Author preprint. Published version DOI: 10.1002/cav.1608

Dynamic Sprites: Artistic Authoring of Interactive Animations

Ben Jones
University of Utah

Jovan Popovic
Adobe Systems, Inc.

James McCann
Adobe Systems, Inc.

Wilmot Li
Adobe Systems, Inc.

Adam Bargteil
University of Utah

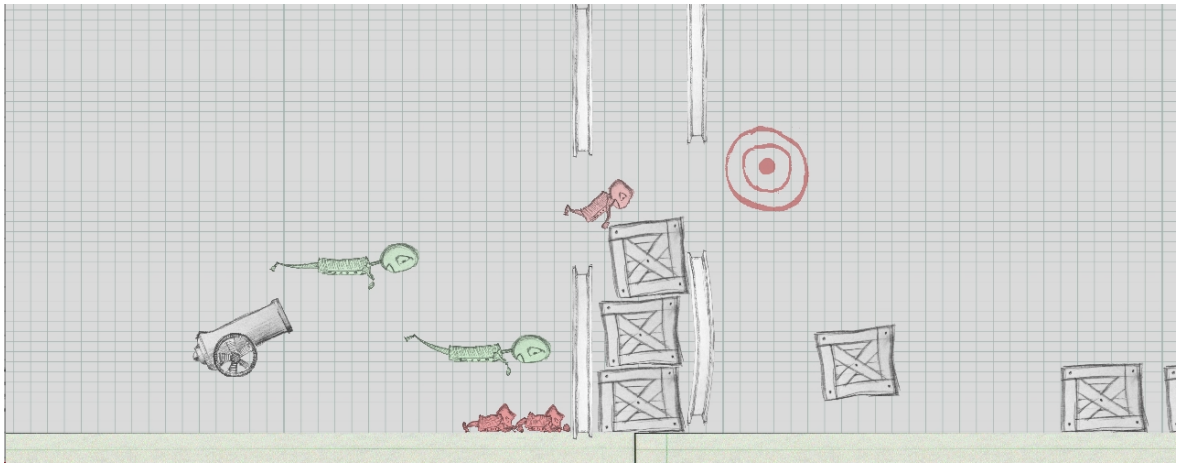

Figure 1: *A set of fearless stuntmen are fired out of a cannon. The dynamic sprites interact to create a rich, stylized animation.*

Abstract

Traditional methods for creating dynamic objects and characters from static drawings involve careful tweaking of animation curves and/or simulation parameters. Sprite sheets offer a more drawing-centric solution, but they do not encode timing information or the logic that determines how objects should transition between poses and cannot generalize outside the given drawings. We present an approach for creating dynamic sprites that leverages sprite sheets while addressing these limitations. In our system, artists create a drawing, deform it to specify a small number of example poses, and indicate which poses can be interpolated. To make the object move, we design a procedural simulation to navigate the pose manifold in response to external or user-controlled forces. Powerful artistic control is achieved by allowing the artist to specify both the pose manifold *and* how it is navigated, while physics is leveraged to provide timing and generality. We used our method to create sprites with a range of different dynamic properties.

CR Categories: I.3.4 [Computer Graphics]: Graphics Utilities—Graphics Editors

Keywords: physics-based animation

Introduction

Drawing pictures is one of the oldest and most fundamental forms of human communication. Ever since our ancestors began creating cave paintings in prehistoric times, humans have been making pictures to tell stories, explain ideas and express emotions. Over the years, technical advances have provided artists with an ever expanding arsenal of tools and techniques for creating and editing images, and today, sophisticated software packages like Adobe Photoshop and GIMP include a variety of features that enable users to copy, paste, compose, morph and otherwise manipulate their images. Unfortunately, while existing digital tools make it easier than ever to create high quality *static* images, making drawings come "alive" through movement and interactive behaviors in cartoons, videos, or games is a very different and challenging task.

Turning a drawing into a dynamic, interactive entity typically involves several steps that require different types of expertise: a rigger defines articulation variables; an animator creates keyframes that specify how those variables change over time; and a programmer encodes the keyframed motions into behaviors. In some cases, simulation can provide a more automated alternative for generating motions and behaviors, but it often requires significant amounts of tuning to produce results that exhibit specific, desired characteristics. For professional film and

game production, these steps can be distributed to separate teams of artists, riggers, animators and programmers, which makes for a highly modular content creation pipeline. However, for more casual users, the gap in required skills and expertise between creating a drawing and making it come to life represents a significant barrier. This is one likely reason why it is much easier to find examples of high quality static drawings (e.g., in online image repositories) than compelling dynamic content.

Figure 2: *Traditional sprite sheets capture all the poses a character or object can assume in a game. (Example from "Age of Umpires"; http://hockey.spacebar.org/. Copyright Tom Murphy VII, used with permission.)*

Sprite sheets, collections of static 2D drawings that depict representative poses for an object (Figure 2), offer a more cohesive, drawing-centric approach to creating dynamic images. Producing motion with a sprite sheet involves cycling through drawings of the object. Thus, artists can change both the appearance and dynamic behavior of an object using traditional drawing tools by creating or editing various poses. Unfortunately, sprite sheets require many drawings to produce smooth animations since even small deformations to a pose require an entirely new drawing. Furthermore, even if these in-between poses can be generated automatically, sprite sheets alone encode neither the timing information that is critical for producing high quality motions, nor the logic that defines how an object should behave in response to external stimuli. As a result, sprite sheets still require a significant amount of work to create and use. In contrast, we use physics to automatically provide timing and achieve generalization outside the hand-drawn examples.

In this work, we present an approach for transforming static drawings into dynamic sprites. To produce a dynamic sprite, the artist creates a pose manifold by drawing an object, deforming it into example poses, and specifying sets of these example poses that can be interpolated as the object moves. Our system uses example-based physical simulation to automatically move the object through this pose manifold based on forces applied from the external environment or user commands. In this way, we allow the artist to create dynamic objects and characters without specifying many in-between frames, adjusting animation curves, or writing code that defines object behavior.

The key feature of our approach is that it provides a set of explicit artistic controls over various characteristics of dynamic sprites. First, the example poses themselves give artists significant control over the appearance of the pose manifold.

However, combining several example poses at once can lead to a "muddy" manifold where interesting features of the individual examples get lost in the blended pose. Thus, we allow the artist to explicitly construct a simplicial complex (mostly line segments with the occasional triangle) over the set of example poses. Furthermore, since external physical forces alone may not induce sufficient motion within the pose manifold, our method provides several additional knobs for controlling the manner in which objects transition between the poses. For example, artists can tell the system to favor particular poses for specific object states, such as a stretched out pose when an object has high velocity, or a neutral pose that represents the object at equilibrium. The artist can also adjust how much energy an object retains after interactions (e.g., the bounciness of a ball sprite). Modifying these parameters allows artists to control the look and feel of motions and behaviors in a more direct, intuitive manner than fine-tuning physical properties like the elasticity or density of deformable objects.

For more complicated behaviors, arbitrary controllers can be used to traverse the pose manifold. Compared to controllers used in traditional physics-based character animation, our controllers are simpler because the sprites themselves can maintain important features of the motion, such as balance, and otherwise "bend" the laws of physics when desired.

We used our approach to generate a variety of sprites that exhibit different types of dynamic behavior, including a cartoony ball, bouncy characters, lively construction materials, and articulated ragdolls, either passively animated or controlled by a finite state machine. We also created three games that combine multiple sprites into interactive environments. Our results demonstrate how our approach facilitates the creation of dynamic objects from static drawings and gives artists intuitive and powerful control over not only the pose manifold, but also how the pose manifold is navigated.

Related Work

Our dynamic sprites apply and extend ideas from the fields of shape interpolation, physics based animation, and character animation

Shape Interpolation Beginning with the pioneering work of Beier and Neely,[1] object interpolation has delivered techniques for natural shape interpolation,[2] and the development of interactive systems that enable posing, bending, and stretching of images and drawings.[3] While such techniques enable interactive performance-based animation of a single drawing, our work investigates a method that animates a collection of drawings automatically.

Our approach addresses the need to produce non-realistic animation of expressive and even exaggerated shapes.

Ngo and colleagues propose manual construction of a simplicial complex that can support animation of arbitrary drawing collections.[4] Bregler and colleagues use a similar geometric structure to retarget motions from one cartoon to another.[5] Our approach extends these ideas with a simulation method that navigates configuration space with proper timing and according to the interaction with the environment.

This approach could also be combined with techniques for camera animation.[6,7] In particular, the proposal for a 2.5D cartoon model[8] echoes our desire to preserve stylization of hand-drawn vector art while expanding its use with freer viewpoint manipulation. Our work complements these efforts by addressing the animation due to the internal motion of the object instead of the external camera motion.

Physics Based Animation Simulation has become an indispensable tool for the computer animation of natural phenomena. In the context of elastically deformable bodies,[9,10] two general approaches have become popular both in the literature and in practice: the finite element method[11,12] (and its co-rotational[13,14] and invertible variants[15]) and position-based dynamics/shape matching[16–20] (and the related strain limiting approaches[21,22]).

Finite element methods have a rich history in applied mathematics and engineering with the nice property that finer discretizations converge to a solution of some equations of motion. In contrast, shape matching replaces second-order elastic forces with positional constraints, which are satisfied by iteratively filtering particle positions, to deliver plausible, fast, and stable approximations. These features make shape matching extremely popular in real-time and interactive contexts such as video games. Moreover, shape matching methods are more amenable to artistic control—a variety of artistic goals can be expressed in terms of the filters that shape matching repeatedly applies, whereas applying such filters in a finite element method would lead to complex non-linear effects and likely instability. Because interactivity and artistic control are our primary goals, the shape matching framework is a better choice for our task.

Unlike keyframing, simulation easily coordinates many degrees-of-freedom, produces natural timing and physically correct motion, and generalizes to a wide variety of contexts and environments. Unfortunately, it is less capable of stylized cartoon animation,[23] and it is more difficult to direct.[24–26] Faloutsos and colleagues[27] proposed simulating directly in an artist-defined deformation subspace given by deformation modes on a lattice. Recent works have extended this idea, using arbitrary mesh deformations to create example-based simulation: a more automated solution to stylized animation.[28–30] We explore a similar framework, expand it to facilitate navigation on the pose manifold, and implement it with a new formulation.

Character Animation Interaction with other animated characters is a critical requirement for creating immersive virtual worlds. Existing approaches for animating characters can be roughly grouped into two categories: kinematic or dynamic. Kinematic methods ignore physics and animate the character's pose directly, typically using existing motion data (eg. motion capture or artist animated motions). In order to react to user commands or other stimuli at runtime, these methods replay[31–34] or interpolate[35] between motions in a database. By leveraging existing motions, timing is handled implicitly and these methods excel at generating very natural motion. However, these approaches have difficulty generalizing to new interactions or environments.[36]

On the other hand, dynamically simulated characters can react to arbitrary stimuli, but require complex control strategies to achieve natural motion. Motion control has long been studied in robotics. However, in robotics the question of naturalness is often ignored. Almost 20 years ago, Hodgins and colleagues[37] used control startegies from robotics, including proportional-derivative controllers and finite state machines to animate physically simulated human athletes. More recently, more complex control strategies have been introduced to improve the naturalness of the motion. For example, by combing pose tracking with balance feedback controls, stable characters who respond naturally to stimuli can be created.[38–40] However, most characters are naturally unstable and under-actuated, requiring careful control of contact forces even to maintain balance. Performing agile actions, and remaining upright in the face of large disturbances remains challenging. Authoring such controllers is also difficult, requiring tradeoffs between overly-stiff control, deviation from desired motions, and robustness.

Controlling dynamic sprites leverages the advantages of both approaches by incorporating the positional and trajectory data, defined by paths through the pose manifold, with the dynamic response and interaction of physics based animation.

Although artistic control manipulates the timing just like locomotion control or other control tasks, it requires a different set of tools, controls, and solutions. Two recent works exemplify this dichotomy for simulation of elastic bodies.[41,42] Coros and colleagues[41] use rest-shape adaptation to locomote deformable creatures within FEM simulations. While this approaches uses examples to parameterize the rest shape, it is largely unconcerned about whether any example is recognizable with an animation frame: the essential task is to move the center of mass. In contrast, rig-space physics[42] strives to introduce physics into the animation pipeline. Hence, it follows the specified animation curves precisely while inferring the missing degrees of freedom with reduced-order FEM simulation. The aim of our work is to introduce simulation to static drawings – not animations. As a result, we design an approach that provides controls for manipulating the drawing and the timing model without requiring keyfram-

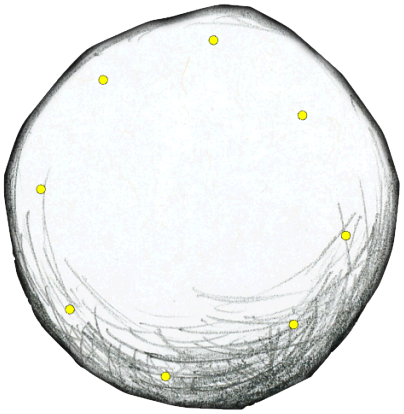

Figure 3: *The rig used to generate poses of the cartoon ball. The yellow dots are control handles.*

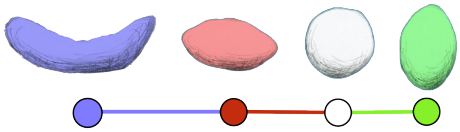

Figure 4: *The example poses and the simplicial complex used to create a stylized bouncing ball.*

ing or other animation skills. This timing model intentionally departs from the FEM framework so that physically invalid but still plausible motions remain as an artistic option.

Method

Our approach proceeds in two phases: an authoring phase, in which an artist designs a low-dimensional deformation rig (see Figure 3), uses this rig to create example poses, and defines a simplicial complex connecting these poses (see Figure 4); and a simulation phase, where our system uses example-based simulation to create a dynamic, interactive animation (see Figure 5). We note that though the details of our example-based simulation differ from previous work, our primary technical contribution is in how we provide explicit artistic control over the pose manifold and how it is navigated.

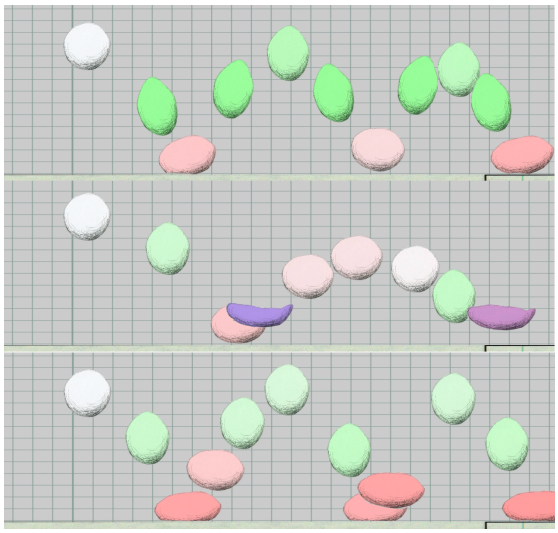

Figure 5: *Three stylized behaviors generated by our system.*

Authoring Phase

As the first step to creating stylized interactive animations, the artist specifies a low degree of freedom rig using the method of Jacobson and colleagues.[43] Specifically, the artist places a small number (5-10) of bones and/or pins on the input shape. By manipulating this simple rig the artist creates a set of example poses that will guide the simulation. This rig also determines how the example poses will be interpolated. The artist additionally defines a simplicial complex over the example shapes that determines which shapes may be blended. Then the artist loads the simplicial complex into our example-based simulation and interactively tunes how the pose manifold is navigated by choosing what optional filters to apply and with what strengths.

Example-based Simulation

Our runtime environment builds on recent work that applies example-based simulation to shape matching/position-based dynamics.[29,30] Deformable objects are modeled as a set of particles, $\mathbf{p}_i \in \mathcal{P}$ with positions, $\mathbf{x}_i$, and velocities, $\mathbf{v}_i$. The neighborhood of a particle $\mathcal{N}(p_i)$ is the set of particles in the k-ring (typically, $k = 3$) of $\mathbf{p}_i$ in a precomputed triangulation of the particle set. At its most basic level, our method applies a series of filters to the particle's positions and velocities to satisfy the goals of the artist. Our approach can also be cast in the prediction/correction framework: we *predict* particle positions with a forward Euler integrator and then *correct* them to achieve various goals; including pulling toward the pose manifold, removing interpenetration, and artistic goals, such as setting the global orientation. See

Algorithm 1 for a summary of the simulation timestep. We now describe these operations in more detail.

Algorithm 1 Simulation Timestep

Apply body forces (e.g. gravity): $v \mathrel{+}= f/m * dt$
Forward-euler position update: $x \mathrel{+}= v * dt$
Compute desired rest shape
Interleave and Iterate
  Local-neighborhood shape match
  Global shape match
  Resolve collisions

Compute velocity
Global momentum adjustment
Correct global orientation

Compute Desired Rest Shape

Our runtime environment takes as input the artist-authored *pose manifold* described by a set of example poses, a low degree of freedom animation rig, a simplicial complex that describes which poses can be blended, and any other rules (such as an equilibrium pose) that guide navigation on the manifold. Then, during each simulation step we must select a pose on this pose manifold that will be used as a rest state for shape matching.

To interpolate poses within a simplex, we first factor each handle transform into translational, scale/shear, and rotational components using a polar decomposition. Translation and scale/shear terms are interpolated linearly, while rotations are combined using spherical linear interpolation. The final pose is generated by applying linear blend skinning to the interpolated transforms. Specifically, to blend two example poses with weights α and β we would have,

$$\begin{aligned}\mathbf{e}_i = \quad & \textstyle\sum_{j=1}^{m} w_j(\mathbf{r}_i)\,\mathrm{Slerp}(\alpha\mathbf{R}_1, \beta\mathbf{R}_2)(\alpha\mathbf{S}_1 + \beta\mathbf{S}_2)\mathbf{r}_i \\ & +\alpha\mathbf{t}_1 + \beta\mathbf{t_2}, \end{aligned} \tag{1}$$

where $\mathbf{e}_i$ is the particle's position in the pose selected from the pose manifold, $\mathbf{r}_i$ is the particle's position in the initial rest configuration where the weights were computed, $w_j(\cdot)$ gives the weight of the j^{th} handle at the specified position, and $\mathbf{R}$, $\mathbf{S}$, and $\mathbf{t}$ are the rotation, scale/sheer, and translations computed by the polar decomposition. Generalization to blending more poses is straightforward.

It remains to describe how we compute the weights (α and β above) we will use to combine the example poses. The straightforward solution is to simply project the current simulated shape onto the pose manifold. When external forces deform an object toward an example pose, this straightforward manifold projection generates animations that smoothly and plausibly transition between the user provided examples. However, projection alone is not adequate for forces orthogonal to the pose manifold, or when other artistic control is desired. Thus, we have included additional control over how the simulation navigates the pose manifold, resulting in richer behavior. The weights are computed by performing the following operations:

1. Project current shape onto pose manifold
2. Adjust the equilibrium pose
3. Attract toward equilibrium pose
4. Apply energy adjustment
5. Apply velocity-based adjustment

Note that the first two elements have been incorporated into previous example-based simulation approaches, while the last two are critical to achieving our results.

Project current shape onto pose manifold The goal of this step is to find interpolation weights in the simplicial complex, such that the skinned shape matches the current shape as closely as possible. To account for rotations we additionally compute a global, best-fit rotation. We alternate between solving for optimal weights with this rotation held fixed, and solving for the optimal rotation with weights held fixed. To compute optimal weights, we define our cost function as

$$\sum_{i,j} ||\mathbf{R}_g\mathbf{e}_{ij} - \mathbf{x}_{ij}||^2 \tag{2}$$

where i and j are connected particles, $\mathbf{R}_g$ is the current global rotation, $\mathbf{x}_{ij}$ is the vector between particles i and j in world space, and $\mathbf{e}_{ij}$ is the vector between i and j computed via linear blend skinning with the current interpolation weights (see Equation (1)). This projection yields barycentric coordinates, $\mathbf{m_p}$, in the simplicial complex that defines the manifold.

As noted by Jacobson and colleagues,[43] positions of nearby particles computed by linear blend skinning are highly correlated, so we adopt their "rotation clusters" optimization. Instead of summing over all particles i and j, we compute a smaller set of representative particles using k-means clustering and sum over them. We use a simple Newton solver to optimize the objective, using automatic differentiation to compute the gradient and Hessian.[44] To compute the optimal global rotation, we use the method of Sorkine and Alexa.[45]

Adjusting the equilibrium pose As described thus far, our example-based framework has no notion of a *rest pose*—all poses in the pose manifold generate zero elastic energy. To address this limitation, we introduce the notion of an *equilibrium pose*, $\mathbf{q}$, in the example manifold. High level planning and control strategies are incorporated into our method by allowing artists explicit control over how this equilibrium pose is chosen. In our prototype, complex behaviors are implemented using a 2-level finite state machine. The high level state machine switches between character behaviors (walking, jumping, falling, etc), and

is controlled by user input and dynamic properties of the simulation, such as whether or not the character is standing on the ground. The lower-level state machine controls individual behaviors by adjusting the location of the equilibrium pose in the manifold, such as transitioning through the poses in a walk cycle.

Attract toward equilibrium pose In the spirit of position based dynamics, we use a first-order spring to attract toward the equilibrium pose:

$$\mathbf{m}_{eq} = \mathbf{m}_e + k_{eq}\left(\mathbf{q} - \mathbf{m}_e\right), \tag{3}$$

where $\mathbf{m}_{eq}$ is the pose after attracting to the equilibrium pose, k_{eq} is a stiffness, $\mathbf{m_e}$ the barycentric coordinates after energy adjustment, and $\mathbf{q}$ the barycentric coordinates of the equilibrium pose. By adjusting the stiffness of the manifold spring, the artist can control how closely the specified trajectory is followed, and how the motion is influenced by other other aspects of the simulation (e.g. energy adjustment).

Apply energy adjustment Often, an artist will want the simulation to closely match the pose manifold, requiring very high material stiffness. In these cases, any energy from deformations orthogonal to the pose manifold is lost. To combat this, we apply some of this lost energy in the pose manifold. Position-based dynamics does not have a explicit notion of energy, however, a good proxy for kinetic energy is

$$e = \sum_{i \in \mathcal{P}} m_i (\Delta \mathbf{x})^2, \tag{4}$$

where m_i is the mass of p_i and $\Delta\mathbf{x}$ is the change in a particle's position. We compute this energy when particles interact with other objects (e.g. during collisions). Our simulator then pushes the interpolation weights by an amount proportional to this energy,

$$\mathbf{m}_e = \mathbf{m}_p + k_e \mathbf{d}_e e, \tag{5}$$

where $\mathbf{m}_e$ is the pose after applying energy adjustment, $\mathbf{m}_p$ is the initial projection onto the manifold, k_e is a stiffness, and $\mathbf{d}_e$ is the direction of energy offset. This direction can be variably chosen as the direction away from the equilibrium ($\mathbf{m}_p - \mathbf{m}_{eq}$), the velocity in the manifold, or an explicitly specified direction. To avoid unwanted oscillations, we ignore energy contributions below a user-defined threshold.

The result is in-manifold deformation for shapes, even when experiencing orthogonal interactions, and a reduction of out-of-manifold deformation. Transferring energy normal to the manifold to a tangent direction may seem unphysical—*it is*. However, in practice we have found that this approach does an excellent job of preserving the artist's intent, expressed through the example shapes, as unphysical as this intent may be.

Apply velocity-based adjustment In our framework it is straightforward to use any information from the simulation state to guide navigation of the manifold. For example, in order to add cartoon-inspired stretch to a fast moving object, we attract interpolation weights to a manifold vertex corresponding to the stretched pose, $\mathbf{a}$, with a strength proportional to the object's speed, s.

$$\mathbf{m}_v = \mathbf{m}_{eq} + k_v s\left(\mathbf{a} - \mathbf{m}_{eq}\right), \tag{6}$$

where $\mathbf{m}_v$ is the pose after applying velocity adjustment, $\mathbf{m}_{eq}$ is the pose after being attracted to the equilibrium, k_v is a stiffness, and $\mathbf{a}$ is the pose being attracted to (e.g. the stretched pose).

Shape Matching

Once we compute the current rest pose of the object using linear blend skinning with appropriate interpolation weights, we are ready to compute dynamics. While any elastic simulation method could be used, we use shape matching[16] for its efficiency and implementation simplicity. Furthermore, because shape-matching models elasticity with simple first-order dynamics, it fits in well with our framework of filtering of positions and velocities, allowing a greater degree of artistic control.

While the initial shape matching work[16] supported only global shape matching, more recently Rivers and James[17] introduced a hierarchical approach. We take a middle ground and interleave global shape matching passes, which quickly correct any out-of-manifold deformation, and passes over local neighborhoods, which allow elastic deformations not described by example poses. For completeness, we briefly describe the shape matching approach. Given corresponding points in world space, $\mathbf{x}_i$, and in our example pose, $\mathbf{e}_i$, we solve for a translations, $\mathbf{t}_x$ and $\mathbf{t}_e$, and rotation, $\mathbf{R}$, that minimize

$$\sum_i m_i \left(\mathbf{R}(\mathbf{e}_i - \mathbf{t}_e) - (\mathbf{x}_i - \mathbf{t}_x)\right)^2. \tag{7}$$

The translations correspond to the center of mass for each set of particles and the rotation is found by a polar decomposition. We can then compute goal positions, $\mathbf{g}_i$

$$\mathbf{g}_i = \mathbf{R}\left(\mathbf{e}_i - \mathbf{t}_e\right) + \mathbf{t}_x. \tag{8}$$

We include all particles in the global shape matches. For the local-neighborhood passes, we iterate over all the particles performing the shape match using only the particles in the local neighborhood, $\mathcal{N}(p_i)$, that is,

$$\sum_{j \in \mathcal{N}(p_i)} m_j \left(\mathbf{R}(\mathbf{e}_j - \mathbf{t}_e) - (\mathbf{x}_j - \mathbf{t}_x)\right)^2. \tag{9}$$

We then compute a particle's goal position by averaging over all neighborhoods that contain it.

$$\mathbf{g}_i = \frac{\sum_{j|i \in \mathcal{N}(p_j)} \mathbf{g}_{ij}}{\sum_{j|i \in \mathcal{N}(p_j)} 1}, \tag{10}$$

where $\mathbf{g_{ij}}$ is p_i's goal position when shape matching using $\mathcal{N}(p_j)$.

Additionally, we allow the artist to decompose an image into disjoint layers. To propagate constraints between layers, we add an additional constraint on a set of "pin" particles which join two layers. At initialization, for each pin, we find the triangle on the connecting layer that it is contained in. Then, we compute the barycentric coordinates of the pin with respect to its containing triangle. To enforce the constraint, we compute the world position of the stored barycentric coordinates with the current triangle vertex positions, and pull the pin particle toward it.

Collisions are handled by projecting overlapping particles out of objects, using the underlying triangle mesh to detect and resolve collisions.

Global Momentum Adjustment

Position-based dynamics has difficulty producing highly elastic, "bouncy" collisions. The inclusion of squashed poses in the pose manifold exacerbates this issue as they appear to be rest poses to the shape matching. However, such collisions are essential to lively, cartoon-style animation. To address this limitation, we add momentum directly to the system after collisions with the ground. Specifically, for each object we store the momentum we wish to add, $\mathbf{p}$, which is updated each timestep with a contribution from each colliding particle.

$$\mathbf{p} \leftarrow \mathbf{p} + m_i k_m \left(\mathbf{x}_{pro} - \mathbf{x}_{pen}\right) \tag{11}$$

where m_i is the particle mass, k_m is a scale, $\mathbf{x}_{pro}$ is the particle's (projected) position after resolving the collision and $\mathbf{x}_{pen}$ is the penetrating particle position. Over time we add this momentum to the system, for each particle,

$$\mathbf{v}_i \leftarrow \mathbf{v}_i + m_i r \mathbf{p} / m_o^2, \tag{12}$$

where $\mathbf{v}_i$ is the particle's velocity, m_i its mass, r the rate at which we add the momentum, and m_o is the total mass of the object. Finally, we update $\mathbf{p}$

$$\mathbf{p} \leftarrow (1 - r)\mathbf{p} \tag{13}$$

This approach has two important features. First, by computing $\mathbf{p}$ per object rather than per particle, we avoid spatial discontinuities. Second, by adding momentum over time, we allow the collisions to occur over a finite time period, allowing the object to deform while it is on the ground, before jumping back into the air.

Orientation Correction

In addition to the shape control provided by example-based shape matching, artists may desire control over other aspects of the object's motion. For example, an artist may want to keep a character upright or aligned with its velocity. Providing such control is straightforward in our framework and requires only applying another filter to the set of particles. For the case of rotation control, we apply a first order spring to the global orientation of the object about its center of mass,

$$\theta_a = \theta_b + k_{oc} \left(\theta_g - \theta_b\right), \tag{14}$$

where θ_b and θ_a are the global orientation before and after orientation control respectively, k_{oc} is a stiffness, and θ_g is the goal orientation. This goal angle can be a particular value, a scripted trajectory, or other user defined criteria, such as the current velocity direction. The ordering of filters is important. For example, applying this filter before updating particle velocities leads to the computed velocities having an unwanted "spin" component.

Flipping Sprites

In most sprite-based applications, characters can turn around simply by reflecting the sprite about a vertical axis. Incorporating such a discontinuous change in our physics-based simulator requires care to maintain plausible elastic simulation. Since the velocity at the next timestep is computed as

$$\mathbf{v}^{n+1} = \frac{\mathbf{p}^{n+1} - \mathbf{p}^n}{dt} \tag{15}$$

after flipping the sprite, we adjust the previous positions, $\mathbf{p}^n$ so that velocities are not changed by flipping. Artistically, this gives a sense of weight to objects as they change direction; physically, it preserves linear momentum.

Results

To illustrate our authoring process we consider the classic animation task of creating a bouncing ball with squash and stretch. Figure 3 shows the simple rig we use to create the example poses in Figure 4. Figure 4 also visualizes the three line segments that make up the example manifold, which allows interpolation between the undeformed pose and either the squashed or stretched poses. In this example, the undeformed pose is set as the equilibrium pose. In addition, we use energy adjustment to move the selected example toward the squashed pose, which is otherwise largely ignored. Velocity-based adjustment pushes the selected example toward the stretched pose when the ball is moving quickly, and global orientation control aligns the pose with the velocity direction. Finally, global momentum adjustment allows the ball to bounce, appearing to "come alive" and move of its own volition. Figure 5 shows a variety of behaviors that can be achieved by changing the parameters.

We have also incorporated dynamic sprites into several simple games. In this context, the dynamic sprites enhance both visual complexity as well as gameplay compared to static objects or previous example-based approaches.

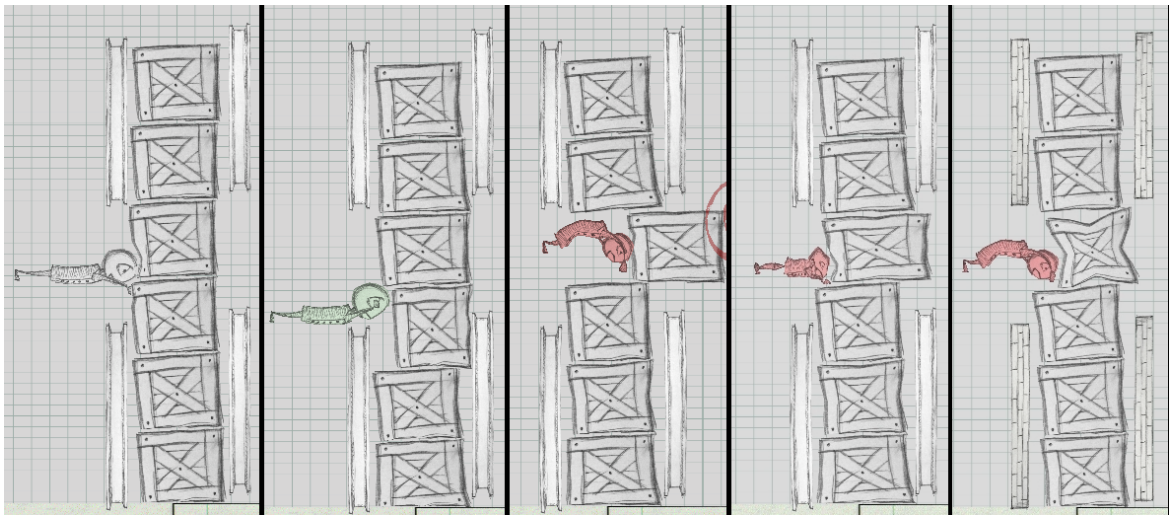

Figure 6: *From left to right: Shape matching only, shape matching with examples, 3 different dynamic sprites.*

In our first game, players attempt to navigate a character vertically towards a finish line by jumping on a sequence of platforms. The player can control the rest pose of the character and apply left and right forces while in the air. With dynamic sprites, it is easy to create a variety of platform types (shown in Figure 7) that look and behave differently based on the underlying examples and settings for the artistic controls. For example, the brick platforms are mainly rigid and provide little vertical boost after impact, while the I-beams bend elastically when the player lands. The rope platforms are softer than the I-beams and do not spring back to their original shape as quickly.

In the second game, players position I-beams, catapults and other objects to direct a passive object (e.g., a bouncy ball) that is dropped from above towards a target. Our dynamic sprites can be used to animate a variety of game objects ranging from sturdy brick obstacles, to an energetic catapult and launching pad. A sample level is shown in Figure 8.

In our third game, ragdoll figures are fired from a cannon toward a target while various boxes and I-beams serve as obstacles (see Figure 6). We represent all game elements as dynamic sprites with different behaviors: the cannon contracts on firing; the character is drawn to an elongated, "superman" pose when moving quickly but assumes a fetal position upon impact; the I-beams wiggle elastically due to bent example poses; and energy adjustment draws the boxes to a variety of poses that are largely orthogonal to the external forces. As can be seen in the figure and accompanying video our dynamic sprites generate richer deformations than shape matching or standard example-based physics, where the red poses are not activated. Unless a large number of particles are perturbed toward an example pose, the manifold projection does not move the current rest pose significantly through the manifold; the bulk of the material, which is undeformed, has lowest cost for the current rest pose, especially for the stiff materials we desire. The "rotation cluster" approximation we use for efficiency exacerbates this, since only the representative particles are considered during our manifold projection step. However, even when considering all particles during the projection step, it is difficult to trigger example poses through local collisions when using stiff materials.

In the final game, our ragdoll character is now actively controlled using a 2-level finite state machine (see Figures 9 and 10). The walking behavior state machine moves the equilibrium pose through 6 keyframe poses on a manifold with a loop topology. Unlike in traditional physics-based controllers, we did not need to consider balance and robustness of the character's locomotion; global orientation control keeps the character upright.

When the user wants to jump, first the character must crouch to prepare, and then can spring upward into a standing pose, similarly to the snake character in the platformer game. The character's inflatable jacket can also be used to shoot him further up in the air, and allow him to float down slowly. The floating behavior slowly blends from the inflated pose to the neutral pose. The position in the pose manifold also determines the magnitude of a drag force on the man as he slowly floats to the ground. Figure 11 shows the ragdoll using a bouncy I-beam to help him float across a dangerous ball pit.

Our results exhibit many behaviors that might be considered artifacts in other physics based approaches. However, many of these are both desirable, and easily controllable by an artist. For example, the "jiggliness" and excitability of the I-Beams in the cannon game are easily adjustable from passive and stiff, to energetic and oscil-

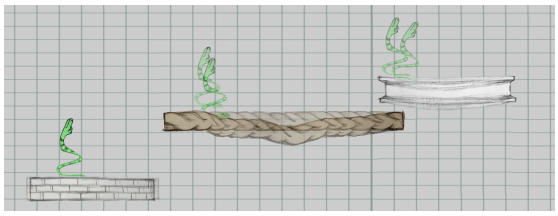

Figure 7: *These dynamic sprites platforms cover a broad range of behaviors.*

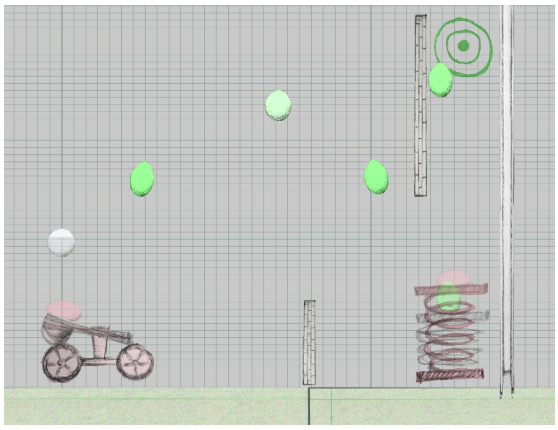

Figure 8: *The user places the catapult and springy platform to help the ball reach the goal.*

latory, as seen in the accompanying video. In the same way that cartoon drawings don't accurately reflect their real world inspiration, our physics-inspired (nonphysical) behavior is a stylized exaggeration of real world motion.

Our method does exhibit some behaviors that we consider to be artifacts. Some of the blended poses seem unnatural or undesirable. This can be alleviated to some extent by including more in-between poses in the manifold. Contact handling sometimes introduces oscillations near collisions, especially during resting contact. Finally, the global rotations, applied by our orientation control can be visually jarring. A first-order spring based control method may not be adequate to achieve smooth, subtle rotation control.

Conclusion

Our current system allows artists to turn sketches of example poses into dynamic, physical, reactive objects and actively controlled characters. We see several promising directions for future work. In our current pipeline, objects are posed using a warping system. This is not a requirement of the method; adding support for automatic registration (e.g.[46]) would allow us to use traditional sprite sheets as input. While we believe that our parameters

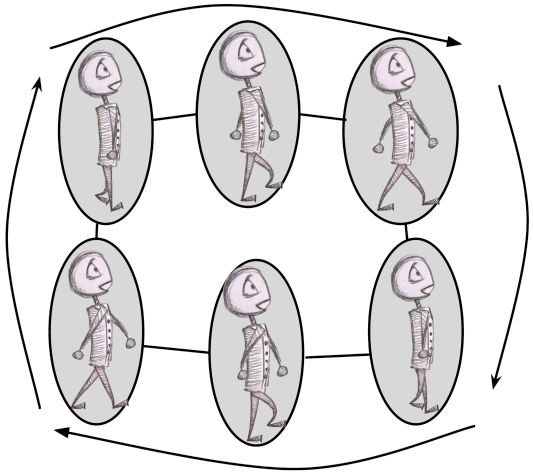

Figure 9: *With 3 example poses and their reflections, a simple manifold, and a finite state machine, our system generates a lively, stylized walking behavior.*

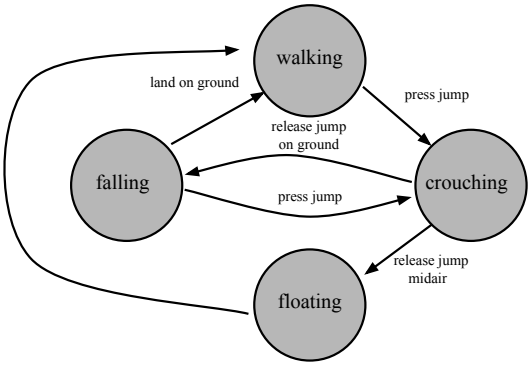

Figure 10: *A simple state machine transitions between behaviors based on user input, the ragdoll's state.*

	Platform Game	Puzzle Game	Launcher Game
Neighborhood Shape Matching	3.3	2.2	6.0
Global Shape Matching	0.3	0.1	0.1
Manifold Projection	5.0	8.9	2.0
Collisions	0.7	13.2	0.3
Other	1.2	1.0	0.5

Table 1: *Timing results for selected examples (ms per 60Hz frame).*

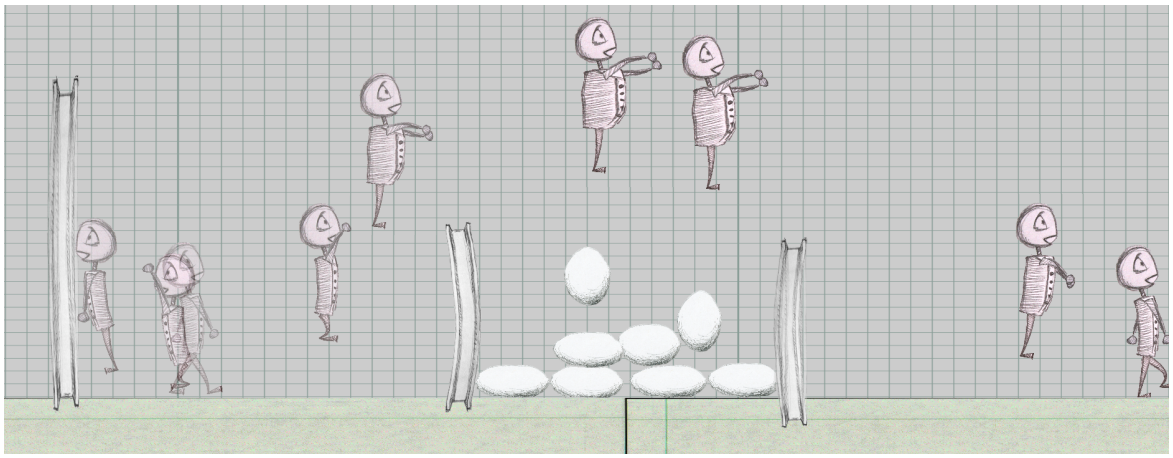

Figure 11: *Our ragdoll uses a bouncy I-beam like a trampoline to gain enough sideways momentum to float over a pit of balls.*

are intuitive, it might still be interesting to investigate automatic tuning – e.g. changing momentum-preservation based on an artist-sketched bounce. Similarly, it may be difficult for users of our system to understand if a behavior they are seeing results from badly-tuned parameters or the need for more example poses; this is also a question that we may be able to answer algorithmically. Because we focus on sketched input, our technique works only in 2D; however, we believe that the pose manifold concept and the control methods we propose should generalize to 3D.

Overall, we find that our system sits at an interesting point in the design space of methods for creating dynamic content. By directly connecting artist-created poses with physical properties, dynamic sprites enable artists to create more interesting and detailed physics-based characters and objects with less effort than either traditional sprite sheets (which sacrifice physical realism) or rigging and animation systems (which require several different areas of expertise).

Acknowledgments The authors wish to thank the anonymous reviewers for their helpful comments. This work was supported in part by a gift from Adobe Systems Incorporated and National Science Foundation Awards CNS-0855167, IIS-1249756, and IIS-1314896.

References

[1] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. In *ACM SIGGRAPH Computer Graphics*, volume 26, pages 35–42. ACM, 1992.

[2] Marc Alexa, Daniel Cohen-Or, and David Levin. As-rigid-as-possible shape interpolation. In *Proceedings of ACM SIGGRAPH*, pages 157–164, July 2000.

[3] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. *ACM Trans. Graph.*, 24(3):1134–1141, August 2005.

[4] Tom Ngo, Doug Cutrell, Jenny Dana, Bruce Donald, Lorie Loeb, and Shunhui Zhu. Accessible animation and customizable graphics via simplicial configuration modeling. In *Proceedings of ACM SIGGRAPH*, pages 403–410, July 2000.

[5] Christoph Bregler, Lorie Loeb, Erika Chuang, and Hrishi Deshpande. Turning to the masters: Motion capturing cartoons. *ACM Trans. Graph.*, 21(3):399–407, July 2002.

[6] Youichi Horry, Ken ichi Anjyo, and Kiyoshi Arai. Tour into the picture: Using a spidery mesh interface to make animation from a single image. In *Proceedings of ACM SIGGRAPH*, pages 225–232, August 1997.

[7] Byong Mok Oh, Max Chen, Julie Dorsey, and Frédo Durand. Image-based modeling and photo editing. In *Proceedings of ACM SIGGRAPH*, pages 433–442, August 2001.

[8] Alec Rivers, Takeo Igarashi, and Frédo Durand. 2.5d cartoon models. *ACM Trans. Graph.*, 29(4):59:1–59:7, July 2010.

[9] Demetri Terzopoulos and Kurt Fleischer. Deformable models. *Visual Computer*, 4(6):306–331, 1988.

[10] Demetri Terzopoulos and Kurt Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *Proceedings of ACM SIGGRAPH*, pages 269–278, August 1988.

[11] James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. In *Pro-*

ceedings of ACM SIGGRAPH, pages 111–120, August 1999.

[12] Eric G. Parker and James F. O'Brien. Real-time deformation and fracture in a game environment. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 156–166, August 2009.

[13] Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. Stable real-time deformations. In *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 49–54, 2002.

[14] M. Müller and M. Gross. Interactive virtual materials. In *The Proccedings of Graphics Interface*, pages 239–246, 2004.

[15] G. Irving, J. Teran, and R. Fedkiw. Invertible finite elements for robust simulation of large deformation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 131–140, 2004.

[16] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM Trans. Graph.*, 24(3):471–478, July 2005.

[17] Alec R. Rivers and Doug L. James. Fastlsm: fast lattice shape matching for robust real-time deformation. *ACM Trans. Graph.*, 26(3), July 2007.

[18] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *J. Vis. Comun. Image Represent.*, 18(2):109–118, 2007.

[19] Jos Stam. Nucleus: Towards a unified dynamics solver for computer graphics. In *CAD/Graphics*, pages 1–11, 2009.

[20] Matthias Müller and Nuttapong Chentanez. Solid simulation with oriented particles. *ACM Trans. Graph.*, 30(4):92:1–92:10, July 2011.

[21] B. Thomaszewski, S. Pabst, and W. Strasser. Continuum-based strain limiting. *Comput. Graph. Forum*, 28(2):569–576, 2009.

[22] Huamin Wang, James O'Brien, and Ravi Ramamoorthi. Multi-resolution isotropic strain limiting. *ACM Trans. Graph.*, 29(6):156:1–156:10, 2010.

[23] Ronen Barzel, John F. Hughes, and Daniel N. Wood. Plausible motion simulation for computer graphics animation. In *Computer Animation and Simulation '96*, Proceedings of the Eurographics Workshop, pages 184–197, Poitiers, France, September 1996.

[24] Stephen Chenney and David A. Forsyth. Sampling plausible solutions to multi-body constraint problems. In *Proceedings of ACM SIGGRAPH*, pages 219–228, July 2000.

[25] Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. Interactive manipulation of rigid body simulations. In *Proceedings of ACM SIGGRAPH*, pages 209–218, July 2000.

[26] Miklós Bergou, Saurabh Mathur, Max Wardetzky, and Eitan Grinspun. TRACKS: Toward directable thin shells. *ACM Trans. Graph.*, 26(3):50:1–50:10, July 2007.

[27] Petros Faloutsos, Michiel Van De Panne, and Demetri Terzopoulos. Dynamic free-form deformations for animation synthesis. *Visualization and Computer Graphics, IEEE Transactions on*, 3(3):201–214, 1997.

[28] Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. Example-based elastic materials. *ACM Trans. Graph.*, 30(4):72:1–72:8, July 2011.

[29] Christian Schumacher, Bernhard Thomaszewski, Stelian Coros, Sebastian Martin, Robert Sumner, and Markus Gross. Efficient simulation of example-based materials. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 1–8, 2012.

[30] Yuki Koyama, Kenshi Takayama, Nobuyuki Umetani, and Takeo Igarashi. Real-time example-based elastic deformation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '12, pages 19–24, 2012.

[31] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM Trans. Graph.*, 21(3):473–482, July 2002.

[32] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. *ACM Trans. Graph.*, 21(3):491–500, July 2002.

[33] Okan Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Trans. Graph.*, 21(3):483–490, July 2002.

[34] Philippe Beaudoin, Stelian Coros, Michiel van de Panne, and Pierre Poulin. Motion-motif graphs. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, pages 117–126, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[35] Alla Safonova and Jessica K. Hodgins. Construction and optimal search of interpolated motion graphs. *ACM Trans. Graph.*, 26(3), July 2007.

[36] Paul S. A. Reitsma and Nancy S. Pollard. Evaluating motion graphs for character animation. *ACM Trans. Graph.*, 26(4), October 2007.

[37] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, pages 71–78, August 1995.

[38] Kangkang Yin, Kevin Loken, and Michiel van de Panne. SIMBICON: Simple biped locomotion control. *ACM Trans. Graph.*, 26(3):105:1–105:10, July 2007.

[39] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Generalized biped walking control. *ACM Transctions on Graphics*, 29(4):Article 130, 2010.

[40] Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel van de Panne. Locomotion skills for simulated quadrupeds. *ACM Transactions on Graphics*, 30(4):Article TBD, 2011.

[41] Stelian Coros, Sebastian Martin, Bernhard Thomaszewski, Christian Schumacher, Robert Sumner, and Markus Gross. Deformable objects alive! *ACM Trans. Graph.*, 31(4):69:1–69:9, 2012.

[42] Fabian Hahn, Sebastian Martin, Bernhard Thomaszewski, Robert Sumner, Stelian Coros, and Markus Gross. Rig-space physics. *ACM Trans. Graph.*, 31(4):72:1–72:8, 2012.

[43] Alec Jacobson, Ilya Baran, Ladislav Kavan, Jovan Popović, and Olga Sorkine. Fast automatic skinning transformations. *ACM Trans. Graph.*, 31(4):77:1–77:10, July 2012.

[44] J.A. Fike and J.J. Alonso. The development of hyper-dual numbers for exact second-derivative calculations. *AIAA paper*, 886, 2011.

[45] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, pages 109–116, 2007.

[46] Daniel Sýkora, John Dingliana, and Steven Collins. As-rigid-as-possible image registration for hand-drawn cartoon animations. In *Proceedings of International Symposium on Non-photorealistic Animation and Rendering*, pages 25–33, 2009.