# Sketching Mesh Deformations

Youngihn Kho* Michael Garland†

University of Illinois at Urbana-Champaign

Figure 1: Step-by-step editing of a dragon character in under 3 minutes using our system. Each step represents 1–3 individual deformations.

## Abstract

Techniques for interactive deformation of unstructured polygon meshes are of fundamental importance to a host of applications. Most traditional approaches to this problem have emphasized precise control over the deformation being made. However, they are often cumbersome and unintuitive for non-expert users.

In this paper, we present an interactive system for deforming unstructured polygon meshes that is very easy to use. The user interacts with the system by sketching curves in the image plane. A single stroke can define a free-form skeleton and the region of the model to be deformed. By sketching the desired deformation of this reference curve, the user can implicitly and intuitively control the deformation of an entire region of the surface. At the same time, the reference curve also provides a basis for controlling additional parameters, such as twist and scaling. We demonstrate that our system can be used to interactively edit a variety of unstructured mesh models with very little effort. We also show that our formulation of the deformation provides a natural way to interpolate between character poses, allowing generation of simple key framed animations.

**CR Categories:** I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric Transformations I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques

**Keywords:** interactive mesh deformation, sketch-based editing, intuitive interfaces

*e-mail: kho@uiuc.edu

†e-mail: garland@uiuc.edu

## 1 Introduction

While many techniques for geometric mesh deformation have been developed, finding effective interactive techniques is still a challenging topic in modeling and animation. Observing that direct interaction in 3-D space is often a confusing task for non-expert users, we propose an intuitive interface for mesh deformation by sketching curves in the image plane. Our system allows a user to easily apply a broad range of deformations to unstructured polygon meshes.

In our system, users sketch a reference curve in the image plane both to determine a region of interest and to serve as a means of controlling an individual deformation. By sketching a second curve indicating the desired deformation of the reference curve, users can easily achieve the deformation of the entire region of interest specified by the reference curve. By constructing a mapping of the region of interest onto the reference curve, our system also provides a simple method for controlling additional parameters such as local twisting or scaling.

Our system provides a great deal of flexibility to the user. It can accept as input triangulated manifold meshes of any genus, containing any number of boundary loops. No further structural information about the object is required. The user can use free-form reference curves without being constrained by geometrically defined skeletal structures. The deformation curves can be significantly different than the "natural" skeleton of the surface, and work well even when such skeleton structures would be ill-defined.

By using a sketch-based screen space interface, we avoid the need for complex 3-D interactions that can be cumbersome for non-expert users. Users can achieve relatively complex deformations by simply drawing two strokes on the screen. Furthermore, we can use these sketch-based deformations to achieve a natural interpolation between character poses, thus producing simple key framed animations.

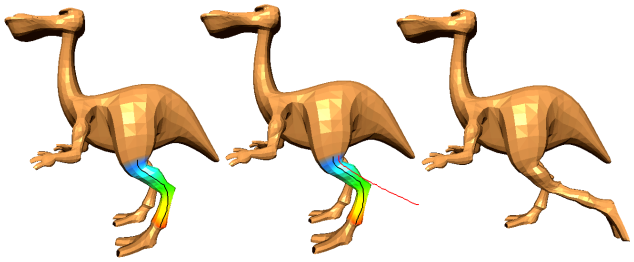

Figure 2: A simple sketch-based mesh deformation. The user draws a reference curve along the leg, followed by a second target curve. This induces a deformation of the leg itself.

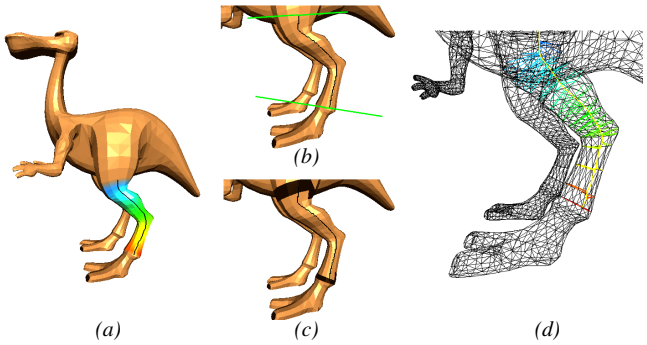


Figure 3: Preparing for deformation of the left leg. We compute two cutting planes (b) that will define the two loops bounding the region of interest (c). Each vertex in the region is mapped to the closest point on the reference curve (d).

## 2 Related Work

There has been a great deal of work done in the past on developing techniques for the modeling and deformation of geometric objects. Here we survey only the most relevant work, with an emphasis on techniques for interactive deformation.

Free-Form Deformations (FFD) are one of the most important techniques for deforming surfaces [Sederberg and Parry 1986; Coquillart 1990; MacCracken and Joy 1996]. While their complex control lattices provide very precise control over the resulting deformation, editing these lattices can be an unintuitive and time-consuming process. Handle-manipulation approaches [Kobbelt et al. 1998; Bendels and Klein 2003; Yu et al. 2004] are also popular and can produce pleasingly smooth deformations. However, the range of deformations that can be produced with a single handle manipulation is generally quite limited. Therefore a user must perform a sequence of several individual step to achieve a more complex result.

Curve-based deformation approaches such as Wires [Singh and Fiume 1998] or medial-based shape deformations [Bloomenthal 2002; Yoshizawa et al. 2003] have received considerable attention in recent years. Curves or skeletons in this type of methods can provide a natural means of capturing the structure of surfaces. Thus, deforming surfaces by editing those entities provides a good means of achieving large scale deformations. For this reason, our system uses a curve-based approach with an emphasis on providing an extremely easy method for specifying free-form control curves.

There has been substantial interest of late in developing intuitive interactive techniques, such as deformation by painting over surfaces [Lawrence and Funkhouser 2003]. Sketch-based interfaces have emerged as one of the more popular approaches to building user-friendly deformation tools. In Teddy [Igarashi et al. 1999], users can create and edit objects by simply sketching strokes in the screen. The system also proposes a deformation method based on *warp* [Corrêa et al. 1998]. Recently, a sketching interface to FFDs has been developed [Hua and Qin 2003]. Here sketch strokes are used to manipulate scalar field embeded in 3-D space. In our system, we employ sketching both to specify and deform the regions of interest.

## 3 Overview

In our system, the user initiates a deformation by drawing a reference curve on the image plane. This curve implicitly defines a *region of interest* — that part of the surface which will be deformed. The user then applies the deformation either by sketching a new target shape for the reference curve or by directly manipulating a deformation parameter such as twist or scaling.

Figure 2 shows a simple example of our system in action. The user begins by drawing a reference curve along the leg. The region of interest is highlighted with a red-to-blue color ramp. The user then draws a target curve indicating the desired deformation. From this pair of curves, the system automatically generates the deformation of the leg. A more complex editing session is shown in Figure 1. Each step in this editing sequence corresponds to 1–3 individual deformations.

## 4 Beginning a Deformation

The user begins the process of deformation by drawing a reference curve, which must be projected into the 3-D world space. From the reference curve, we implicitly recognize the region of the surface that the user wishes to deform. The user can optionally refine this region selection using an interactive partitioning scheme. Once the region of interest is identified, a "skinning" step associates each vertex within this region with the closest point on the 3-D reference curve. This basic process is illustrated in Figure 3.

### 4.1 Building the Reference Curve

We begin with a free-form sketch of the reference curve in the image plane. We represent the raw sketch curve as a collection of line segments taken directly from mouse events produced by the user's stroke. This raw curve is likely to be fairly noisy, especially when drawn with a mouse rather than a tablet device. Therefore, before proceeding, we smooth and regularize the raw sketch. We apply a simple averaging filter and simplify the polyline by merging neighboring segments so that each segment will be at least 5 pixels in length.

Having regularized the reference curve in the image plane, we must project it into the 3-D world space of the model. We first compute the point of intersection of a ray from the view point through the first point on the sketch curve. This hit point, along with the normal of the viewing plane, defines a plane in world space parallel to the image plane. We project the sketch curve onto this plane to compute the 3-D reference curve.

### 4.2 Recognizing the Region of Interest

After the user-drawn reference curve is mapped into 3-D space, we implicitly partition the model into three parts: (1) a static component, (2) the region of interest, and (3) a rigid component. The static component of the mesh will be unchanged by the deformation. The

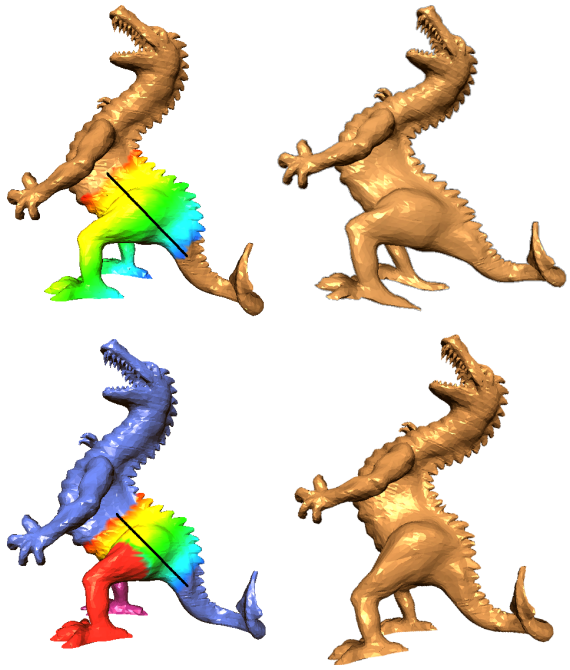

Figure 4: (Top) Deformation of the body without partitioning. The legs and feet are undesirably distorted. (Bottom) The two legs are partitioned so that they can be transformed rigidly, producing a more natural result.

region of interest is that part of the mesh to which the deformation will actually be applied. The rigid component will be transformed rigidly to maintain its connectivity with the region of interest. Figure 3 shows a simple example in which (1) the whole body beyond the upper thigh forms the static component, (2) the leg is the region of interest, and (3) the foot below the ankle is the rigid component.

The underlying assumption of our system is that the region of interest is the part of the surface "covered" by the user's sketch curve. At each end point of the curve, the system computes a cutting plane perpendicular to the reference curve (see Figure 3b). The intersection of these planes with the surface define triangle loops that partition the input mesh. We define these triangle loops using a graph cut formulation outlined in Section 4.2.1. In cases where a plane defines multiple intersection loops, we select the loop containing the nearest triangle to the end point. Having selected one boundary loop per cutting plane, we now have two triangle loops bounding the region of interest (as in Figure 3c). The vertices in each of the 3 regions are labelled by a breadth first style "flood fill". The static component will be the region bounded by the loop created by the starting point of the reference curve and the rigid component is bounded by its ending point.

In higher genus cases, a single loop may fail to cut the object into two parts. In such cases, multiple loops are required for the partition. We begin with the set of all loops defined by the cutting plane; these *must* collectively partition the object, as the plane itself does. We then consider each loop other than the initially one in succession. We remove a candidate loop from the cut only if does not merge the two components separated by the initially selected loop. This process eventually produces exactly two disjoint components.

Our implicit recognition scheme works well in many cases. However, additional explicit partitioning is useful in certain circumstances. For example, in Figure 4, when we lift up the back part of the dragon, we probably want to rigidly transform each leg, while the body is smoothly deformed. To do this, users can interactively augment (or override) the automatically generated partition.

#### 4.2.1 Interactive Partitioning

In our system, we adopt a graph cut partitioning scheme controlled by the selection of cutting planes. The user draws a line to define a cutting plane containing that line and perpendicular to the view plane. In general, this cutting plane will not follow existing edges in the mesh but will cut across many triangles. Therefore, we apply a fuzzy decomposition technique [Katz and Tal 2003] to find the actual boundaries. We collect all triangles within a certain screen-space distance from the cutting plane — we typically adopt a 5 pixel distance limit. In general, the cutting plane might create multiple separated fuzzy regions. In this case, the system picks the region which is nearest to the view point. Then this set of triangles serves as a fuzzy region. To compute the boundary, a dual graph of the fuzzy region is created, where the weight of an edge in the dual graph is the dihedral angle of the corresponding primal edge multiplied by its length. Finally, a min-cut method on the dual graph produces a cut corresponding to the boundary. The method produces natural and smooth boundaries since the resulting cut follows relatively lower dihedral angles which are a good criterion for natural boundaries. The freely deformable region is bounded by only one partition. All other parts will be rigidly transformed.

### 4.3 Skinning and Parameterization

At this point, all the vertices in the region of interest have been collected. We skin the surface by associating each vertex with the closest point on the reference curve. Note that these closest points are simply required to be on the curve; they need not be vertices of the curve. Figure 3d shows an example of this association, connecting each vertex with its corresponding point on the curve.

We represent each point on the reference curve by its *normalized arc length* $s$. That is, for a given point we add up the length of all segments from the origin of the curve to the point. We normalize these values so that they range between 0 and 1. Thus $s = 0$ and $s = 1$ are, respectively, the origin and end points of the curve. This induces a parameterization of the region of interest onto the range $[0, 1]$. For each vertex $\mathbf{v}$ we have the normalized arc length parameter $s(\mathbf{v})$ corresponding to the associated point on the reference curve. This parameterization is shown by the color ramp on the leg in Figure 3a, with blue corresponding to $s = 0$ and red to $s = 1$.

As outlined earlier, one or more regions of the surface will be rigidly transformed. To assign the single transformation to each partition, we introduce a *virtual vertex* for each of them. The virtual vertex is placed in the center of the boundary between partitions and is associated with the closest point on the reference curve. The transformation of a virtual vertex determines the transformation of all vertices in its partition.

## 5 Deformation Techniques

In the previous section, we have discussed the preparation steps necessary to begin a deformation. The user draws a reference curve, which is filtered and projected into 3-D. The region of interest is determined, and each vertex within this region is mapped to a point on the 3-D reference curve. Once this initial phase is complete, the user can apply any one of the fundamental deformations described in this section.

### 5.1 Sketch-Based Mesh Deformation

The primary deformation that we support is accomplished by sketching. The user simply draws a new *target* curve, which we

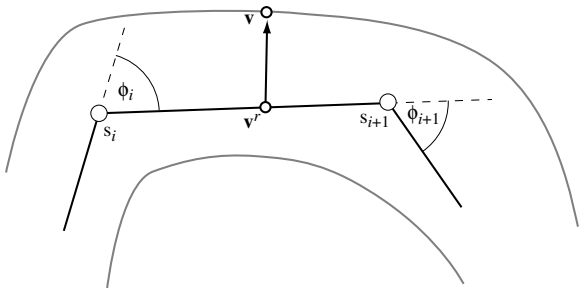


Figure 5: Illustration of sketch-based deformation. For each vertex $\mathbf{v}$ in the mesh, we compute the closest point $\mathbf{v}^r$ in the reference curve. The total turning angle for the vertex $\mathbf{v}$ is linearly interpolated in the line segment $\mathbf{v}^r$ lies on.

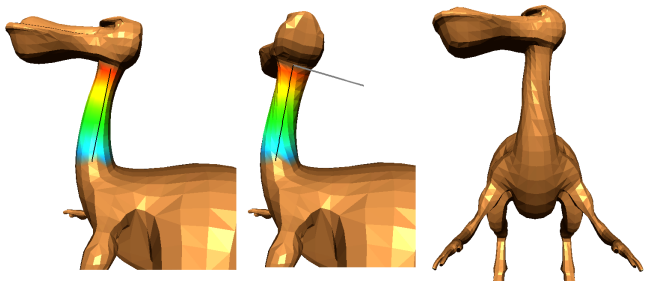


Figure 6: An example of twisting the neck of a dinopet model. (a) We first specify the approximate part of the neck to twist. (b) The neck is twisted by specifying the rotation axis (black) and the amount of angles (grey). Note that in this example, rotational angle is linear to the parametrization which is color ramp coded. (c) The result from a different view point.

interpret as a deformation of the original *reference* curve. Our system then deforms the entire region of interest in an analogous way. This provides the user with intuitive and flexible control over the object's shape. Figure 2 shows a simple example of this style of deformation.

Recall that we have already assigned a value $s(\mathbf{v})$ to each vertex $\mathbf{v}$ which connects it to a corresponding point on the reference curve. We can thus think of the deformation process as follows. The reference curve is scaled and bent to match the target curve. The vertices of the mesh are connected to the reference curve through rigid iron wires, so they are translated and rotated along with the reference curve.

**Preparation** We begin by filtering and smoothing the target curve in the image plane, just as we did the reference curve. To prevent the mesh from tearing, we translate the target curve so that it's starting point is coincident with the starting point of the reference curve. We then map the target curve to the sketch plane constructed in Section 4.1. Consequently, the user is not allowed to change viewing parameters while drawing the two curves.

For each vertex $\mathbf{v}$ in the region of interest, we have a corresponding normalized arc length value $s(\mathbf{v})$ that provides us with the corresponding closest point $\mathbf{v}^r$ on the reference curve. Similarly, we use the same normalized arc length parameter to compute the corresponding point $\mathbf{v}^t$ on the target curve. This provides us with all the information necessary to compute to desired transformation at $\mathbf{v}$.

**Computing Rotational Angles** We deform the surface by rotating each vertex $\mathbf{v}$ about the corresponding reference point $\mathbf{v}^r$. The axis of rotation is simply the normal of the sketch plane and the rotational angle $\theta(\mathbf{v})$ is the signed angle between the tangents at $\mathbf{v}^r$ and $\mathbf{v}^t$. However, since our curves are sequences of line segments, we must interpolate rotational angles along the curve in order to avoid significant discontinuities.

To compute the rotational angle $\theta(\mathbf{v})$, we must first locate the line segment $[s_i, s_{i+1}]$ of the reference curve on which $\mathbf{v}^r$ lies. We define $\phi_i$ to be the signed exterior turning angle of the curve at node $i$ (see Figure 5). We will define the tangent direction at the node $s_i$ by the total turning angle $\Phi_i$:

$$\Phi_i = \sum_{j=0}^{i-1} \phi_j + \frac{\phi_i}{2} \tag{1}$$

Note that we use the half-angle $\frac{\phi_i}{2}$ so that the tangent at $s_i$ will be the average direction of the two incident line segments. This defines the total turning angle at the nodes of the curve. For a point on the interior of a segment $[s_i, s_{i+1}]$ we interpolate the turning angle

$$\Phi(s) = \Phi_i + \frac{\phi_i}{2} b(2\alpha) + \frac{\phi_{i+1}}{2} b(2\alpha - 1) \tag{2}$$

where

$$\alpha = \frac{s - s_i}{s_{i+1} - s_i} \tag{3}$$

and

$$b(x) = \begin{cases} 1 & \text{if } x > 1, \\ x & \text{if } 0 \le x \le 1, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

The blending function $b$ is chosen so that $\Phi(\frac{s_i + s_{i+1}}{2}) = \Phi_i + \frac{\phi_i}{2}$. In other words, so that the tangent direction at the midpoint of the segment will be parallel to the segment.

We can compute these total turning angles for both the reference and target curves. They tell us the signed angle difference between the initial segment of the curves and the given points $\mathbf{v}^r$ and $\mathbf{v}^t$. We also need to account for the global rotation $\theta_g$, which is the angle between the initial segments of the two curves. Our desired rotational angle is now simply

$$\theta(\mathbf{v}) = \Phi^r(s(\mathbf{v})) - \Phi^t(s(\mathbf{v})) + \theta_g \tag{5}$$

Once we have computed the target position and the desired rotational angle for a vertex, the final deformed position $\mathbf{v}'$ of the vertex $\mathbf{v}$ will be

$$\mathbf{v}' = T(\mathbf{v}^t) R(\theta(\mathbf{v})) T(-\mathbf{v}^r)\mathbf{v} \tag{6}$$

where $T$ indicates translation and $R$ indicates rotation about the normal of the sketch plane.

## 5.2 Twisting

We can also achieve twisting deformations by locally rotating the the region of interest about the reference curve. For example, consider the simple twisting operation shown in Figure 6. The reference curve is now the rotational axis and the user's second mouse stroke is used to control the amount of twisting being performed.

To use the reference curve as a rotational axis, it must be placed inside of the the model. To do this, we must alter the way in which we project the reference curve into world space. We first compute a set of *joints*. For each node in the reference curve in the image

plane, a joint is defined as the average of hit points on the nearest front face and the back face by the ray from the viewpoint to the vertex. Consequently, we disallow twisting if any of nodes in the sketch curve has only one hit point. Now, the rotational axis is the curve connecting these joints. Note that this new curve looks the same from the user's perspective, as it still projects to the same image space sketch curve. As we did in sketch-based deformation, the rotational axis for the vertex $\mathbf{v}$ (i.e., the tangent direction at $s(\mathbf{v})$) is linearly blended.

We must now compute the rotational angle $\theta(\mathbf{v})$. Obviously using the same rotational angle at all vertices would not produce the desired result. We have found that the most natural twisting is achieved when the rotational angles $\theta(\mathbf{v})$ vary linearly with the normalized arc length $s(\mathbf{v})$. Thus, we compute a maximum angle $\theta_{\max}$ proportional to the length of the second line drawn by the user and use a rotational angle $\theta(\mathbf{v}) = s(\mathbf{v})\theta_{\max}$ at the vertex $\mathbf{v}$. The new position for vertex $\mathbf{v}$ will therefore be

$$\mathbf{v}' = T(\mathbf{v}^r)R(\mathbf{t}^r(\mathbf{v}), \theta(\mathbf{v}))T(-\mathbf{v}^r)\mathbf{v} \tag{7}$$

where $\mathbf{t}^r(\mathbf{v})$ is the interpolated tangent direction of the reference curve at $s(\mathbf{v})$.

## 5.3 Indirect Control Using Parameterization

The parameterization $s(\mathbf{v})$ that we have established to map the region of interest onto the reference and target curves also provides a natural mechanism for adjusting deformation parameters. We allow the the user to gain finer control over the deformation by using a standard spline control to specify modifications of the deformation parameters as a function of $s$. In this section, we briefly outline three such controls.

**Adjusting Target Curve Turning Angles** In our sketch-based deformation, the end result is obviously controlled by the shape of the target curve. By fine tuning the target curve, we can fine tune the deformation. This allows the user to draw a fairly simple base target curve and then interactively adjust its shape to achieve a specific intended deformation.

We control the shape of the target curve by adjusting the exterior turning angles $\phi_i$, which were discussed in Section 5.1. By controlling these angles, we can radically alter the shape of the target curve.

We present to the user a standard spline box with which they can define an offset function $F(s)$, which we initialize to the identity function $F(s) = s$. For each vertex at position $s_i$ along the target curve, we compute an adjusted turning angle $\phi_i'$ as:

$$\phi_i' = \phi_i + F(s_i) - s_i \tag{8}$$

and use these adjusted turning angle to compute an adjusted deformation.

An example of this type of deformation is shown in Figure 7. We begin by sketching a very simple deformation that achieves a coarse deformation of the overall shape. We then adjusted the rotational angles to achieve a final S-shaped shark.

**Scaling Control** The scaling control allows users to locally inflate or deflate the surface along the reference curves. This technique can be thought as an interactive version of generalized cylinders [Snyder and Kajiya 1992]. We achieve local scaling by controlling the magnitude of the offset vector $\mathbf{v} - \mathbf{v}^r$ of a vertex $\mathbf{v}$. We use the same skeleton curve connecting joints as in twisting for computing these offsets. The magnitude of the offset vectors is scaled by a function $F(s)$ which is initially the constant function $F(s) = 1$. We see a typical example in Figure 8. We have locally inflated and deflated the left Queen along a vertical reference curve using the spline function shown on the right.

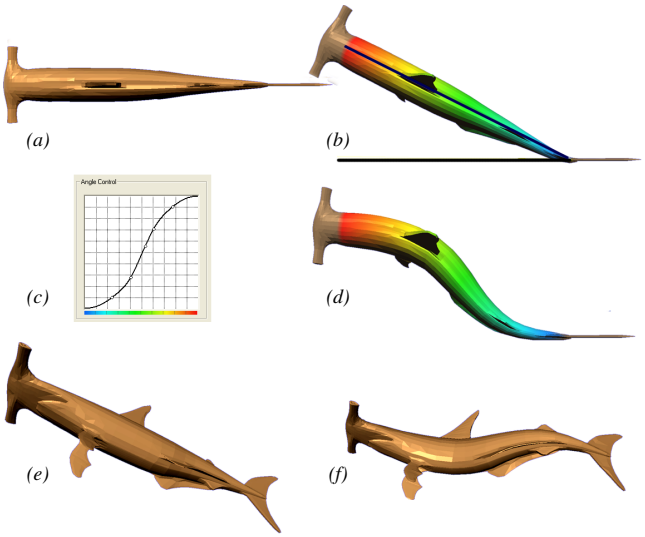


Figure 7: Adjusting target curve turning angles can fine tune the deformation. An initial coarse deformation (b) is adjusted using a spline control (c) to produce a more nuanced pose (d–f).

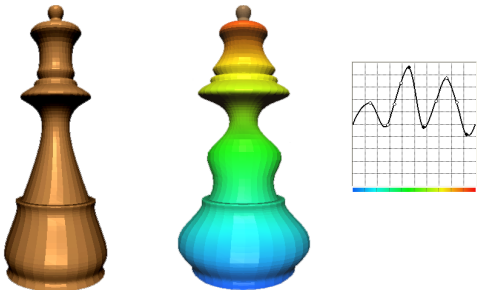

Figure 8: By locally scaling offset vectors, we can locally inflate and deflate the initial shape.

**Rotational Angle Control for Twisting** As discussed in Section 5.2, we linearly increase the twisting angle as a function of $s$. We can just as easily add an additional scaling adjustment to produce $\theta(\mathbf{v}) = F(s(\mathbf{v}))\theta_{\max}$. This allows the user to control the relative "speed" of the twist along the region of interest.

## 5.4 Increasing Smoothness

In most cases, our techniques produce pleasingly smooth deformations. However, local jaggedness can occur as the result of factors such as excessive noise in the user's sketch or the limited screen resolution. For these circumstances, we introduce a deformation optimization technique that can automatically smooth away such artifacts. It also provides a straightforward mechanism to blend the boundaries between rigidly transformed components and the freely deformable region.

Any one of the many general mesh smoothing algorithms could be applied to smooth the deformation. However, this would have the undesirable side-effect of removing actual small-scale *features* from the surface as well. Therefore, we seek to directly optimize deformation parameters to produce a smooth result. The central idea is neighboring vertices should undergo similar transformations to maintain smoothnes, while still remaining faithful to the user's

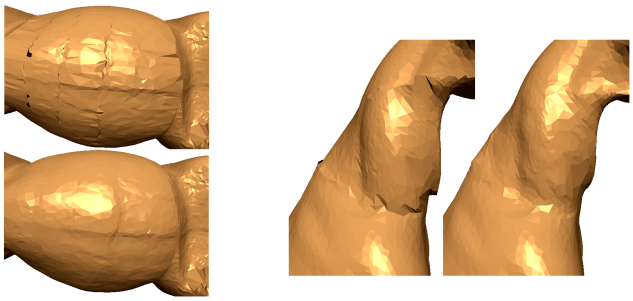

Figure 9: The benefits of automatically smoothing the deformation. (Left) Sketch-based deformation. The top picture is before the optimization, and the bottom picture is after the process. (Right) Twist deformation. The left one shows non-optimized twisting deformation, and the right one shows the result of the optimization.

specified transformation.

For each vertex $\mathbf{v}_i$ in the region of interest, let us consider its deformation parameter $u_i$. For sketch-based deformation, we would separately consider both rotational angles $\theta(\mathbf{v}_i)$ and the normalized arc length $s(\mathbf{v}_i)$. Starting from an initial set of parameters $(u_1,\ldots,u_n)$ generated from the user's input, we wish to find a new set of parameters $(u'_1,\ldots,u'_n)$ that minimizes the following energy:

$$E = \sum_i \left( \sum_{j \in N_i} w_{ij}(u'_i - u'_j)^2 \right) + w_c(u_i - u'_i)^2 \qquad (9)$$

where $N_i$ is the set of vertices adjacent to vertex $\mathbf{v}_i$.

The first term of this energy function provides a measure of the smoothness of the parameter $u_i$. We choose the edge weights $w_i j$ to be the inverse edge length $w_{ij} = \|\mathbf{v}_j - \mathbf{v}_i\|^{-1}$. The second term in the summation measures the deviation of the new parameters $u'_i$ from the parameters $u_i$ derived from the user's input. Empirically, we find that setting the weight $w_c$ such that $\sum_{j \in N(i)} w_{ij} = 10 w_c$ provides a generally good balance between smoothness and this fidelity term.

We solve this optimization problem using a traditional Newton method. The initial values are simply those before the optimization, which makes the second cost term zero. Since our cost function is quadratic and the number of neighboring vertices are usually very small, the Hessian matrix is constant and sparse. Therefore, the optimization problem can be very efficiently computed by solving a linear system through one-time LU factorization. Furthermore, the initial values tend to be fairly close to the optimal solution, so we observe that the number of iterations necessary is generally very small.

The result of this automatic optimization process are shown in Figure 9. The fairly obvious artifacts in the unoptimized deformation are entirely removed following the optimization process.

### 5.5 Adaptive Mesh Refinement

If the user applies a significant deformation to the model, the resolution of the input mesh may be insufficient to support smooth deformation. The result will be a locally jagged deformation. To account for this, we propose a simple adaptive refinement scheme for smooth deformation. Our refinement primitive is the *edge split*. For each edge, we test two criteria to decide whether to split or not. We split the edge only if the two criteria are both satisfied.

The first criterion is edge *stretching*, defined as the ratio of the edge length in the original mesh to its length in the deformed mesh.

Figure 10: When performing significant deformations, the result can appear jagged if the input mesh is too coarse. Adaptive mesh refinement removes these artifacts.

We will consider splitting any edge whose ratio exceeds a specified limit. We also apply this criterion recursively to new edges produced by splitting previous edges. Experimentally, we find that a ratio of 1.5 works well as a threshold; here an edge will be split if it is stretched by more than 50%. This criterion will tend to refine more along the direction of deformation, but less in the orthogonal direction, as can be seen in Figure 10. In this example, we subdivide more along the direction in which the neck is stretched, but relatively less around the neck.

The second criterion is edge *curvature*. We only wish to split those edges that are sufficiently bent by the deformation. Each candidate edge in the original mesh is tentatively splitted by inserting a vertex at the midpoint of the edge. Then the inserted vertex is also transformed to a deformed position. If the angle between the two new edges in the deformed mesh is less than some threshold, we accept the split. Otherwise, the split is cancelled. This means that if the split edges are close to parallel, the split is not necessary. We have found that an angle threshold of $\pi - \frac{\pi}{24}$ produces good results. In Figure 10, the bent part in the lower neck is more refined, while the middle part is less refined.

Our refinement scheme is designed to minimize the number of faces added while maintaining the smoothness of the deformation. However, the resulting mesh might have many triangles of bad aspect ratio. If more nearly equilateral triangles are desired, then a regularization process that can perform additional operations such as edge flipping could be considered [Welch and Witkin 1994; Kobbelt et al. 2000].

## 6 Results

In this section, we consider several examples of using our system to edit unstructured polygon meshes. All results were generated by interactive editing on a standard consumer-level Windows PC. We render all sample models with flat shading in order to better highlight the structure of the surface mesh.

Figure 1 provides a step-by-step illustration of an editing session in which we repose a dragon character. We begin by opening the mouth, which requires only two sketch-based deformations or exactly 4 lines to be drawn by the user. We subsequently twisted both arms and the neck. We conclude by using two sketch-based deformations to bend the tail. The entire editing session — including program startup, loading the mesh from disk, and interactive editing — required less than 3 minutes.

In Figure 11 we see an example of using sketch-based deformation to bend an initial cylinder into multiple letter forms. Each letter was created by drawing a single reference curve on the cylinder followed by a single target curve in the shape of the intended letter. Even though the original cylinder has been stretched and bent fairly significantly, the deformed surface remains smooth. Note that, in order to keep the number of triangles fixed across all examples, we have not applied our adaptive refinement scheme in this case.


Figure 11: Applying significant deformations to bend a cylinder into various letter forms still results in smooth surfaces.

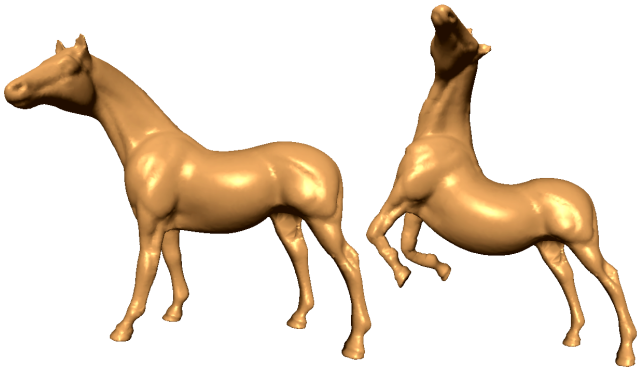

Figure 12: Reposing a horse with 4 simple deformations.

Figure 12 demonstrates the deformation of another more complex figure. We begin by interactively partitioning the front two legs. The body is then deformed by a single sketch-based deformation (requiring only 2 strokes). We conclude by twisting the neck, and bending the front legs by a single sketch-based deformation for each.

We can deform the hand shown in Figure 13 into a number of other poses quite easily. For the examples shown, we applied 1–2 sketch-based deformations to each of the fingers. Total modelling time is a mere 1 minute per hand pose. Note that the input is fairly smooth and the mesh is rather dense. Our system can still handle this mesh at interactive speeds and the deformed meshes are just as smooth as the input.

Figure 14 demonstrates a leg deformation using three different reference curves. We can clearly see that the system behaves well even when given quite different reference curves. The user is not constrained to draw a reference curve that follows the "natural" skeleton of the leg, and indeed can even draw a reference curve beyond the edge of the surface. All of these reference curves are sufficient to unfold the leg. We can also see that by drawing somewhat different reference curves, the user can easily excercise control over the nuances of the deformation result. For instance, the middle result bends the leg more rigidly than the others because the reference curve follows the shape of the leg less closely.

### 6.1 Skeleton Based Morphing

In our sketch-based deformation, the reference curves can be thought of as skeletons for the regions of interest. Utilizing these implicit skeletons, we can produce quite natural pose transitions that are much more pleasing than using simple linear interpolation (see Figure 15).

To morph using the skeletons at each intermediate frame, we must interpolate the curve from the reference and the target curves

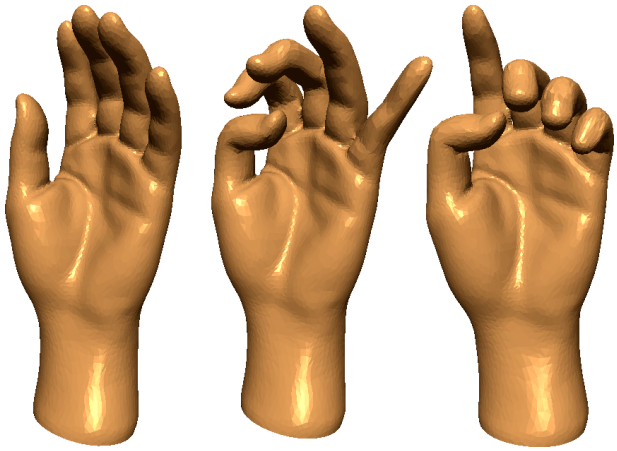

Figure 13: The original hand model (left) and two deformed hands.

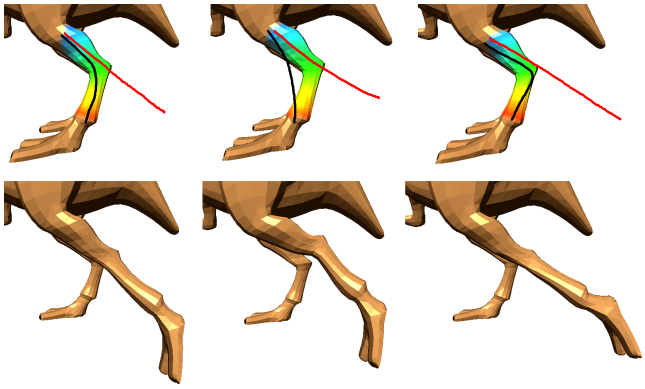

Figure 14: By using different reference curves, the user can produce subtly different results.

then deform the object according to the *interpolated curve*. The interpolated curve is computed automatically by linearly interpolating lengths and rotational angles. Its total length is linearly interpolated from the lengths of the reference and target curves. As defined in Section 5.1, there is a rotational angle $\theta(\mathbf{v})$ at each vertex of the reference curve that deforms it into the target curve. The rotational angles for the interpolated curve are simply $\alpha\theta(\mathbf{v})$ for $\alpha \in [0,1]$. This approach is similar to as-rigid-as-possible interpolation [Alexa et al. 2000], in the sense that we separate the rotation and the scale components.

Once the interpolated curve is computed, the vertex positions are computed in the same way as in the sketch-based deformation described in Section 5.1. The only difference is that we substitute the interpolated curve as the target curve.

## 7 Conclusion and Future Work

We have proposed a new and intuitive approach to interactive deformation of unstructured polygon meshes. Users of our system can make significant edits to 3-D objects by simply drawing a pair of curves on the image plane. The reference curves drawn by the user simultaneously partition the mesh, serve as a control handle for the deformation, and provide a scalar field that parameterizes the region of interest. This parameterization can be used to easily

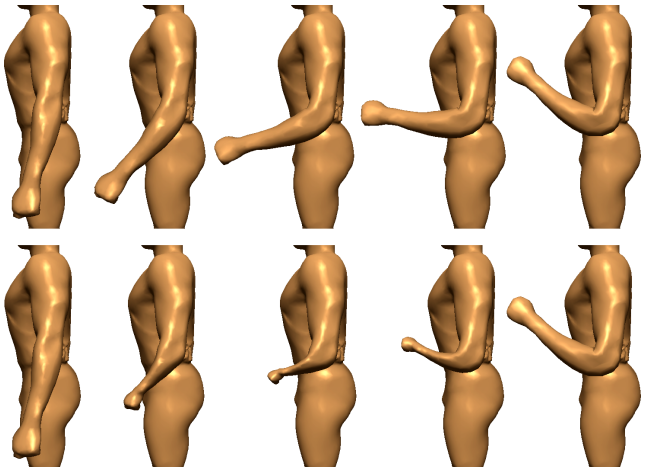

Figure 15: Interpolating our deformation parameters generates much more natural in-between frames than linear interpolation of vertex positions.

control additional deformation parameters such as twist and scaling. As our method does not use any fixed skeletal structure, users have great freedom in choosing the way in which the surface will be deformed. Since our system is based on simple 2-D sketching operations, it is both intuitive and easy to use, and allows users to quickly create deformed objects. It is also relatively appealing to non-expert users. Furthermore, our deformation method can provide for natural morphing between two key frames by using the sketch curves as implicit skeletons.

Our current system is a very effective tool, but there are also numerous ways in which it could be improved and extended. Our current approach is targeted more towards reposing bodies and limbs. Editing fine-grained surface features, such as the shape of the eyes on a face, is more difficult. Extending our sketch-based methodology to support editing of such surface features would be very desirable. Although our method is quite fast and suitable for interactive editing of fairly large models, the performance could be improved by incorporating multiresolution techniques such as [Lee et al. 2000]. Extending skeleton-based morphing into a complete system for easily creating simple key-framed animations is another very appealing direction.

## References


ALEXA, M., COHEN-OR, D., AND LEVIN, D. 2000. As-rigid-as-possible shape interpolation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 157–164.

BENDELS, G. H., AND KLEIN, R. 2003. Mesh forging: Editing of 3d-meshes using implicitly defined occluders. In *Symposium on Geometry Processing 2003*.

BLOOMENTHAL, J. 2002. Medial-based vertex deformation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, 147–151.

COQUILLART, S. 1990. Extended free-form deformation: a sculpturing tool for 3d geometric modeling. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, ACM Press, 187–196.

CORRÊA, W. T., JENSEN, R. J., THAYER, C. E., AND FINKELSTEIN, A. 1998. Texture mapping for cel animation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, 435–446.

HUA, J., AND QIN, H. 2003. Free-form deformations via sketching and manipulating scalar fields. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, ACM Press, 328–333.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: a sketching interface for 3d freeform design. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 409–416.

KATZ, S., AND TAL, A. 2003. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph. 22*, 3, 954–961.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, 105–114.

KOBBELT, L. P., BAREUTHER, T., AND SEIDEL, H.-P. 2000. Multiresolution shape deformations for meshes with dynamic vertex connectivity. In *EUROGRAPHICS 2000*.

LAWRENCE, J., AND FUNKHOUSER, T. A. 2003. A painting interface for interactive surface deformations. In *Pacific conference on computer graphics and applications*.

LEE, A., MORETON, H., AND HOPPE, H. 2000. Displaced subdivision surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 85–94.

MACCRACKEN, R., AND JOY, K. I. 1996. Free-form deformations with lattices of arbitrary topology. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, 181–188.

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, ACM Press, 151–160.

SINGH, K., AND FIUME, E. 1998. Wires: a geometric deformation technique. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, 405–414.

SNYDER, J. M., AND KAJIYA, J. T. 1992. Generative modeling: a symbolic system for geometric modeling. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, ACM Press, 369–378.

WELCH, W., AND WITKIN, A. 1994. Free-form shape design using triangulated surfaces. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM Press, 247–256.

YOSHIZAWA, S., BELYAEV, A. G., AND SEIDEL, H.-P. 2003. Free-form skeleton-driven mesh deformations. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, ACM Press, 247–253.

YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2004. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph. 23*, 3, 644–651.