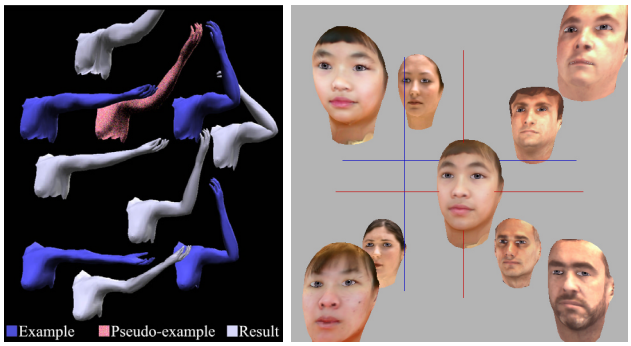
Shape by Example

Peter-Pike J. Sloan
ppsloan@microsoft.com

Charles F. Rose, III
chuckr@microsoft.com

Michael F. Cohen
mcohen@microsoft.com

Microsoft Research



Abstract

Modern modeling systems enable artists to create high-quality content, but provide limited support for interactive applications. Although complex forms can be constructed either by hand or with geometry capture technologies, once they are created, they are difficult to modify, particularly at runtime.

Interpolation provides a way to leverage artist-generated source material. We present a methodology for efficient runtime interpolation between multiple forms. Linear plus radial basis functions provide the key mathematical support for the interpolation. Once our system is provided with example forms, it generates a continuous range of forms we call a shape. We also apply the shape interpolation methodology to articulated figures and human face models to create smoothly skinned figures that deform in natural ways. Unlike previous formulations, the one presented here is efficient enough to support interactive design of the abstract interpolation space as well as support runtime interpolation of the forms in interactive applications such as games.

The reader is encouraged to visit the project's website for more information and late-breaking results at *http:\\research.microsoft.com\graphics\hfap.*

Keywords: Curves & Surfaces, Geometric Modeling, Human Body Simulation, Interpolation, Radial Basis, Shape Blending

1 Introduction

The magic of computer graphics as seen in many current movies and computer games comes at a cost. Creating the geometric forms with which to generate synthetic characters and animating the characters to bring them to life requires either highly skilled artists and/or sophisticated capture devices. Both are expensive and, in the case of highly skilled artists, rare. This paper discusses and demonstrates a methodology to automatically create new shapes from existing geometric forms. The paradigm presented here is one of *design by example*. New shapes are created on the fly through multi-way blending of examples. We also apply the shape interpolation methodology to articulated figures to create smoothly skinned figures that deform in natural ways. Unlike previous formulations, the one presented here is efficient enough to support interactive design of the abstract interpolation space as well as support runtime interpolation of the forms in interactive applications such as games. Our approach also differs from previous work by allowing for both extrapolation and interpolation between multiple forms.

Skilled artists using current modeling software can construct complex geometric shapes. Alternatively, a variety of 3D scanning methodologies are available that can capture shapes that exist in the real world. Both of these means of creating shapes are limited in the same way. In particular, neither has a simple means of automatically modifying the shapes once they have been created. In scripted settings, such as films, automatic modification would make it easier to avoid redundancy. For instance, an entire colony of different looking ants could be automatically created from a few distinct ants. Another desire might be to create smooth variations of shape within an individual such as the bulging of a muscle when an arm is bent. In addition, in non-scripted interactive runtime settings, such as games, it may not be possible to anticipate all shapes that will be needed when the game is played.

There are at least two solutions to the issues outlined above. At a high cost, more shapes can be created. An alternate solution is to try to leverage existing forms to automatically generate variations on the fly. The second approach is the topic of the work presented here.

We will present a methodology for efficient interactive design of interpolation spaces as well as runtime interpolation between multiple forms. The forms may have been created by artists or through geometry capture technologies. Once the system is provided with *example* forms, it can generate a continuous range of forms we call a *shape*. We also apply the shape blending methodology to articulated figures to create smoothly skinned figures that deform in natural ways by leveraging the transform blending operations in current graphics hardware. Finally, since the same formulation can be applied to images or texture maps, we show the system used to interpolate between human face models by blending both shape and texture.

2 Related work

The idea of leveraging existing shapes and animations by modifying them is certainly not new. Morphing methods, initially applied to images [16] [1], have recently also been applied to 3D geometry [17, 4, 2, 6, 3]. Interpolation methods were applied to 2D vector-based drawings in [8, 11]. Much of the morphing methodology concentrates on establishing correspondences between models and/or is limited to morphing between two models. An exception is the work on N-way morphing in the case of images[5]. In our work, we assume that the correspondence problem has either been solved implicitly in the creation of the original forms or that existing methods such as these can be used to establish correspondences. We focus, instead, on the problem of efficient blending between multiple *example* forms.

Rossignac [15] presented a method to blend between an initial and final shape influenced by intermediate shapes. Essentially, it is a weighted blend of the meshes where the weights are defined by a bezier curve. The only shapes that are interpolated in this construction are the initial and final one, thus it provides only a one-dimensional parameterization of the blending function.

Rademacher's study [13] is similar in spirit to the work presented here. He parameterizes geometry based on view direction and always uses three-way blends. He also applies his method to animated view-dependent geometry. Our work uses the more general notion of an abstract space to parameterize the object. In addition, we use scattered data interpolation, instead of simplex decomposition (which becomes intractable in higher dimensions), to do the blending. Our work also allows us to leverage the underlying anatomical structure in skeleton based figures.

Rose *et al.*'s paper "Verbs and Adverbs" [14] has shown results in blending of animations. We adopt a similar interpolation structure as the one described there and apply it to shape blending. However, rather than solving a linear system per degree of freedom (e.g., vertex coordinates or motion curve coefficients), our formulation solves a linear system *per example pose* that can be shown to be *mathematically equivalent*. We also show how such blending can be extended to articulated shapes and to textures.

A recent work by Lewis *et al.* [7] applies radial basis interpolation to articulated shapes. There are several key differences from our work. One is that, like in [14], they set up the problem as a linear system per degree of freedom. Since there are typically many more vertex coordinates in the mesh then there are examples, the system in [7] is significantly less efficient. Although there are no explicit timing results given in [7], based on the video presented, it appears our system is perhaps two orders of magnitude faster.

In addition, per example blending can be mapped directly to current graphics hardware support for transform blending as shown in section 5. Finally, as in [14], we use a combination of radial basis functions plus a linear hyperplane. Without the linear component, you cannot reproduce linear or constant changes between the examples, which can result in higher energy (wiggly) interpolation spaces. The addition of the hyperplane to the formulation allows for extrapolation and results in a smoother interpolation. The efficient formulation provides the ability to interactively parameterize (and reparameterize, see section 4) the abstract interpolation space. This is very powerful as a modeling tool, and is only possible if the deformations are represented as functions over example poses. We will show that our formulation is efficient enough for both interactive (re)parameterization and runtime blending.

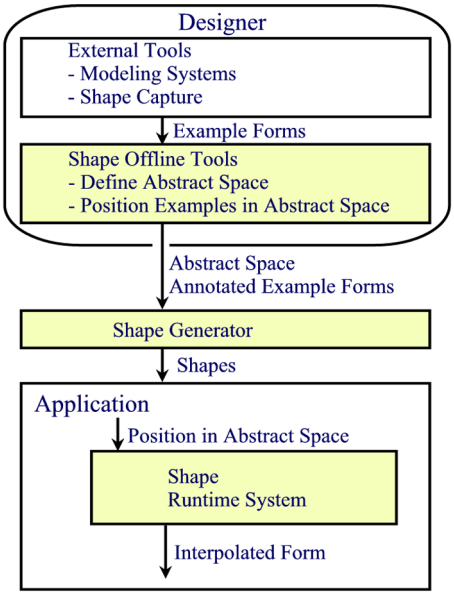


Figure 1: System overview

3 Shapes and Adjectives

Figure 1 provides an overview of the three parts that comprise our system: Shape Offline Tools, the Shape Generator, and the Shape Runtime System. The latter is embedded in an application, such as a game, that uses our system to modify a shape or skin a character as it is animated.

A designer relies on external modeling and/or geometry capture technology to create initial forms. We refer to these as example forms or simply *examples*. The Shape Offline Tools enable the designer to organize these *example* forms to serve as input into our Shape Generator. To do so, the designer chooses a set of adjectives that characterize the forms. For instance, adjectives that describe the form of a human may include static aspects such as gender and age. These axes are of interest because the form of a human arm changes depending on whether it belongs to a male or female or whether the person is old or young. Other "adjectives" may describe more dynamic aspects, such as the bend of the elbow, since the arm also deforms when the skeleton bends. In the latter case we are not referring to the rigid body transformations induced by, for instance, bending the elbow, but rather the more subtle non-rigid shape changes in muscles and skin.

The set of adjectives define an *abstract space* with each adjective representing a separate axis in the space. Once the abstract space has been defined, each example is annotated by the designer by setting values for each adjective. For example, this could include indicating the age of an example shape and the pose. Given the annotated examples, the Shape Generator solves for the coefficients of a smoothly varying interpolation of the forms across the abstract space. These coefficients provide the means to interpolate between the *example* forms at runtime.

At runtime, the application chooses, at each moment in time, desired values for the adjectives, thus selecting a specific location in the abstract space. For instance, an arm

Object	Variable	Subscript	Range
Example	X	i	$1..N$
DOF	x	i	$1..N$
		j	$1..M$
Point in Abstract Space	$\mathbf{p}$	i	
Radial basis	R	i	
Radial Coefficient	r	i, j	
Linear Basis	A	l	$0..D$
Linear Coefficient	a	i, l	
Distance	d	i	

Table 1: Terminology

can be specified to be more male or female, or to respond to the bending of the elbow. The Shape Runtime System then, given the selected location in the abstract space, efficiently blends the *examples* to produce an interpolated form.

The number of adjectives defines the dimension of the abstract space. We use D to denote the dimension. We denote the number of *examples* included in the construction of a shape by N. Each example form in a shape is required to have the same structure. That is, all forms in a shape must have the same number of vertices with the same connectivity. Since *example* forms must all have the same topological structure, we do not need to address the correspondence problem here. The number of DOFs, denoted by M, equals three times the number of vertices in a form (i.e., for the x,y,z coordinates).

As we describe more details of our approach, please refer to Table 1 for an explanation of the symbols used throughout the text.

We denote an *example* as X_i, where

$$X_i = \{x_{ij}, \mathbf{p}_i : i = 1 \ldots N, \\ j = 1 \ldots M\} \tag{1}$$

Each x_{ij}, the j^{th} DOF for the i^{th} *example* represents a coordinate of a vertex. $\mathbf{p}_i$ is the location in the abstract space assigned to the *example*.

3.1 Shape Generation

Given a set of *examples*, a continuous interpolation over the abstract space is generated for the shape. The goal is to produce at any point $\mathbf{p}$ in the abstract space a new form $X(\mathbf{p})$ derived through interpolation of the *examples*. When $\mathbf{p}$ is equal to the position p_i for a particular *example* i, then $X(\mathbf{p_i})$ should equal X_i. In between the *examples*, smooth intuitive changes should take place.

In Lewis *et al.* [7] each vertex coordinate was treated as a separate interpolation problem. Similarly, in Rose *et al.* [14], each B-spline coefficient was treated separately (1200 separate problems in their walk verb) which leads to serious inefficiencies. We develop, instead, a *cardinal basis* where we associate one basis function with each *example*. This leads to greatly improved efficiency as there are typically many fewer examples than degrees of freedom (vertices in a mesh or coefficients of motion curves in an animation). As a cardinal basis, each basis function has a value of 1 at the *example* location in the abstract space and a value of 0 at all other *example* locations. This specification guarantees an exact interpolation. For each DOF, the bases are then simply scaled by the DOF values and summed. This approach is mathematically equivalent to the formulation in Rose *et al.*, but is (a) more efficient in the case of animations, (b) can be applied to shape interpolation, and (c) is amenable to implementation on current graphics hardware.

We still need to select the shape of the individual cardinal basis functions. Our problem is essentially one of scattered data interpolation, as we have few data points, the *examples*, in a relatively high dimensional space. Most published scattered data interpolation methods focus on one and two-dimensional problems. Linear interpolation using Delauney triangulation, for instance, does not scale well in high dimensions. Based on our need to work in high dimensions with very sparse samples, we adopt a combination of radial basis functions and low order (linear) polynomials.

The N cardinal basis functions have the form

$$w_{i_1}(\mathbf{p}) = \sum_{i_2=1}^{N} r_{i_2 i_1} R_{i_2}(\mathbf{p}) + \sum_{l=0}^{D} a_{i_1 l} A_l(\mathbf{p}) \tag{2}$$

where the $r_{i_2 i_1}$ and R_i are the radial basis function weights and radial basis functions themselves and the a_{i_l} and A_l are the linear coefficients and linear bases. The subscripts i_1 and i_2 both indicate *example* indices. Given these bases, the value of each DOF is computed at runtime based on the momentary location, $\mathbf{p}$, in the abstract space. The value of each DOF, x_j, at location $(\mathbf{p})$ is, thus, given by:

$$x_j(\mathbf{p}) = \sum_{i_1=1}^{N} w_{i_1}(\mathbf{p}) x_{i_1 j} \tag{3}$$

.

The linear portion of the basis functions provide an overall approximation to the space defined by the *examples* and permits extrapolation outside the convex hull of the locations of the *examples*. The radial bases locally adjust the solution to exactly interpolate the *examples*. We discuss the linear approximation and radial bases in more detail below.

3.2 Linear Approximation

We first would like to form a best (in the least squares sense) linear approximation for each DOF based on the *examples* given for that DOF. In other words, we would like to find the hyperplane through the abstract space that comes closest to approximating the *example* values of that DOF. This defines M separate least squares problems, one for each DOF variable. However, since each *example* places all variables at a single location, $\mathbf{p}$, we can instead determine a set of N hyperplanes, one for each *example*, that form a basis for the M hyperplanes. The basis hyperplanes are derived by fitting a least squares hyperplane to the case where one *example* has a value of 1 and the rest have a value of 0.

$$\mathbf{p}_h \mathbf{a} = \mathbf{F},$$

where $\mathbf{p}_h$ is a matrix of homogeneous points in the abstract space (i.e., each row is a point location followed by a 1), $\mathbf{a}$ are the unknown linear coefficients, and $\mathbf{F}$ is a *Fit* matrix expressing the values we would like the linear approximation to fit. In this case $\mathbf{F}$ is simply the identity matrix since we are constructing a cardinal basis. Later, $\mathbf{F}$ will take on a slightly different structure as we discuss reparameterization of the abstract space.

For didactic purposes we depict a simple one dimensional abstract space with three examples. Figure 2 shows the linear approximation and the three radial basis functions associated with the first of the three *examples*. The sum of the line plus three radial bases forms one of the three cardinal bases. The one-dimensional abstract space is depicted

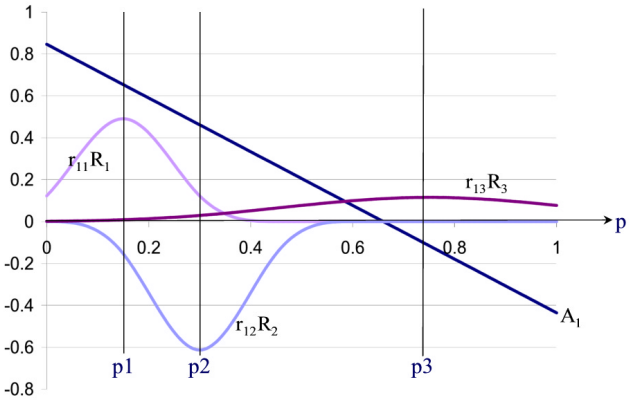


Figure 2: Linear and radial parts of the cardinal basis function for the first *example*.

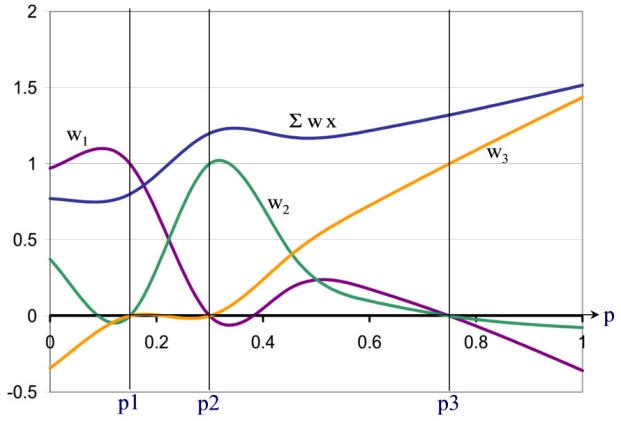


Figure 3: Cardinal basis functions and scaled sum for particular degree of freedom.

as the horizontal axis. The three *examples* are located at $\mathbf{p} = 0.15, 0.30, 0.75$. The straight line labeled A_1 is the line that fits best through $(0.15, 1), (0.30, 0), (0.75, 0)$. A best fit line for any particular DOF could then be evaluated by simply scaling this line (plus the other two not shown) by the DOF value and summing.

3.3 Radial Basis

Radial basis function are characterized by having a value that is determined solely by one parameter, the *distance* from a center point in the multi-dimensional abstract space. Radial basis interpolation is discussed in Micchelli [10] and in the survey article by Powell [12]. Radial basis functions have been used in computer graphics for image warping by Ruprecht and Müller [16], Arad *et al.* [1] and for 3D interpolation by Turk *et al.* [17].

Given the linear approximation, there still remain residuals between the *example* values x_{ij} and the scaled and summed hyperplanes. To account for the residuals, we could associate a radial basis with each DOF. Instead, we proceed as before and use the radial bases to account for the residuals in the cardinal bases. The residuals in the cardinal bases are given by

$$q_{i_1 i_2} = \delta_{i_1 i_2} - \sum_{l=0}^{D} a_{i_2 l} A_l(\mathbf{p}_{i_1}) \tag{4}$$

To correct for these residuals, we associate N radial basis functions with each *example*. Since there are N *examples*, we, thus, need N^2 radial basis functions. We solve for the weights of each radial bases, $r_{i_2 i_1}$, such that, when the weighted radial bases are summed with the hyperplanes, they complete the cardinal bases (i.e., pass through 1 at the example location and zero at the other example locations). The three curves shown in Figure 2 are the radial bases associated with the first example.

This leaves us with the problem of choosing the specific shape of the radial bases and determining the radial coefficients. Radial basis functions have the form:

$$R_i\left(d_i(\mathbf{p})\right)$$

where R_i is the radial basis associated with X_i and $d_i(\mathbf{p})$ is a measure of the distance between $\mathbf{p}$ and $\mathbf{p}_i$, most often the Euclidean norm $\|\mathbf{p} - \mathbf{p}_i\|$. There are a number of choices for this basis. Rose *et al.* chose a basis with a cross section of a cubic B-spline centered on the *example* and with radius twice the Euclidean distance to the nearest other *example*. Turk and O'Brien [17] selected a family of basis functions that generate the smoothest (thin plate) interpolations. We tried both bases. Bases with the B-spline cross-section have compact support, thus, outside their support, the cardinal bases are simply equal to the linear approximation. They therefore extrapolate better than the smoother, but not compact, radial basis functions used by Turk and O'Brien. We chose to use the B-spline bases for this extrapolation property, although both choices performed well.

The radial bases weights, $r_{i_2 i_1}$, can now be found by solving the matrix system,

$$\mathbf{Qr} = \mathbf{q}$$

where $\mathbf{r}$ is an NxN matrix of the unknown radial bases weights, $\mathbf{Q}$ is defined by the radial bases such that $\mathbf{Q}_{i_1 i_2} = R_{i_2}(p_{i_1})$, the value of the unscaled radial basis function centered on *example* i_2 at the location of *example* i_1, and $\mathbf{q}$ is the matrix of residuals between the kronicker delta and the hyperplane approximation (Equation 4). The diagonal terms of $\mathbf{Q}$ are all 2/3 since this is the value of the generic cubic B-spline at its center. Many of the off diagonal terms are zero since the B-spline cross- sections drop to zero at twice the distance to the nearest *example*.

In practice, solving for the linear and radial portions of the cardinal bases takes a fraction of a second. Thus, this portion of the process can be tightly coupled in the design loop providing artists with instant feedback on the abstract space and an ability to move the examples within the space interactively.

Referring back to Figure 2, we see the three radial basis functions associated with the first of the three *examples*. If these three are summed with the linear approximation, we get the first cardinal basis, the line labeled w_1 in Figure 3. Note that it passes through 1 at the location of the first *example* and is 0 at the other *example* locations. The same is true for the other two cardinal bases w_2 and w_3.

3.4 Shape Runtime System

Given the solutions for the radial basis weights, we now have all the values needed to evaluate Equations 2 and 3 at run-

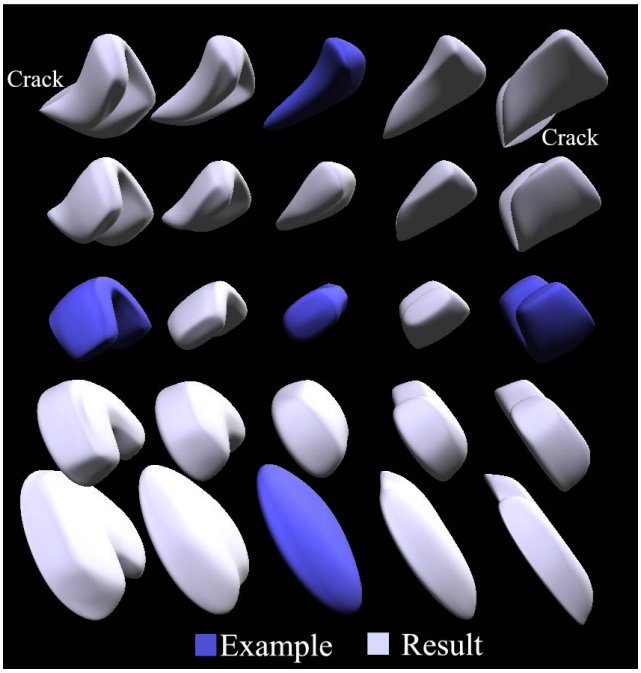


Figure 4: Exploration of the space can reveal problems with the interpolation.

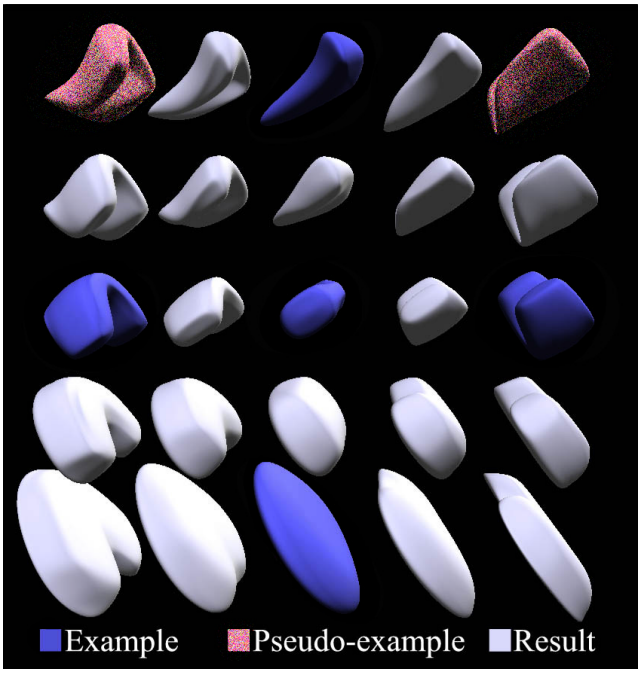


Figure 5: *Pseudo-examples* (the upper right and left corner forms) can reparameterize the space, fixing up problem regions.

time. The upper ($\sum w\mathbf{x}$) line in Figure 3 is simply an instance of the three cardinal bases scaled by an *example* DOF value and summed as in Equation 3.

At runtime, the application selects the point of interest in the abstract space. This point may move continuously from moment to moment as the elbow bends for instance. The Shape Runtime System takes this point and generates an interpolated form and delivers it to the application for display.

4 Reparameterization and Modification of Shapes

Once a shape has been generated, the abstract space can be quickly explored to see the interpolated forms. It is possible that some of the regions in this space are not entirely to the designer's liking. For instance, Figure 4 shows a 2D space of interpolations of a simple three-dimensional shape. Darker forms indicate *examples* and lighter forms are the result of the Shape Runtime System (Blue and gray on the color plate). Problem regions can be seen in each of the two upper corners. Both corners exhibit cracks due to surface interpenetration. Another type of problem can occur when an interpolation changes more quickly than desired in one region and more slowly than desired in a neighboring region, in other words one would like to "bend" the abstract space itself.

Three methods are available to modify the results produced by the Shape Generator. In one method a new *example* can be inserted at the problem location and a new shape generated. A quick way to bootstrap this process is to use the undesirable results at the problem location as a starting point, fix up the offending vertices and reinsert this into the abstract space as a new *example*.

In the second method, the designer can modify the abstract space using the current set of examples by moving the locations of the examples in the space. The cardinal bases must then be resolved for, but this is fast enough to be done interactively as the examples are moved.

The above method, however, may not be desirable since the examples were typically placed at specific locations with a conceptual purpose and the artist may not want to move them. A third approach is to leave the original example in place. Instead, one can first selects an acceptable interpolated form from a location in the abstract space near a problem region (call this position $\mathbf{p}$). Next, move this form to a new location ($\tilde{\mathbf{p}}$) in the problem region. The relocated form will be treated as if it were a new *example*, however as discussed below the computational and storage complexity is less. We call this relocated form a *pseudo-example*.

The *pseudo-example* is not a "true" *example* as it is a linear sum of other *examples*. However, it acts like an *example* in terms of changing the shape of the radial basis functions. In essence this method *reparameterizes* the abstract space. Reparameterization is very effective in producing shapes of linked figures, as we demonstrate later with a human arm shape. Figure 5 shows this technique applied to the 2D space of 3D forms. The space has been reparameterized with two *pseudo-examples* shown with a grainy texture (red on color plate). Arrows indicate the locations from which the *pseudo-examples* were drawn.

Creating the reparameterized shape proceeds much as before. We need to introduce some new notation before we can formalize the reparameterization approach. We have already mentioned $\tilde{\mathbf{p}}$, the new location of the *pseudo-examples*. $\tilde{\mathbf{p}}$ also includes the real *example* locations during reparameterization. We, furthermore, use a tilde to denote the new radial basis weights, $\tilde{r}$, the new linear coefficients $\tilde{a}$, and the new cardinal bases $\tilde{w}$. We also denote the total number of real and *pseudo-examples* as $\tilde{N}$.

As before, the form at any point in the abstract space is:

$$x_j(\mathbf{p}) = \sum_{i_1=1}^{N} \tilde{w}_{i_1}(\mathbf{p}) x_{i_1 j} \tag{5}$$

Note that the interpolated form still only includes a sum over the real *examples*. In other words, there are still only N cardinal bases after the reparameterization. The *pseudo-examples* reshape these N bases from w to $\tilde{w}$.

$$\tilde{w}_{i_1}(\mathbf{p}) = \sum_{i_2=1}^{\tilde{N}} \tilde{r}_{i_2 i_1} R_{i_2}(\mathbf{p}) + \sum_{l=0}^{D} \tilde{a}_{i_1 l} A_l(\mathbf{p}) \tag{6}$$

Also as before

$$\tilde{\mathbf{p}}_h \tilde{\mathbf{a}} = \tilde{\mathbf{F}}$$

However, $\tilde{\mathbf{F}}$ is no longer an identity matrix. It now has $\tilde{N}$ rows and N columns. Assuming the new *pseudo-examples* are all located at the bottom of the matrix, the top NxN square is still an identity. The lower $\tilde{N} - N$ rows are the values of the original cardinal bases w at the location where the *pseudo-examples* were taken from (see Equation 6). That is, in each row, the *Fit* matrix contains the desired values of the N cardinal bases, now at $\tilde{N}$ locations. These are 1 or 0 for the real *example* locations and the original cardinal weights for the *pseudo-examples*, in other words, the weights at the locations from which the *pseudo-examples* were drawn.

The radial portion of the cardinal basis construction proceeds similarly. The residuals are now

$$\tilde{q}_{i_1 i_2} = \tilde{F}_{i_1 i_2} - \sum_{l=0}^{D} \tilde{a}_{i_2 l} A_l(\mathbf{p}_{i_1})$$

Note that instead of the Kronecker delta we now have the values from $\tilde{F}$.

The coefficients, $\tilde{r}_{i_2 i_1}$, are now found by solving the matrix system,

$$\mathbf{Q}\tilde{\mathbf{r}} = \tilde{\mathbf{q}}$$

As before, $\mathbf{Q}_{i_1 i_2}$ has terms equal to $R_{i_2}(\tilde{p}_{i_1})$, however, these terms now include the new *pseudo-example* locations. As a consequence, the radii of the radial basis functions may change.

5 Shapes and Skeletons

Animated characters are often represented as an articulated skeleton of links connected by joints. Geometry is associated with each link and moves with the link as the joints are rotated.

If the geometry associated with the links is represented simply as individual rigid bodies, these parts of the character will separate and interpenetrate when a joint is rotated. A standard way to create a continuous *skinned* model is to smoothly blend the joint transforms associated with each link of the character, in other words, vertices near a joint will respond to a blend of the transforms associated with both links on either side of the joint. This is supported in current graphics hardware, making it an attractive technique.

Unfortunately, transform blending exhibits some problems. These problems were recognized by Lewis *et al.* [7] who present results similar to those presented here. As noted earlier, in contrast to Lewis *et al.* our solution allows interactive editing of the abstract space due to the single weight per example formulation with no loss of authoring flexibility.

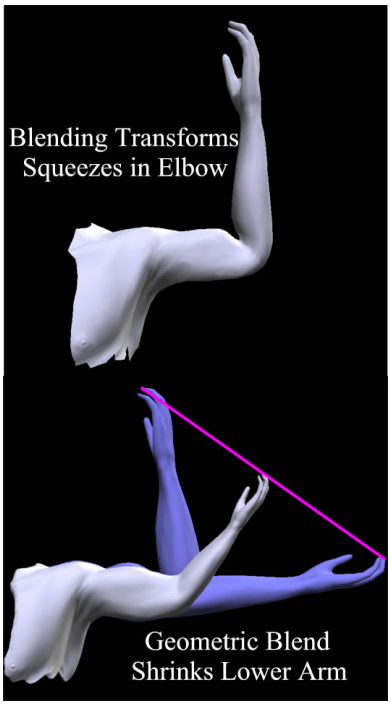


Figure 6: Simple transformation blending exhibits shrinking about joints.

This also provides real-time interaction since it can leverage the transform blending in graphics hardware.

We use an arm bending at the elbow to first discuss the problems associated with naive transform blending and then to demonstrate how we can overcome these difficulties in the context of shape blending. To perform the simple transform blending, a blending weight α is assigned to each vertex. For instance, in an elbow bend, vertices sufficiently below the elbow would have weight $\alpha = 1$, and those sufficiently above, $\alpha = 0$. Those vertices near the elbow would have α values between 0 and 1.

We denote the transformation matrix for the upper arm as T_0, and the transformation for the lower arm as T_1. We use superscripts to denote the amount of rotation of the joint. Thus, as the elbow rotates, we say that T_1 changes from T_1^0 when the elbow is straight to T_1^1 at a bent position. In between the transform is T_1^β for a bending amount β. We refer to the position of the arm when $T_1 = T_1^0$ as the *rest position*.

We denote the position of a vertex as x_0 when the transformation of the joint is T_1^0. The simple transform blending is defined as

$$x = \alpha T_1^\beta x_0 + (1 - \alpha) T_0 x_0$$

where α is the blending weight assigned to that vertex. Unfortunately, linearly summed transformation matrices do not behave as one would like. The result is that the skin appears to shrink near the rotating joint (see Figure 6). In addition, simple transform blending does not provide a rich set of tools for creating effects such as a muscle bulging as the elbow bends.

We can solve both problems with shape blending. To do this, the designer first creates *examples* of a straight arm, $X_{\beta=0}$, and a bent arm, $X_{\beta=1}$, for, say, a 90 degree bend (Figure 7). The bent arm includes any muscle bulging or

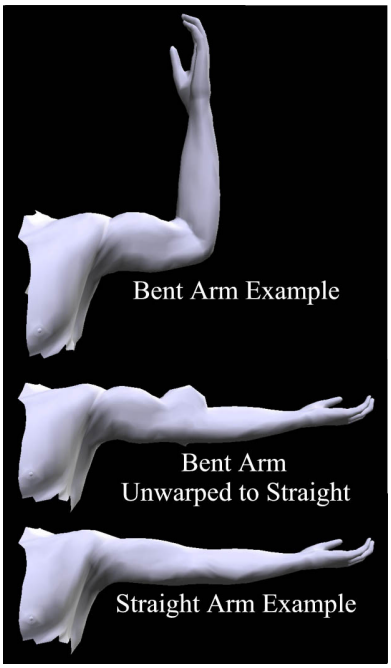


Figure 7: *Example* forms are warped into a canonical pose.

other deformations the designer wants to include. We obviously cannot simply perform a direct geometric blend between these forms. At 45 degrees the lower arm would have shrunk considerably, since the chord from the fingertip of the bent arm to the fingertip of the straight arm passes nearer to the elbow than an arc would.

Instead, what we would like is a combination of shape blending and transform blending. We will do this by first "unbending" the bent arm into the rest position in such a way that when it is passes through the transform blending it will exactly match the bent arm originally specified by the designer. These produce the strange forms seen in Figure 7. To perform the interpolation, these unbent forms are first blended as usual, this new interpolated form is then sent through the blended transform.

More formally, we call the vertex on the bent arm, corresponding to x_0 on the arm in the rest position, x^1. As in the simple blended transforms, we also have a blending weight, α, associated with each vertex. We now seek a second arm in the rest position with a corresponding vertex, x_0^1, such that when subjected to the simple transform blending at an angle of $\beta = 1$ it will exactly match the bent arm specified by the designer. Thus

$$x^1 = \alpha T_1^{\beta=1} x_0^1 + (1-\alpha) T_0 x_0^1$$

Now we can solve for the vertices of this new arm in the rest position (see Figure 7)

$$x_0^1 = (\alpha T_1^{\beta=1} + (1-\alpha) T_0)^{-1} x^1 \tag{7}$$

Finally, we perform a geometric blend of the new arm in the rest position with the original arm in the rest position and then transform it. The result is

$$x_\beta = (\alpha T_1^\beta + (1-\alpha) T_0)(\beta x_0^1 + (1-\beta) x_0) \tag{8}$$

This geometric blend followed by the blended transform will match the original arm geometry when $\beta = 0$ and will match the bent arm created by the designer when $\beta = 1$.

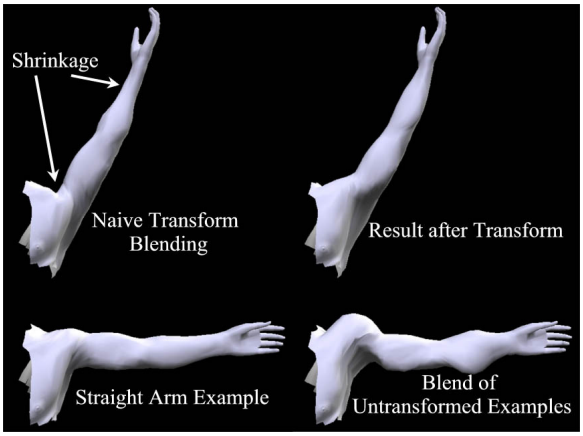


Figure 8: Naive transform blending vs. interpolated blending.

The arm model discussed above contains only two *example* forms, the arm in the rest position and the bent arm. But we can now use all the machinery of the previous sections to perform a multi-way blend of the articulated geometry. First, all *example* forms are untransformed to the rest position via Equation 7. Then the multi-way shape blending is applied to the untransformed forms. Finally, the blending result is transformed in the normal way. In other words, the interpolation of the forms in the rest position simply replaces the latter half of Equation 8. Figure 8 shows the straight arm *example* one of 6 *examples* in this shape. The results above the straight arm are the result of naively blending transforms on this one *example*. On the bottom right is a blend of the 6 *examples* untransformed to the rest position. Finally, this strange blended form is pushed through the blended transformation to the give the result shown in the upper right.

6 Results

We have applied the paradigm described above to simple shapes, linked figures, and to human face models. All performance statistics measure raw interpolation speeds and do not include time to render. All timings were gathered on a 450 Mhz *Pentium-II* machine.

6.1 Simple Shapes

The shape demonstrated in Figure 5 includes 5 *example* forms each consisting of 642 vertices (1926 DOFs) and 1280 faces. One step of Loop subdivision [9] is applied to this mesh and the limit positions and normals are computed. This results in a mesh with 2562 vertices (7686 DOFs) and 5120 faces that is rendered at runtime. A two dimensional abstract space was created with the horizontal axis being "bend" and the vertical being "thickness." The initial space had regions in the upper right and left corners where the blended surface was interpenetrating - the space was reparameterized by adding *pseudo-examples* near these locations that were not interpenetrating. Blending in this space can be done at 5500 frames per second (fps). If real *examples* had been added instead of *pseudo-examples*, the blending slows down to around 4000 fps.

The abstract space can also be modified at runtime by moving the *examples* and/or creating *pseudo-examples*. This

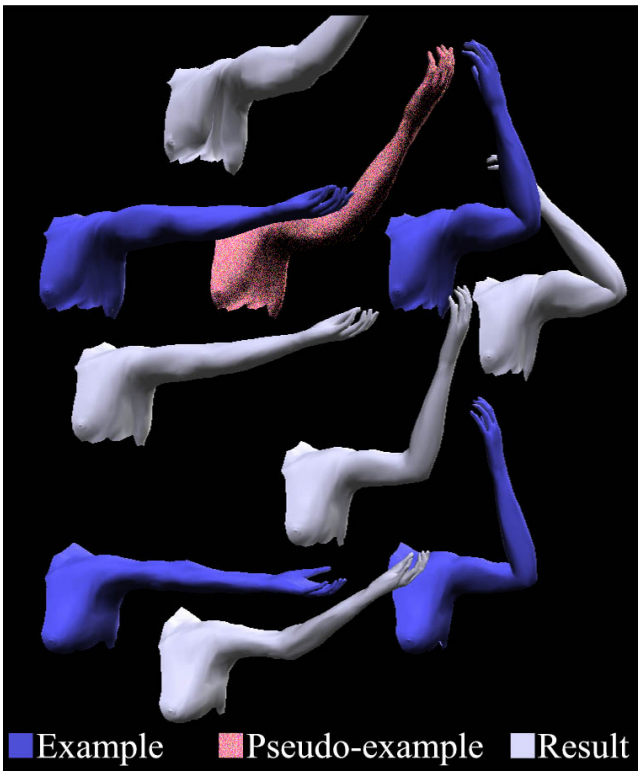


Figure 9: This figure shows 4 *example* arms (darker), one *pseudo-exmaple* (grainy), and five interpolated arms (lighter). Two axes of the abstract space shown are gender in the vertical dimension and elbow bend horizontally. Note that the arm can bend more and become more muscular than any example through extrapolation.

requires that the shape be resolved for new basis function coefficients. The formulation presented in this paper allows this step to be done within the interactive loop. The solver plus interpolation still maintains more than 30 frames per second.

6.2 Arm

The arm was created by modifying *Viewpoint* models inside *3D-Studio/Max*. The arm *example* has an underlying skeleton that has 3 rotational degrees of freedom - one for the shoulder, elbow and wrist. We have a fourth variable for the parameterized arm, gender. The abstract space has 8 real *examples* (straight arm, shoulder up, shoulder down and elbow bent for male and female models) and 6 *pseudo-examples* which are mostly placed to smooth out bulges where the shrinking from blended transforms was being overcompensated for. These *examples* have 1335 vertices (4005 DOF) and 2608 faces each. This dataset can be interpolated at 2074 fps. As with the simple blob shape, the arm's abstract space can be constructed and reparameterized within the interactive loop within a frame time.

Figure 9 is a visualization of a 2D slice of the 4D abstract space space of arms with 4 real *examples*, 1 *pseudo-example*. Elbow bend is parameterized along the horizontal axis and gender along the vertical. Extrapolation produces arms that are more bent and more muscular than any of the examples.

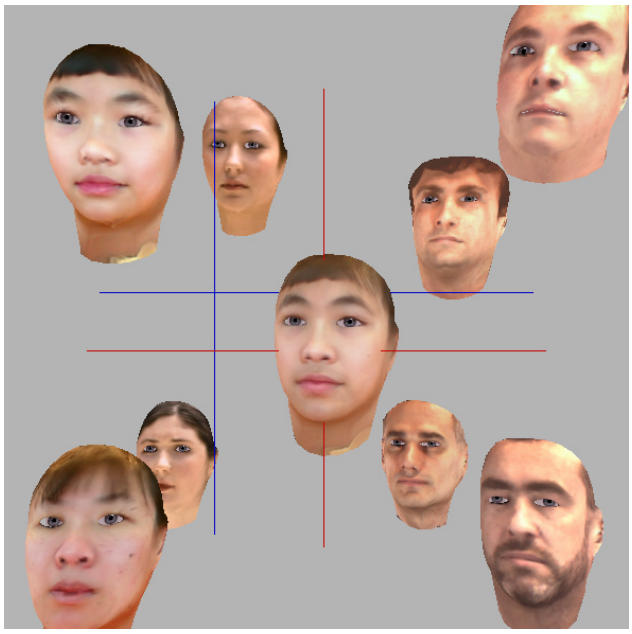


Figure 10: Both geometry and texture are interpolated separately to create an infinite variety of faces at runtime. The figure shows 8 examples plus the interpolated result. The crosshair centered outside the head indicates the position in the abstract space at which the texture is interpolated. The crosshair centered on the head indicates the interpolated face and also the position for the geometry interpolation.

6.3 Faces

We also applied the shape interpolation to human face models. The face model is constructed by modifying a generic face mesh by linear combinations of "metrics" represented as vertex offsets. Some metrics change the overall form while others adjust more local details such as the shape of the nose and/or mouth. Taken together, they can generate a very wide range of human facial forms. These forms can then be animated to blink, smile, talk, etc. The resulting face forms are texture mapped to create further differentiation.

Figure 10 shows a set of example face models positioned in a 3D abstract space. An interpolation can be carried out independently on the geometry of the faces and/or on the textures associated with them. The interpolated face shape can also be animated at the same time while maintaining frame rates. Thus from a few example faces, an application could generate an infinite variety of faces at runtime.

7 Conclusions

Shape interpolation has been shown to be an effective and highly efficient way of altering shape for real-time applications such as computer games. The front-end solution process is also efficient, (a fraction of a second), so a tight coupling between artist, solver, and interpolator provides a new way for an artist to work without fundamentally altering their workflow. With our system, artists can more easily extend their work into the interactive domain.

In the context of skeleton based figures, we are able to combine both shape blending with blended transforms to create a smoothly skinned character. We overcome the limitations of blended transforms, while including the artist's

input for how muscles deform as the skeleton moves, and still maintain interactive performance.

We intend to continue enhancing this work. In order to improve our design cycle time, we intend to incorporate the system into a commercial 3D package. While our processor requirements are low, improving efficiency is still a key goal. We would like to be able to control many realistic characters at once with very small processor budgets – on the order of one or two percent of the processor budget. Memory usage patterns and usage of SIMD floating point hardware such as found on the *Katmai* (*Pentium-III*) chip will be explored.

Shape interpolation poses some interesting problems for level of detail. Simplification hierarchies can be constructed to optimize for quality in the presence of changing pose and shape. We intend to explore this facet as it is particularly important for handling large numbers of characters such as in a crowd scene.

References

[1] Arad, N., Dyn, N., Reisfeld, D., and Yeshurun, Y. Image warping by radial basis functions: Applications to facial expressions. *Computer Vision, Graphics, and Image Processing 56*, 2 (Mar. 1994), 161–172.

[2] Cohen-Or, D., Solomovici, A., and Levin, D. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics 17*, 2 (April 1998), 116–141. ISSN 0730-0301.

[3] Kent, J. R., Carlson, W. E., and Parent, R. E. Shape transformation for polyhedral objects. *Computer Graphics 26*, 2 (July 1992), 47–54. Proceedings of SIGGRAPH 1992.

[4] Lee, A. W. F., Dobkin, D., Sweldens, W., and Schröder, P. Multiresolution mesh morphing. In *Computer Graphics* (Aug. 1999), pp. 343–350. Proceedings of SIGGRAPH 1999.

[5] Lee, S., Wolberg, G., and Shin, S. Y. Polymorph: Morphing among multiple images. *IEEE Computer Graphics and Applications 18*, 1 (Jan. 1998), 60–73.

[6] Lerios, A., Garfinkle, C. D., and Levoy, M. Feature-based volume metamorphosis. In *Computer Graphics* (Aug. 1995), pp. 449–456. Proceedings of SIGGRAPH 1995.

[7] Lewis, J. P., Cordner, M., and Fong, N. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In *Computer Graphics* (July 2000). Proceedings of SIGGRAPH 2000.

[8] Librande, S. E. Example-based character drawing. Master's thesis, MIT, Cambridge, MA, 1992.

[9] Loop, C. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, Salt Lake City, UT, 1987.

[10] Micchelli, C. A. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation 2* (1986).

[11] Ngo, T., Cutrell, D., Dana, J., Donald, B., Loeb, L., and Zhu, S. Accessible animation and customizable graphics via simplicial configuration modeling. In *Computer Graphics* (July 2000), pp. 403–410. Proceedings of SIGGRAPH 2000.

[12] Powell, M. J. D. Radial basis functions for multivariable interpolation: A review. In *Algorithms for Approximation*, J. C. Mason and M. G. Cox, Eds. Oxford University Press, Oxford, UK, 1987, pp. 143–167.

[13] Rademacher, P. View-dependent geometry. *Proceedings of SIGGRAPH 99* (August 1999), 439–446. ISBN 0-20148-560-5. Held in Los Angeles, California.

[14] Rose, C. F., Cohen, M. F., and Bodenheimer, B. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications 18*, 5 (Sept. 1998), 32–40.

[15] Rossignac, J., and Kaul, A. Agrels and bips: Metamorphosis as a bezier curve in the space of polyhedra. In *Computer Graphics Forum: The International Journal of the Eurographics Association* (Sept. 1994), pp. 179–184. Proceedings of Eurographics 1994.

[16] Ruprecht, R., and Müller, H. Image warping with scattered data interpolation. *IEEE Computer Graphics And Applications 15*, 2 (Mar. 1995), 37–43.

[17] Turk, G., and O'Brien, J. F. Shape transformation using variational implicit functions. In *Computer Graphics* (Aug. 1999), pp. 335–342. Proceedings of SIGGRAPH 1999.