com/3dsmax.

AUTODESK, 2012. Fbx. http://usa.autodesk.com/fbx.

AUTODESK, 2012. Maya. http://www.autodesk.com/maya.

BARAN, I., AND POPOVIĆ, J. 2007. Automatic rigging and animation of 3D characters. *ACM Trans. Graph. 26*, 3 (July).

BERNHARDT, A., PIHUIT, A., CANI, M.-P., AND BARTHE, L. 2008. Matisse: Painting 2D regions for modeling free-form shapes. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM)*, 57–64.

BIEDERMAN, I. 1987. Recognition-by-components: A theory of human image understanding. *Psychological Review 94*, 115–147.

BLOOMENTHAL, J. 2002. Medial-based vertex deformation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 147–151.

CHAUDHURI, S., KALOGERAKIS, E., GUIBAS, L., AND KOLTUN, V. 2011. Probabilistic reasoning for assembly-based 3D modeling. *ACM Trans. Graph. 30*, 4.

CHERLIN, J. J., SAMAVATI, F., SOUSA, M. C., AND JORGE, J. A. 2005. Sketch-based modeling with few strokes. In *Proceedings of the Spring Conference on Computer Graphics*, 137–145.

CHEW, L. P. 1989. Constrained delaunay triangulations. *Algorithmica 4*, 1, 97–108.

CORDIER, F., SEO, H., PARK, J., AND NOH, J. Y. 2011. Sketching of mirror-symmetric shapes. *IEEE Transactions on Visualization and Computer Graphics 17*, 11 (Nov.), 1650–1662.

CORNEA, N. D., SILVER, D., AND MIN, P. 2007. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on Visualization and Computer Graphics 13* (May), 530–548.

DOUGLAS, D. H., AND PEUCKER, T. K. 1973. Algorithms for the reduction of the number of points required to represent a line or its caricature. *The Canadian Cartographer 10*, 2, 112–122.

FUNKHOUSER, T., KAZHDAN, M., SHILANE, P., MIN, P., KIEFER, W., TAL, A., RUSINKIEWICZ, S., AND DOBKIN, D. 2004. Modeling by example. *ACM Trans. Graph. 23*, 3 (Aug.), 652–663.

GINGOLD, Y., IGARASHI, T., AND ZORIN, D. 2009. Structured annotations for 2D-to-3D modeling. *ACM Trans. Graph. 28* (December), 148:1–148:9.

GROCHOW, K., MARTIN, S. L., HERTZMANN, A., AND POPOVIĆ, Z. 2004. Style-based inverse kinematics. *ACM Trans. Graph. 23*, 3 (Aug.), 522–531.

HECKER, C., RAABE, B., ENSLOW, R. W., DEWEESE, J., MAYNARD, J., AND VAN PROOIJEN, K. 2008. Real-time motion retargeting to highly varied user-created morphologies. *ACM Trans. Graph. 27*, 3 (Aug.), 27:1–27:11.

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

KARPENKO, O. A., AND HUGHES, J. F. 2006. SmoothSketch: 3D free-form shapes from complex sketches. *ACM Trans. Graph. 25* (July), 589–598.

KAVAN, L., COLLINS, S., ŽÁRA, J., AND O'SULLIVAN, C. 2008. Geometric skinning with approximate dual quaternion blending. *ACM Trans. Graph. 27* (November), 105:1–105:23.

KRAEVOY, V., JULIUS, D., AND SHEFFER, A. 2007. Model composition from interchangeable components. In *Proceedings of Pacific Graphics*, 129–138.

KRY, P. G., JAMES, D. L., AND PAI, D. K. 2002. Eigenskin: real time large deformation character skinning in hardware. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, 153–159.

LEWIS, J. P., AND ANJYO, K. 2009. Identifying salient points. In *ACM SIGGRAPH ASIA 2009 Sketches*, 41:1–41:1.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of ACM SIGGRAPH*, 165–172.

MAGNENAT-THALMANN, N., LAPERRIÈRE, R., AND THALMANN, D. 1988. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings of Graphics Interface*, 26–33.

MAXIS, 2008. Spore creature creator. http://www.spore.com/.

MI, X., DECARLO, D., AND STONE, M. 2009. Abstraction of 2D shapes in terms of parts. In *Proceedings of NPAR*, 15–24.

MILLER, C., ARIKAN, O., AND FUSSELL, D. 2010. Frankenrigs: building character rigs from multiple sources. In *ACM Symposium on Interactive 3D Graphics (I3D)*, 31–38.

MOHR, A., AND GLEICHER, M. 2003. Building efficient, accurate character skins from examples. *ACM Trans. Graph. 22* (July), 562–568.

NEALEN, A., SORKINE, O., ALEXA, M., AND COHEN-OR, D. 2005. A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph. 24*, 3 (July), 1142–1147.

NEALEN, A., IGARASHI, T., SORKINE, O., AND ALEXA, M. 2007. FiberMesh: Designing freeform surfaces with 3D curves. *ACM Trans. Graph. 26*, 3 (July).

NEALEN, A., PETT, J., ALEXA, M., AND IGARASHI, T. 2009. GridMesh: Fast and high quality 2D mesh generation for interactive 3D shape modeling. In *Shape Modeling International (SMI)*, 155–162.

NIELSEN, J. 1993. *Usability Engineering*. Morgan Kaufmann publishers Inc.

OLSEN, L., SAMAVATI, F., SOUSA, M., AND JORGE, J. 2009. Sketch-based modeling: A survey. *Computers & Graphics 33*, 85–103.

PAN, J., YANG, X., XIE, X., WILLIS, P., AND ZHANG, J. J. 2009. Automatic rigging for animation characters with 3D silhouette. *Comput. Animat. Virtual Worlds 20*, 23 (June), 121–131.

PIHUIT, A., CANI, M.-P., AND PALOMBI, O. 2010. Sketch-based modeling of vascular systems: a first step towards interactive teaching of anatomy. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM)*, 151–158.

PIXOLOGIC, 2012. ZBrush. http://www.pixologic.com/zbrush/.

PRASAD, L. 1997. Morphological analysis of shapes. *CNLS Newsletter 139*, 1–18.

SCHMIDT, R., AND SINGH, K. 2010. meshmixer: an interface for rapid mesh composition. In *ACM SIGGRAPH 2010 Talks*, 6:1–6:1.

SCHMIDT, R., WYVILL, B., SOUSA, M. C., AND JORGE, J. A. 2005. ShapeShop: Sketch-based solid modeling with blob-Trees. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM)*, 53–62.

SHARF, A., BLUMENKRANTS, M., SHAMIR, A., AND COHEN-OR, D. 2006. SnapPaste: an interactive technique for easy mesh composition. *Vis. Comput. 22*, 9 (Sept.), 835–844.

SHARF, A., LEWINER, T., SHAMIR, A., AND KOBBELT, L. 2007. On-the-fly curve-skeleton computation for 3D shapes. *Computer Graphics Forum 26*, 3 (october), 323–328.

SHERSTYUK, A. 1999. Interactive shape design with convolution surfaces. In *Shape Modeling International (SMI)*, 56–65.

SHUKAN, Z. 2012. Personal communication.

SORKINE, O., AND COHEN-OR, D. 2004. Least-squares meshes. In *Shape Modeling International (SMI)*, 191–199.

SUGIHARA, M., DE GROOT, E., WYVILL, B., AND SCHMIDT, R. 2008. A sketch-based method to control deformation in a skeletal implicit surface modeler. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM)*, 65–72.

TAI, C.-L., ZHANG, H., AND FONG, J. C.-K. 2004. Prototype modeling from sketched silhouettes based on convolution surfaces. *Computer Graphics Forum 23*, 1, 71–83.

THACKER, J. 2012. Rigmesh: putting the fun back into rigging. http://www.cgchannel.com/2012/05/rigmesh-putting-the-fun-back-into-rigging/.

WANG, R. Y., PULLI, K., AND POPOVIĆ, J. 2007. Real-time enveloping with rotational regression. *ACM Trans. Graph. 26* (July).

WAREHAM, R., AND LASENBY, J. 2008. Bone glow: An improved method for the assignment of weights for mesh deformation. In *Proceedings of the 5th international conference on Articulated Motion and Deformable Objects*, AMDO '08, 63–71.

WATT, A., AND WATT, M. 1991. *Advanced animation and rendering techniques*.

WEBER, O., SORKINE, O., LIPMAN, Y., AND GOTSMAN, C. 2007. Context-aware skeletal shape deformation. *Computer Graphics Forum 26*, 3.

WYVILL, B., GUY, A., AND GALIN, E. 1999. Extending the CSG tree: Warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum 18*, 2, 149–158.

YANG, R., AND WÜNSCHE, B. C. 2010. Life-sketch: a framework for sketch-based modelling and animation of 3D objects. In *Proceedings of the Eleventh Australasian Conference on User Interface - Volume 106*, 61–70.