Shape and Animation by Example

Peter-Pike Sloan Charles F. Rose, III Michael F. Cohen

July 2000

Technical Report
MSR-TR-2000-79

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
`http://www.research.microsoft.com`

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 and animations from existing geometric forms and motions. Our approach differs from previous work by allowing for extrapolation and interpolation between multiple forms as well as motions. In addition, our methodology is efficient enough to be used in an interactive runtime setting. The paradigm presented here is one of *design by example*. New shapes and animations are created on the fly through multi-way blending of examples.

At this time, a variety of 3D scanning methodologies are available that can capture shapes that exist in the real world. Motion capture technologies are also capable of recording complex performances. One drawback of these capture systems is the cost that is incurred in their purchase and operation. Another limitation of capture devices is that they are restricted to recording shapes and motions that can be observed (e.g., no dinosaurs). In contrast to the scanning devices, highly skilled artists have the advantage of being able to model complex existing as well as imaginary shapes and animations.

Both of these means of creating shapes and animations are limited in the same way. In particular, neither has a simple means of automatically modifying the shapes and animations once they have been created. Automatically modifying shapes and animations is desirable for two reasons. One, in scripted settings, such as films, automatic modification makes it easier to avoid redundancy. For instance, an entire colony of different looking ants could be automatically created from a few distinct ants. In addition, the ants' motion could automatically adapt to changes in their mood or to the terrain they are walking on. Two, in non-scripted interactive runtime settings, such as games, it is not possible to anticipate all shapes and animations that will be needed when the game is played. Thus, most games simply try to reuse the closest fitting model or motion for the situation.

There are at least two solutions to the issues outlined above. One solution consists of creating more shapes and animations. Unfortunately, this approach would again be costly. Another solution is to try to leverage existing forms and motions 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 runtime interpolation between multiple forms or multiple motion segments. The forms and motions may have been created by artists or through geometry or motion capture technologies. Once our system is provided with *example* forms and motions, it can generate a continuous range of forms we call a *shape* or a continuous range of motions we call a *verb*. We also apply the shape blending methodology to articulated figures to create smoothly skinned figures that deform in natural ways. The runtime interpolation of the forms or motions runs fast enough to be used in interactive applications such as games.

2 Related work

The idea of leveraging existing shapes and animations by modifying them is certainly not new. We begin with a discussion of the relevant morphing literature. Morphing methods, initially applied to images [21] [2], have recently also been applied to 3D geometry [22, 11, 6, 14, 10]. Much of the morphing methodology concentrates either 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[13]. We assume that the correspondence problem has either been solved implicitly in the creation of the original forms or that existing methods can be used to establish correspondences. We focus, instead, on the problem of efficient blending between multiple *examples*.

Lee et al. [11] primarily focus on the correspondence problem and two-way blending. Their approach could easily be extended to incorporate our blending functions and morphs between multiple *examples*.

Turk and O'Brien [22] focus on shape transformation. While their method is very flexible and can deal with N-way blends between shapes of arbitrary genus, it is not interactive since it represents shapes as implicit functions in moderately high dimensional spaces.

Rademacher's study [18] 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, to do the blending. Our work also allows us to leverage the underlying anatomical structure in skeleton based figures.

Blanz and Vetter [3] present a parametric model for a human face. While they focus on representing rather than animating a face, they do discuss expressions in their section on future work. They project their *examples* into a principal components basis to reduce the necessary number of *examples* and have very impressive results.

The second area of research related to our work is animation. Parameterized motion is typically generated in one of three ways: procedurally, dynamically, or by interpolation. Recently, some efforts have focused on modifying control functions for dynamically-based animation to create new animations [9, 16]. While this approach appears promising, to date, the control functions have proven quite fragile when altered.

Our methodology falls into the last category of animation, that of interpolation. A considerable amount of work has been undertaken with respect to editing existing animations and blending between two segments [20, 7, 12].

Methods that have addressed modifying existing animation include Fourier techniques as used by [23] to modify periodic motions. Amaya et al. used infor-

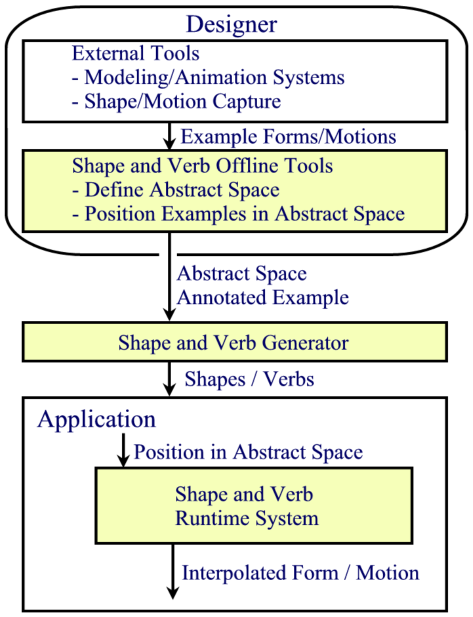

Figure 1: System overview

mation from one motion to modify the emotional content of another, primarily through changes in timing and intensity of motion [1]. Both of these techniques rely on periodic examples.

Bruderlin and Williams [5] use multitarget interpolation with dynamic time warping to blend between motions. Linear interpolation over a set of *example* motions is used by Wiley and Hahn [24] and Guo and Robergé [8]. Both of these techniques use $O(2^d)$ *examples* and fail to address the usefulness of *examples* with a limited region of influence.

As in the case of shape, we will concentrate on efficient multi-way blending for animation. Rose *et al.*'s paper "Verbs and Adverbs" [19] has shown results in this area using radial basis functions. The methods presented here extend the ones described in this earlier article by showing that the authors did not recognize a key mathematical feature of their system. We will discuss and demonstrate a reformulation of their work that leads to a much more efficient runtime system. This reformulation also allows us to apply the same ideas to shape blending.

3 Shapes and Adjectives, Verbs and Adverbs

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

A designer relies on external modeling and animation systems and/or geometry and motion capture technology to create initial forms and motions. We refer to these initial forms and motions as *example* forms and *example* motions, or *examples*, for short.

The Shape and Verb Offline Tools enable the designer to organize these *example* forms or motions that serve as input into our Shape and Verb Generator. To do so, the designer must choose a set of adjectives that characterize the forms or a set of adverbs that characterize the motions. The adjectives or adverbs define an abstract space. Each adjective or adverb represents a separate axis in the abstract space.

For instance, adjectives that describe the form of a human arm may include gender, age, and elbow bend. 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. 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 changes in muscles and skin. Adverbs for a walk may include the walker's mood and aspects of the environment in which the walk takes place. A happy walk is quite different from a sad or angry walk. Walks also differ with the slope or the surface walked on.

Once the designer has defined the abstract space, each *example* form or motion is annotated with its location in the abstract space. In addition, motion *examples* are tagged with *keytimes*, such as the moment when each foot touches the ground. The *keytimes* provide the means to perform automatic *time warping* at runtime. The details of the time warping can be found in Rose *et al.* [19].

Based on the annotated *examples* and the abstract space, The Shape and Verb Generator solves for the coefficients of a smoothly-varying interpolation of the forms and motions across the abstract space. These coefficients provide the means to interpolate between the *example* forms and motions at runtime. We refer to the output produced by the Shape and Verb Generator as a *shape* when interpolating forms and as a *verb* when interpolating motions.

At runtime, the application chooses at each moment in time desired values for the adjectives or adverbs, thus defining a specific location in the abstract space. For instance, a character can be set to be happy or sad or anywhere in between; an arm can be specified to be more male or female, or to respond to the bending of the elbow. The Shape and Verb Runtime System then responds to the selected location in the abstract space by efficiently blending the annotated *examples* to produce an interpolated form or motion.

The number of adverbs or 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 or verb by N. Each motion in a verb as well as each 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. Each motion

Object	Variable	Subscript	Range
Example	X	i	$1..N$
DOF	x	i	$1..N$
		j	$1..M$
Point in Adverb 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

in a verb must include the same number of curves that define the trajectory of the character's joints, and these curves must all have the same number of control points defining them. All *example* motions in a particular verb must also represent the same action. A set of *example* walks, for instance, must all start out on the same foot, take the same number of steps, and have the same arm swing phase. Thus, all *examples* (forms or motions) must have the same number of degrees of freedom (DOF). The number of DOF, denoted by M, equals three times the number of vertices in a form (for the x,y,z coordinates). In the case of a motion, the number of DOF is the number of joint trajectories times the number of control points per curve.

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

$$\begin{aligned} X_i = \{x_{ij}, \mathbf{p}_i, K_m : i = 1 \ldots N, \\ j = 1 \ldots M, m = 0 \ldots \mathit{NumKeyTimes}\} \end{aligned} \tag{1}$$

Each x_{ij}, the j^{th} DOF for the i^{th} *example* represents a coordinate of a vertex or, in the case of a motion, a uniform cubic B-spline curve control point. $\mathbf{p}_i$ is the location in the abstract space assigned to the *example.* K is the set of keytimes which describe the phrasing (relative timing of the structural elements) of the *example* in the case of a motion. Based on the keytime annotations for each motion *example*, the curves are all time-warped into a common generic time frame. Keytimes are, in essence, additional DOFs and are interpolated at runtime to undo the time-warp. See Rose *et al.* for details on the use of keytimes [19].

3.1 Shape and Verb Generation

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

In Rose *et al.* [19], each B-spline coefficient was treated as a separate interpolation problem (1200 separate problems in their walk verb) which leads to inefficiencies. We develop, instead, a *cardinal basis* where we associate one basis function with each *example.* 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 simply scaled by the DOF values and then summed. We show that this approach is not only equivalent to the formulation in Rose *et al.*, but also that it is (a) more efficient in the case of animations and (b) can be applied to shape interpolation.

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 cardinal basis we use has 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 function provides 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,

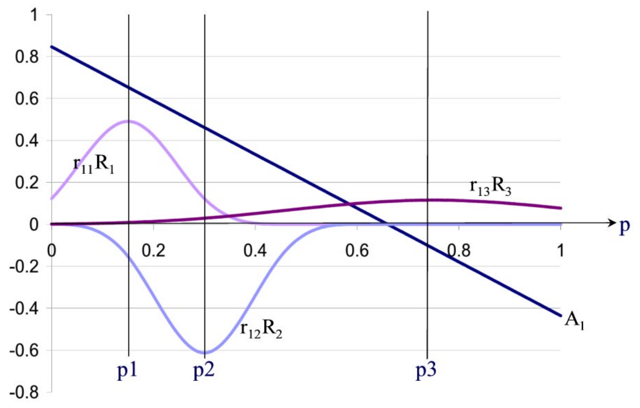

Figure 2: Linear and radial parts of the cardinal basis function for the first *example*.

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.

Figure 2 shows the linear approximation and the three radial basis functions associated with the first of three *examples* for a simple one- dimensional abstract space. 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 hyperplane for any particular DOF could then be evaluated by simply scaling the bases by the DOF values and summing.

3.3 Radial Basis

Radial basis interpolation is discussed in Micchelli [15] and in the survey article by Powell [17]. Radial basis functions have been used in computer graphics for image warping by Ruprecht and Müller [21], Arad *et al.* [2] and for 3D interpolation by Turk *et al.* [22].

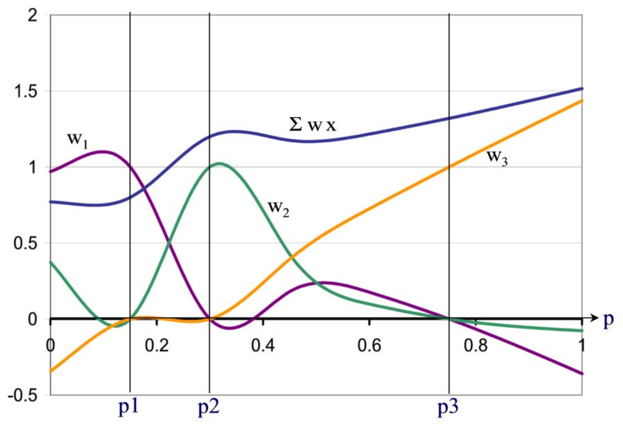

Figure 3: Cardinal basis functions and scaled sum for particular degree of freedom.

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 as in Rose *et al.*. 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

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

To account 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, $r_{i_2 i_1}$, to account for the residuals such that, when the weighted radial bases are summed with the hyperplanes, they complete the cardinal bases.

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 [22] selected the basis

$$R(d) = d^2 log(|d|)$$

as this generates 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.

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 and $\mathbf{Q}$ 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. The diagonal terms 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*.

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.

Given the solutions for the radial basis weights, we now have all the values needed to evaluate equations 3 and 2 at runtime. The upper (blue) line in Figure 3 is simply the three cardinal bases scaled by the *example* values and summed as in equation 3.

3.4 Shape and Verb Runtime System

At runtime, the application selects the point of interest in the abstract space. This point may move continuously from moment to moment, if for instance, the character's mood changes or the character begins to walk up or downhill. The Shape and Verb Runtime System takes this point and generates an interpolated form or motion and delivers it to the application for display.

3.5 A note on Complexity

The runtime complexity of the formulation in equations 3 and 2 is

$$MN + N(N + S) = MN + N^2 + NS \tag{4}$$

where M is the number of DOF variables, N is the number of *examples*, and S is the dimension plus one of the abstract space.

In Rose *et al.* there was a separate linear hyperplane and radial basis per DOF (approximately 1200 B-spline coefficients in their walking verb). The complexity in their formulation is given by:

$$M(N + S) = MN + MS \tag{5}$$

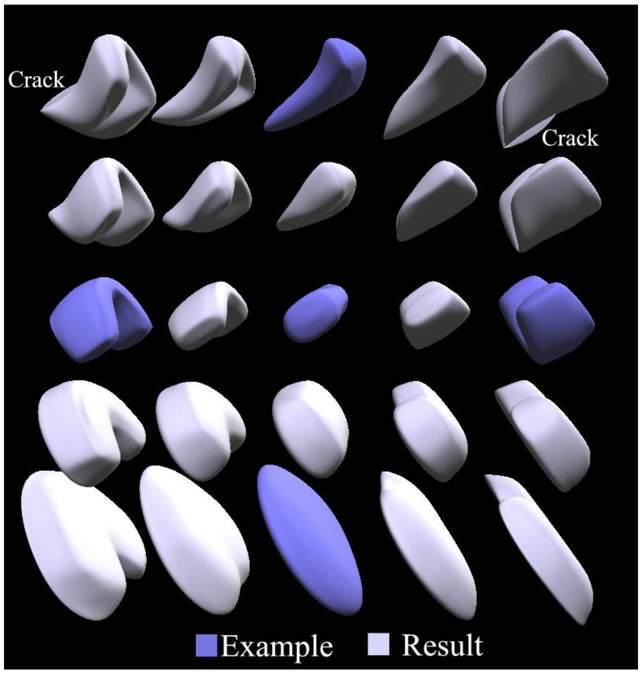

Figure 4: Exploration of the space can reveal problems with the interpolation.

. In our formulation above there is only one set of linear bases and one set of radial bases per *example*. The efficiency is gained due to the fact that there are many fewer *examples* than DOF variables. For instance, if M=1200, N=10, and S=6, then the comparison is $MN + N^2 + NS = 12,160$ vs. $MN + MS + N^2 = 19,200$. Due to some additional constants the efficiencies are actually a bit better. We have implemented both Rose *et al.*'s formulation and the one presented here. We found an approximate speedup factor of about 2 in most cases.

4 Reparameterization and Modification of Shapes and Verbs

Once a shape or verb has been generated, the abstract space can be quickly explored to see the interpolated forms and motions. 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. Blue forms indicate *examples* and grey forms are the result of the Shape and Verb Runtime System. 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

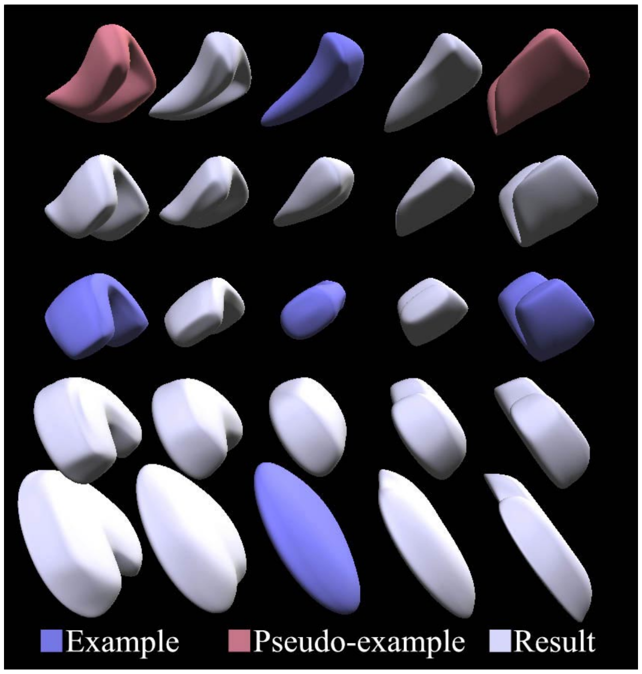

Figure 5: *Pseudo-examples* can reparameterize the space, fixing up problem regions.

desired in one region and more slowly than desired in a neighboring region. For instance, at the point in the abstract space that is half way between a happy walk and a sad walk, the character may appear more sad than happy, rather than neutral.

Two methods are available to modify the results produced by the Shape and Verb Generator. In one method a new *example* can be inserted at the problem location and a new verb or 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 or motion details and reinsert this into the abstract space as a new *example*.

In the second method, the designer first selects an acceptable interpolated form or motion from a location in the abstract space near the problem region. Next, he/she moves this form to a location in the problem region. This relocated form will be treated exactly the same as an *example*. We call this relocated form a *pseudo-example*.

Going back to the problem with the walk verb described above, a neutral walk cycle found elsewhere in the abstract space may be moved to the half way point between happy and sad walks.

The place from which the *pseudo-example* is taken will be point $\mathbf{p}$. We denote the new position to which it is moved to as $\tilde{\mathbf{p}}$. 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 in red. Arrows indicate the locations from which the *pseudo-examples* were drawn.

Creating the reparameterized verb or shape proceeds much as before. We need to introduce a few new notations 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 or motion at any point in the abstract space is:

$$x_j(\mathbf{p}) = \sum_{i_1=1}^{N} \tilde{w}_{i_1}(\mathbf{p}) x_{i_1 j} \tag{6}$$

Note that the interpolated form or motion 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 $\tilde{w}$.

$$\tilde{w}_{i_1}(\mathbf{p}) = \sum_{i_2=1}^{\tilde{N}} \tilde{r}_{i_2 i_1} R_{i_2}(\mathbf{p}) + \sum_{l=0}^{D} \tilde{a}_{i_1 l} A_l(\mathbf{p}) \tag{7}$$

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 7). 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*.

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 as 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. This is supported in current graphics hardware, making it an attractive technique. This simple method exhibits some problems. We use an arm bending at the elbow to discuss the problems associated with transform blending and 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, for the elbow bending, 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 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.

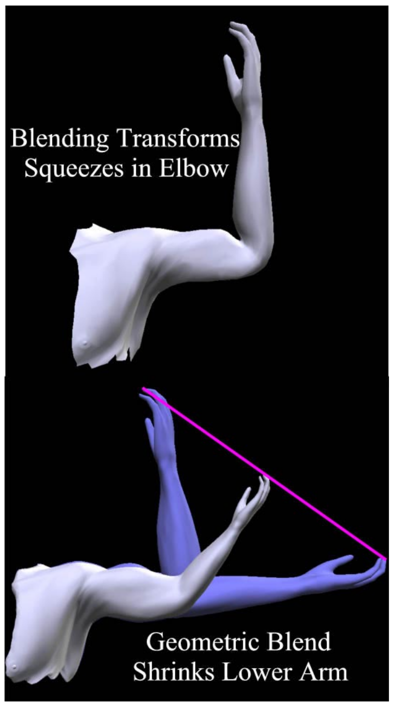

Figure 6: Simple transformation blending exhibits shrinking about joints.

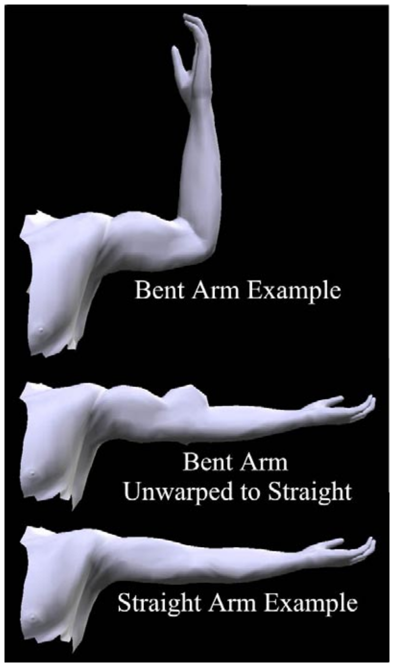

Figure 7: *Example* forms are warped into a cononical pose.

Instead, what we would like is a combination of transform blending and shape blending. 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 it is 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)

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

We can now first 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

$$x_\beta = (\alpha T_1^{\beta} + (1-\alpha) T_0)(\beta x_0^1 + (1-\beta) x_0) \tag{9}$$

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 $\beta = 1$.

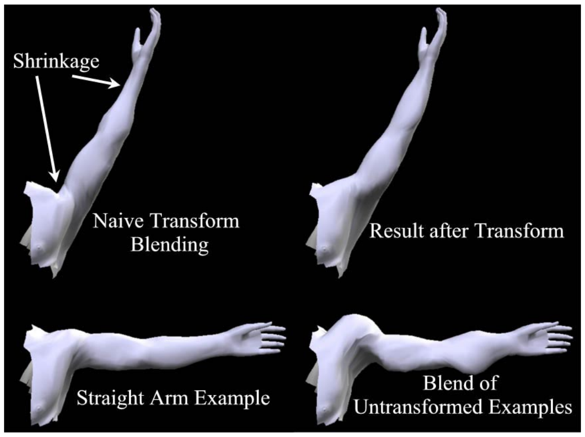

Figure 8: Naive transform blending vs. interpolated blending.

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 8. 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 9. 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 a number of shapes and linked-figure animations. We will discuss two of each. All performace 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 and 1280 faces. Blending in this space can be done at 5500 frames per second (fps).

One step of Loop subdivision is applied to this mesh and the limit positions and normals are computed. This results in a mesh with 2562 vertices and 5120

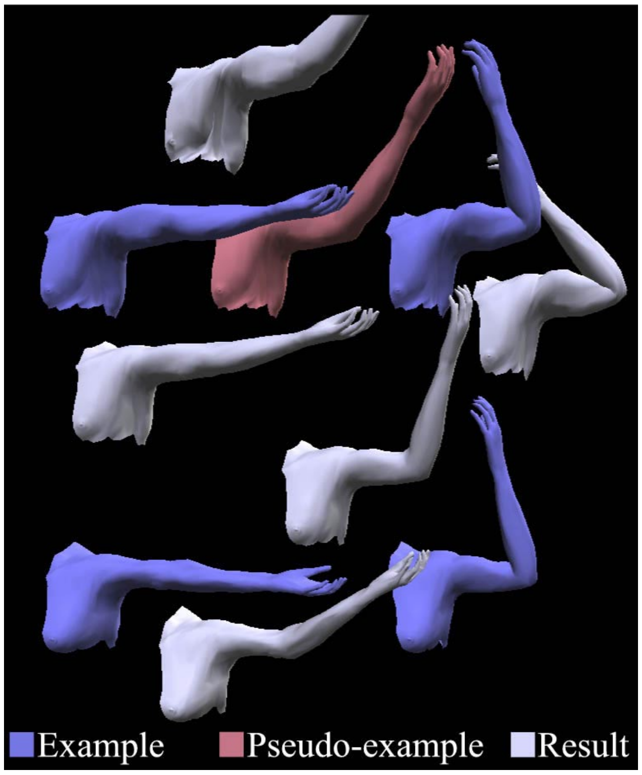

Figure 9: Blue arms are *examples*, red *pseudo-exmaples*, and gray generated by the system.

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. If real *examples* had been added instead, the blending slows down to around 4000 fps.

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 and 2608 faces each. This dataset can be

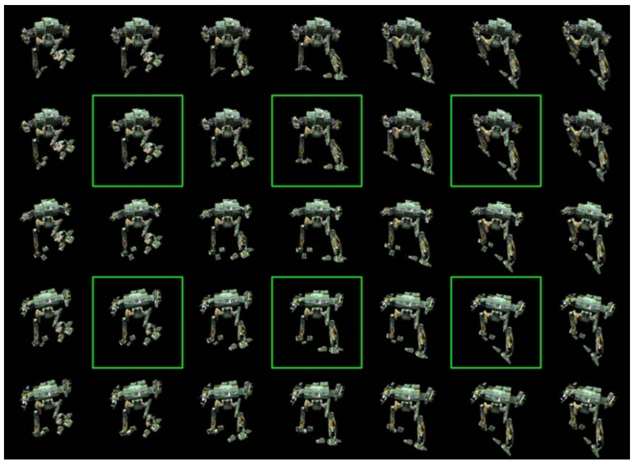

Figure 10: Health increases as y increases. Slope of surface ranges from sloping to left to right.

interpolated at 2074 fps.

Figure 9 is a visualization of a 2D space of arms with 4 real *examples*, 1 *pseudo-example*. Elbow bend is parameterized along the horizontal axis and gender along the vertical. Extrapolation can produce some interesting results.

6.3 Animation

We constructed parameterized motions, verbs, for two different characters: a human figure named Stan and a robot warrior. While both bipedal, they move in very different ways. Stan moves in a humanlike, though often exaggerated, manner, while the robot warrior moves very mechanically. Our technique had no trouble encompassing both forms. Our motion *examples* were chosen to encompass emotional states such as mopiness, cockiness, fear, anger, and for physical parameters such as damage and surface slope.

The verbs shown here were constructed from a mix of motion capture and hand animated source. The motion capture data was gathered optically. The hand-animated source was generated on two popular software animation packages: *3D-Studio/Max* for the robot and *Maya* for some of Stan's data. This data is applied to a skeleton positioned with a translation at the root and a hierarchy of Euler angle joints internally. A technique similar to Bodenheimer, *et. al.*[4] ensured that the angles remained well- behaved.

Our robot warrior has 60 degrees of freedom. Our robot data exhibits two primary kinds of variation: slope of walking surface and level of damage to the robot. Damage results in limping and slope causes the character to rotate the joints in the leg, foot, and waist, and shift weight to accommodate. Figure 10 shows a sampling of the robot walking verb. Green boxed figures are *examples*. The others are interpolations and extrapolations. The verticle axis shows robot

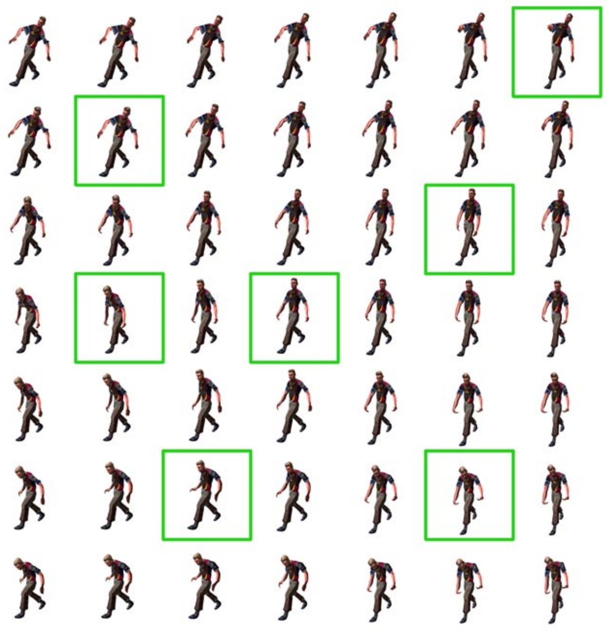

Figure 11: Knowledge increases from left to right. Happiness increases from bottom to top.

health and the horizontal axis surface slope. Figure 12 shows the robots in the center column of Figure 10 rotated to the side to better show the character limping.

We constructed a number of robot verbs: powered-down idle, power-up, powered-up idle, power-down, idle-to-walk, walk, and walk-to-idle. These verbs, together with a transitioning mechanism can be combined to form a verb graph such as discussed in Rose, et. al. [19]. The robot verbs evaluate at 9700 fps in the absence of rendering. Thus, a robot evaluated using verbs can be run at 30 fps with only a 0.3% percent processor budget.

We also constructed a walking verb from a human figure, Stan, with 138 degrees of freedom. We chose 3 emotional axes for this verb which seemed to encompass the gamut of variation present in the data. These were happy-to-sad, knowledgeable-to-clueless, and low-to-high-energy. Our example motions exhibit these subjective characteristics: easygoing, mopey, cocky, goofy, angry, afraid, and neutral. Figure 11 shows a sampling of our walk along the knowledge and happiness axes. Unlike the robot data, which was designed to fall at regular places in the adverb plane, our Stan motions are less regular. As there are no restrictions upon example placement, we are able to construct verbs with irregular example placement, as shown by the green-boxed figures. Evaluating a pose for Stan is extremely efficient: 3900 fps or a 0.7% processor budget at 30 fps.

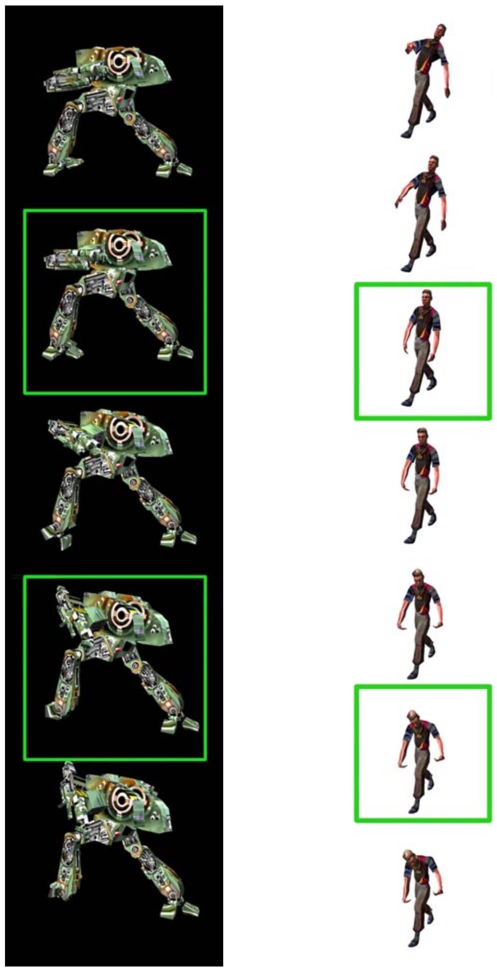

Figure 12: Closer views of our robot and Stan

7 Conclusions

Shape and motion interpolation has been shown to be an effective and highly efficient way of altering shape and motion 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 able to 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 patters 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] Amaya, K., Bruderlin, A., and Calvert, T. Emotion from motion. In *Graphics Interface* (May 1996), pp. 222–229. Proceedings of Graphics Interface 1996.

[2] 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.

[3] Blanz, V., and Vetter, T. A morphable model for the synthesis of 3d faces. In *Computer Graphics* (Aug. 1999), pp. 187–194. Proceedings of SIGGRAPH 1999.

[4] Bodenheimer, B., Rose, C. F., Rosenthal, S., and Pella, J. The process of motion capture: Dealing with the data. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation* (Sept. 1997), pp. 3–18.

[5] Bruderlin, A., and Williams, L. Motion signal processing. In *Computer Graphics* (Aug. 1995), pp. 97–104. Proceedings of SIGGRAPH 1995.

[6] 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.

[7] Gleicher, M. Retargetting motion to new characters. In *Computer Graphics* (July 1998), pp. 33–42. Proceedings of SIGGRAPH 1998.

[8] Guo, S., and Robergé, J. A high-level control mechanism for human locomotion based on parametric frame space interpolation. In *Proceedings of the 6th EuroGraphics Workshop on Animation and Simulation* (Aug. 1996), pp. 95–107.

[9] Hodgins, J. K., and Pollard, N. S. Adapting simulated behaviors for new characters. In *Computer Graphics* (Aug. 1997), pp. 153–162. Proceedings of SIGGRAPH 1997.

[10] 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.

[11] 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.

[12] Lee, J., and Shin, S. Y. A hierarchical approach to interactive motion editing for human-like figures. In *Computer Graphics* (Aug. 1999), pp. 39–48. Proceedings of SIGGRAPH 1999.

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

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

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

[16] Popović, Z., and Witkin, A. Physically-based motion transformation. In *Computer Graphics* (Aug. 1999), pp. 11–20. Proceedings of SIGGRAPH 1999.

[17] 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.

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

[19] 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.

[20] Rose, C. F., Guenter, B., and Bodenheimer, B. Efficient generation of motion transitions using spacetime constraints. In *Computer Graphics* (Aug. 1996), pp. 147–154. Proceedings of SIGGRAPH 1996.

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

[22] 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.

[23] Unuma, M., Anjyo, K., and Tekeuchi, R. Fourier principles for emotion-based human figure animation. In *Computer Graphics* (Aug. 1995), pp. 91–96. Proceedings of SIGGRAPH 1995.

[24] Wiley, D. J., and Hahn, J. K. Interpolation synthesis for articulated figure motion. In *Proceedings of Virtual Reality Annual International Symposium* (Mar. 1997), pp. 157–160.